

"Must-Know Libraries for Efficient Data Processing and Visualization"

```
In [6]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [13]: # Load dataset into notebook
```

```
In [14]: df = pd.read_csv('sales_data_sample.csv',encoding='ISO-8859-1')
```

```
In [15]: # check first five of dataset
```

```
In [16]: df.head()
```

```
Out[16]:
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	OR
0	10107	30	95.70	2	2871.00	
1	10121	34	81.35	5	2765.90	
2	10134	41	94.74	2	3884.34	
3	10145	45	83.26	6	3746.70	
4	10159	49	100.00	14	5205.27	10

5 rows × 25 columns



```
In [17]: # last five row of dataset
```

```
In [18]: df.tail()
```

Out[18]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES
2818	10350	20	100.00	15	2244.40
2819	10373	29	100.00	1	3978.51
2820	10386	43	100.00	4	5417.57
2821	10397	34	62.24	1	2116.16
2822	10414	47	65.52	9	3079.44

5 rows × 25 columns

In [19]: *# Total number of rows and columns present dataset*In [20]: `df.shape`

Out[20]: (2823, 25)

In [21]: *#check datatype of columns*In [22]: `df.dtypes`

```
Out[22]: ORDERNUMBER      int64
          QUANTITYORDERED  int64
          PRICEEACH        float64
          ORDERLINENUMBER  int64
          SALES            float64
          ORDERDATE        object
          STATUS           object
          QTR_ID           int64
          MONTH_ID         int64
          YEAR_ID          int64
          PRODUCTLINE      object
          MSRP             int64
          PRODUCTCODE      object
          CUSTOMERNAME     object
          PHONE            object
          ADDRESSLINE1     object
          ADDRESSLINE2     object
          CITY             object
          STATE            object
          POSTALCODE       object
          COUNTRY          object
          TERRITORY        object
          CONTACTLASTNAME  object
          CONTACTFIRSTNAME object
          DEALSIZE         object
          dtype: object
```

In [23]: *# an overview on dataset*

In []:

```
df.info()
```

"Optimizing Data: Refinement and Preprocessing Strategies"

In []:

In []:

In []:

Handling Missing Data: Dropping Rows with Null Values

In [24]: *# total number of null value present in the dataset*In [25]:

```
df.isnull().sum()
```

```
Out[25]: ORDERNUMBER      0
          QUANTITYORDERED  0
          PRICEEACH        0
          ORDERLINENUMBER  0
          SALES             0
          ORDERDATE        0
          STATUS           0
          QTR_ID           0
          MONTH_ID        0
          YEAR_ID         0
          PRODUCTLINE      0
          MSRP             0
          PRODUCTCODE      0
          CUSTOMERNAME     0
          PHONE            0
          ADDRESSLINE1     0
          ADDRESSLINE2    2521
          CITY             0
          STATE            1486
          POSTALCODE       76
          COUNTRY          0
          TERRITORY        1074
          CONTACTLASTNAME  0
          CONTACTFIRSTNAME 0
          DEALSIZE         0
          dtype: int64
```

In [26]: *# check total % of null value present in overall dataset*In [27]:

```
df.isnull().sum().sum()/(df.shape[0]*df.shape[1])*100
```

```
Out[27]: np.float64(7.30712008501594)
```

```
In [28]: # total 7% of data is missing from the dataset
```

```
In [29]: #check % of null values in each column
```

```
In [30]: (df.isnull().sum()/df.shape[0]*100)
```

```
Out[30]: ORDERNUMBER      0.000000
          QUANTITYORDERED  0.000000
          PRICEEACH        0.000000
          ORDERLINENUMBER  0.000000
          SALES             0.000000
          ORDERDATE        0.000000
          STATUS           0.000000
          QTR_ID           0.000000
          MONTH_ID         0.000000
          YEAR_ID          0.000000
          PRODUCTLINE      0.000000
          MSRP             0.000000
          PRODUCTCODE      0.000000
          CUSTOMERNAME     0.000000
          PHONE            0.000000
          ADDRESSLINE1     0.000000
          ADDRESSLINE2     89.302161
          CITY             0.000000
          STATE            52.639036
          POSTALCODE       2.692171
          COUNTRY          0.000000
          TERRITORY        38.044633
          CONTACTLASTNAME  0.000000
          CONTACTFIRSTNAME 0.000000
          DEALSIZE         0.000000
          dtype: float64
```

```
In [31]: # here we got to know that in
          # column % of null value
          #ADDRESSLINE2      89.302161
          #STATE      52.639036
          #POSTALCODE  2.692171
          #TERRITORY   38.044633
          # % of null value present
```

```
In [32]: # if our column contain more than 80-90 percentage of null value then we have to
          # b'coz fill this much of data manually can give oinaccurate output
```

```
In [33]: # dropping column
          df.drop('ADDRESSLINE2',axis =1,inplace=True)
```

```
In [34]: df.columns
```

```
Out[34]: Index(['ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER',
                'SALES', 'ORDERDATE', 'STATUS', 'QTR_ID', 'MONTH_ID', 'YEAR_ID',
                'PRODUCTLINE', 'MSRP', 'PRODUCTCODE', 'CUSTOMERNAME', 'PHONE',
                'ADDRESSLINE1', 'CITY', 'STATE', 'POSTALCODE', 'COUNTRY', 'TERRITORY',
                'CONTACTLASTNAME', 'CONTACTFIRSTNAME', 'DEALSIZE'],
                dtype='object')
```

```
In [35]: # sucessfully drop the column
```

Imputation Techniques for Handling Missing Data

```
In [36]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   ORDERNUMBER           2823 non-null   int64
 1   QUANTITYORDERED       2823 non-null   int64
 2   PRICEEACH             2823 non-null   float64
 3   ORDERLINENUMBER       2823 non-null   int64
 4   SALES                 2823 non-null   float64
 5   ORDERDATE             2823 non-null   object
 6   STATUS                2823 non-null   object
 7   QTR_ID                2823 non-null   int64
 8   MONTH_ID              2823 non-null   int64
 9   YEAR_ID               2823 non-null   int64
10   PRODUCTLINE           2823 non-null   object
11   MSRP                  2823 non-null   int64
12   PRODUCTCODE           2823 non-null   object
13   CUSTOMERNAME          2823 non-null   object
14   PHONE                 2823 non-null   object
15   ADDRESSLINE1          2823 non-null   object
16   CITY                  2823 non-null   object
17   STATE                 1337 non-null   object
18   POSTALCODE            2747 non-null   object
19   COUNTRY               2823 non-null   object
20   TERRITORY             1749 non-null   object
21   CONTACTLASTNAME       2823 non-null   object
22   CONTACTFIRSTNAME      2823 non-null   object
23   DEALSIZE              2823 non-null   object
dtypes: float64(2), int64(7), object(15)
memory usage: 529.4+ KB
```

```
In [37]: df.isnull().sum()
```

```
Out[37]: ORDERNUMBER      0
          QUANTITYORDERED  0
          PRICEEACH        0
          ORDERLINENUMBER  0
          SALES             0
          ORDERDATE        0
          STATUS           0
          QTR_ID           0
          MONTH_ID         0
          YEAR_ID          0
          PRODUCTLINE      0
          MSRP             0
          PRODUCTCODE      0
          CUSTOMERNAME     0
          PHONE            0
          ADDRESSLINE1     0
          CITY             0
          STATE            1486
          POSTALCODE       76
          COUNTRY          0
          TERRITORY        1074
          CONTACTLASTNAME  0
          CONTACTFIRSTNAME 0
          DEALSIZE         0
          dtype: int64
```

```
In [38]: # here the column which contain null values are of object datatypes
```

```
In [39]: for i in df.select_dtypes(include='object').columns:
          df[i].fillna(df[i].mode()[0],inplace=True)
```

C:\Users\sunstone\AppData\Local\Temp\ipykernel_14892\2459431018.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df[i].fillna(df[i].mode()[0],inplace=True)
```

```
In [40]: df.isnull().sum
```

```

Out[40]: <bound method DataFrame.sum of
ORDERLINENUMBER  SALES  \
0                False      False      False      False      False
1                False      False      False      False      False
2                False      False      False      False      False
3                False      False      False      False      False
4                False      False      False      False      False
...              ...        ...        ...        ...        ...
2818             False      False      False      False      False
2819             False      False      False      False      False
2820             False      False      False      False      False
2821             False      False      False      False      False
2822             False      False      False      False      False

ORDERDATE  STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  PHONE  ADDRESSLINE1  \
0          False  False   False     False   False  ...  False      False
1          False  False   False     False   False  ...  False      False
2          False  False   False     False   False  ...  False      False
3          False  False   False     False   False  ...  False      False
4          False  False   False     False   False  ...  False      False
...          ...    ...     ...       ...     ...  ...    ...
2818       False  False   False     False   False  ...  False      False
2819       False  False   False     False   False  ...  False      False
2820       False  False   False     False   False  ...  False      False
2821       False  False   False     False   False  ...  False      False
2822       False  False   False     False   False  ...  False      False

CITY  STATE  POSTALCODE  COUNTRY  TERRITORY  CONTACTLASTNAME  \
0     False  False      False     False     False      False
1     False  False      False     False     False      False
2     False  False      False     False     False      False
3     False  False      False     False     False      False
4     False  False      False     False     False      False
...      ...    ...       ...       ...       ...      ...
2818  False  False      False     False     False      False
2819  False  False      False     False     False      False
2820  False  False      False     False     False      False
2821  False  False      False     False     False      False
2822  False  False      False     False     False      False

CONTACTFIRSTNAME  DEALSIZE
0                False      False
1                False      False
2                False      False
3                False      False
4                False      False
...              ...        ...
2818             False      False
2819             False      False
2820             False      False
2821             False      False
2822             False      False

[2823 rows x 24 columns]>

```

```
In [41]: #we sucessfully able to fill missing values
```

Eliminating Duplicates: Finding and Dropping Duplicate Rows

```
In [42]: df.duplicated().sum()
```

```
Out[42]: np.int64(0)
```

Standardizing Date and Number Formats for Improved Data Uniformity

```
In [43]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 24 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ORDERNUMBER           2823 non-null   int64
1   QUANTITYORDERED       2823 non-null   int64
2   PRICEEACH             2823 non-null   float64
3   ORDERLINENUMBER       2823 non-null   int64
4   SALES                 2823 non-null   float64
5   ORDERDATE             2823 non-null   object
6   STATUS                2823 non-null   object
7   QTR_ID               2823 non-null   int64
8   MONTH_ID             2823 non-null   int64
9   YEAR_ID              2823 non-null   int64
10  PRODUCTLINE           2823 non-null   object
11  MSRP                  2823 non-null   int64
12  PRODUCTCODE           2823 non-null   object
13  CUSTOMERNAME          2823 non-null   object
14  PHONE                 2823 non-null   object
15  ADDRESSLINE1          2823 non-null   object
16  CITY                  2823 non-null   object
17  STATE                 2823 non-null   object
18  POSTALCODE            2823 non-null   object
19  COUNTRY               2823 non-null   object
20  TERRITORY             2823 non-null   object
21  CONTACTLASTNAME       2823 non-null   object
22  CONTACTFIRSTNAME      2823 non-null   object
23  DEALSIZE              2823 non-null   object
dtypes: float64(2), int64(7), object(15)
memory usage: 529.4+ KB
```

```
In [44]: # ORDERDATE column should be in date format but it is in object
```

```
In [45]: df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])
```

```
In [46]: # drive year from ORDERDATE column
df['YEAR'] = df['ORDERDATE'].dt.year
df['YEAR'] = df['YEAR'].round().astype(int)
```

```
In [47]: df.info()
```

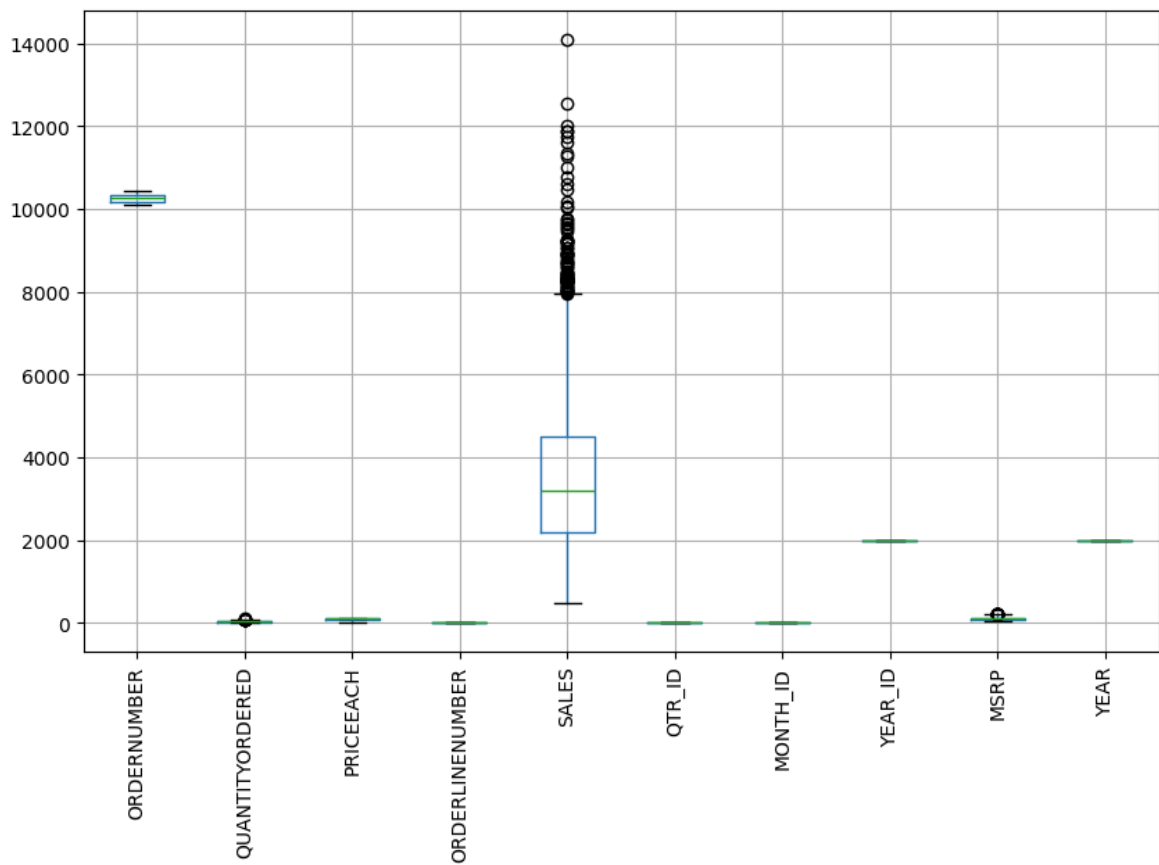


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ORDERNUMBER           2823 non-null   int64
1   QUANTITYORDERED       2823 non-null   int64
2   PRICEEACH             2823 non-null   float64
3   ORDERLINENUMBER       2823 non-null   int64
4   SALES                 2823 non-null   float64
5   ORDERDATE             2823 non-null   datetime64[ns]
6   STATUS               2823 non-null   object
7   QTR_ID               2823 non-null   int64
8   MONTH_ID             2823 non-null   int64
9   YEAR_ID              2823 non-null   int64
10  PRODUCTLINE           2823 non-null   object
11  MSRP                  2823 non-null   int64
12  PRODUCTCODE           2823 non-null   object
13  CUSTOMERNAME          2823 non-null   object
14  PHONE                 2823 non-null   object
15  ADDRESSLINE1          2823 non-null   object
16  CITY                  2823 non-null   object
17  STATE                 2823 non-null   object
18  POSTALCODE            2823 non-null   object
19  COUNTRY               2823 non-null   object
20  TERRITORY             2823 non-null   object
21  CONTACTLASTNAME       2823 non-null   object
22  CONTACTFIRSTNAME      2823 non-null   object
23  DEALSIZE              2823 non-null   object
24  YEAR                  2823 non-null   int64
dtypes: datetime64[ns](1), float64(2), int64(8), object(14)
memory usage: 551.5+ KB
```

Identifying and Handling Outliers for Accurate Data Analysis

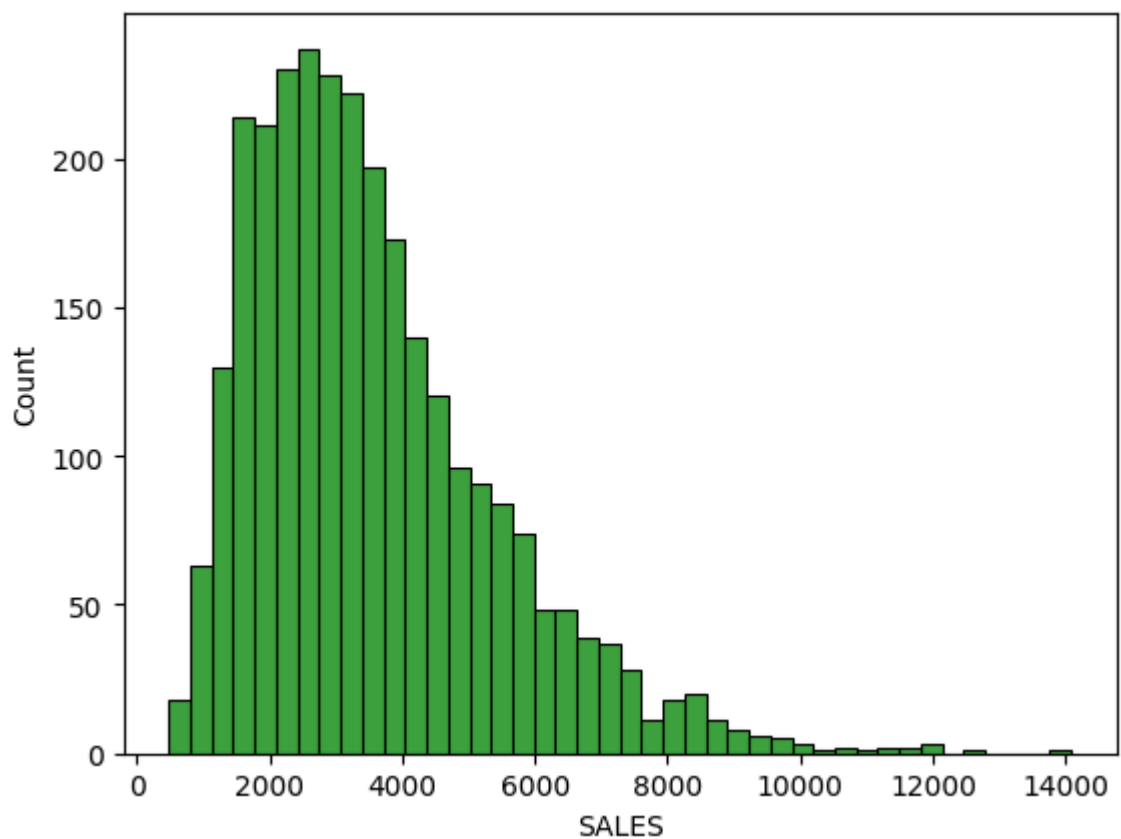
```
In [48]: # for multiple columns
```

```
In [49]: df.select_dtypes(include=['int64', 'float64']).boxplot(figsize=(10,6))
plt.xticks(rotation=90)
plt.show()
```



```
In [50]: # for single columns
```

```
In [51]: sns.histplot(df['SALES'],color='green')  
plt.show()
```




Introduction to Summary Statistics for Data Analysis

In [52]: `df.describe()`

Out[52]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.0
mean	10258.725115	35.092809	83.658544	6.466171	3553.8
min	10100.000000	6.000000	26.880000	1.000000	482.1
25%	10180.000000	27.000000	68.860000	3.000000	2203.4
50%	10262.000000	35.000000	95.700000	6.000000	3184.8
75%	10333.500000	43.000000	100.000000	9.000000	4508.0
max	10425.000000	97.000000	100.000000	18.000000	14082.8
std	92.085478	9.741443	20.174277	4.225841	1841.8




In [53]: `# for categorical datatype`

In [54]: `df.describe(include='object')`

Out[54]:

	STATUS	PRODUCTLINE	PRODUCTCODE	CUSTOMERNAME	PHONE	ADDRESSL
count	2823	2823	2823	2823	2823	
unique	6	7	109	92	91	
top	Shipped	Classic Cars	S18_3232	Euro Shopping Channel	(91) 555 94 44	C/ Moralz
freq	2617	967	52	259	259	



Managing Missing Data and Anomalies: Statistical and ML Methods

univariate analysis

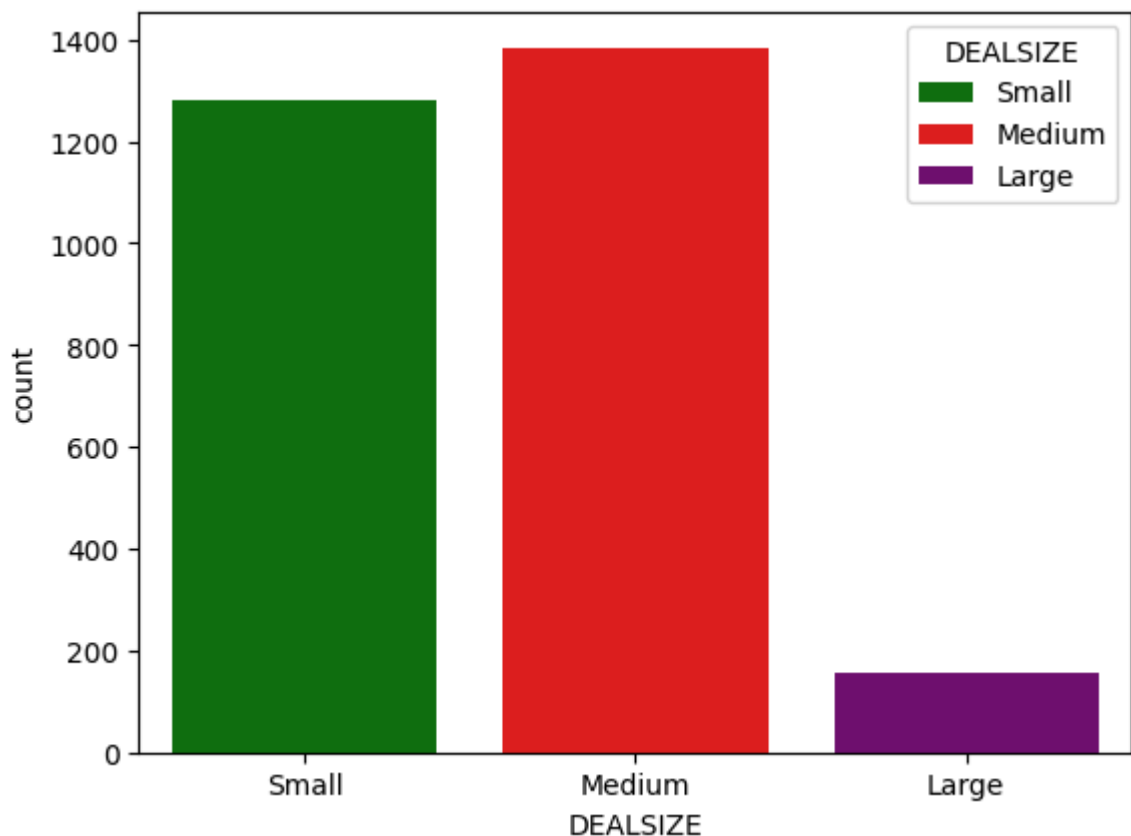
In []:

```
In [55]: sns.countplot(x = df['DEALSIZE'],palette=['green', 'red', 'purple'])  
plt.legend(title="DEALSIZE", labels=df['DEALSIZE'].unique())  
plt.show()
```

C:\Users\sunstone\AppData\Local\Temp\ipykernel_14892\2098529877.py:1: FutureWarning:

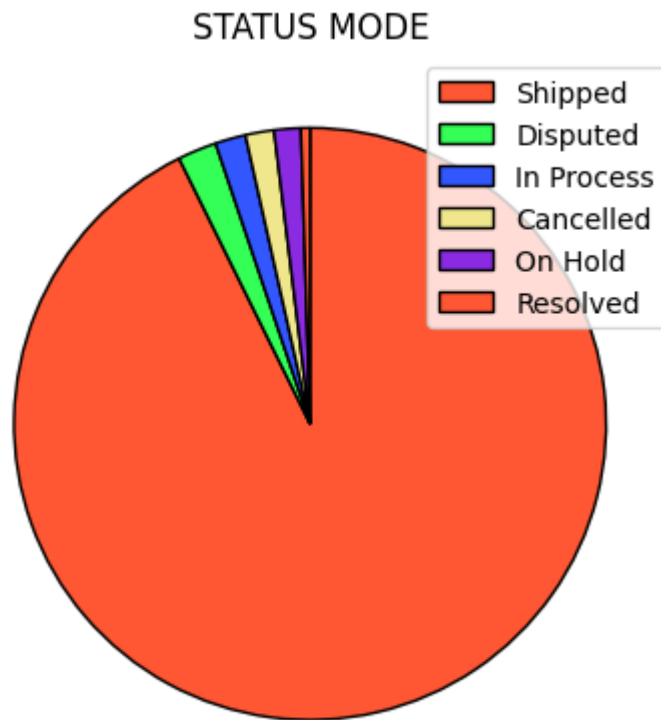
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.countplot(x = df['DEALSIZE'],palette=['green', 'red', 'purple'])
```



```
In [56]: # Moderate Deal Size Leading to High Revenue
```

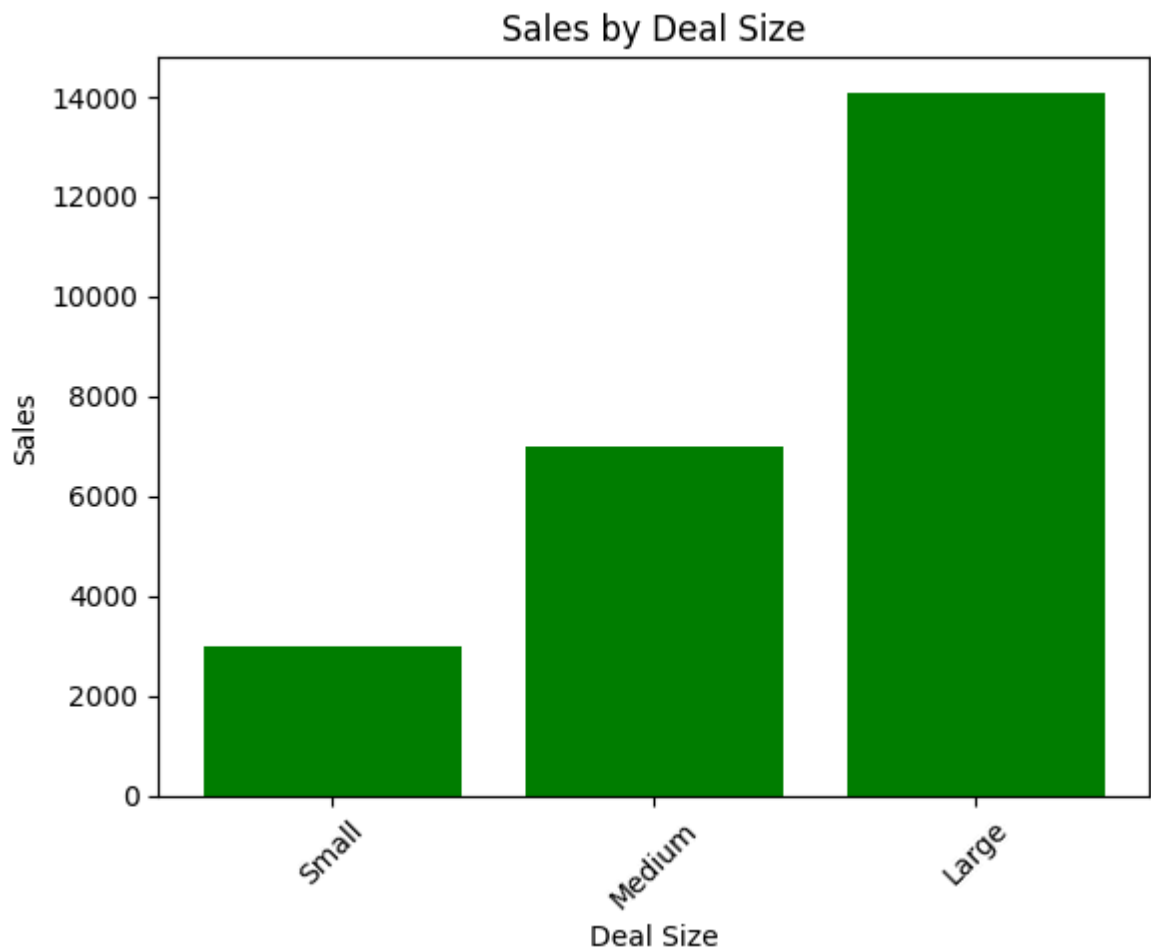
```
In [75]: plt.pie(df['STATUS'].value_counts(),startangle=90,counter-clockwise=False,wedgeprops=  
)  
plt.title("STATUS MODE")  
plt.legend(df['STATUS'].unique())  
plt.show()
```



```
In [58]: # shipped mode have best performance
```

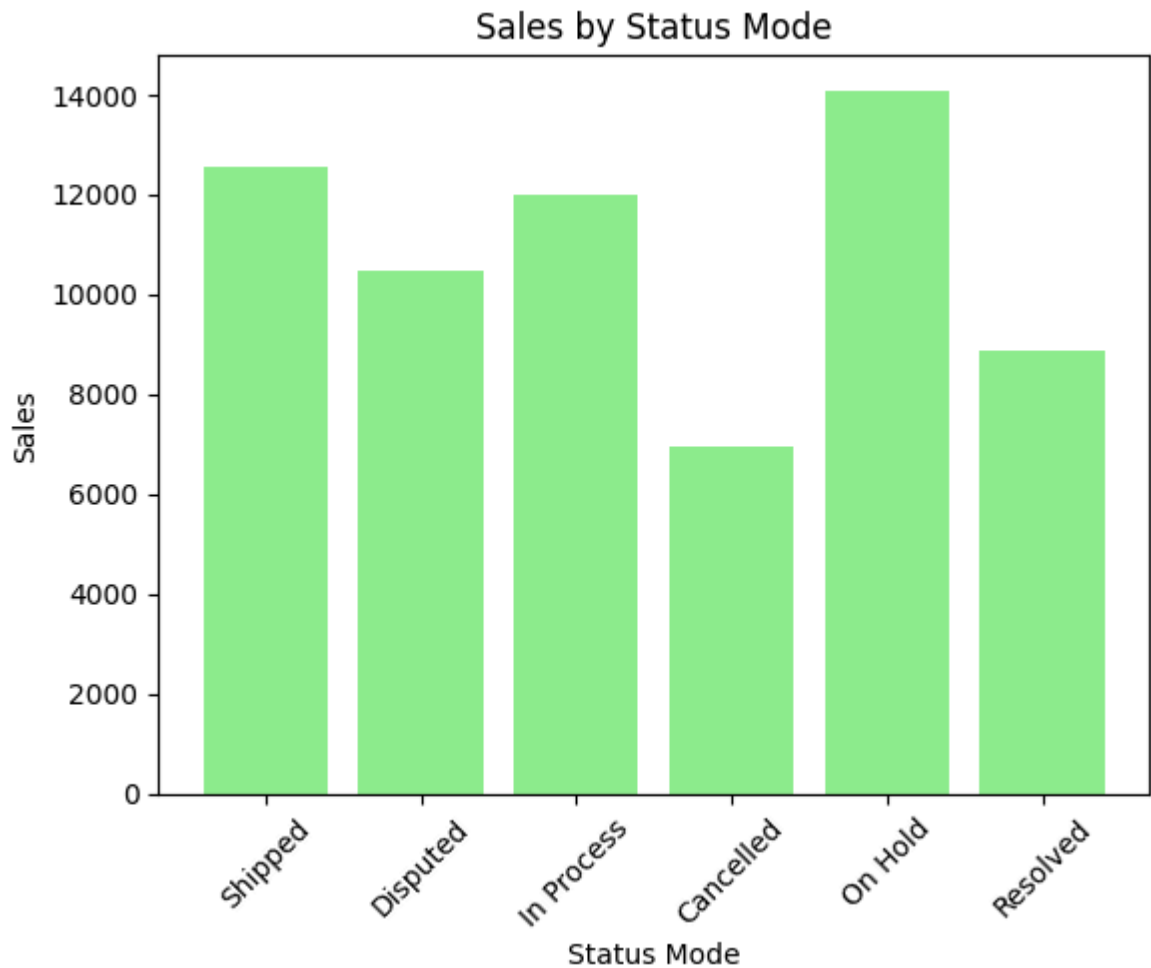
Bivariate Analysis: Methods, Techniques, and Applications

```
In [76]: plt.bar(df['DEALSIZE'], df['SALES'],color='green')
plt.xlabel('Deal Size')
plt.ylabel('Sales')
plt.title('Sales by Deal Size')
plt.xticks(rotation=45)
plt.show()
```



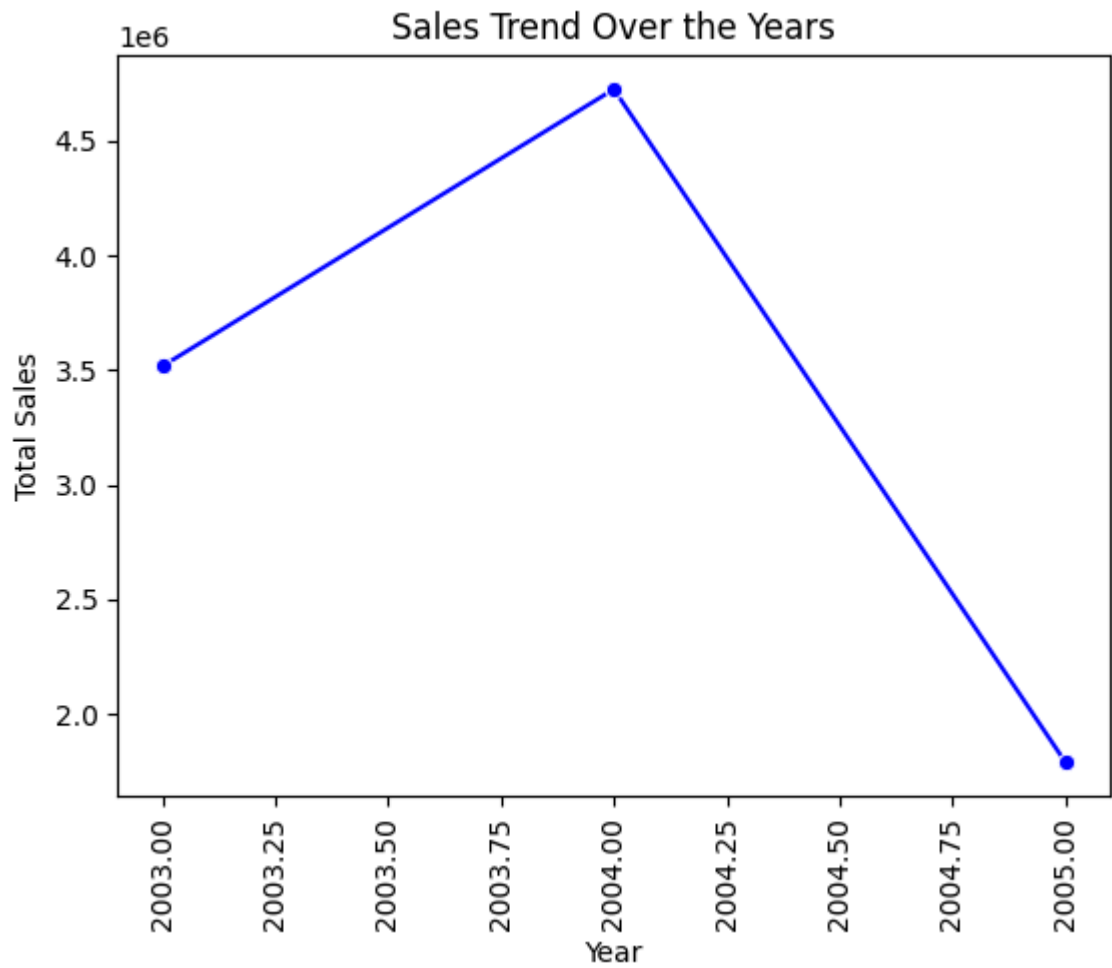
```
In [60]: # Large deal size have high sales
```

```
In [77]: plt.bar(df['STATUS'], df['SALES'],color='lightgreen')
plt.xlabel('Status Mode')
plt.ylabel('Sales')
plt.title('Sales by Status Mode')
plt.xticks(rotation=45)
plt.show()
```



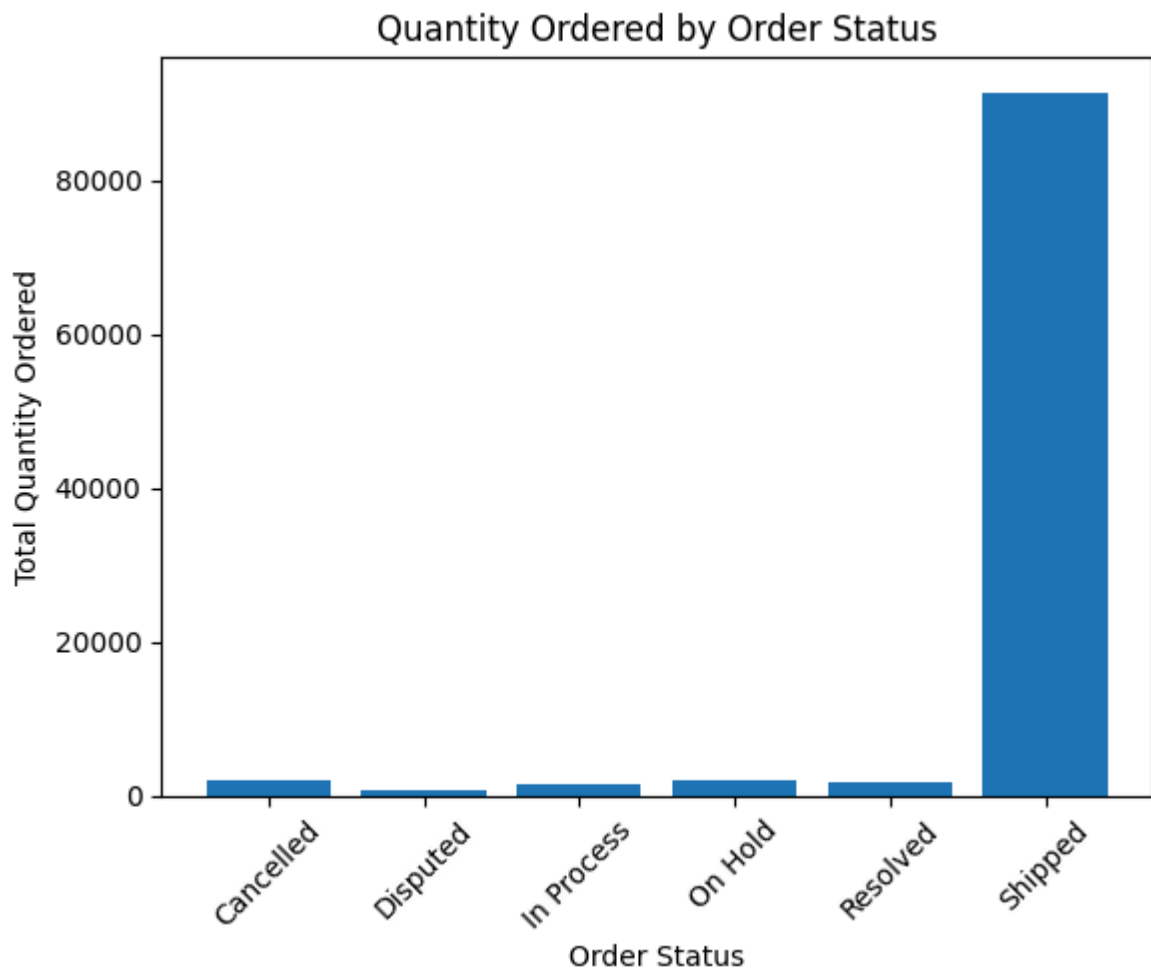
In [62]: *# in 2005 year company generate high revenue*

```
In [63]: yearly_sales = df.groupby('YEAR')['SALES'].sum().reset_index()
sns.lineplot(x=yearly_sales['YEAR'], y=yearly_sales['SALES'], marker='o', color=
plt.xlabel('Year')
plt.xticks(rotation=90)
plt.ylabel('Total Sales')
plt.title('Sales Trend Over the Years')
plt.show()
```



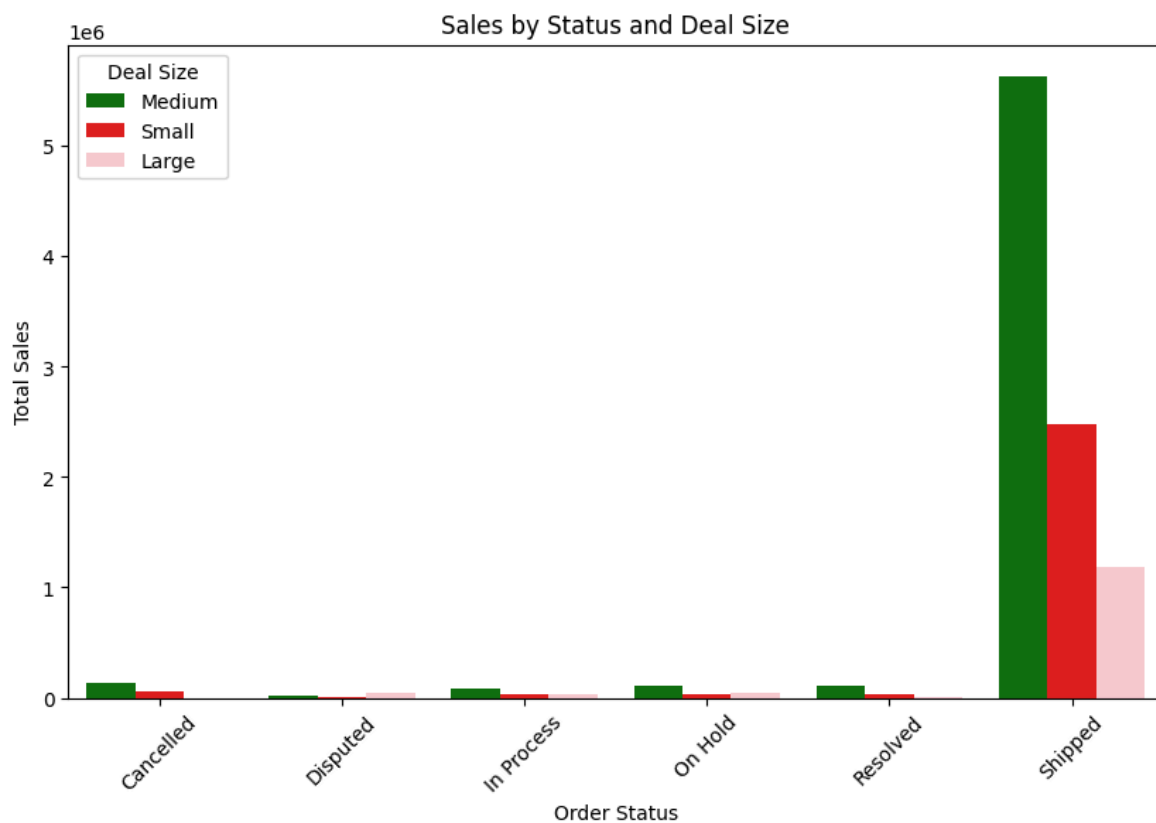
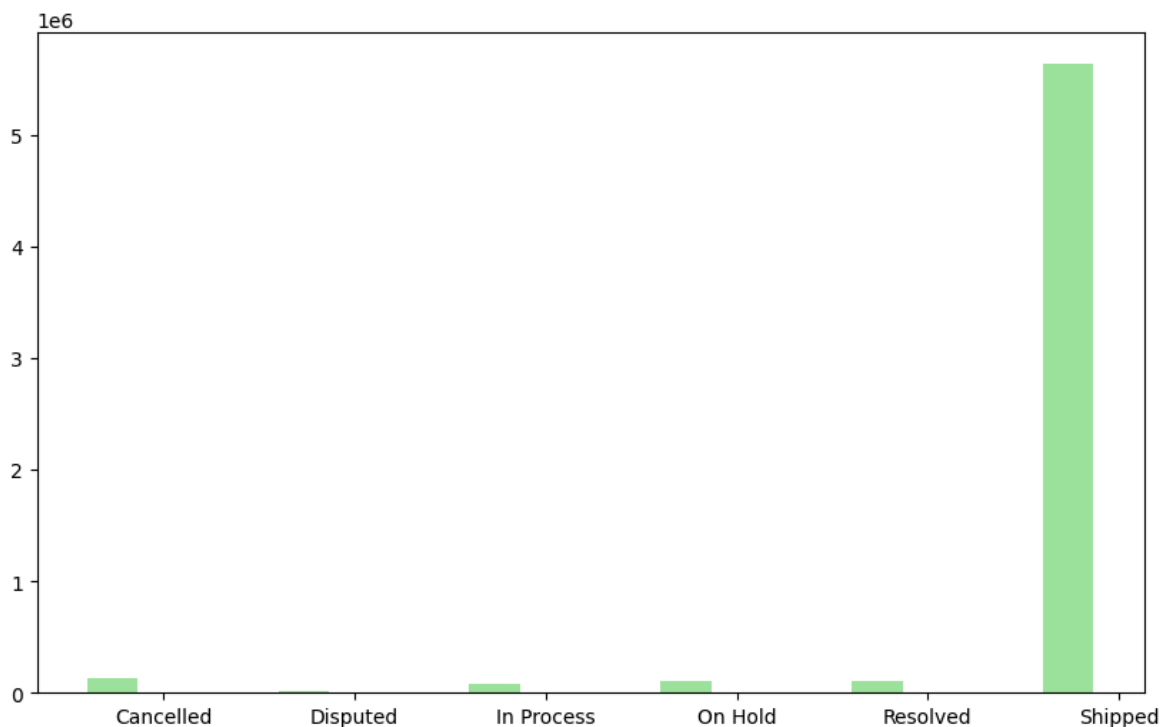
```
In [64]: status_quantity = df.groupby('STATUS')['QUANTITYORDERED'].sum()
plt.bar(status_quantity.index, status_quantity.values)

plt.xlabel('Order Status')
plt.ylabel('Total Quantity Ordered')
plt.title('Quantity Ordered by Order Status')
plt.xticks(rotation=45)
plt.show()
```

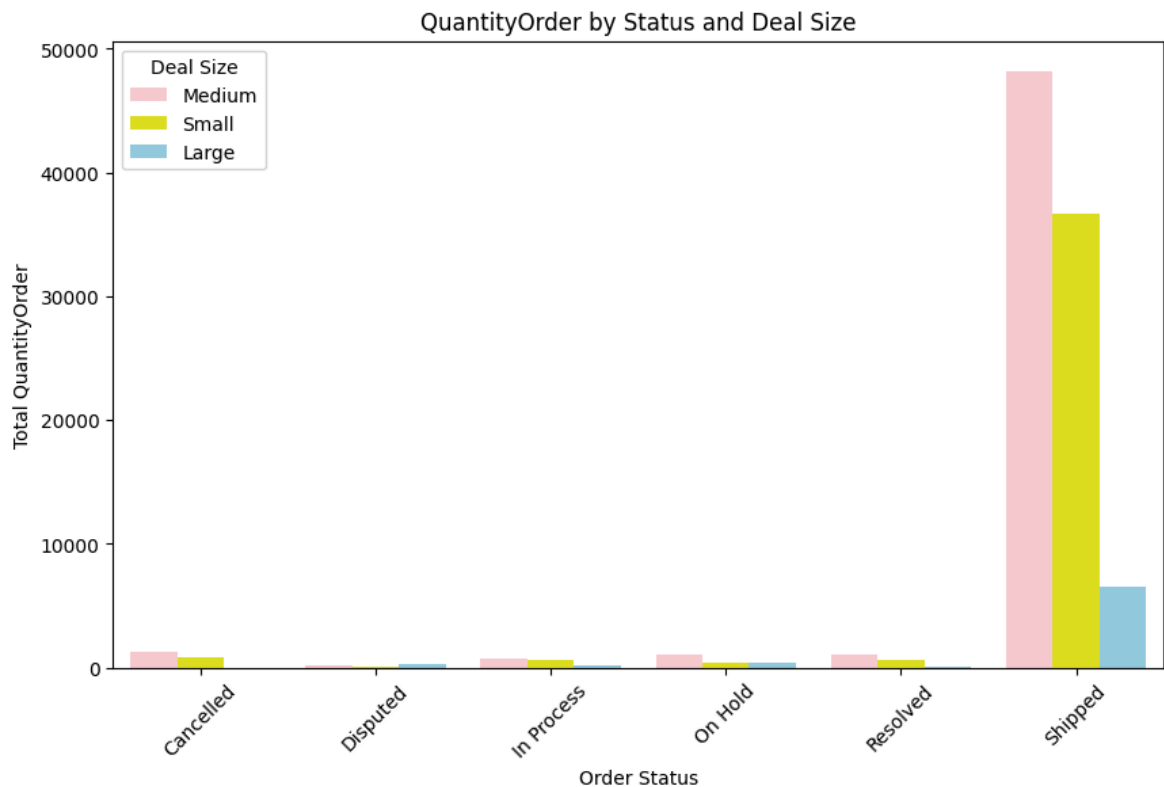
```
In [65]: ## multivariate analysis
```

```
In [79]: status_dealsize_sales = df.groupby(['STATUS', 'DEALSIZE'])['SALES'].sum().reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(x='STATUS', y='SALES', hue='DEALSIZE', data=status_dealsize_sales, palette='magma')
plt.xlabel('Order Status')
plt.ylabel('Total Sales')
plt.title('Sales by Status and Deal Size')
plt.xticks(rotation=45)
plt.legend(title='Deal Size')
plt.show()
```



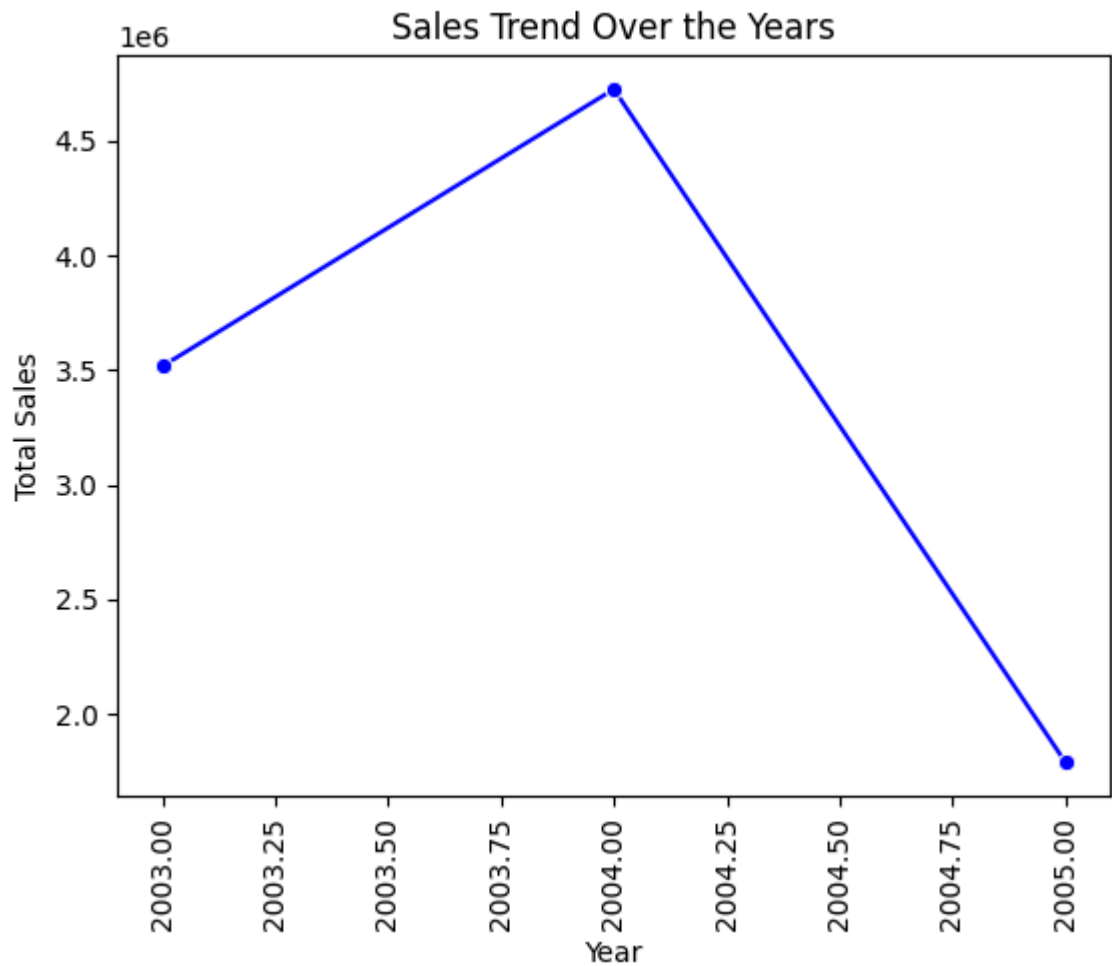
```
In [67]: # In shipped mode medium dealsize performance is good
```

```
In [68]: status_dealsize_QuantityOrder = df.groupby(['STATUS', 'DEALSIZE'])['QUANTITYORDERED'].sum()
plt.figure(figsize=(10, 6))
sns.barplot(x='STATUS', y='QUANTITYORDERED', hue='DEALSIZE', data=status_dealsize_QuantityOrder)
plt.xlabel('Order Status')
plt.ylabel('Total QuantityOrdered')
plt.title('QuantityOrder by Status and Deal Size')
plt.xticks(rotation=45)
plt.legend(title='Deal Size')
plt.show()
```



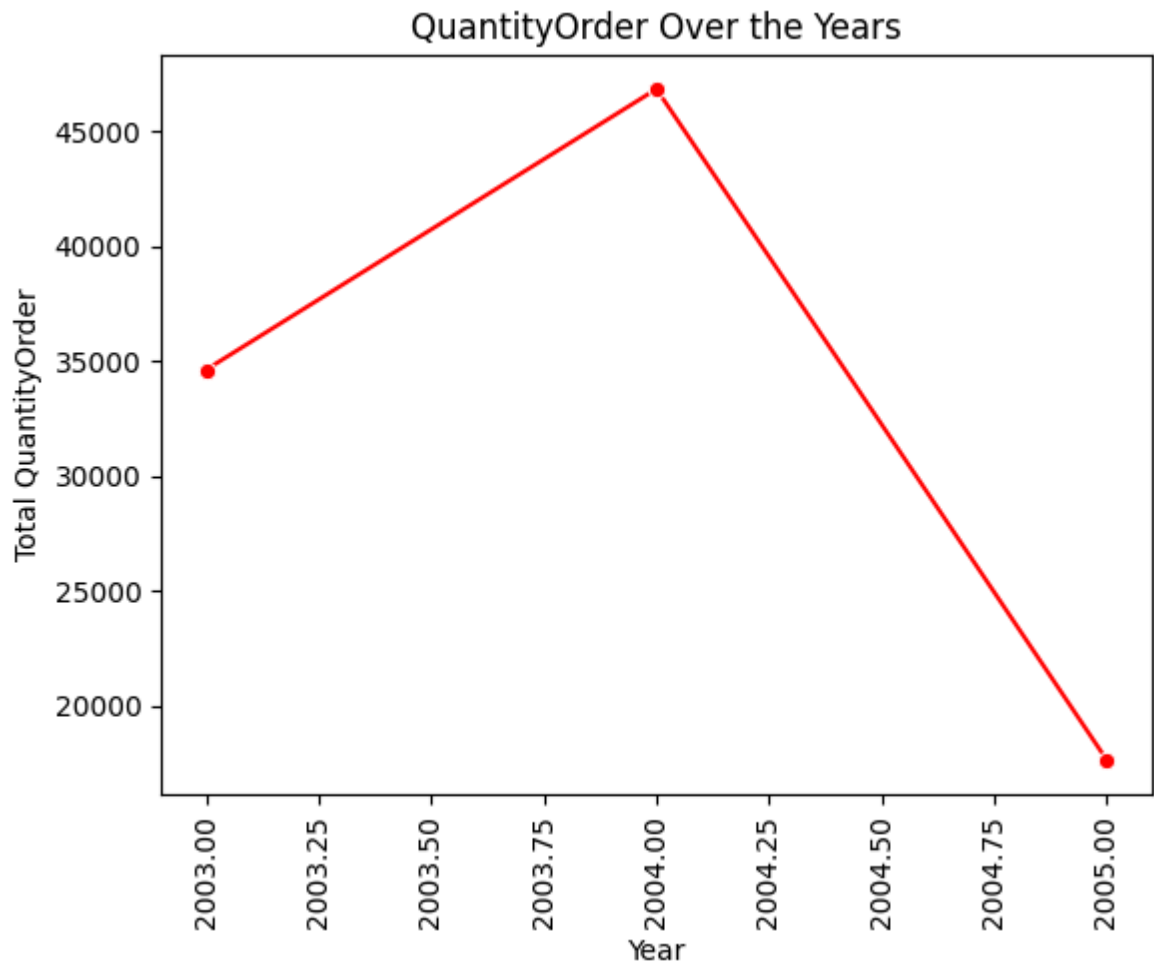
Sales Analytics: Key Trends, Seasonal Patterns, and Top Products

```
In [69]: yearly_sales = df.groupby('YEAR')['SALES'].sum().reset_index()
sns.lineplot(x=yearly_sales['YEAR'], y=yearly_sales['SALES'], marker='o', color=
plt.xlabel('Year')
plt.xticks(rotation=90)
plt.ylabel('Total Sales')
plt.title('Sales Trend Over the Years')
plt.show()
```



```
In [70]: # sales fist increase after that it deacrese overe year
```

```
In [71]: yearly_sales = df.groupby('YEAR')['QUANTITYORDERED'].sum().reset_index()
sns.lineplot(x=yearly_sales['YEAR'], y=yearly_sales['QUANTITYORDERED'], marker='o')
plt.xlabel('Year')
plt.xticks(rotation=90)
plt.ylabel('Total QuantityOrder')
plt.title('QuantityOrder Over the Years')
plt.show()
```



Thank you for this wonderful opportunity. I truly appreciate it!

In []: