COMP5721M: Programming for Data Science

Coursework 3: Data Analysis Project

Last modified: 30 November 2023

Consumer Study based on Automobile Sales Data

- Sayan Banerjee, mm23sb2@leeds.ac.uk
- Mohita Parwani, mm23mp@leeds.ac.uk
- Shivam Sharma, mm232ss@leeds.ac.uk
- Rajarshi Nandi, mm23rn@leeds.ac.uk

Project Plan

The Data (10 marks)

Selecting the dataset for a project where all data analysis and data science skills need to be put to use is the most intregal part of the project. Hence, The dataset is sourced from Kaggle (https://www.kaggle.com/datasets/ddosad/auto-sales-data/data) [1]. It encompasses information on customer orders, individual prices of items sold in bulk orders, and other relevant details and presents a variety of information that provides crucial insights into the sales trends, purchasing and behavior, offering a valuable resource for businesses working to understand and deliver to their customer base more effectively. This dataset, contains a total of 2747 records, each contributing to a deep exploration of customer preferences and purchase trends.

A deeper understanding of the attributes of the dataset are as follows [1]:

- ORDERNUMBER: This column represents the unique identification number assigned to each order.
- QUANTITYORDERED: It indicates the number of items ordered in each order.
- PRICEEACH: This column specifies the price of each item in the order.
- ORDERLINENUMBER: It represents the line number of each item within an order.
- SALES: This column denotes the total sales amount for each order, which is calculated by multiplying the quantity ordered by the price of each item.
- ORDERDATE: It denotes the date on which the order was placed.
- DAYS_SINCE_LASTORDER: This column represents the number of days that have passed since the last order for each customer. It can be used to analyze customer purchasing patterns.
- STATUS: It indicates the status of the order, such as "Shipped," "In Process," "Cancelled," "Disputed," "On Hold," or "Resolved."

- PRODUCTLINE: This column specifies the product line categories to which each item belongs.
- MSRP: It stands for Manufacturer's Suggested Retail Price and represents the suggested selling price for each item.
- PRODUCTCODE: This column represents the unique code assigned to each product.
- CUSTOMERNAME: It denotes the name of the customer who placed the order.
- PHONE: This column contains the contact phone number for the customer.
- ADDRESSLINE1: It represents the first line of the customer's address.
- CITY: This column specifies the city where the customer is located.
- POSTALCODE: It denotes the postal code or ZIP code associated with the customer's address.
- COUNTRY: This column indicates the country where the customer is located.
- CONTACTLASTNAME: It represents the last name of the contact person associated with the customer.
- CONTACTFIRSTNAME: This column denotes the first name of the contact person associated with the customer.
- DEALSIZE: It indicates the size of the deal or order, which are the categories "Small," "Medium," or "Large."

Altogether, sales datasets are a very important part in the world of business as they help in understanding purchasing behaviour at a deeper level along with contributing towards building marketing strategies that increase profitability, tailoring product offers, enhance customer satisfaction and to build a monopoly in the market of retailing.

Project Aim and Objectives (5 marks)

The primary aim of this project is to conduct a comprehensive analysis of the Automobile Sales Dataset, using various data analysis and machine learning techniques. By looking into into the details and attributes of customer details and purchasing patterns, the project seeks to extract actionable insights that businesses can employ to optimize their products, marketing strategies, and overall profitability.

The project will begin with exploratory data analysis (EDA) to gain a deep understanding of the dataset's structure and characteristics. This involves statistical summaries, visualizations, and the identification of patterns and trends within the data. Subsequently, data cleaning and preprocessing steps will be implemented to ensure the dataset's integrity and prepare it for analysis.

Clustering algorithms will be employed to identify distinct customer segments based on different criteria such as an RFM analysis, allowing for targeted marketing strategies tailored to specific groups of customers.

Following the Clustering of data, Machine learning models will be implemented such that the data is trained and tested to provide prediction for a target variable, helping the business to be prepared according to the proposed predictions. Along with modelling, accuracy of the models will be tested and the best among them will be chosen.

Furthermore, the project aims to generate insightful visualizations, such as interactive charts and graphical representations, to communicate the findings effectively. Visualizations will be created using popular Python libraries like Matplotlib, and Seaborn, providing a compelling and accessible means to present complex patterns and trends within the dataset.

To sum up, the objective of this project is to not only study the Automobile Sales Dataset but also to offer practical suggestions to companies trying to match their marketing plans with market demands. The project's goal is to give organizations the tools they need to stay competitive in a changing market, improve customer happiness, and make informed decisions by combining machine learning modeling, customer segmentation, exploratory data analysis, and impactful visualizations.

Specific Objective(s)

Project Objectives:

1. **Exploratory Analysis of Purchase Patterns:**
   - Conduct in-depth exploratory data analysis to uncover patterns, relationships between vairables and trends in customer purchase behavior, considering factors such as deal size, sales, product line, and geographic variables.
2. **Customer Segmentation and Profiling [2]:**
   - Employ clustering algorithms to identify distinct customer segments based on recency, frequency and monetary value.
3. **Predictive Modeling:**
   - Develop a predictive model to determine the price of each order using variables such as MSRP.
4. **Visualization of Predictive models:**
   - Create visualizations to represent the accuracy of different models by comparing true values vs predicted values and therefore choosing the best model.

System Design (5 marks)

Architecture

1) Data Collection: Raw data is collected from various sources, such as customer transactions, preferences, and interactions.

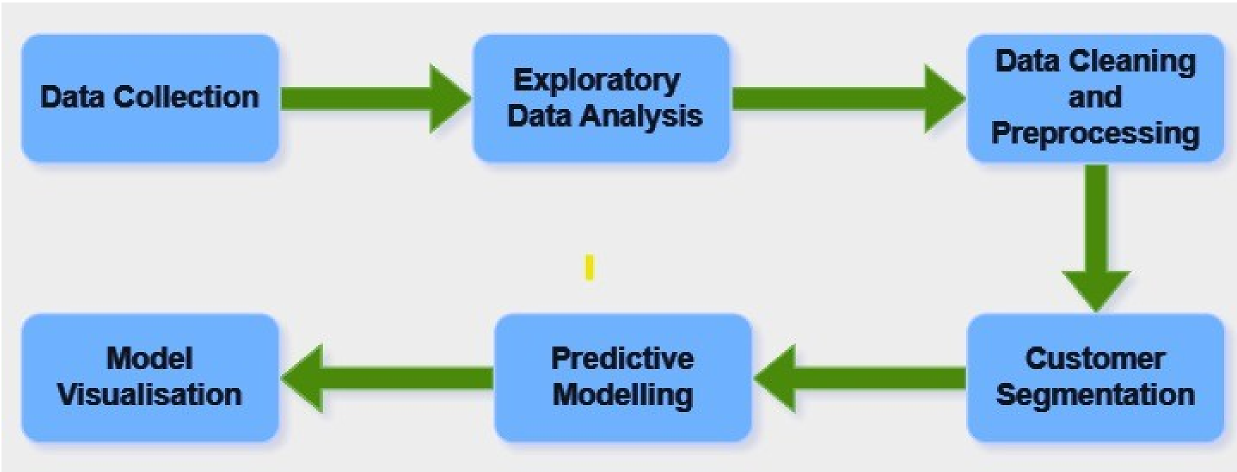
2) Exploratory Data Analysis (EDA): The raw data undergoes EDA, where statistical analyses, visualizations, and pattern identification take place. This phase provides a deep understanding of the dataset.

3) Data Cleaning and Preprocessing: The dataset is cleaned and preprocessed to ensure data integrity and to prepare it for training different models on it.

4) Machine Learning Modeling: Utilizing machine learning models, relationships between customer attributes and purchasing decisions are explored. This involves classification models for predicting various customer behaviors and clustering algorithms to identify distinct customer segments.

5) Visualization: The project generates insightful visualizations using Python libraries like Matplotlib, Seaborn, and Plotly. These visualizations include interactive dashboards and

graphical representations, making it easier for stakeholders to comprehend complex patterns and trends.



Processing Modules and Algorithms

1. **Exploratory Data Analysis (EDA):**
   - **Objective:** Gain insights into the dataset and uncover patterns, relationships, and trends in customer purchase behavior.
   - **Algorithm/Technique:** Use descriptive statistics, data visualization (such as histograms, scatter plots, and heatmaps), and correlation analysis to explore the distribution of variables, identify patterns, and understand the underlying structure of the data. Additionally, to perform outlier detection and handling as part of the exploratory process to ensure the data's integrity for subsequent analysis.
2. **Customer Segmentation using Clustering [2]:**
   - **Objective:** Identify distinct customer segments based on recency, frequency, and monetary value (RFM) [2].
   - **Algorithm/Technique:** Utilize clustering algorithms such as K-means or hierarchical clustering to group customers with similar purchasing behavior. This involves transforming the data into feature vectors representing RFM values.
3. **Predictive Modeling for Price Determination:**
   - **Objective:** Develop a model to predict the price of each order based on relevant variables such as MSRP and RFM score.
   - **Algorithm/Technique:** Employ regression algorithms such as linear regression or more advanced methods like random forests or bayesian ridge. This involves training the model on historical data with known prices and then applying it to predict prices for test data.
4. **Visualization of Model Accuracy:**
   - **Objective:** Evaluate and compare the accuracy of different predictive models.
   - **Algorithm/Technique:** Utilize visualizations such as scatter plots or line charts to compare true values against predicted values from different models. This involves implementing metrics like Mean Squared Error (MSE) or R-squared to quantify model performance.

Program Code (15 marks)

Importing necessary libraries and data

```
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, Lasso, Ridge, 
BayesianRidge
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Suppressing warnings
warnings.filterwarnings('ignore')
```

Importing the data by creating a dataframe of it

```
df = pd.read_csv('Auto Sales data.csv', parse_dates = ['ORDERDATE'], 
dayfirst = True)
```

Getting basic information on the data such as top 5 rows, shape and information about the variables

```
display(df.head())
print('\n\nShape of the dataset: ',df.shape, '\n\n')
print(df.info())
```

```
   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 2018-02-24                   828  Shipped  Motorcycles    95
S10_1678
1 2018-05-07                   757  Shipped  Motorcycles    95
S10_1678
2 2018-07-01                   703  Shipped  Motorcycles    95
S10_1678
3 2018-08-25                   649  Shipped  Motorcycles    95
S10_1678
4 2018-10-28                   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



Shape of the dataset:  (2747, 20)


<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   datetime64[ns]
 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: datetime64[ns](1), float64(2), int64(5), object(12)
memory usage: 429.3+ KB
None
```

The output presents the initial rows of the DataFrame 'df' to offer a glimpse of its structure. Additionally, it provides essential details about the dataset, including its shape (number of rows and columns) and a summary of data types, aiding in a quick overview of the dataset's characteristics.

Checking for duplicate values in the 'ORDEREDNUMBER' column

```
print(df['ORDERNUMBER'].duplicated().any(), '\n\n')

display(df[df['ORDERNUMBER'] == 10107].sort_values(by =
'ORDERLINENUMBER'))
print('\n\n')
print(df.info(), '\n\n')
display(round(df.describe(),2).T)
```

```
True
```

```
      ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER
SALES  \
226         10107               21     144.60                1
3036.60
0           10107               30      95.70                2
2871.00
1415        10107               25     113.83                3
2845.75
74          10107               27     224.65                4
6065.55
50          10107               39      99.91                5
3896.49
773         10107               29      70.87                6
2055.23
1510        10107               38      83.03                7
3155.14
2104        10107               20      92.90                8
1858.00

      ORDERDATE  DAYS_SINCE_LASTORDER   STATUS  PRODUCTLINE  MSRP
PRODUCTCODE  \
226  2018-02-24                  1054  Shipped  Motorcycles   150
S12_2823
0    2018-02-24                   828  Shipped  Motorcycles    95
S10_1678
1415 2018-02-24                  2243  Shipped  Motorcycles   112
S24_1578
74   2018-02-24                   902  Shipped  Motorcycles   193
S10_4698
50   2018-02-24                   878  Shipped  Motorcycles   118
S10_2016
773  2018-02-24                  1601  Shipped  Motorcycles    60
S18_2625
1510 2018-02-24                  2338  Shipped  Motorcycles    76
S24_2000
2104 2018-02-24                  2932  Shipped  Motorcycles    99
S32_1374

           CUSTOMERNAME       PHONE             ADDRESSLINE1 CITY
POSTALCODE  \
226   Land of Toys Inc.  2125557818  897 Long Airport Avenue  NYC
10022
0     Land of Toys Inc.  2125557818  897 Long Airport Avenue  NYC
10022
1415  Land of Toys Inc.  2125557818  897 Long Airport Avenue  NYC
10022
74    Land of Toys Inc.  2125557818  897 Long Airport Avenue  NYC
10022
50    Land of Toys Inc.  2125557818  897 Long Airport Avenue  NYC
10022
```

```
773   Land of Toys Inc.  2125557818  897 Long Airport Avenue  NYC
10022
1510  Land of Toys Inc.  2125557818  897 Long Airport Avenue  NYC
10022
2104  Land of Toys Inc.  2125557818  897 Long Airport Avenue  NYC
10022

     COUNTRY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
226      USA               Yu             Kwai   Medium
0        USA               Yu             Kwai    Small
1415     USA               Yu             Kwai    Small
74       USA               Yu             Kwai   Medium
50       USA               Yu             Kwai   Medium
773      USA               Yu             Kwai    Small
1510     USA               Yu             Kwai   Medium
2104     USA               Yu             Kwai    Small
```

```
<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   datetime64[ns]
 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: datetime64[ns](1), float64(2), int64(5), object(12)
memory usage: 429.3+ KB
None
```

```
                       count      mean      std       min       25%
50%  \
ORDERNUMBER           2747.0  10259.76    91.88  10100.00  10181.00
10264.00
QUANTITYORDERED       2747.0     35.10     9.76      6.00     27.00
35.00
PRICEEACH             2747.0    101.10    42.04     26.88     68.74
95.55
ORDERLINENUMBER       2747.0      6.49     4.23      1.00      3.00
6.00
SALES                 2747.0   3553.05  1838.95    482.13   2204.35
3184.80
DAYS_SINCE_LASTORDER  2747.0   1757.09   819.28     42.00   1077.00
1761.00
MSRP                  2747.0    100.69    40.11     33.00     68.00
99.00

                           75%       max
ORDERNUMBER           10334.50  10425.00
QUANTITYORDERED          43.00     97.00
PRICEEACH               127.10    252.87
ORDERLINENUMBER           9.00     18.00
SALES                  4503.09  14082.80
DAYS_SINCE_LASTORDER   2436.50   3562.00
MSRP                    124.00    214.00
```

In many business scenarios, an order may consist of multiple items, each with its own quantity, order line number, and price. Therefore, it's common to see duplicate order numbers in a dataset where each row corresponds to a unique item within an order.

For example, consider the following hypothetical scenario:

Order 10107 has two items:

Item 1: Quantity 3, Order Line Number 1, Price $20

Item 2: Quantity 2, Order Line Number 2, Price $15

In this case, both rows would have the same order number (10107) and order date but differ in quantity, order line number, and price.

This code adds a 'DAYS_SINCE_LASTORDER' feature to the DataFrame 'df,' representing the days between each order's 'ORDERDATE' and June 1, 2020

We already had a 'DAYS_SINCE_LASTORDER' column in the dataset present but failed to validate the values in that row when compared with the date of last order of each customer.

Hence, to avoid data integrity and validation issue we decided to recalculate the entire column based on the logic explained later.

```
#Creating new feature 'DAYS_SINCE_LASTORDER' based on the time
difference:
df = df.drop('DAYS_SINCE_LASTORDER', axis = 1)
df['DAYS_SINCE_LASTORDER'] = (datetime(2020, 6, 1)-
df['ORDERDATE']).dt.days
df.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   STATUS  PRODUCTLINE  MSRP PRODUCTCODE
CUSTOMERNAME  \
0 2018-02-24  Shipped  Motorcycles    95    S10_1678     Land of Toys
Inc.
1 2018-05-07  Shipped  Motorcycles    95    S10_1678    Reims
Collectables
2 2018-07-01  Shipped  Motorcycles    95    S10_1678       Lyon
Souveniers
3 2018-08-25  Shipped  Motorcycles    95    S10_1678
Toys4GrownUps.com
4 2018-10-28  Shipped  Motorcycles    95    S10_1678  Technics Stores
Inc.

              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

  COUNTRY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
```

```
DAYS_SINCE_LASTORDER
0     USA              Yu             Kwai    Small
828
1  France         Henriot             Paul    Small
756
2  France        Da Cunha           Daniel   Medium
701
3     USA           Young            Julie   Medium
646
4     USA          Hirano             Juri   Medium
582
```

This code snippet generates descriptive statistics for object-type columns in the DataFrame 'df' using the describe() method, providing insights into categorical data characteristics.

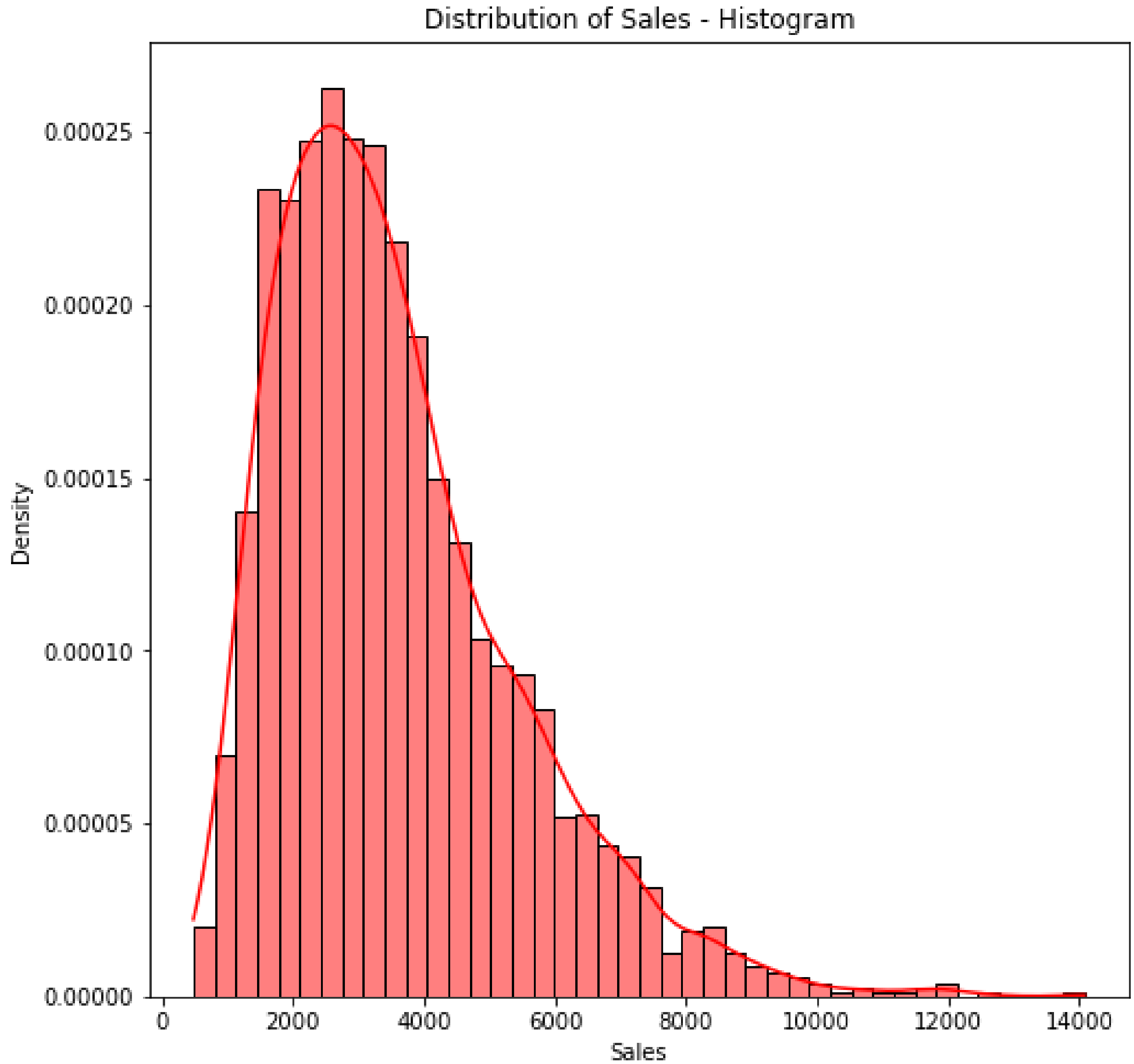
```
df.select_dtypes(include = ['object']).describe().T
```

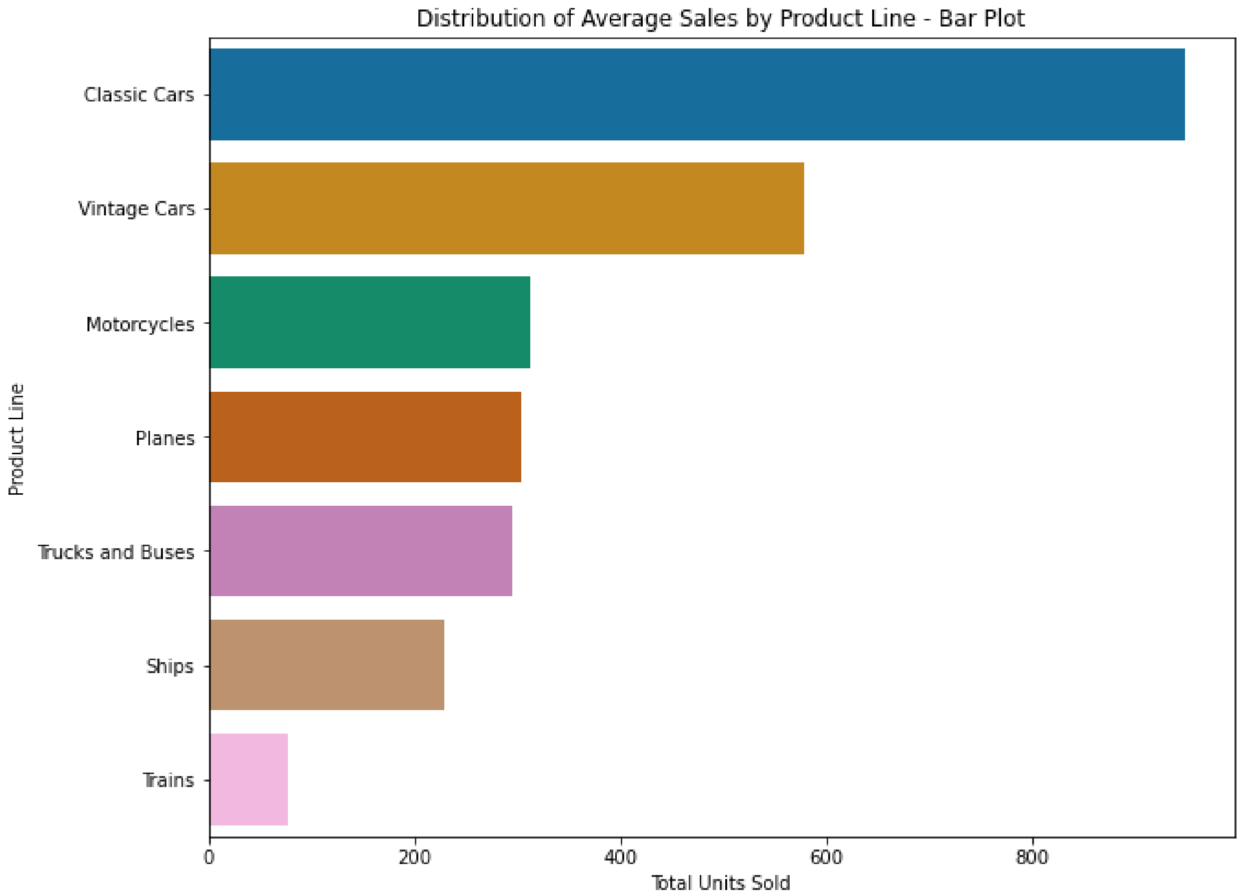
	count	unique	top	freq
STATUS	2747	6	Shipped	2541
PRODUCTLINE	2747	7	Classic Cars	949
PRODUCTCODE	2747	109	S18_3232	51
CUSTOMERNAME	2747	89	Euro Shopping Channel	259
PHONE	2747	88	(91) 555 94 44	259
ADDRESSLINE1	2747	89	C/ Moralzarzal, 86	259
CITY	2747	71	Madrid	304
POSTALCODE	2747	73	28034	259
COUNTRY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

Exploratory data analysis (EDA)

This code visualizes sales distribution through a histogram and product line distribution via a countplot in the DataFrame 'df.'

```
#Distribution of Sales:
plt.figure(figsize = (8,8))
sns.histplot(df['SALES'], kde = True, color = 'red', stat = 'density')
plt.title('Distribution of Sales - Histogram')
plt.xlabel('Sales')
plt.ylabel('Density')
plt.show()
print('\n\n')

#Distribution of Product line:
plt.figure(figsize = (10,8))
sns.countplot(y = 'PRODUCTLINE', data = df, palette = 'colorblind',
order = df['PRODUCTLINE'].value_counts().index)
plt.title('Distribution of Average Sales by Product Line - Bar Plot')
plt.xlabel('Total Units Sold')
plt.ylabel('Product Line')
plt.show()
print('\n\n')
```

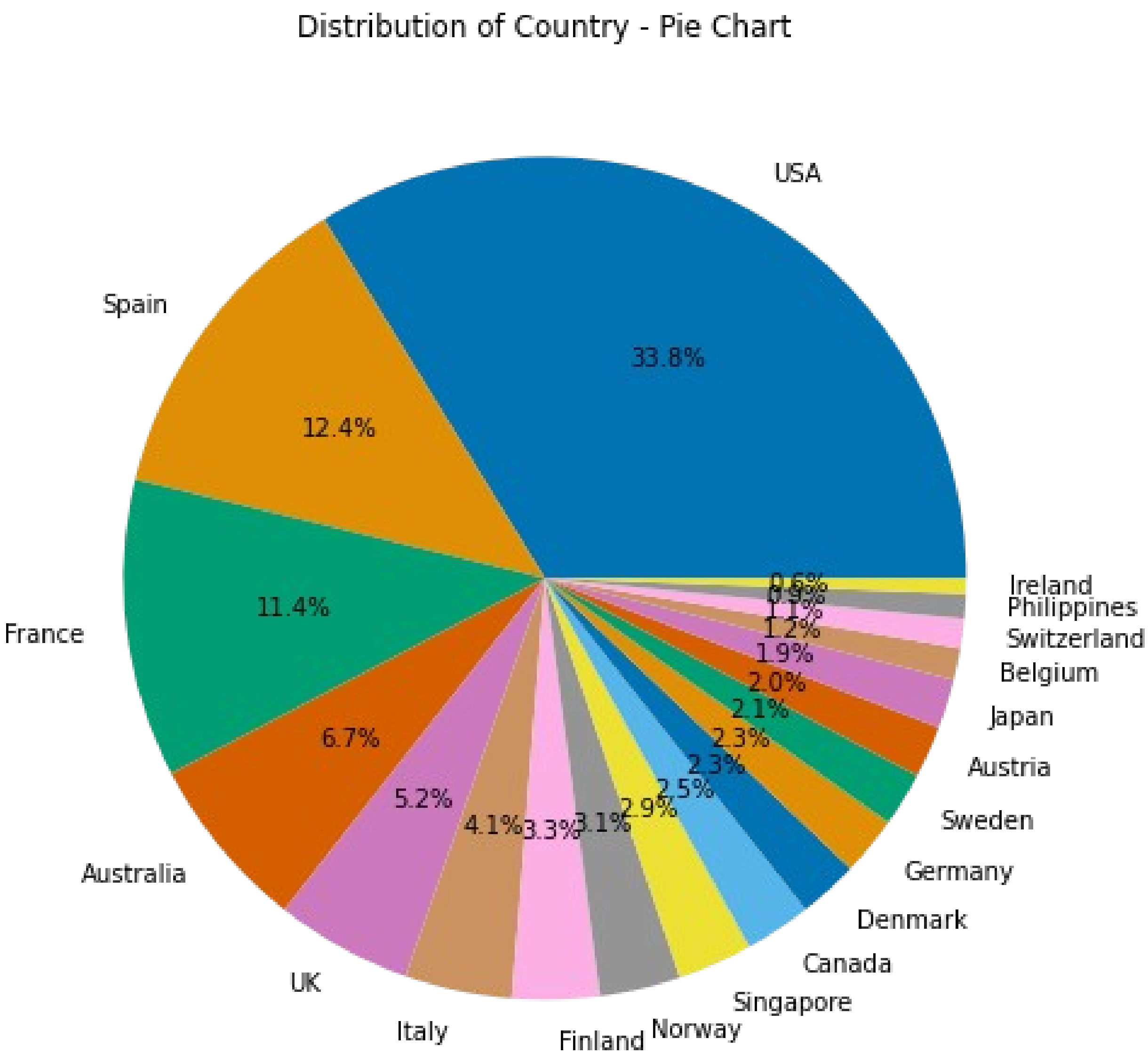

Distribution of Sales - Histogram
Density
Sales
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025
0
2000
4000
6000
8000
10000
12000
14000



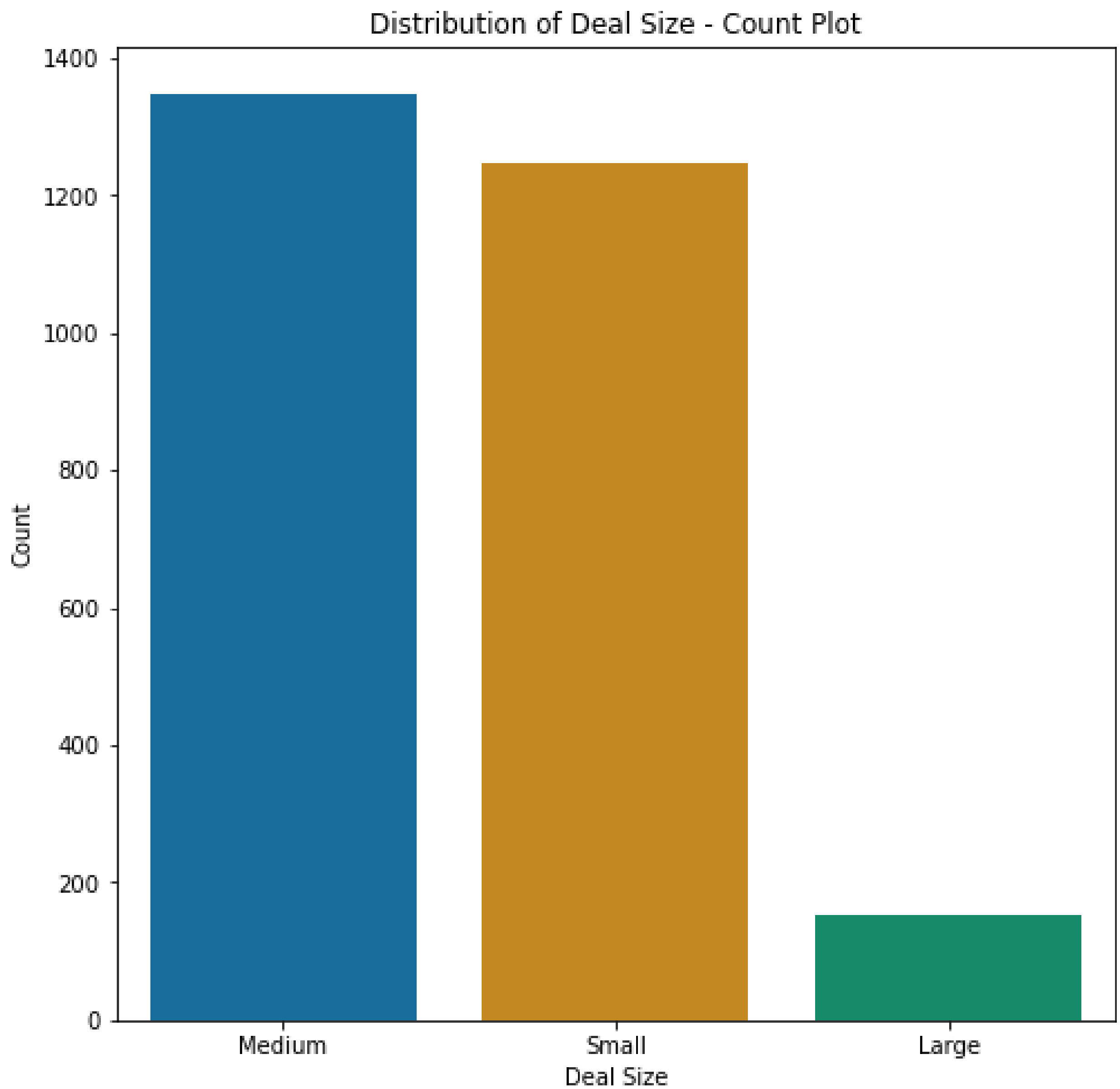
This code visually represents the distribution of countries using a pie chart and the distribution of deal sizes through a count plot in the DataFrame 'df.'

```
#Distribution of Country:
plt.figure(figsize = (10,8))
country_counts = df['COUNTRY'].value_counts()
plt.pie(country_counts, labels = country_counts.index, autopct =
'%1.1f%%', colors = sns.color_palette('colorblind'))
plt.title('Distribution of Country - Pie Chart')
plt.show()
print('\n\n')

#Distribution of Deal Size:
plt.figure(figsize = (8,8))
sns.countplot(x = 'DEALSIZE', data = df, palette = 'colorblind', order
= df['DEALSIZE'].value_counts().index)
```

```
plt.title('Distribution of Deal Size - Count Plot')
plt.xlabel('Deal Size')
plt.ylabel('Count')
plt.show()
print('\n\n')
```



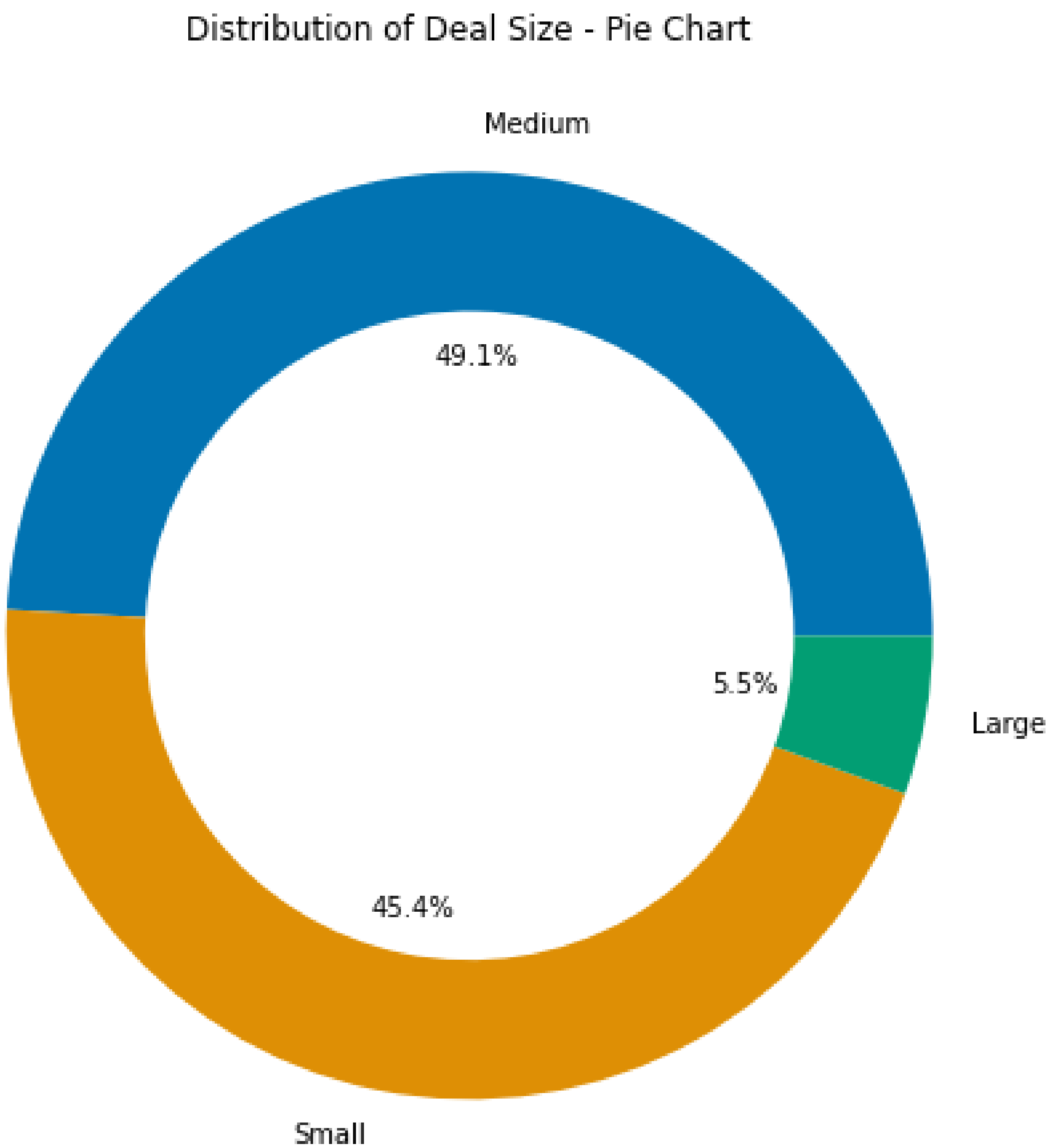

```

```

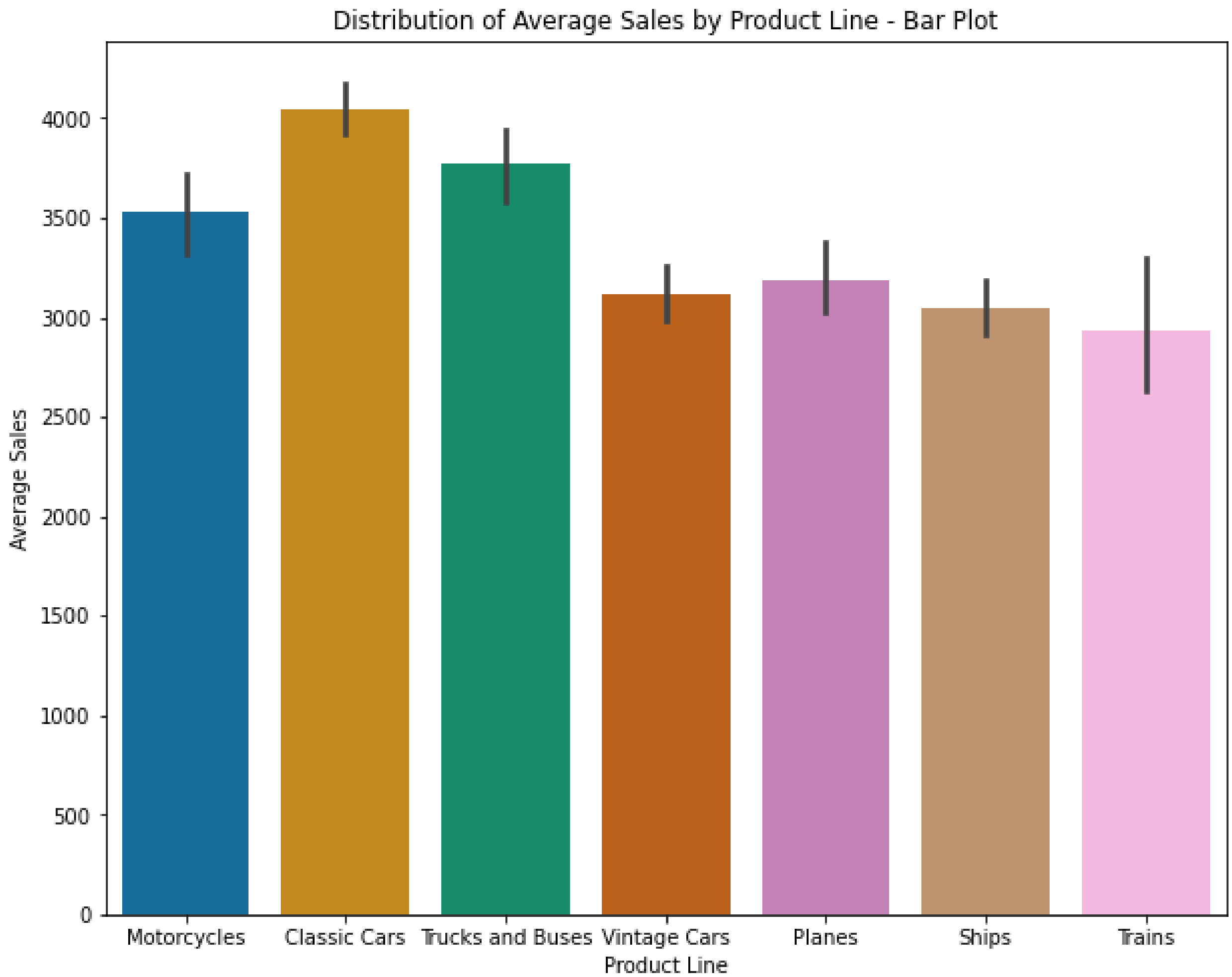


This code visually represents the distribution of deal sizes through a pie chart and the distribution of average sales by product line using a bar plot in the DataFrame 'df.'

```
#Distribution of Deal Size - Pie Chart:
plt.figure(figsize = (8,8))
count_dealsize = df['DEALSIZE'].value_counts()
```

```
plt.pie(count_dealsize, labels = count_dealsize.index, autopct =
'%1.1f%%', colors = sns.color_palette('colorblind'), wedgeprops =
dict(width = 0.3))
plt.title('Distribution of Deal Size - Pie Chart')
plt.show()

#Distribution of Average Sales by Product line:
plt.figure(figsize = (10,8))
sns.barplot(x = 'PRODUCTLINE', y = 'SALES', data = df, palette =
'colorblind')
plt.title('Distribution of Average Sales by Product Line - Bar Plot')
plt.xlabel('Product Line')
plt.ylabel('Average Sales')
plt.show()
print('\n\n')
```

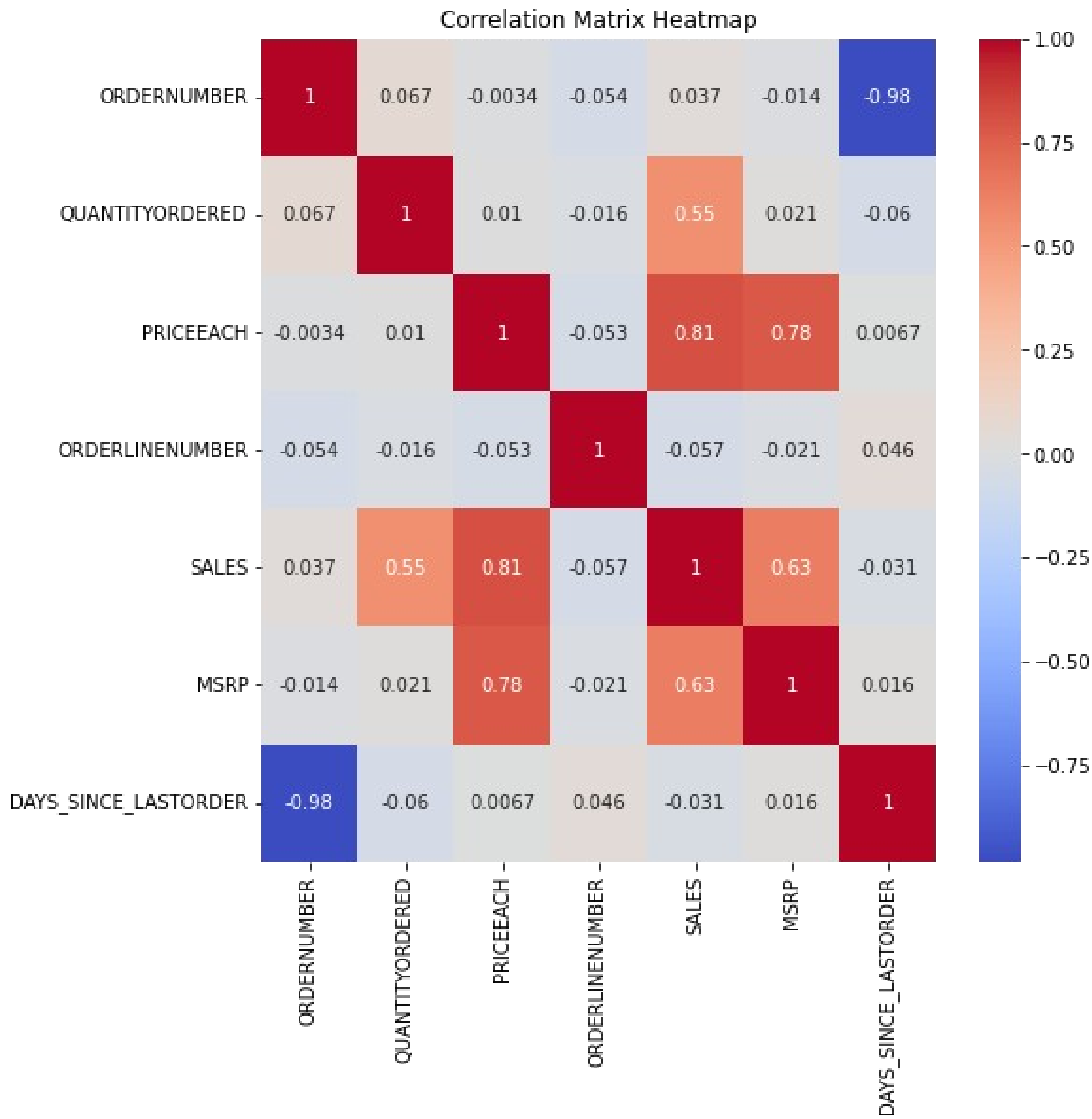

Distribution of Deal Size - Pie Chart
Medium
49.1%
5.5%
Large
45.4%
Small



This code calculates the correlation matrix for variables in the DataFrame 'df' and visualizes the relationships using a heatmap.

```
#Finding relationships between variables:

correlation_matrix = df.corr()
print(correlation_matrix, '\n\n')
plt.figure(figsize = (8,8))
sns.heatmap(correlation_matrix, annot = True, cmap = 'coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
                      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   
MSRP                    -0.013910         0.020551   0.778393   
DAYS_SINCE_LASTORDER    -0.982862        -0.059549   0.006688   

                      ORDERLINENUMBER     SALES      MSRP  \
ORDERNUMBER                 -0.054300  0.037289 -0.013910   
QUANTITYORDERED             -0.016295  0.553359  0.020551   
PRICEEACH                   -0.052646  0.808287  0.778393   
ORDERLINENUMBER              1.000000 -0.057414 -0.020956   
SALES                       -0.057414  1.000000  0.634849   
MSRP                        -0.020956  0.634849  1.000000   
DAYS_SINCE_LASTORDER         0.045635 -0.030891  0.016465   

                      DAYS_SINCE_LASTORDER  
ORDERNUMBER                      -0.982862  
QUANTITYORDERED                  -0.059549  
PRICEEACH                         0.006688  
ORDERLINENUMBER                   0.045635  
SALES                            -0.030891  
MSRP                              0.016465  
DAYS_SINCE_LASTORDER              1.000000  
```



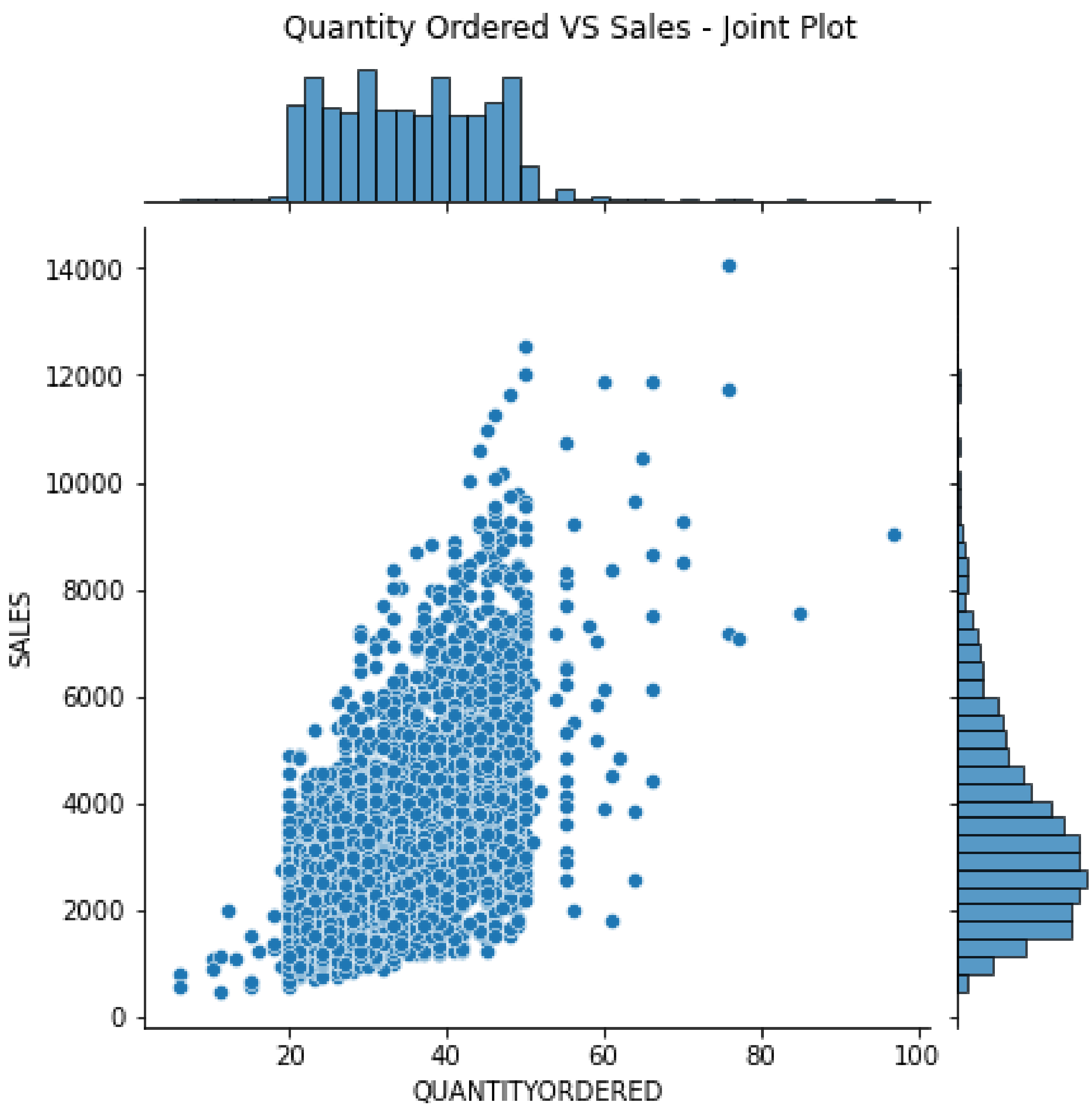
From the above heatmap we can see that QUANTITYORDERED vs SALES, PRICEEACH vs SALES, PRICEEACH vs MSRP and SALES vs MSRP are the strongest relationships.

This code explores the relationship between 'QUANTITYORDERED' and 'SALES' through two joint plots: a scatter plot and a hexbin plot.

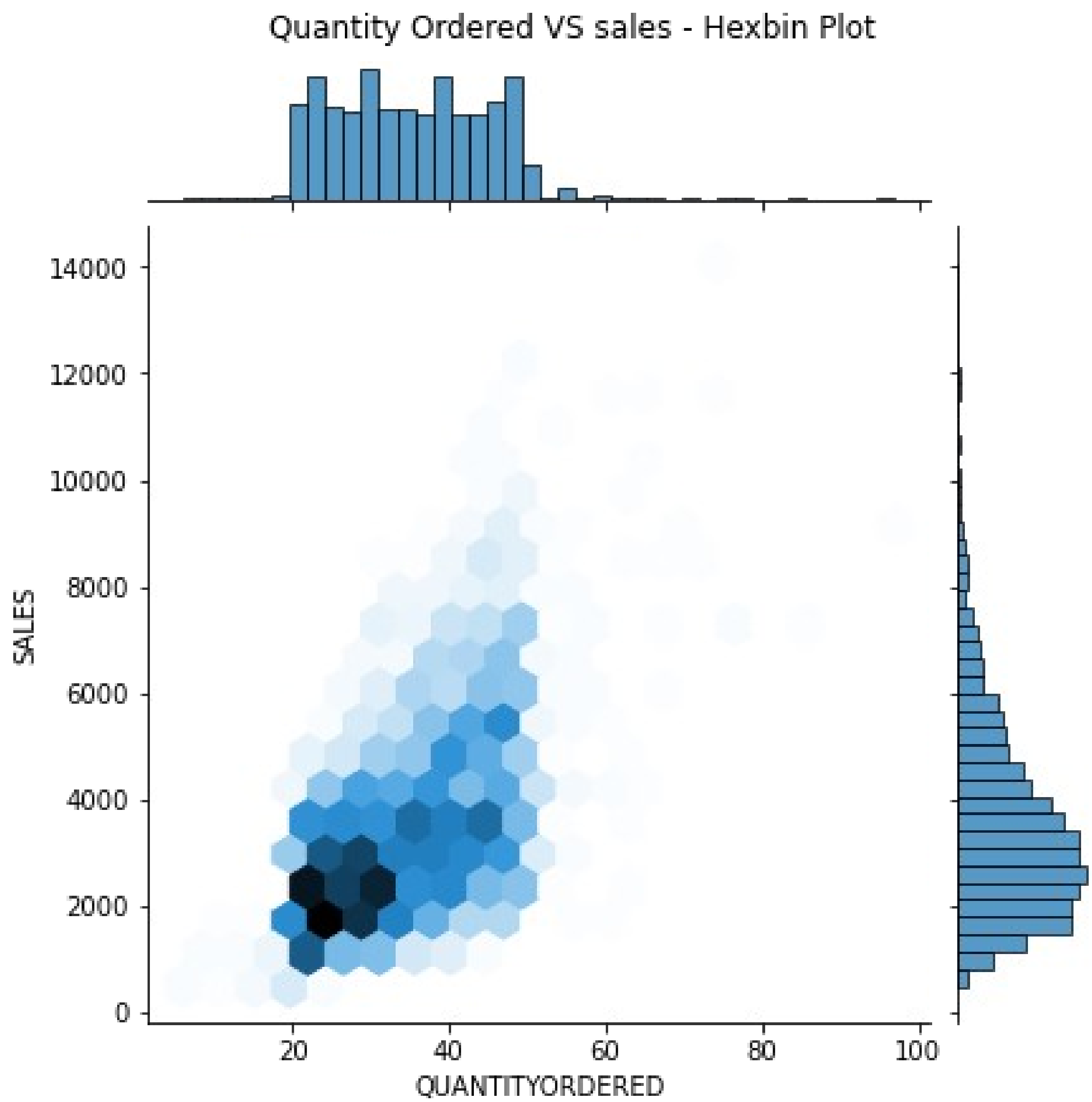
```
#Relationship between Quantity Ordered and Sales:

#Joint Plot (Scatter) for Quantity Ordered VS Sales:
```

```python
plt.figure(figsize=(8,8))
sns.jointplot(x = 'QUANTITYORDERED', y = 'SALES', data = df, kind =
'scatter')
plt.suptitle('Quantity Ordered VS Sales - Joint Plot', y = 1.02)
plt.show()
print('\n')

#Joint Plot for Quantity Ordered VS Sales:
plt.figure(figsize = (8,8))
sns.jointplot(x = 'QUANTITYORDERED', y = 'SALES', data = df, kind =
'hex', gridsize=20)
plt.suptitle('Quantity Ordered VS sales - Hexbin Plot', y = 1.02)
plt.show()
print('\n')
```

```
<Figure size 576x576 with 0 Axes>
```



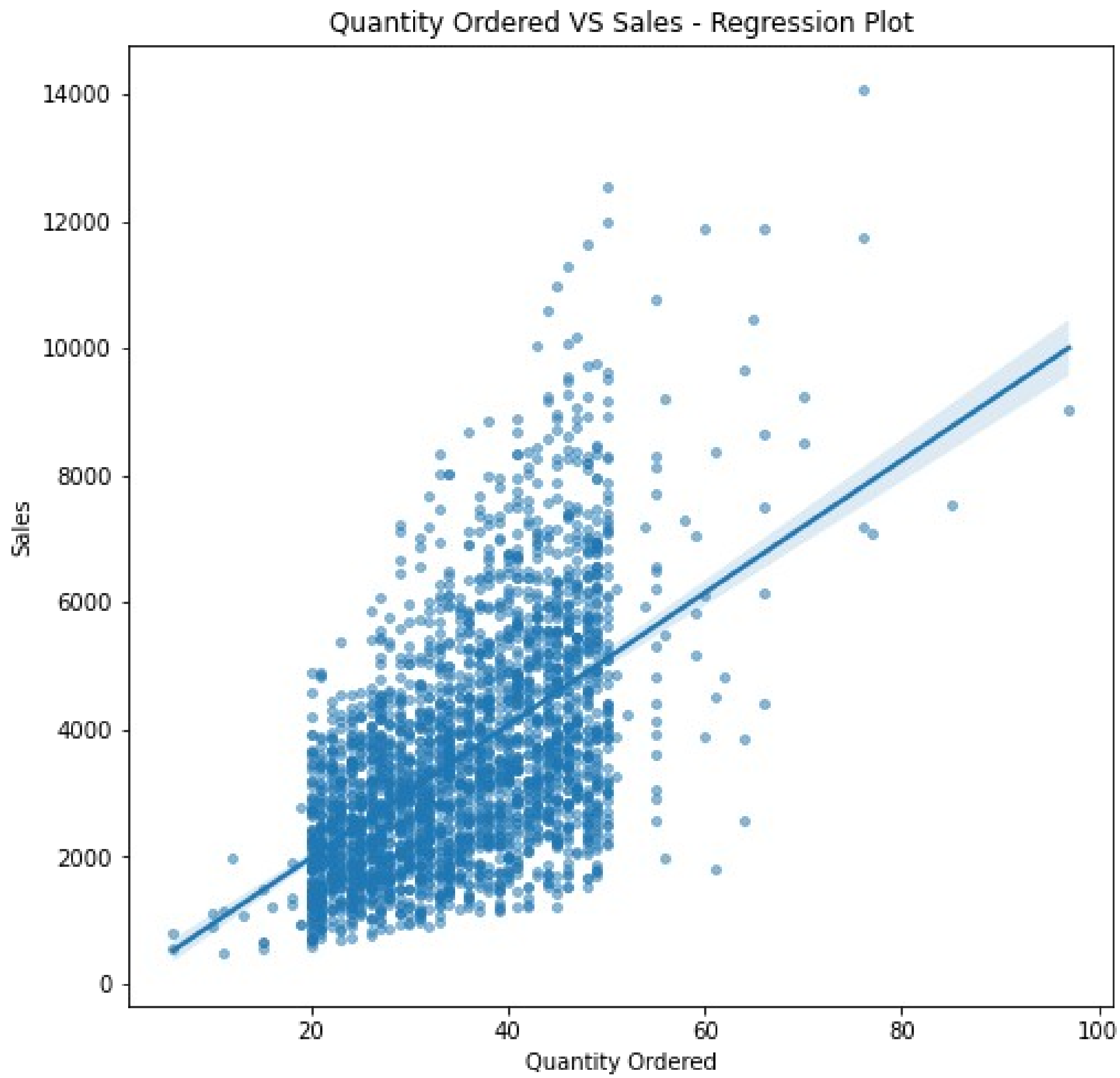

```
<Figure size 576x576 with 0 Axes>
```



This code fits a regression line to the scatter plot depicting the relationship between 'QUANTITYORDERED' and 'SALES' in the DataFrame 'df.'

```
#Fitting the regression line to the Scatter Plot:
plt.figure(figsize = (8,8))
sns.regplot(x = 'QUANTITYORDERED', y = 'SALES', data = df,
scatter_kws={'s': 15, 'alpha': 0.5})
plt.title('Quantity Ordered VS Sales - Regression Plot')
```

```
plt.xlabel('Quantity Ordered')
plt.ylabel('Sales')
plt.show()
```



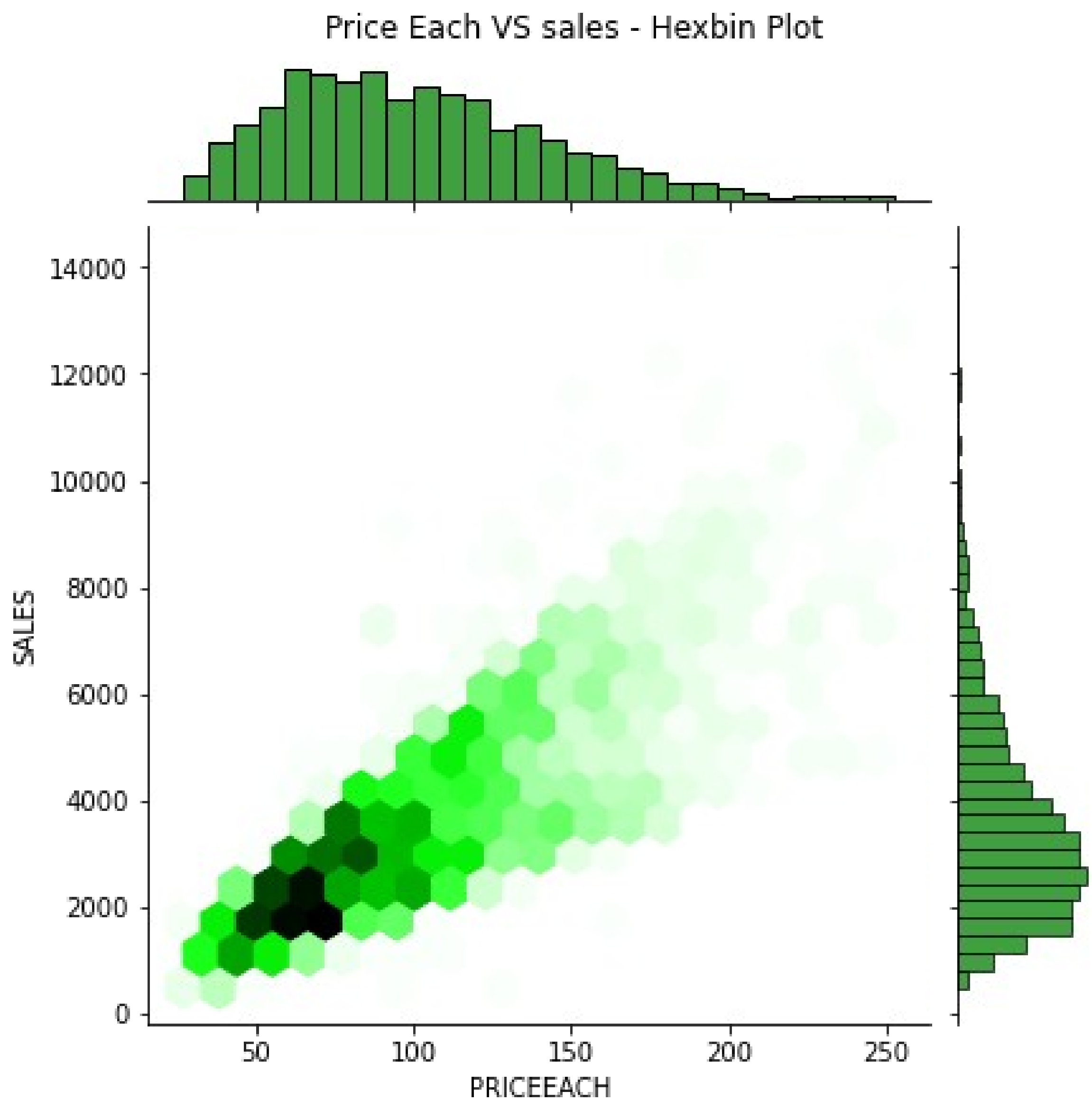
The output presents a scatter plot with a fitted regression line, illustrating the relationship between 'QUANTITYORDERED' and 'SALES' in the DataFrame 'df.' The regression line provides insights into the trend and direction of the association between the two variables.

This code explores the relationship between 'PRICEEACH' and 'SALES' through two visualizations: a scatter plot and a hexbin plot.

```
#Relationship between Price Each and Sales:

#Joint plot (Scatter) for Price Each VS Sales:
plt.figure(figsize=(8,8))
sns.jointplot(x = 'PRICEEACH', y = 'SALES', data = df, kind =
'scatter', color = 'green')
plt.suptitle('Price Each VS Sales - Joint Plot', y = 1.02)
plt.show()
print('\n')

#Joint plot (Hexbin) for Price Each VS Sales:
plt.figure(figsize = (8,8))
sns.jointplot(x = 'PRICEEACH', y = 'SALES', data = df, kind = 'hex',
gridsize=20, color='green')
plt.suptitle('Price Each VS sales - Hexbin Plot', y = 1.02)
plt.show()
print('\n')
```

```
<Figure size 576x576 with 0 Axes>
```


```
<Figure size 576x576 with 0 Axes>
```



This code fits a regression line to the scatter plot, depicting the relationship between 'Price Each' and 'Sales' in the DataFrame 'df.'

```
#Fitting the regression line to the Scatter Plot:
plt.figure(figsize = (8,8))
sns.regplot(x = 'PRICEEACH', y = 'SALES', data = df, color = 'green',
scatter_kws={'s': 15, 'alpha': 0.5})
plt.title('Price Each VS Sales - Regression Plot')
```

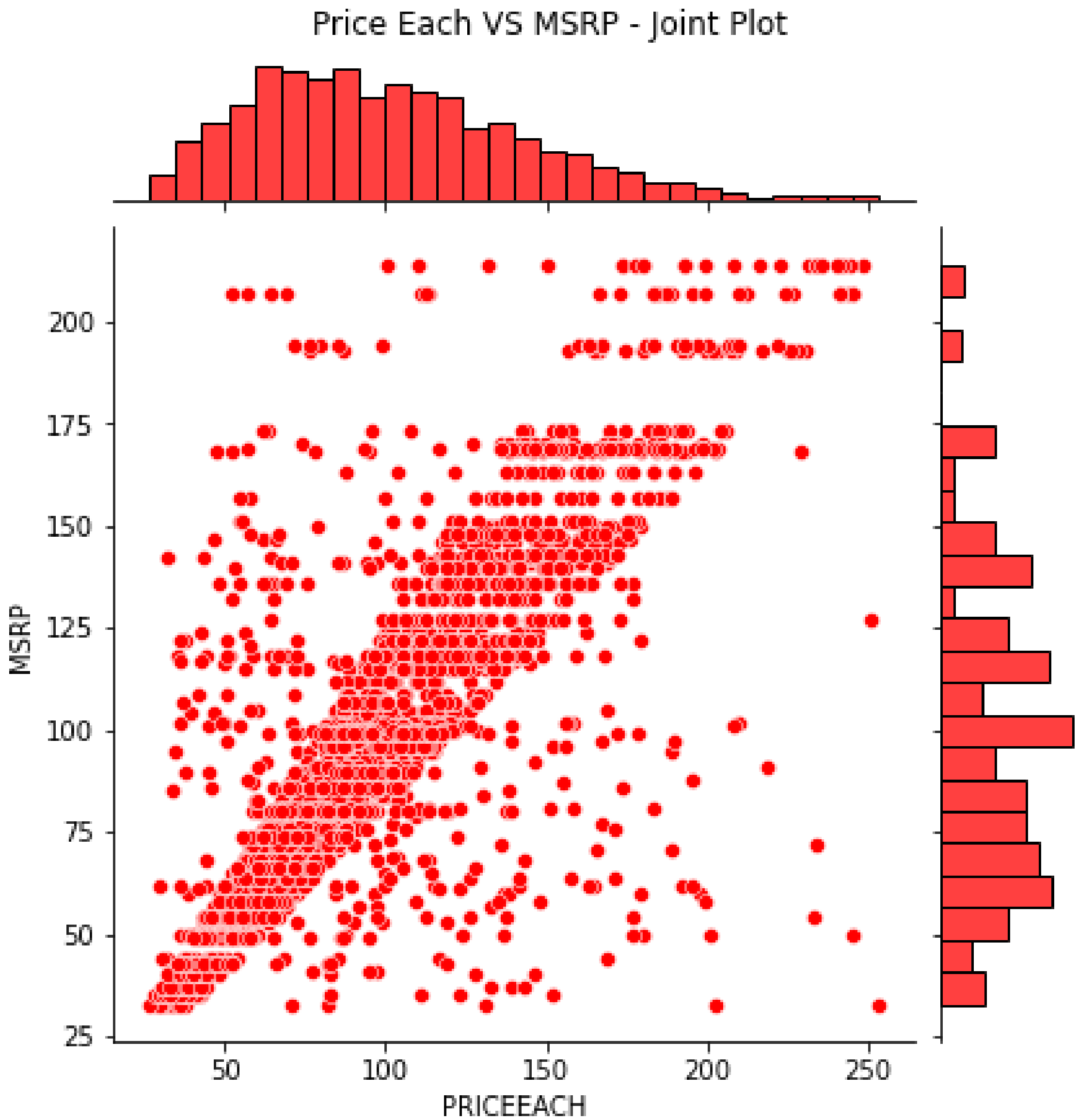
```
plt.xlabel('Price Each')
plt.ylabel('Sales')
plt.show()
```

The output of the above code cell shows that regression line fits the scatter plot, giving insight on the behaviour and trend of the variables.

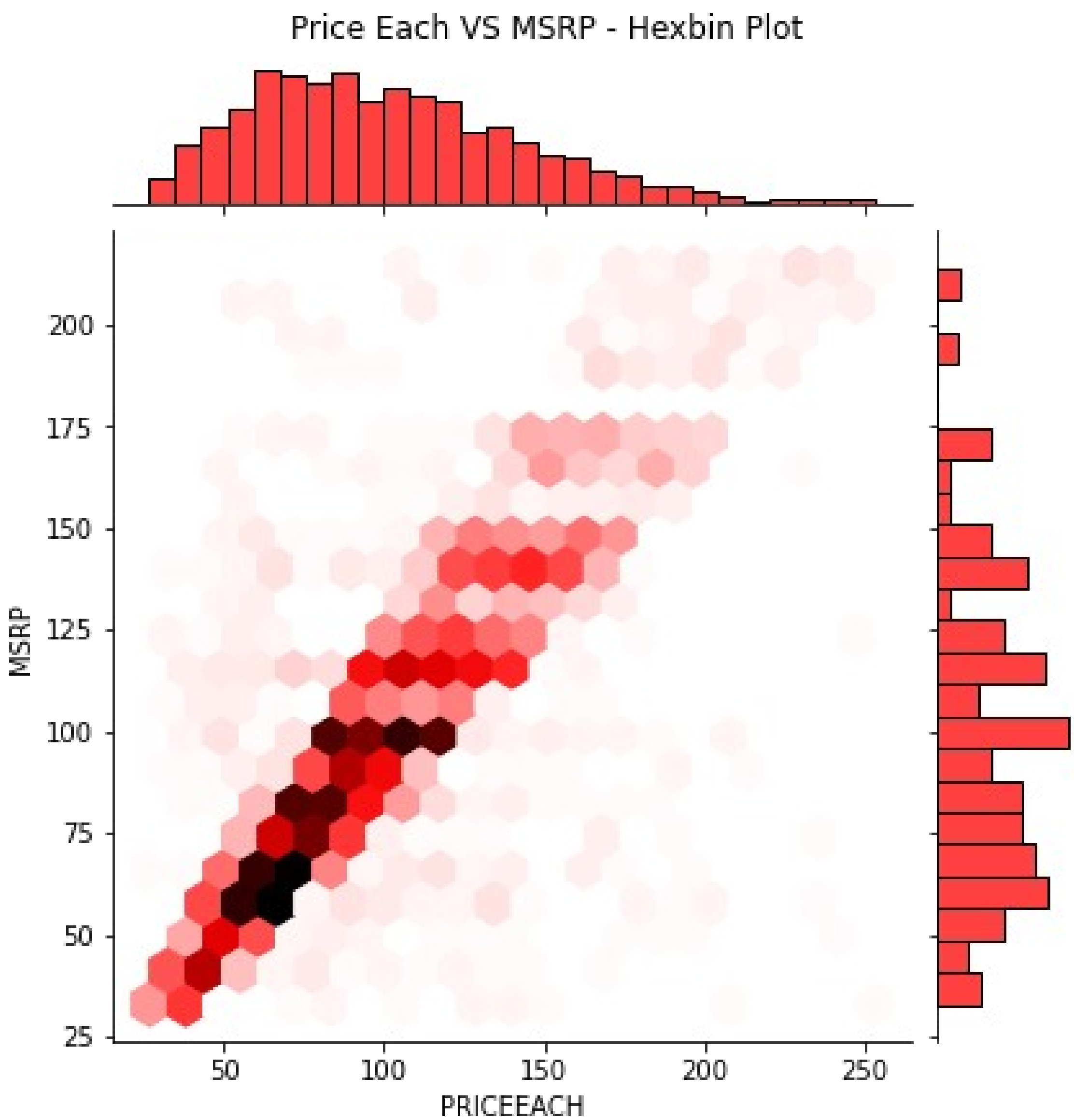
This code explores the relationship between 'PRICEEACH' and 'MSRP' through two visualizations: a scatter plot and a hexbin plot.

```
#Relationship between Price Each and MSRP:

#Joint Plot of Price Each VS MSRP:
plt.figure(figsize=(8,8))
sns.jointplot(x = 'PRICEEACH', y = 'MSRP', data = df, kind =
'scatter', color = 'red')
plt.suptitle('Price Each VS MSRP - Joint Plot', y = 1.02)
plt.show()
print('\n')

#Hexbin Plot of Price Each VS MSRP:
plt.figure(figsize = (8,8))
sns.jointplot(x = 'PRICEEACH', y = 'MSRP', data = df, kind = 'hex',
gridsize=20, color='red')
plt.suptitle('Price Each VS MSRP - Hexbin Plot', y = 1.02)
plt.show()
print('\n')
```

```
<Figure size 576x576 with 0 Axes>
```



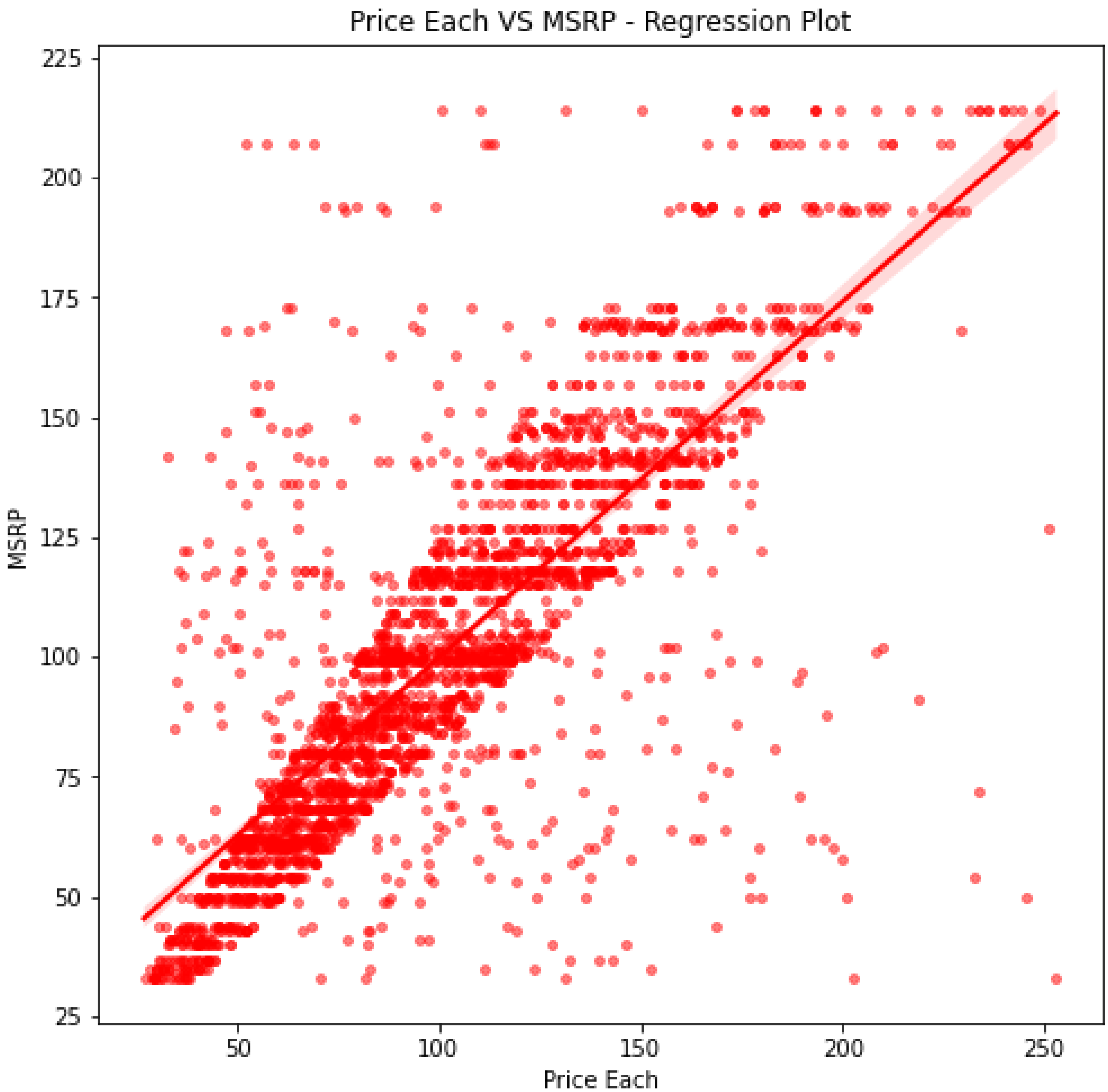

```
<Figure size 576x576 with 0 Axes>
```



This code fits a regression line to the scatter plot, depicting the relationship between 'PRICEEACH' and 'MSRP' in the DataFrame 'df.'

```
#Fitting the Regression line to the Scatter Plot:
plt.figure(figsize = (8,8))
sns.regplot(x = 'PRICEEACH', y = 'MSRP', data = df, color = 'red',
scatter_kws={'s': 15, 'alpha': 0.5})
plt.title('Price Each VS MSRP - Regression Plot')
```

```
plt.xlabel('Price Each')
plt.ylabel('MSRP')
plt.show()
```

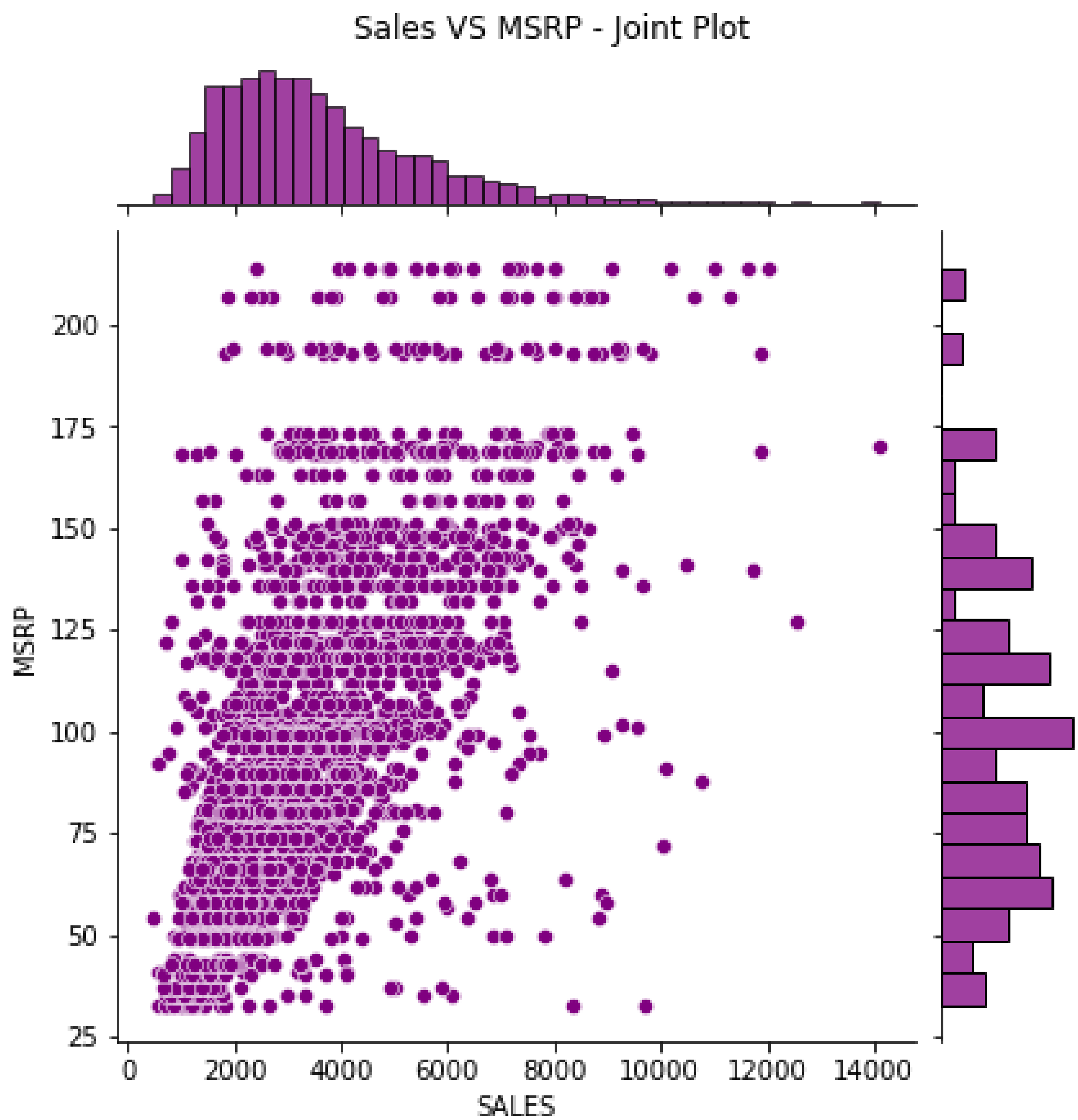


The output showcases a scatter plot with a fitted regression line, illustrating the relationship between 'PRICEEACH' and 'MSRP' in the DataFrame 'df.' The regression line provides insights into the trend and direction of the association between the two variables.

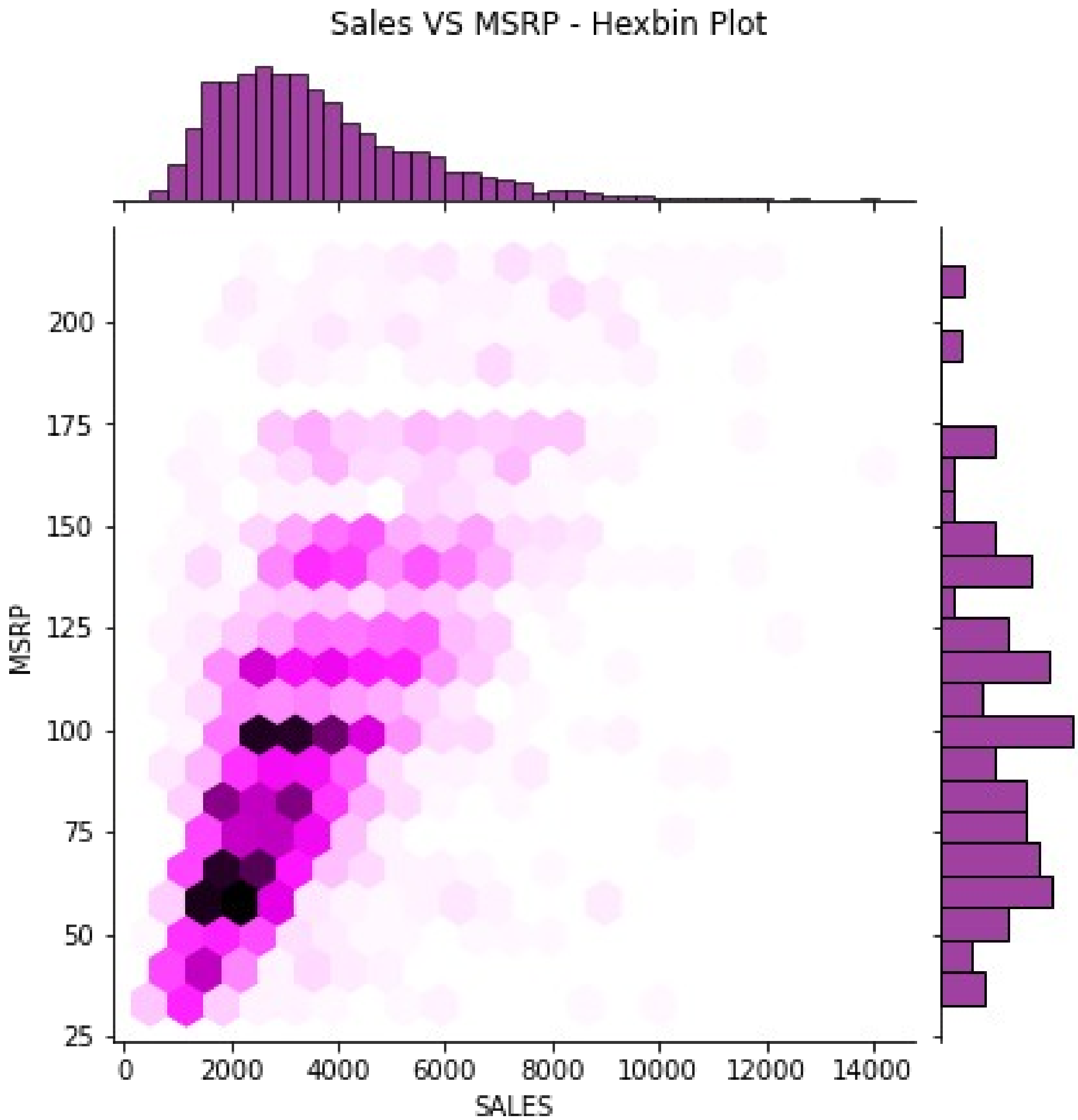
This code explores the relationship between 'SALES' and 'MSRP' through two visualizations: a scatter plot and a hexbin plot.

```
#Relationship between Sales and MSRP:

#Joint Plot (Scatter) for Sales VS MSRP:
plt.figure(figsize=(8,8))
sns.jointplot(x = 'SALES', y = 'MSRP', data = df, kind = 'scatter',
color = 'purple')
plt.suptitle('Sales VS MSRP - Joint Plot', y = 1.02)
plt.show()
print('\n')

#Joint Plot (Hexbin) for Sales VS MSRP:
plt.figure(figsize = (8,8))
sns.jointplot(x = 'SALES', y = 'MSRP', data = df, kind = 'hex',
gridsize=20, color='purple')
plt.suptitle('Sales VS MSRP - Hexbin Plot', y = 1.02)
plt.show()
print('\n')
```

```
<Figure size 576x576 with 0 Axes>
```



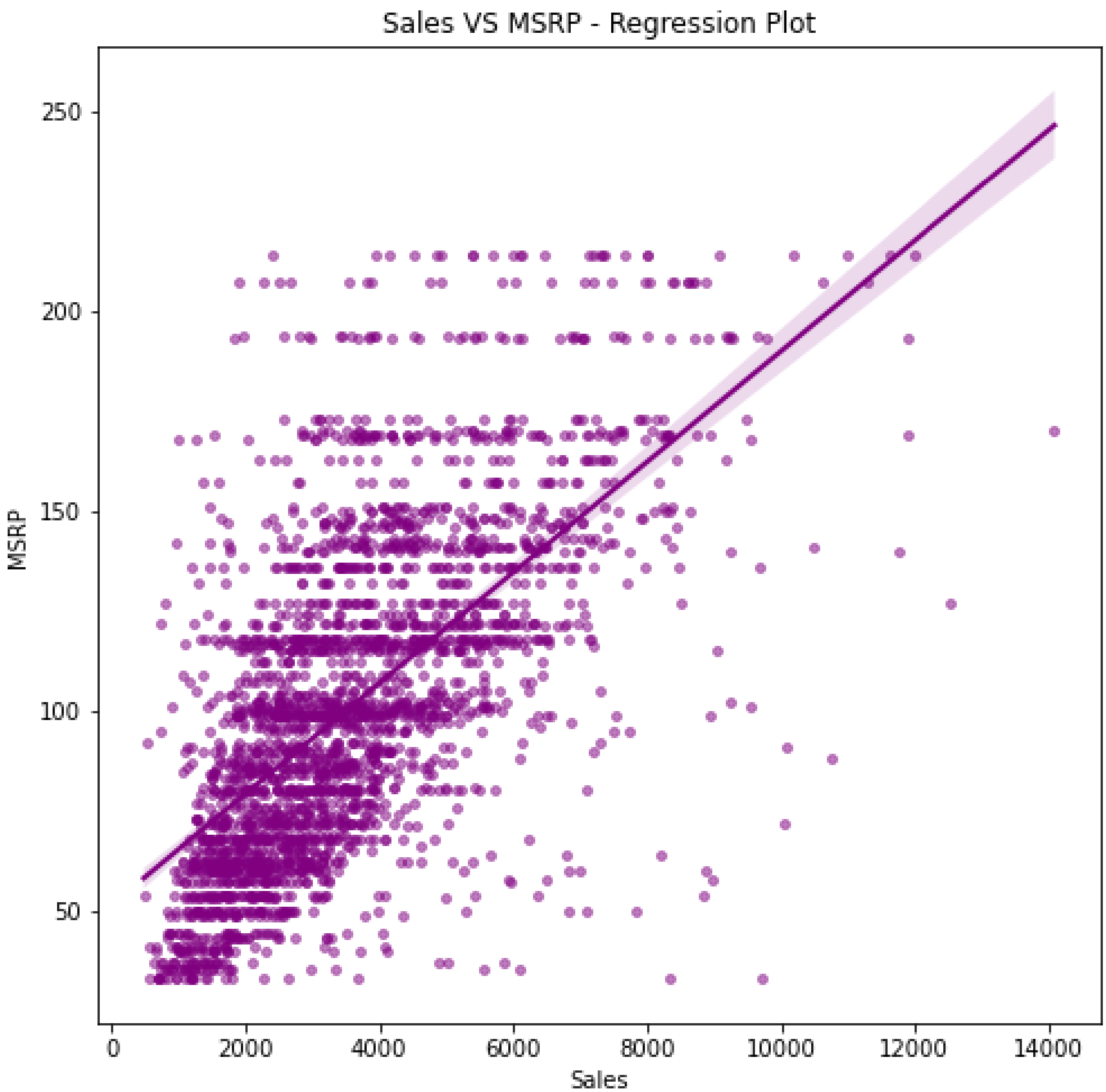

```
<Figure size 576x576 with 0 Axes>
```



This code fits a regression line to the scatter plot, depicting the relationship between 'SALES' and 'MSRP' in the DataFrame 'df.'

```
#Fitting the regression line to the Scatter Plot:
plt.figure(figsize = (8,8))
sns.regplot(x = 'SALES', y = 'MSRP', data = df, color = 'purple',
scatter_kws={'s': 15, 'alpha': 0.5})
plt.title('Sales VS MSRP - Regression Plot')
```

```
plt.xlabel('Sales')
plt.ylabel('MSRP')
plt.show()
```

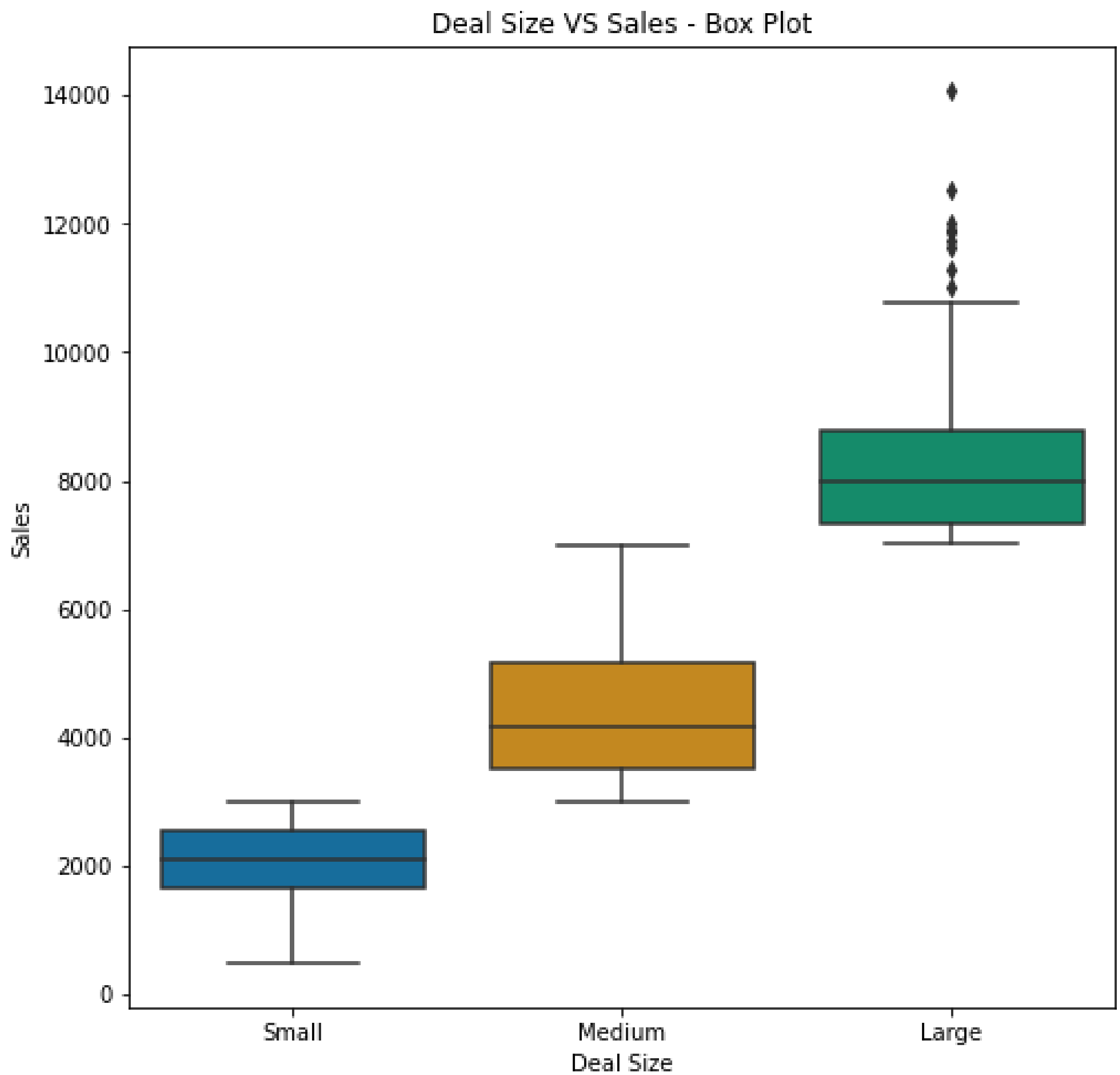


The output showcases a scatter plot with a fitted regression line, illustrating the relationship between 'SALES' and 'MSRP' in the DataFrame 'df.' The regression line provides insights into the trend and direction of the association between the two variables.

This code explores the relationship between 'DEALSIZE' and 'SALES' through Box plot

```
#Relationship between Deal Size and Sales:
print(df['DEALSIZE'].value_counts(), '\n\n')

#Box Plot for Deal Size VS Sales:
plt.figure(figsize =(8,8))
sns.boxplot(x= 'DEALSIZE', y = 'SALES', data = df, palette =
'colorblind')
plt.title('Deal Size VS Sales - Box Plot')
plt.xlabel('Deal Size')
plt.ylabel('Sales')
plt.show()
print('\n')
```

```
Medium    1349
Small     1246
Large      152
Name: DEALSIZE, dtype: int64
```



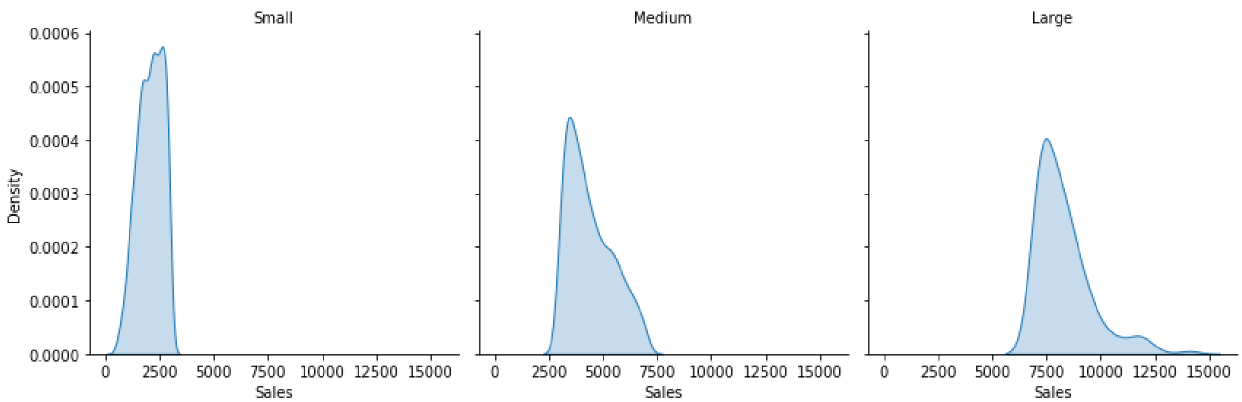
The output of this code block provides a summary of the distribution of sales across different deal sizes, offering insights into the variability and central tendency of sales within each deal size category. The box plot visually represents the quartiles, median, and potential outliers, aiding in the analysis of sales patterns based on deal sizes.

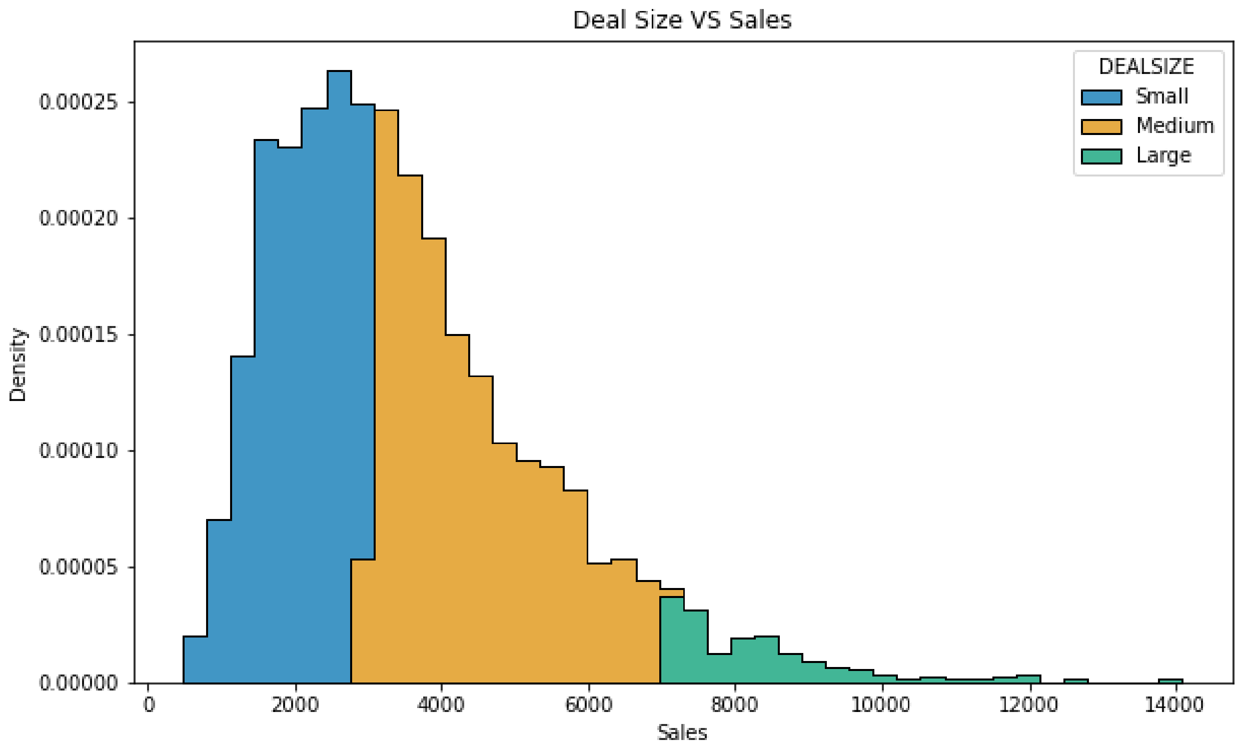
Further exploration of the relationship between 'DEALSIZE' and 'SALES' through KDE plots and Histogram

```
#Individual KDE plot for different Deal Size (Density VS Sales):
x1 = plt.figure(figsize=(8,8))
```

```
x1 = sns.FacetGrid(df, col = 'DEALSIZE', col_wrap = 3, height = 4)
x1.map(sns.kdeplot, 'SALES', fill = True, cmap = 'viridis')
x1.set_titles(col_template = "{col_name}")
x1.set_axis_labels('Sales', 'Density')
plt.show()
print('\n')

#Individual Histogram plot for different Deal Size (Density VS Sales):
plt.figure(figsize = (10,6))
sns.histplot(data = df, x = 'SALES', hue = 'DEALSIZE', multiple =
'stack', palette = 'colorblind', element = 'step', stat = 'density')
plt.title('Deal Size VS Sales')
plt.xlabel('Sales')
plt.ylabel('Density')
plt.show()
```

```
<Figure size 576x576 with 0 Axes>
```





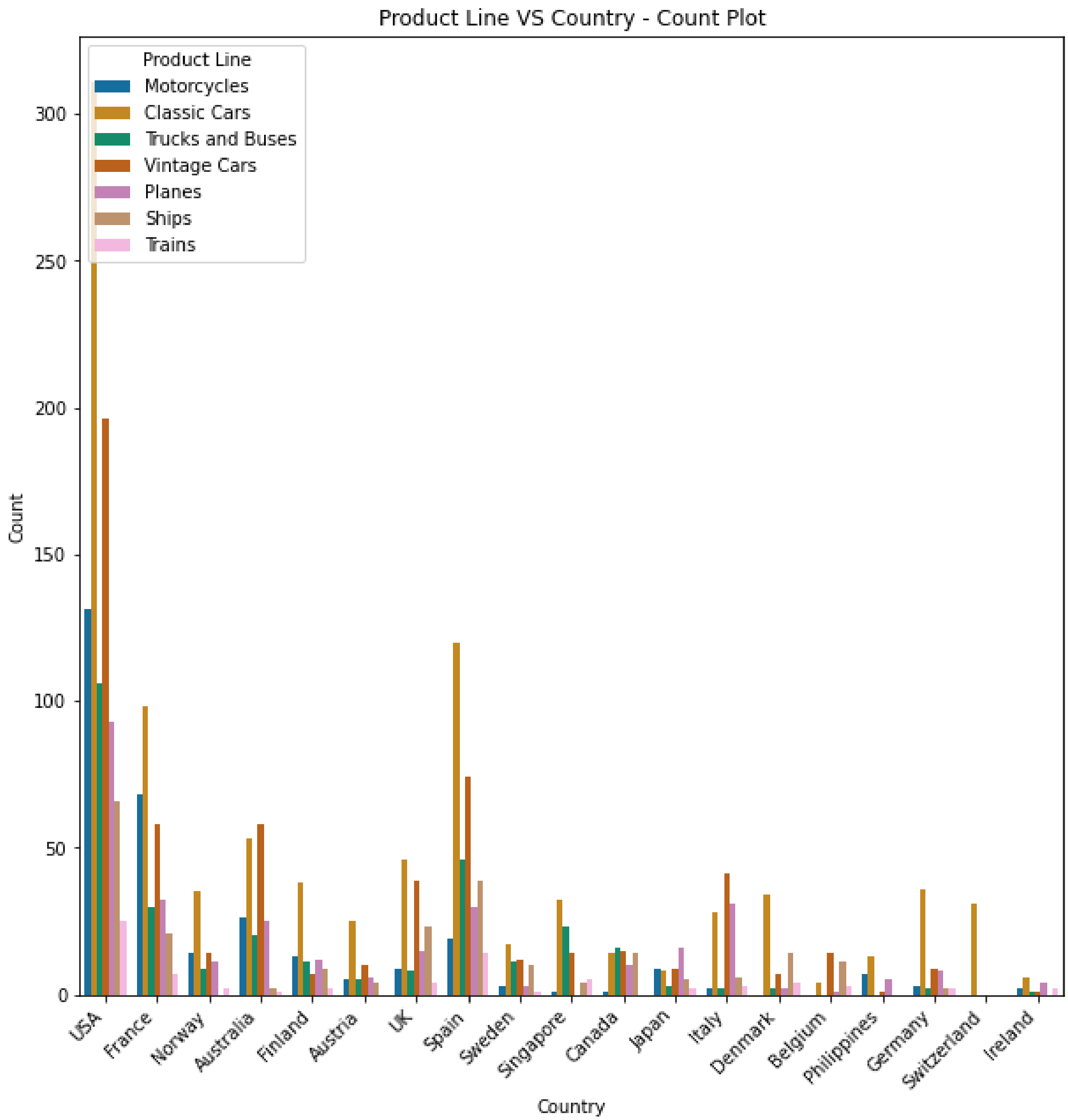
The output of this code block provides a detailed view of the distribution of sales density for different deal sizes. The individual KDE (Kernel Density Estimation) plots show the estimated probability density of sales values, while the stacked histogram plot provides a comparative visualization of sales density across various deal sizes. These visualizations help in understanding the variation in sales patterns within each deal size category and identifying potential trends or differences.

Exploring the distribution of 'SALES' and 'PRODUCTLINE' on the basis of geography (Countries) using Count plot, Heatmap and KDE plot.

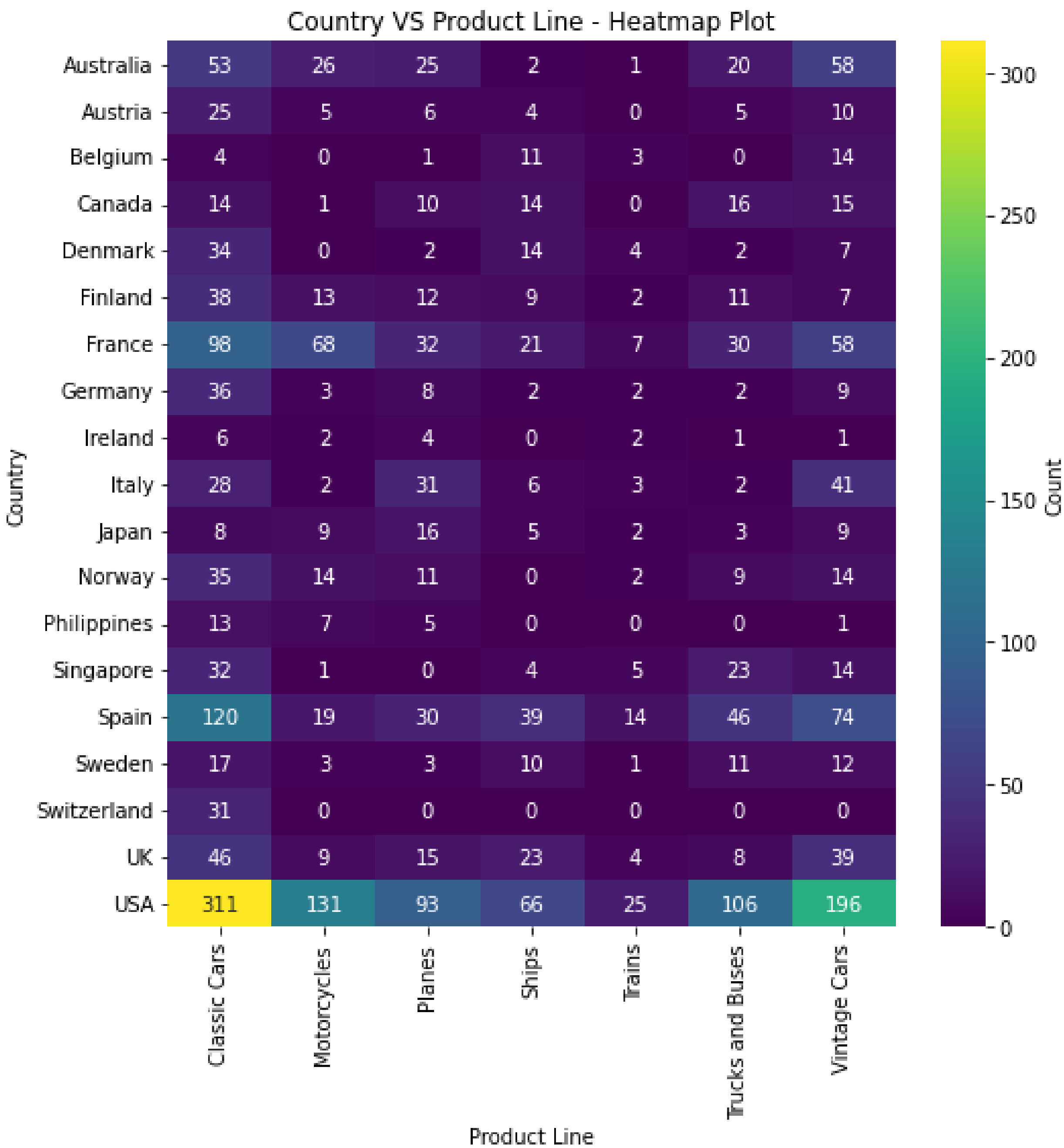
```
#Geographical distribution of sales and productline:


#Table for each Product line count in various Countries:
product_line_counts = df.groupby(['COUNTRY',
'PRODUCTLINE']).size().reset_index(name='COUNT')
product_line_counts = product_line_counts.pivot(index='COUNTRY',
columns='PRODUCTLINE', values='COUNT').fillna(0).astype(int)
print(product_line_counts, '\n\n')

#Count plot for Product line VS Country:
plt.figure(figsize=(10,10))
sns.countplot(x = 'COUNTRY', hue = 'PRODUCTLINE', data = df, palette =
'colorblind')
plt.title('Product Line VS Country - Count Plot')
```

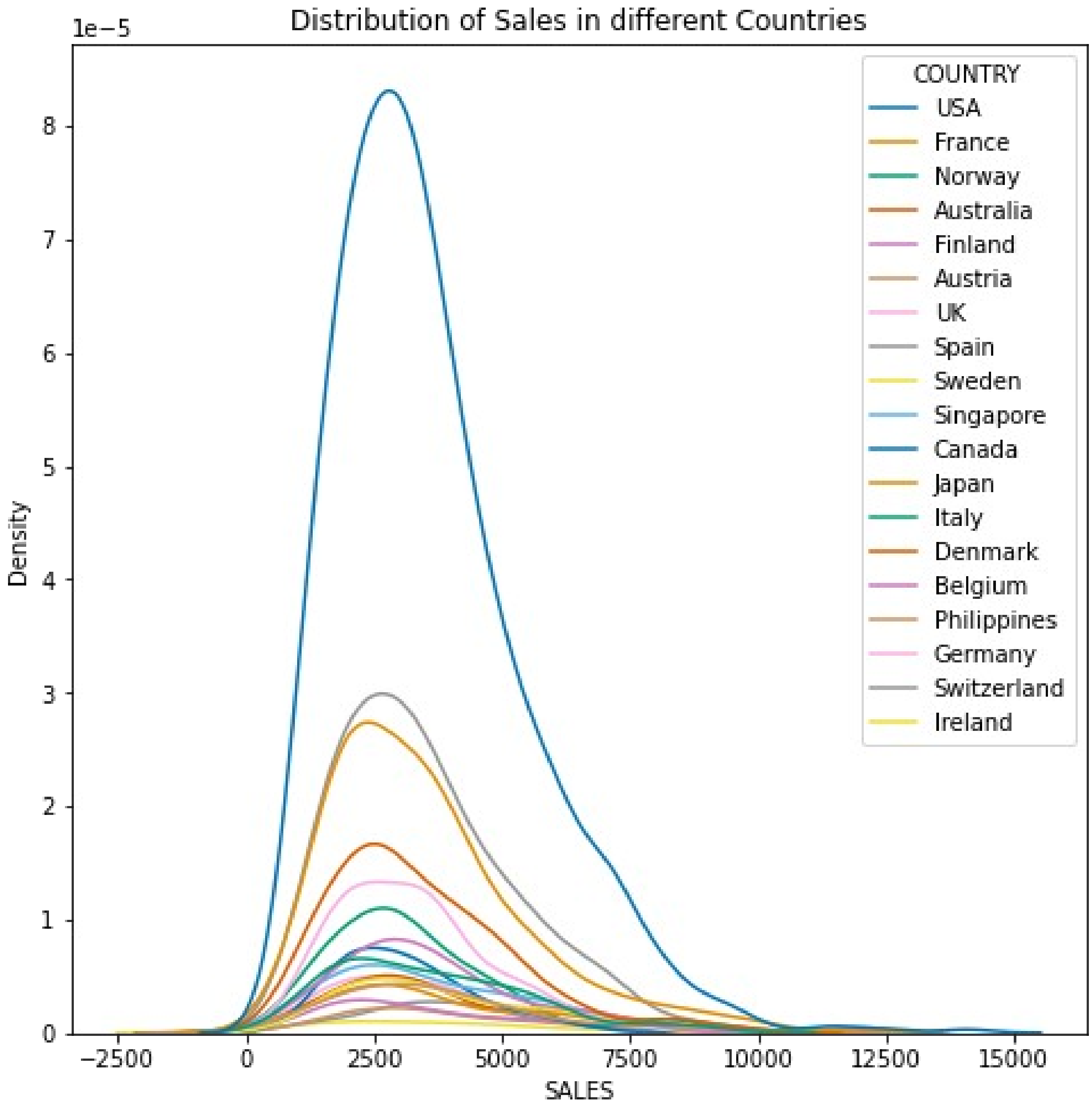
```
plt.xlabel('Country')
plt.ylabel('Count')
plt.xticks(rotation=45, ha = 'right')
plt.legend(title = 'Product Line')
plt.show()
print('\n')

#Heatmap for Country VS Product line:
plt.figure(figsize=(8,8))
crosstab = pd.crosstab(df['COUNTRY'], df['PRODUCTLINE'])
sns.heatmap(crosstab, cmap = 'viridis', annot = True, fmt = 'd',
cbar_kws = {'label': 'Count'})
plt.title('Country VS Product Line - Heatmap Plot')
plt.xlabel('Product Line')
plt.ylabel('Country')
plt.show()
print('\n')

#Distributions of Products based on Various Countries:
plt.figure(figsize=(8,8))
sns.kdeplot(x = 'SALES', data = df, hue = 'COUNTRY', palette =
'colorblind')
plt.title('Distribution of Sales in different Countries')
plt.show()
print('\n')

PRODUCTLINE  Classic Cars  Motorcycles  Planes  Ships  Trains  \
COUNTRY
Australia              53           26      25      2       1
Austria                25            5       6      4       0
Belgium                 4            0       1     11       3
Canada                 14            1      10     14       0
Denmark                34            0       2     14       4
Finland                38           13      12      9       2
France                 98           68      32     21       7
Germany                36            3       8      2       2
Ireland                 6            2       4      0       2
Italy                  28            2      31      6       3
Japan                   8            9      16      5       2
Norway                 35           14      11      0       2
Philippines            13            7       5      0       0
Singapore              32            1       0      4       5
Spain                 120           19      30     39      14
Sweden                 17            3       3     10       1
Switzerland            31            0       0      0       0
UK                     46            9      15     23       4
USA                   311          131      93     66      25

PRODUCTLINE  Trucks and Buses  Vintage Cars
COUNTRY
```

Australia	20	58
Austria	5	10
Belgium	0	14
Canada	16	15
Denmark	2	7
Finland	11	7
France	30	58
Germany	2	9
Ireland	1	1
Italy	2	41
Japan	3	9
Norway	9	14
Philippines	0	1
Singapore	23	14
Spain	46	74
Sweden	11	12
Switzerland	0	0
UK	8	39
USA	106	196



Product Line VS Country - Count Plot
Product Line
Motorcycles
Classic Cars
Trucks and Buses
Vintage Cars
Planes
Ships
Trains
Count
0
50
100
150
200
250
300
USA
France
Norway
Australia
Finland
Austria
UK
Spain
Sweden
Singapore
Canada
Japan
Italy
Denmark
Belgium
Philippines
Germany
Switzerland
Ireland
Country


Country VS Product Line - Heatmap Plot
Country
Australia
Austria
Belgium
Canada
Denmark
Finland
France
Germany
Ireland
Italy
Japan
Norway
Philippines
Singapore
Spain
Sweden
Switzerland
UK
USA
53 26 25 2 1 20 58
25 5 6 4 0 5 10
4 0 1 11 3 0 14
14 1 10 14 0 16 15
34 0 2 14 4 2 7
38 13 12 9 2 11 7
98 68 32 21 7 30 58
36 3 8 2 2 2 9
6 2 4 0 2 1 1
28 2 31 6 3 2 41
8 9 16 5 2 3 9
35 14 11 0 2 9 14
13 7 5 0 0 0 1
32 1 0 4 5 23 14
120 19 30 39 14 46 74
17 3 3 10 1 11 12
31 0 0 0 0 0 0
46 9 15 23 4 8 39
311 131 93 66 25 106 196
Classic Cars
Motorcycles
Planes
Ships
Trains
Trucks and Buses
Vintage Cars
Product Line
Count
300
250
200
150
100
50
0


Distribution of Sales in different Countries
1e−5
Density
SALES
COUNTRY
USA
France
Norway
Australia
Finland
Austria
UK
Spain
Sweden
Singapore
Canada
Japan
Italy
Denmark
Belgium
Philippines
Germany
Switzerland
Ireland

Here, the recency, frequency and monetary value for each customer will be calculated for RFM analysis using relevant information [2]

```
# Convert 'ORDERDATE' to datetime format
df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])

# Calculating Recency, Frequency, and Monetary Value
current_date = df['ORDERDATE'].max()

rfm_df = df.groupby('CUSTOMERNAME').agg({
    'ORDERDATE': lambda x: (current_date - x.max()).days,  # Recency
    'ORDERNUMBER': 'count',  # Frequency
    'SALES': 'sum'  # Monetary Value
}).reset_index()


rfm_df.columns = ['CUSTOMERNAME', 'Recency', 'Frequency',
'MonetaryValue']

# RFM DataFrame
print(rfm_df.head(50))
```

```
                          CUSTOMERNAME  Recency  Frequency
MonetaryValue
0                       AV Stores, Co.      196         51
157807.81
1                        Alpha Cognac       64         20
70488.44
2                   Amica Models & Co.      265         26
94117.26
3              Anna's Decorations, Ltd       83         46
153996.13
4                    Atelier graphique      188          7
24179.96
5         Australian Collectables, Ltd       22         23
64591.46
6           Australian Collectors, Co.      184         55
200995.41
7          Australian Gift Network, Co      119         15
59469.12
8                   Auto Assoc. & Cie.      233         18
64834.32
9                     Auto Canal Petit       54         27
93170.66
10             Auto-Moto Classics Inc.      180          8
26479.26
```

```
11                   Baane Mini Imports      208         32
116599.19
12  Bavarian Collectables Imports, Co.      259         14
34993.92
13                 Blauer See Auto, Co.      208         22
85171.59
14                    Boards & Toys Co.      113          3
9129.35
15                          CAF Imports      439         13
49642.05
16         Cambridge Collectables Co.      389         11
36163.62
17      Canadian Gift Exchange Network      222         22
75238.92
18             Classic Gift Ideas, Inc      230         21
67506.97
19                 Classic Legends Inc.      192         20
77795.20
20             Clover Collections, Co.      258         16
57756.43
21        Collectable Mini Designs Co.      460         25
87489.23
22         Collectables For Less Inc.      132         24
81577.98
23         Corrida Auto Replicas, Ltd      212         32
120615.28
24                    Cruz & Sons Co.      197         26
94015.73
25            Daedalus Designs Imports      465         20
69052.41
26            Danish Wholesale Imports       46         36
145041.60
27               Diecast Classics Inc.        1         31
122138.14
28                Diecast Collectables      401         18
70859.78
29      Double Decker Gift Stores, Ltd      495         12
36019.04
30             Dragon Souveniers, Ltd.       90         43
172989.68
31                  Enaco Distributors      189         23
78411.86
32               Euro Shopping Channel        0        259
912294.11
33                    FunGiftIdeas.com       89         26
98923.73
34                    Gift Depot Inc.       26         25
101894.79
35                   Gift Ideas Corp.      179         19
```

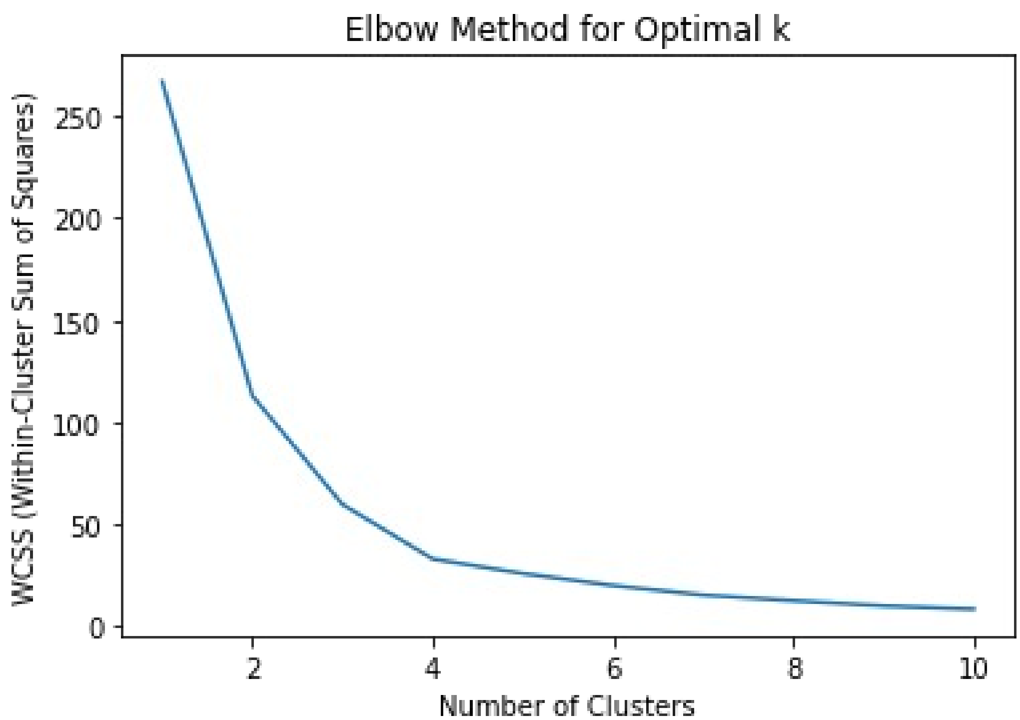
```
57294.42
36                  Gifts4AllAges.com        25          26
83209.88
37                   Handji Gifts& Co        38          36
115498.73
38               Heintze Collectables       222          27
100595.55
39                       Herkku Gifts       271          29
111640.28
40         Iberia Gift Imports, Corp.       238          15
54723.62
41                L'ordine Souveniers        21          39
142601.33
42          La Corne D'abondance, Co.       193          23
97203.68
43                  La Rochelle Gifts         0          53
180124.90
44                  Land of Toys Inc.       198          49
164069.44
45                    Lyon Souveniers        75          20
78570.34
46               Marseille Mini Autos       146          25
74936.14
47               Marta's Replicas Co.       231          27
103080.38
48                    Microscale Inc.       210          10
33144.93
49                    Mini Auto Werke        82          15
52263.90
```

The rfm_df is created to calculate the rfm values and assign them to the respective customer.

Once the RFM for all customers has been calculated, K will be determined using the Elbow method to use K-MEANS method later on

```
# RFM features
rfm_features = rfm_df[['Recency', 'Frequency', 'MonetaryValue']]


scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm_features)

# determining k using elbow method
```

```python
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(rfm_scaled)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.show()
```



The output displays an elbow method plot with an L-shaped pattern, indicating distinct edges. The optimal cluster count (k) for subsequent KMeans clustering on standardized RFM features can be inferred from the point where the rate of within-cluster sum of squares reduction diminishes.

Once k has been determined, K-MEANS method will be used to cluster our customers in different segments [2]

```python
#based on the Elbow method
num_clusters = 4
```

```
#k-means clustering
kmeans = KMeans(n_clusters=num_clusters, init='k-means++',
random_state=42)
rfm_df['Cluster'] = kmeans.fit_predict(rfm_scaled)

#clustered RFM DataFrame
print(rfm_df.head(50))
```

```
                          CUSTOMERNAME  Recency  Frequency
MonetaryValue  \
0                    AV Stores, Co.      196         51
157807.81
1                      Alpha Cognac       64         20
70488.44
2                Amica Models & Co.      265         26
94117.26
3           Anna's Decorations, Ltd       83         46
153996.13
4                 Atelier graphique      188          7
24179.96
5        Australian Collectables, Ltd       22         23
64591.46
6          Australian Collectors, Co.      184         55
200995.41
7         Australian Gift Network, Co      119         15
59469.12
8               Auto Assoc. & Cie.      233         18
64834.32
9                   Auto Canal Petit       54         27
93170.66
10            Auto-Moto Classics Inc.      180          8
26479.26
11                Baane Mini Imports      208         32
116599.19
12  Bavarian Collectables Imports, Co.      259         14
34993.92
13              Blauer See Auto, Co.      208         22
85171.59
14                Boards & Toys Co.      113          3
9129.35
15                      CAF Imports      439         13
49642.05
16          Cambridge Collectables Co.      389         11
36163.62
17      Canadian Gift Exchange Network      222         22
75238.92
18            Classic Gift Ideas, Inc      230         21
67506.97
19              Classic Legends Inc.      192         20
```

```
77795.20
20               Clover Collections, Co.       258           16
57756.43
21         Collectable Mini Designs Co.       460           25
87489.23
22           Collectables For Less Inc.       132           24
81577.98
23           Corrida Auto Replicas, Ltd       212           32
120615.28
24                       Cruz & Sons Co.       197           26
94015.73
25             Daedalus Designs Imports       465           20
69052.41
26             Danish Wholesale Imports        46           36
145041.60
27                Diecast Classics Inc.         1           31
122138.14
28                 Diecast Collectables       401           18
70859.78
29      Double Decker Gift Stores, Ltd       495           12
36019.04
30              Dragon Souveniers, Ltd.        90           43
172989.68
31                   Enaco Distributors       189           23
78411.86
32                 Euro Shopping Channel         0          259
912294.11
33                      FunGiftIdeas.com        89           26
98923.73
34                       Gift Depot Inc.        26           25
101894.79
35                     Gift Ideas Corp.       179           19
57294.42
36                      Gifts4AllAges.com        25           26
83209.88
37                      Handji Gifts& Co        38           36
115498.73
38                 Heintze Collectables       222           27
100595.55
39                          Herkku Gifts       271           29
111640.28
40           Iberia Gift Imports, Corp.       238           15
54723.62
41                 L'ordine Souveniers        21           39
142601.33
42          La Corne D'abondance, Co.       193           23
97203.68
43                     La Rochelle Gifts         0           53
180124.90
```

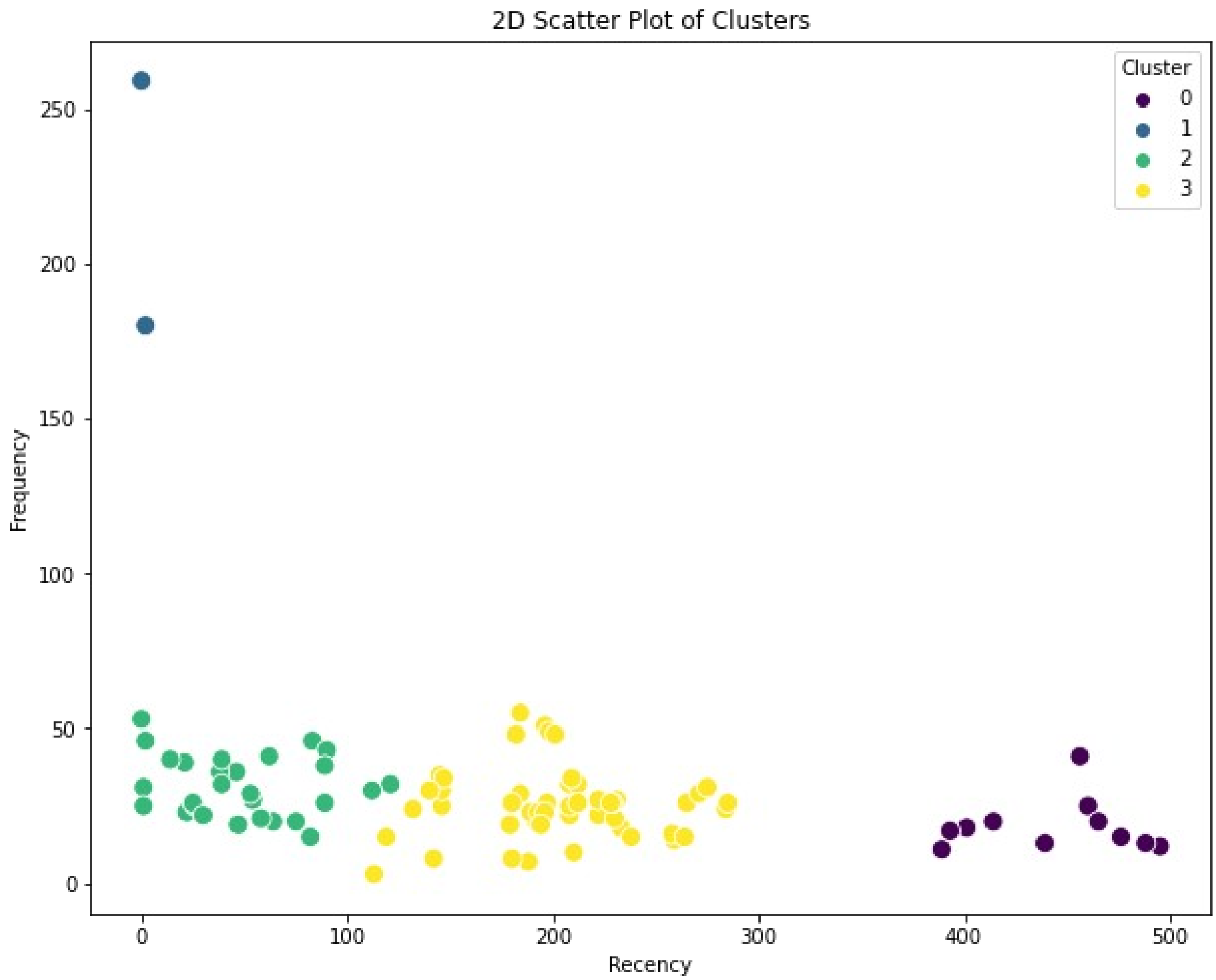
```
44                    Land of Toys Inc.     198         49
164069.44
45                      Lyon Souveniers      75         20
78570.34
46                Marseille Mini Autos      146         25
74936.14
47                Marta's Replicas Co.      231         27
103080.38
48                      Microscale Inc.     210         10
33144.93
49                      Mini Auto Werke      82         15
52263.90

    Cluster
0         3
1         2
2         3
3         2
4         3
5         2
6         3
7         3
8         3
9         2
10        3
11        3
12        3
13        3
14        3
15        0
16        0
17        3
18        3
19        3
20        3
21        0
22        3
23        3
24        3
25        0
26        2
27        2
28        0
29        0
30        2
31        3
32        1
33        2
34        2
```

```
35         3
36         2
37         2
38         3
39         3
40         3
41         2
42         3
43         2
44         3
45         2
46         3
47         3
48         3
49         2
```

The output of this cell shows the RFM DataFrame with an additional 'Cluster' column, indicating the assigned cluster for each observation based on the KMeans clustering with four clusters. The clustering results provide insights into grouping similar customer behaviors within the dataset.

Here the clusters will be visualised using a scatter plot for better understanding

```
#2D scatter plot
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Recency', y='Frequency', hue='Cluster',
data=rfm_df, palette='viridis', s=100)


plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.title('2D Scatter Plot of Clusters')


plt.show()
```



The 2D scatter plot illustrates distinct clusters based on 'Recency' and 'Frequency,' with one cluster containing only two values, positioned above another cluster. The rest of the clusters are clearly defined in the graph, showcasing the clustering structure identified in the DataFrame 'rfm_df.'

Here the RFM score will be evaluated by first standardizing the r, f, and m

```
rfm_df['R_Score'] = pd.qcut(rfm_df['Recency'], q=5, labels=[5, 4, 3,
2, 1])
rfm_df['F_Score'] = pd.qcut(rfm_df['Frequency'], q=5, labels=[1, 2, 3,
4, 5])
rfm_df['M_Score'] = pd.qcut(rfm_df['MonetaryValue'], q=5, labels=[1,
2, 3, 4, 5])

# Combine the scores to create the RFM Score
rfm_df['RFM_Score'] = rfm_df['R_Score'].astype(str) +
```

```
rfm_df['F_Score'].astype(str) + rfm_df['M_Score'].astype(str)
print(rfm_df.head(80))
```

```
                  CUSTOMERNAME  Recency  Frequency  MonetaryValue
Cluster  \
0                AV Stores, Co.      196         51      157807.81
3
1                 Alpha Cognac        64         20       70488.44
2
2           Amica Models & Co.      265         26       94117.26
3
3      Anna's Decorations, Ltd        83         46      153996.13
2
4            Atelier graphique      188          7       24179.96
3
..                         ...      ...        ...            ...
...
75             Super Scale Inc.      393         17       79472.07
0
76         Technics Stores Inc.      147         34      120783.07
3
77     Tekni Collectables Inc.        58         21       83228.19
2
78   The Sharp Gifts Warehouse        39         40      160010.27
2
79     Tokyo Collectables, Ltd        39         32      120562.74
2

   R_Score F_Score M_Score RFM_Score
0        3       5       5       355
1        4       2       2       422
2        1       3       3       133
3        4       5       5       455
4        3       1       1       311
..     ...     ...     ...       ...
75       1       1       2       112
76       3       4       4       344
77       4       2       3       423
78       5       5       5       555
79       5       4       4       544

[80 rows x 9 columns]
```

The output showcases the RFM DataFrame augmented with individual scores for Recency ('R_Score'), Frequency ('F_Score'), MonetaryValue ('M_Score'), and a combined RFM Score ('RFM_Score'). These scores categorize customers based on their transaction recency, frequency, and monetary value, facilitating further segmentation and analysis.

To further apply predictive models to our data using the RFM score and other relevant features, some features will be dropped to increase the overall computation of our model.

```
columns_to_exclude = ['CUSTOMERNAME']

# Extract the 'CUSTOMERNAME' column and drop all other object-type
columns
df1 = df[columns_to_exclude +
df.select_dtypes(exclude=['object']).columns.tolist()]
df1.drop('ORDERDATE', axis=1, inplace=True)
print(df1.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2747 entries, 0 to 2746
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   CUSTOMERNAME         2747 non-null   object
 1   ORDERNUMBER          2747 non-null   int64
 2   QUANTITYORDERED      2747 non-null   int64
 3   PRICEEACH            2747 non-null   float64
 4   ORDERLINENUMBER      2747 non-null   int64
 5   SALES                2747 non-null   float64
 6   MSRP                 2747 non-null   int64
 7   DAYS_SINCE_LASTORDER 2747 non-null   int64
dtypes: float64(2), int64(5), object(1)
memory usage: 171.8+ KB
None
```

The dataframe has been excluded of all data types other than int and float, the segmenting data and sales data will be merged for modelling.

```
merged_df = pd.merge(df1, rfm_df, on='CUSTOMERNAME', how='inner')
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2747 entries, 0 to 2746
Data columns (total 16 columns):
```

```
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   CUSTOMERNAME          2747 non-null   object
 1   ORDERNUMBER           2747 non-null   int64
 2   QUANTITYORDERED       2747 non-null   int64
 3   PRICEEACH             2747 non-null   float64
 4   ORDERLINENUMBER       2747 non-null   int64
 5   SALES                 2747 non-null   float64
 6   MSRP                  2747 non-null   int64
 7   DAYS_SINCE_LASTORDER  2747 non-null   int64
 8   Recency               2747 non-null   int64
 9   Frequency             2747 non-null   int64
 10  MonetaryValue         2747 non-null   float64
 11  Cluster               2747 non-null   int32
 12  R_Score               2747 non-null   category
 13  F_Score               2747 non-null   category
 14  M_Score               2747 non-null   category
 15  RFM_Score             2747 non-null   object
dtypes: category(3), float64(3), int32(1), int64(7), object(2)
memory usage: 298.4+ KB

merged_df.head()

        CUSTOMERNAME  ORDERNUMBER  QUANTITYORDERED  PRICEEACH  \
0  Land of Toys Inc.        10107               30      95.70
1  Land of Toys Inc.        10329               42     104.67
2  Land of Toys Inc.        10107               39      99.91
3  Land of Toys Inc.        10329               20     158.80
4  Land of Toys Inc.        10107               27     224.65

   ORDERLINENUMBER    SALES  MSRP  DAYS_SINCE_LASTORDER  Recency
Frequency  \
0                2  2871.00    95                   828      198
49
1                1  4396.14    95                   199      198
49
2                5  3896.49   118                   828      198
49
3                2  3176.00   118                   199      198
49
4                4  6065.55   193                   828      198
49

   MonetaryValue  Cluster R_Score F_Score M_Score RFM_Score
0      164069.44        3       2       5       5       255
1      164069.44        3       2       5       5       255
2      164069.44        3       2       5       5       255
3      164069.44        3       2       5       5       255
4      164069.44        3       2       5       5       255
```

Once our data frame is ready for modelling, the data will be split in test and train data followed by applying various predicitive models

```
# Extract features and target variable
X = merged_df.drop(['PRICEEACH', 'CUSTOMERNAME', 'Cluster',
'RFM_Score','MSRP'], axis=1)  # Features
y = merged_df['PRICEEACH']  # Target variable

# Convert categorical RFM scores to numeric values
X['R_Score'] = X['R_Score'].astype(int)
X['F_Score'] = X['F_Score'].astype(int)
X['M_Score'] = X['M_Score'].astype(float)  # Convert to float since
it's originally a category

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#function definition for evaluating the models based on MSE and R-
SQUARED values

def evaluate_model(predictions, true_values, model_type):
    if model_type == 'regression':
        mse = mean_squared_error(true_values, predictions)
        r2 = r2_score(true_values, predictions)
        return mse, r2

    else:
        print("Invalid model type specified.")


# Train and test models for the entire dataset
# Linear Regression
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train, y_train)
linear_reg_predictions = linear_reg_model.predict(X_test)

# Evaluating the linear Regression model
model = "regression"
print("\nLinear Regression:")
linear_mse, linear_r2 = evaluate_model(linear_reg_predictions, y_test,
model)
```

```python
print('Mean Squared Error: ',linear_mse)
print('R-squared (R2): ',linear_r2, '\n')

#Decision Tree Regressor
decision_tree_model = DecisionTreeRegressor(random_state=42)
decision_tree_model.fit(X_train, y_train)
decision_tree_predictions = decision_tree_model.predict(X_test)

# Evaluating the Decision Tree model
model = "regression"
print("\nDecision Tree Regressor:")
decision_mse, decision_r2 = evaluate_model(decision_tree_predictions,
y_test, model)
print('Mean Squared Error: ',decision_mse)
print('R-squared (R2): ',decision_r2, '\n')

# Random Forest Regressor
random_forest_model = RandomForestRegressor(random_state=42)
random_forest_model.fit(X_train, y_train)
random_forest_predictions = random_forest_model.predict(X_test)

# Evaluate the Random Forest model
model = "regression"
print("\nRandom Forest Regressor:")
randomf_mse, randomf_r2 = evaluate_model(random_forest_predictions,
y_test, model)
print('Mean Squared Error: ',randomf_mse)
print('R-squared (R2): ',randomf_r2, '\n')

#Lasso Regression
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
lasso_predictions = lasso_model.predict(X_test)

# Evaluating the Lasso Regression model
model = "regression"
print("\nLasso Regression:")
lasso_mse, lasso_r2 = evaluate_model(lasso_predictions, y_test, model)
print('Mean Squared Error: ',lasso_mse)
print('R-squared (R2): ',lasso_r2, '\n')

#Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
ridge_predictions = ridge_model.predict(X_test)

# Evaluating the Ridge Regression model
model = "regression"
print("\nRidge Regression:")
ridge_mse, ridge_r2 = evaluate_model(ridge_predictions, y_test, model)
```

```
print('Mean Squared Error: ',ridge_mse)
print('R-squared (R2): ',ridge_r2, '\n')

#Bayesian Ridge Regression
bayesian_ridge_model = BayesianRidge()
bayesian_ridge_model.fit(X_train, y_train)
bayesian_ridge_predictions = bayesian_ridge_model.predict(X_test)

# Evaluating the Bayesian Ridge Regression model
model = "regression"
print("\nBayesian Ridge Regression:")
bayesian_mse, bayesian_r2 = evaluate_model(bayesian_ridge_predictions,
y_test, model)
print('Mean Squared Error: ',bayesian_mse)
print('R-squared (R2): ',bayesian_r2, '\n')

mse_values = [['Linear Regression', linear_mse],['Decision Tree
Regressor',decision_mse], ['Random Forest Regressor',randomf_mse],
['Lasso Regression',lasso_mse],['Ridge Regression',ridge_mse],
['Bayesian Ridge Regression',bayesian_mse]]
r2_values = [['Linear Regression', linear_r2],['Decision Tree
Regressor',decision_r2], ['Random Forest Regressor',randomf_r2],
['Lasso Regression',lasso_r2],['Ridge Regression',ridge_r2],
['Bayesian Ridge Regression',bayesian_r2]]
```

```
Linear Regression:
Mean Squared Error:  122.86593012977127
R-squared (R2):  0.9266219317968518


Decision Tree Regressor:
Mean Squared Error:  50.63467272727273
R-squared (R2):  0.9697599288517036


Random Forest Regressor:
Mean Squared Error:  14.11574762805455
R-squared (R2):  0.9915697843080192


Lasso Regression:
Mean Squared Error:  120.95826468962737
R-squared (R2):  0.9277612289529293


Ridge Regression:
Mean Squared Error:  122.52491002573079
R-squared (R2):  0.9268255960382449
```

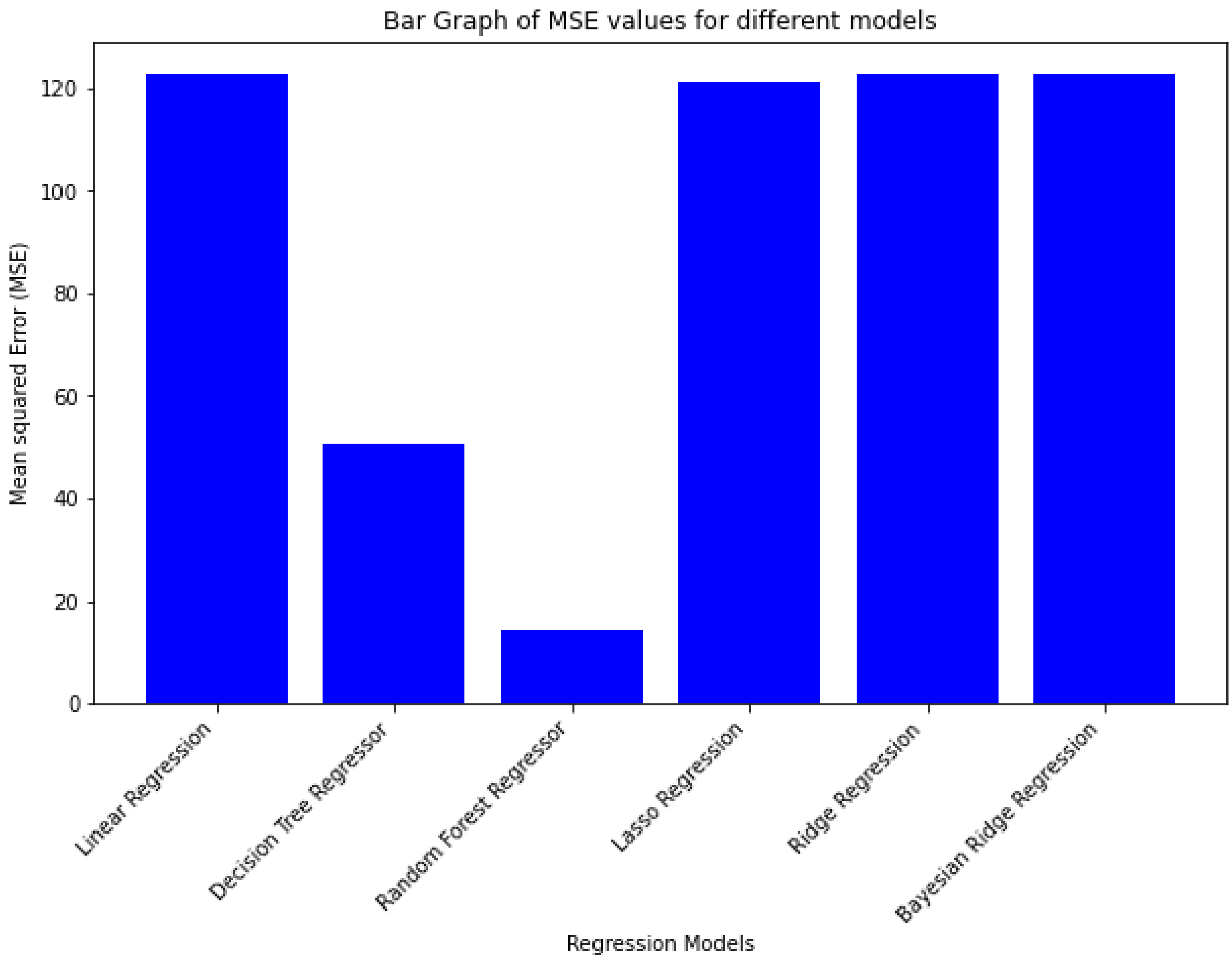
```
Bayesian Ridge Regression:
Mean Squared Error:  122.67696544197086
R-squared (R2):  0.9267347854067562
```

The code includes feature extraction, dataset splitting, standardization, and evaluation of various regression models.

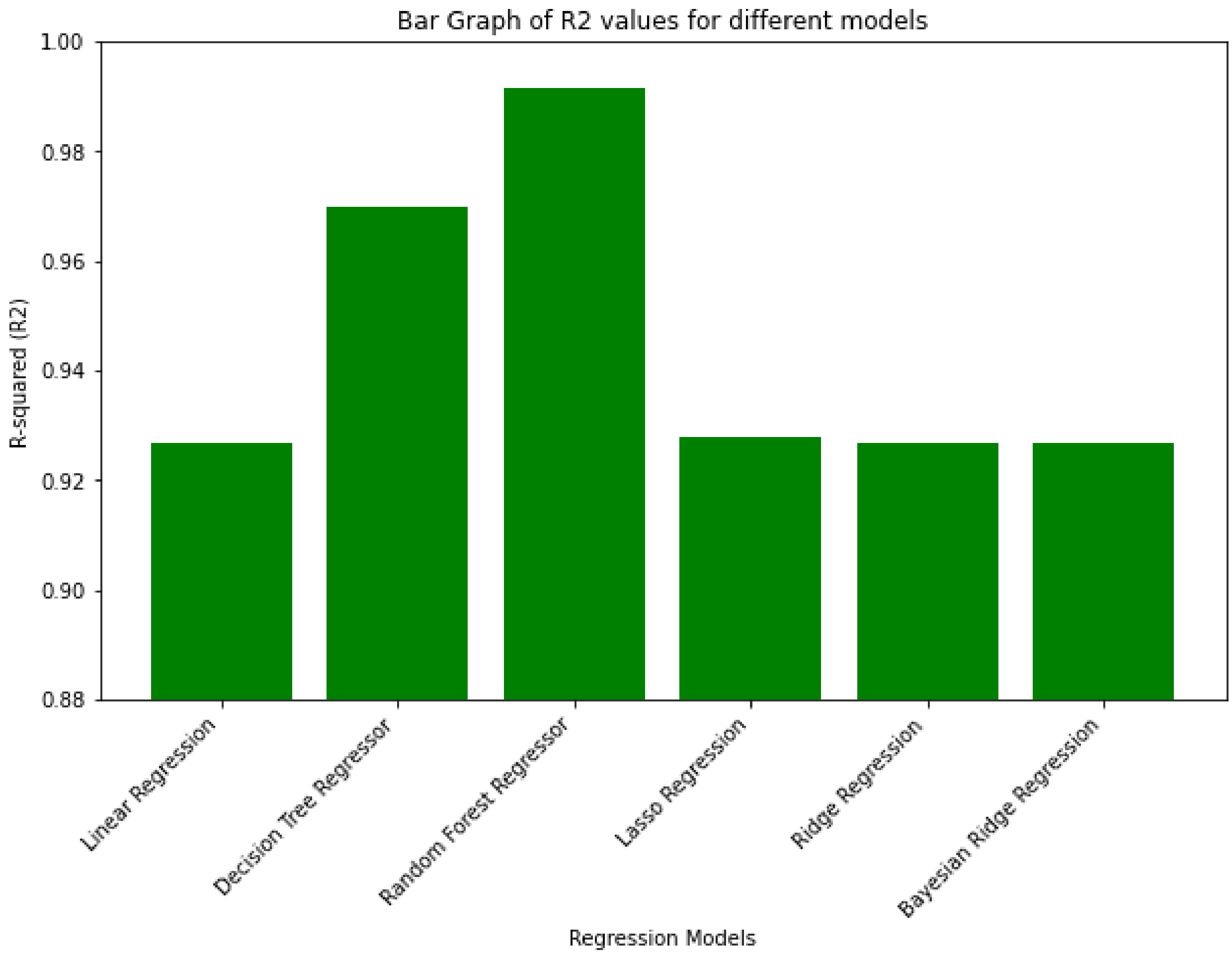
Model performance is assessed using Mean Squared Error (MSE) and R-squared values and from the above output we can observe that amonst all the models, Random Forest Regressor performed the best.

Visualisation of the performances of the models

```python
# Extracting data for plotting
regression_name, values = zip(*mse_values)

# Creating the bar graph
plt.figure(figsize=(10, 6))
plt.bar(regression_name, values, color='blue')
plt.xlabel('Regression Models')
plt.ylabel('Mean squared Error (MSE)')
plt.title('Bar Graph of MSE values for different models')
plt.xticks(rotation=45, ha='right')
plt.show()
print('\n\n')

# Extracting data for plotting
regression_name, values = zip(*r2_values)

# Creating the bar graph
plt.figure(figsize=(10, 6))
plt.bar(regression_name, values, color='green')
plt.xlabel('Regression Models')
plt.ylabel('R-squared (R2)')
plt.ylim(0.88, 1.0)
plt.title('Bar Graph of R2 values for different models')
plt.xticks(rotation=45, ha='right')
plt.show()
```



Bar Graph of MSE values for different models
Mean squared Error (MSE)
0
20
40
60
80
100
120
Linear Regression
Decision Tree Regressor
Random Forest Regressor
Lasso Regression
Ridge Regression
Bayesian Ridge Regression
Regression Models

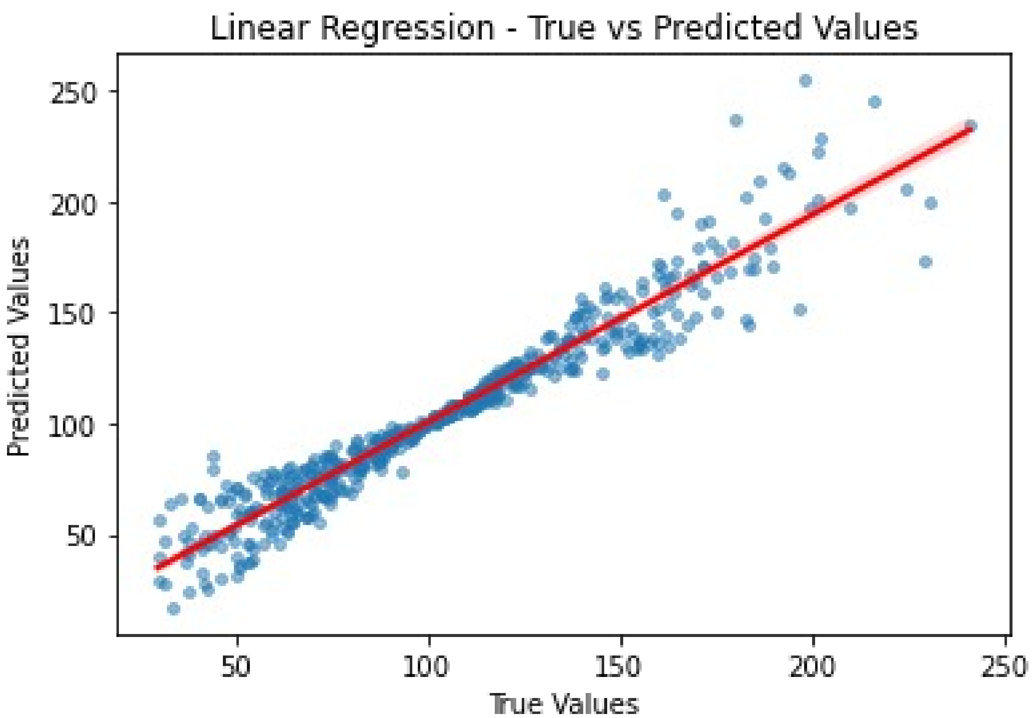


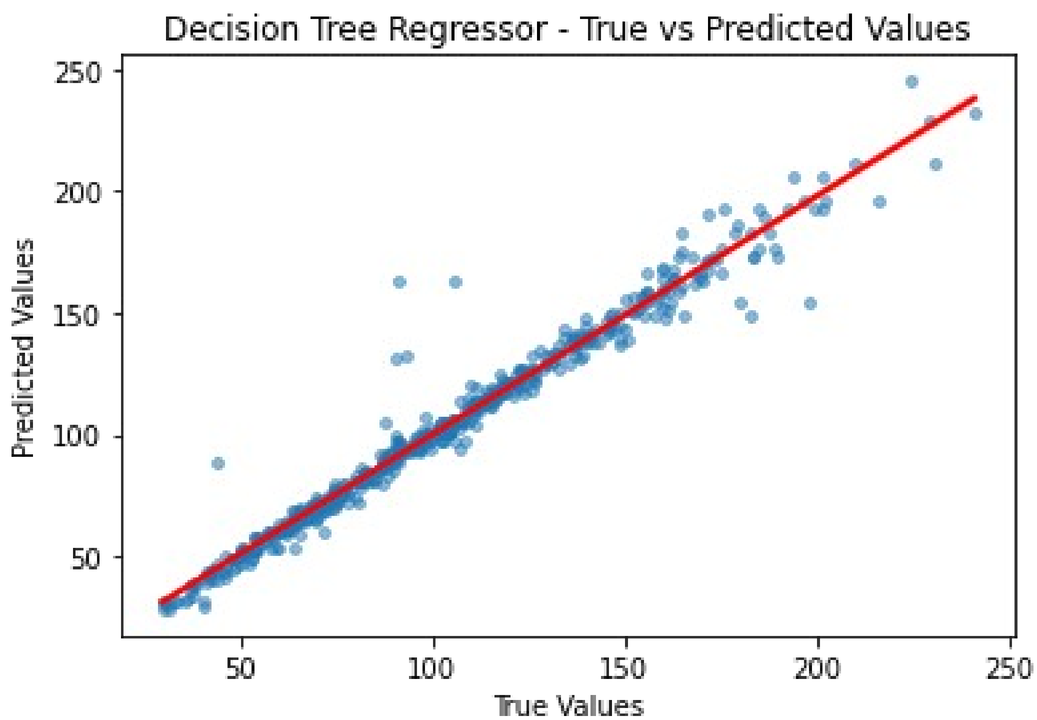
From the above graphs we can see that Random Forest regressor has the highest R2 score and lowest mean squared error.

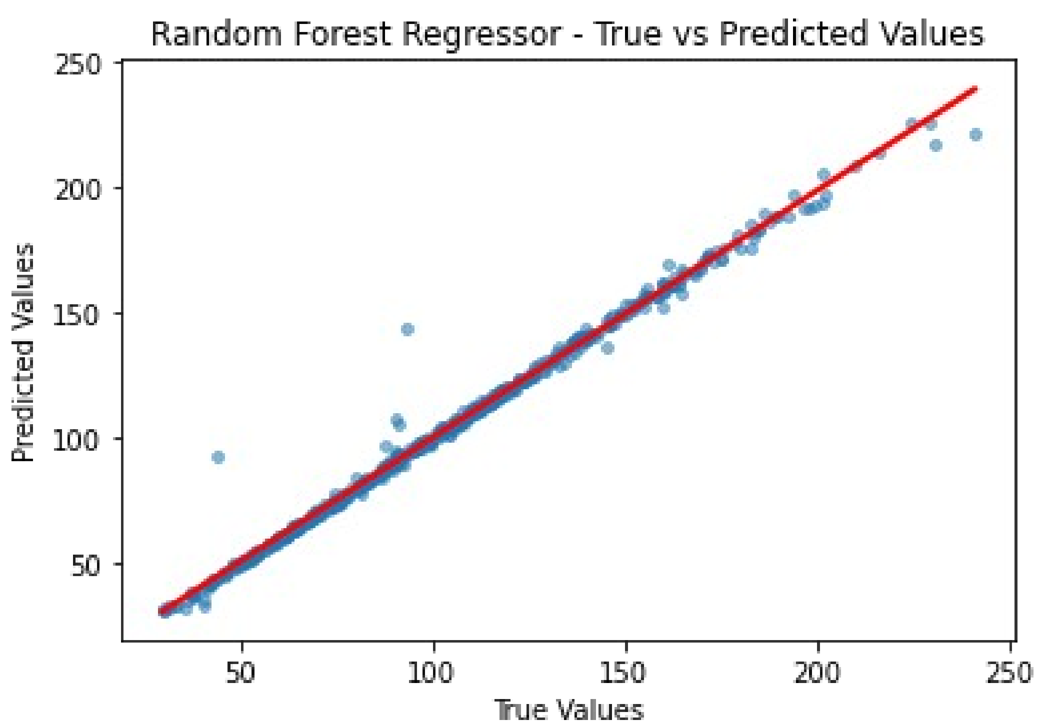
Once alll models have been made, the predicted values vs true values will be visualised for better understanding

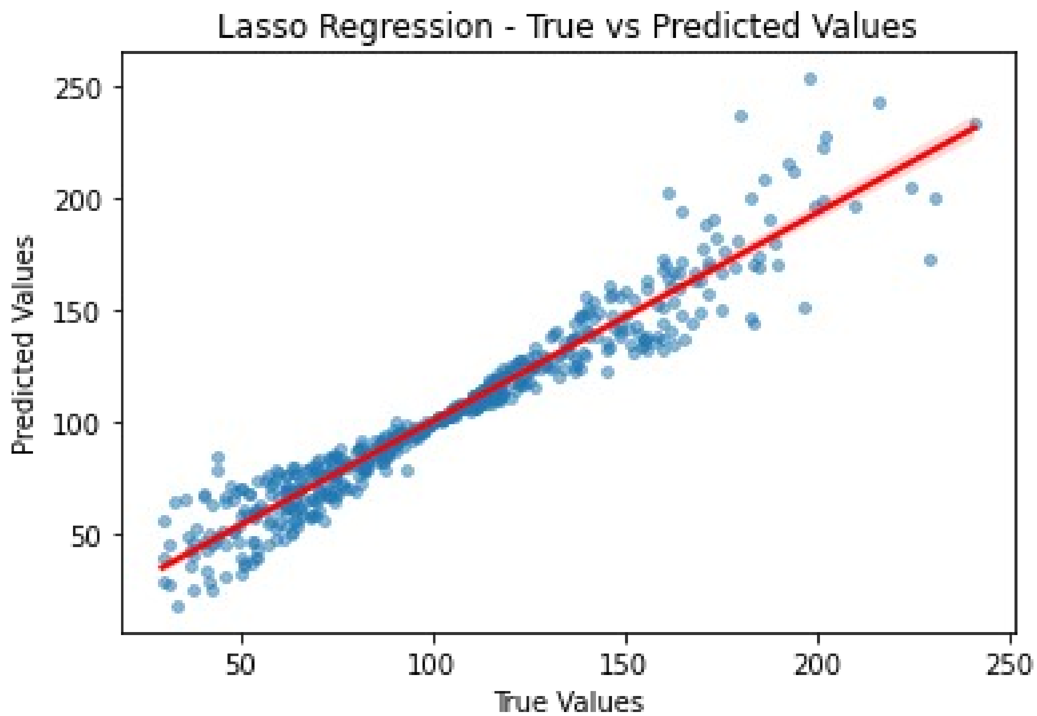
```
# Function to plot predicted vs true values
def plot_predictions(true_values, predicted_values, model_name):
    sns.regplot(x = true_values, y =
predicted_values,scatter_kws={'s': 15, 'alpha': 0.5},
line_kws={'color': 'red'} )
    plt.xlabel('True Values')
    plt.ylabel('Predicted Values')
    plt.title(f'{model_name} - True vs Predicted Values')
    plt.show()
```

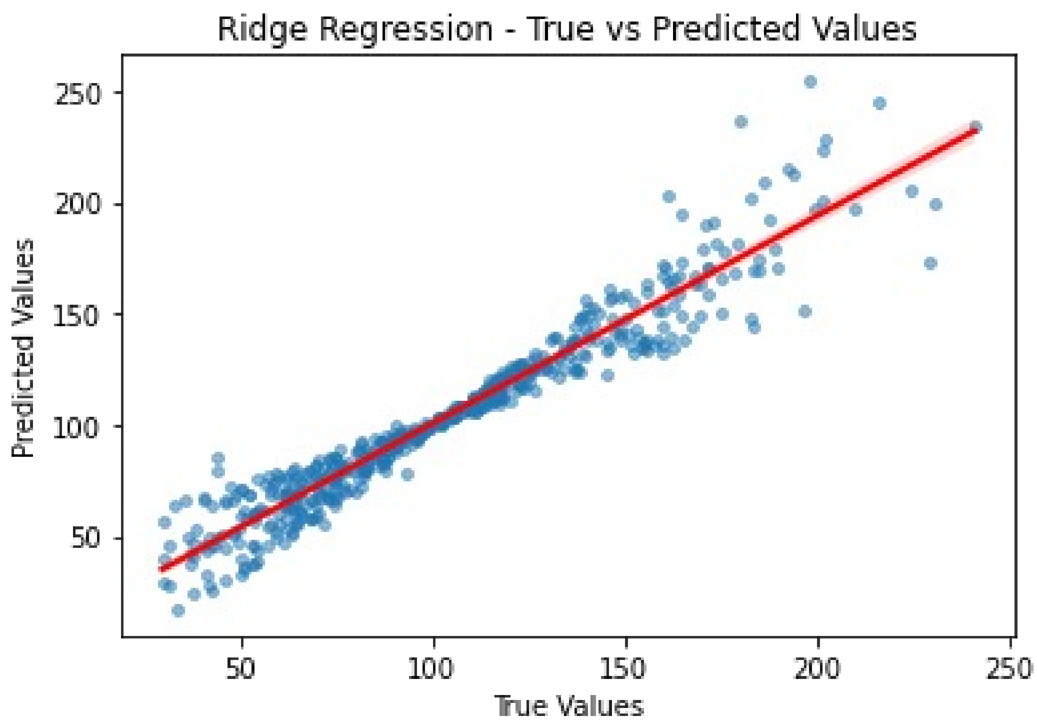
```
    print('\n')


# Plot Linear Regression predictions
plot_predictions(y_test, linear_reg_predictions, 'Linear Regression')

# Plot Decision Tree predictions
plot_predictions(y_test, decision_tree_predictions, 'Decision Tree
Regressor')

# Plot Random Forest predictions
plot_predictions(y_test, random_forest_predictions, 'Random Forest
Regressor')

# Plot Lasso Regression predictions
plot_predictions(y_test, lasso_predictions, 'Lasso Regression')

# Plot Ridge Regression predictions
plot_predictions(y_test, ridge_predictions, 'Ridge Regression')

# Plot Bayesian Ridge Regression predictions
plot_predictions(y_test, bayesian_ridge_predictions, 'Bayesian Ridge
Regression')
```

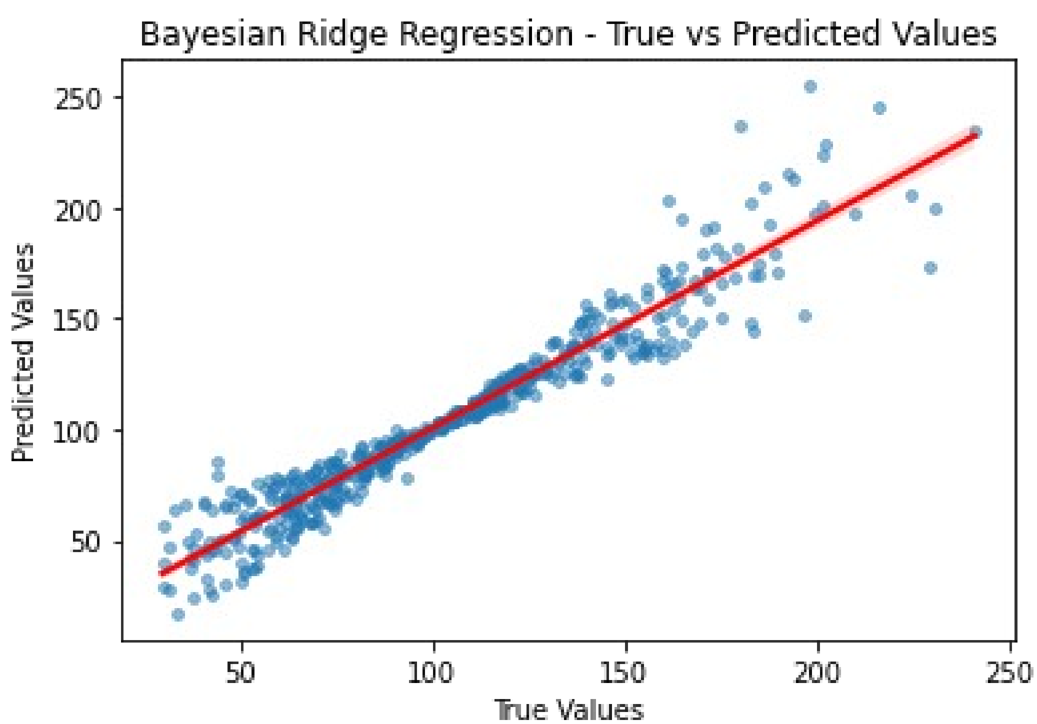



Decision Tree Regressor - True vs Predicted Values
Predicted Values
True Values
50
100
150
200
250


Random Forest Regressor - True vs Predicted Values
Predicted Values
True Values
50
100
150
200
250


Lasso Regression - True vs Predicted Values
Predicted Values
True Values
250
200
150
100
50
50
100
150
200
250


Ridge Regression - True vs Predicted Values
Predicted Values
True Values
50
100
150
200
250


Bayesian Ridge Regression - True vs Predicted Values
Predicted Values
True Values
50
100
150
200
250

The output shows the regression plots (true values vs predicted values) for the different models that we used and fits a regression line to them. After looking at the output we can see that Random Forest regressor and Decision Tree regressor are the best performers, and out of these two, Random Forest regressor performs the best.

Project Outcome (10 + 10 marks)

Overview of Results

Comprehensive analysis of the automotive sales dataset has provided valuable insights to optimize business strategies in the automotive industry. At first after having a brief overview about the data, exploratory data analysis (EDA) identified important patterns and trends in customer buying behavior by analsying factors such as Product line, deal size, Sales etc. A deeper understanding of the data was driven by analysing various factors according to geographical distribution as well.

By using clustering algorithms to segment customers based on recency, frequency, and monetary value, it is now possible to develop targeted marketing strategies tailored to different customer profiles based on the RFM score of customer segments.

Developing and evaluating order price prediction models that incorporate variables such as MSRP and RFM score provided a means to improve pricing strategies according to the segments created. Visualizations created with Matplotlib and Seaborn effectively conveyed the accuracy of different models and helped us choose the most reliable predictive model. Despite the potential challenges and limitations faced during analysis, this project supports companies with actionable recommendations derived from a combination of EDA, customer segmentation, predictive modeling, and impactful visualizations.

These insights not only contribute to a deeper understanding of customer preferences, but also provide the basis for informed decision-making, ultimately increasing customer satisfaction and competitiveness in a dynamic automotive market.

Exploratory Analysis of Purchase Patterns

Explanation of Results

Performing Exploratory Data Analysis using various techniques on our dataset, we have identified the following patterns in vehicle sales:

1) As observed in the Sales distribution graph, the maximum number of orders is having Sales between 1800 and 4000 with average Sales of the manufacturer being 3553.

2) The highest selling product line of the manufacturer being 'Classic Cars' with 949 units sold, followed by 'Vintage Cars' and 'Motorcycles' which has significantly lower Sales figure as

compared to 'Classic Cars' while the least sold Product Line being 'Train' which accounts for less than 3% of the total Sales.

3) USA is the biggest market for the manufacturer accounting for over one third of the net orders, followed by Spain(12.4%) and France(11.4%).

4) An interesting observation which can be inferred from the country wise product line count plot is that the top two product lines for USA and Spain are 'Classic Cars' and 'Vintage Cars' while that for France are 'Classic Cars' and 'Motorcycles'.

5) Deal size for the majority of orders is 'Medium' and 'Small' while 'Large' contributes less than 6% of the manufacturer's net sales.

6) The visualisation of the relationship between price of each product, quantity ordered and sales depict a linear relationship which infers the higher the order quantity or the higher the price of each product, the higher will be the total sales value.

7) Quantity ordered is majorly concentrated between 20 and 50 with few outliers reaching above 70.

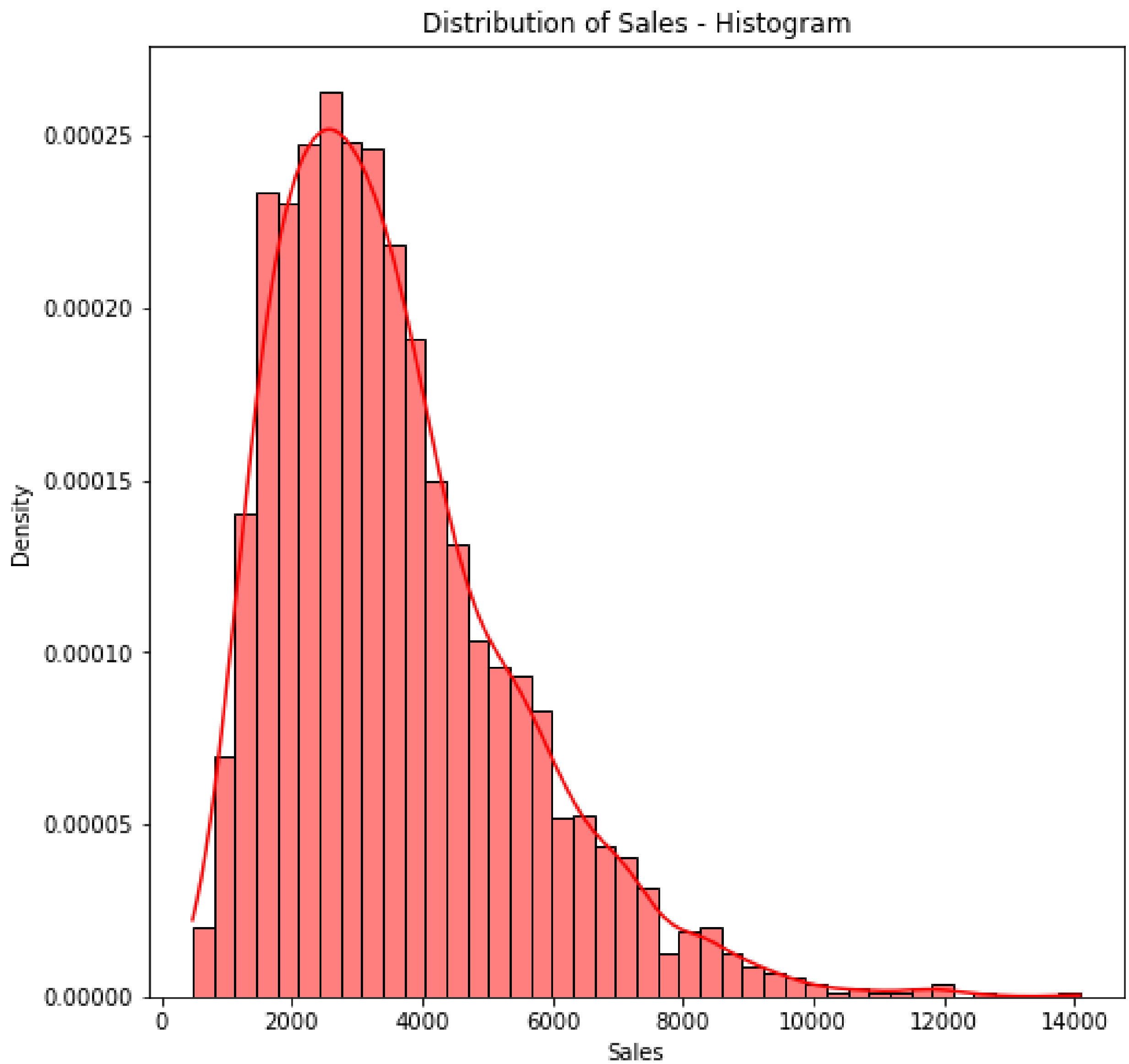
Note: 'DEALSIZE' is a categorised version of 'SALES' where if, 0 < 'SALES' < 3000 => 'DEALSIZE' = 'Small'; 3000 < 'SALES' < 5000 => 'DEALSIZE' = 'Medium'; 'SALES' > 5000 => 'DEALSIZE' = 'Large'.

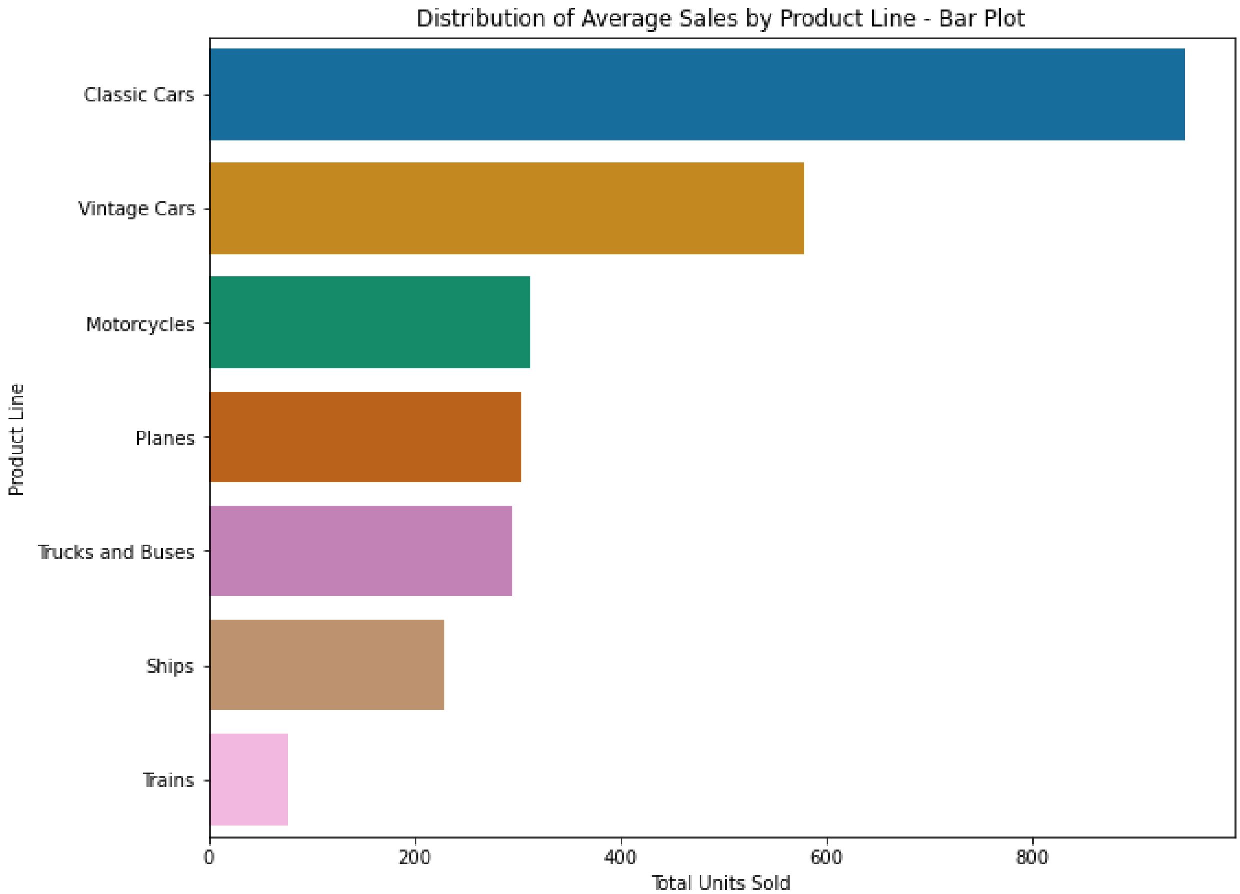
Visualisation

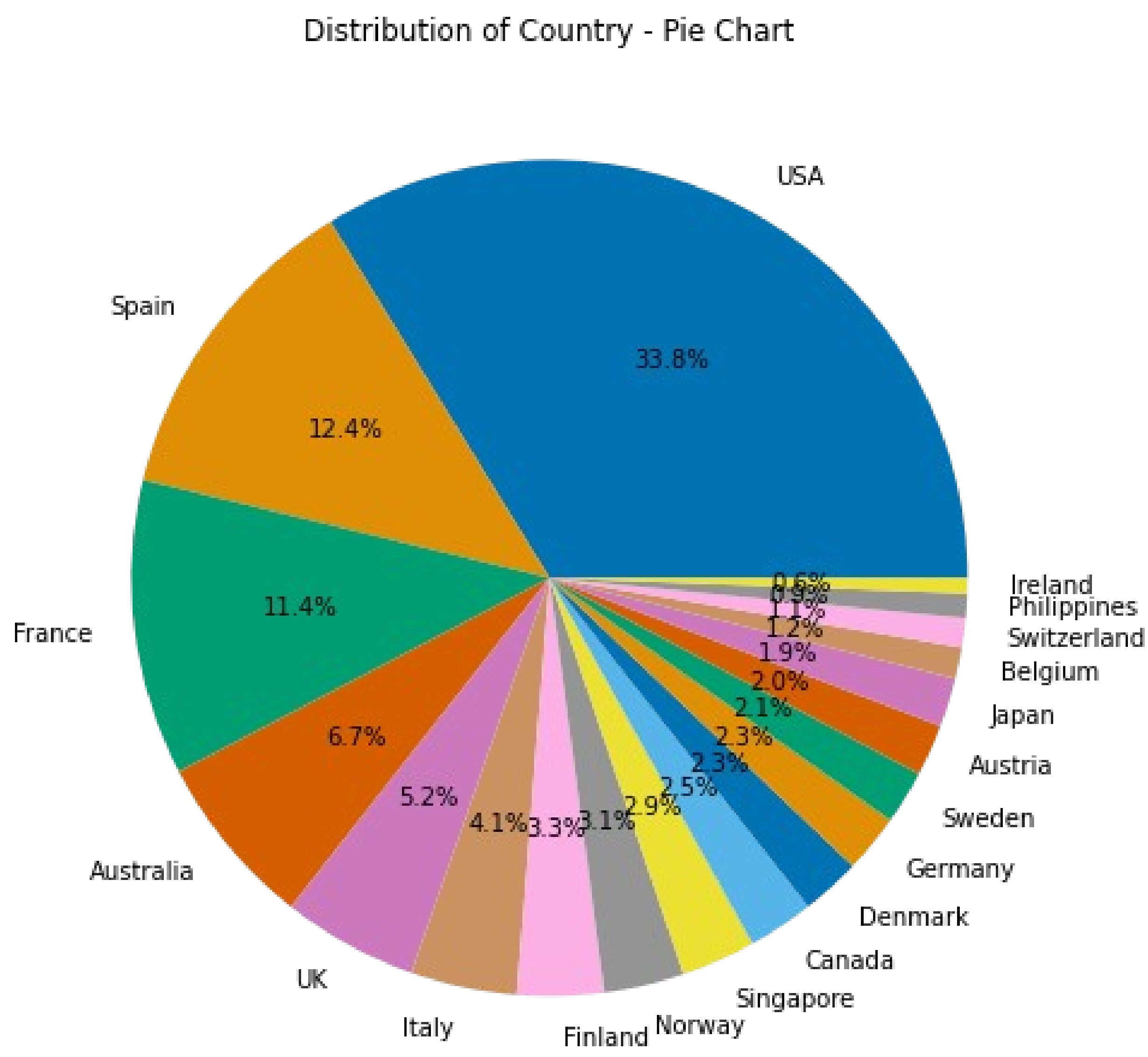
The following visualisations depict all the above finding.

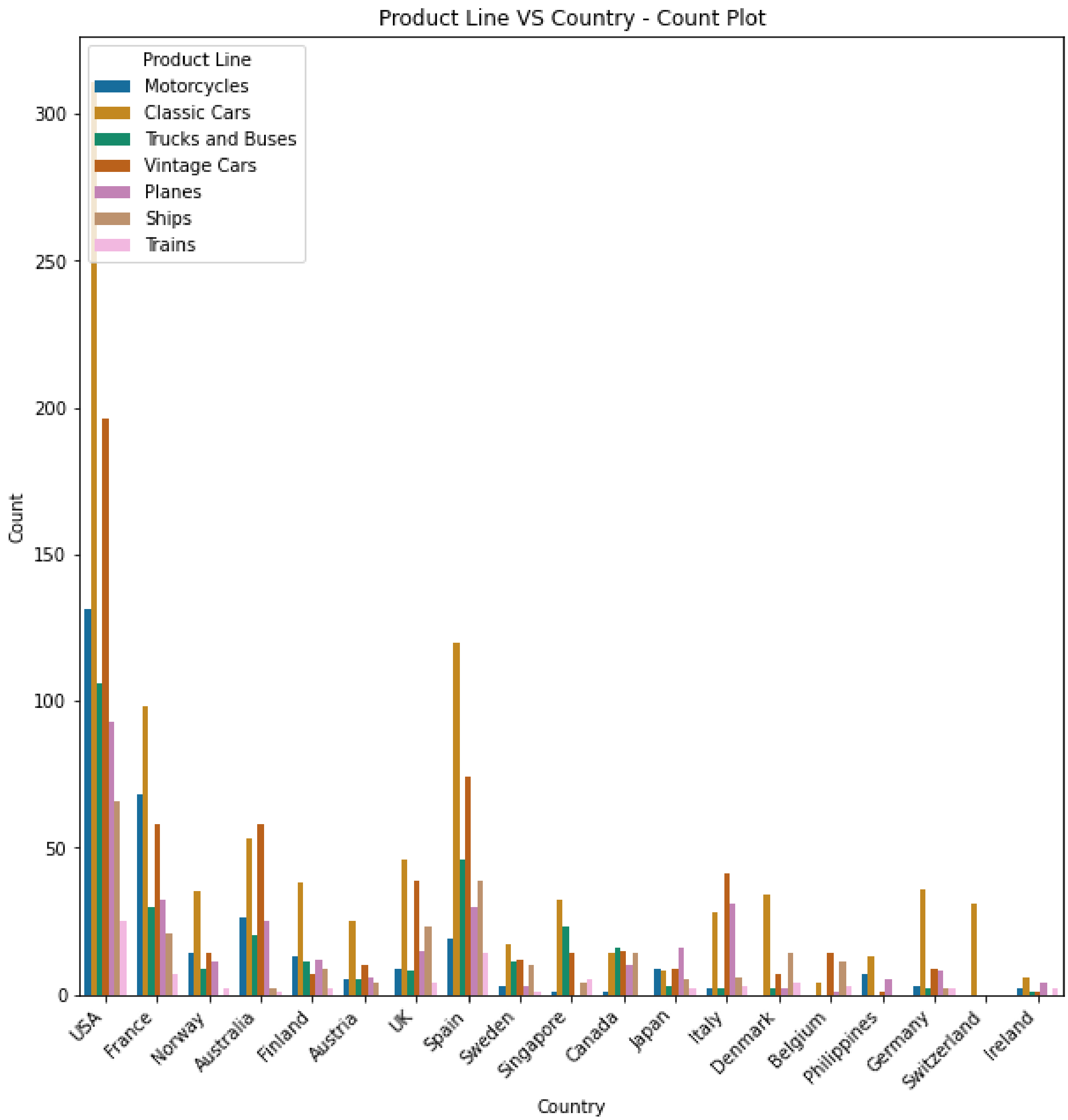
```
#Distribution of Sales:
plt.figure(figsize = (8,8))
sns.histplot(df['SALES'], kde = True, color = 'red', stat = 'density')
plt.title('Distribution of Sales - Histogram')
plt.xlabel('Sales')
plt.ylabel('Density')
plt.show()
print('\n\n')

#Distribution of Product line:
plt.figure(figsize = (10,8))
sns.countplot(y = 'PRODUCTLINE', data = df, palette = 'colorblind',
order = df['PRODUCTLINE'].value_counts().index)
plt.title('Distribution of Average Sales by Product Line - Bar Plot')
plt.xlabel('Total Units Sold')
plt.ylabel('Product Line')
plt.show()
print('\n\n')

#Distribution of Country:
plt.figure(figsize = (10,8))
country_counts = df['COUNTRY'].value_counts()
plt.pie(country_counts, labels = country_counts.index, autopct =
```

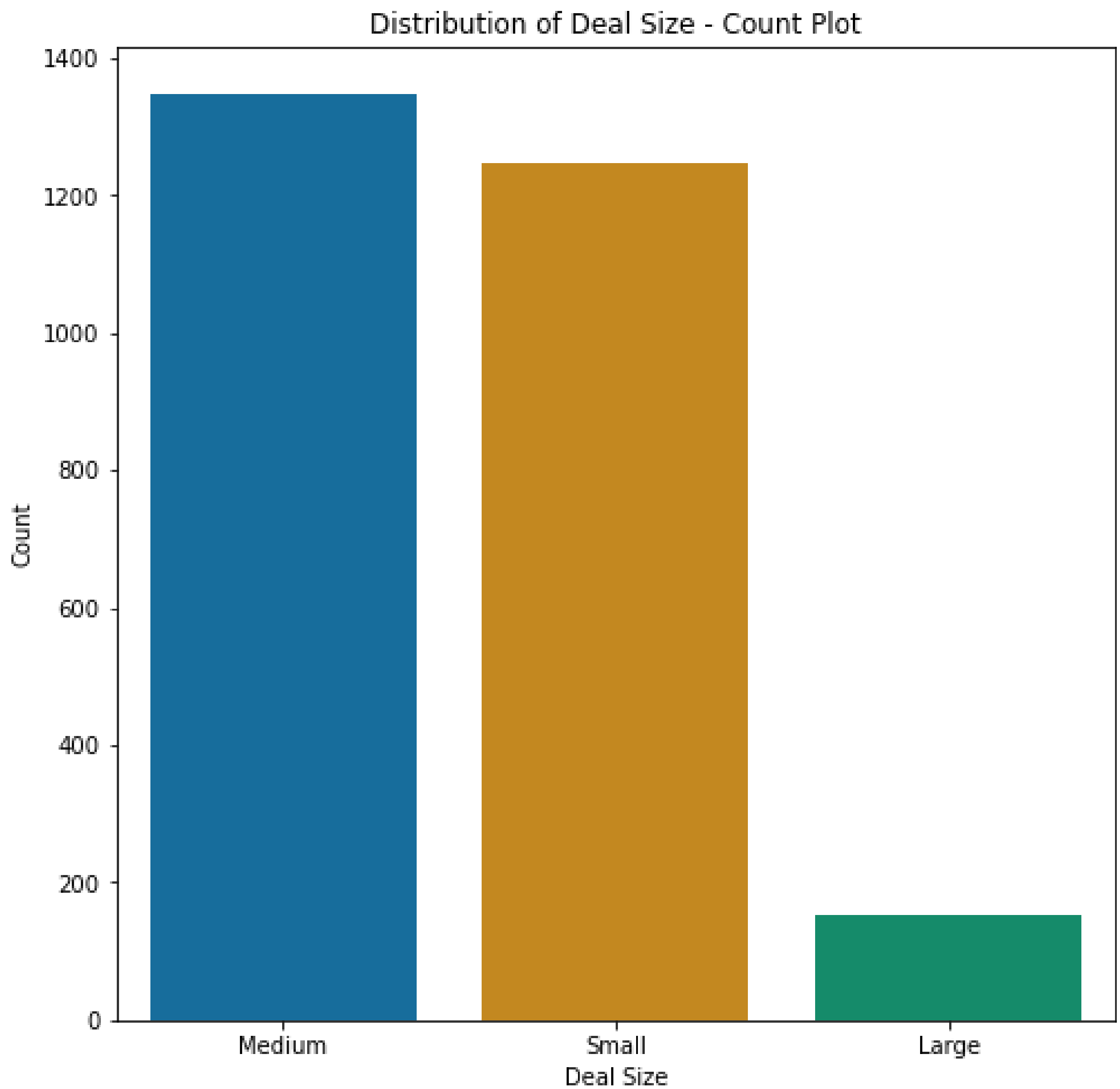
```
'%1.1f%%', colors = sns.color_palette('colorblind'))
plt.title('Distribution of Country - Pie Chart')
plt.show()
print('\n\n')

#Count plot for Product line VS Country:
plt.figure(figsize=(10,10))
sns.countplot(x = 'COUNTRY', hue = 'PRODUCTLINE', data = df, palette =
'colorblind')
plt.title('Product Line VS Country - Count Plot')
plt.xlabel('Country')
plt.ylabel('Count')
plt.xticks(rotation=45, ha = 'right')
plt.legend(title = 'Product Line')
plt.show()
print('\n')


#Distribution of Deal Size:
plt.figure(figsize = (8,8))
sns.countplot(x = 'DEALSIZE', data = df, palette = 'colorblind', order
= df['DEALSIZE'].value_counts().index)
plt.title('Distribution of Deal Size - Count Plot')
plt.xlabel('Deal Size')
plt.ylabel('Count')
plt.show()
print('\n\n')

#Joint plot (Scatter) for Price Each VS Sales:
plt.figure(figsize=(8,8))
sns.jointplot(x = 'PRICEEACH', y = 'SALES', data = df, kind =
'scatter', color = 'green')
plt.suptitle('Price Each VS Sales - Joint Plot', y = 1.02)
plt.show()
print('\n')

#Fitting the regression line to the Scatter Plot:
plt.figure(figsize = (8,8))
sns.regplot(x = 'QUANTITYORDERED', y = 'SALES', data = df,
scatter_kws={'s': 15, 'alpha': 0.5})
plt.title('Quantity Ordered VS Sales - Regression Plot')
plt.xlabel('Quantity Ordered')
plt.ylabel('Sales')
plt.show()
```


Distribution of Sales - Histogram
Density
Sales
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025
0
2000
4000
6000
8000
10000
12000
14000


Distribution of Average Sales by Product Line - Bar Plot
Classic Cars
Vintage Cars
Motorcycles
Planes
Trucks and Buses
Ships
Trains
Product Line
0
200
400
600
800
Total Units Sold

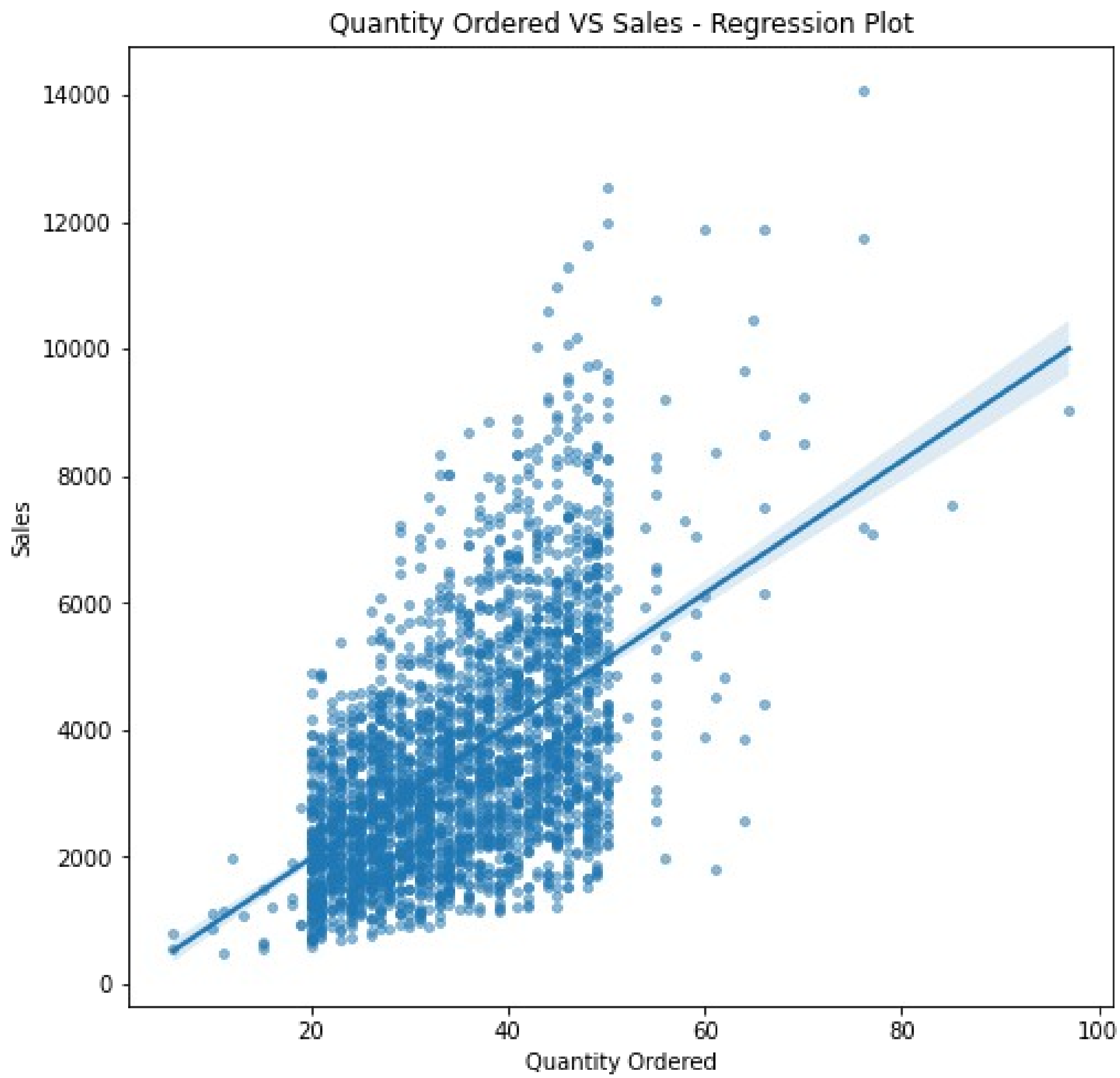

Distribution of Country - Pie Chart
USA
33.8%
Spain
12.4%
France
11.4%
Australia
6.7%
UK
5.2%
Italy
4.1%
Finland
3.3%
Norway
3.1%
Singapore
2.9%
Canada
2.5%
Denmark
2.3%
Germany
2.3%
Sweden
2.1%
Austria
2.0%
Japan
1.9%
Belgium
1.2%
Switzerland
1.1%
Philippines
0.9%
Ireland
0.6%


Product Line VS Country - Count Plot
Product Line
Motorcycles
Classic Cars
Trucks and Buses
Vintage Cars
Planes
Ships
Trains
Count
0
50
100
150
200
250
300
Country
USA
France
Norway
Australia
Finland
Austria
UK
Spain
Sweden
Singapore
Canada
Japan
Italy
Denmark
Belgium
Philippines
Germany
Switzerland
Ireland




```
<Figure size 576x576 with 0 Axes>
```


Price Each VS Sales - Joint Plot
SALES
14000
12000
10000
8000
6000
4000
2000
0
50
100
150
200
250
PRICEEACH



Customer Segmentation using Clustering

Explanation of Results

RFM is a technique that segments customer by their purchasing pattern [3]. We are segregating the customers into groups using RFM to assign the existing customers a loyalty rating and also to use the RFM score for predictive modelling later in this project.

In this section we have classified each unique customer based on three different factors:

1) Recency(time since last order): The recency value for each customer is calculated as the number of days between the most recent purchase date of the customer and the overall latest date in the dataset.

2) Frequency(number of orders): The frequency value is calculated by counting unique order numbers of each customer, representing the frequency of orders made by that particular customer.

3) Monetary(total sales): The monetary score is calculated by just summing up the total sales from each customer.

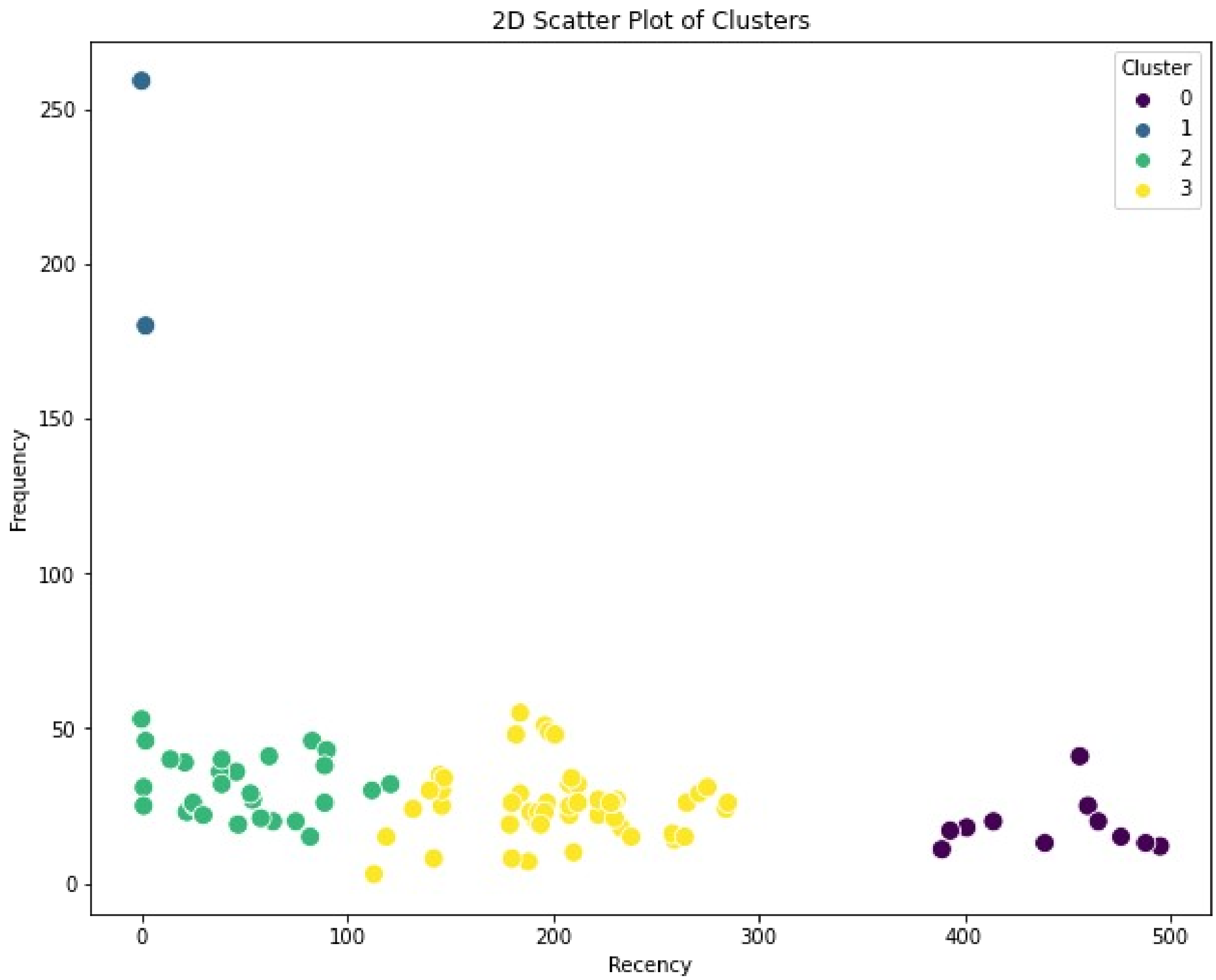
Once we get the RFM features of each customer we apply standardisation to transform the data to have a mean of 0 and a standard deviation of 1 as it ensures that all features contribute equally to the distance computation when k-means algorithm is applied.

Then we try to figure out the optimum number of customer cluster for the scaled RFM data by plotting the sum of square(distance) within cluster versus number of cluster and as observed from the elbow method plot, the entire customer list can be segmented into 4 clusters based on their loyalty score.

Visualisation

The 2-D scatter plot below perfectly depicts the customer clusters.

```
#2D scatter plot
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Recency', y='Frequency', hue='Cluster',
data=rfm_df, palette='viridis', s=100)

plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.title('2D Scatter Plot of Clusters')

plt.show()
```



Predictive Modeling for Price Determination

Explanation of Results

In this section we merged the coustomer segmentation dataset with the main one on customer name to assign previously calculated RFM scores to each customer as we are going to use the score as one of our input feature to train various models. Once we get the complete dataset we start looking for features which has significant(not too high) correlation with the feature we are trying to predict('PRICEEACH'). Observing the coorelation matrix, we conclude to drop the following features from the final dataset:

1) 'PRICEEACH': As we are trying to predict this feature it becomes the output of the model hence droped.

2) 'CUSTOMERNAME': It does not have any coorelation with 'PRICEEACH'.

3) 'Cluster': This is a categorical value of RFM scores which we have already taken as input feature.

4) 'RFM_Score': This is an aggregate score of R_score, M_score and F_score which we have already taken as input feature.

5) 'MSRP': It has high correlation with 'PRICEEACH', so to avoid redundancy we drop this feature.

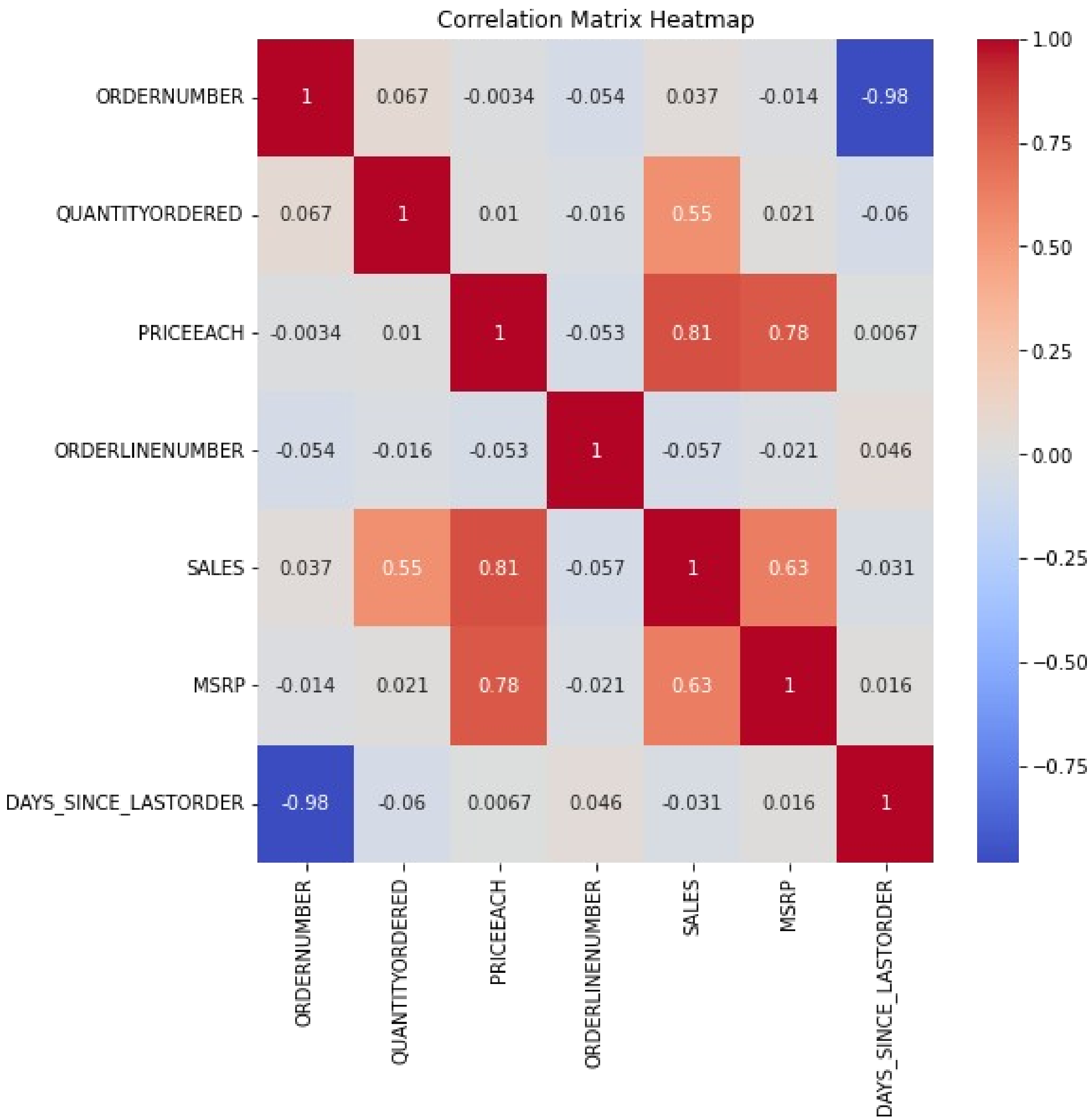
Then we split the entire dataset into train-test set of 4:1 ratio and standardised both the sets before feeding them to various models and evaluating accuracies of those models.

Visualisation

The below heat map vividly illustrates the correlation between different features in the dataset.

```
#Finding relationships between variables:
correlation_matrix = df.corr()
print(correlation_matrix, '\n\n')
plt.figure(figsize = (8,8))
sns.heatmap(correlation_matrix, annot = True, cmap = 'coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
                      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
MSRP                    -0.013910         0.020551   0.778393
DAYS_SINCE_LASTORDER    -0.982862        -0.059549   0.006688

                      ORDERLINENUMBER     SALES      MSRP  \
ORDERNUMBER                 -0.054300  0.037289 -0.013910
QUANTITYORDERED             -0.016295  0.553359  0.020551
PRICEEACH                   -0.052646  0.808287  0.778393
ORDERLINENUMBER              1.000000 -0.057414 -0.020956
SALES                       -0.057414  1.000000  0.634849
MSRP                        -0.020956  0.634849  1.000000
DAYS_SINCE_LASTORDER         0.045635 -0.030891  0.016465

                      DAYS_SINCE_LASTORDER
ORDERNUMBER                      -0.982862
QUANTITYORDERED                  -0.059549
PRICEEACH                         0.006688
ORDERLINENUMBER                   0.045635
SALES                            -0.030891
MSRP                              0.016465
DAYS_SINCE_LASTORDER              1.000000
```



Visualization of Model Accuracy

Explanation of Results

Finally, after training various models on the same set of features and testing on the remaining dataset, we have quite a few interesting observations:

1) Linear Regression:

Mean Squared Error: 122.87 R-squared (R2): 0.93 Linear Regression seems to provide a good fit with an R2 of 0.93, indicating that about 93% of the variance in the data is captured by the model. The Mean Squared Error is decent, though not the lowest.

2) Decision Tree Regressor:

Mean Squared Error: 50.63 R-squared (R2): 0.97 Decision Tree Regressor shows improved performance compared to Linear Regression, with a lower Mean Squared Error and a higher R2 value. It seems to capture the underlying patterns in the data more effectively.

3) Random Forest Regressor:

Mean Squared Error: 14.12 R-squared (R2): 0.99 Random Forest Regressor stands out with the lowest Mean Squared Error and a near-perfect R2 value of 0.99. This suggests that it's doing an excellent job in predicting the target variable, likely due to its ensemble nature.

4) Lasso Regression:

Mean Squared Error: 120.96 R-squared (R2): 0.93 Lasso Regression performs similarly to Linear Regression, but with a slightly higher Mean Squared Error. The R2 value is still in the same ballpark, indicating good explanatory power.

5) Ridge Regression:

Mean Squared Error: 122.52 R-squared (R2): 0.93 Ridge Regression results are close to Linear Regression, suggesting that regularization might not be significantly impacting the model's performance in this case.

6) Bayesian Ridge Regression:

Mean Squared Error: 122.68 R-squared (R2): 0.93 Similar to Ridge Regression, Bayesian Ridge Regression provides results comparable to Linear Regression, indicating stability but not a substantial improvement.

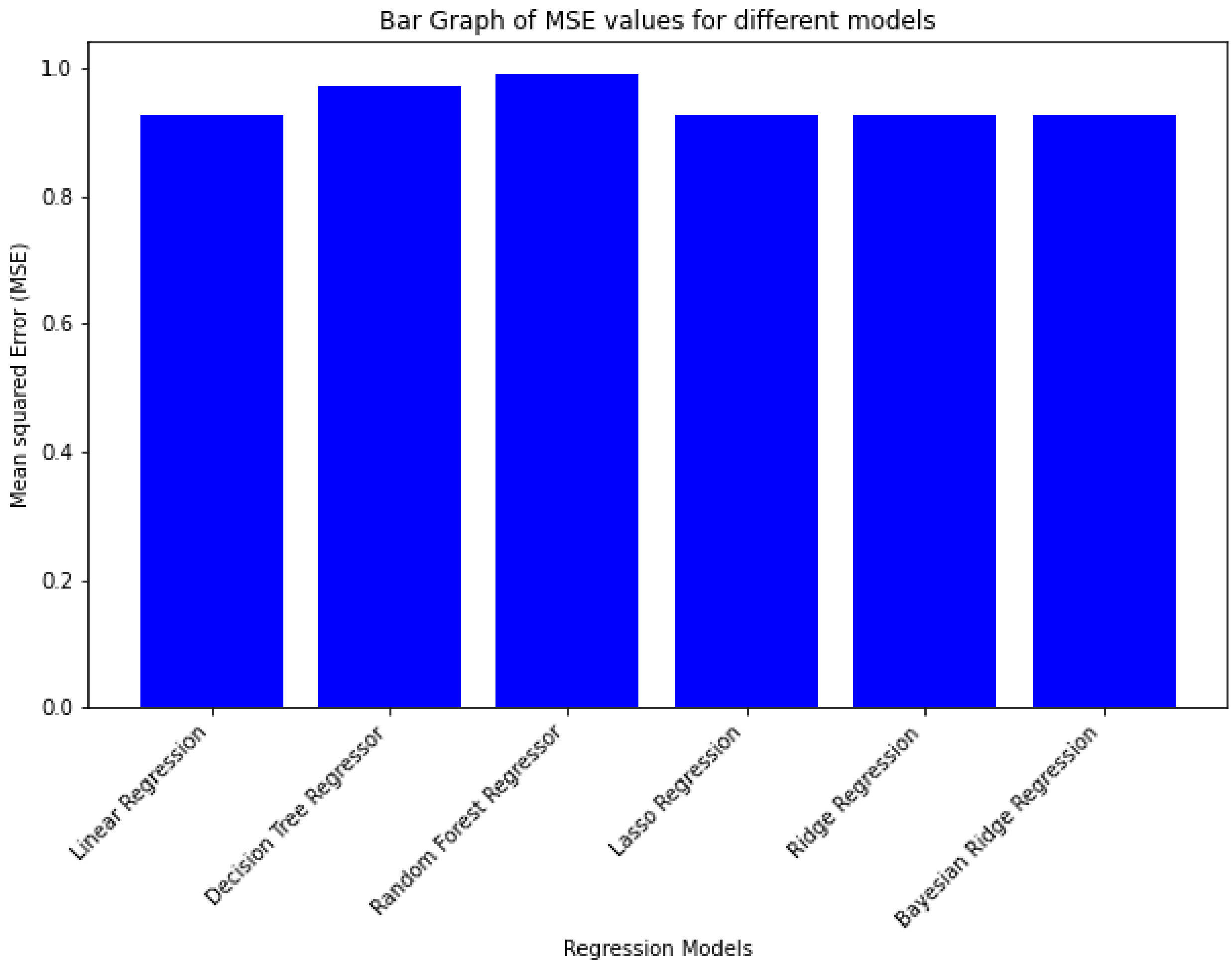
Among the models, Random Forest Regressor appears to be the winner, delivering the lowest Mean Squared Error and the highest R-squared value.

Visualisation

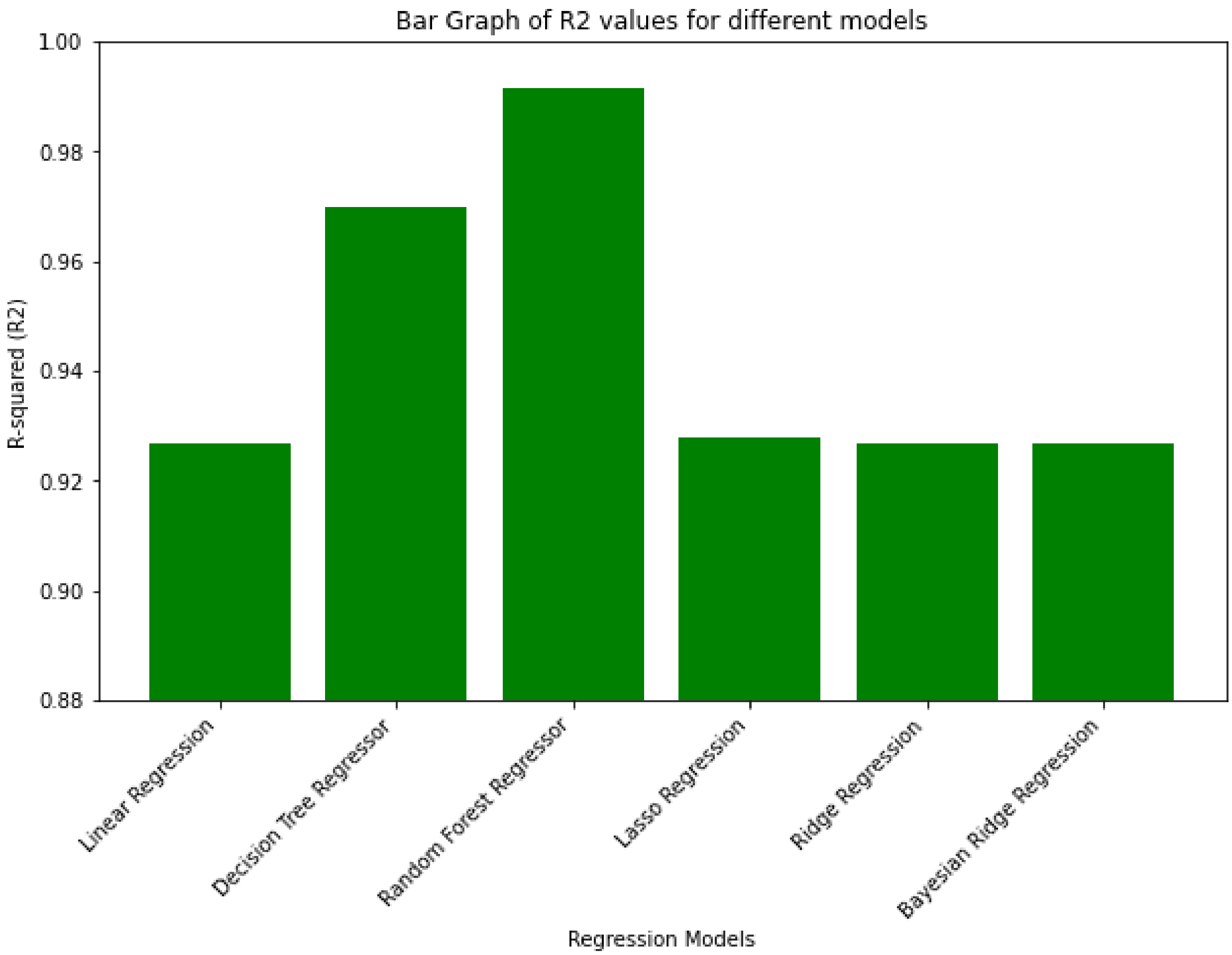
The below bar graph along with the regression plot clearly proves Random Forest Regression model is the best performer among all the models.

```
# Creating the bar graph
plt.figure(figsize=(10, 6))
plt.bar(regression_name, values, color='blue')
plt.xlabel('Regression Models')
plt.ylabel('Mean squared Error (MSE)')
plt.title('Bar Graph of MSE values for different models')
plt.xticks(rotation=45, ha='right')
plt.show()
print('\n\n')

# Creating the bar graph
plt.figure(figsize=(10, 6))
plt.bar(regression_name, values, color='green')
plt.xlabel('Regression Models')
```

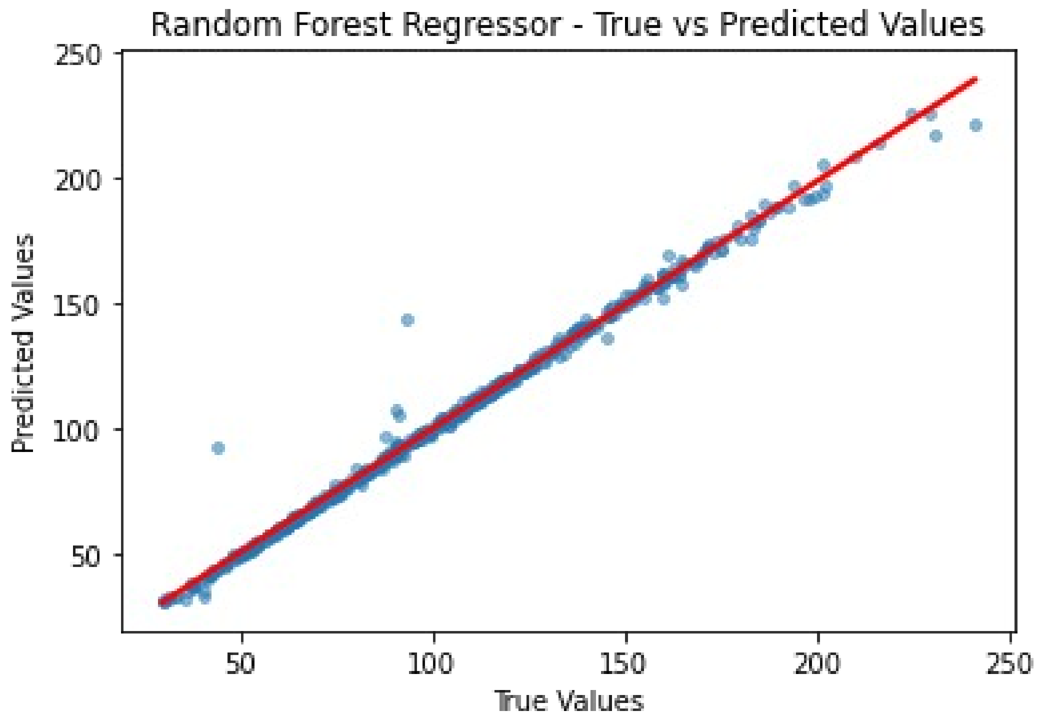
```
plt.ylabel('R-squared (R2)')
plt.ylim(0.88, 1.0)
plt.title('Bar Graph of R2 values for different models')
plt.xticks(rotation=45, ha='right')
plt.show()

# Plot Random Forest predictions
plot_predictions(y_test, random_forest_predictions, 'Random Forest
Regressor')
```




```

```



Bar Graph of R2 values for different models
R-squared (R2)
1.00
0.98
0.96
0.94
0.92
0.90
0.88
Linear Regression
Decision Tree Regressor
Random Forest Regressor
Lasso Regression
Ridge Regression
Bayesian Ridge Regression
Regression Models



Conclusion (5 marks)

Overall our project can be concluded as follows:

Achievements

In this project we have thoroughly analysed the sales data of an automotive manufacturer and uncovered key facts and figures about their customer's behaviour and demographics which will assist the business in understanding, serving and retaining their customers better.

We have also successfully segregated the customer base into multiple groups based on their value and loyalty which will eventually help the business to design loyalty schemes to retain high value customers and improve new customer acquisition strategy better.

Finally we noticed that price each of different products are being assigned dynamic values, hence we scrutinized relationship between price of each product with various factors like quantity ordered, MSRP and RMF scores of the customer to train a machine learning model to predict the price each for every order. This model will help automate the dynamic pricing of each product and hence will optimize the the Order To Cash process of the business. The accuracy of the model is very close to that of the original data.

Limitations

This project has the following limitations:

1) The dataset used has no missing values, neither does it have any inconsistencies hence it does not represent real world data where there might be many inconsistencies giving rise to complications.

2) The dataset also has very little outliers which can be a problem for the predictive when used in the real world scenario.

2) The model and the analysis is not generalised, hence is valid only for this automobile manufacturer and their dataset.

Future Work

In future work we would like to acquire high volume of data corresponding more to real world sales of the business to train the model for improved real world performance. We would also like to drop or combine few features used in the model to generalise it's application to make it industry independent. For example: to predict the sales of ice cream or clothes and not just vehicles.

References

[1] dee dee(https://www.kaggle.com/ddosad). 2023. Automobile Sales data. Kaggle. [Online]. [Accessed 22 November 2023]. Available from: https://www.kaggle.com/datasets/ddosad/auto-sales-data/data .

[2] Kabasakal, I. 2020. Customer Segmentation Based On Recency Frequency Monetary Model: A Case Study in E-Retailing. Bilişim Teknolojileri Dergisi. 13 (1), pp.48-55.

[3] Murphy, C. 2022. What Is Recency, Frequency, Monetary Value (RFM) in Marketing?. [Online]. [Accessed 22 November 2023]. Available from: https://www.investopedia.com/