School of IT & Business Technologies
Graduate Diploma in Data Analytics
Cover Sheet and Student Declaration

This sheet must be signed by the student and attached to the submitted assessment.

Course Title:	**Data Collection and Analysis**	**Course code:**	**GD604**
Student Name:	Mira Torririt	**Student ID:**	764707793
Assessment No & Type:	Assessment 2- Report	**Cohort:**	GDDA7123C
Due Date:	March 04, 2024	**Date Submitted:**	March 04, 2024
Tutor's Name:	Harsh Tiwari		
Assessment Weighting	60%		
Total Marks	100		

Student Declaration:

I declare that:

- I have read the New Zealand School of Education Ltd policies and regulations on assessments and understand what plagiarism is.
- I am aware of the penalties for cheating and plagiarism as laid down by the New Zealand School of Education Ltd.
- This is an original assessment and is entirely my own work.
- Where I have quoted or made use of the ideas of other writers, I have acknowledged the source.
- This assessment has been prepared exclusively for this course and has not been or will not be submitted as assessed work in any other course.
- It has been explained to me that this assessment may be used by NZSE Ltd, for internal and/or external moderation.
- If I am late in handing in this assessment without prior approval (see student regulations in handbook), marks will be deducted, to a maximum of 50%.

Student signature:

Date: 04/03/24

Tutor only to complete		
Assessment result:	**Mark /100**	**Grade**

Assessment 2: GD604 – Data Collection and Analysis

Project 1: Sales Analysis

Mira Torririt

GD604

Tutor: Harshvardhan Tiwari

School of Technology
Graduate Diploma in Data Analytics (Level 7)

March 04, 2024

Table of Contents

Chapter 1: Introduction

Sales analysis is vital in all businesses. It shows the past, status, and business trends that help decision-making. It will tell you the weaknesses and strengths of the business by reviewing its aspects.

The correct sales analysis will give you much information to provide a better sales forecast and achieve business success.

This assessment aims to show how data-driven decision-making is to be done using sales analysis as a tool.

The sample dataset includes both categorical and numerical data, which, in the end, will give us meaningful insight into the current sales situation.

Task A

a. Loading of Data – the dataset was loaded into the data frame using Jupyter Python. It has an initial of 2,823 rows and 25 columns. To read the dataset, I used the syntax:

sales_df=pd. read_csv('Sales_Sample_Public_Dataset.csv')
sales_df

```
# Task A
# a.
sales_df=pd.read_csv('Sales_Sample_Public_Dataset.csv')
sales_df
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSL
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long A Av
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped	2	5	2003	...	59 r l'At
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped	3	7	2003	...	27 r Colonel F
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hi
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Stror
...	...	...	...	...	...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004 0:00	Shipped	4	12	2004	...	C/ Moralz
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped	1	1	2005	...	Torika
2820	10386	43	100.00	4	5417.57	3/01/2005 0:00	Resolved	1	3	2005	...	C/ Moralz
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped	1	3	2005	...	1 rue Al Lor
2822	10414	47	65.52	9	3079.44	5/06/2005 0:00	On Hold	2	5	2005	...	8616 Spini

2823 rows × 25 columns

The dataset can be found in my GitHub account:

https://github.com/Myres16/Data-Analytics-Assessments/tree/main/604

***Note**: The copies of the Python notebook and dataset are available in my GitHub.*

b. The syntax used to show the first few rows was sales_df.head(25)

```
# b.
sales_df.head(10)
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLINE
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long Airpor Avenu
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped	2	5	2003	...	59 rue d l'Abbay
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped	3	7	2003	...	27 rue d Colonel Pierr Avi
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hillsid D
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Strong S
5	10168	36	96.66	1	3479.76	10/28/2003 0:00	Shipped	4	10	2003	...	9408 Furth Circl
6	10180	29	86.13	9	2497.77	11/11/2003 0:00	Shipped	4	11	2003	...	184, chausse d Tourn
7	10188	48	100.00	1	5512.32	11/18/2003 0:00	Shipped	4	11	2003	...	Drammen 121 PR 744 Sentrun
8	10201	22	98.57	2	2168.54	12/01/2003 0:00	Shipped	4	12	2003	...	5557 Nort Pendale Stree
9	10211	41	100.00	14	4708.44	1/15/2004 0:00	Shipped	1	1	2004	...	25, rue Lauristo

10 rows × 25 columns

c. Managing the missing values.

c.1) In preparation for the analysis, I identified the number of missing values by using the syntax; sales_df.isnull().sum()

```
sales_df.isnull().sum()
```

```
ORDERNUMBER            0
QUANTITYORDERED        0
PRICEEACH              0
ORDERLINENUMBER        0
SALES                  0
ORDERDATE              0
STATUS                 0
QTR_ID                 0
MONTH_ID               0
YEAR_ID                0
PRODUCTLINE            0
MSRP                   0
PRODUCTCODE            0
CUSTOMERNAME           0
PHONE                  0
ADDRESSLINE1           0
ADDRESSLINE2        2521
CITY                   0
STATE               1486
POSTALCODE            76
COUNTRY                0
TERRITORY           1074
CONTACTLASTNAME        0
CONTACTFIRSTNAME       0
DEALSIZE               0
dtype: int64
```

c.2) The fillna function was used to TERRITORY, STATE and POSTALCODE columns to replace it with other data.

TERRITORY column – missing values were replaced by empty values using the syntax:
sales_df['TERRITORY']=sales_df['TERRITORY'].fillna('NA')
sales_df

```
# c.2 : Replacing TERRITORY null values by empty value
sales_df['TERRITORY']=sales_df['TERRITORY'].fillna('')
sales_df
```

YEAR_ID	...	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE
2003	...	897 Long Airport Avenue	NaN	NYC	NY	10022	USA		Yu	Kwai	Small
2003	...	59 rue de l'Abbaye	NaN	Reims	NaN	51100	France	EMEA	Henriot	Paul	Small
2003	...	27 rue du Colonel Pierre Avia	NaN	Paris	NaN	75508	France	EMEA	Da Cunha	Daniel	Medium
2003	...	78934 Hillside Dr.	NaN	Pasadena	CA	90003	USA		Young	Julie	Medium
2003	...	7734 Strong St.	NaN	San Francisco	CA	NaN	USA		Brown	Julie	Medium
...	...	...	...	...	...	...	...	...	...	...	...
2004	...	C/ Moralzarzal, 86	NaN	Madrid	NaN	28034	Spain	EMEA	Freyre	Diego	Small
2005	...	Torikatu 38	NaN	Oulu	NaN	90110	Finland	EMEA	Koskitalo	Pirkko	Medium
2005	...	C/ Moralzarzal, 86	NaN	Madrid	NaN	28034	Spain	EMEA	Freyre	Diego	Medium
2005	...	1 rue Alsace-Lorraine	NaN	Toulouse	NaN	31000	France	EMEA	Roulet	Annette	Small
2005	...	8616 Spinnaker Dr.	NaN	Boston	MA	51003	USA		Yoshido	Juri	Medium

- STATE column – missing values were replaced with 'NA' using the syntax:
  sales_df['STATE']=sales_df['STATE'].fillna('NA')
  sales_df

```
sales_df['STATE']=sales_df['STATE'].fillna('NA')
sales_df
```

YEAR_ID	...	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE
2003	...	897 Long Airport Avenue	NaN	NYC	NY	10022	USA	NA	Yu	Kwai	Small
2003	...	59 rue de l'Abbaye	NaN	Reims	NA	51100	France	EMEA	Henriot	Paul	Small
2003	...	27 rue du Colonel Pierre Avia	NaN	Paris	NA	75508	France	EMEA	Da Cunha	Daniel	Medium
2003	...	78934 Hillside Dr.	NaN	Pasadena	CA	90003	USA	NA	Young	Julie	Medium
2003	...	7734 Strong St.	NaN	San Francisco	CA	NaN	USA	NA	Brown	Julie	Medium
...	...	...	...	...	...	...	...	...	...	...	...
2004	...	C/ Moralzarzal, 86	NaN	Madrid	NA	28034	Spain	EMEA	Freyre	Diego	Small
2005	...	Torikatu 38	NaN	Oulu	NA	90110	Finland	EMEA	Koskitalo	Pirkko	Medium
2005	...	C/ Moralzarzal, 86	NaN	Madrid	NA	28034	Spain	EMEA	Freyre	Diego	Medium
2005	...	1 rue Alsace-Lorraine	NaN	Toulouse	NA	31000	France	EMEA	Roulet	Annette	Small
2005	...	8616 Spinnaker Dr.	NaN	Boston	MA	51003	USA	NA	Yoshido	Juri	Medium

- POSTALCODE – the null values were replaced by ‘None’ using the syntax:
  sales_df['POSTALCODE']=sales_df['POSTALCODE'].fillna('None')
  sales_df

```
sales_df['POSTALCODE']=sales_df['POSTALCODE'].fillna('None')
sales_df
```

_ID	YEAR_ID	...	ADDRESSLINE1	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE	CUSTOMEF
2	2003	...	897 Long Airport Avenue	NYC	NY	10022	USA	NA	Yu	Kwai	Small	
5	2003	...	59 rue de l'Abbaye	Reims	NA	51100	France	EMEA	Henriot	Paul	Small	
7	2003	...	27 rue du Colonel Pierre Avia	Paris	NA	75508	France	EMEA	Da Cunha	Daniel	Medium	
8	2003	...	78934 Hillside Dr.	Pasadena	CA	90003	USA	NA	Young	Julie	Medium	
10	2003	...	7734 Strong St.	San Francisco	CA	None	USA	NA	Brown	Julie	Medium	
...	...	...	...	...	...	...	...	...	...	...	...	
11	2004	...	5905 Pompton St.	NYC	NY	10022	USA	NA	Hernandez	Maria	Medium	28
12	2004	...	C/ Moralzarzal, 86	Madrid	NA	28034	Spain	EMEA	Freyre	Diego	Small	28
1	2005	...	Torikatu 38	Oulu	NA	90110	Finland	EMEA	Koskitalo	Pirkko	Medium	28
3	2005	...	1 rue Alsace-Lorraine	Toulouse	NA	31000	France	EMEA	Roulet	Annette	Small	28
5	2005	...	8616 Spinnaker Dr.	Boston	MA	51003	USA	NA	Yoshido	Juri	Medium	28

c.3) The ADDRESSLINE2 column – was dropped as it was not needed in the analysis. There was already data for the primary address, the ADDRESSLINE1 column. Also, due to numerous null values, ADDRESSLINE2 became meaningless in the analysis. The syntax used was:

sales_df=sales_df.drop(columns=['ADDRESSLINE2'])
sales_df

```
# c.2 : Dropping ADDRESSLINE2 as it is not needed in the analysis
sales_df=sales_df.drop(columns=['ADDRESSLINE2'])
sales_df
```

ERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	PHONE	ADDRESSLINE1	CITY	STATE	POSTALCODE	COUNTRY
2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	2125557818	897 Long Airport Avenue	NYC	NY	10022	USA
5	2765.90	5/07/2003 0:00	Shipped	2	5	2003	...	26.47.1555	59 rue de l'Abbaye	Reims	NA	51100	France
2	3884.34	7/01/2003 0:00	Shipped	3	7	2003	...	+33 1 46 62 7555	27 rue du Colonel Pierre Avia	Paris	NA	75508	France
6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	6265557265	78934 Hillside Dr.	Pasadena	CA	90003	USA
14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	6505551386	7734 Strong St.	San Francisco	CA	NaN	USA
...	...	...	...	...	...	...	...	...	...	...	...	...	...
15	2244.40	12/02/2004 0:00	Shipped	4	12	2004	...	(91) 555 94 44	C/ Moralzarzal, 86	Madrid	NA	28034	Spain
1	3978.51	1/31/2005 0:00	Shipped	1	1	2005	...	981-443655	Torikatu 38	Oulu	NA	90110	Finland
4	5417.57	3/01/2005 0:00	Resolved	1	3	2005	...	(91) 555 94 44	C/ Moralzarzal, 86	Madrid	NA	28034	Spain
1	2116.16	3/28/2005 0:00	Shipped	1	3	2005	...	61.77.6555	1 rue Alsace-Lorraine	Toulouse	NA	31000	France
9	3079.44	5/06/2005 0:00	On Hold	2	5	2005	...	6175559555	8616 Spinnaker Dr.	Boston	MA	51003	USA

Checking if the null values were all addressed:

```
sales_df.isnull().sum()
```

```
ORDERNUMBER         0
QUANTITYORDERED     0
PRICEEACH           0
ORDERLINENUMBER     0
SALES               0
ORDERDATE           0
STATUS              0
QTR_ID              0
MONTH_ID            0
YEAR_ID             0
PRODUCTLINE         0
MSRP                0
PRODUCTCODE         0
CUSTOMERNAME        0
PHONE               0
ADDRESSLINE1        0
CITY                0
STATE               0
POSTALCODE          0
COUNTRY             0
TERRITORY           0
CONTACTLASTNAME     0
CONTACTFIRSTNAME    0
DEALSIZE            0
dtype: int64
```

I also checked for duplicates as part of data preparation:

```
# Checking for duplicates
duplicated_rows = sales_df[sales_df.duplicated()]
duplicated_rows
```

ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	PHONE	ADDRE

0 rows × 24 columns

Formatted the PHONE and ORDER DATE columns for readability and consistency:

```
# formatting the PHONE number

sales_df['PHONE'] = sales_df['PHONE'].str.replace(r'\D', '', regex=True)
sales_df
```

)	YEAR_ID	...	PHONE	ADDRESSLINE1	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE
2	2003	...	2125557818	897 Long Airport Avenue	NYC	NY	10022	USA		Yu	Kwai	Small
5	2003	...	26471555	59 rue de l'Abbaye	Reims	NA	51100	France	EMEA	Henriot	Paul	Small
7	2003	...	33146627555	27 rue du Colonel Pierre Avia	Paris	NA	75508	France	EMEA	Da Cunha	Daniel	Medium
3	2003	...	6265557265	78934 Hillside Dr.	Pasadena	CA	90003	USA		Young	Julie	Medium
)	2003	...	6505551386	7734 Strong St.	San Francisco	CA	None	USA		Brown	Julie	Medium
.	...	...	...	...	...	...	...	...	...	...	...	...
2	2004	...	915559444	C/ Moralzarzal, 86	Madrid	NA	28034	Spain	EMEA	Freyre	Diego	Small
1	2005	...	981443655	Torikatu 38	Oulu	NA	90110	Finland	EMEA	Koskitalo	Pirkko	Medium
3	2005	...	915559444	C/ Moralzarzal, 86	Madrid	NA	28034	Spain	EMEA	Freyre	Diego	Medium
3	2005	...	61776555	1 rue Alsace-Lorraine	Toulouse	NA	31000	France	EMEA	Roulet	Annette	Small
5	2005	...	6175559555	8616 Spinnaker Dr.	Boston	MA	51003	USA		Yoshido	Juri	Medium

```
#removing the date stamp

sales_df['ORDERDATE'] = pd.to_datetime(sales_df['ORDERDATE'], format='%m/%d/%Y %H:%M')
sales_df['ORDERDATE'] = sales_df['ORDERDATE'].dt.strftime('%m/%d/%Y')
sales_df
```

UANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	PHONE	ADDRESSLINE1	CITY
30	95.70	2	2871.00	02/24/2003	Shipped	1	2	2003	...	2125557818	897 Long Airport Avenue	NY
34	81.35	5	2765.90	05/07/2003	Shipped	2	5	2003	...	26471555	59 rue de l'Abbaye	Reir
41	94.74	2	3884.34	07/01/2003	Shipped	3	7	2003	...	33146627555	27 rue du Colonel Pierre Avia	Pa
45	83.26	6	3746.70	08/25/2003	Shipped	3	8	2003	...	6265557265	78934 Hillside Dr.	Pasade
49	100.00	14	5205.27	10/10/2003	Shipped	4	10	2003	...	6505551386	7734 Strong St.	S Francis
...	...	...	...	...	...	...	...	...	...	...	...	...
20	100.00	15	2244.40	12/02/2004	Shipped	4	12	2004	...	915559444	C/ Moralzarzal, 86	Madi
29	100.00	1	3978.51	01/31/2005	Shipped	1	1	2005	...	981443655	Torikatu 38	Ou
43	100.00	4	5417.57	03/01/2005	Resolved	1	3	2005	...	915559444	C/ Moralzarzal, 86	Madi
34	62.24	1	2116.16	03/28/2005	Shipped	1	3	2005	...	61776555	1 rue Alsace-Lorraine	Toulou
47	65.52	9	3079.44	05/06/2005	On Hold	2	5	2005	...	6175559555	8616 Spinnaker Dr.	Bost

d. Sorting Values – I sorted the status in ascending order using the syntax:

sorted_df = sales_df.sort_values(by='STATUS', ascending=True)

sorted_df

```
# d. sorting status

sorted_df = sales_df.sort_values(by='STATUS', ascending=True)
sorted_df
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	PHONE
638	10253	33	100.00	4	4459.62	06/01/2004	Cancelled	2	6	2004	...	171555228
2593	10167	46	70.11	10	3225.06	10/23/2003	Cancelled	4	10	2003	...	069534655
1899	10167	32	63.12	3	2019.84	10/23/2003	Cancelled	4	10	2003	...	069534655
117	10248	20	100.00	3	2910.40	05/07/2004	Cancelled	2	5	2004	...	212555781
2573	10262	33	90.75	6	2994.75	06/24/2004	Cancelled	2	6	2004	...	91555944
...	...	...	...	...	...	...	...	...	...	...	...	...
1079	10282	36	100.00	3	4174.92	08/20/2004	Shipped	3	8	2004	...	415555145
1080	10293	22	100.00	6	2418.24	09/09/2004	Shipped	3	9	2004	...	011498855
1081	10306	40	91.76	11	3670.40	10/14/2004	Shipped	4	10	2004	...	171555155
1063	10419	35	100.00	6	5926.90	05/17/2005	Shipped	2	5	2005	...	6562955
0	10107	30	95.70	2	2871.00	02/24/2003	Shipped	1	2	2003	...	212555781

2823 rows × 24 columns

e. Filtering – I filtered the ORDER DATE and STATUS columns by which ones were in process in ascending order to see which order should be prioritized using the syntax:

filteredsales_df = sales_df[sales_df['STATUS'] == 'In Process']
sorted_filtered_df = filteredsales_df.sort_values(by='ORDERDATE', ascending=True)
sorted_filtered_df

```
# e. Filtering the In Process status and order date to see which one to prioritize

filteredsales_df = sales_df[sales_df['STATUS'] == 'In Process']
sorted_filtered_df = filteredsales_df.sort_values(by='ORDERDATE', ascending=True)
sorted_filtered_df
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	PHONE
855	10421	35	100.00	1	5433.75	05/29/2005	In Process	2	5	2005	...	415555
1515	10420	37	60.37	13	2233.69	05/29/2005	In Process	2	5	2005	...	6129495
1437	10420	60	64.67	11	3880.20	05/29/2005	In Process	2	5	2005	...	6129495
1414	10420	36	57.73	7	2078.28	05/29/2005	In Process	2	5	2005	...	6129495
1715	10420	39	100.00	9	3933.93	05/29/2005	In Process	2	5	2005	...	6129495

f. I created a CUSTOMER ID column in the incremental ID number to establish the uniqueness of the customer details. The syntax used was:

sales_df['CUSTOMERID']=range(1,len(sales_df)+1)
sales_df

```
#f. Create a new column names Customer ID and put incremental details

sales_df['CUSTOMERID']=range(1,len(sales_df)+1)
sales_df
```

YEAR_ID	...	ADDRESSLINE1	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE	CUSTOMERID
2003	...	897 Long Airport Avenue	NYC	NY	10022	USA		Yu	Kwai	Small	1
2003	...	59 rue de l'Abbaye	Reims	NA	51100	France	EMEA	Henriot	Paul	Small	2
2003	...	27 rue du Colonel Pierre Avia	Paris	NA	75508	France	EMEA	Da Cunha	Daniel	Medium	3
2003	...	78934 Hillside Dr.	Pasadena	CA	90003	USA		Young	Julie	Medium	4
2003	...	7734 Strong St.	San Francisco	CA	None	USA		Brown	Julie	Medium	5
...	...	...	...	...	...	...	...	...	...	...	...
2004	...	C/ Moralzarzal, 86	Madrid	NA	28034	Spain	EMEA	Freyre	Diego	Small	2819
2005	...	Torikatu 38	Oulu	NA	90110	Finland	EMEA	Koskitalo	Pirkko	Medium	2820
2005	...	C/ Moralzarzal, 86	Madrid	NA	28034	Spain	EMEA	Freyre	Diego	Medium	2821
2005	...	1 rue Alsace-Lorraine	Toulouse	NA	31000	France	EMEA	Roulet	Annette	Small	2822
2005	...	8616 Spinnaker Dr.	Boston	MA	51003	USA		Yoshido	Juri	Medium	2823

g. Aggregating the PRODUCT LINE column with the SALES column to get the total sales per product line. The syntax use was:

aggregated_data = sales_df.groupby('PRODUCTLINE')['SALES'].sum().reset_index()
aggregated_data

```
# g. Aggregated categorical column
aggregated_data = sales_df.groupby('PRODUCTLINE')['SALES'].sum().reset_index()
aggregated_data
```

	PRODUCTLINE	SALES
0	Classic Cars	3919615.66
1	Motorcycles	1166388.34
2	Planes	975003.57
3	Ships	714437.13
4	Trains	226243.47
5	Trucks and Buses	1127789.84
6	Vintage Cars	1903150.84

Task B

a. The PRODUCT LINE and STATUS were aggregated to show the summary statistics of SALES with their count per status. The syntax used was:

```
summary_stats = sales_df.groupby(['PRODUCTLINE', 'STATUS']).agg({
    'SALES': ['mean', 'sum', 'std'],
    'ORDERNUMBER': 'count'
}).reset_index()
summary_stats
```

	PRODUCTLINE	STATUS	SALES			ORDERNUMBER
			mean	sum	std	count
0	Classic Cars	Cancelled	3631.200833	43574.41	1147.023087	12
1	Classic Cars	In Process	3530.389091	38834.28	1107.403880	11
2	Classic Cars	On Hold	3077.958571	21545.71	797.978147	7
3	Classic Cars	Resolved	3844.761667	23068.57	895.653495	6
4	Classic Cars	Shipped	3464.477812	2026719.52	996.085443	585
5	Motorcycles	Disputed	3066.770000	9200.31	750.241755	3
6	Motorcycles	On Hold	4992.610000	4992.61	NaN	1
7	Motorcycles	Shipped	3229.456109	771840.01	971.185301	239
8	Planes	Cancelled	2859.080000	17154.48	504.403104	6
9	Planes	Disputed	2419.620000	2419.62	NaN	1
10	Planes	On Hold	3160.745000	18964.47	939.803621	6
11	Planes	Resolved	3358.007778	30222.07	810.210446	9
12	Planes	Shipped	3062.357170	649219.72	860.606497	212
13	Ships	Cancelled	3109.642000	46644.63	622.426695	15
14	Ships	Disputed	3070.400000	3070.40	NaN	1
15	Ships	On Hold	2958.076250	23664.61	433.912826	8
16	Ships	Resolved	3329.964000	33299.64	831.390686	10
17	Ships	Shipped	2994.070739	526956.45	834.283462	176
18	Trains	Cancelled	5082.420000	5082.42	NaN	1
19	Trains	Shipped	2839.127119	167508.50	922.627379	59
20	Trucks and Buses	In Process	3491.200000	20947.20	1038.500987	6
21	Trucks and Buses	On Hold	2086.920000	2086.92	NaN	1
22	Trucks and Buses	Resolved	3477.422500	13909.69	1109.316094	4
23	Trucks and Buses	Shipped	3396.479350	679295.87	1016.229372	200
24	Vintage Cars	Cancelled	3039.224167	36470.69	856.532936	12
25	Vintage Cars	In Process	3139.425000	25115.40	1591.612039	8
26	Vintage Cars	On Hold	3722.800000	18614.00	1228.940295	5
27	Vintage Cars	Resolved	3372.344000	16861.72	1039.679163	5
28	Vintage Cars	Shipped	3140.164343	1243505.08	972.067127	396

b. Four columns were chosen to get the correlations: QUANTITY ORDERED, PRICE EACH, SALES, and MSRP. The reason is that they were the columns that affected each other.

```
# b.
numeric_columns = ['QUANTITYORDERED', 'PRICEEACH', 'SALES',
                   'MSRP']

correlation_matrix = sales_df[numeric_columns].corr()
correlation_matrix
```

	QUANTITYORDERED	PRICEEACH	SALES	MSRP
QUANTITYORDERED	1.000000	0.005564	0.551426	0.017881
PRICEEACH	0.005564	1.000000	0.657841	0.670625
SALES	0.551426	0.657841	1.000000	0.635239
MSRP	0.017881	0.670625	0.635239	1.000000

c. Syntax used to export the dataset to a CSV file:

```
# c.

sales_df.to_csv('Sales.csv', index=False)
print('Successfully saved file: Sales.csv')
```

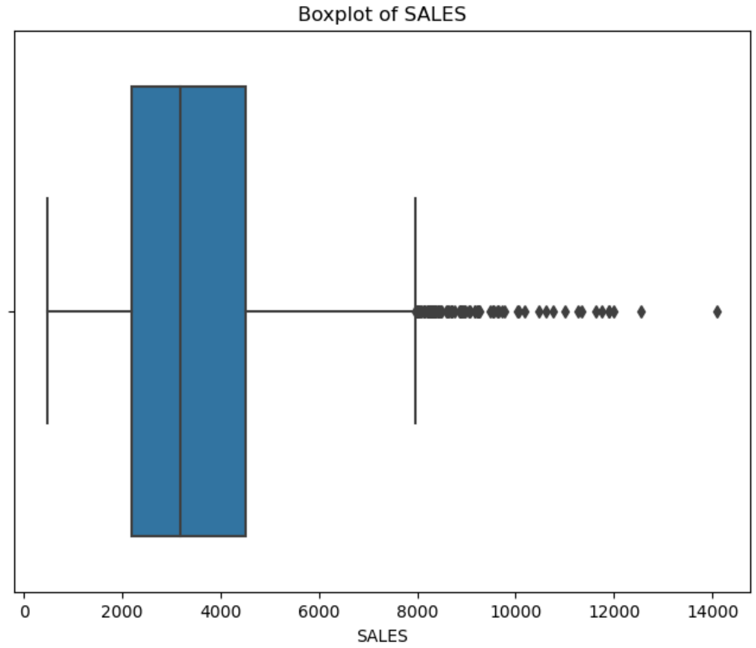
```
Successfully saved file: Sales.csv
```

d. Data Analysis and Visualization

I used the box plot to show the number of outliers in the SALES column. The shape of the dataset was 2,823 rows and 25 columns. Create a boxplot using Seaborn to identify outliers based on SALES data.

