In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/auto-sales-data/Auto Sales data.csv
```

This dataset provides sales data from an automobile company, including details on orders, products, and customers. It includes information on order quantities, pricing, sales totals, and order statuses, as well as customer demographics and product categories.

For exploratory data analysis (EDA), I will examine trends and patterns in sales, analyze customer purchasing behaviors, and assess the impact of various factors on sales performance. The EDA will involve exploring order frequencies, sales distributions, and customer activity to gain insights that could inform business strategies and improve decision-making and much morw so let's begin.

In [2]:
```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.10/site-packages (23.0.1)
Collecting pip
  Downloading pip-24.2-py3-none-any.whl (1.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 19.8 MB/s eta 0:00:0000:010:01
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.0.1
    Uninstalling pip-23.0.1:
      Successfully uninstalled pip-23.0.1
Successfully installed pip-24.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
pip install joypy
```

```
Collecting joypy
  Downloading joypy-0.2.6-py2.py3-none-any.whl.metadata (812 bytes)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/site-packages (from joypy) (1.26.4)
Requirement already satisfied: scipy>=0.11.0 in /usr/local/lib/python3.10/site-packages (from joypy) (1.14.0)
Requirement already satisfied: pandas>=0.20.0 in /usr/local/lib/python3.10/site-packages (from joypy) (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/site-packages (from joypy) (3.9.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/site-packages (from pandas>=0.20.0->joypy) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/site-packages (from pandas>=0.20.0->joypy) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/site-packages (from pandas>=0.20.0->joypy) (2024.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/site-packages (from matplotlib->joypy) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/site-packages (from matplotlib->joypy) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/site-packages (from matplotlib->joypy) (4.53.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/site-packages (from matplotlib->joypy) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/site-packages (from matplotlib->joypy) (24.1)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.10/site-packages (from matplotlib->joypy) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/site-packages (from matplotlib->joypy) (3.1.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/site-packages (from python-dateutil>=2.8.2->pandas>=0.20.0->joypy) (1.16.0)
Downloading joypy-0.2.6-py2.py3-none-any.whl (8.6 kB)
Installing collected packages: joypy
Successfully installed joypy-0.2.6
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable.It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
pip install plotly
```

```
Collecting plotly
  Downloading plotly-5.23.0-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/site-packages (from plotly) (8.5.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/site-packages (from plotly) (24.1)
Downloading plotly-5.23.0-py3-none-any.whl (17.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 91.1 MB/s eta 0:00:00:00:01
Installing collected packages: plotly
Successfully installed plotly-5.23.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable.It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
pip install squarify
```

```
Collecting squarify
  Downloading squarify-0.4.4-py3-none-any.whl.metadata (600 bytes)
Downloading squarify-0.4.4-py3-none-any.whl (4.1 kB)
Installing collected packages: squarify
Successfully installed squarify-0.4.4
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable.It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.
Note: you may need to restart the kernel to use updated packages.
```

In [6]:
```
pip install geopandas
```

```
Collecting geopandas
  Downloading geopandas-1.0.1-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.10/site-packages (from geopandas) (1.26.4)
Collecting pyogrio>=0.7.2 (from geopandas)
  Downloading pyogrio-0.9.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.8 kB)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/site-packages (from geopandas) (24.1)
Requirement already satisfied: pandas>=1.4.0 in /usr/local/lib/python3.10/site-packages (from geopandas) (2.2.2)
Collecting pyproj>=3.3.0 (from geopandas)
  Downloading pyproj-3.6.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (31 kB)
Collecting shapely>=2.0.0 (from geopandas)
  Downloading shapely-2.0.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (7.0 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/site-packages (from pandas>=1.4.0->geopandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/site-packages (from pandas>=1.4.0->geopandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/site-packages (from pandas>=1.4.0->geopandas) (2024.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/site-packages (from pyogrio>=0.7.2->geopandas) (2024.7.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/site-packages (from python-dateutil>=2.8.2->pandas>=1.4.0->geopandas) (1.16.0)
Downloading geopandas-1.0.1-py3-none-any.whl (323 kB)
Downloading pyogrio-0.9.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (23.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.1/23.1 MB 107.7 MB/s eta 0:00:0000:01
Downloading pyproj-3.6.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.3/8.3 MB 91.6 MB/s eta 0:00:00
Downloading shapely-2.0.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 60.3 MB/s eta 0:00:00
Installing collected packages: shapely, pyproj, pyogrio, geopandas
Successfully installed geopandas-1.0.1 pyogrio-0.9.0 pyproj-3.6.1 shapely-2.0.5
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable.It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.
Note: you may need to restart the kernel to use updated packages.
```

In [7]:
```
pip install wordcloud
```

```
Collecting wordcloud
  Downloading wordcloud-1.9.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.10/site-packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/site-packages (from wordcloud) (10.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/site-packages (from wordcloud) (3.9.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/site-packages (from matplotlib->wordcloud) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/site-packages (from matplotlib->wordcloud) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/site-packages (from matplotlib->wordcloud) (4.53.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/site-packages (from matplotlib->wordcloud) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/site-packages (from matplotlib->wordcloud) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/site-packages (from matplotlib->wordcloud) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/site-packages (from matplotlib->wordcloud) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
Downloading wordcloud-1.9.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (511 kB)
Installing collected packages: wordcloud
Successfully installed wordcloud-1.9.3
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable.It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.
Note: you may need to restart the kernel to use updated packages.
```

In [8]:
```
pip install statsmodels
```

```
Collecting statsmodels
  Downloading statsmodels-0.14.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.2 kB)
Requirement already satisfied: numpy>=1.22.3 in /usr/local/lib/python3.10/site-packages (from statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.10/site-packages (from statsmodels) (1.14.0)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.10/site-packages (from statsmodels) (2.2.2)
Collecting patsy>=0.5.6 (from statsmodels)
  Downloading patsy-0.5.6-py2.py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/site-packages (from statsmodels) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/site-packages (from patsy>=0.5.6->statsmodels) (1.16.0)
Downloading statsmodels-0.14.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (10.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.8/10.8 MB 78.0 MB/s eta 0:00:00
Downloading patsy-0.5.6-py2.py3-none-any.whl (233 kB)
Installing collected packages: patsy, statsmodels
Successfully installed patsy-0.5.6 statsmodels-0.14.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable.It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.
Note: you may need to restart the kernel to use updated packages.
```

Understanding the Dataset

In [9]:
```
import numpy as np                # For numerical operations
import pandas as pd               # For data manipulation and analysis
import matplotlib.pyplot as plt   # For creating static, interactive, and animated visualizations
import seaborn as sns             # For statistical data visualization
import plotly.express as px       # For interactive plots
import scipy.stats as stats
import squarify
import plotly.graph_objects as go
from joypy import joyplot
import networkx as nx
import geopandas as gpd
from wordcloud import WordCloud
import warnings                   # For handling warnings
warnings.filterwarnings('ignore') # To ignore warnings (if needed)

# Define the path to the dataset
file_path = '/kaggle/input/auto-sales-data/Auto Sales data.csv'

# Read the dataset
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
print(data.head())
```

```
   ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER    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        10168               36      96.66                1  3479.76

    ORDERDATE  DAYS_SINCE_LASTORDER   STATUS  PRODUCTLINE  MSRP PRODUCTCODE  \
0  24/02/2018                  828  Shipped  Motorcycles    95    S10_1678
1  07/05/2018                  757  Shipped  Motorcycles    95    S10_1678
2  01/07/2018                  703  Shipped  Motorcycles    95    S10_1678
3  25/08/2018                  649  Shipped  Motorcycles    95    S10_1678
4  28/10/2018                  586  Shipped  Motorcycles    95    S10_1678

           CUSTOMERNAME             PHONE                ADDRESSLINE1  \
0    Land of Toys Inc.        2125557818        897 Long Airport Avenue
1   Reims Collectables        26.47.1555              59 rue de l'Abbaye
2      Lyon Souveniers  +33 1 46 62 7555  27 rue du Colonel Pierre Avia
3    Toys4GrownUps.com        6265557265              78934 Hillside Dr.
4  Technics Stores Inc.        6505556809               9408 Furth Circle

         CITY POSTALCODE COUNTRY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
0         NYC      10022     USA             Yu             Kwai    Small
1       Reims      51100  France         Henriot             Paul    Small
2       Paris      75508  France        Da Cunha           Daniel   Medium
3    Pasadena      90003     USA           Young            Julie   Medium
4  Burlingame      94217     USA          Hirano             Juri   Medium
```

In [10]:
```
data.head() #View the First Few Rows
# Other options
# data.tail() : Returns the last n rows of the DataFrame.
# data.sample(n) : Returns a random sample of n rows from the DataFrame.
```

Out[10]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	DAYS_SINCE_LASTORDER	STATUS	PRODUCTLINE	MSRP	PRODUCTCODE	CUSTOMERNAME	PHONE	ADDRESSLINE1	CITY	POSTALCODE	COUNTRY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE
0	10107	30	95.70	2	2871.00	24/02/2018	828	Shipped	Motorcycles	95	S10_1678	Land of Toys Inc.	2125557818	897 Long Airport Avenue	NYC	10022	USA	Yu	Kwai	Small
1	10121	34	81.35	5	2765.90	07/05/2018	757	Shipped	Motorcycles	95	S10_1678	Reims Collectables	26.47.1555	59 rue de l'Abbaye	Reims	51100	France	Henriot	Paul	Small
2	10134	41	94.74	2	3884.34	01/07/2018	703	Shipped	Motorcycles	95	S10_1678	Lyon Souveniers	+33 1 46 62 7555	27 rue du Colonel Pierre Avia	Paris	75508	France	Da Cunha	Daniel	Medium
3	10145	45	83.26	6	3746.70	25/08/2018	649	Shipped	Motorcycles	95	S10_1678	Toys4GrownUps.com	6265557265	78934 Hillside Dr.	Pasadena	90003	USA	Young	Julie	Medium
4	10168	36	96.66	1	3479.76	28/10/2018	586	Shipped	Motorcycles	95	S10_1678	Technics Stores Inc.	6505556809	9408 Furth Circle	Burlingame	94217	USA	Hirano	Juri	Medium

In [11]:
```
data.shape
```

Out[11]:
```
(2747, 20)
```

In [12]:
```
data.columns # Returns the column labels of the DataFrame.
```

Out[12]:
```
Index(['ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER',
       'SALES', 'ORDERDATE', 'DAYS_SINCE_LASTORDER', 'STATUS', 'PRODUCTLINE',
       'MSRP', 'PRODUCTCODE', 'CUSTOMERNAME', 'PHONE', 'ADDRESSLINE1', 'CITY',
       'POSTALCODE', 'COUNTRY', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME',
       'DEALSIZE'],
      dtype='object')
```

In [13]:
```
data.index # Returns the index (row labels) of the DataFrame.
```

Out[13]:
```
RangeIndex(start=0, stop=2747, step=1)
```

In [14]:
```
data.info()
# Other options
# data.dtypes : Check Data Types
# data.isnull().sum() : Check for Missing Values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2747 entries, 0 to 2746
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ORDERNUMBER           2747 non-null   int64
 1   QUANTITYORDERED       2747 non-null   int64
 2   PRICEEACH             2747 non-null   float64
 3   ORDERLINENUMBER       2747 non-null   int64
 4   SALES                 2747 non-null   float64
 5   ORDERDATE             2747 non-null   object
 6   DAYS_SINCE_LASTORDER  2747 non-null   int64
 7   STATUS                2747 non-null   object
 8   PRODUCTLINE           2747 non-null   object
 9   MSRP                  2747 non-null   int64
 10  PRODUCTCODE           2747 non-null   object
 11  CUSTOMERNAME          2747 non-null   object
 12  PHONE                 2747 non-null   object
 13  ADDRESSLINE1          2747 non-null   object
 14  CITY                  2747 non-null   object
 15  POSTALCODE            2747 non-null   object
 16  COUNTRY               2747 non-null   object
 17  CONTACTLASTNAME       2747 non-null   object
 18  CONTACTFIRSTNAME      2747 non-null   object
 19  DEALSIZE              2747 non-null   object
dtypes: float64(2), int64(5), object(13)
memory usage: 429.3+ KB
```

Descriptive Statistics

In [15]:
```
# Identify numerical columns
numerical_columns = data.select_dtypes(include=['int64', 'float64'])

# Identify categorical columns
categorical_columns = data.select_dtypes(include=['object'])

print("Numerical Columns:")
print(numerical_columns)

print("\nCategorical Columns:")
print(categorical_columns)
```

```
Numerical Columns:
      ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER    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           10168               36      96.66               1  3479.76
...           ...              ...        ...             ...      ...
2742        10350               20     112.22              15  2244.40
2743        10373               29     137.19               1  3978.51
2744        10386               43     125.99               4  5417.57
2745        10397               34      62.24               1  2116.16
2746        10414               47      65.52               9  3079.44

      DAYS_SINCE_LASTORDER  MSRP
0                      828    95
1                      757    95
2                      703    95
3                      649    95
4                      586    95
...                    ...   ...
2742                  2924    54
2743                  2865    54
2744                  2836    54
2745                  2810    54
2746                  2772    54

[2747 rows x 7 columns]

Categorical Columns:
       ORDERDATE    STATUS  PRODUCTLINE PRODUCTCODE             CUSTOMERNAME  \
0     24/02/2018   Shipped  Motorcycles    S10_1678        Land of Toys Inc.
1     07/05/2018   Shipped  Motorcycles    S10_1678       Reims Collectables
2     01/07/2018   Shipped  Motorcycles    S10_1678          Lyon Souveniers
3     25/08/2018   Shipped  Motorcycles    S10_1678        Toys4GrownUps.com
4     28/10/2018   Shipped  Motorcycles    S10_1678     Technics Stores Inc.
...          ...       ...          ...         ...                      ...
2742  02/12/2019   Shipped        Ships    S72_3212    Euro Shopping Channel
2743  31/01/2020   Shipped        Ships    S72_3212  Oulu Toy Supplies, Inc.
2744  01/03/2020  Resolved        Ships    S72_3212    Euro Shopping Channel
2745  28/03/2020   Shipped        Ships    S72_3212             Alpha Cognac
2746  06/05/2020   On Hold        Ships    S72_3212       Gifts4AllAges.com

                 PHONE                  ADDRESSLINE1        CITY POSTALCODE  \
0          2125557818        897 Long Airport Avenue         NYC      10022
1          26.47.1555              59 rue de l'Abbaye       Reims      51100
2     +33 1 46 62 7555  27 rue du Colonel Pierre Avia       Paris      75508
3          6265557265               78934 Hillside Dr.    Pasadena      90003
4          6505556809                9408 Furth Circle  Burlingame      94217
...               ...                              ...         ...        ...
2742    (91) 555 94 44            C/ Moralzarzal, 86      Madrid      28034
2743       981-443655                   Torikatu 38        Oulu      90110
2744    (91) 555 94 44            C/ Moralzarzal, 86      Madrid      28034
2745       61.77.6555          1 rue Alsace-Lorraine    Toulouse      31000
2746       6175559555               8616 Spinnaker Dr.      Boston      51003

      COUNTRY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
0         USA             Yu             Kwai    Small
1      France         Henriot             Paul    Small
2      France        Da Cunha           Daniel   Medium
3         USA           Young            Julie   Medium
4         USA          Hirano             Juri   Medium
...       ...             ...              ...      ...
2742    Spain          Freyre            Diego    Small
2743  Finland       Koskitalo           Pirkko   Medium
2744    Spain          Freyre            Diego   Medium
2745   France          Roulet          Annette    Small
2746      USA         Yoshido             Juri   Medium

[2747 rows x 13 columns]
```

In [16]:
```
numerical_columns.describe().T # Summary Statistics for numerical columns
```

Out[16]:

	count	mean	std	min	25%	50%	75%	max
ORDERNUMBER	2747.0	10259.761558	91.877521	10100.00	10181.000	10264.00	10334.500	10425.00
QUANTITYORDERED	2747.0	35.103021	9.762135	6.00	27.000	35.00	43.000	97.00
PRICEEACH	2747.0	101.098952	42.042549	26.88	68.745	95.55	127.100	252.87
ORDERLINENUMBER	2747.0	6.491081	4.230544	1.00	3.000	6.00	9.000	18.00
SALES	2747.0	3553.047583	1838.953901	482.13	2204.350	3184.80	4503.095	14082.80
DAYS_SINCE_LASTORDER	2747.0	1757.085912	819.280576	42.00	1077.000	1761.00	2436.500	3562.00
MSRP	2747.0	100.691664	40.114802	33.00	68.000	99.00	124.000	214.00

In [17]:
```
categorical_columns.describe().T # Summary Statistics for categorical columns
# count: Number of non-null entries.
# unique: Number of unique categories.
# top: Most frequent category.
# freq: Frequency of the most frequent category.
```

Out[17]:

	count	unique	top	freq
ORDERDATE	2747	246	14/11/2018	38
STATUS	2747	6	Shipped	2541
PRODUCTLINE	2747	7	Classic Cars	949
PRODUCTCODE	2747	109	S18_3232	51
CUSTOMERNAME	2747	89	Euro Shopping Channel	259
PHONE	2747	88	(91) 555 94 44	259
ADDRESSLINE1	2747	89	C/ Moralzarzal, 86	259
CITY	2747	71	Madrid	304
POSTALCODE	2747	73	28034	259
COUNTRY	2747	19	USA	928
CONTACTLASTNAME	2747	76	Freyre	259
CONTACTFIRSTNAME	2747	72	Diego	259
DEALSIZE	2747	3	Medium	1349

In [18]:
```
data['STATUS'].value_counts() #Value Counts for Categorical Variables
```

Out[18]:
```
STATUS
Shipped       2541
Cancelled       60
Resolved        47
On Hold         44
In Process      41
Disputed        14
Name: count, dtype: int64
```

In [19]:
```
# Skewness and Kurtosis:

# Define thresholds for categorization
def categorize_skewness(value):
    if value > 1:
        category = 'High'
    elif value > 0.5:
        category = 'Moderate'
    else:
        category = 'Less'

    direction = 'Right Skewed' if value > 0 else 'Left Skewed'
    return f'{category} {direction}'

def categorize_kurtosis(value):
    if value > 3:
        return 'High'
    elif value > 1:
        return 'Moderate'
    else:
        return 'Less'

# Calculate skewness and kurtosis
skewness = numerical_columns.skew()  # Skewness
kurtosis = numerical_columns.kurt()  # Kurtosis

# Apply categorization functions
skewness_categories = skewness.apply(categorize_skewness)
kurtosis_categories = kurtosis.apply(categorize_kurtosis)

# Print results with categories
print("Skewness:")
print(pd.DataFrame({'Skewness': skewness, 'Category': skewness_categories}))

print("\nKurtosis:")
print(pd.DataFrame({'Kurtosis': kurtosis, 'Category': kurtosis_categories}))


#Other option
# skewness = numerical_columns.skew() #Skewness
# kurtosis = numerical_columns.kurt() #Kurtosis
# print("Skewness:")
# print(skewness)
# print("\nKurtosis:")
# print(kurtosis)
```

```
Skewness:
                    Skewness              Category
ORDERNUMBER         -0.006995      Less Left Skewed
QUANTITYORDERED      0.369286     Less Right Skewed
PRICEEACH            0.697222  Moderate Right Skewed
ORDERLINENUMBER      0.575327  Moderate Right Skewed
SALES                1.155940     High Right Skewed
DAYS_SINCE_LASTORDER -0.002983      Less Left Skewed
MSRP                 0.575646  Moderate Right Skewed

Kurtosis:
                    Kurtosis  Category
ORDERNUMBER         -1.154407     Less
QUANTITYORDERED      0.442865     Less
PRICEEACH            0.228519     Less
ORDERLINENUMBER     -0.591036     Less
SALES                1.773100  Moderate
DAYS_SINCE_LASTORDER -1.024466     Less
MSRP                 -0.139490     Less
```

In [20]:
```
# Correlation Matrix:
correlation_matrix = numerical_columns.corr()
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                    ORDERNUMBER  QUANTITYORDERED  PRICEEACH  \
ORDERNUMBER            1.000000         0.067110  -0.003369
QUANTITYORDERED        0.067110         1.000000   0.010161
PRICEEACH             -0.003369         0.010161   1.000000
ORDERLINENUMBER       -0.054300        -0.016295  -0.052646
SALES                  0.037289         0.553359   0.808287
DAYS_SINCE_LASTORDER  -0.251476        -0.021923  -0.397091
MSRP                  -0.013910         0.020551   0.778393

                    ORDERLINENUMBER     SALES  DAYS_SINCE_LASTORDER  \
ORDERNUMBER               -0.054300  0.037289             -0.251476
QUANTITYORDERED           -0.016295  0.553359             -0.021923
PRICEEACH                 -0.052646  0.808287             -0.397091
ORDERLINENUMBER            1.000000 -0.057414              0.046615
SALES                     -0.057414  1.000000             -0.334274
DAYS_SINCE_LASTORDER       0.046615 -0.334274              1.000000
MSRP                      -0.020956  0.634849             -0.524285

                        MSRP
ORDERNUMBER         -0.013910
QUANTITYORDERED      0.020551
PRICEEACH            0.778393
ORDERLINENUMBER     -0.020956
SALES                0.634849
DAYS_SINCE_LASTORDER -0.524285
MSRP                 1.000000
```

Data Cleaning

Treating missing values

In [21]:
```
# Filling missing values depends on the nature of the column and the context of the dataset.

# For Numerical Columns

# Mean Imputation
# data['QUANTITYORDERED'].fillna(data['QUANTITYORDERED'].mean(), inplace=True)

# Median Imputation
# data['QUANTITYORDERED'].fillna(data['QUANTITYORDERED'].median(), inplace=True)

# Predictive Imputation: Use a machine learning model to predict missing values based on other features.
# from sklearn.impute import KNNImputer
# imputer = KNNImputer(n_neighbors=5)
# data[['QUANTITYORDERED']] = imputer.fit_transform(data[['QUANTITYORDERED']])

# Interpolation
# data['QUANTITYORDERED'].interpolate(method='linear', inplace=True)

# Domain knowLedge
# Check with a domain expert to fill the missing values

# For Categorical Columns

# Mode Imputation
# data['PRODUCTLINE'].fillna(data['PRODUCTLINE'].mode()[0], inplace=True)

# Forward Fill / Backward Fill (Useful in time series data)
# data['PRODUCTLINE'].fillna(method='ffill', inplace=True)  # Forward fill
# data['PRODUCTLINE'].fillna(method='bfill', inplace=True)  # Backward fill

# Predictive Imputation : Use a machine learning model to predict the missing values based on other features.
# from sklearn.impute import SimpleImputer
# imputer = SimpleImputer(strategy='most_frequent')
# data[['PRODUCTLINE']] = imputer.fit_transform(data[['PRODUCTLINE']])

# Group-Based Imputation: If there are groups or clusters within the data, you can impute missing values within each group.
# data['PRODUCTLINE'] = data.groupby('ANOTHER_COLUMN')['PRODUCTLINE'].transform(lambda x: x.fillna(x.mode()[0]))
```

Treating duplicated values

In [22]:
```
# Removing duplicates

# Remove Duplicates Across All Columns
# data = data.drop_duplicates()

# Remove Duplicates Based on Specific Columns
# data = data.drop_duplicates(subset=['column1', 'column2'])


# Identifying duplicates

# Find Duplicates Across All Columns
# duplicates = data[data.duplicated()]

# Find Duplicates Based on Specific Columns
# duplicates = data[data.duplicated(subset=['column1', 'column2'])]


# Keep the specific occurrence of duplicates

# Keep First Occurrence (default)
# data = data.drop_duplicates(keep='first')

# Keep Last Occurrence
# data = data.drop_duplicates(keep='last')


# Handling Duplicates in a Grouped Context : Suppose you have a DataFrame with sales data for different salespeople, and you want to remove duplicates within each salesperson's records.

# data = data.groupby('group_column').apply(lambda x: x.drop_duplicates())


# Removing Duplicates with a Threshold : fuzzywuzzy helps find similar company names and remove duplicates based on a similarity threshold of i.e 90 it is used for categorical columns

# When you want to remove duplicates based on a similarity threshold rather than exact matches, you might use libraries like fuzzywuzzy for fuzzy string matching or recordlinkage for more advanced techniques.
# Using Numeric Thresholds: We might want to identify and handle numerical values that are close to each other but not exactly the same. For instance, when dealing with measurements or scores where a small difference is considered negligible.
```

Treating outliers

In [23]:
```
# Statistical Methods

# Z-Score Method
# from scipy import stats
# data['z_score'] = stats.zscore(data['QUANTITYORDERED'])
# data_cleaned = data[abs(data['z_score']) < 3]

# Modified Z-Score Method
# from statsmodels import robust
# median = data['QUANTITYORDERED'].median()
# mad = robust.mad(data['QUANTITYORDERED'])
# modified_z_score = 0.6745 * (data['QUANTITYORDERED'] - median) / mad
# data_cleaned = data[abs(modified_z_score) < 3.5]

# Interquartile Range (IQR) Method
# Q1 = data['QUANTITYORDERED'].quantile(0.25)
# Q3 = data['QUANTITYORDERED'].quantile(0.75)
# IQR = Q3 - Q1
# data_cleaned = data[(data['QUANTITYORDERED'] >= (Q1 - 1.5 * IQR)) & (data['QUANTITYORDERED'] <= (Q3 + 1.5 * IQR))]


# Model-Based Methods
# Isolation Forest
# from sklearn.ensemble import IsolationForest
# model = IsolationForest(contamination=0.01)
# outliers = model.fit_predict(data[['QUANTITYORDERED']])
# data_cleaned = data[outliers == 1]

# Local Outlier Factor (LOF)
# from sklearn.neighbors import LocalOutlierFactor
# model = LocalOutlierFactor(n_neighbors=20, contamination=0.01)
# outliers = model.fit_predict(data[['QUANTITYORDERED']])
# data_cleaned = data[outliers == 1]


# Visualization-Based Methods

# Box Plot
# import seaborn as sns
# import matplotlib.pyplot as plt
# sns.boxplot(data['QUANTITYORDERED'])
# plt.show()

# Scatter Plot
# plt.scatter(range(len(data['QUANTITYORDERED'])), data['QUANTITYORDERED'])
# plt.show()


# Transformations

# Log Transformation
# data['log_QUANTITYORDERED'] = np.log(data['QUANTITYORDERED'] + 1)  # Adding 1 to avoid log(0)

# Winsorization : Example : Replace values below the 5th percentile with the value at the 5th percentile and values above the 95th percentile with the value at the 95th percentile.
# from scipy.stats import mstats
# data['QUANTITYORDERED'] = mstats.winsorize(data['QUANTITYORDERED'], limits=[0.05, 0.05])
```

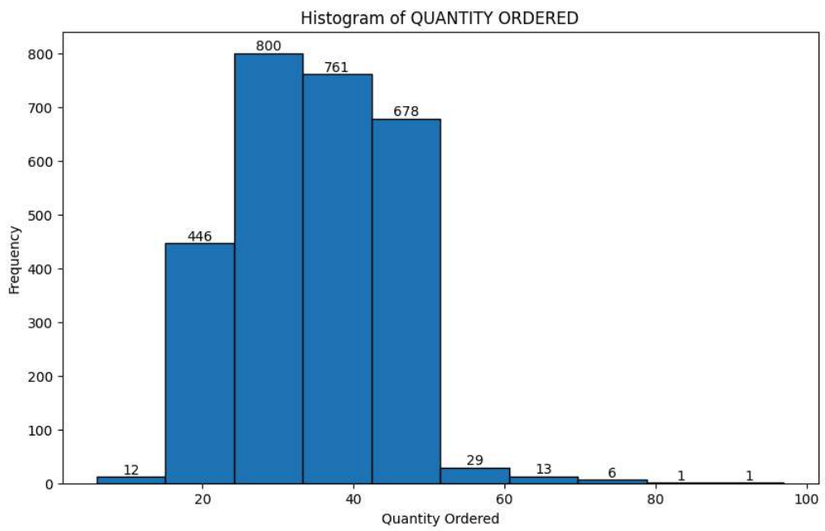
```
# Domain-Specific Techniques
# Apply specific business logic or domain expertise to identify and treat outliers based on practical constraints or known patterns.
```

Correct Data Types

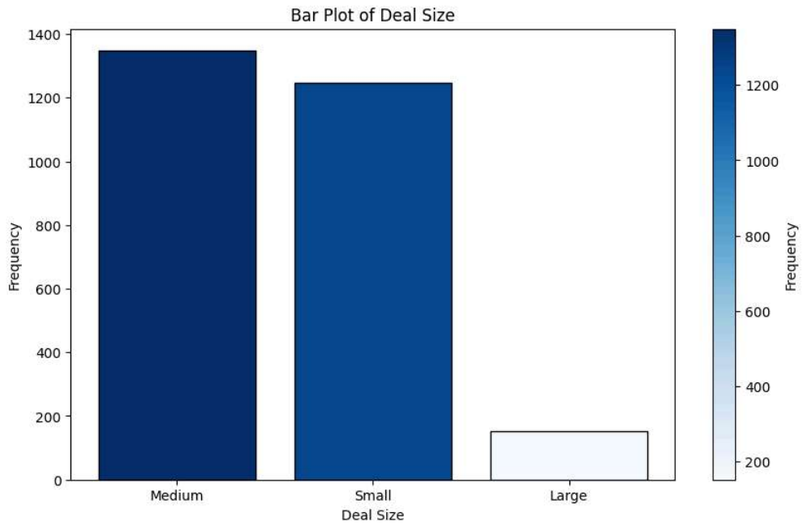
In [24]:
```
data['ORDERDATE'] = pd.to_datetime(data['ORDERDATE'])
```

Univariate Analysis

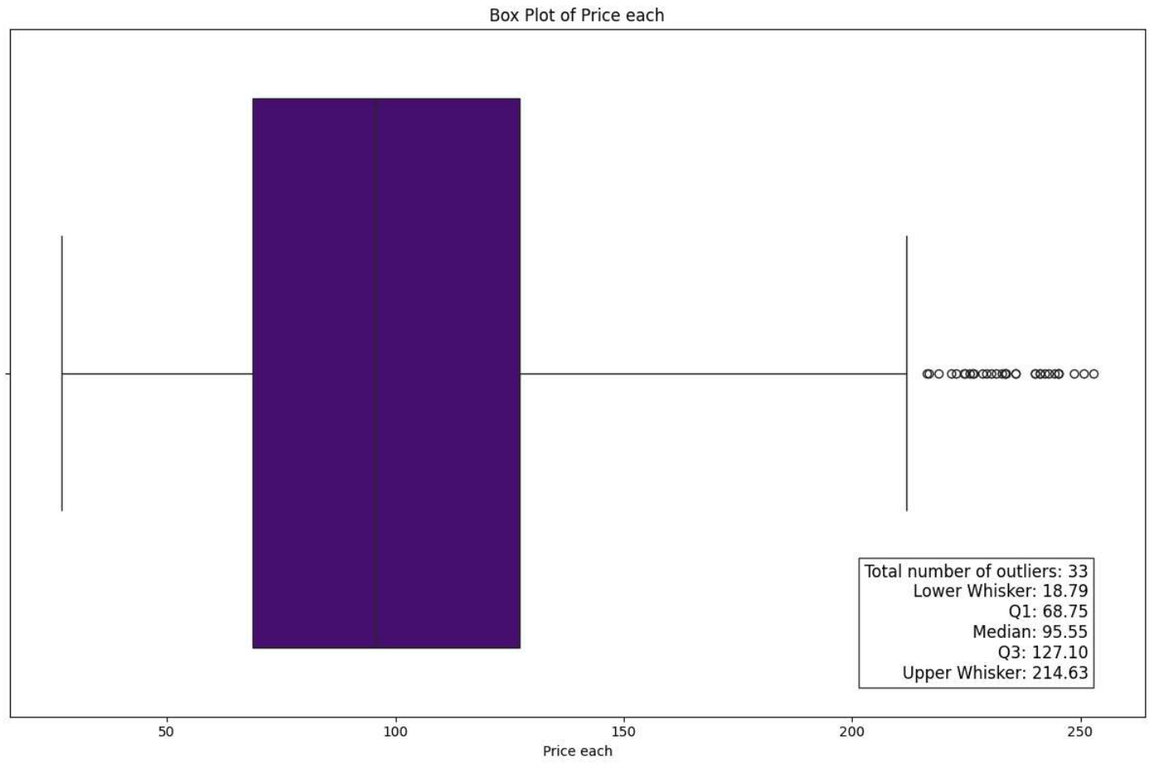
In [25]:
```
# Histogram

plt.figure(figsize=(10, 6))
counts, bins, patches = plt.hist(data['QUANTITYORDERED'], bins=10, edgecolor='black')

# Add the exact values on top of each bar
for count, bin_edge in zip(counts, bins):
    height = count
    plt.text(bin_edge + (bins[1] - bins[0]) / 2, height, int(count), ha='center', va='bottom')

plt.title('Histogram of QUANTITY ORDERED')
plt.xlabel('Quantity Ordered')
plt.ylabel('Frequency')
# plt.grid(True)
plt.show()
```



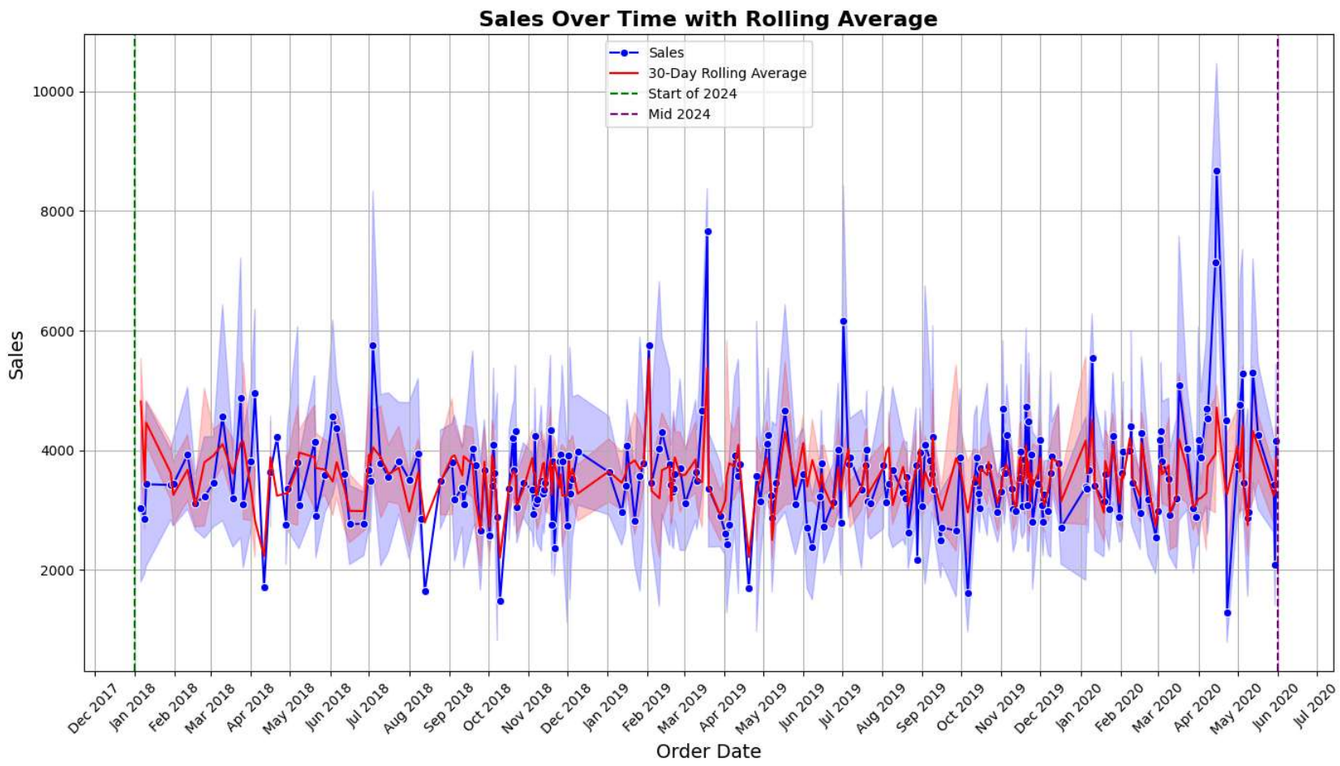
In [26]:
```
# Bar plot

# Count the occurrences of each deal size and sort in descending order
deal_size_counts = data['DEALSIZE'].value_counts()
deal_size_counts = deal_size_counts.sort_values(ascending=False)

# Create a color gradient based on the value counts
norm = plt.Normalize(deal_size_counts.min(), deal_size_counts.max())
sm = plt.cm.ScalarMappable(cmap="Blues", norm=norm)
sm.set_array([])

# Create a bar plot with color gradient
fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.bar(deal_size_counts.index, deal_size_counts.values, color=sm.to_rgba(deal_size_counts.values), edgecolor='black')

# Add color bar
cbar = fig.colorbar(sm, ax=ax, orientation='vertical')
cbar.set_label('Frequency')

ax.set_title('Bar Plot of Deal Size')
ax.set_xlabel('Deal Size')
ax.set_ylabel('Frequency')
# ax.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```



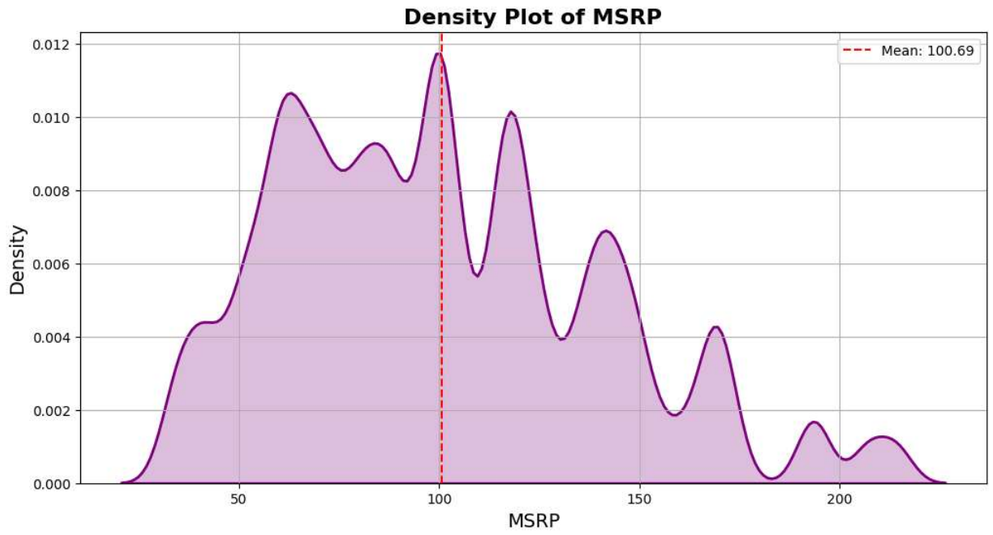
In [27]:
```
# Box Plot

plt.figure(figsize=(12, 8))
ax = sns.boxplot(x=data['PRICEEACH'], color='indigo')

# Calculate whiskers and quartiles
lower_whisker = - (data['PRICEEACH'].quantile(0.25) - (1.5 * (data['PRICEEACH'].quantile(0.75) - data['PRICEEACH'].quantile(0.25))))
upper_whisker = data['PRICEEACH'].quantile(0.75) + (1.5 * (data['PRICEEACH'].quantile(0.75) - data['PRICEEACH'].quantile(0.25)))
Q1 = data['PRICEEACH'].quantile(0.25)
median = data['PRICEEACH'].median()
Q3 = data['PRICEEACH'].quantile(0.75)

# Find outliers
outliers = data[(data['PRICEEACH'] < lower_whisker) | (data['PRICEEACH'] > upper_whisker)]
num_outliers = outliers.shape[0]

# Get x and y-axis limits
x_min, x_max = ax.get_xlim()
y_min, y_max = ax.get_ylim()

# Define position for the text
text_x_position = x_max - (x_max - x_min) * 0.05  # Adjust as needed for padding from right edge
text_y_position = y_min + (y_max - y_min) * 0.05  # Adjust as needed for padding from bottom edge

# Add text for the number of outliers and the summary statistics
plt.text(text_x_position, text_y_position,
         f'Total number of outliers: {num_outliers}\n'
         f'Lower Whisker: {lower_whisker:.2f}\n'
         f'Q1: {Q1:.2f}\n'
         f'Median: {median:.2f}\n'
         f'Q3: {Q3:.2f}\n'
         f'Upper Whisker: {upper_whisker:.2f}',
         horizontalalignment='right', verticalalignment='bottom',
         color='black', fontsize=12,
         bbox=dict(facecolor='white', alpha=0.8, edgecolor='black'))

# Adjust plot layout to make it tight
plt.title('Box Plot of Price each')
plt.xlabel('Price each')
plt.tight_layout()  # Adjust layout to fit elements tightly
plt.show()
```



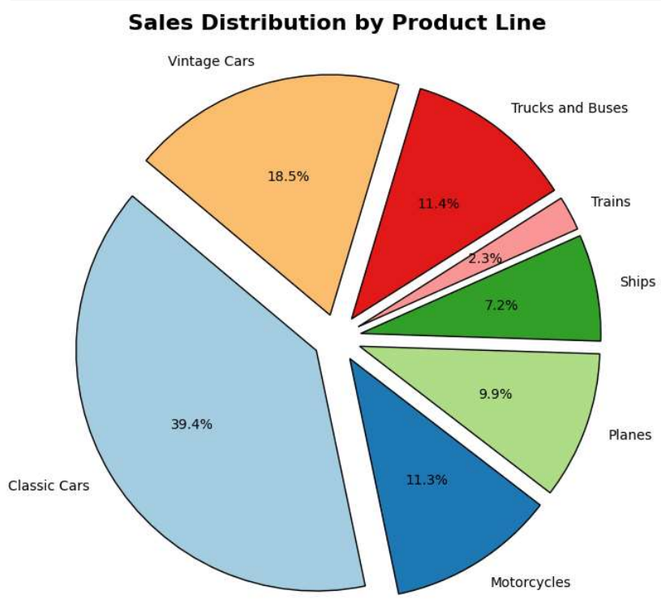
In [28]:
```
# Line plot

plt.figure(figsize=(14, 8))

# Plot the line for Sales
sns.lineplot(x='ORDERDATE', y='SALES', data=data, marker='o', color='b', label='Sales')

# Add a rolling average (e.g., 30-day rolling average)
data['ROLLING_AVG'] = data['SALES'].rolling(window=30).mean()
sns.lineplot(x='ORDERDATE', y='ROLLING_AVG', data=data, color='r', label='30-Day Rolling Average')

# Add titles and labels
plt.title('Sales Over Time with Rolling Average', fontsize=16, fontweight='bold')
plt.xlabel('Order Date', fontsize=14)
plt.ylabel('Sales', fontsize=14)

# Add grid
plt.grid(True)

# Rotate date labels for better readability
plt.xticks(rotation=45)

# Highlight significant dates or periods
plt.axvline(pd.Timestamp('2018-01-01'), color='g', linestyle='--', label='Start of 2024')
plt.axvline(pd.Timestamp('2020-06-01'), color='purple', linestyle='--', label='Mid 2024')

# Add a legend
plt.legend()

# Format x-axis dates
plt.gca().xaxis.set_major_formatter(plt.matplotlib.dates.DateFormatter('%b %Y'))
plt.gca().xaxis.set_major_locator(plt.matplotlib.dates.MonthLocator())

# Adjust layout to fit labels
plt.tight_layout()

plt.show()
```



In [29]:
```
# Density Plot (Kernel Density Estimate)

plt.figure(figsize=(12, 6))

# Create the KDE plot with additional details
sns.kdeplot(data['MSRP'],
            fill=True,
            color='purple',
            linewidth=2,
            bw_adjust=0.5)  # Adjust bandwidth for smoothness

# Add titles and labels
plt.title('Density Plot of MSRP', fontsize=16, fontweight='bold')
plt.xlabel('MSRP', fontsize=14)
plt.ylabel('Density', fontsize=14)

# Add vertical line for mean
mean_days = data['MSRP'].mean()
plt.axvline(mean_days, color='red', linestyle='--', label=f'Mean: {mean_days:.2f}')

# Add legend
plt.legend()

# Add grid for better readability
plt.grid(True)

plt.show()
```

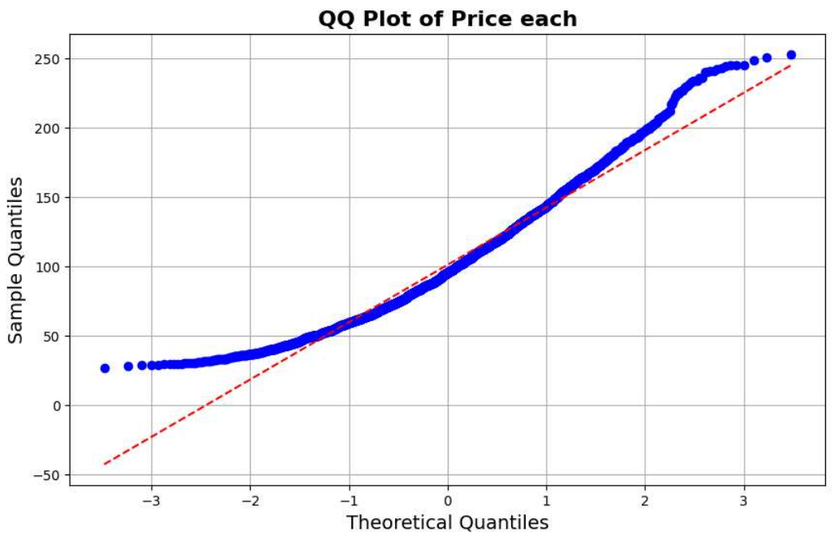


In [30]:
```
# Pie Chart

# Aggregate sales by product line
sales_by_product_line = data.groupby('PRODUCTLINE')['SALES'].sum()

# Create explode array to highlight the largest slice
explode = [0.1] * len(sales_by_product_line)  # Adjust as needed

# Create the pie chart
plt.figure(figsize=(10, 8))
plt.pie(sales_by_product_line,
        labels=sales_by_product_line.index,
        autopct='%1.1f%%',
        explode=explode,
        colors=plt.cm.Paired(range(len(sales_by_product_line))),
        startangle=140,
        wedgeprops={'edgecolor': 'black'})

# Add title
plt.title('Sales Distribution by Product Line', fontsize=16, fontweight='bold')

plt.show()
```

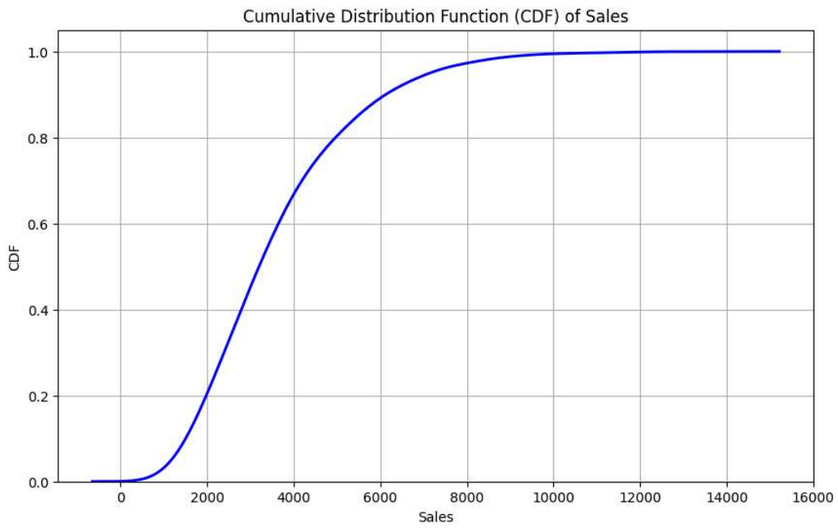


In [31]:
```
# QQ Plot

plt.figure(figsize=(10, 6))
```

```
# Create the QQ plot
res = stats.probplot(data['PRICEEACH'], dist="norm", plot=plt)

# Customize the reference line
line = plt.gca().get_lines()[1]  # Get the reference line
line.set_linestyle('--')
line.set_color('red')

# Add titles and labels
plt.title('QQ Plot of Price each', fontsize=16, fontweight='bold')
plt.xlabel('Theoretical Quantiles', fontsize=14)
plt.ylabel('Sample Quantiles', fontsize=14)

plt.grid(True)

plt.show()
```



In [32]:
```
# Cumulative Distribution Function (CDF) Plot

# Sample data assuming your DataFrame is named 'data'
sales = data['SALES']

# Plot
plt.figure(figsize=(10, 6))
sns.kdeplot(sales, cumulative=True, color='blue', linestyle='-', linewidth=2)
plt.xlabel('Sales')
plt.ylabel('CDF')
plt.title('Cumulative Distribution Function (CDF) of Sales')
plt.grid(True)
plt.show()
```



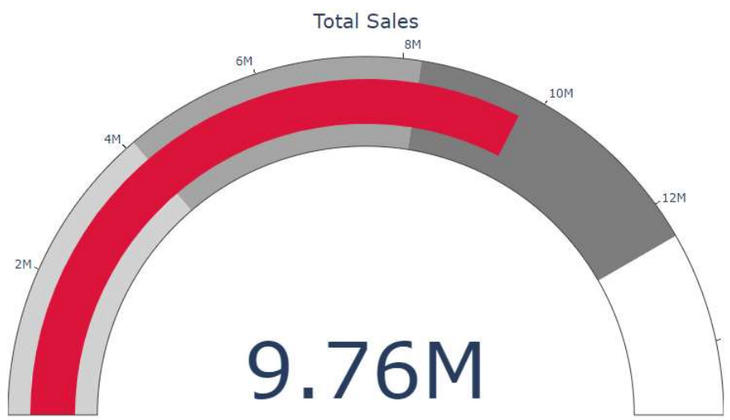
In [33]:
```
# Word cloud

# Concatenate all city names into a single string
```

```
text = ' '.join(data['CITY'].astype(str))

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

# Plot the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Hide the axis
plt.show()
```

In [ ]:
```
# Area chart

# Convert ORDERDATE to datetime
data['ORDERDATE'] = pd.to_datetime(data['ORDERDATE'])

# Aggregate sales by month
data['Month'] = data['ORDERDATE'].dt.to_period('M')
monthly_sales = data.groupby('Month')['SALES'].sum().reset_index()

# Create the area chart
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=monthly_sales['Month'].astype(str),
    y=monthly_sales['SALES'],
    fill='tozeroy',  # Fills the area under the line
    mode='lines',
    name='Total Sales'
))

# Update layout
fig.update_layout(
    title='Total Sales Over Time',
    xaxis_title='Month',
    yaxis_title='Total Sales',
    xaxis=dict(tickangle=-45)
)

fig.show()
```

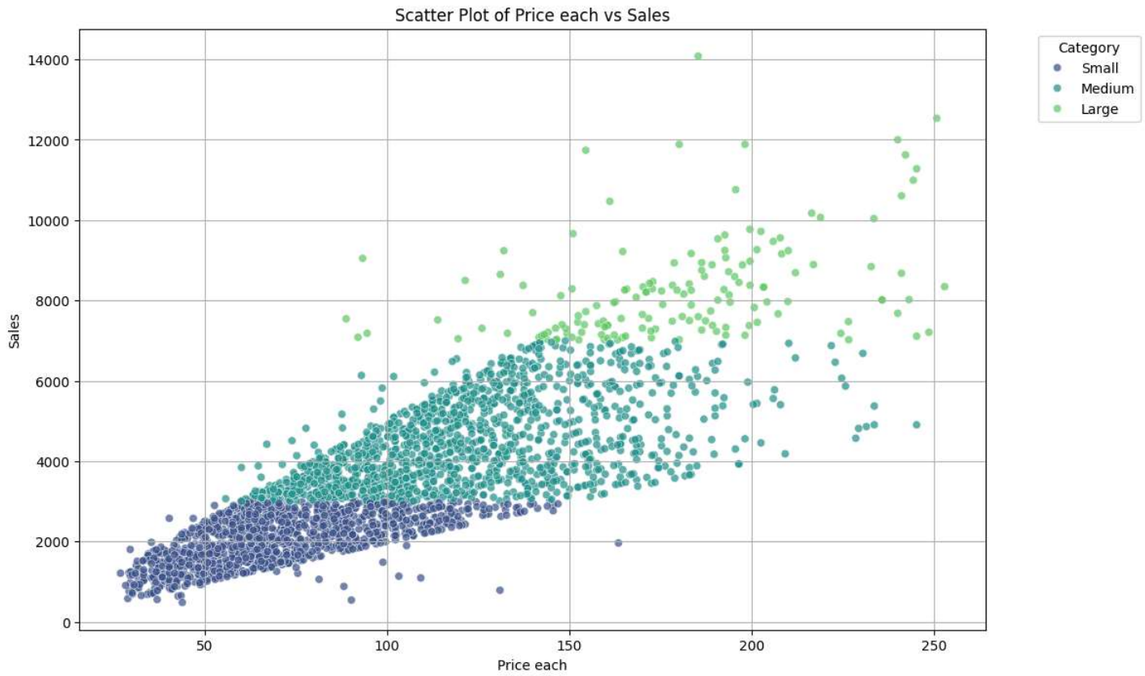
In [ ]:
```
# Gauge chart

# Calculate the total sales
```

```
total_sales = data['SALES'].sum()

# Define your target values
target_value = 12500000  # target value
max_value = 15000000    # maximum value for the gauge

# Create the gauge chart
fig = go.Figure(go.Indicator(
    mode="gauge+number",
    value=total_sales,
    title={'text': "Total Sales"},
    gauge={'axis': {'range': [None, max_value]},
           'bar': {'color': "crimson"},
           'steps': [
               {'range': [0, target_value * 0.33], 'color': "lightgray"},
               {'range': [target_value * 0.33, target_value * 0.66], 'color': "darkgray"},
               {'range': [target_value * 0.66, target_value], 'color': "gray"},
           ]},
))

# Update layout for better presentation
fig.update_layout(
    margin={'t': 0, 'b': 0, 'l': 0, 'r': 0}
)

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



Bivariate Analysis

In [36]:
```
# Scatter plot

plt.figure(figsize=(12, 8))

# Assuming you have an additional categorical variable 'CATEGORY' and a size variable 'SIZE'
sns.scatterplot(x='PRICEEACH', y='SALES', data=data, hue='DEALSIZE', palette='viridis', sizes=(20, 200), alpha=0.7)

# Title and Labels
plt.title('Scatter Plot of Price each vs Sales')
plt.xlabel('Price each')
plt.ylabel('Sales')

# Add legend
plt.legend(title='Category', bbox_to_anchor=(1.05, 1), loc='upper left')

# Add grid
plt.grid(True)

plt.show()
```

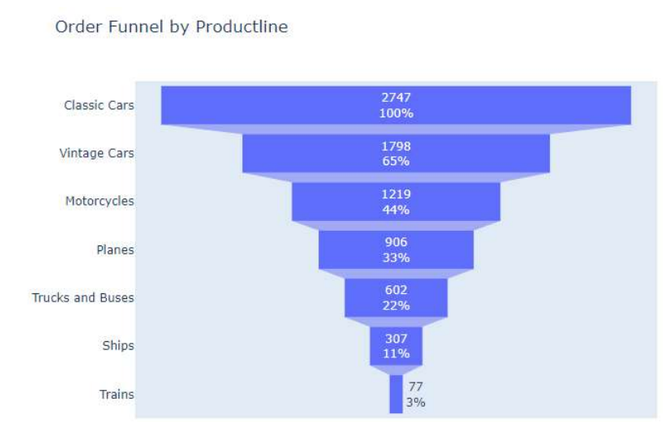


In [ ]:
```
# Radial plot

# Sample data aggregation
data_grouped = data.groupby('PRODUCTLINE').agg({'SALES': 'sum'}).reset_index()

# Create a radial plot
fig = go.Figure()

# Add a trace for each product line
for product_line in data_grouped['PRODUCTLINE']:
    sales = data_grouped[data_grouped['PRODUCTLINE'] == product_line]['SALES'].values[0]
    fig.add_trace(go.Scatterpolar(
        r=[sales],
        theta=[product_line],
        mode='markers',
        marker=dict(size=14),
        name=product_line
    ))

# Update layout for better readability
fig.update_layout(
    polar=dict(
        radialaxis=dict(visible=True, range=[0, data_grouped['SALES'].max()]),
    ),
    showlegend=True
)

fig.show()
```

In [ ]:
```
# Map

# Aggregate sales by country
```

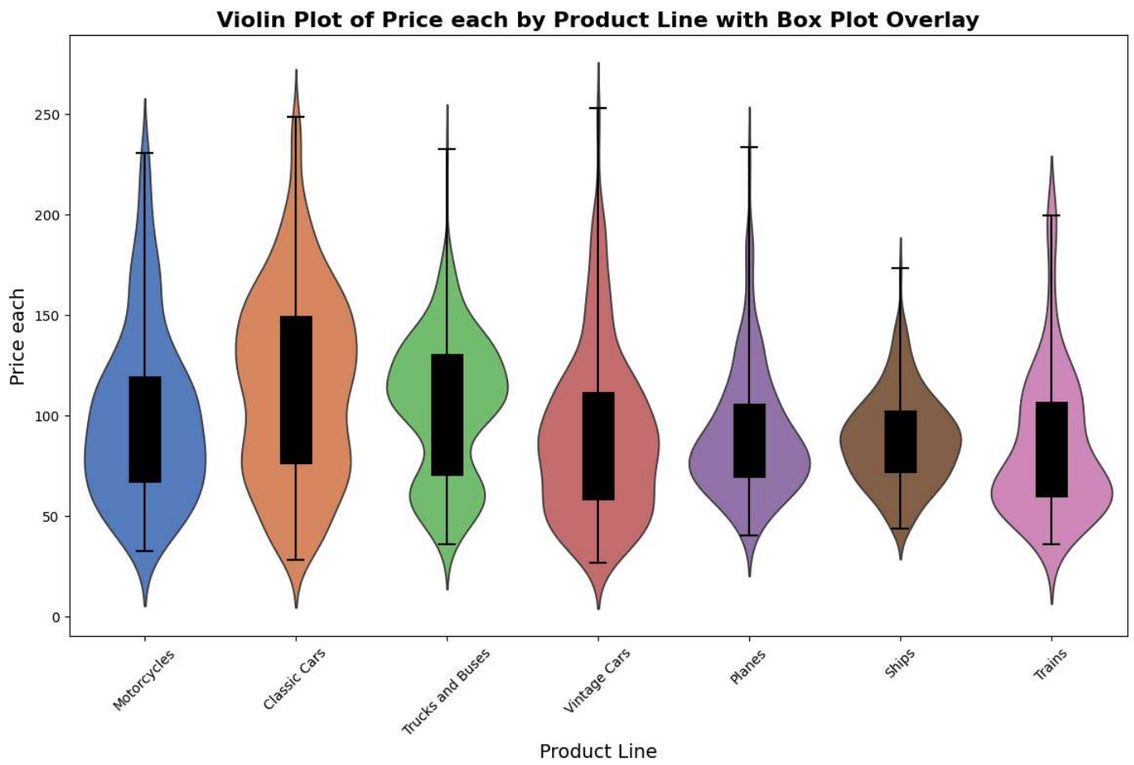
```
sales_by_country = data.groupby('COUNTRY')['SALES'].sum().reset_index()

# Plot interactive map
fig = px.choropleth(sales_by_country,
                    locations='COUNTRY',
                    locationmode='country names',
                    color='SALES',
                    color_continuous_scale='OrRd',
                    title='Sales by Country')

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

In [ ]:
```
# Funnel chart

data_funnel = pd.DataFrame({
    'PRODUCTLINE': ['Classic Cars', 'Vintage Cars', 'Motorcycles', 'Planes', 'Trucks and Buses', 'Ships', 'Trains'],
    'ORDERNUMBER': [2747, 1798, 1219, 906, 602, 307, 77]  # numbers for each stage
})

# Aggregate data by STATUS
funnel_data = data_funnel.groupby('PRODUCTLINE')['ORDERNUMBER'].sum().reset_index()

# Sort by stages if necessary (e.g., from highest to lowest)
funnel_data = funnel_data.sort_values('ORDERNUMBER', ascending=False)

# Create the funnel chart
fig = go.Figure(go.Funnel(
    y=funnel_data['PRODUCTLINE'],
    x=funnel_data['ORDERNUMBER'],
    textinfo="value+percent initial"
))

# Update layout for better presentation
fig.update_layout(
    title="Order Funnel by Productline",
    funnelmode="stack",  # Stack the funnel stages
)

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



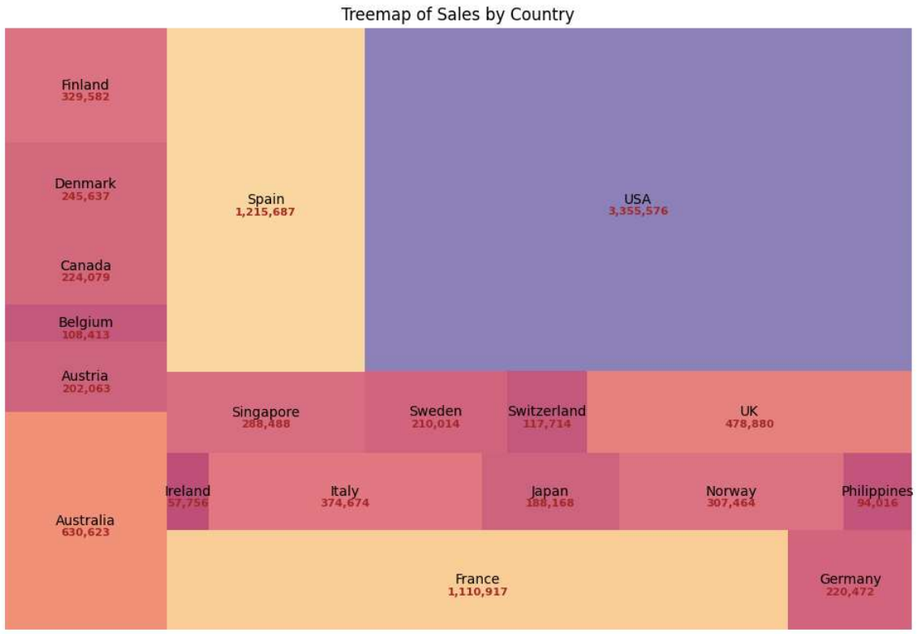
In [40]:
```
# Violet plot
```

```
plt.figure(figsize=(14, 8))

# Create the violin plot with additional details
sns.violinplot(x='PRODUCTLINE', y='PRICEEACH', data=data, palette='muted', inner=None, linewidth=1.25)

# Overlay with a box plot
sns.boxplot(x='PRODUCTLINE', y='PRICEEACH', data=data, whis=np.inf, width=0.2, color='k', fliersize=0, linewidth=1.5)

# Add titles and labels
plt.title('Violin Plot of Price each by Product Line with Box Plot Overlay', fontsize=16, fontweight='bold')
plt.xlabel('Product Line', fontsize=14)
plt.ylabel('Price each', fontsize=14)

plt.xticks(rotation=45)  # Rotate x-axis labels if needed
# plt.grid(True)

plt.show()
```

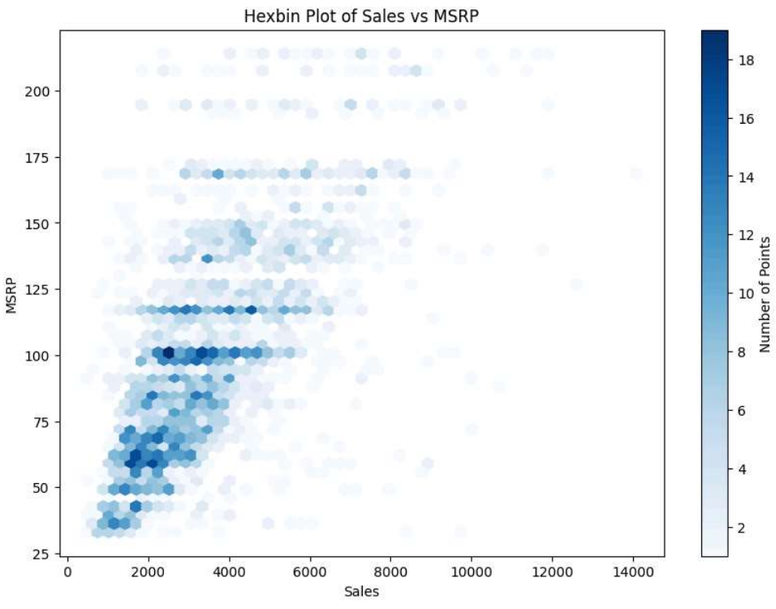


In [41]:

```
# Treemap

# Aggregating sales by country
treemap_data = data.groupby('COUNTRY').agg({'SALES': 'sum'}).reset_index()
sizes = treemap_data['SALES']
labels = treemap_data['COUNTRY']

# Define a colormap
cmap = plt.get_cmap('Spectral')  # You can choose any colormap from matplotlib
colors = cmap(sizes / max(sizes))  # Normalize sizes for color mapping

# Plot
fig, ax = plt.subplots(figsize=(12, 8))
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=0.7, ax=ax)

# Add annotations for total sales just below each country name
for i, (label, size) in enumerate(zip(labels, sizes)):
    # Calculate the position for text annotation
    rect = ax.patches[i]
    x = rect.get_x() + rect.get_width() / 2
    y = rect.get_y() + rect.get_height() / 2
    # Add total sales value just below the country name
    ax.text(x, y - 1.2, f'{size:,.0f}', ha='center', va='top', fontsize=8, color='brown', weight='bold')

plt.title('Treemap of Sales by Country')
plt.axis('off')  # Turn off the axis
plt.show()
```

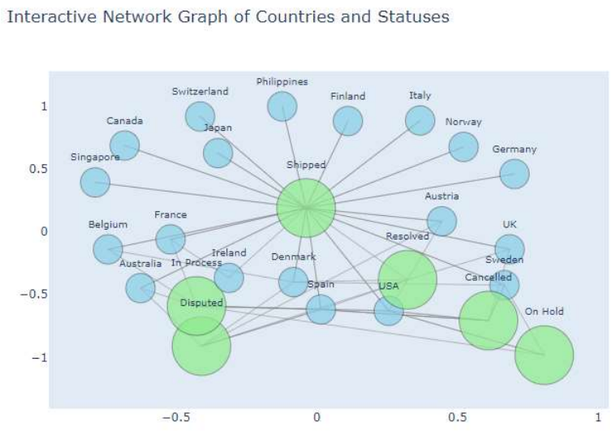


In [42]:
```
# Hexbin plot

# Plotting hexbin plot for SALES vs MSRP
plt.figure(figsize=(10, 7))
plt.hexbin(data['SALES'], data['MSRP'], gridsize=50, cmap='Blues', mincnt=1)

# Add color bar
plt.colorbar(label='Number of Points')

# Set labels and title
plt.xlabel('Sales')
plt.ylabel('MSRP')
plt.title('Hexbin Plot of Sales vs MSRP')

plt.show()
```



In [ ]:
```
# Network graph

# Create a graph object
G = nx.Graph()

# Add edges based on country-status relationships
for _, row in data.iterrows():
```

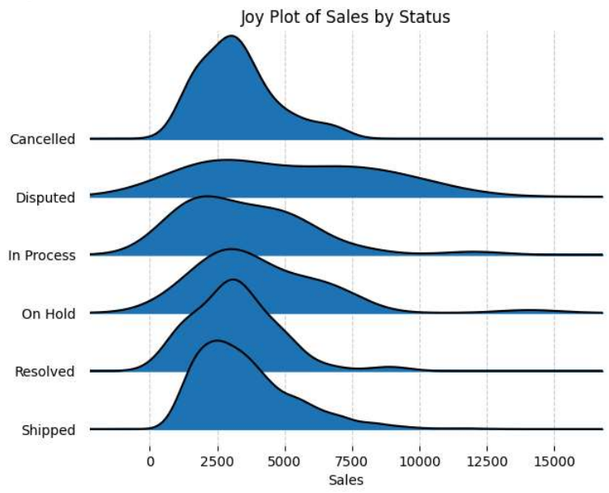
```
    country = row['COUNTRY']
    status = row['STATUS']

    # Add nodes and edges to the graph
    G.add_node(country, bipartite=0)  # Country nodes
    G.add_node(status, bipartite=1)   # Status nodes
    G.add_edge(country, status)       # Edge between country and status

# Compute the position of nodes using the spring layout
pos = nx.spring_layout(G)

# Extract node and edge information
edge_x = []
edge_y = []
for edge in G.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    edge_x.append(x0)
    edge_x.append(x1)
    edge_y.append(y0)
    edge_y.append(y1)

node_x = [pos[node][0] for node in G.nodes()]
node_y = [pos[node][1] for node in G.nodes()]

# Define node colors and sizes
node_color = ['skyblue' if G.nodes[node]['bipartite'] == 0 else 'lightgreen' for node in G.nodes()]
node_size = [30 if G.nodes[node]['bipartite'] == 0 else 60 for node in G.nodes()]  # Adjust sizes as needed

# Create the Plotly figure
fig = go.Figure()

# Add edges to the plot
fig.add_trace(go.Scatter(
    x=edge_x,
    y=edge_y,
    mode='lines',
    line=dict(width=0.5, color='gray'),
    hoverinfo='none'
))

# Add nodes to the plot
fig.add_trace(go.Scatter(
    x=node_x,
    y=node_y,
    mode='markers+text',
    text=[node for node in G.nodes()],  # Show text for all nodes
    textposition='top center',
    marker=dict(
        size=[size for size in node_size],
        color=node_color,
        line=dict(width=0.5, color='black')
    ),
    hoverinfo='text',
    textfont=dict(size=10)  # Adjust text size as needed
))

# Update layout
fig.update_layout(
    showlegend=False,
    title='Interactive Network Graph of Countries and Statuses',
    xaxis=dict(showgrid=False, zeroline=False),
    yaxis=dict(showgrid=False, zeroline=False),
    hovermode='closest'
)

fig.show()
```



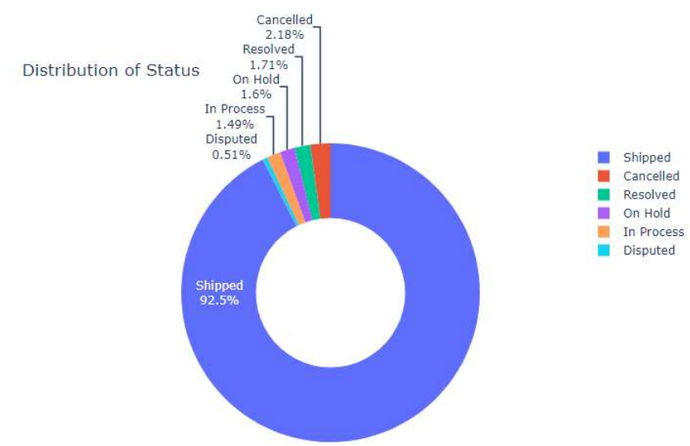
In [44]:
```
# Joy plot

# Prepare data for the Joy Plot
```

```
grouped_data = data.groupby('STATUS')['SALES'].apply(list).to_dict()

# Create the Joy Plot
plt.figure(figsize=(12, 8))
joyplot(data=grouped_data, labels=grouped_data.keys())
plt.title('Joy Plot of Sales by Status')
plt.xlabel('Sales')
plt.ylabel('Density')

# Customize plot appearance
plt.grid(True, linestyle='--', alpha=0.6)  # Add a grid for better readability

plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```



In [ ]:
```
# Donut chart

# Calculate the counts for each status
status_counts = data['STATUS'].value_counts()

# Create a 3D-like donut chart
fig = go.Figure(data=[go.Pie(
    labels=status_counts.index,
    values=status_counts,
    hole=0.5,  # This creates the donut effect
    textinfo='label+percent',
    insidetextorientation='radial'
)])

fig.update_layout(
    title_text='Distribution of Status',
    showlegend=True
)

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

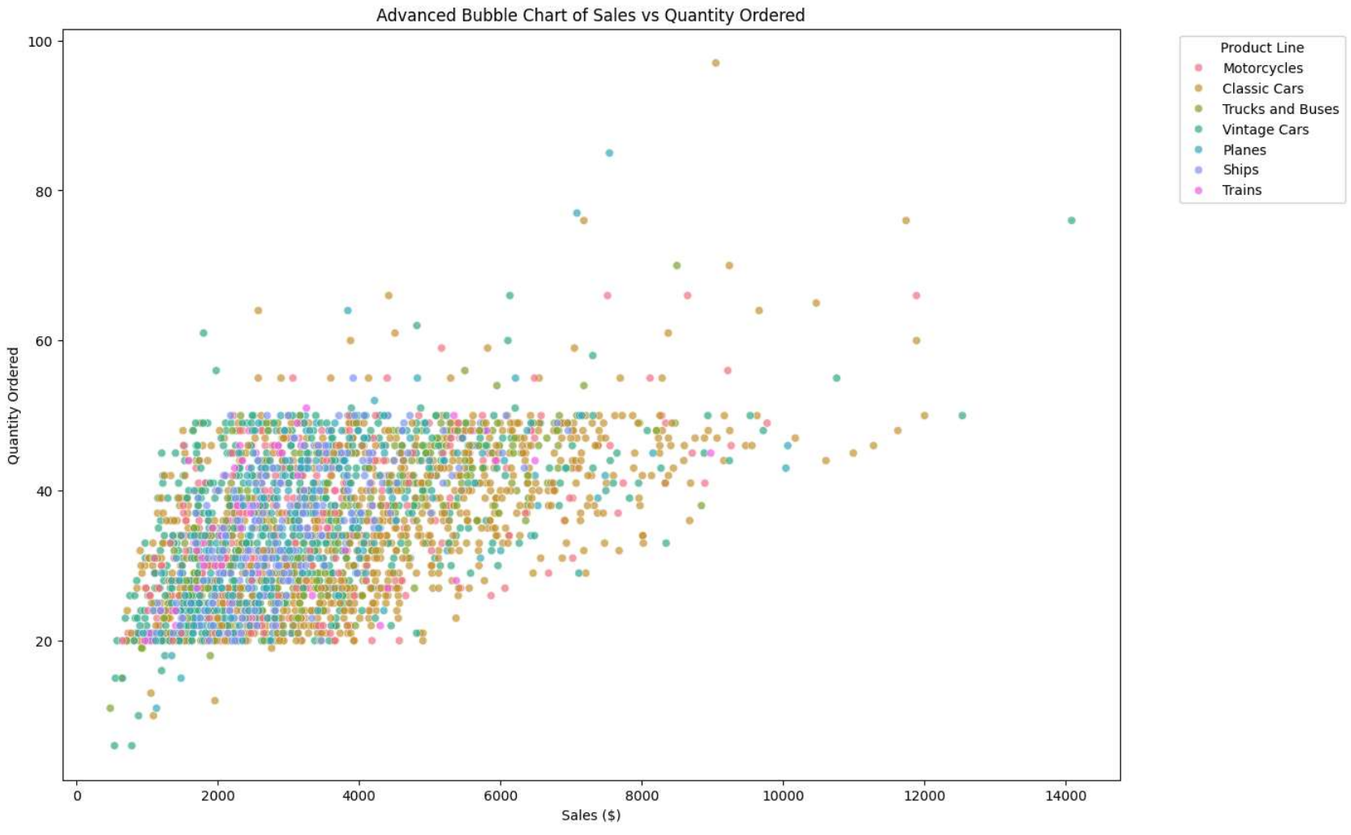


Multivariate Analysis

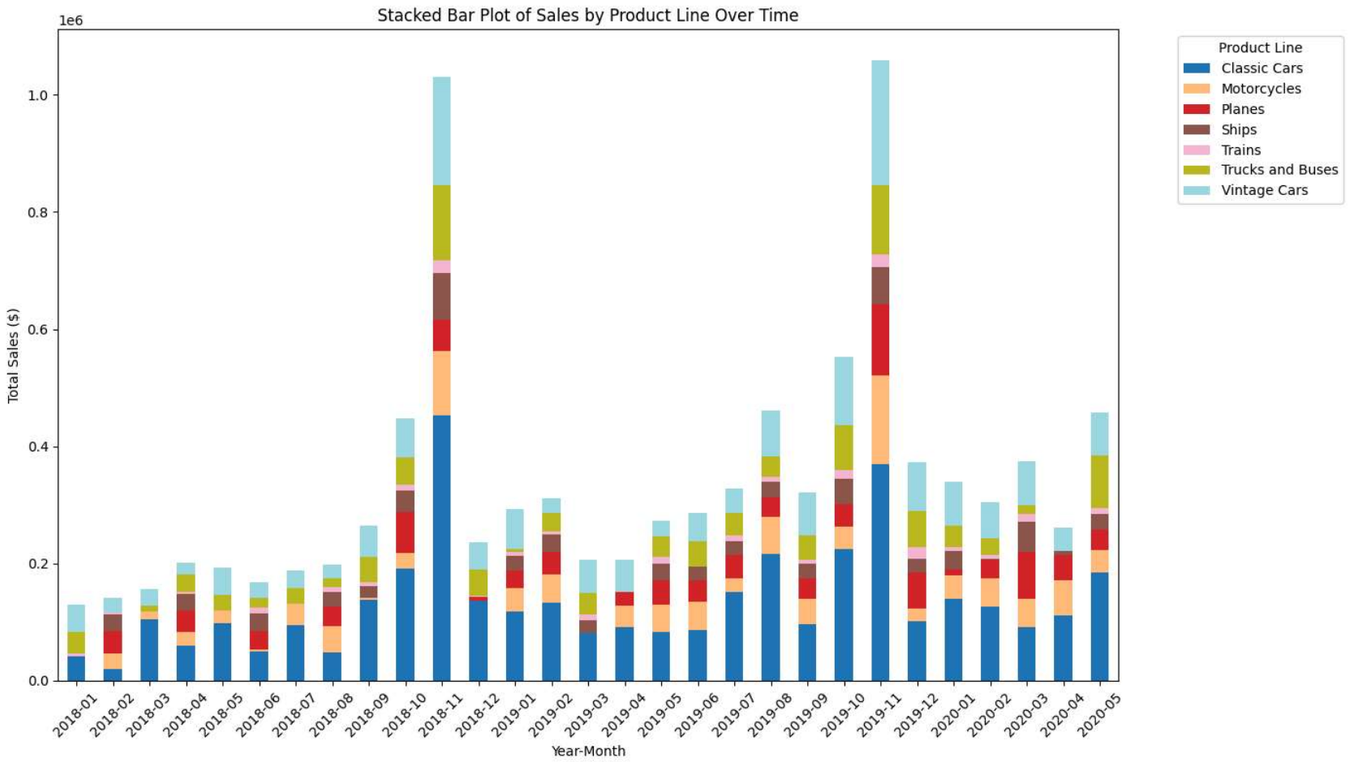
In [46]:
```
# Cross tab

cross_tab = pd.crosstab(index=[data['STATUS'], data['PRODUCTLINE']], columns=data['DEALSIZE'])

# Display the cross-tabulation
print(cross_tab)
```

```
DEALSIZE                    Large  Medium  Small
STATUS     PRODUCTLINE
Cancelled  Classic Cars          0      10      6
           Planes                0       4      8
           Ships                 0      10      8
           Trains                0       1      0
           Vintage Cars          0       8      5
Disputed   Classic Cars          3       0      0
           Motorcycles           2       3      1
           Planes                0       0      2
           Ships                 0       1      0
           Vintage Cars          0       1      1
In Process Classic Cars          1       8      5
           Trucks and Buses      2       5      4
           Vintage Cars          0       5     11
On Hold    Classic Cars          2       5      5
           Motorcycles           0       1      0
           Planes                2       4      3
           Ships                 0       5      3
           Trains                0       1      0
           Trucks and Buses      0       3      1
           Vintage Cars          1       5      3
Resolved   Classic Cars          0       5      3
           Planes                0       7      5
           Ships                 0       8      4
           Trucks and Buses      0       3      2
           Vintage Cars          1       3      6
Shipped    Classic Cars         89     490    317
           Motorcycles          16     144    146
           Planes                7     113    149
           Ships                 0      82    109
           Trains                1      24     50
           Trucks and Buses      5     162    108
           Vintage Cars         20     228    281
```

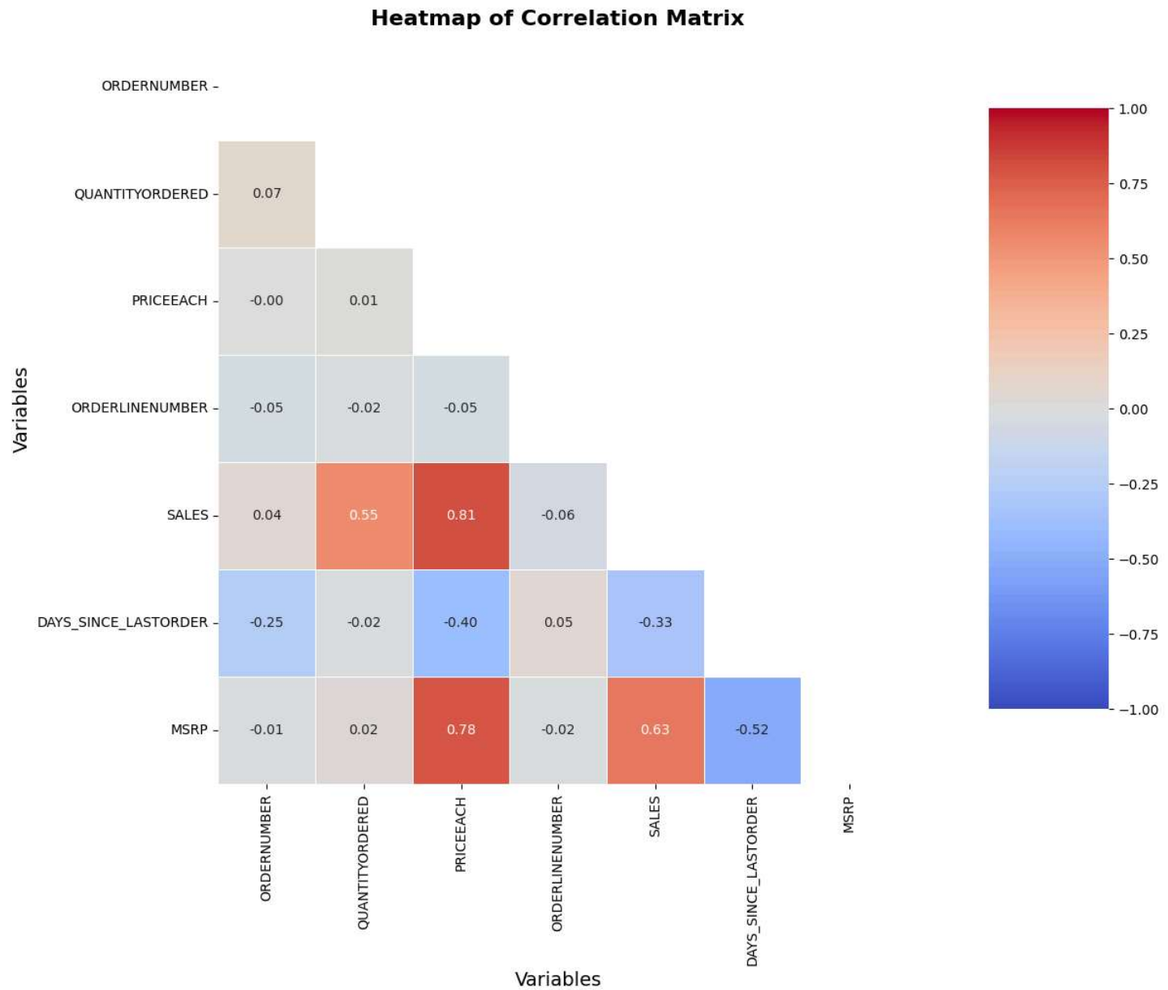
In [47]:
```
# Bubble chart

# Custom color palette
palette = sns.color_palette("husl", n_colors=data['PRODUCTLINE'].nunique())

# Create a bubble chart
plt.figure(figsize=(14, 10))
scatter = sns.scatterplot(
    data=data,
    x='SALES',
    y='QUANTITYORDERED',
    # size='DEALSIZE',
    hue='PRODUCTLINE',
    palette=palette,
    sizes=(50, 500),
    alpha=0.7,  # Adjust transparency
    edgecolor='w',  # Bubble edge color
    linewidth=0.5  # Edge width
)

# Add labels and title
plt.title('Advanced Bubble Chart of Sales vs Quantity Ordered')
plt.xlabel('Sales ($)')
plt.ylabel('Quantity Ordered')

# Add a legend with title and make it more readable
plt.legend(title='Product Line', bbox_to_anchor=(1.05, 1), loc='upper left')

# Grid and layout adjustments
# plt.grid(True)
# plt.tight_layout()
plt.show()
```



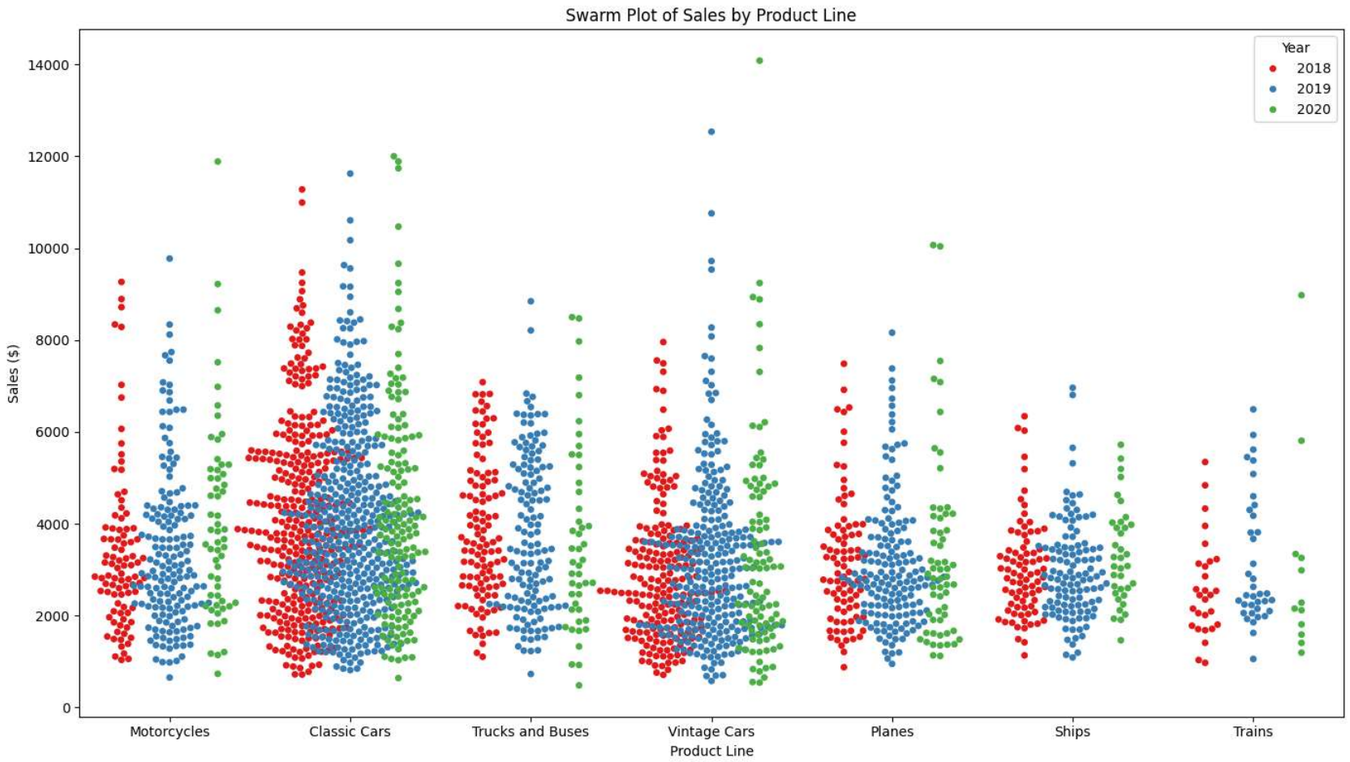
In [48]:
```python
# Stacked Bar Plot

data['ORDERDATE'] = pd.to_datetime(data['ORDERDATE'])
data['YEAR_MONTH'] = data['ORDERDATE'].dt.to_period('M')

# Aggregate sales by year-month and product line
pivot_data = data.pivot_table(
    index='YEAR_MONTH',
    columns='PRODUCTLINE',
    values='SALES',
    aggfunc='sum'
)

# Create a stacked bar plot
pivot_data.plot(kind='bar', stacked=True, figsize=(14, 8), colormap='tab20')

# Add labels and title
plt.title('Stacked Bar Plot of Sales by Product Line Over Time')
plt.xlabel('Year-Month')
plt.ylabel('Total Sales ($)')
plt.legend(title='Product Line', bbox_to_anchor=(1.05, 1), loc='upper left')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Grid and layout adjustments
# plt.grid(True)
plt.tight_layout()
plt.show()
```



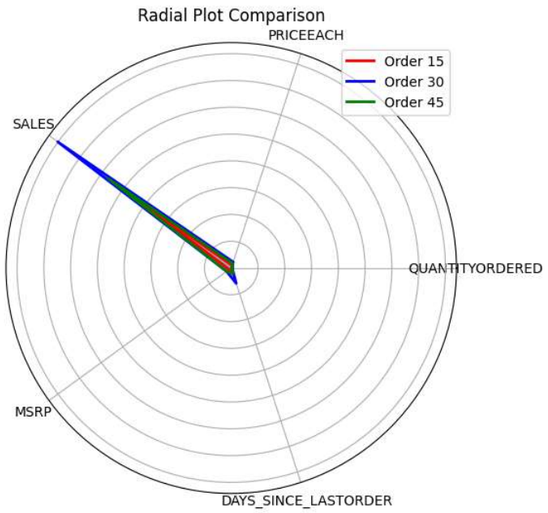
In [49]:
```
# Heatmap

# Calculate the correlation matrix
correlation_matrix = numerical_columns.corr()

# Create a mask for the upper triangle
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

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

# Create the heatmap with mask
sns.heatmap(correlation_matrix,
            mask=mask,
            annot=True,
            cmap='coolwarm',
            fmt='.2f',
            linewidths=0.5,
            vmin=-1,
            vmax=1,
            annot_kws={"size": 10},  # Adjust the size of the annotations
            cbar_kws={"shrink": .8, "aspect": 5, "pad": 0.1})  # Customize the color bar

# Add titles and labels with increased font size
plt.title('Heatmap of Correlation Matrix', fontsize=16, fontweight='bold')
plt.xlabel('Variables', fontsize=14)
plt.ylabel('Variables', fontsize=14)

plt.show()
```



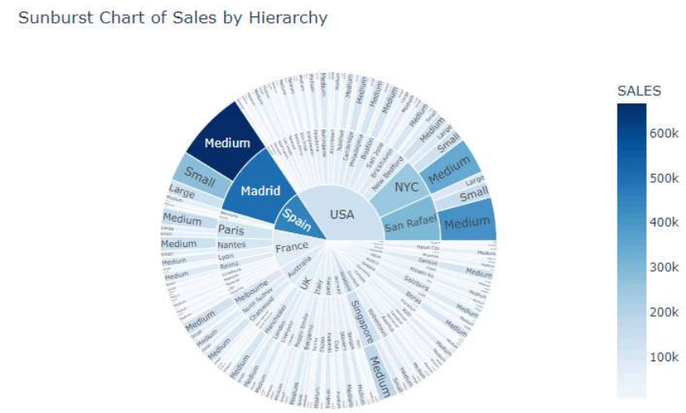
In [50]:
```
# Swarm plot

data['YEAR'] = data['ORDERDATE'].dt.year

plt.figure(figsize=(14, 8))
sns.swarmplot(data=data, x='PRODUCTLINE', y='SALES', hue='YEAR', palette='Set1', dodge=True)

# Add labels and title
plt.title('Swarm Plot of Sales by Product Line')
plt.xlabel('Product Line')
plt.ylabel('Sales ($)')

# Add a legend if using hue
plt.legend(title='Year')

# Grid and layout adjustments
# plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



In [51]:
```
# Radial Plot (Spider Plot)

# Selecting rows to compare
rows = [15, 30, 45]  # Example indices for comparison
colors = ['red', 'blue', 'green']  # Colors for different rows

# Data for plotting
labels = ['QUANTITYORDERED', 'PRICEEACH', 'SALES', 'MSRP', 'DAYS_SINCE_LASTORDER']
num_vars = len(labels)

# Create a plot
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))

for i, row_idx in enumerate(rows):
    example_row = data.loc[row_idx]
    values = [example_row['QUANTITYORDERED'], example_row['PRICEEACH'], example_row['SALES'], example_row['MSRP'], example_row['DAYS_SINCE_LASTORDER']]

    # Append the first value to close the plot
    values += values[:1]

    # Compute angle for each axis
    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()
    angles += angles[:1]

    # Plot
    ax.plot(angles, values, color=colors[i], linewidth=2, label=f'Order {row_idx}')
    ax.fill(angles, values, color=colors[i], alpha=0.25)

# Labels
ax.set_yticklabels([])
ax.set_xticks(angles[:-1])
ax.set_xticklabels(labels)

# Add legend
ax.legend(loc='best')

plt.title('Radial Plot Comparison')
plt.show()
```

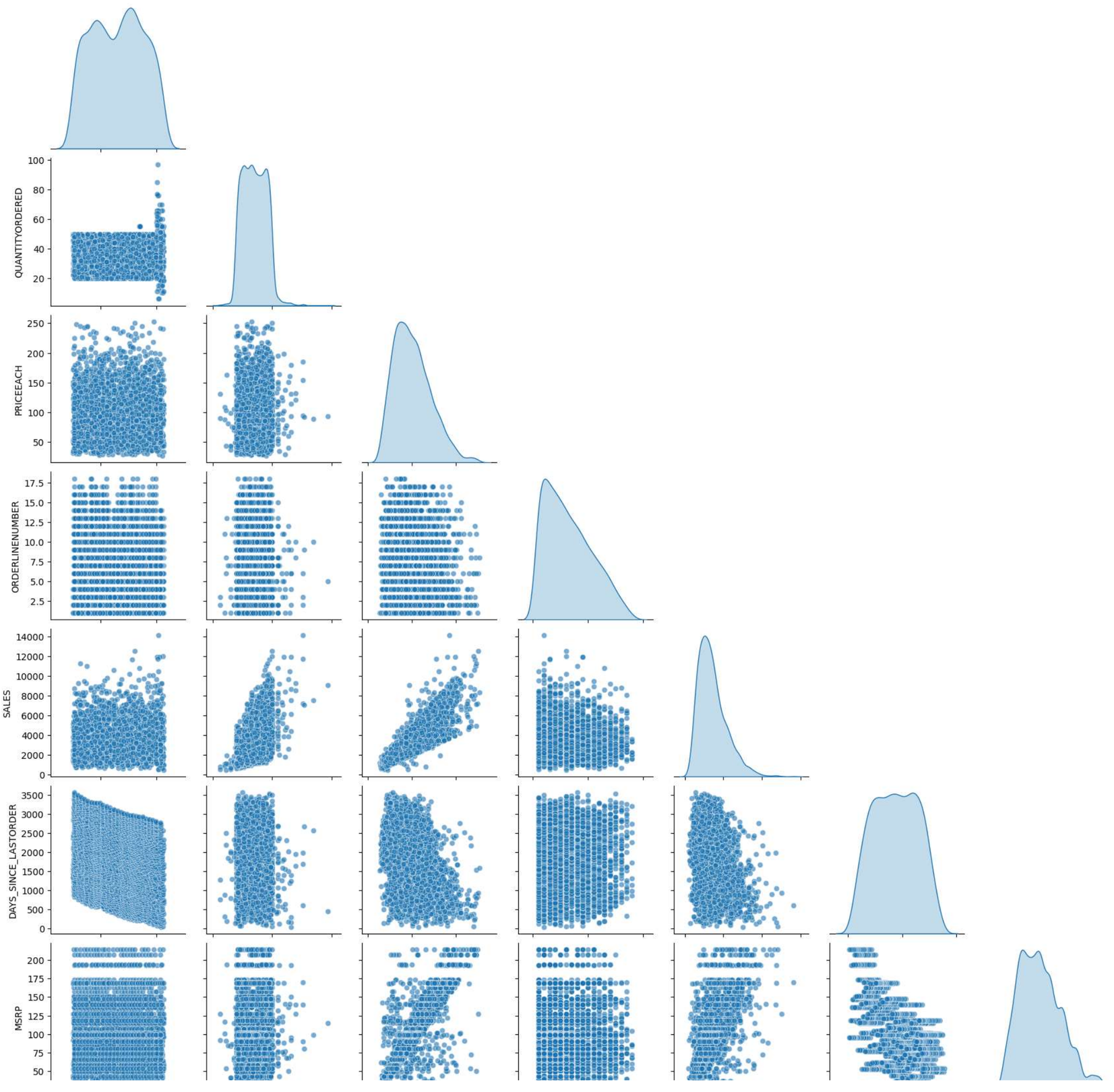


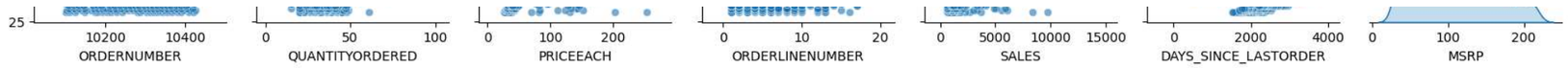
In [ ]:
```
# Sunburst Chart

# Create hierarchical data structure for the sunburst chart
sunburst_data = data.groupby(['COUNTRY', 'CITY', 'DEALSIZE']).agg({'SALES': 'sum'}).reset_index()

# Create the sunburst chart
fig = px.sunburst(sunburst_data,
                  path=['COUNTRY', 'CITY', 'DEALSIZE'],
                  values='SALES',
                  color='SALES',  # Optional: Color by sales for better visual differentiation
                  color_continuous_scale='Blues',  # Optional: Choose a color scale
                  title='Sunburst Chart of Sales by Hierarchy')

# Customize hover data
fig.update_traces(
    hovertemplate='<b>%{label}</b><br>Sales: %{value:,.0f}<extra></extra>'
)

fig.show()
```



In [53]:
```
# Pair plot

# Create the pair plot
pair_plot = sns.pairplot(numerical_columns,
                         diag_kind='kde',  # Use 'hist' for histograms or 'kde' for density plots
                         plot_kws={'alpha':0.6},  # Adjust transparency for better visibility
                         corner=True)  # Remove the upper triangle

# Set the title
pair_plot.fig.suptitle('Pair Plot of Numerical Variables', y=1.02)

plt.show()
```

Pair Plot of Numerical Variables





In [ ]:

```
# Sankey Diagram

# Convert ORDERDATE to datetime and extract year
data['ORDERDATE'] = pd.to_datetime(data['ORDERDATE'])
data['YEAR'] = data['ORDERDATE'].dt.year

# Filter data for the years 2018, 2019, and 2020
filtered_data = data[data['YEAR'].isin([2018, 2019, 2020])]

# Aggregate total sales by PRODUCTLINE and YEAR
sales_by_productline_year = filtered_data.groupby(['YEAR', 'PRODUCTLINE'])['SALES'].sum().reset_index()

# Prepare data for Sankey diagram
years = sales_by_productline_year['YEAR'].unique()
productlines = sales_by_productline_year['PRODUCTLINE'].unique()

# Create mappings for the nodes
year_to_index = {year: idx for idx, year in enumerate(years)}
productline_to_index = {productline: idx + len(years) for idx, productline in enumerate(productlines)}

# Create lists for Sankey diagram
sources = []
targets = []
values = []

for _, row in sales_by_productline_year.iterrows():
    source = year_to_index[row['YEAR']]
    target = productline_to_index[row['PRODUCTLINE']]
    sources.append(source)
    targets.append(target)
    values.append(row['SALES'])

# Define node labels
node_labels = list(years) + list(productlines)

# Create the Sankey diagram
fig = go.Figure(data=[go.Sankey(
    node=dict(
        pad=15,
        thickness=20,
        line=dict(color='black', width=0.5),
        label=node_labels
    ),
    link=dict(
        source=sources,
        target=targets,
        value=values,
        color='rgba(255, 0, 0, 0.5)'  # Optional: Color of the links
    )
)])

# Customize Layout
fig.update_layout(
    title_text='Sankey Diagram of Total Sales by Product Line for 2018, 2019, and 2020',
    font_size=10,
    height=600,
    width=800
)

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

Feature engineering

Feature selection

Domain knowledge

In [55]:
```
# Drop the unnecessary columns based on domain knowledge
# data = data.drop(columns=['ORDERNUMBER', 'ORDERLINENUMBER', 'PHONE', 'ADDRESSLINE1', 'POSTALCODE', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME'])

# Display the first few rows to verify the changes
# print(data.head())
```

VIF

In [56]:
```
import numpy as np
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

def calculate_vif(df):
    """Calculate VIF for each feature in the DataFrame."""
    df_with_const = add_constant(df)
    vif_data = pd.DataFrame()
    vif_data['Feature'] = df_with_const.columns
    vif_data['VIF'] = [variance_inflation_factor(df_with_const.values, i)
                       for i in range(df_with_const.shape[1])]
    return vif_data[vif_data['Feature'] != 'const']

def remove_high_vif_columns(data, max_vif=6):
    """Iteratively remove columns with VIF above the threshold."""
    while True:
        # Calculate VIF
        vif_data = calculate_vif(data)

        # Display the VIF values
        print("\nCurrent VIF values:")
        print(vif_data)

        # Check if all VIFs are below the threshold
        if vif_data['VIF'].max() < max_vif:
            print("\nAll VIF values are below the threshold.")
            break

        # Remove the column with the highest VIF
        column_to_remove = vif_data.sort_values('VIF', ascending=False).iloc[0]['Feature']
        print(f"\nRemoving column: {column_to_remove} with VIF: {vif_data['VIF'].max()}")
        data = data.drop(columns=[column_to_remove])

    return data

# Create a DataFrame with only numeric columns
numeric_data = data.select_dtypes(include=['float64', 'int64'])

# Check and handle missing and infinite values
print("Missing values in the dataset:")
print(numeric_data.isnull().sum())

print("\nInfinite values in the dataset:")
print((numeric_data == float('inf')).sum())

# Fill or handle missing values
numeric_data.fillna(numeric_data.mean(), inplace=True)

# Replace infinite values with NaN and then handle NaNs
numeric_data.replace([float('inf'), -float('inf')], np.nan, inplace=True)
numeric_data.fillna(numeric_data.mean(), inplace=True)

# Remove columns with high VIF iteratively
cleaned_data = remove_high_vif_columns(numeric_data)

# Display the final DataFrame after removing columns
print("\nFinal numerical data after VIF reduction:")
print(cleaned_data.head())
```

```
Missing values in the dataset:
ORDERNUMBER              0
QUANTITYORDERED          0
PRICEEACH                0
ORDERLINENUMBER          0
SALES                    0
DAYS_SINCE_LASTORDER     0
MSRP                     0
ROLLING_AVG             29
dtype: int64

Infinite values in the dataset:
ORDERNUMBER             0
QUANTITYORDERED         0
PRICEEACH               0
ORDERLINENUMBER         0
SALES                   0
DAYS_SINCE_LASTORDER    0
MSRP                    0
ROLLING_AVG             0
dtype: int64

Current VIF values:
                Feature        VIF
1           ORDERNUMBER   1.124155
2       QUANTITYORDERED   7.018928
3             PRICEEACH  15.667126
4       ORDERLINENUMBER   1.008941
5                 SALES  20.252091
6  DAYS_SINCE_LASTORDER   1.768444
7                  MSRP   3.640017
8           ROLLING_AVG   2.225270

Removing column: SALES with VIF: 20.2520913912722

Current VIF values:
                Feature       VIF
1           ORDERNUMBER  1.124053
2       QUANTITYORDERED  1.009409
3             PRICEEACH  2.549511
4       ORDERLINENUMBER  1.008202
5  DAYS_SINCE_LASTORDER  1.768253
6                  MSRP  3.638637
7           ROLLING_AVG  2.224672

All VIF values are below the threshold.

Final numerical data after VIF reduction:
   ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER  \
0        10107               30      95.70                2
1        10121               34      81.35                5
2        10134               41      94.74                2
3        10145               45      83.26                6
4        10168               36      96.66                1

   DAYS_SINCE_LASTORDER  MSRP  ROLLING_AVG
0                   828    95  3557.195741
1                   757    95  3557.195741
2                   703    95  3557.195741
3                   649    95  3557.195741
4                   586    95  3557.195741
```

RFE

In [57]:
```python
# Done based on model

from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import pandas as pd

# Define the path to the dataset
file_path = '/kaggle/input/auto-sales-data/Auto Sales data.csv'

# Read the dataset
data = pd.read_csv(file_path)

# Define columns to be excluded and categorical columns
columns_to_exclude = ['ORDERNUMBER', 'ORDERDATE', 'SALES', 'PHONE', 'ADDRESSLINE1', 'POSTALCODE', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME']
categorical_cols = ['STATUS', 'PRODUCTLINE', 'PRODUCTCODE', 'CUSTOMERNAME', 'CITY', 'COUNTRY', 'DEALSIZE']
numerical_cols = [col for col in data.columns if col not in columns_to_exclude + categorical_cols]

# Drop columns that are not needed
X_rfe = data.drop(columns=columns_to_exclude)
y_rfe = data['SALES']  # Target variable

# Handle missing values separately
# For numerical columns
num_imputer = SimpleImputer(strategy='mean')
X_rfe[numerical_cols] = num_imputer.fit_transform(X_rfe[numerical_cols])

# For categorical columns
cat_imputer = SimpleImputer(strategy='most_frequent')
X_rfe[categorical_cols] = cat_imputer.fit_transform(X_rfe[categorical_cols])

# Apply Label Encoding to categorical columns
```

```
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    X_rfe[col] = le.fit_transform(X_rfe[col])
    label_encoders[col] = le

# Ensure all remaining columns are numeric
non_numeric_cols = X_rfe.select_dtypes(include=['object']).columns
if len(non_numeric_cols) > 0:
    raise ValueError(f"Non-numeric columns found after encoding: {non_numeric_cols}")

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_rfe, y_rfe, test_size=0.3, random_state=42)

# Initialize the model (e.g., Linear Regression)
model = LinearRegression()

# Initialize RFE with the model and number of features to select
rfe = RFE(model, n_features_to_select=3)  # Adjust n_features_to_select based on your needs

# Fit RFE
rfe = rfe.fit(X_train, y_train)

# Get the ranking of features
feature_ranking = pd.DataFrame({
    'Feature': X_rfe.columns,
    'Ranking': rfe.ranking_
}).sort_values(by='Ranking')

# Display the ranking of features
print("Feature ranking:")
print(feature_ranking)

# Display selected features
selected_features = X_rfe.columns[rfe.support_]
print("\nSelected features:")
print(selected_features)
```

```
Feature ranking:
                Feature  Ranking
0        QUANTITYORDERED        1
1              PRICEEACH        1
11             DEALSIZE         1
4                STATUS         2
2        ORDERLINENUMBER        3
10               COUNTRY        4
6                  MSRP         5
5            PRODUCTLINE        6
7            PRODUCTCODE        7
9                  CITY         8
3   DAYS_SINCE_LASTORDER        9
8           CUSTOMERNAME        10

Selected features:
Index(['QUANTITYORDERED', 'PRICEEACH', 'DEALSIZE'], dtype='object')
```

RFECV

In [58]:

```
# Done based on model

from sklearn.feature_selection import RFECV
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Define the path to the dataset
file_path = '/kaggle/input/auto-sales-data/Auto Sales data.csv'

# Read the dataset
data = pd.read_csv(file_path)

# Define columns to be excluded and categorical columns
columns_to_exclude = ['ORDERNUMBER', 'ORDERDATE', 'SALES', 'PHONE', 'ADDRESSLINE1', 'POSTALCODE', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME']
categorical_cols = ['STATUS', 'PRODUCTLINE', 'PRODUCTCODE', 'CUSTOMERNAME', 'CITY', 'COUNTRY', 'DEALSIZE']
numerical_cols = [col for col in data.columns if col not in columns_to_exclude + categorical_cols]

# Drop columns that are not needed
X_rfe = data.drop(columns=columns_to_exclude)
y_rfe = data['SALES']  # Target variable

# Apply Label Encoding to categorical columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    X_rfe[col] = le.fit_transform(X_rfe[col])
    label_encoders[col] = le

# Ensure all remaining columns are numeric
non_numeric_cols = X_rfe.select_dtypes(include=['object']).columns
if len(non_numeric_cols) > 0:
    raise ValueError(f"Non-numeric columns found after encoding: {non_numeric_cols}")

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_rfe, y_rfe, test_size=0.3, random_state=42)

# Initialize the model (e.g., Linear Regression)
model = LinearRegression()

# Initialize RFECV with the model and cross-validation settings
```

```
rfecv = RFECV(estimator=model, step=1, cv=5)  # 5-fold cross-validation

# Fit RFECV
rfecv = rfecv.fit(X_train, y_train)

# Get the ranking of features
feature_ranking = pd.DataFrame({
    'Feature': X_rfe.columns,
    'Ranking': rfecv.ranking_
}).sort_values(by='Ranking')

# Display the ranking of features
print("Feature ranking:")
print(feature_ranking)

# Display selected features
selected_features = X_rfe.columns[rfecv.support_]
print("\nSelected features:")
print(selected_features)

# Display the optimal number of features
print("\nOptimal number of features:")
print(rfecv.n_features_)
```

```
Feature ranking:
                 Feature  Ranking
0        QUANTITYORDERED        1
1             PRICEEACH        1
11              DEALSIZE        1
4                 STATUS        2
2        ORDERLINENUMBER        3
10               COUNTRY        4
6                   MSRP        5
5            PRODUCTLINE        6
7            PRODUCTCODE        7
9                   CITY        8
3   DAYS_SINCE_LASTORDER        9
8          CUSTOMERNAME       10

Selected features:
Index(['QUANTITYORDERED', 'PRICEEACH', 'DEALSIZE'], dtype='object')

Optimal number of features:
3
```

Modified dataset for further modelling

In [59]:
```
# Define the path to the dataset
file_path = '/kaggle/input/auto-sales-data/Auto Sales data.csv'

# Read the dataset
data = pd.read_csv(file_path)

# Define columns to be excluded based on Domain expertise and VIF
columns_to_exclude = ['ORDERNUMBER', 'PHONE', 'ADDRESSLINE1', 'POSTALCODE', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME']

# Drop columns that are not needed
data_eda = data.drop(columns=columns_to_exclude)

# Convert data types

# Convert ORDERDATE to datetime if it's not already
data_eda['ORDERDATE'] = pd.to_datetime(data_eda['ORDERDATE'])

# Convert 'Order Line Number' to object data type
data_eda['ORDERLINENUMBER'] = data_eda['ORDERLINENUMBER'].astype('object')

# Set ORDERDATE as the index
data_eda.set_index('ORDERDATE', inplace=True)

# Show first 5 records
data_eda.head()
```

Out[59]:

ORDERDATE	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	DAYS_SINCE_LASTORDER	STATUS	PRODUCTLINE	MSRP	PRODUCTCODE	CUSTOMERNAME	CITY	COUNTRY	DEALSIZE
2018-02-24	30	95.70	2	2871.00	828	Shipped	Motorcycles	95	S10_1678	Land of Toys Inc.	NYC	USA	Small
2018-05-07	34	81.35	5	2765.90	757	Shipped	Motorcycles	95	S10_1678	Reims Collectables	Reims	France	Small
2018-07-01	41	94.74	2	3884.34	703	Shipped	Motorcycles	95	S10_1678	Lyon Souveniers	Paris	France	Medium
2018-08-25	45	83.26	6	3746.70	649	Shipped	Motorcycles	95	S10_1678	Toys4GrownUps.com	Pasadena	USA	Medium
2018-10-28	36	96.66	1	3479.76	586	Shipped	Motorcycles	95	S10_1678	Technics Stores Inc.	Burlingame	USA	Medium

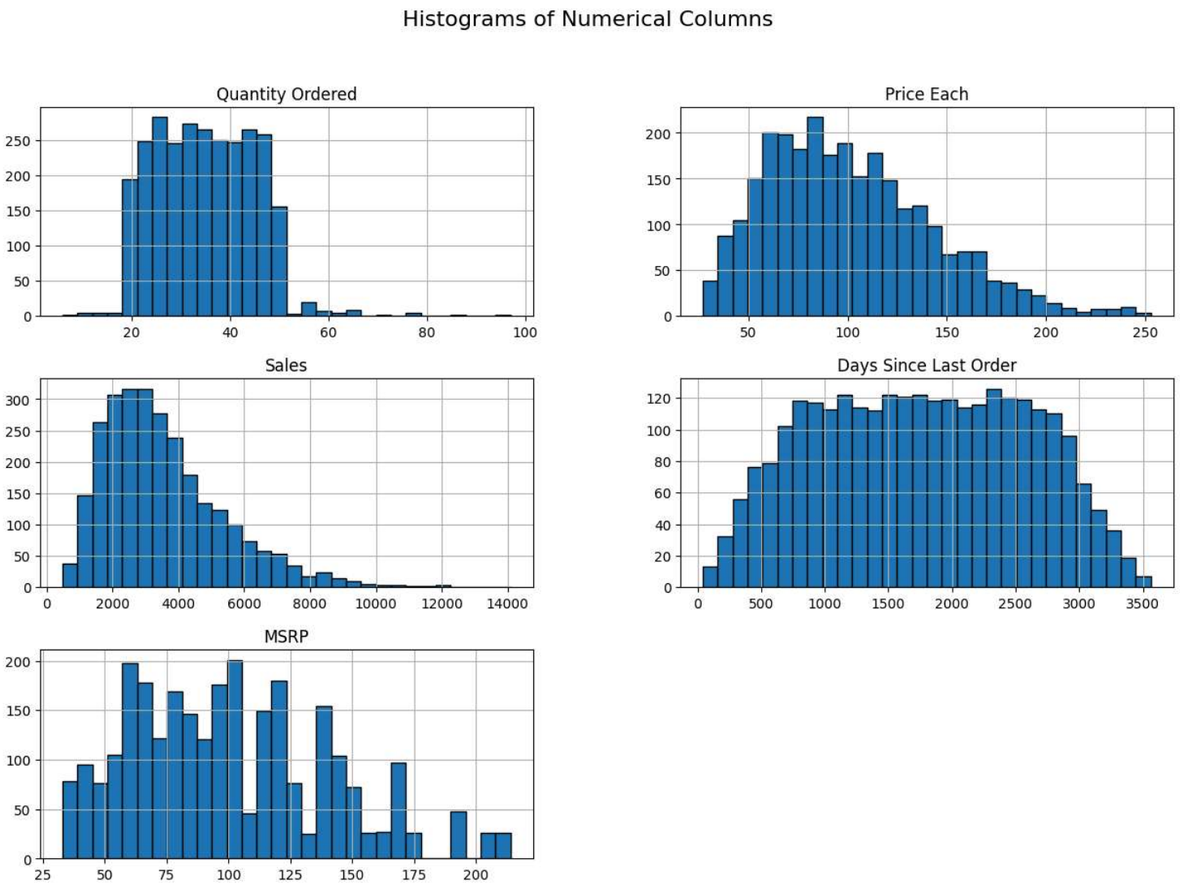
In [60]:
```
# Define the new column names
new_column_names = {
    'QUANTITYORDERED': 'Quantity Ordered',
    'PRICEEACH': 'Price Each',
    'ORDERLINENUMBER': 'Order Line Number',
    'SALES': 'Sales',
    'DAYS_SINCE_LASTORDER': 'Days Since Last Order',
    'STATUS': 'Status',
    'PRODUCTLINE': 'Product Line',
    'MSRP': 'MSRP',
    'PRODUCTCODE': 'Product Code',
    'CUSTOMERNAME': 'Customer Name',
```

```
    'CITY': 'City',
    'COUNTRY': 'Country',
    'DEALSIZE': 'Deal Size',
}

# Rename the columns
data_eda.rename(columns=new_column_names, inplace=True)

data_eda.head()
```

Out[60]:

ORDERDATE	Quantity Ordered	Price Each	Order Line Number	Sales	Days Since Last Order	Status	Product Line	MSRP	Product Code	Customer Name	City	Country	Deal Size
2018-02-24	30	95.70	2	2871.00	828	Shipped	Motorcycles	95	S10_1678	Land of Toys Inc.	NYC	USA	Small
2018-05-07	34	81.35	5	2765.90	757	Shipped	Motorcycles	95	S10_1678	Reims Collectables	Reims	France	Small
2018-07-01	41	94.74	2	3884.34	703	Shipped	Motorcycles	95	S10_1678	Lyon Souveniers	Paris	France	Medium
2018-08-25	45	83.26	6	3746.70	649	Shipped	Motorcycles	95	S10_1678	Toys4GrownUps.com	Pasadena	USA	Medium
2018-10-28	36	96.66	1	3479.76	586	Shipped	Motorcycles	95	S10_1678	Technics Stores Inc.	Burlingame	USA	Medium

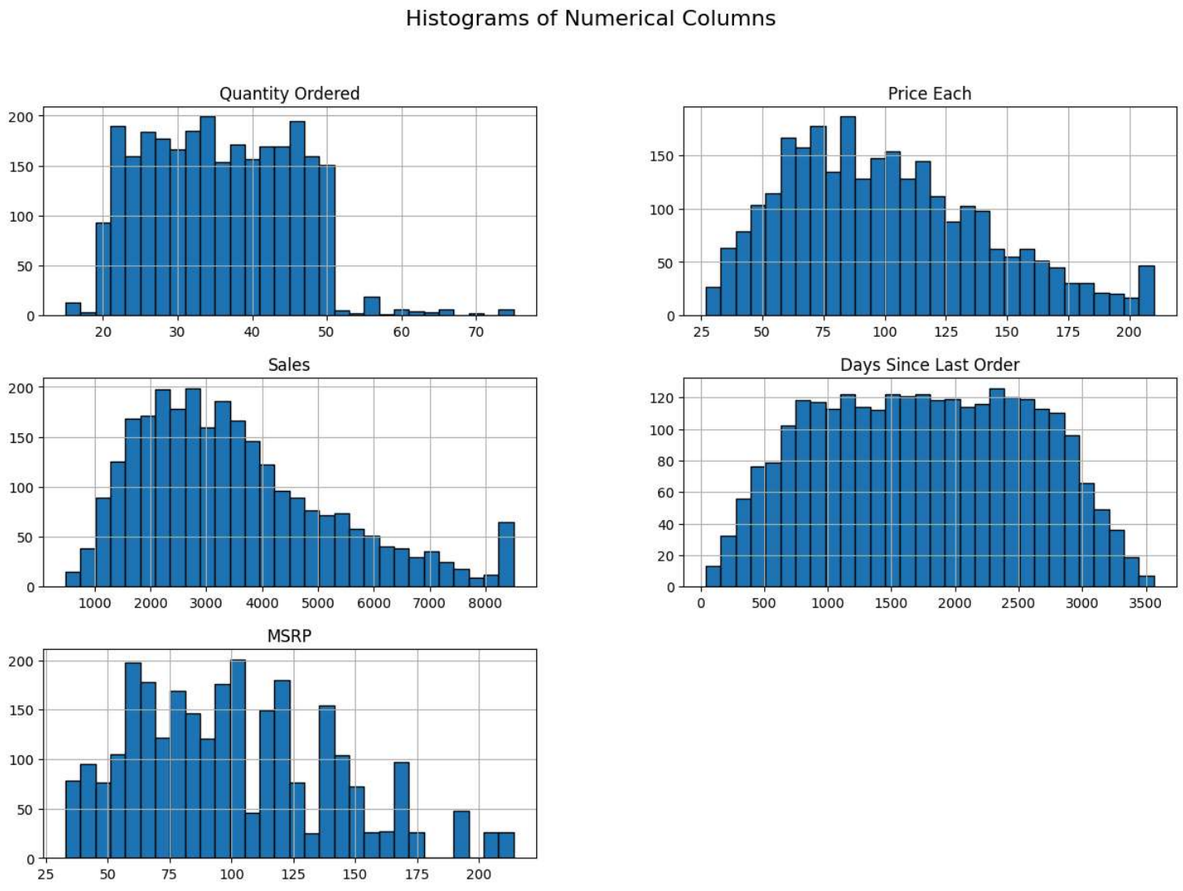
In [61]:
```
# Select numerical columns
numerical_columns = data_eda.select_dtypes(include=['int64', 'float64']).columns

# Plot histograms for all numerical columns
data_eda[numerical_columns].hist(bins=30, figsize=(15, 10), edgecolor='black')
plt.suptitle('Histograms of Numerical Columns', fontsize=16)
plt.show()
```



In [62]:
```
# Apply clipping to 'Quantity Ordered'
data_eda['Quantity Ordered'] = data_eda['Quantity Ordered'].clip(lower=15, upper=75)

# Apply clipping to 'Price Each'
data_eda['Price Each'] = data_eda['Price Each'].clip(lower=0, upper=210)

# Apply clipping to 'Sales'
data_eda['Sales'] = data_eda['Sales'].clip(lower=0, upper=8500)

# Optionally, check the result
print(data_eda[['Quantity Ordered', 'Price Each', 'Sales']].describe())
```

```
       Quantity Ordered   Price Each        Sales
count       2747.000000  2747.000000  2747.000000
mean          35.104478   100.806676  3530.953735
std            9.659969    41.177685  1762.691378
min           15.000000    26.880000   482.130000
25%           27.000000    68.745000  2204.350000
50%           35.000000    95.550000  3184.800000
75%           43.000000   127.100000  4503.095000
max           75.000000   210.000000  8500.000000
```

In [63]:
```
# Select numerical columns
numerical_columns = data_eda.select_dtypes(include=['int64', 'float64']).columns

# Plot histograms for all numerical columns
data_eda[numerical_columns].hist(bins=30, figsize=(15, 10), edgecolor='black')
plt.suptitle('Histograms of Numerical Columns', fontsize=16)
plt.show()
```



We have selected the important columns after that we have changed the data type of few columns.

Then we reset the index. Later we checked for outliers and fix the issue with clipping method.

Now we will create some plots from above and try to gain some insights.

Insights from the cleaned dataset

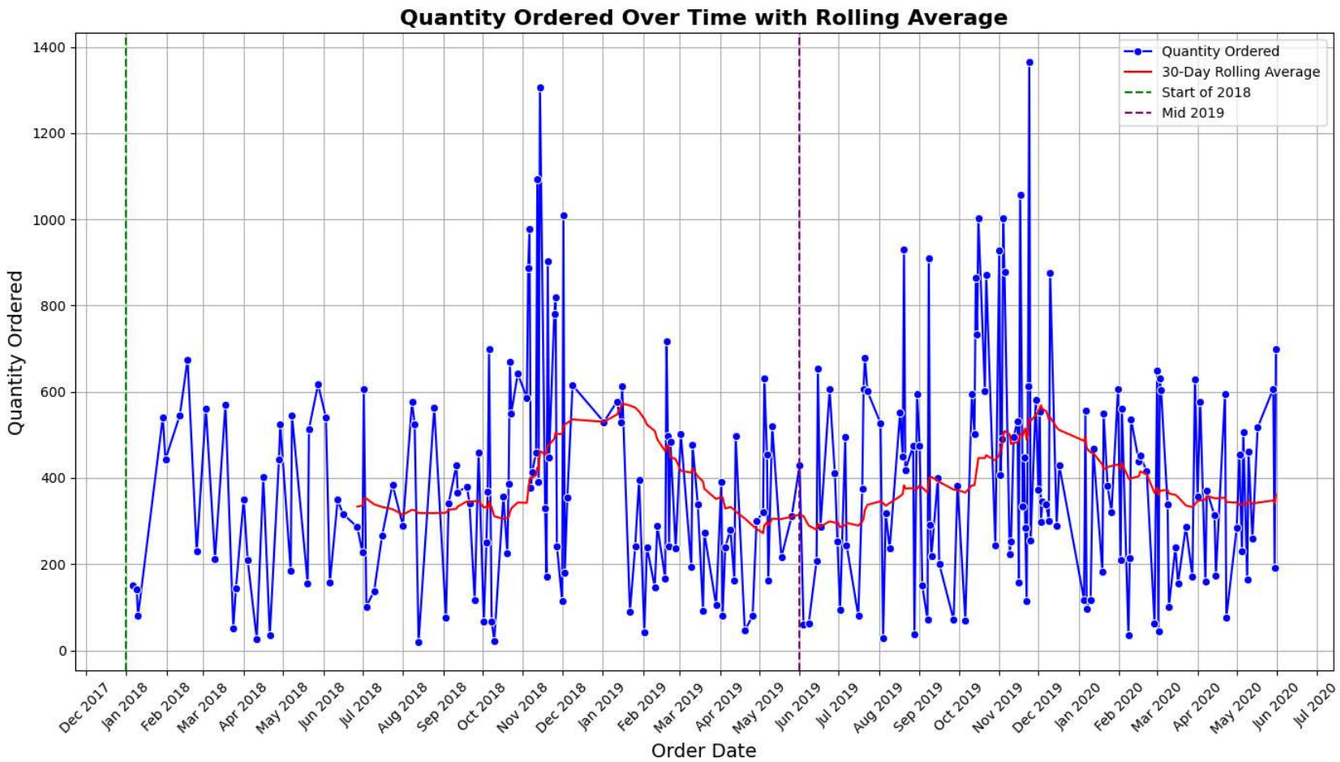
In [64]:
```
data_eda.head()
```

Out[64]:

ORDERDATE	Quantity Ordered	Price Each	Order Line Number	Sales	Days Since Last Order	Status	Product Line	MSRP	Product Code	Customer Name	City	Country	Deal Size
2018-02-24	30	95.70	2	2871.00	828	Shipped	Motorcycles	95	S10_1678	Land of Toys Inc.	NYC	USA	Small
2018-05-07	34	81.35	5	2765.90	757	Shipped	Motorcycles	95	S10_1678	Reims Collectables	Reims	France	Small
2018-07-01	41	94.74	2	3884.34	703	Shipped	Motorcycles	95	S10_1678	Lyon Souveniers	Paris	France	Medium
2018-08-25	45	83.26	6	3746.70	649	Shipped	Motorcycles	95	S10_1678	Toys4GrownUps.com	Pasadena	USA	Medium
2018-10-28	36	96.66	1	3479.76	586	Shipped	Motorcycles	95	S10_1678	Technics Stores Inc.	Burlingame	USA	Medium

Line Plot between Order date and Quantity

In [65]:
```
# Group by ORDERDATE and sum Quantity Ordered
daily_summary = data_eda.groupby('ORDERDATE')['Quantity Ordered'].sum().reset_index()

plt.figure(figsize=(14, 8))

# Plot the line for Quantity Ordered
sns.lineplot(x='ORDERDATE', y='Quantity Ordered', data=daily_summary, marker='o', color='b', label='Quantity Ordered')

# Add a rolling average (e.g., 30-day rolling average)
daily_summary['ROLLING_AVG'] = daily_summary['Quantity Ordered'].rolling(window=30).mean()
sns.lineplot(x='ORDERDATE', y='ROLLING_AVG', data=daily_summary, color='r', label='30-Day Rolling Average')

# Add titles and labels
plt.title('Quantity Ordered Over Time with Rolling Average', fontsize=16, fontweight='bold')
plt.xlabel('Order Date', fontsize=14)
plt.ylabel('Quantity Ordered', fontsize=14)

# Add grid
plt.grid(True)

# Rotate date labels for better readability
plt.xticks(rotation=45)

# Highlight significant dates or periods (adjust dates as needed)
plt.axvline(pd.Timestamp('2018-01-01'), color='g', linestyle='--', label='Start of 2018')
plt.axvline(pd.Timestamp('2019-06-01'), color='purple', linestyle='--', label='Mid 2019')

# Add a legend
plt.legend()

# Format x-axis dates
plt.gca().xaxis.set_major_formatter(plt.matplotlib.dates.DateFormatter('%b %Y'))
plt.gca().xaxis.set_major_locator(plt.matplotlib.dates.MonthLocator())

# Adjust layout to fit labels
plt.tight_layout()

plt.show()
```



Insights gained: There is no trend in the quantity ordered over time. But a seasonality pattern can be seen in the month of Nov and Dec. There is also a lot of noise in the line plot which makes this time series graph Non stationary.

Scatter Plot between Price Each and Sales

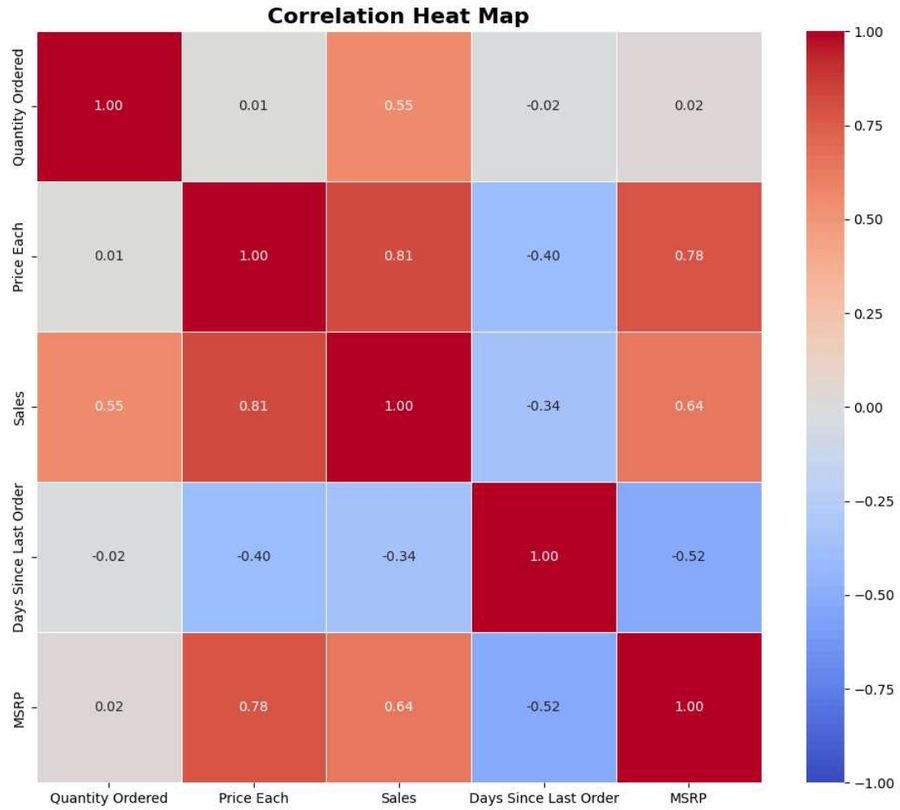
In [66]:
```
plt.figure(figsize=(10, 6))
plt.scatter(data_eda['Price Each'], data_eda['Sales'], alpha=0.5, c='r', edgecolors='w', s=100)
plt.title('Scatter Plot of Price Each vs. Sales')
plt.xlabel('Price Each')
plt.ylabel('Sales')
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

Insights gained: Thre is a positive correlation between Price and Sales. Which means price may help in predicting the sales.

Heat Map

In [67]:
```
# Filter only numerical columns
numerical_data = data_eda.select_dtypes(include=['number'])

# Compute the correlation matrix
corr_matrix = numerical_data.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, vmin=-1, vmax=1)
plt.title('Correlation Heat Map', fontsize=16, fontweight='bold')
plt.show()
```

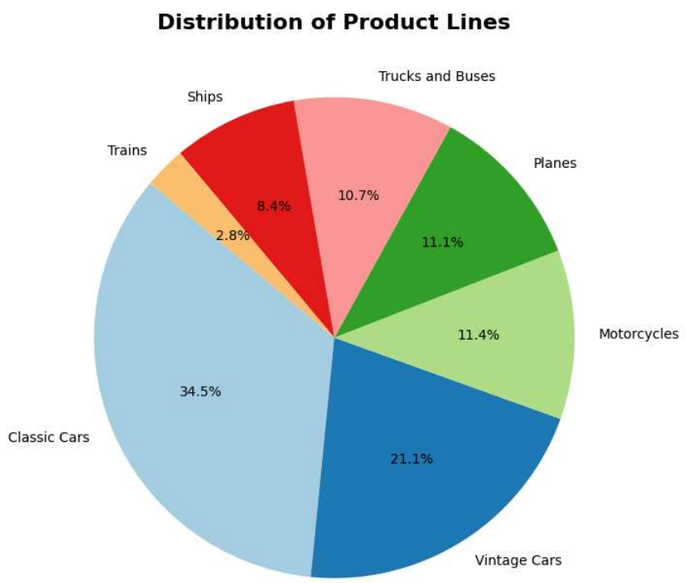


Insights gained: Positive correltaion between: Price and Sales // Price and MSRP // Sales and MSRP

Negative correlation between: MSRP and Days since last order

Pie Chart for Product line

In [68]:
```
# Aggregate the count of each Product Line
product_line_counts = data_eda['Product Line'].value_counts()

plt.figure(figsize=(8, 8))
plt.pie(product_line_counts, labels=product_line_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired(range(len(product_line_counts))))
plt.title('Distribution of Product Lines', fontsize=16, fontweight='bold')
plt.show()
```



Insights gained: Classic cars has the highest market share followed by Vintage cars whiles trains have the least market share by units.

Pivot table for summary

In [69]:
```
# Pivot table to aggregate Sales by Status and Product Line
pivot_table = data_eda.pivot_table(index=['Status', 'Product Line'],
                                   values='Sales',
                                   aggfunc='sum').reset_index()

print(pivot_table)
```

```
       Status      Product Line       Sales
0    Cancelled      Classic Cars    59242.81
1    Cancelled            Planes    35432.71
2    Cancelled             Ships    56665.65
3    Cancelled            Trains     5082.42
4    Cancelled      Vintage Cars    38063.89
5     Disputed      Classic Cars    24043.97
6     Disputed       Motorcycles    31103.74
7     Disputed            Planes     3843.84
8     Disputed             Ships     3070.40
9     Disputed      Vintage Cars     7463.85
10  In Process      Classic Cars    54259.66
11  In Process  Trucks and Buses    43026.41
12  In Process      Vintage Cars    43942.89
13     On Hold      Classic Cars    49039.65
14     On Hold       Motorcycles     4992.61
15     On Hold            Planes    34727.53
16     On Hold             Ships    23664.61
17     On Hold            Trains     5808.48
18     On Hold  Trucks and Buses    20193.29
19     On Hold      Vintage Cars    34970.22
20    Resolved      Classic Cars    25799.34
21    Resolved            Planes    34532.92
22    Resolved             Ships    39863.71
23    Resolved  Trucks and Buses    20472.75
24    Resolved      Vintage Cars    29664.76
25     Shipped      Classic Cars  3596311.11
26     Shipped       Motorcycles  1060515.95
27     Shipped            Planes   857680.22
28     Shipped             Ships   576774.85
29     Shipped            Trains   214875.52
30     Shipped  Trucks and Buses  1027522.62
31     Shipped      Vintage Cars  1636877.53
```

Insights gained: Non of the trains are in disputed status.

Vintage Cars and Motorcycles are not cancelled.

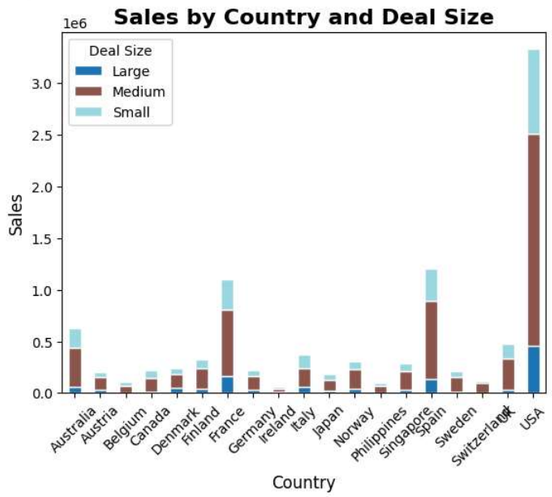
Aditionally all issues related Motorcycles and Trains.

Stacked Bar Chart for Country and Deall Size

In [70]:
```
# Aggregate sales by Country and Deal Size
bar_data = data_eda.groupby(['Country', 'Deal Size'])['Sales'].sum().unstack()

# Plot the stacked bar chart
plt.figure(figsize=(18,12))
bar_data.plot(kind='bar', stacked=True, colormap='tab20', edgecolor='w')
```

```
plt.title('Sales by Country and Deal Size', fontsize=16, fontweight='bold')
plt.xlabel('Country', fontsize=12)
plt.ylabel('Sales', fontsize=12)
plt.legend(title='Deal Size')
# plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(rotation=45)
#plt.tight_layout()

plt.show()
```

```
<Figure size 1800x1200 with 0 Axes>
```



Insights gained: Medium and large deal comes mostly from Australia, France, Spain, UK and USA.

Specific queries: Top 5 salws giving Country and City

In [71]:
```
# Aggregate total sales by Country
country_sales = data_eda.groupby('Country')['Sales'].sum().reset_index()

# Get the top 5 countries by total sales
top_5_countries = country_sales.sort_values(by='Sales', ascending=False).head(5).reset_index(drop=True)

print("Top 5 sales-generating countries are:")
print(top_5_countries)

# Aggregate total sales by City
city_sales = data_eda.groupby('City')['Sales'].sum().reset_index()

# Get the top 5 cities by total sales
top_5_cities = city_sales.sort_values(by='Sales', ascending=False).head(5).reset_index(drop=True)

print("\nTop 5 sales-generating cities are:")
print(top_5_cities)
```

```
Top 5 sales-generating countries are:
     Country       Sales
0        USA  3331682.51
1      Spain  1207045.46
2     France  1100664.69
3  Australia   628584.21
4         UK   475345.22

Top 5 sales-generating cities are:
         City       Sales
0      Madrid  1073909.98
1  San Rafael   649658.92
2         NYC   558008.57
3   Singapore   285645.70
4       Paris   266188.83
```

Here we are concluding our EDA with insights Now the ML team can take this information along with the dataset to make models.