

✓ Group 12 CIND830 Teamwork Assignment (Assignment 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.

```
1 #Upload the SuperStoreOrders.csv dataset to the session storage
2 #Run the following code to import the file
3 from google.colab import drive
4 import pandas as pd
5 df = pd.read_csv('SuperStoreOrders.csv')
6 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):

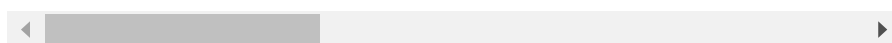
1. Design classes to read the dataset from a CSV file.
2. Implement methods to handle missing values and outliers, especially in the "Sales" column.
3. Transform data types, especially the "Date" column, to datetime or epoch timestamp format if necessary.

```

1 # 1. Design classes to read the dataset from a CSV file.
2 import pandas as pd
3
4 # Read the CSV File
5 class DataReader:
6     def read(self,file_name):
7         raise NotImplementedError("Subclass must implement this method")
8
9 class CSVReader(DataReader):
10     def read(self, file_name):
11         return pd.read_csv(file_name)
12
13 def load_data(reader, file_name):
14     return reader.read(file_name)
15
16 data_reader = CSVReader()
17 data = load_data (data_reader, 'SuperStoreOrders.csv')
18 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



```

1 # 2. Implement methods to handle missing values and outliers, especially in the "Sales" column.
2 import pandas as pd
3 import numpy as np
4
5 class DataProcessor:
6     def __init__(self, file_path):
7         self.file_path = file_path
8         self.df = pd.read_csv(file_path)
9
10    def handle_missing_values(self):    #replace missing value in "Sales" with mean value
11        self.df['Sales'].fillna(self.df['Sales'].mean(), inplace=True)
12
13    def handle_outliers(self, threshold=3):    # Remove outliers in the "Sales" column using z-score
14        z_scores = np.abs ((self.df['Sales'] - self.df['Sales'].mean()) / self.df['Sales'].std())
15        self.df = self.df[(z_scores < threshold)]
16
17    def save_processed_data(self, New_File_Name):    # Save the processed data to a new CSV file
18        self.df.to_csv(New_File_Name, index=False)
19
20 file_name = "SuperStoreOrders.csv"
21 New_file_name = "New CSV File.csv"
22 processor = DataProcessor(file_name)
23
24     # Handling missing values
25 processor.handle_missing_values()
26
27     # Handling outliers
28 processor.handle_outliers()
29
30     # Save the processed data
31 processor.save_processed_data(New_file_name)
32 df = pd.read_csv(New_file_name)
33 df

```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate

```
1 # 3. Transform data types, especially the "Date" column, to datetime or epoch timestamp format if necessary
2 import pandas as pd
3
4 class CSVTransformer:
5     def __init__(self, file_path):
6         self.file_path = file_path
7         self.df = pd.read_csv(file_path)
8
9     def transform_date_column(self, Order_Date):
10        try:
11            # Convert the specified "Order_date" column to datetime format
12            self.df[Order_Date] = pd.to_datetime(self.df[Order_Date], errors='raise')
13            self.df[Order_Date] = self.df[Order_Date].astype(int) / 10**9
14            self.df.to_csv(self.file_path, index=False)
15            print(f"Transformation successful. Check the updated file at: {self.file_path}")
16
17        except Exception as e:
18            print(f"Error occurred during transformation: {e}")
19
20 #Create new CVS file.csv
21 file_path = 'SuperStoreOrders.csv'
22 transformer = CSVTransformer(file_path)
23 transformer.transform_date_column('Date')
24 df = pd.read_csv(New_file_name)
25 df
```

Error occurred during transformation: 'Date'

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate

✓ 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.

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

```
1 #Extract month from order date and add year/month column
2 from re import M
3 import datetime
4 df['month'] = pd.to_datetime(df.order_date, dayfirst= True)
5 #print(data.month)
6 df['month'] = df['month'].dt.month
7 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) was deprecated in 1.1.0 and will be removed in 1.2.0. Use dayfirst=True to silence this warning.
```

```

1 class iDataAnalysis:
2     '''Algorithms for Data Analysis on input dataframe (df)'''
3     def __init__(self, dataframe):
4         self._dataframe = dataframe
5
6 #1. Implement an algorithm to identify the top 10 subcategories by sales
7
8 #Algorithm top_ten was made generalizable for the user to be able to group the top 10 of the defined
9 #To answer question 1, the user could call the method using the dataset's sales column as variable x,
10 def top_ten (self, x, group):
11     '''Summation grouping of "x" by "group".
12     Returns a dictionary of the top ten results of x for each group'''
13     #Initializing list of dictionary key values pairs of key("group") : value("x")
14     group = tuple(group)
15     x = list(x)
16     my_list = []
17     for i in range(len(group)):
18         my_dict = {"group": group[i], "x": x[i]}
19         my_list.append(my_dict)
20     #Group and sum values("x") by key("group")
21     group_keys = "group"
22     sum_keys = "x"
23     result = {}
24     for i in my_list:
25         subgroup = i[group_keys]
26         if subgroup not in result:
27             for x in sum_keys:
28                 result[subgroup] = int(0)
29             for n in sum_keys:
30                 result[subgroup] += int(i[n])
31     #Sort results in descending order and return the top 10 results
32     #Python's sorted() function uses Timsort which is a hybrid of merge sort and insertion sort
33     name = lambda x: (x[1], x[0])
34     sorted_result = sorted(result.items(), key = name, reverse = True)
35     return (sorted_result[0:10])
36
37 #Algorithm top_sales_by_subcategory is another sample algorithm but made specific to grouping sales by subcategory
38 def top_sales_by_subcategory (self):
39     '''Summation grouping of sales by subcategory.
40     Returns the top ten sales by group'''
41     #Group by sub category and sum number of sales
42     sumSales = df.groupby(['sub_category'])['sales'].sum().sort_values(ascending = False)
43     return(sumSales[0:10])
44
45 #2. Design a data structure to store the monthly sales trend for the entire superstore.
46 def store_monthly_sales(self):
47     '''Stores the monthly sales trend (both "sales" and "profit") into a list of lists'''
48     #Store month/year values into list
49     month_year = df.month_year
50     month_year_list = []
51     for i in range(len(month_year)):
52         if month_year[i] not in month_year_list:
53             month_year_list.append(month_year[i])
54     #Group and sum monthly sales and profit and store values into lists
55     monthlySales = df.groupby(['year', 'month'])['sales', 'profit'].sum()
56     sales_list = monthlySales['sales'].tolist()
57     profit_list = monthlySales['profit'].tolist()
58     #Store values in a list of lists
59     col_list = []
60     row_list = []
61     for i in range(len(month_year_list)):
62         col_list = month_year_list[i], sales_list[i], profit_list[i]

```

```

63         row_list.append(col_list)
64     return row_list
65
66 #3. Implement a search algorithm to find all days where a promo led to sales above a certain threshold.
67     def searchDays(self, treshhold):
68         '''Search for days where a promotional item led for sales above selected sales "treshhold".
69         Returns tuple(days, item, sales, and discount) above selected sales treshhold'''
70         treshhold = int(treshhold)
71         #Linear Search for position of treshhold and store sales above treshhold
72         above_treshhold_position = []
73         position = 0
74         while position < len(df.sales):
75             if treshhold < df.sales[position]:
76                 if df.discount[position] > 0:
77                     above_treshhold_position.append(position)
78                 position += 1
79         my_lyst = []
80         for i in above_treshhold_position:
81             my_tuple = (df.order_date[i], df.product_name[i], df.sales[i], df.discount[i])
82             my_lyst.append(my_tuple)
83         return my_lyst

```

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

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

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

```
2 #Algorithm option 1
```

```
3 data_analysis.top_ten(df.sales, df.sub_category)
```

```
4 #help(analysis.top_ten)
```

```

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

```

```
1 #Algorithm option 2
```

```
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

```

```
1 #2. Design a data structure to store the monthly sales trend for the entire superstore.
```

```
2 a = data_analysis.store_monthly_sales()
```

```
3 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()
```

1 #3. Implement a search algorithm to find all days where a promo led to sales above a certain threshold.
 2 #Here we calling the method using a sales treshold of 5000 to return only the days and items that had ove
 3 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 fulfillment center for all orders. Create a queue for the store's the first 10 orders (order_id) by priority (order_priority).

```
1 from queue import PriorityQueue
2 import queue
```

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

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



```

1 #Store priority_num and order_id columns in a list of tuples using above function
2 order_list = data_by_priority(df.priority_num, df.order_id)
3
4 #Initiate priority queue and give worker 10 orders from order_list.
5 #Will print the order in which order_items are inserted into the queue
6 pq = PriorityQueue(10)
7 for order in range(10):
8     print(order_list[order])
9     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')

1 #Will print the order in which order_items are extracted from the priority queue
2 #Note that the order is different from above since the order_items have been prioritized
3 for order in range(10):
4     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 therefore 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 amount 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 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*.

```

1 #2. Designing an array class to store the monthly sales trend
2 class iArray():
3     '''A representation of an array'''
4     def __init__(self, capacity, fillValue = None):
5         '''Capacity is the size of the array.
6         None object will be placed at each position.'''
7         self._items = list()
8         for count in range(capacity):
9             self._items.append(fillValue)
10
11     def __len__(self):
12         #Obtaining the size of the array
13         return len(self._items)
14
15     def __str__(self):
16         #Representing the array as String
17         return str(self._items)
18
19     def __setitem__(self, index, newItem):
20         #Replacing an element or assigning a value to an element
21         self._items[index] = newItem
22
23     def __getitem__(self, index):
24         #Obtaining an element in the array
25         return self._items[index]
26
27     def __iter__(self):
28         #Traversing elements with loops
29         return iter(self._items)
30
31     def loadFactor(self):
32         '''Returns the load factor of the array'''
33         logicalSize = len([x for x in self if x is not None])
34         physicalSize = len(self)
35         loadFactor = logicalSize/physicalSize
36         return(loadFactor)

```

```

1 #Changing the algorithm to store the lists in an array
2 def store_monthly_sales_optimized():
3     '''Stores the monthly sales trend into list of lists'''
4     #Store month/year values into list
5     month_year = df.month_year
6     month_year_list = []
7     for i in range(len(month_year)):
8         if month_year[i] not in month_year_list:
9             month_year_list.append(month_year[i])
10    #Group and sum monthly sales and profit and store values into lists
11    monthlySales = df.groupby(['year','month'])['sales', 'profit'].sum()
12    sales_list = monthlySales['sales'].tolist()
13    profit_list = monthlySales['profit'].tolist()
14
15    #Store values in a array of lists
16    col_list = []
17    my_array = iArray(len(month_year_list))
18    for i in range(len(month_year_list)):
19        col_list = month_year_list[i], sales_list[i], profit_list[i]
20        my_array[i] = col_list
21    return my_array

```

```

1 import sys
2
3 #Documenting size of original datastructure(list of lists)
4 data_analysis = iDataAnalysis(df)
5 monthly_sales = data_analysis.store_monthly_sales()
6 print(monthly_sales)
7 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

```

```

1 #Documenting size of new datastructure (array of lists)
2 msales = store_monthly_sales_optimized()
3 print(msales)
4 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 threshold. 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 threshold with a worst case scenario logarithmic $O(\log_2(n))$ 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`.

```

1 #Optimizing question number 3 for time efficiency (using Python's bisect library binary search algorithm)
2 import bisect
3
4 #3. Implement a search algorithm to find all days where a promo led to sales above a certain threshold.
5 def searchDaysOptimized (treshhold):
6     '''Search for days where a promotional item led to sales above "treshhold".
7     Returns tuple(days, item, sales, and discount)'''
8
9     #Sort dataframe by sales
10    sorted_sales = df.sort_values(['sales'], ascending= True)
11
12    #Remove rows withouth a discount
13    sorted_sales = sorted_sales[sorted_sales['discount'] > 0]
14
15    #Store values with discounts in list to assign new index position and iterate through list:
16    sales_list = list(sorted_sales.sales)
17    date_list = list(sorted_sales.order_date)
18    product_list = list(sorted_sales.product_name)
19    promotion_list = list(sorted_sales.discount)
20
21    #Use python bisect for binary search for position of treshhold
22    treshhold_position = bisect.bisect_right(sales_list, treshhold)
23
24    #Store data, sales, product, and discount into new list of lists
25    my_lyst = []
26    i = treshhold_position
27    while i < len(sales_list):
28        my_tuple = (date_list[i], product_list[i], sales_list[i], promotion_list[i])
29        my_lyst.append(my_tuple)
30        i += 1
31    return my_lyst

```



```

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

```



```

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

```

1.2154368649999014


```

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

```

0.055800958999952854

✓ Data Visualization (by Maliheh Garoosiha):

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

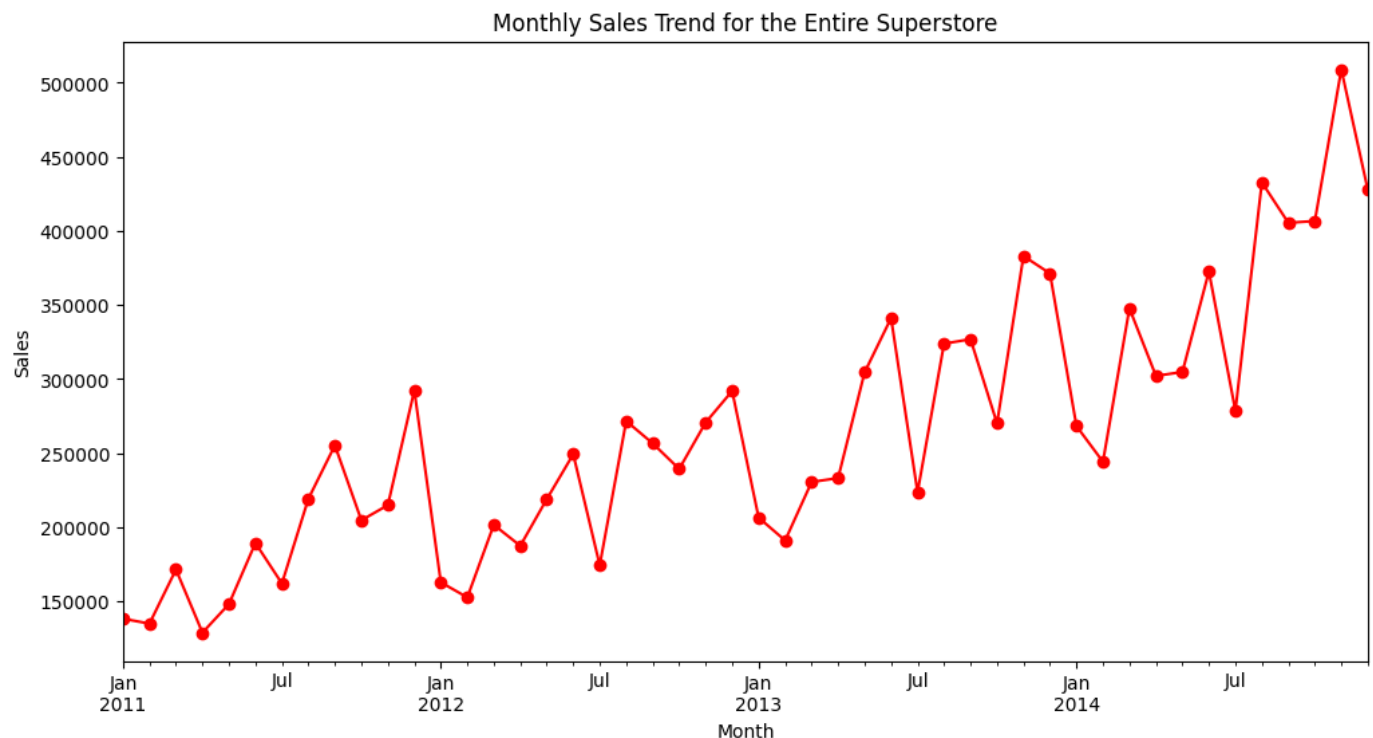
1. Create a bar chart showing the top 10 stores categories by sales.

```
1 #top_categories = df.groupby('Sub-Category')['Sales'].sum().nlargest(10)
2 df['sales'] = pd.to_numeric(df['sales'], errors='coerce')
3
4 # Group by category and calculate total sales for each category
5 category_sales = df.groupby('sub_category')['sales'].sum()
6
7 # Select the top 10 categories by sales
8 top_10_categories = category_sales.nlargest(10)
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```



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

```
1 df['order_date'] = pd.to_datetime(df['order_date'])
2
3 # Extract month and year from 'Order Date'
4 df['Month'] = df['order_date'].dt.to_period('M')
5
6 # Group by month and sum the Sales for each month
7 monthly_sales = df.groupby('Month')['sales'].sum()
8
9 # Plotting the line chart
10 plt.figure(figsize=(12, 6))
11 monthly_sales.plot(kind='line', marker='o', color='red', xlabel='Month', ylabel='Sales', title='Monthly Sal
12 plt.show()
```



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

```
1 # Group by Customer ID and sum the Sales for each customer
2 customer_sales = df.groupby('customer_name')['sales'].sum()
3
4 # Group by Customer ID and count the number of orders for each customer
5 customer_order_count = df['customer_name'].value_counts()
6
7
8 # Create a scatter plot
9 plt.figure(figsize=(12, 6))
10
11 plt.scatter(customer_order_count, customer_sales, color='orange')
12 plt.xlabel('Number of Customers-perchase') # Set the label for the x-axis
13 plt.ylabel('Customer Sales') # Set the label for the y-axis
```

```
14 plt.title('Comparing number of Customer Order vs Sales')
15 plt.show()
```



1 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