Syntax:

```
#
plt.figure(figsize=(8, 6))
sns.boxplot(x=sales_df['SALES'])
plt.title('Boxplot of SALES')
plt.xlabel('SALES')
plt.show()
```



I used the Z-scores to identify the outliers and remove them. The shape was reduced to 2005 rows and 25 columns.

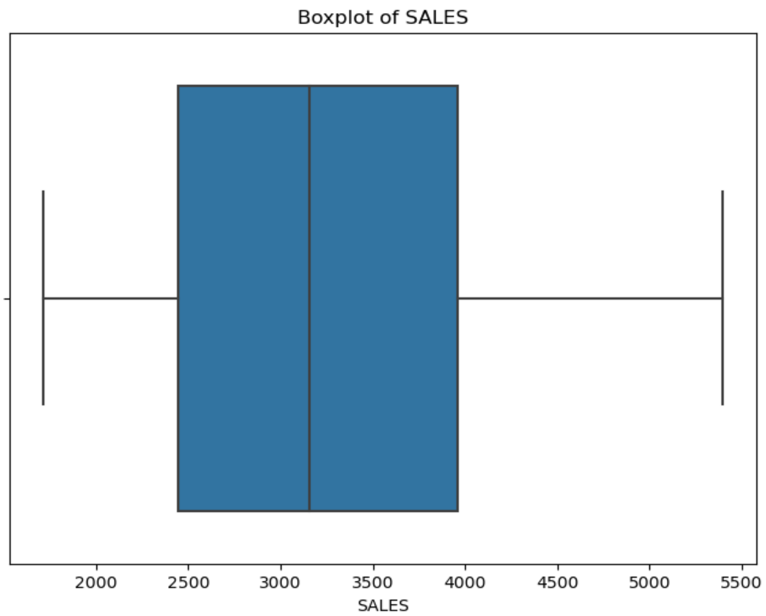
```
from scipy.stats import zscore

# Calculate Z-scores for the 'SALES' column
sales_df['Z_SCORES'] = zscore(sales_df['SALES'])

# Set a threshold for Z-scores to identify outliers
z_score_threshold = 1

# Filter the DataFrame to exclude outliers
sales_df = sales_df[abs(sales_df['Z_SCORES']) <= z_score_threshold]

# Drop the temporary column 'Z_SCORES'
sales_df = sales_df.drop(columns=['Z_SCORES'])
sales_df
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLI
0	10107	30	95.70	2	2871.00	02/24/2003	Shipped	1	2	2003	...	897 Long Ai Ave
1	10121	34	81.35	5	2765.90	05/07/2003	Shipped	2	5	2003	...	59 ru l'Abl
2	10134	41	94.74	2	3884.34	07/01/2003	Shipped	3	7	2003	...	27 ru Colonel P
3	10145	45	83.26	6	3746.70	08/25/2003	Shipped	3	8	2003	...	78934 Hil
4	10159	49	100.00	14	5205.27	10/10/2003	Shipped	4	10	2003	...	7734 Stron
...	...	...	...	...	...	...	...	...	...	...	...	
2817	10337	42	97.16	5	4080.72	11/21/2004	Shipped	4	11	2004	...	5905 Pom
2818	10350	20	100.00	15	2244.40	12/02/2004	Shipped	4	12	2004	...	C/ Moralza
2819	10373	29	100.00	1	3978.51	01/31/2005	Shipped	1	1	2005	...	Torikat
2821	10397	34	62.24	1	2116.16	03/28/2005	Shipped	1	3	2005	...	1 rue Als Lorr
2822	10414	47	65.52	9	3079.44	05/06/2005	On Hold	2	5	2005	...	8616 Spinn

2005 rows × 25 columns

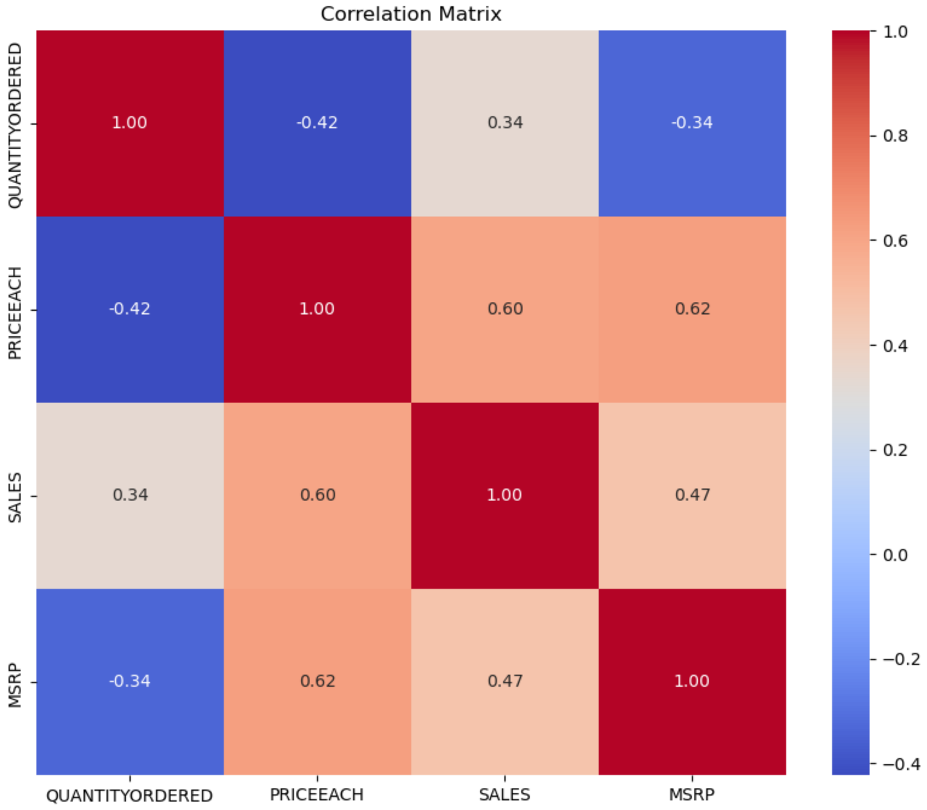
Box plot without the outliers:

```
# Boxplot without outliers
plt.figure(figsize=(8, 6))
sns.boxplot(x=sales_df['SALES'])
plt.title('Boxplot of SALES')
plt.xlabel('SALES')
plt.show()
```



The correlation matrix was done after removing the outliers to ensure the linearity between variables.

```
numeric_columns = ['QUANTITYORDERED', 'PRICEEACH', 'SALES',
                   'MSRP']

correlation_matrix = sales_df[numeric_columns].corr()
correlation_matrix

# Visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



I used the describe function to summarize the main features of the datasets.

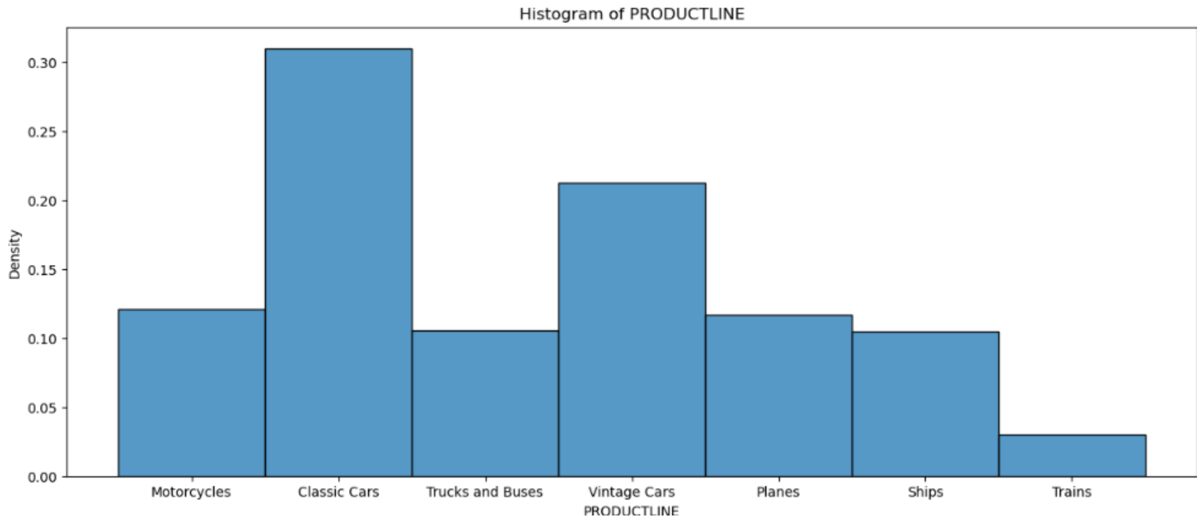
```
sales_df.describe()
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP	CUSTOMER
count	2005.000000	2005.000000	2005.000000	2005.000000	2005.000000	2005.000000	2005.000000	2005.000000	2005.000000	2005.0000
mean	10257.016958	34.563591	86.189845	6.642394	3252.263840	2.724190	7.104738	2003.803990	98.756608	1486.1685
std	91.365674	8.861406	16.590177	4.298528	973.191566	1.212558	3.685084	0.690674	33.814668	820.7139
min	10100.000000	12.000000	29.540000	1.000000	1713.690000	1.000000	1.000000	2003.000000	33.000000	1.0000
25%	10178.000000	27.000000	73.980000	3.000000	2443.290000	2.000000	4.000000	2003.000000	72.000000	790.0000
50%	10262.000000	34.000000	94.580000	6.000000	3157.440000	3.000000	8.000000	2004.000000	97.000000	1487.0000
75%	10331.000000	42.000000	100.000000	10.000000	3958.460000	4.000000	11.000000	2004.000000	118.000000	2221.0000
max	10425.000000	66.000000	100.000000	18.000000	5393.640000	4.000000	12.000000	2005.000000	214.000000	2823.0000

A histogram of the PRODUCT LINE was done to show the count of each product in a scaled density for better representation.

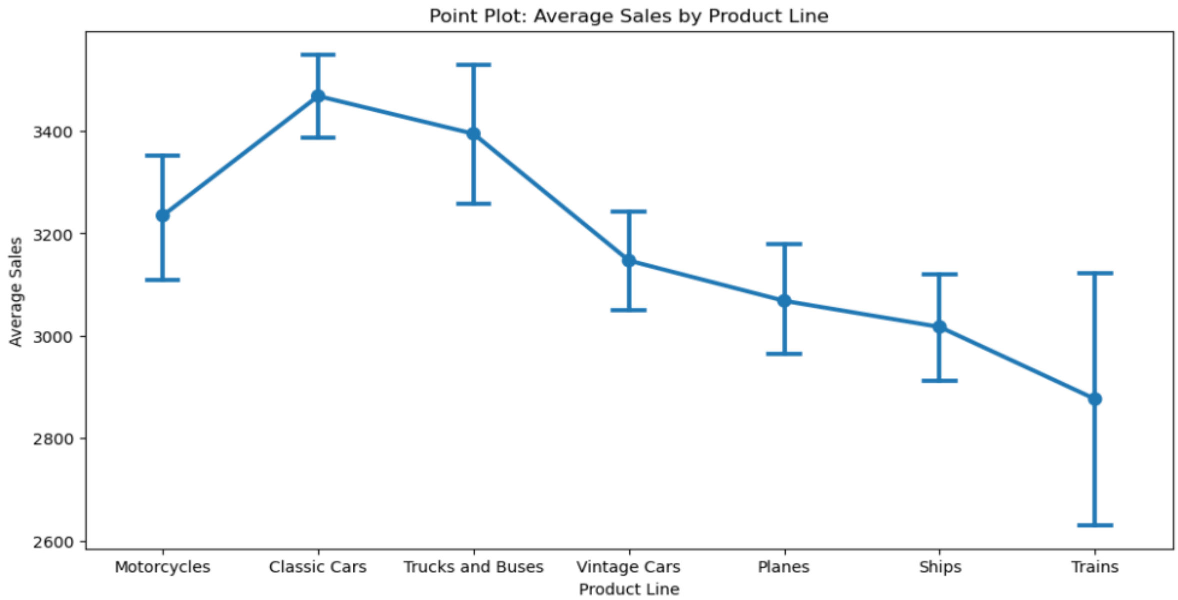
```
sales_df = sales_df.replace([np.inf, -np.inf], np.nan)

# Visualize a histogram of SALES
plt.figure(figsize=(15, 6))
sns.histplot(data=sales_df, x='PRODUCTLINE', element='bars', stat='density', common_norm=False, kde=False)
plt.title('Histogram of PRODUCTLINE')
plt.show()
```

```
C:\Users\torri\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



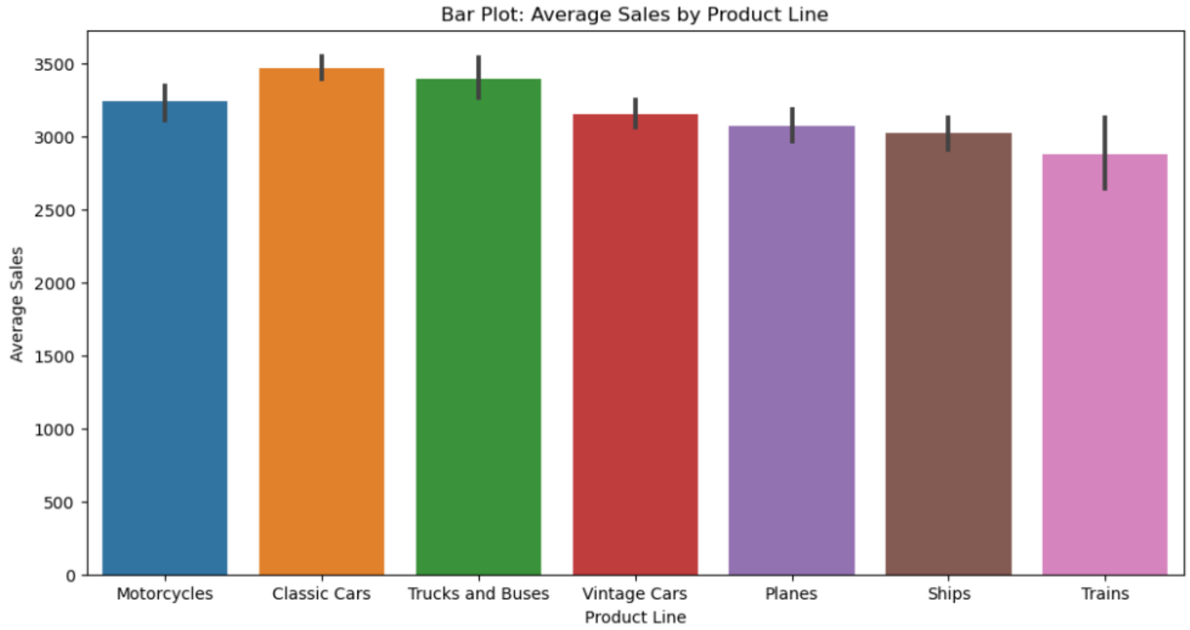
The plot point was made to show each product line's minimum and maximum average sales and central tendency and shows the comparison of each product line.

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.pointplot(x='PRODUCTLINE', y='SALES', data=sales_df, capsize=0.2)
plt.title('Point Plot: Average Sales by Product Line')
plt.xlabel('Product Line')
plt.ylabel('Average Sales')
plt.show()
```



Another method is using the bar plot.

```
#Bar Plot

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.barplot(x='PRODUCTLINE', y='SALES', data=sales_df)
plt.title('Bar Plot: Average Sales by Product Line')
plt.xlabel('Product Line')
plt.ylabel('Average Sales')
plt.show()
```



e. Analysis of Variance was done to test the significant differences in total sales amongst product lines.

```
#e.
# Analysis of Variance (ANOVA)
# Testing if there are any statistically significant differences among Product Lines.

from scipy.stats import f_oneway

grouped_data = [sales_df[sales_df['PRODUCTLINE'] == productline]['SALES'] for productline in sales_df['PRODUCTLINE'].unique()]

# Perform ANOVA
f_statistic, p_value = f_oneway(*grouped_data)

# Print the results
print(f"F-statistic: {f_statistic}")
print(f"P-value: {p_value}")

# Interpret the results
if p_value < 0.05:
    print("There are significant differences among PRODUCTLINE means.")
else:
    print("There are no significant differences among PRODUCTLINE means.")
```

```
F-statistic: 11.998883325816832
P-value: 2.8058279230390855e-13
There are significant differences among PRODUCTLINE means.
```

Task C

a. Analysis

Grouping and Summarizing
Calculate the total sales for each Product line.
Calculate the total quantity of orders for each Product line.
Calculate the Average Price for each Product line.

Analysis:
Classic Cars leads in sales, averaging 89 units per order across 621 transactions, while Trains exhibit the lowest total sales, averaging 77 units over 60 orders.

```
#a.

grouped_data = sales_df.groupby(['PRODUCTLINE']).agg({
    'SALES': ['sum'],
    'ORDERNUMBER': 'count',
    'QUANTITYORDERED': 'mean',
    'PRICEEACH': 'mean'
}).reset_index()

grouped_data = grouped_data.sort_values(by=('SALES', 'sum'), ascending=False)
grouped_data
```

	PRODUCTLINE	SALES	ORDERNUMBER	QUANTITYORDERED	PRICEEACH
		sum	count	mean	mean
0	Classic Cars	2153742.49	621	33.315620	89.450258
6	Vintage Cars	1340566.89	426	35.488263	83.643333
1	Motorcycles	786032.93	243	34.744856	85.616749
2	Planes	717980.36	234	35.692308	83.272906
5	Trucks and Buses	716239.68	211	34.364929	89.427725
3	Ships	633635.73	210	34.776190	84.934095
4	Trains	172590.92	60	35.733333	77.230500

Investigating Correlations
Understanding the correlations between Quantity Ordered and Sales:

- The correlation coefficient of 0.335612 suggests a positive relationship between Quantity Ordered and Sales. In other words, sales also tend to rise as the quantity ordered increases. However, it’s important to note that this correlation is not very strong.

Understanding the Correlations between Price Each and Sales:

- The total sales amount demonstrates a moderately to strongly positive relationship with the price of each product. As the price of each product rises, the sales amount tends to increase, as indicated by the correlation coefficient of 0.604662.

Understanding the Correlations between MSRP and Sales:

- The total sales amount exhibits a moderately positive relationship with the MSRP (Manufacturer’s Suggested Retail Price). As the MSRP increases, sales tend to rise, as the correlation coefficient 0.471174 indicates.

```
related_columns = ['QUANTITYORDERED', 'PRICEEACH',
                   'MSRP', 'SALES']

correlation_matrix = sales_df[related_columns].corr()
correlation_matrix
```

	QUANTITYORDERED	PRICEEACH	MSRP	SALES
QUANTITYORDERED	1.000000	-0.423357	-0.337305	0.335612
PRICEEACH	-0.423357	1.000000	0.623352	0.604682
MSRP	-0.337305	0.623352	1.000000	0.471174
SALES	0.335612	0.604682	0.471174	1.000000

Inferential Statistical Method
Using Analysis of Variance (ANOVA)

- Considering the p-value, it suggests a discernible difference in the average sales value for at least one product line compared to the others.

```
from scipy.stats import f_oneway

grouped_data = [sales_df[sales_df['PRODUCTLINE'] == productline]['SALES'] for productline in sales_df['PRODUCTLINE'].unique()]

# Perform ANOVA
f_statistic, p_value = f_oneway(*grouped_data)

# Print the results
print(f"F-statistic: {f_statistic}")
print(f"P-value: {p_value}")

# Interpret the results
if p_value < 0.05:
    print("There are significant differences among PRODUCTLINE means.")
else:
    print("There are no significant differences among PRODUCTLINE means.")
```

```
F-statistic: 11.998883325816832
P-value: 2.8058279230390855e-13
There are significant differences among PRODUCTLINE means.
```

b. Relationships between variables

Increased sales usually result from larger order quantities, as the two variables have a positive correlation. The 'Classic Cars' shows the highest average order quantity (621 transactions) among all product lines when sales are broken down by category. For the "Classic Cars" brand in particular, this information can help direct marketing tactics and inventory control. The sales and orders are noticeably highest in the 4th quarter. And regarding the geographical sales concentration, the USA has the highest sales.

```
# Grouping and Summarizing
grouped_data = sales_df.groupby(['QTR_ID','YEAR_ID']).agg({
    'SALES': ['sum', 'mean'],
    'QUANTITYORDERED': 'sum',
    'ORDERNUMBER': 'count'
}).reset_index()

grouped_data = grouped_data.sort_values(by='YEAR_ID')
grouped_data
```

	QTR_ID	YEAR_ID	SALES		QUANTITYORDERED	ORDERNUMBER
			sum	mean	sum	count
0	1	2003	322149.74	3254.037778	3439	99
3	2	2003	373837.87	3141.494706	4090	119
6	3	2003	425650.83	3299.618837	4288	129
8	4	2003	1208171.53	3301.015109	12516	366
1	1	2004	538661.11	3244.946446	5806	166
4	2	2004	477551.12	3100.981299	5265	154
7	3	2004	749028.62	3270.867336	7752	229
9	4	2004	1365788.46	3228.814326	14610	423
2	1	2005	725961.45	3314.892466	7686	219
5	2	2005	333988.27	3306.814554	3848	101

```
# Grouping and Summarizing
grouped_data = sales_df.groupby(['COUNTRY']).agg({
    'SALES': ['sum', 'mean'],
    'QUANTITYORDERED': 'sum',
    'ORDERNUMBER': 'count'
}).reset_index()

grouped_data = grouped_data.sort_values(by=('SALES','sum'), ascending=False)
grouped_data
```

	COUNTRY	SALES		QUANTITYORDERED	ORDERNUMBER
		sum	mean	sum	count
18	USA	2363474.28	3278.050319	24999	721
14	Spain	793625.96	3293.053776	8666	241
6	France	710609.24	3200.942523	7755	222
0	Australia	435914.58	3327.592214	4446	131
17	UK	355834.91	3234.862818	3772	110
9	Italy	270273.12	3296.013659	2795	82
5	Finland	212502.37	3269.267231	2243	65
11	Norway	178885.56	3252.464727	1811	55
3	Canada	178231.78	3020.877627	1879	59
13	Singapore	172338.02	3251.660755	1847	53
1	Austria	130121.66	3336.452821	1392	39
7	Germany	128958.19	3145.321707	1389	41
15	Sweden	125193.06	3210.078462	1348	39
4	Denmark	117850.27	3021.801795	1253	39
10	Japan	96442.59	2922.502727	1194	33
16	Switzerland	86661.96	3767.911304	796	23
2	Belgium	63621.70	3029.604762	663	21
12	Philippines	62113.33	3105.666500	713	20
8	Ireland	38136.42	3466.947273	339	11

Key findings

Product Line Performance:

The majority of sales are concentrated in the "Classic Cars" product line, which warrants further analysis and strategic attention.

Geographic Diversity:

The EMEA region leads in revenue and transaction volume, while Japan contributes less to the total sales, suggesting a smaller market share than other regions. Territories with unspecified codes also exhibit significant sales, indicating the need for additional investigation to determine and allocate these sales to specific regions.

Pricing and Sales Correlation:

Preliminary analysis suggests a potentially weak positive correlation between pricing and sales, emphasizing the importance of further investigation into pricing strategies.

c. Challenges and Suggestions

Challenges	**Suggestions**
The data indicates noticeable variations among product lines when it comes to sales.	Analyze the sales data for each product line. Consider implementing targeted marketing campaigns, product improvements, or repositioning to boost sales in the underperforming lines.
The correlation between pricing and sales exhibits a weak positive relationship.	Conduct a comprehensive pricing analysis. Implement dynamic pricing strategies, consider

	bundling products, or introduce loyalty programs to maximize the perceived value for customers.

Chapter 5

Observation

The dataset can be used on various business aspects mainly the marketing, sales, operations and customer service.

In the task A, we identified the null values and inconsistencies and then manage it by using various data cleaning techniques, to enhance the data quality and for accurate analysis. In task B, we analysed the data by first, describing it and using the correlations of the variables to see which have significant values on the analysis. Using inferential statistics (in my case, I used the ANOVA), we calculated the variance's probability, which can be used in sales forecasting.

Conclusion

This assessment suggests the importance of the correct data analysis and techniques. The relationships of the variables in the sales dataset contribute to better business analysis, focusing on the sales, marketing, and operations aspects, which significantly impact customer service.

Recommendations

I recommend further analysis using the variables Quarter ID, Month ID, and Year ID to investigate the current trend using the time-series analysis. This is also vital in sales forecasting, considering the consistency of sales. For marketing aspects, the clustering can be done using the state, postal code, country, and territory variables to analyze the concentration of sales. This will help the marketing department develop a new marketing strategy for the locations with low sales and canceled orders. In operations, monitoring the order status and deal size can be described using the central tendency.

In [1]:
```
import pandas as pd
import numpy as np
```

```
C:\Users\torri\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pand
as requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
# Task A
# a.
sales_df=pd.read_csv('Sales_Sample_Public_Dataset.csv')
sales_df
```

Out[2]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped
...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004 0:00	Shipped
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped
2820	10386	43	100.00	4	5417.57	3/01/2005 0:00	Resolved
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped
2822	10414	47	65.52	9	3079.44	5/06/2005 0:00	On Hold

2823 rows × 25 columns

In [3]:
```
sales_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ORDERNUMBER       2823 non-null   int64  
 1   QUANTITYORDERED   2823 non-null   int64  
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64  
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object 
 6   STATUS            2823 non-null   object 
 7   QTR_ID            2823 non-null   int64  
 8   MONTH_ID          2823 non-null   int64  
 9   YEAR_ID           2823 non-null   int64  
 10  PRODUCTLINE       2823 non-null   object 
 11  MSRP              2823 non-null   int64  
 12  PRODUCTCODE       2823 non-null   object 
 13  CUSTOMERNAME      2823 non-null   object 
 14  PHONE             2823 non-null   object 
 15  ADDRESSLINE1      2823 non-null   object 
 16  ADDRESSLINE2      302 non-null    object 
 17  CITY              2823 non-null   object 
 18  STATE             1337 non-null   object 
 19  POSTALCODE        2747 non-null   object 
 20  COUNTRY           2823 non-null   object 
 21  TERRITORY         1749 non-null   object 
 22  CONTACTLASTNAME   2823 non-null   object 
 23  CONTACTFIRSTNAME  2823 non-null   object 
 24  DEALSIZE          2823 non-null   object 
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

In [4]:
```
sales_df.describe
```

Out[4]:

```
<bound method NDFrame.describe of       ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUM
BER    SALES  \
0           10107               30      95.70                2  2871.00
1           10121               34      81.35                5  2765.90
2           10134               41      94.74                2  3884.34
3           10145               45      83.26                6  3746.70
4           10159               49     100.00               14  5205.27
...           ...              ...        ...              ...      ...
2818        10350               20     100.00               15  2244.40
2819        10373               29     100.00                1  3978.51
2820        10386               43     100.00                4  5417.57
2821        10397               34      62.24                1  2116.16
2822        10414               47      65.52                9  3079.44

            ORDERDATE    STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  \
0      2/24/2003 0:00   Shipped       1         2     2003  ...
1      5/07/2003 0:00   Shipped       2         5     2003  ...
2      7/01/2003 0:00   Shipped       3         7     2003  ...
3      8/25/2003 0:00   Shipped       3         8     2003  ...
4     10/10/2003 0:00   Shipped       4        10     2003  ...
...               ...       ...     ...       ...      ...  ...
2818  12/02/2004 0:00   Shipped       4        12     2004  ...
2819   1/31/2005 0:00   Shipped       1         1     2005  ...
2820   3/01/2005 0:00  Resolved       1         3     2005  ...
2821   3/28/2005 0:00   Shipped       1         3     2005  ...
2822   5/06/2005 0:00   On Hold       2         5     2005  ...

                        ADDRESSLINE1  ADDRESSLINE2           CITY STATE  \
0            897 Long Airport Avenue           NaN            NYC    NY
1                 59 rue de l'Abbaye           NaN          Reims   NaN
2      27 rue du Colonel Pierre Avia           NaN          Paris   NaN
3                 78934 Hillside Dr.           NaN       Pasadena    CA
4                    7734 Strong St.           NaN  San Francisco    CA
...                              ...           ...            ...   ...
2818             C/ Moralzarzal, 86           NaN         Madrid   NaN
2819                    Torikatu 38           NaN           Oulu   NaN
2820             C/ Moralzarzal, 86           NaN         Madrid   NaN
2821          1 rue Alsace-Lorraine           NaN       Toulouse   NaN
2822              8616 Spinnaker Dr.           NaN         Boston    MA

     POSTALCODE  COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
0         10022      USA       NaN              Yu             Kwai    Small
1         51100   France      EMEA         Henriot             Paul    Small
2         75508   France      EMEA        Da Cunha           Daniel   Medium
3         90003      USA       NaN           Young            Julie   Medium
4           NaN      USA       NaN           Brown            Julie   Medium
...         ...      ...       ...             ...              ...      ...
2818      28034    Spain      EMEA          Freyre            Diego    Small
2819      90110  Finland      EMEA       Koskitalo           Pirkko   Medium
2820      28034    Spain      EMEA          Freyre            Diego   Medium
2821      31000   France      EMEA          Roulet          Annette    Small
2822      51003      USA       NaN         Yoshido             Juri   Medium

[2823 rows x 25 columns]>
```

In [5]:

```
# b.
sales_df.head(10)
```

Out[5]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTI
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped	
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped	
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	
5	10168	36	96.66	1	3479.76	10/28/2003 0:00	Shipped	
6	10180	29	86.13	9	2497.77	11/11/2003 0:00	Shipped	
7	10188	48	100.00	1	5512.32	11/18/2003 0:00	Shipped	
8	10201	22	98.57	2	2168.54	12/01/2003 0:00	Shipped	
9	10211	41	100.00	14	4708.44	1/15/2004 0:00	Shipped	

10 rows × 25 columns

In [6]:

```
#c.1 Identifying the null values
sales_df.isnull().sum()
```

Out[6]:

```
ORDERNUMBER            0
QUANTITYORDERED        0
PRICEEACH              0
ORDERLINENUMBER        0
SALES                  0
ORDERDATE              0
STATUS                 0
QTR_ID                 0
MONTH_ID               0
YEAR_ID                0
PRODUCTLINE            0
MSRP                   0
PRODUCTCODE            0
CUSTOMERNAME           0
PHONE                  0
ADDRESSLINE1           0
ADDRESSLINE2        2521
CITY                   0
STATE               1486
POSTALCODE            76
COUNTRY                0
TERRITORY           1074
CONTACTLASTNAME        0
CONTACTFIRSTNAME       0
DEALSIZE               0
dtype: int64
```

In [7]:

```
# c.2 : Replacing TERRITORY null values by empty value
sales_df['TERRITORY']=sales_df['TERRITORY'].fillna('')
sales_df
```

Out[7]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped
...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004 0:00	Shipped
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped
2820	10386	43	100.00	4	5417.57	3/01/2005 0:00	Resolved
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped
2822	10414	47	65.52	9	3079.44	5/06/2005 0:00	On Hold

2823 rows × 25 columns

In [8]:

```
#c.2
sales_df['STATE']=sales_df['STATE'].fillna('NA')
sales_df
```

Out[8]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped
...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004 0:00	Shipped
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped
2820	10386	43	100.00	4	5417.57	3/01/2005 0:00	Resolved
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped
2822	10414	47	65.52	9	3079.44	5/06/2005 0:00	On Hold

2823 rows × 25 columns

In [9]:

```
# c.3 : Dropping ADDRESSLINE2 as it is not needed in the analysis
sales_df=sales_df.drop(columns=['ADDRESSLINE2'])
sales_df
```

Out[9]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped
...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004 0:00	Shipped
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped
2820	10386	43	100.00	4	5417.57	3/01/2005 0:00	Resolved
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped
2822	10414	47	65.52	9	3079.44	5/06/2005 0:00	On Hold

2823 rows × 24 columns

In [10]:

```
#c.2
sales_df['POSTALCODE']=sales_df['POSTALCODE'].fillna('None')
sales_df
```

Out[10]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped
...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004 0:00	Shipped
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped
2820	10386	43	100.00	4	5417.57	3/01/2005 0:00	Resolved
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped
2822	10414	47	65.52	9	3079.44	5/06/2005 0:00	On Hold

2823 rows × 24 columns

In [11]:

```
sales_df.isnull().sum()
```

Out[11]:

```
ORDERNUMBER         0
QUANTITYORDERED     0
PRICEEACH           0
ORDERLINENUMBER     0
SALES               0
ORDERDATE           0
STATUS              0
QTR_ID              0
MONTH_ID            0
YEAR_ID             0
PRODUCTLINE         0
MSRP                0
PRODUCTCODE         0
CUSTOMERNAME        0
PHONE               0
ADDRESSLINE1        0
CITY                0
STATE               0
POSTALCODE          0
COUNTRY             0
TERRITORY           0
CONTACTLASTNAME     0
CONTACTFIRSTNAME    0
DEALSIZE            0
dtype: int64
```

In [12]:

```
# Checking for duplicates
duplicated_rows = sales_df[sales_df.duplicated()]
duplicated_rows
```

Out[12]:

ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_I

0 rows × 24 columns

In [13]:

```
# formatting the PHONE number

sales_df['PHONE'] = sales_df['PHONE'].str.replace(r'\D', '', regex=True)
sales_df
```

Out[13]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped
1	10121	34	81.35	5	2765.90	5/07/2003 0:00	Shipped
2	10134	41	94.74	2	3884.34	7/01/2003 0:00	Shipped
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped
...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004 0:00	Shipped
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped
2820	10386	43	100.00	4	5417.57	3/01/2005 0:00	Resolved
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped
2822	10414	47	65.52	9	3079.44	5/06/2005 0:00	On Hold

2823 rows × 24 columns

In [14]:

```
#removing the date stamp

sales_df['ORDERDATE'] = pd.to_datetime(sales_df['ORDERDATE'], format='%m/%d/%Y %H:%M')
sales_df['ORDERDATE'] = sales_df['ORDERDATE'].dt.strftime('%m/%d/%Y')
sales_df
```

Out[14]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	02/24/2003	Shipped
1	10121	34	81.35	5	2765.90	05/07/2003	Shipped
2	10134	41	94.74	2	3884.34	07/01/2003	Shipped
3	10145	45	83.26	6	3746.70	08/25/2003	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003	Shipped
...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004	Shipped
2819	10373	29	100.00	1	3978.51	01/31/2005	Shipped
2820	10386	43	100.00	4	5417.57	03/01/2005	Resolved
2821	10397	34	62.24	1	2116.16	03/28/2005	Shipped
2822	10414	47	65.52	9	3079.44	05/06/2005	On Hold

2823 rows × 24 columns

In [15]:

```
# d. sorting status

sorted_df = sales_df.sort_values(by='STATUS', ascending=True)
sorted_df
```

Out[15]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
638	10253	33	100.00	4	4459.62	06/01/2004	Cancelled
2593	10167	46	70.11	10	3225.06	10/23/2003	Cancelled
1899	10167	32	63.12	3	2019.84	10/23/2003	Cancelled
117	10248	20	100.00	3	2910.40	05/07/2004	Cancelled
2573	10262	33	90.75	6	2994.75	06/24/2004	Cancelled
...	...	...	...	...	...	...	...
1079	10282	36	100.00	3	4174.92	08/20/2004	Shipped
1080	10293	22	100.00	6	2418.24	09/09/2004	Shipped
1081	10306	40	91.76	11	3670.40	10/14/2004	Shipped
1063	10419	35	100.00	6	5926.90	05/17/2005	Shipped
0	10107	30	95.70	2	2871.00	02/24/2003	Shipped

2823 rows × 24 columns

In [16]:

```
# e. Filtering the In Process status and order date to see which one to prioritize

filteredsales_df = sales_df[sales_df['STATUS'] == 'In Process']
sorted_filtered_df = filteredsales_df.sort_values(by='ORDERDATE', ascending=True)
sorted_filtered_df
```

Out[16]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
855	10421	35	100.00	1	5433.75	05/29/2005	In Process
1515	10420	37	60.37	13	2233.69	05/29/2005	In Process
1437	10420	60	64.67	11	3880.20	05/29/2005	In Process
1414	10420	36	57.73	7	2078.28	05/29/2005	In Process
1715	10420	39	100.00	9	3933.93	05/29/2005	In Process
1288	10420	66	92.95	6	6134.70	05/29/2005	In Process
1791	10420	55	96.30	8	5296.50	05/29/2005	In Process
1563	10420	45	26.88	1	1209.60	05/29/2005	In Process
1867	10420	35	96.74	10	3385.90	05/29/2005	In Process
752	10420	45	100.00	2	4977.00	05/29/2005	In Process
1640	10421	40	45.70	2	1828.00	05/29/2005	In Process
701	10420	36	63.57	4	2288.52	05/29/2005	In Process
2045	10420	15	43.49	3	652.35	05/29/2005	In Process
599	10420	37	100.00	5	5283.60	05/29/2005	In Process
1943	10420	26	100.00	12	2617.16	05/29/2005	In Process
2097	10423	28	78.89	4	2208.92	05/30/2005	In Process
1139	10423	21	89.29	5	1875.09	05/30/2005	In Process
500	10422	51	95.55	2	4873.05	05/30/2005	In Process
987	10423	21	84.82	2	1781.22	05/30/2005	In Process
934	10423	31	53.72	3	1665.32	05/30/2005	In Process

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
907	10423	10	88.14	1	881.40	05/30/2005	In Process
526	10422	25	51.75	1	1293.75	05/30/2005	In Process
1743	10425	31	33.24	5	1030.44	05/31/2005	In Process
2172	10425	41	86.68	11	3553.88	05/31/2005	In Process
2249	10425	11	43.83	6	482.13	05/31/2005	In Process
1667	10425	49	100.00	9	5510.54	05/31/2005	In Process
2302	10424	44	61.41	2	2702.04	05/31/2005	In Process
53	10424	50	100.00	6	12001.00	05/31/2005	In Process
1341	10425	38	100.00	13	4325.16	05/31/2005	In Process
1368	10424	26	59.87	4	1556.62	05/31/2005	In Process
2405	10425	18	100.00	2	1895.94	05/31/2005	In Process
1064	10425	28	100.00	8	3793.16	05/31/2005	In Process
780	10425	19	49.22	10	935.18	05/31/2005	In Process
727	10425	38	99.41	7	3777.58	05/31/2005	In Process
679	10425	28	100.00	3	5318.04	05/31/2005	In Process
447	10424	54	100.00	5	7182.00	05/31/2005	In Process
393	10425	33	100.00	4	4692.60	05/31/2005	In Process
239	10424	49	100.00	3	7969.36	05/31/2005	In Process
160	10425	38	100.00	12	5894.94	05/31/2005	In Process
1465	10425	55	46.82	1	2575.10	05/31/2005	In Process

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
2640	10424	46	80.92	1	3722.32	05/31/2005	In Process

41 rows × 24 columns

In [17]:
```
#f. Create a new column names Customer ID and put incremental details

sales_df['CUSTOMERID']=range(1,len(sales_df)+1)
sales_df
```

Out[17]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	02/24/2003	Shipped
1	10121	34	81.35	5	2765.90	05/07/2003	Shipped
2	10134	41	94.74	2	3884.34	07/01/2003	Shipped
3	10145	45	83.26	6	3746.70	08/25/2003	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003	Shipped
...	...	...	...	...	...	...	...
2818	10350	20	100.00	15	2244.40	12/02/2004	Shipped
2819	10373	29	100.00	1	3978.51	01/31/2005	Shipped
2820	10386	43	100.00	4	5417.57	03/01/2005	Resolved
2821	10397	34	62.24	1	2116.16	03/28/2005	Shipped
2822	10414	47	65.52	9	3079.44	05/06/2005	On Hold

2823 rows × 25 columns

In [18]:
```
# g. Aggregated categorical column
aggregated_data = sales_df.groupby('PRODUCTLINE')['SALES'].sum().reset_index()
aggregated_data
```

Out[18]:

	PRODUCTLINE	SALES
0	Classic Cars	3919615.66
1	Motorcycles	1166388.34
2	Planes	975003.57
3	Ships	714437.13
4	Trains	226243.47
5	Trucks and Buses	1127789.84
6	Vintage Cars	1903150.84

In [19]:
```
# Task B - Data Analysis

#a .
summary_stats = sales_df.groupby(['PRODUCTLINE', 'STATUS']).agg({
    'SALES': ['mean', 'sum', 'std'],
    'ORDERNUMBER': 'count'
}).reset_index()
summary_stats
```

Out[19]:

	PRODUCTLINE	STATUS			SALES	ORDERNUMBER
			mean	sum	std	count
0	Classic Cars	Cancelled	3702.675625	59242.81	1608.945688	16
1	Classic Cars	Disputed	8670.956667	26012.87	1669.641725	3
2	Classic Cars	In Process	4125.761429	57760.66	2644.195007	14
3	Classic Cars	On Hold	4086.637500	49039.65	2061.881799	12
4	Classic Cars	Resolved	3224.917500	25799.34	1375.863172	8
5	Classic Cars	Shipped	4050.066007	3701760.33	2038.922560	914
6	Motorcycles	Disputed	5303.650000	31821.90	2709.079642	6
7	Motorcycles	On Hold	4992.610000	4992.61	NaN	1
8	Motorcycles	Shipped	3486.338981	1129573.83	1807.832891	324
9	Planes	Cancelled	2952.725833	35432.71	1679.434792	12
10	Planes	Disputed	1921.920000	3843.84	703.854090	2
11	Planes	On Hold	3858.614444	34727.53	2200.511530	9
12	Planes	Resolved	2877.743333	34532.92	1111.173212	12
13	Planes	Shipped	3197.293616	866466.57	1504.694860	271
14	Ships	Cancelled	3148.091667	56665.65	1217.907341	18
15	Ships	Disputed	3070.400000	3070.40	NaN	1
16	Ships	On Hold	2958.076250	23664.61	433.912826	8
17	Ships	Resolved	3321.975833	39863.71	1181.129751	12
18	Ships	Shipped	3031.655179	591172.76	1078.229796	195
19	Trains	Cancelled	5082.420000	5082.42	NaN	1
20	Trains	On Hold	5808.480000	5808.48	NaN	1
21	Trains	Shipped	2871.367600	215352.57	1414.576053	75
22	Trucks and Buses	In Process	3911.491818	43026.41	2383.080366	11
23	Trucks and Buses	On Hold	5048.322500	20193.29	2043.636003	4
24	Trucks and Buses	Resolved	4094.550000	20472.75	1681.418569	5
25	Trucks and Buses	Shipped	3715.649075	1044097.39	1636.431042	281
26	Vintage Cars	Cancelled	2927.991538	38063.89	912.883536	13
27	Vintage Cars	Disputed	3731.925000	7463.85	3503.466613	2
28	Vintage Cars	In Process	2746.430625	43942.89	1872.437010	16
29	Vintage Cars	On Hold	4505.891111	40553.02	4030.924180	9
30	Vintage Cars	Resolved	3004.956000	30049.56	2464.721520	10
31	Vintage Cars	Shipped	3129.403285	1743077.63	1725.862196	557

In [20]:
```
sales_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ORDERNUMBER       2823 non-null   int64  
 1   QUANTITYORDERED   2823 non-null   int64  
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64  
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object 
 6   STATUS            2823 non-null   object 
 7   QTR_ID            2823 non-null   int64  
 8   MONTH_ID          2823 non-null   int64  
 9   YEAR_ID           2823 non-null   int64  
 10  PRODUCTLINE       2823 non-null   object 
 11  MSRP              2823 non-null   int64  
 12  PRODUCTCODE       2823 non-null   object 
 13  CUSTOMERNAME      2823 non-null   object 
 14  PHONE             2823 non-null   object 
 15  ADDRESSLINE1      2823 non-null   object 
 16  CITY              2823 non-null   object 
 17  STATE             2823 non-null   object 
 18  POSTALCODE        2823 non-null   object 
 19  COUNTRY           2823 non-null   object 
 20  TERRITORY         2823 non-null   object 
 21  CONTACTLASTNAME   2823 non-null   object 
 22  CONTACTFIRSTNAME  2823 non-null   object 
 23  DEALSIZE          2823 non-null   object 
 24  CUSTOMERID        2823 non-null   int64  
dtypes: float64(2), int64(8), object(15)
memory usage: 551.5+ KB
```

In [21]:
```
# b.
numeric_columns = ['QUANTITYORDERED', 'PRICEEACH', 'SALES',
                   'MSRP']

correlation_matrix = sales_df[numeric_columns].corr()
correlation_matrix
```

Out[21]:

	QUANTITYORDERED	PRICEEACH	SALES	MSRP
QUANTITYORDERED	1.000000	0.005564	0.551426	0.017881
PRICEEACH	0.005564	1.000000	0.657841	0.670625
SALES	0.551426	0.657841	1.000000	0.635239
MSRP	0.017881	0.670625	0.635239	1.000000

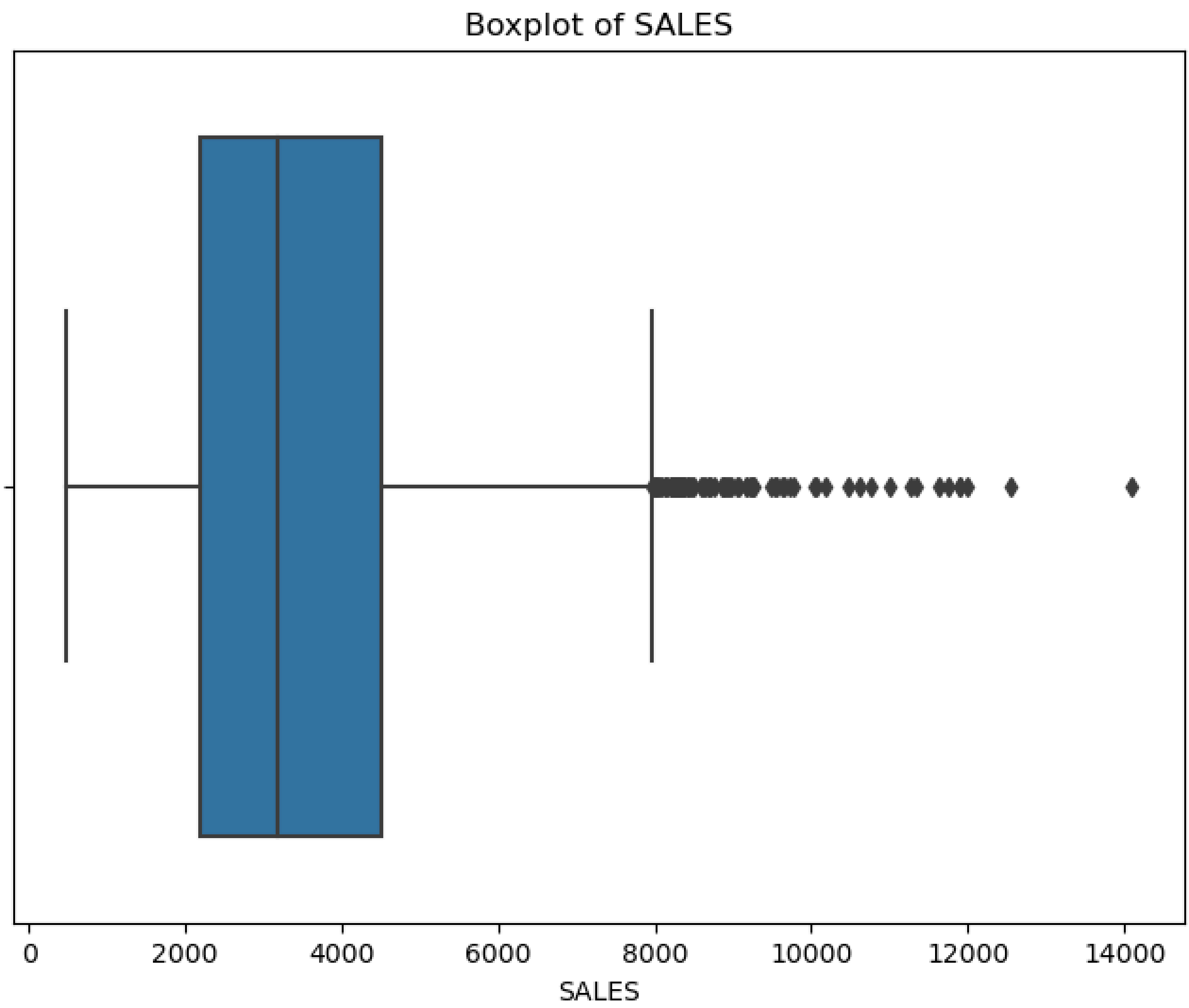
In [22]:
```
# c.

sales_df.to_csv('Sales.csv', index=False)
print('Successfully saved file: Sales.csv')
```

```
Successfully saved file: Sales.csv
```

In [23]:
```
# d.

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [24]:
```
#
plt.figure(figsize=(8, 6))
sns.boxplot(x=sales_df['SALES'])
```

```
plt.title('Boxplot of SALES')
plt.xlabel('SALES')
plt.show()
```



In [25]:
```
sales_df.shape
```

Out[25]:
```
(2823, 25)
```

In [26]:
```
# Perform removal of outliers using ZScore

from scipy.stats import zscore

# Calculate Z-scores for the 'SALES' column
sales_df['Z_SCORES'] = zscore(sales_df['SALES'])

# Set a threshold for Z-scores to identify outliers
z_score_threshold = 1

# Filter the DataFrame to exclude outliers
sales_df = sales_df[abs(sales_df['Z_SCORES']) <= z_score_threshold]

# Drop the temporary column 'Z_SCORES'
sales_df = sales_df.drop(columns=['Z_SCORES'])
sales_df
```

Out[26]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS
0	10107	30	95.70	2	2871.00	02/24/2003	Shipped
1	10121	34	81.35	5	2765.90	05/07/2003	Shipped
2	10134	41	94.74	2	3884.34	07/01/2003	Shipped
3	10145	45	83.26	6	3746.70	08/25/2003	Shipped
4	10159	49	100.00	14	5205.27	10/10/2003	Shipped
...	...	...	...	...	...	...	...
2817	10337	42	97.16	5	4080.72	11/21/2004	Shipped
2818	10350	20	100.00	15	2244.40	12/02/2004	Shipped
2819	10373	29	100.00	1	3978.51	01/31/2005	Shipped
2821	10397	34	62.24	1	2116.16	03/28/2005	Shipped
2822	10414	47	65.52	9	3079.44	05/06/2005	On Hold

2005 rows × 25 columns

In [27]:

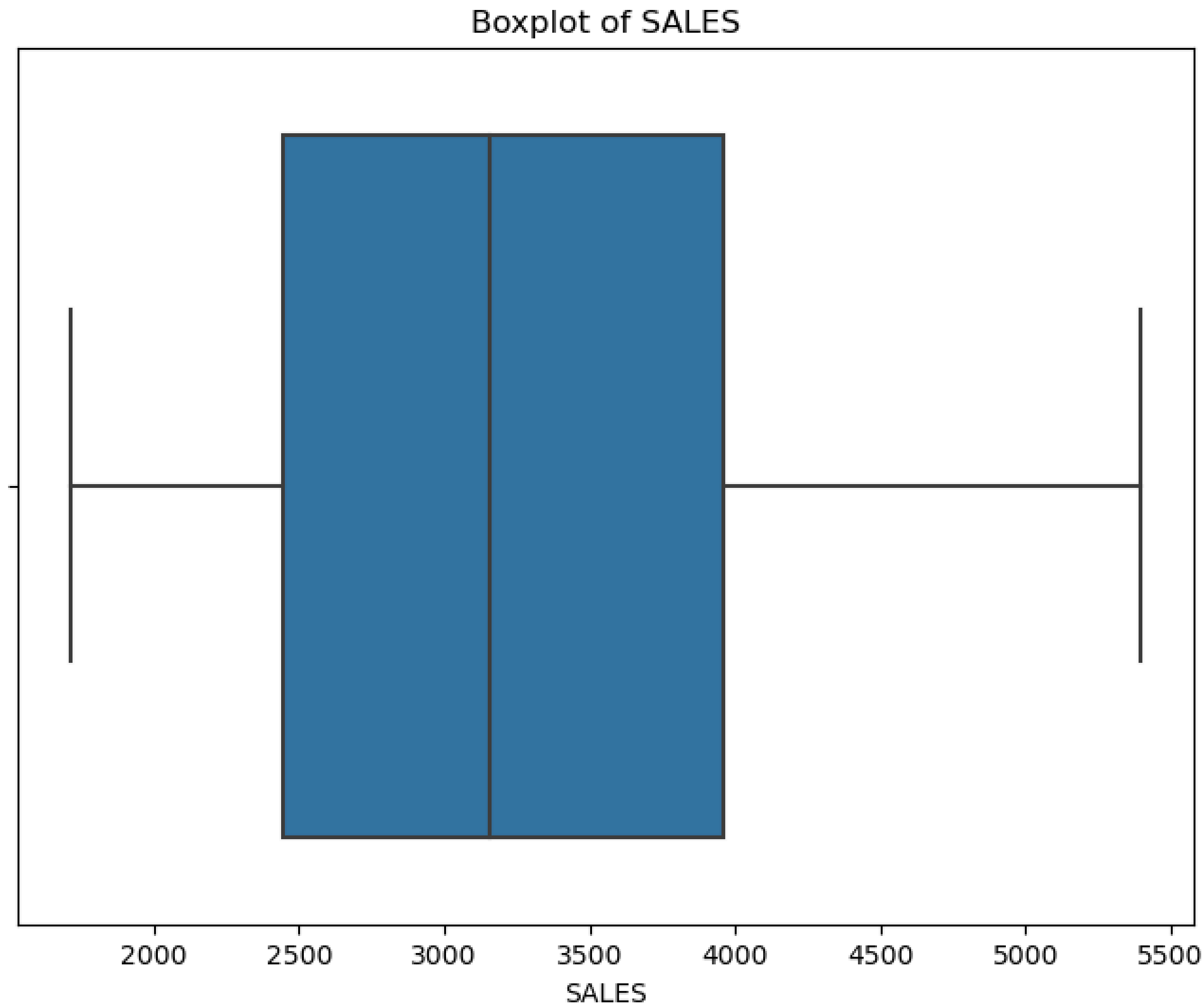
```
sales_df.shape
```

Out[27]:

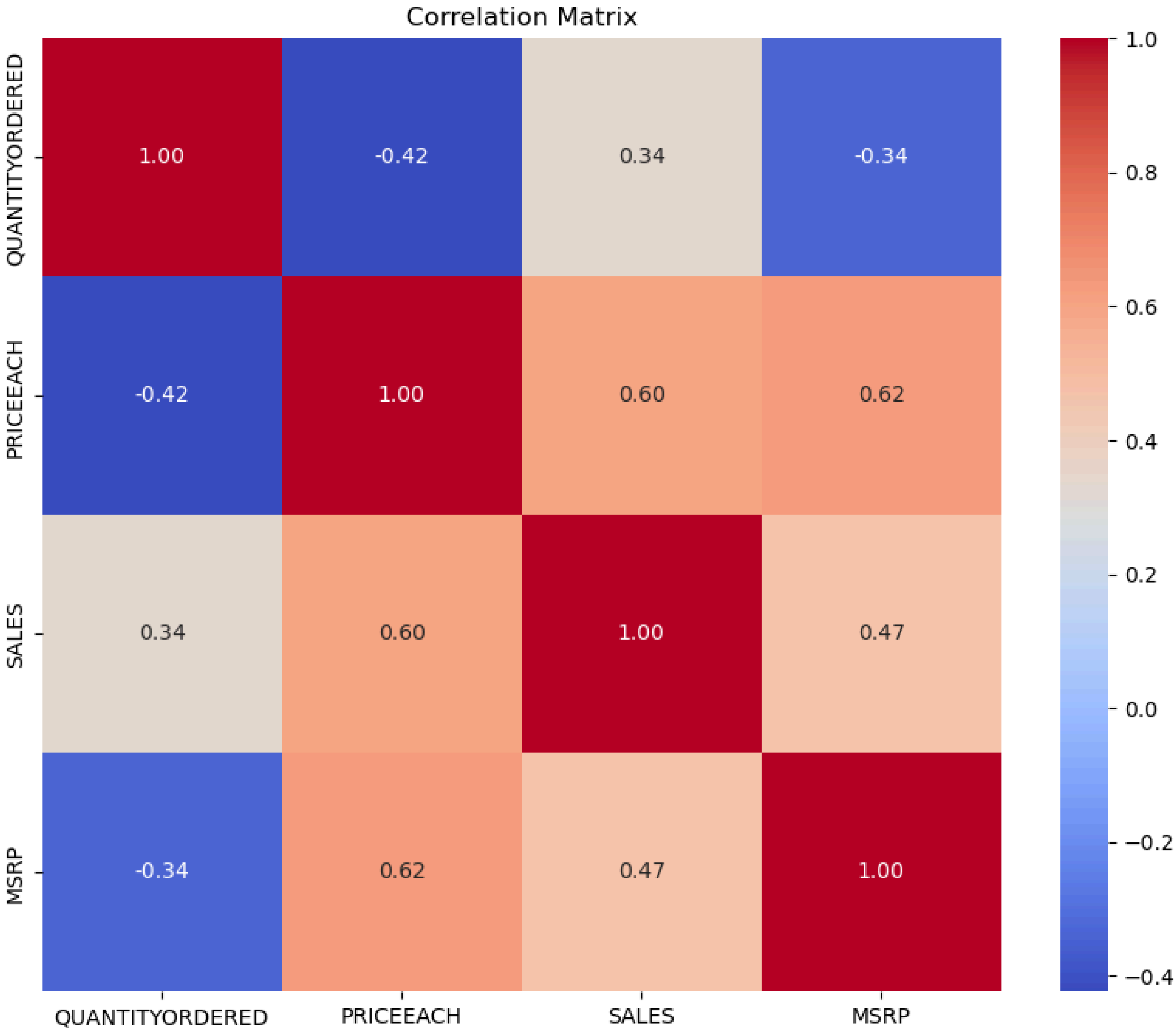
```
(2005, 25)
```

In [28]:

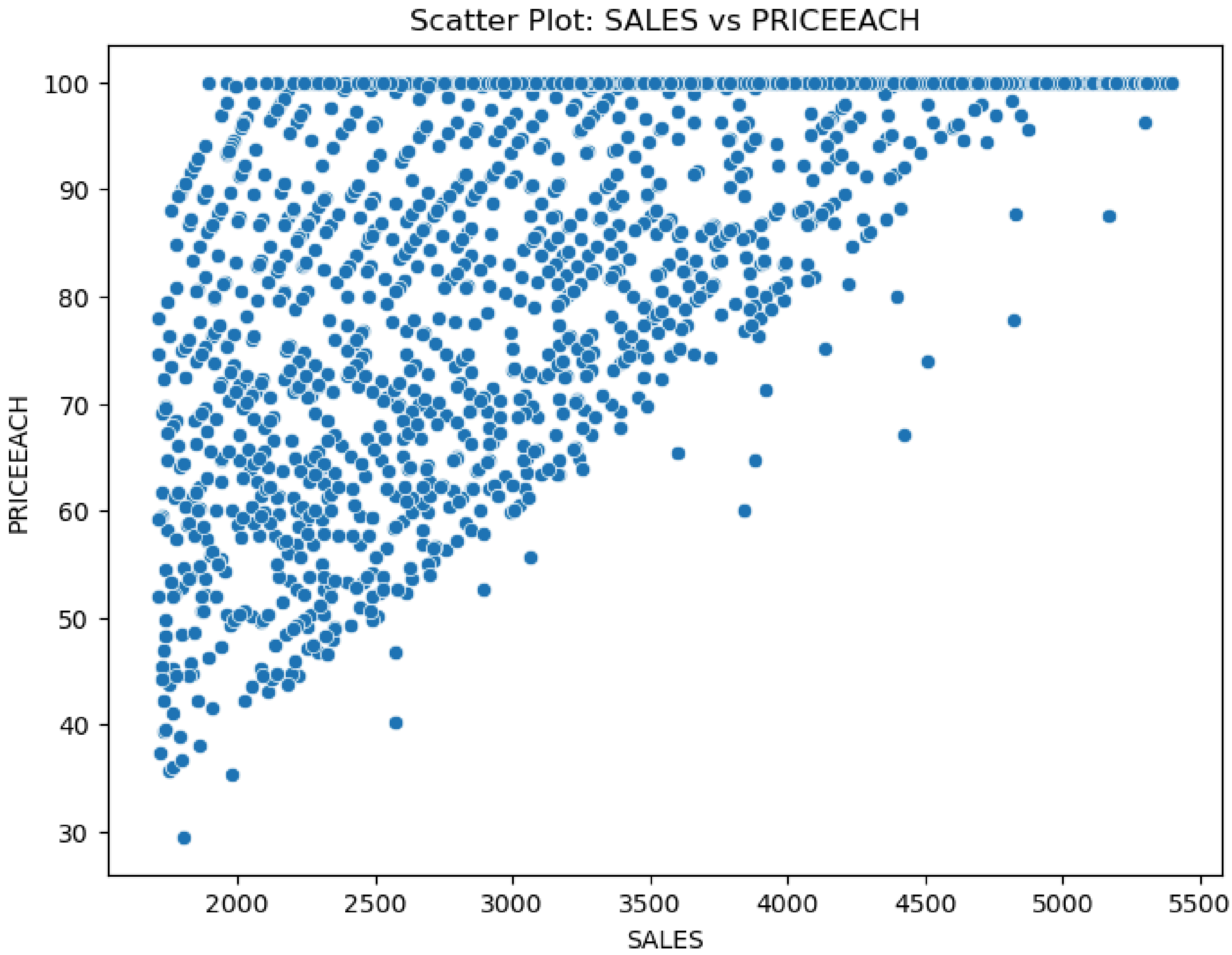
```
# Boxplot without outliers
plt.figure(figsize=(8, 6))
sns.boxplot(x=sales_df['SALES'])
plt.title('Boxplot of SALES')
plt.xlabel('SALES')
plt.show()
```



In [29]:
```
numeric_columns = ['QUANTITYORDERED', 'PRICEEACH', 'SALES',
                    'MSRP']

correlation_matrix = sales_df[numeric_columns].corr()
correlation_matrix

# Visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



In [30]:
```
# Visualize a scatter plot of SALES and PRICEEACH
plt.figure(figsize=(8, 6))
sns.scatterplot(data=sales_df, x='SALES', y='PRICEEACH')
plt.title('Scatter Plot: SALES vs PRICEEACH')
plt.show()
```



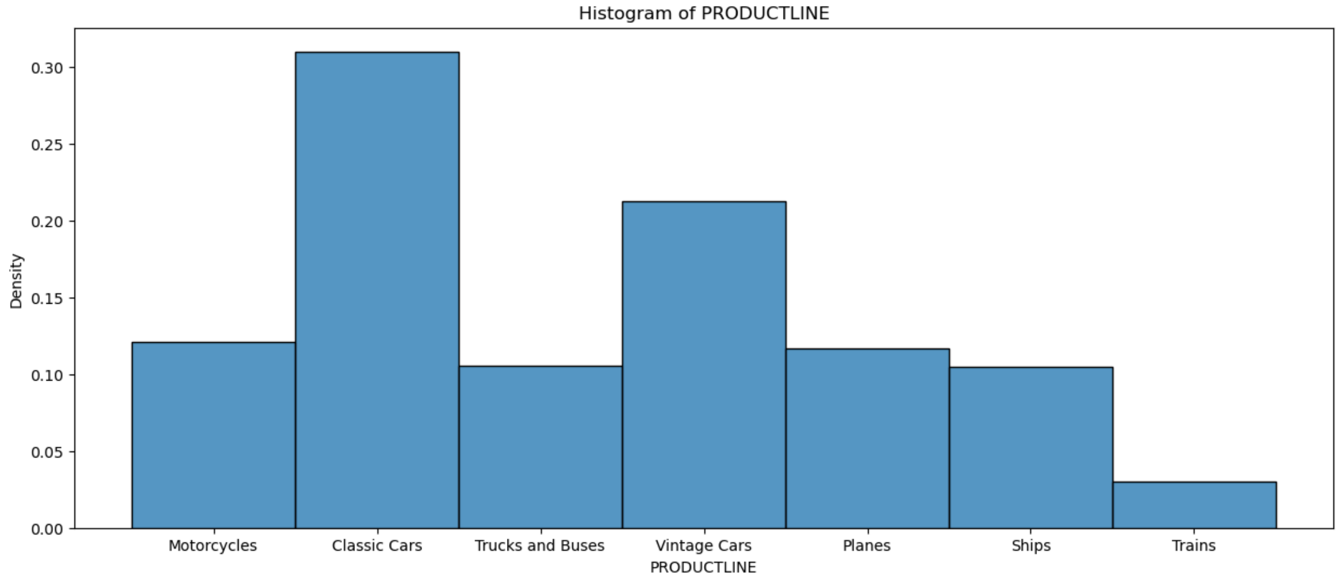
In [31]:
```
sales_df.describe()
```

Out[31]:

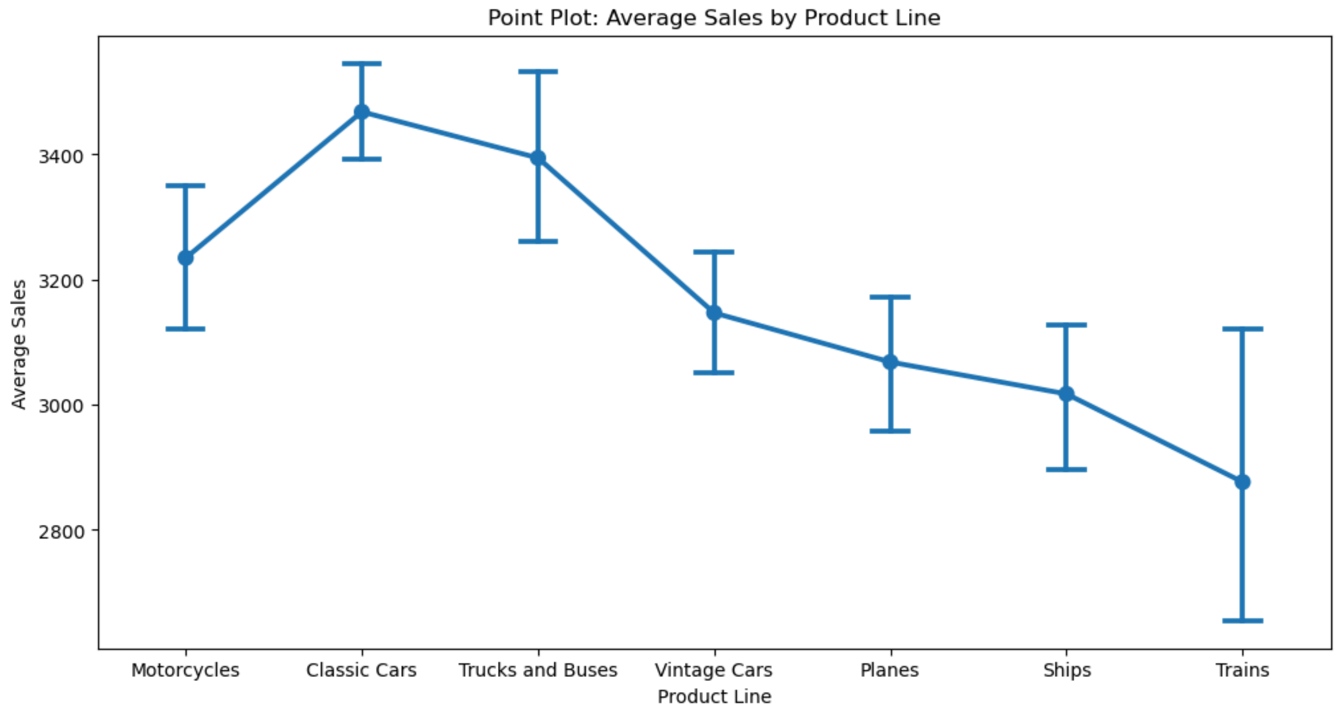
	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MO
count	2005.000000	2005.000000	2005.000000	2005.000000	2005.000000	2005.000000	2005
mean	10257.016958	34.563591	86.189845	6.642394	3252.263840	2.724190	7
std	91.365674	8.861406	16.590177	4.298528	973.191566	1.212558	3
min	10100.000000	12.000000	29.540000	1.000000	1713.690000	1.000000	1
25%	10178.000000	27.000000	73.980000	3.000000	2443.290000	2.000000	4
50%	10262.000000	34.000000	94.580000	6.000000	3157.440000	3.000000	8
75%	10331.000000	42.000000	100.000000	10.000000	3958.460000	4.000000	11
max	10425.000000	66.000000	100.000000	18.000000	5393.640000	4.000000	12

In [32]:
```
sales_df = sales_df.replace([np.inf, -np.inf], np.nan)

# Visualize a histogram of SALES
plt.figure(figsize=(15, 6))
sns.histplot(data=sales_df, x='PRODUCTLINE', element='bars', stat='density', common_norm=Fals
plt.title('Histogram of PRODUCTLINE')
plt.show()
```

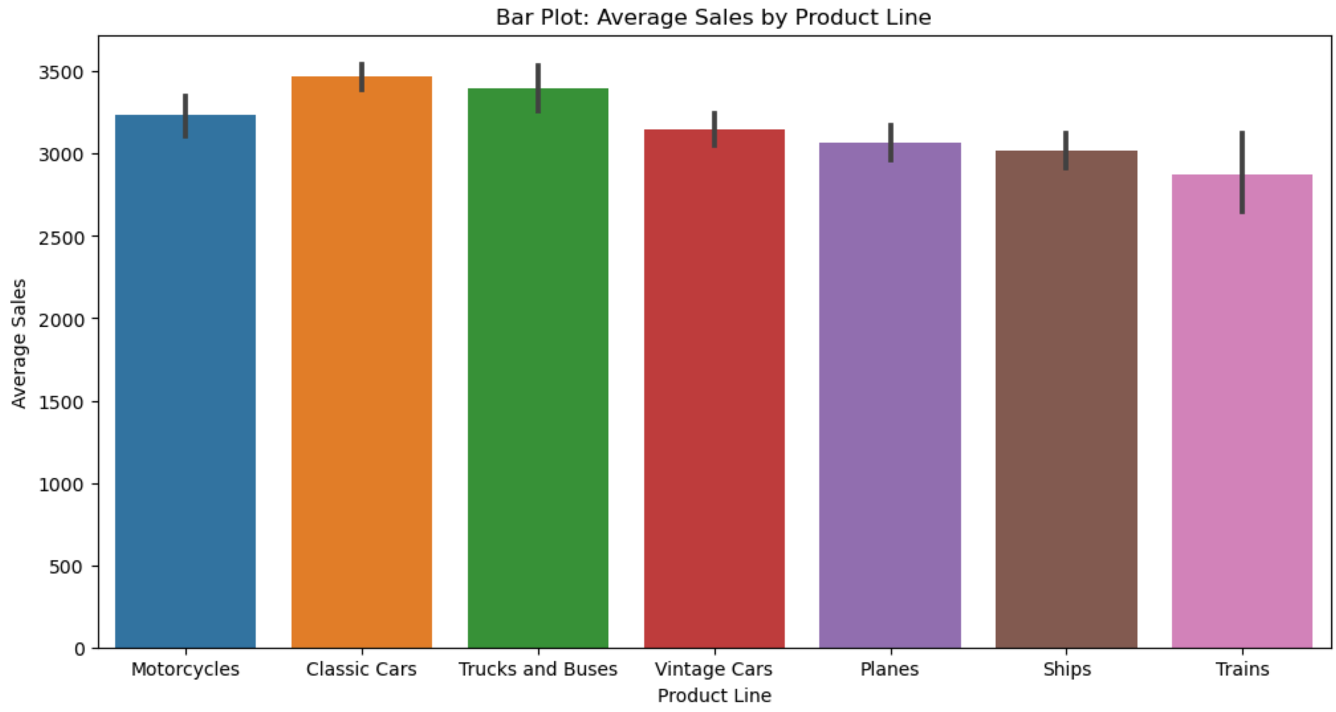
```
C:\Users\torri\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_a
s_na option is deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [33]:
```
#Point plot

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.pointplot(x='PRODUCTLINE', y='SALES', data=sales_df, capsize=0.2)
plt.title('Point Plot: Average Sales by Product Line')
plt.xlabel('Product Line')
plt.ylabel('Average Sales')
plt.show()
```



In [34]:
```
#Bar Plot

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.barplot(x='PRODUCTLINE', y='SALES', data=sales_df)
plt.title('Bar Plot: Average Sales by Product Line')
plt.xlabel('Product Line')
plt.ylabel('Average Sales')
plt.show()
```



In [35]:
```
#e.
# Analysis of Variance (ANOVA)
# Testing if there are any statistically significant differences among Product Lines.

from scipy.stats import f_oneway

grouped_data = [sales_df[sales_df['PRODUCTLINE'] == productline]['SALES'] for productline in

# Perform ANOVA
f_statistic, p_value = f_oneway(*grouped_data)

# Print the results
print(f"F-statistic: {f_statistic}")
print(f"P-value: {p_value}")

# Interpret the results
if p_value < 0.05:
    print("There are significant differences among PRODUCTLINE means.")
else:
    print("There are no significant differences among PRODUCTLINE means.")
```

```
F-statistic: 11.998883325816832
P-value: 2.8058279230390855e-13
There are significant differences among PRODUCTLINE means.
```

In [36]:
```
# Task C - Data Findings and Decision Support

#a. Analyze the results obtained from data analysis, including grouping, summarizing,
# investigating correlations, and applying inferential statistical methods

# Grouping and Summarizing
grouped_data = sales_df.groupby(['PRODUCTLINE']).agg({
    'SALES': ['sum', 'mean'],
    'QUANTITYORDERED': 'sum',
    'PRICEEACH': 'mean'
}).reset_index()

grouped_data
```

Out[36]:

	PRODUCTLINE	SALES		QUANTITYORDERED	PRICEEACH
		sum	mean	sum	mean
0	Classic Cars	2153742.49	3468.184364	20689	89.450258
1	Motorcycles	786032.93	3234.703416	8443	85.616749
2	Planes	717980.36	3068.292137	8352	83.272906
3	Ships	633635.73	3017.313000	7303	84.934095
4	Trains	172590.92	2876.515333	2144	77.230500
5	Trucks and Buses	716239.68	3394.500853	7251	89.427725
6	Vintage Cars	1340566.89	3146.870634	15118	83.643333

In [40]:

```
# Grouping and Summarizing
grouped_data = sales_df.groupby(['QTR_ID','YEAR_ID']).agg({
    'SALES': ['sum', 'mean'],
    'QUANTITYORDERED': 'sum',
    'ORDERNUMBER': 'count'
}).reset_index()

grouped_data = grouped_data.sort_values(by='YEAR_ID')
grouped_data
```

Out[40]:

	QTR_ID	YEAR_ID	SALES		QUANTITYORDERED	ORDERNUMBER
			sum	mean	sum	count
0	1	2003	322149.74	3254.037778	3439	99
3	2	2003	373837.87	3141.494706	4090	119
6	3	2003	425650.83	3299.618837	4288	129
8	4	2003	1208171.53	3301.015109	12516	366
1	1	2004	538661.11	3244.946446	5806	166
4	2	2004	477551.12	3100.981299	5265	154
7	3	2004	749028.62	3270.867336	7752	229
9	4	2004	1365788.46	3228.814326	14610	423
2	1	2005	725961.45	3314.892466	7686	219
5	2	2005	333988.27	3306.814554	3848	101

In [44]:

```
# Grouping and Summarizing
grouped_data = sales_df.groupby(['COUNTRY']).agg({
    'SALES': ['sum', 'mean'],
    'QUANTITYORDERED': 'sum',
    'ORDERNUMBER': 'count'
}).reset_index()

grouped_data = grouped_data.sort_values(by=('SALES','sum'), ascending=False)
grouped_data
```

Out[44]:

	COUNTRY	SALES		QUANTITYORDERED	ORDERNUMBER
		sum	mean	sum	count
18	USA	2363474.28	3278.050319	24999	721
14	Spain	793625.96	3293.053776	8666	241
6	France	710609.24	3200.942523	7755	222
0	Australia	435914.58	3327.592214	4446	131
17	UK	355834.91	3234.862818	3772	110
9	Italy	270273.12	3296.013659	2795	82
5	Finland	212502.37	3269.267231	2243	65
11	Norway	178885.56	3252.464727	1811	55
3	Canada	178231.78	3020.877627	1879	59
13	Singapore	172338.02	3251.660755	1847	53
1	Austria	130121.66	3336.452821	1392	39
7	Germany	128958.19	3145.321707	1389	41
15	Sweden	125193.06	3210.078462	1348	39
4	Denmark	117850.27	3021.801795	1253	39
10	Japan	96442.59	2922.502727	1194	33
16	Switzerland	86661.96	3767.911304	796	23
2	Belgium	63621.70	3029.604762	663	21
12	Philippines	62113.33	3105.666500	713	20
8	Ireland	38136.42	3466.947273	339	11

In [ ]: