Online Business Sales Time Series Analysis

February 8, 2024

1 Import Libraries

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
import itertools
from sklearn.metrics import mean_squared_error, mean_absolute_error
from statsmodels.tsa.arima.model import ARIMA
import warnings
warnings.filterwarnings("ignore")
```

2 EDA

[4]:
```
df_date_sales = pd.read_csv('business.retailsales2.csv')
```

[5]:
```
df_date_sales.head()
```

[5]:
```
       Month  Year  Total Orders  Gross Sales  Discounts  Returns  Net Sales  \
0    January  2017            73       8861.5    -129.40  -448.45    8283.65
1   February  2017            56       6908.5    -104.70  -416.20    6387.60
2      March  2017            60       5778.5    -172.20 -1017.20    4589.10
3      April  2017            70       8814.0    -281.40     0.00    8532.60
4        May  2017            54       6677.0    -185.75  -253.80    6237.45

   Shipping  Total Sales
0   1088.30      9371.95
1    892.45      7280.05
2    707.43      5296.53
3   1068.30      9600.90
4    866.46      7103.91
```

[6]:
```
df_date_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Month         36 non-null     object
 1   Year          36 non-null     int64
 2   Total Orders  36 non-null     int64
 3   Gross Sales   36 non-null     float64
 4   Discounts     36 non-null     float64
 5   Returns       36 non-null     float64
 6   Net Sales     36 non-null     float64
 7   Shipping      36 non-null     float64
 8   Total Sales   36 non-null     float64
dtypes: float64(6), int64(2), object(1)
memory usage: 2.7+ KB
```

[7]:

```
df_date_sales.describe()
```

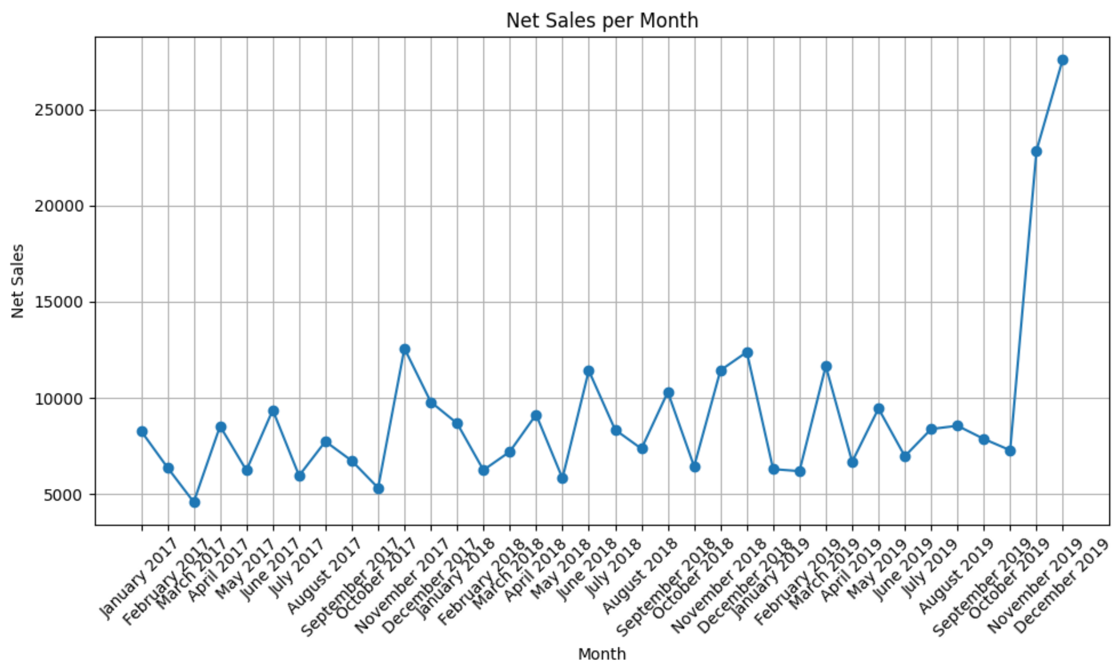
[7]:

```
              Year  Total Orders   Gross Sales    Discounts      Returns  \
count    36.000000     36.000000     36.000000    36.000000    36.000000
mean   2018.000000     97.138889   9844.926389  -311.493889  -474.958056
std       0.828079     57.458632   4936.386351   362.766989   488.820410
min    2017.000000     54.000000   5720.000000 -2269.510000 -1572.550000
25%    2017.000000     68.000000   7059.875000  -300.375000  -867.200000
50%    2018.000000     82.500000   8850.500000  -236.160000  -299.875000
75%    2019.000000     97.500000  10150.700000  -169.487500   -73.277500
max    2019.000000    342.000000  31183.900000   -51.500000     0.000000

          Net Sales     Shipping   Total Sales
count     36.000000    36.000000     36.000000
mean    9058.474444  1579.391667  10637.941111
std     4497.185264  1011.170014   5475.621125
min     4589.100000   695.420000   5296.530000
25%     6428.250000  1083.300000   7633.692500
50%     8076.430000  1341.650000   9404.405000
75%     9534.000000  1632.132500  11153.687500
max    27603.210000  5703.250000  33306.460000
```

[8]:

```
#Net Sales per Month/Year

# Line chart for net sales per month
plt.figure(figsize=(10, 6))
plt.plot(df_date_sales['Net Sales'], marker='o', linestyle='-')
plt.title('Net Sales per Month')
plt.xlabel('Month')
plt.ylabel('Net Sales')
```

```python
plt.xticks(ticks=df_date_sales.index, labels=df_date_sales['Month'] + ' ' + df_date_sales['Year'].astype(str), rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



- Is there a general increasing, decreasing or constant trend in sales over time?

It appears as if there was a constant trend in sales over time, with no clear pattern of growth or decline.

- Is there seasonality in sales? What months or periods of the year have higher or lower sales?

If we inspect January over the years, we can notice that the sales stayed below 10000 from 2017, and continued to decrease in the following years.

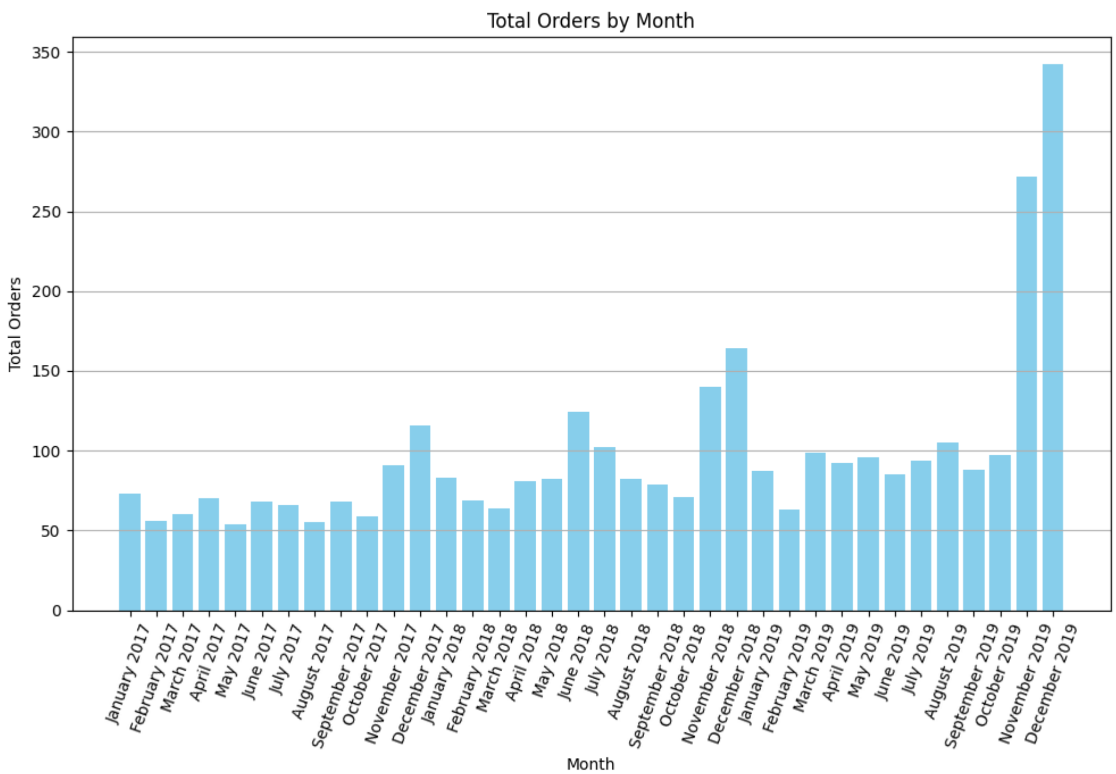
- Can peaks or valleys be identified that coincide with specific company events or actions?
  - From September to October 2019 there is an increasing peak.
  - From December 2018 to January 2019 there is a steady period of sales of approximately 600.

- Is there a point in time where there is a sharp change or a noticeably different trend in sales?

October 2019.

[10]:
```python
#Total orders per Month/Year

# Bar chart for total orders per month
```

```python
plt.figure(figsize=(10, 7))
plt.bar(df_date_sales.index, df_date_sales['Total Orders'], color='skyblue')
plt.title('Total Orders by Month')
plt.xlabel('Month')
plt.ylabel('Total Orders')
plt.xticks(ticks=df_date_sales.index, labels=df_date_sales['Month'] + ' ' + df_date_sales['Year'].astype(str), rotation=70)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



- Which months have the highest number of orders and which have the lowest numbers?

Highest: October 2019; November 2019; December 2018 Lowest: May 2017; August 2017

- Is there any seasonal pattern that repeats year after year in terms of the number of orders?

August 2019 and July 2018 look quite similar as well as January 2018 and 2019.

- Can any long-term trends in the number of orders be identified? Has there been a steady increase or decrease over time?

Over the years it seems to have remained fairly constant and with not so high values generally averaging 100. Except for November 2019 and December 2019 with peaks.

- Are there any specific events or activities that may have affected the number of orders in certain months or years?

N/A. Need more business and context information.

[12]:
```
# Discounts, returns and shipments per month/year

# Line chart for discounts, returns and shipping per month
plt.figure(figsize=(10, 6))
sns.lineplot(data=df_date_sales[['Discounts', 'Returns', 'Shipping']])
plt.title('Discounts, Returns and Shipping by Month')
plt.xlabel('Month')
plt.ylabel('Quantity')
plt.xticks(ticks=df_date_sales.index, labels=df_date_sales['Month'] + ' ' +␣
↪df_date_sales['Year'].astype(str), rotation=45)
plt.legend(['Discounts', 'Returns', 'Shipping'])
plt.grid(True)
plt.tight_layout()
plt.show()
```

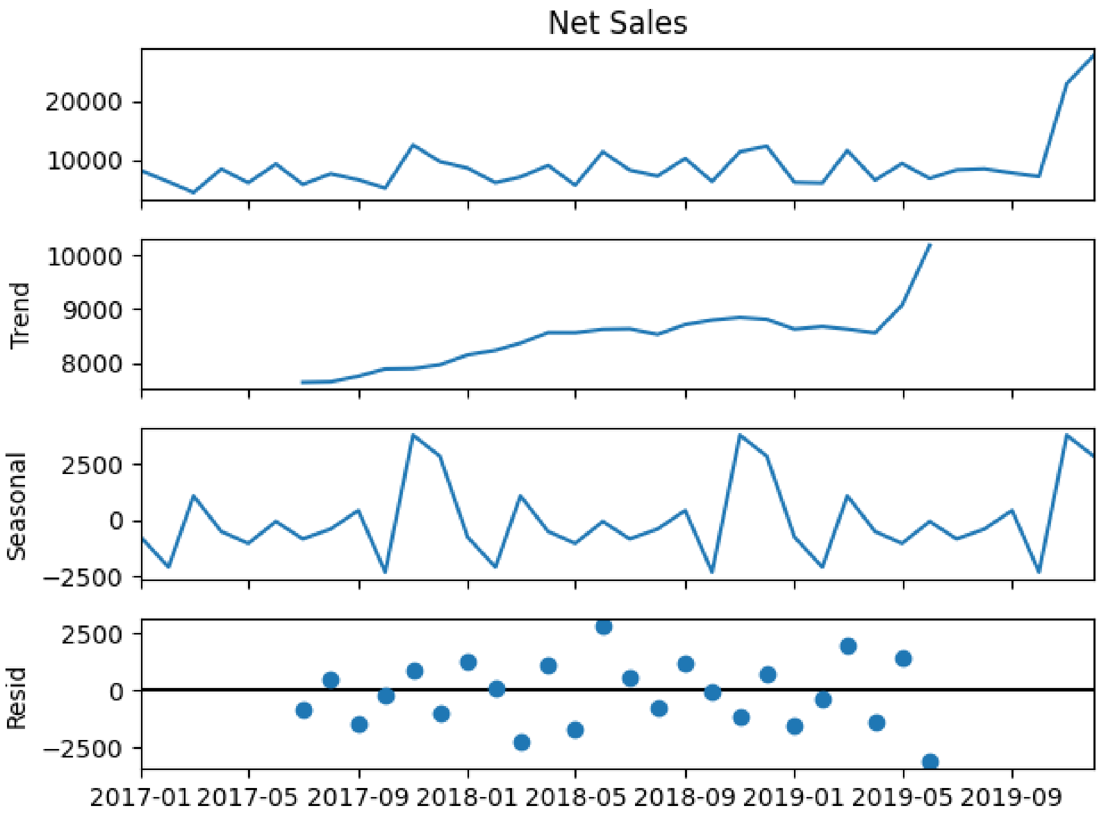
- Are there any months with significant spikes in discounts, returns or shipments? Is there a seasonal reason or event that could explain these spikes?
  - November 2019, drop in discounts.
  - September to October 2019, spike in returns.
  - March 2018 to April 2018, drop in shipments.
- Is there any consistent pattern observed in the relationship between discounts, returns and

shipments over time?

No

[14]:
```python
# Check the data types of the columns 'Month' and 'Year'.
print(df_date_sales[['Month', 'Year']].dtypes)

# Convert 'Year' to string
df_date_sales['Year'] = df_date_sales['Year'].astype(str)

# Check data types again
print(df_date_sales[['Month', 'Year']].dtypes)

# Convert 'Month' to datetime format using a mapping
month_mapping = {'January': '01', 'February': '02', 'March': '03', 'April':␣
↪'04', 'May': '05', 'June': '06',
                 'July': '07', 'August': '08', 'September': '09', 'October':␣
↪'10', 'November': '11', 'December': '12'}

df_date_sales['Month'] = df_date_sales['Month'].map(month_mapping)

# Create a 'Date' column combining 'Month' and 'Year'.
df_date_sales['Date'] = pd.to_datetime(df_date_sales['Year'] + '-' +␣
↪df_date_sales['Month'] + '-01')

# Check the data types again
print(df_date_sales[['Month', 'Year', 'Date']].dtypes)

# Create a time series with 'Date' as the index and 'Net Sales' as the values
ts = df_date_sales.set_index('Date')['Net Sales']

# Time Serie Decomposition
decomposition = sm.tsa.seasonal_decompose(ts, model='additive', period=12)

# Plot graphic
decomposition.plot()
plt.show()
```

```
Month    object
Year      int64
dtype: object
Month    object
Year     object
dtype: object
Month            object
Year             object
Date     datetime64[ns]
dtype: object
```



- How does the trend in net sales behave over time? Is there a general growth or decline?

It's increasing.

- Are recurring seasonal patterns in net sales identified? What are the months or time periods where seasonal peaks or troughs are observed?

It appears as if in September of the three years being analyzed, net sales have declines.

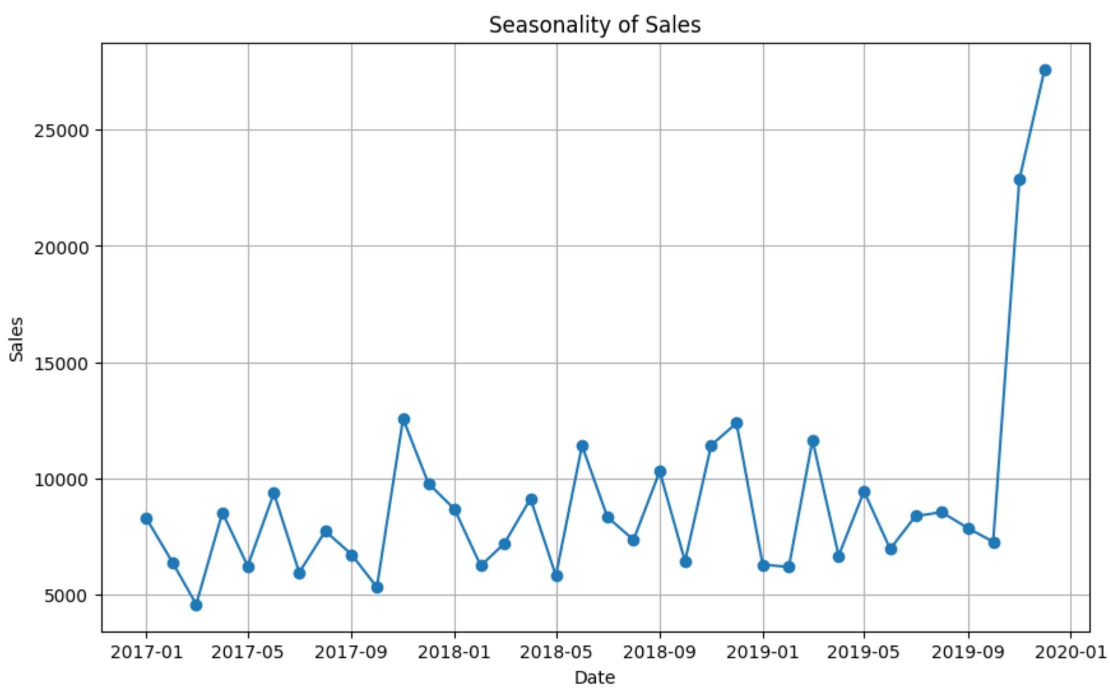
- Do the residuals show any pattern or are they random?

The residuals appear to be random and with a constant variance.

[16]:
```
# Convert Month and Year to Datetime
df_date_sales['Date'] = pd.to_datetime(df_date_sales['Month'] + ' ' +␣
↪df_date_sales['Year'].astype(str))

# Establish Date as index
df_date_sales.set_index('Date', inplace=True)

# Create time serie with relevant data
ts = df_date_sales['Net Sales']
```

```python
# Time Serie decomposition
decomposition = sm.tsa.seasonal_decompose(ts, model='additive', period=12)

# Seasonality chart
plt.figure(figsize=(10, 6))
plt.plot(date_sales_index, df_date_sales['Net Sales'], marker='o',␣
 ↪linestyle='-')
plt.title('Seasonality of Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.grid(True)
plt.show()
```

```
Month              object
Year               object
Date       datetime64[ns]
dtype: object
```



- Are there regular peaks or lows in sales at specific times of the year or month? When do they occur and why?

The lowest point appears to be in March 2017 or so, and the highest in December 2019.

- Do seasonalities maintain consistent patterns across years or months? Are there noticeable changes in seasonality over time?

Through September 2017 and from early 2019 through September 2017, there appears to be a consistent pattern in that sales remain below 10000. However, from September 2017 through early 2019, some sales were above 10000, approximately 12000.

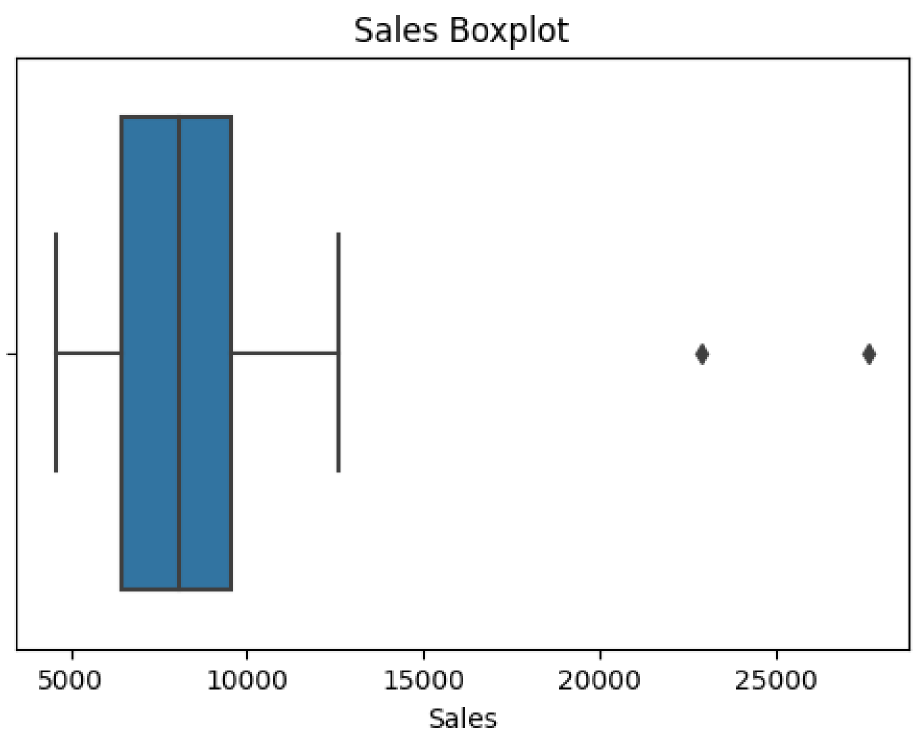
[18]:
```
# Correlation between Variables:
# Let's calculate the correlation matrix to observe how the different metrics␣
↪are related.

# Correlation matrix
correlation_matrix = df_date_sales[['Total Orders', 'Gross Sales', 'Discounts',␣
↪'Returns', 'Net Sales', 'Shipping', 'Total Sales']].corr()

# Visualization of the correlation matrix using a heat map
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
↪linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

Interpretation of the Correlation Matrix: Colors and Values: The colors and values in each cell indicate the strength and direction of the relationship between the variables. Values range from -1 to 1, where 1 represents a perfect positive correlation, -1 a perfect negative correlation and 0 no correlation.

Strongly Correlated Variables: Cells with values close to 1 or -1 represent a strong relationship. Positive values close to 1 indicate a positive correlation, while negative values close to -1 indicate a negative correlation.

- Total orders with total sales shows a strong correlation. With increase in sales, increase in orders.
- Shipping with total sales shows strong correlation.
- Net sales with gross sales shows strong correlation.
- Discounts with total sales shows a negative correlation. As there are more discounts, one would tend to analyze the relationship as decreasing in sales.
- Shipping with returns a -0.32 correlation.

[20]:
```
# Boxplot for sales
plt.figure(figsize=(6, 4))
sns.boxplot(x=df_date_sales['Net Sales'])
plt.title('Sales Boxplot')
plt.xlabel('Sales')
plt.show()
```



Boxplot interpretation: Box: The box represents the interquartile range (IQR) of the data set, where the central 50% of the data is within the box. The line in the center of the box is the median.

Whiskers: Whiskers show the variability outside of the interquartile range. They can represent possible outliers or extremes.

Points Outside the Whiskers: These are potential outliers, also known as "outliers".

We only seem to have 2 outliers, so I would choose as a strategy to delete them from the analysis so they don't disturb the final predictions.

[22]:
```
#Statistical Analysis:
#Calculate statistics such as interquartile range (IQR) and define a threshold␣
↪to identify outliers.

#For example, to identify outliers in sales.
q1 = df_date_sales['Net Sales'].quantile(0.25)
q3 = df_date_sales['Net Sales'].quantile(0.75)
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

outliers = df_date_sales[(df_date_sales['Net Sales'] < lower_bound) |␣
↪(df_date_sales['Net Sales'] > upper_bound)]
print(outliers)
```

```
           Month  Year  Total Orders  Gross Sales  Discounts  Returns  \
Date
2019-11-01    11  2019           272      23997.9    -776.84  -364.51
2019-12-01    12  2019           342      31183.9   -2269.51 -1311.18

            Net Sales  Shipping  Total Sales
Date
2019-11-01   22856.55   4824.75     27681.30
2019-12-01   27603.21   5703.25     33306.46
```

We obtained two rows as outliers for November and December 2019. These 'Net Sales' values are above the upper limit calculated from the IQR. In this case, sales in those months were significantly higher than most other observations, indicating outlier or unusual behavior compared to the rest of the data.

3 Predictions

[25]:
```
# Division into training and test data
train_size = int(len(df_date_sales) * 0.75)
train, test = df_date_sales[:train_size], df_date_sales[train_size:]
```

[26]:
```
print(train['Month'].unique())
```

```
['01' '02' '03' '04' '05' '06' '07' '08' '09' '10' '11' '12']
```

[27]:

```python
# Create a time series with 'Month' as the index and 'Total Orders' as the␣
↪values
time_series = df_date_sales.set_index('Month')['Total Orders']

# Define the ranges of values for p, d, q
p_values = range(0, 6)
d_values = range(0, 3)
q_values = range(0, 6)

# Create all possible combinations of p, d, q
param_combinations = list(itertools.product(p_values, d_values, q_values))

# Initialize variables to store the results
best_aic = float('inf')
best_params = None

# Iterate over all combinations
for param in param_combinations:
    try:
        # Fit ARIMA model with current parameters.
        model = sm.tsa.ARIMA(time_series, order=param)
        results = model.fit()

        # Compute the Akaike information criterion (AIC)
        aic = results.aic

        # Update the best parameters if we find a combination with a lower AIC
        if aic < best_aic:
            best_aic = aic
            best_params = param
    except Exception as e:
        continue


print("Best parameters found:", best_params)
print("Best AIC value:", best_aic)

# Fit the ARIMA model with the best found parameters
order = best_params # Use best parameters found
model = ARIMA(time_series, order=order)
model_fit = model.fit()

# Make predictions for the next 6 months
predictions = model_fit.forecast(steps=6)
```

```python
# Display the predictions
print("ARIMA predictions for the next 6 months:")
print(predictions)
```

```
Best parameters found: (4, 2, 2)
Best AIC value: 351.9888317870625
ARIMA predictions for the next 6 months:
36    307.134200
37    356.143900
38    450.784146
39    525.128746
40    609.364497
41    678.190399
Name: predicted_mean, dtype: float64
```

[28]:
```python
# Make sure 'test' and 'predictions' have the same length
test = test[:len(predictions)]

# Calculate Mean Absolute Percentage Error (MAPE) only for 'Total Orders'.
mape = mean_absolute_error(test['Total Orders'], predictions)
print("MAPE:", mape)
```

```
MAPE: 394.4576479592572
```