Optimizing Retail Forecasting

Capturing Cyclical Trends with Advanced SARIMA Models

Nicolas Jorquera
8-13-2023

Table of Contents

Introduction

The project was initiated to address the challenges faced in time series forecasting, specifically aiming to enhance the prediction of certain trends or cycles. By conducting comprehensive Time Series Analysis; and comparing various models we can determine why are useful in handling inconsistencies, abrupt changes, and the underlying weekly patterns that were inherent to the data.

The significance of this project lies in its potential to provide more accurate and reliable forecasts. By employing different modeling techniques, including log transformations and outlier handling, we aimed to capture the inherent complexities within the data. This could lead to better decision-making in various applications such as inventory management, financial forecasting, or scheduling.

In this project we focused on the daily sales data from a Superstore; however their were various ways to approach this dataset. However; to adequately train ARIMA and SARIMA models to evaluate sales performance it was necessary to aggregate data to a daily frequency. This would allow us to create a curated plan for the Store Manager in order to increase revenue.

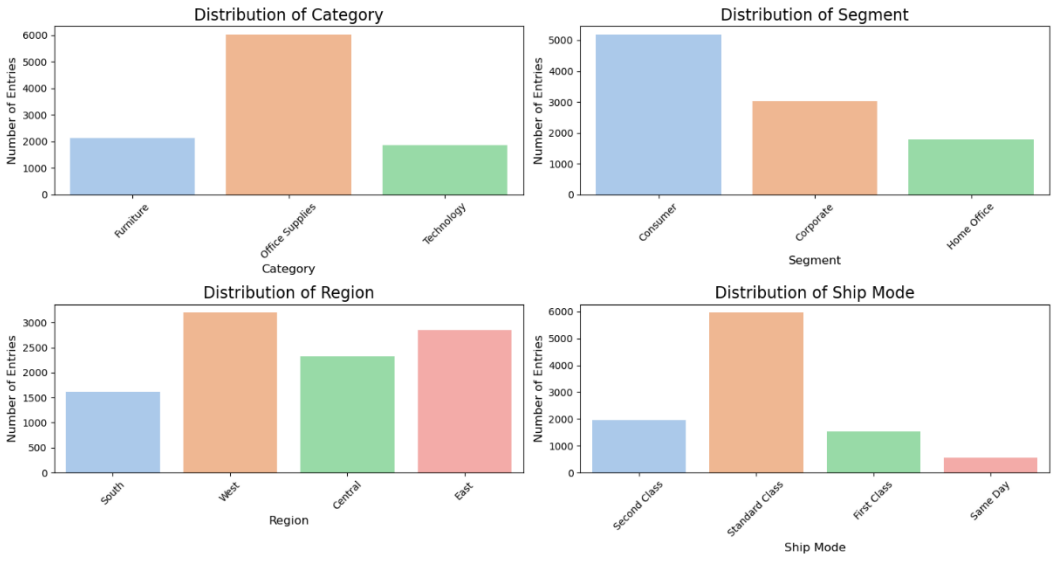
Exploratory Data Analysis

EDA is a fundamental step in the data analysis process. It involves visualizing, summarizing, and interpreting the information that is hidden in rows and columns of data. The main goal of EDA is to understand the data, find patterns, spot anomalies, test hypotheses, and check assumptions.

Data Visualization

Distribution of Sub-Categories

Visualizing the distribution of data features helps in understanding the underlying structure and pattern of the data. It reveals aspects such as skewness or the presence of any outliers.



From the bar plots; we can see that:

Category: "Office Supplies" has the highest number of entries, followed by "Furniture" and "Technology".

Segment: "Consumer" leads, with "Corporate" and "Home Office" following.

Region: The "West" region has the most entries, followed by "East", "Central", and "South". Of all the categories; the distribution of Region is the most evenly distributed.

Ship Mode: "Standard Class" is the most common shipping mode, with "Second Class", "First Class", and "Same Day" in descending order.

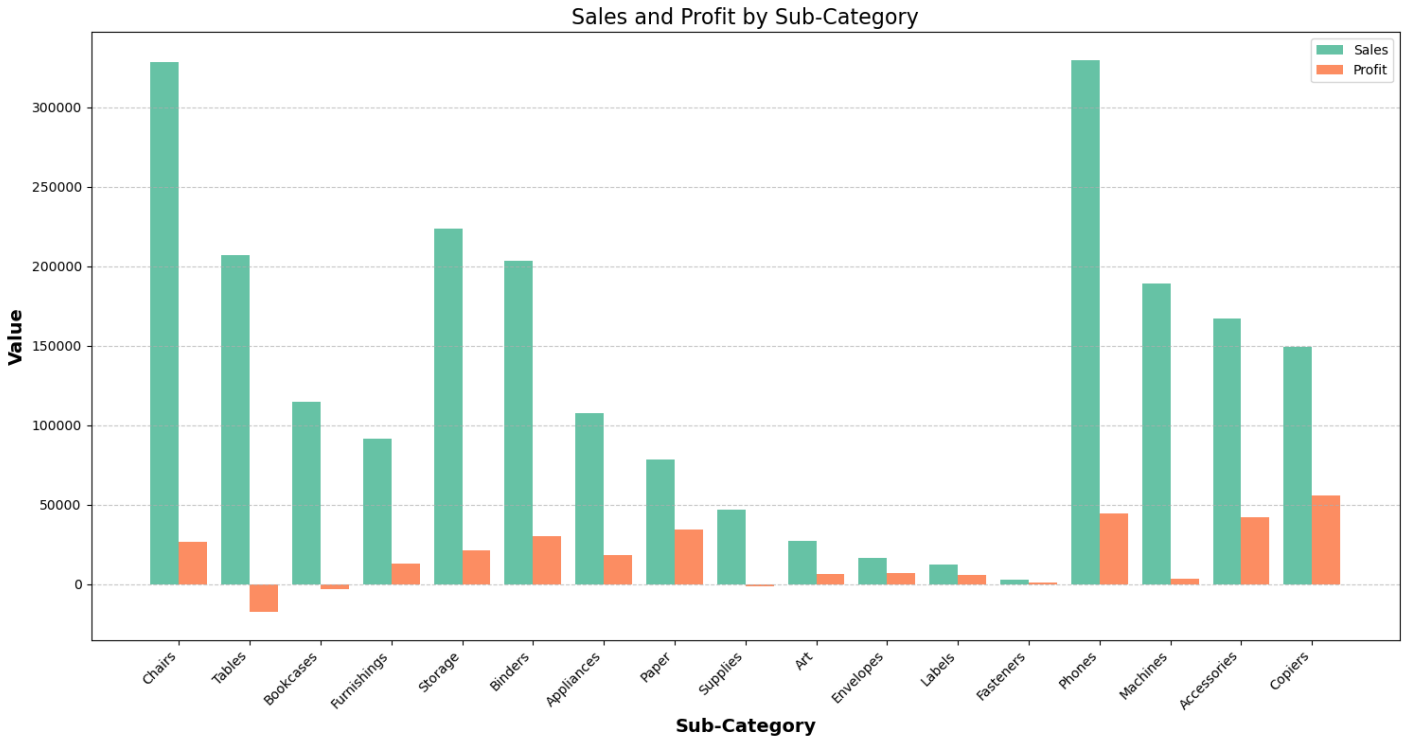
Distribution of Sub-Categories

Analyzing the distribution of Sub-Categories provides insights into the product mix, popularity of different products, and potential market opportunities or challenges. It allows businesses to understand customer preferences and align their strategies accordingly.

Analysis of Sales and Profit by Sub-Category:

Sales: From the graph, we can identify the Sub-Categories with the highest sales. It appears that 'Chairs', 'Tables', 'Phones' and 'Storage' are among the top performers in terms of sales.

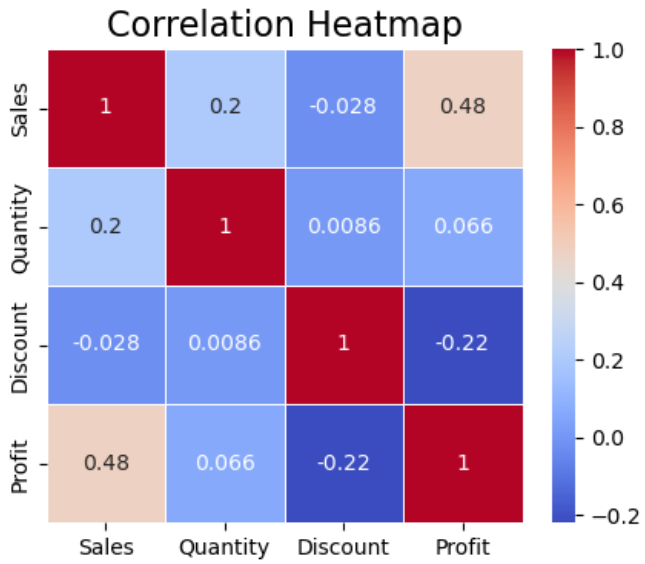
Profitability: Not all high sales correspond to high profits. For example, while 'Tables' has significant sales, it shows negative profit, indicating a loss in that Sub-Category. Furthermore; high sales also don't yield the highest profit; with Copiers and Accesories having the same profit as Phones; but selling significantly less. However; Electronics clearly have the highs Sales to Profit ratio; followed by Office Supplies and Furniture.



Future Investigation: Some Sub-Categories like 'Bookcases' and 'Furnishings' have relatively lower sales but have profits. It would be interesting to investigate why some Sub-Categories are more profitable than others despite lower sales.

Correlation Maps

A correlation heatmap helps us understand the relationships between numerical columns. Correlation values range from -1 to 1, where -1 indicates a strong negative relationship, 1 indicates a strong positive relationship, and 0 indicates no relationship.



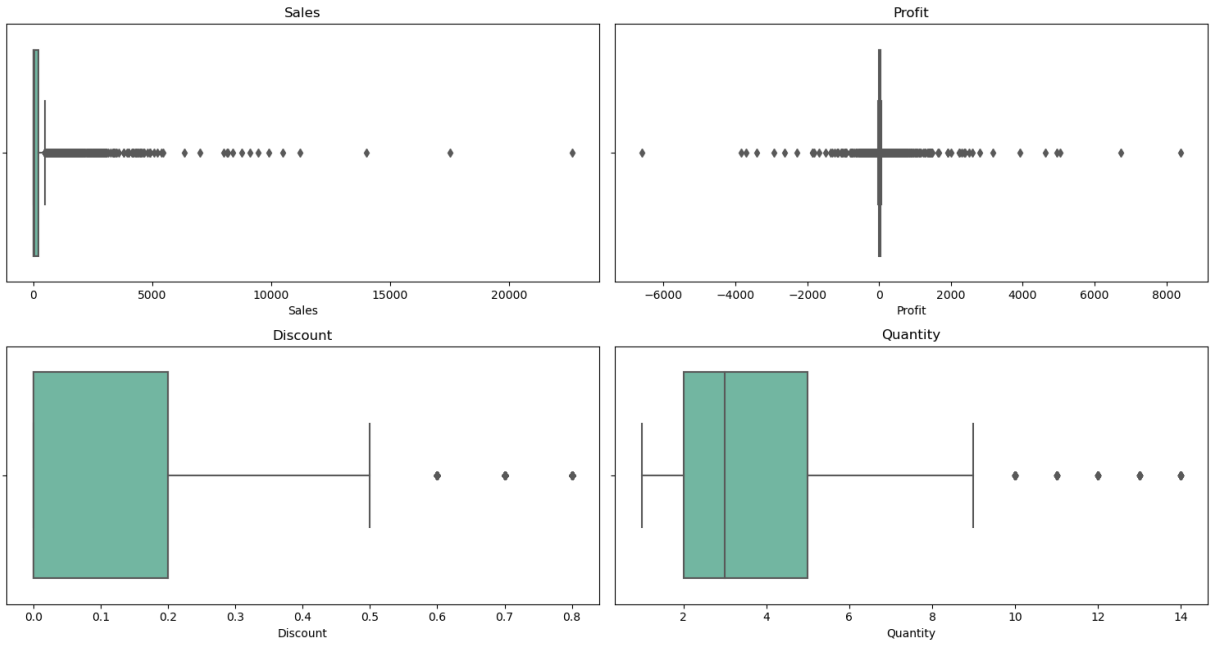
Analysis of Correlation Heatmap:

Sales and Profit: There is a positive correlation between sales and profit, indicating that higher sales generally lead to higher profits.

Discount and Profit: There is a negative correlation between discount and profit, suggesting that higher discounts might lead to lower profits.

Sales and Quantity: A positive correlation between sales and quantity shows that higher quantities sold are associated with higher sales.

Analysis of Outliers



Analysis of Outliers for Sales, Profit, Discount, and Quantity:

Sales: The majority of sales lie within a relatively small range, but there are several outliers, indicating some unusually high sales values.

Profit: There are outliers on both sides, suggesting instances of both unusually high profits and significant losses.

Discounts: Most of the data is clustered around lower discount values, with some outliers representing higher discounts.

Quantity: The quantity variable seems to have a more regular distribution, with a few outliers representing larger quantities.

Insights and Future Exploration

Sales and Profit Outliers: Investigating the reasons behind unusually high sales and profits or significant losses can uncover underlying business factors, such as pricing strategies, marketing campaigns, or customer behavior.

High Discount Outliers: Understanding why certain transactions have higher discounts may reveal insights into promotional strategies or customer segmentation.

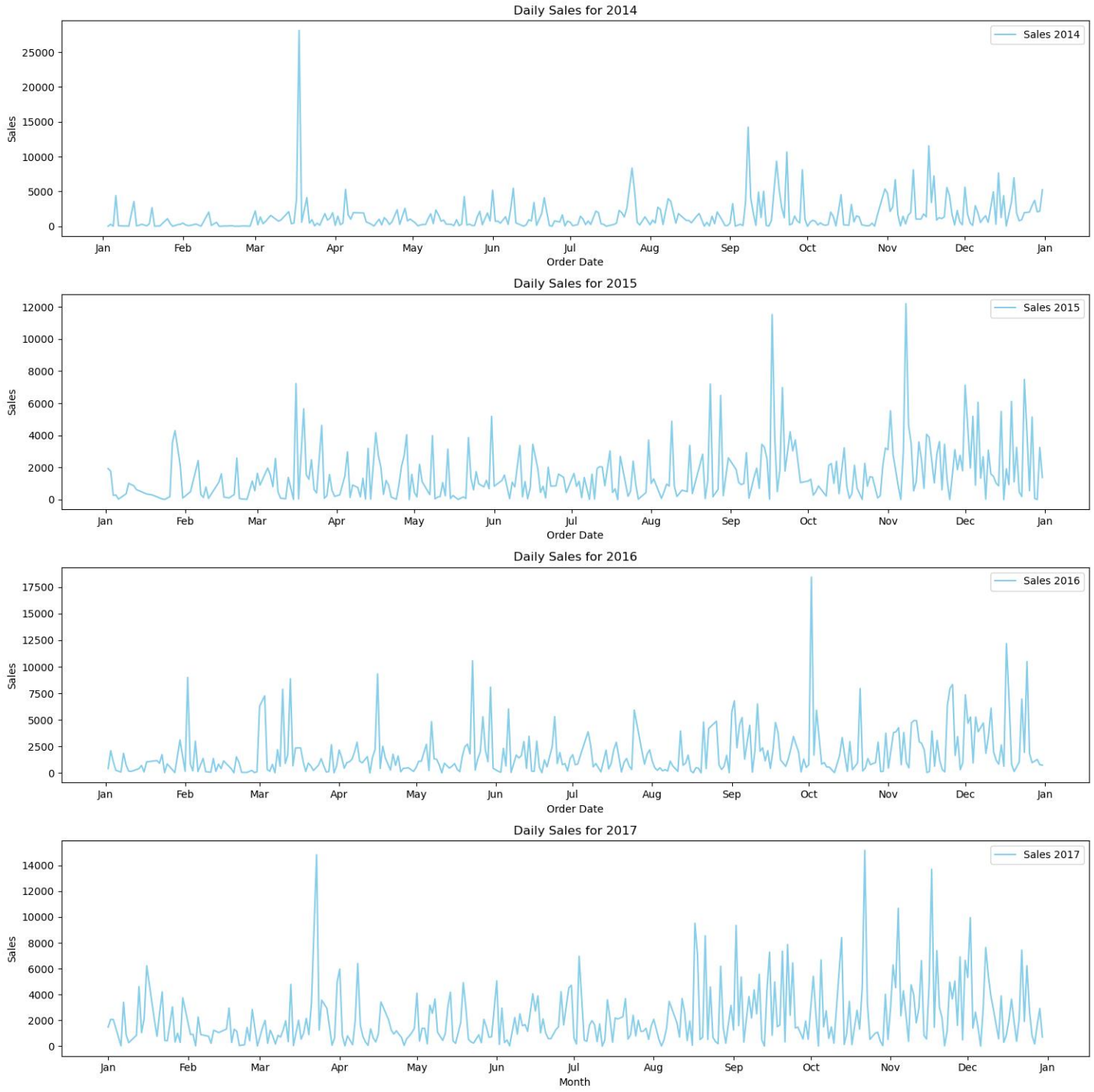
Quantity Outliers: Analyzing large quantity transactions can provide information about bulk purchases or popular products.

Descriptive Statistics

	count	mean	std	min	25%	50%	75%	max
Sales	9994.0	229.858001	623.245101	0.444	17.28000	54.4900	209.940	22638.480
Profit	9994.0	28.656896	234.260108	-6599.978	1.72875	8.6665	29.364	8399.976
Discount	9994.0	0.156203	0.206452	0.000	0.00000	0.2000	0.200	0.800
Quantity	9994.0	3.789574	2.225110	1.000	2.00000	3.0000	5.000	14.000

Time Series Visualization

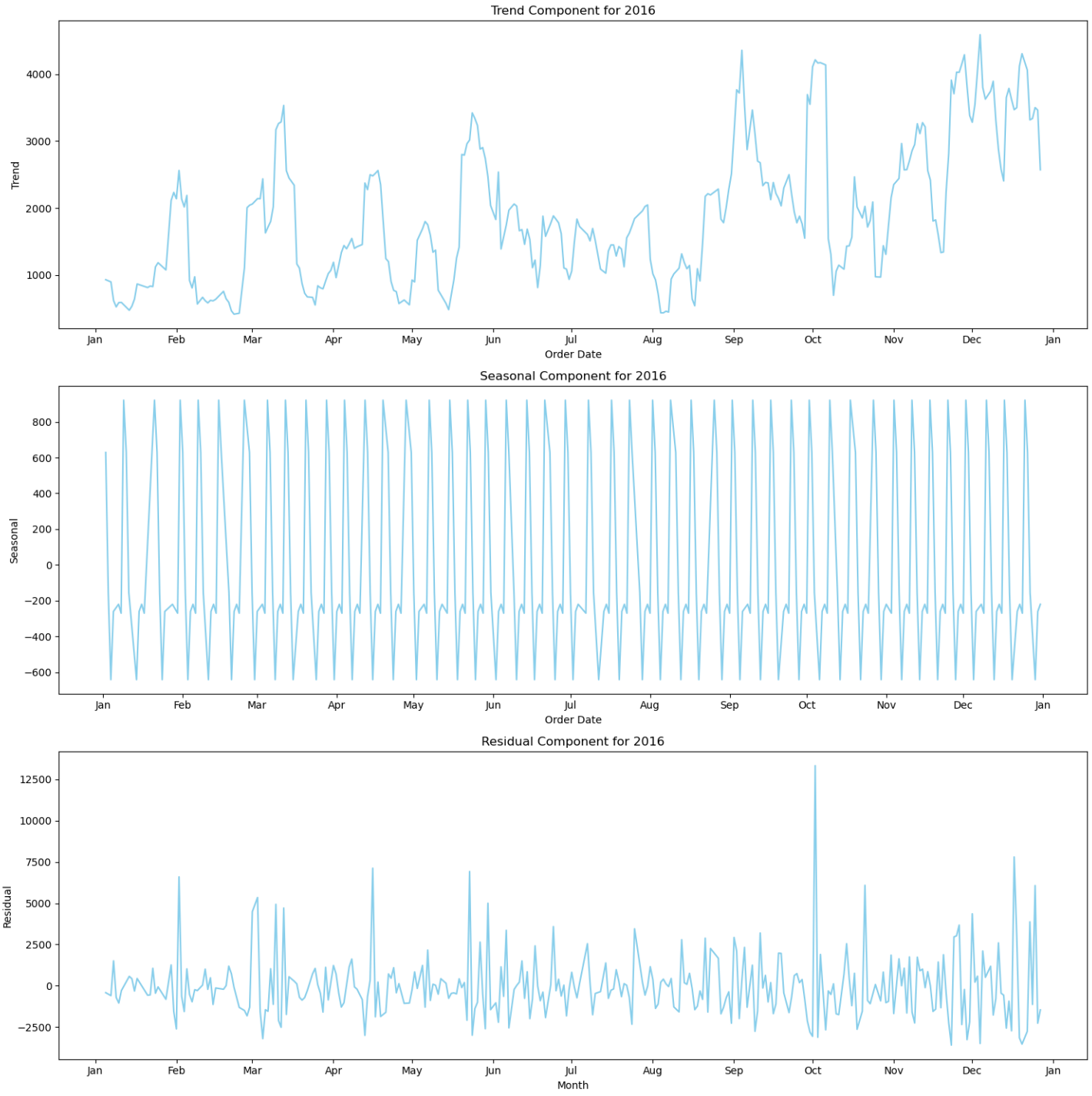
This plot helps us visualize sales trends and identify any patterns or anomalies in the sales data. Regular peaks or troughs suggest seasonality, while abrupt changes might indicate events or anomalies. Here I used the *.resample('D')* operation to change the frequency of time series data. This operation aggregates and transforms the data into a new time series with a daily frequency, making it more consistent and manageable; and allows for more consistent time series analysis. It can also help in handling irregular or higher-frequency data; however it's important to keep in mind that it can also lead to loss of information and overfitting.



The daily sales plot showcases variations in sales over time. We can notice patterns indicating potential weekly seasonality, as well as some spikes which might be due to specific events or promotions.

Time Series Decomposition

Time series decomposition involves breaking down a time series into three main components: Trend Component, Seasonal Component, Residual Component.



Analysis of Time Series Decomposition for 2016

Trend Component: The trend component is obtained by applying a moving average to smooth out short-term fluctuations. This helps us see the underlying growth or decline in the series. It appears to have an upward trend,

especially towards the end of the year. Understanding the trend can guide strategic planning, such as inventory management or marketing campaigns aligned with growth periods.

Seasonal Component: Seasonality is the repeating patterns or cycles in the data. For daily sales data, this could represent weekly patterns, such as increased sales on weekends. In this case, we see a weekly pattern, possibly reflecting increased sales on certain days of the week. Identifying weekly patterns can lead to targeted promotions or staffing adjustments to accommodate increased sales on specific days.

Residual Component: The residual component is what's left after the trend and seasonal components have been removed. Analyzing residuals can help identify anomalies or unexpected variations. In this case we see a few anomolies spaced out. Investigating significant residuals might reveal special events, holidays, or other factors affecting sales that were not captured by the trend and seasonality.

Box-Jenkins Models

Box-Jenkins Models, also known as ARIMA models, are a class of statistical techniques used for time series analysis and forecasting. They consist of Autoregressive (AR), Integrated (I), and Moving Average (MA) components, which capture the past values, differencing, and forecast errors, respectively. These models are prized for their flexibility, interpretability, and accuracy in short-term predictions, making them valuable tools for data scientist dealing with diverse and complex temporal data.

Checking Stationarity

ARIMA models assume that the data's statistical properties, such as mean and variance, do not change over time. If the data is non-stationary and exhibits trends, seasonality, or other fluctuations, the model's assumptions are violated, leading to inaccurate results. **Stationarity** is also crucial for meaningful autocorrelation analysis, which ARIMA models heavily rely on. **Autocorrelation** measures how a data point relates to its past values. In non-stationary data, the autocorrelation can be misleading, complicating the model's ability to capture underlying patterns. In the case of ARIMA models; the I component of the model refers to differencing; which transforms the data into a stationary form.

Visually

The plots above provides a visual representation of the "Sales" time series. We can see fluctuations in the data, but its challenging to determine stationarity from the plots alone. However; by decomposing the time series model we can see that there is a cyclical behavior in the data; as seen in the seasonal trend.

Dickey - Fuller Test

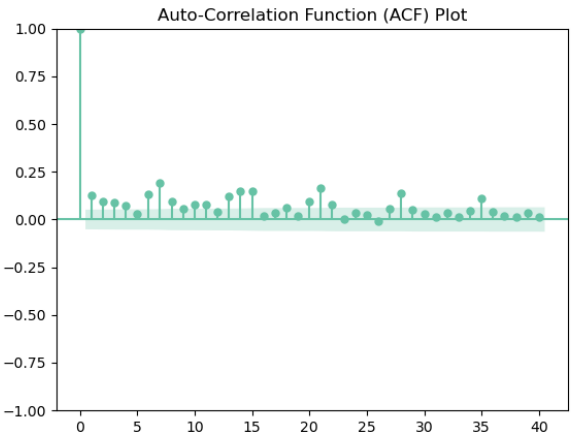
The **Dickey-Fuller test** is a statistical test used to determine if a time series is stationary. In our analysis, we used the Augmented Dickey-Fuller (ADF) version of the test, which is suitable for more complex time series.

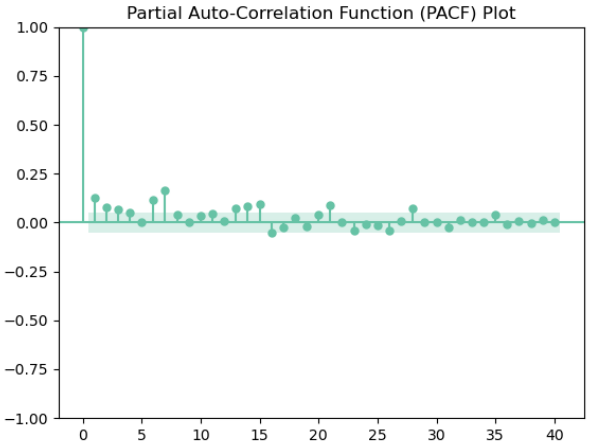
- Null Hypothesis ((H_0)): The series has a unit root (is non-stationary).
- Alternative Hypothesis ((H_1)): The series is stationary.

The test focuses on γ, with the null hypothesis that γ=0 (non-stationary) and the alternative hypothesis that γ<0 (stationary). A p-value < 0.05 indicates that we reject the null hypothesis, suggesting stationarity.

Analysis of the Dickey-Fuller Test

The p-value from the Dickey-Fuller test is approximately 2.91×10^−5; which is less than 0.05. **This allows us to reject the null hypothesis**, suggesting that the "Sales" time series is stationary.





Determining Order of AR and MA Models

ACF Plot

The **ACF plot** shows the correlation between the time series and its lagged values. It helps in identifying the order of the MA (Moving Average) component (q). For a MA model; the theoretical ACF of an MA model cuts off after lag q.

The ACF plot shows a gradual decrease without a distinct cut-off point. This lack of a clear indication can signify no clear MA component or model complexity that cannot be captured by just the MA component. We will explore this further when constructing our models; and explore which performs the best.

PACF Plot

The **PACF plot** shows the partial correlation between the time series and its lagged values, controlling for other lags. It helps in identifying the order of the AR (Autoregressive) component (p). For an AR model, the theoretical PACF of an AR model cuts off after lag p.

The PACF plot shows a significant spike at lag 1, then a gradual decrease, suggesting that the AR order (p) might be 1

Creating Models and Comparing Models

Evaluation Criteria - AIC vs BIC

AIC (Akaike Information Criterion): AIC balances goodness of fit with complexity (penalizes overfitting.

BIC (Bayesian Information Criterion): BIC is similar to AIC but has a stronger penalty for model complexity. In practice, both can be used for model selection. AIC tends to favor more complex models, while BIC favors simpler models. In our analysis, we will use both then compare.

Models Being Tested

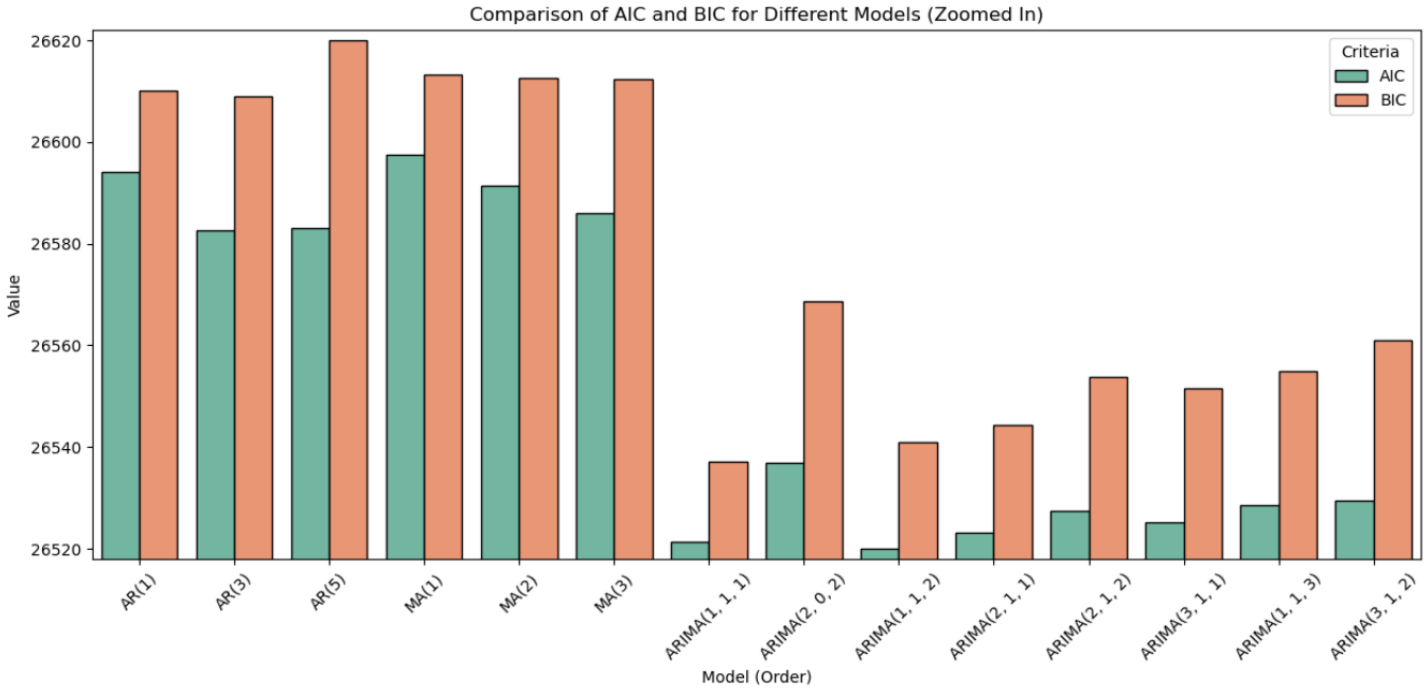
Autoregressive Models - AR models explain a time series based on its own past values.

Moving Average Models - MA models represent a time series as a linear combination of past error terms.

- For both AR and MA models I am using order [1,3,5] to evaluate different parameters; since PACF and ACF plots did not give any conclusive information.

Autoregressive Integrated Moving Average Models - ARIMA combines AR and MA models and includes differencing to achieve stationarity.

- Models with Differencing (d=1) are chosen to account for any potential non-stationarity that might be missed during the initial tests. Differencing helps in making the series stationary by removing trends.
- AR Terms (p) represent the influence of previous observations on the current observation. Variations in p allow us to capture different temporal dependencies.
- MA Terms (q) represent the influence of previous forecast errors on the current observation. Variations in q allow us to model different error structures.



Analysis of AR and MA models

AR and MA models do not include differencing. Even if the series appears stationary, subtle non-stationary patterns might be captured better with differencing which may explain the poor performance compared to the ARIMA models.

Analysis of ARIMA models

Models with Differencing (d=1) generally performed better. This suggests that even though the series was found to be stationary, differencing helped in modeling some hidden non-stationarity.

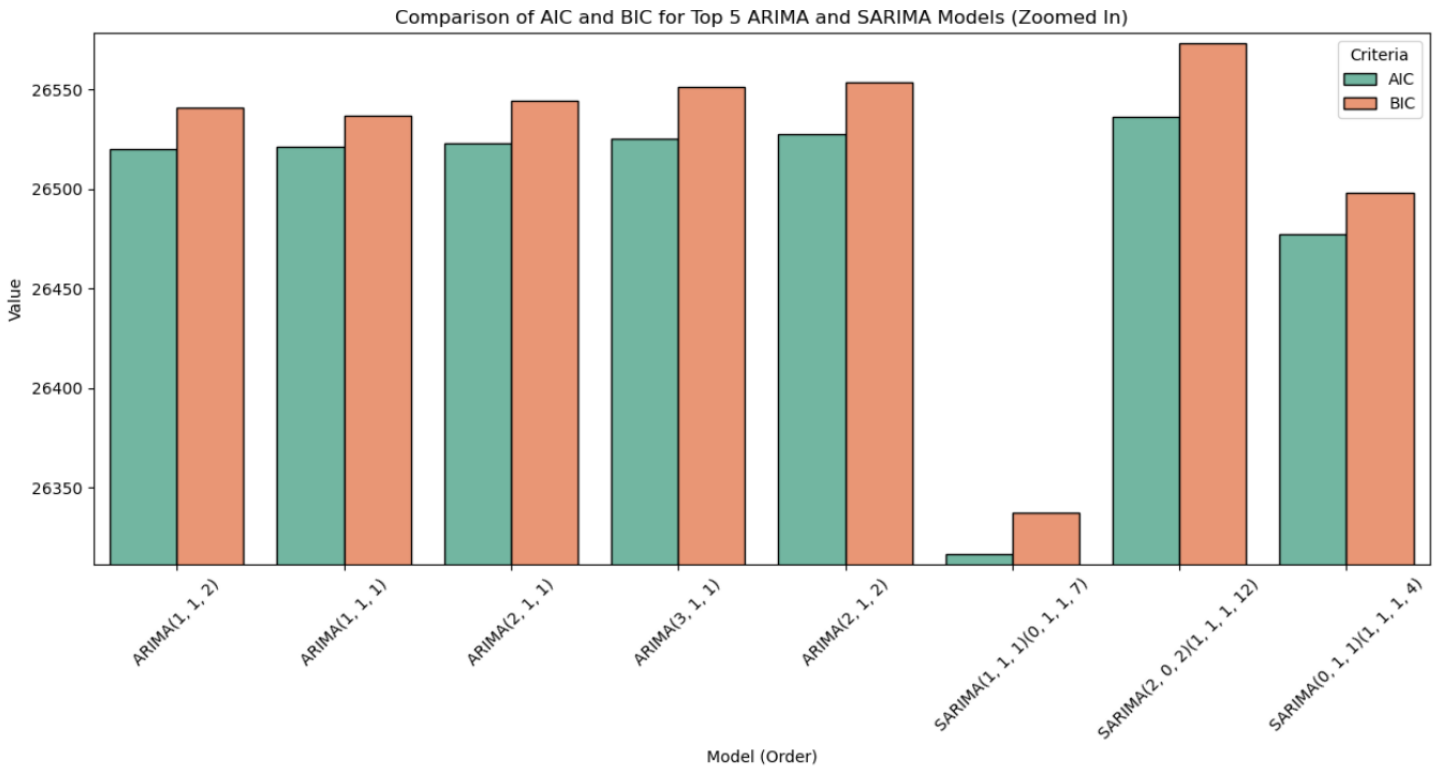
The ARIMA(1, 1, 2) model performed the best in terms of AIC, followed closely by the ARIMA(1, 1, 1) model.

Models with Higher Orders (p=3, q=3) did not necessarily perform better. This indicates that adding complexity does not always lead to better fitting models

Seasonal Autoregressive Integrated Moving Average Models - SARIMA (Seasonal ARIMA) models are extensions of ARIMA models that explicitly account for seasonality in the data. They are defined by four additional parameters: seasonal order (P, D, Q, s). Below are the models I will test:

- **SARIMA(1, 1, 1, 0, 1, 1, 7)**: Weekly seasonality with simple seasonal AR and MA terms.
- **SARIMA(2, 0, 2, 1, 1, 1, 12)**: Monthly seasonality with more complex non-seasonal components.

- **SARIMA(0, 1, 1, 1, 1, 1, 4)**: Quarterly seasonality with a simple non-seasonal MA term.



Analysis of SARIMA and ARIMA models

The SARIMA(1, 1, 1)(0, 1, 1, 7) model, designed to capture weekly seasonality, has the lowest AIC and BIC, indicating a better fit compared to the other two models. This is reasonable since the model did have a weekly cyclical pattern; as indicated in the Time Series Decomposition.

ARIMA models do not account for seasonality; therefore although they performed better than AR and MA models they did not perform better than SARIMA weekly model. However; the SARIMA models accounting for monthly and quarterly trend did not outperform the top 5 performing ARIMA models.

Further Investigation - In the future we can explore other SARIMA weekly seasonality models to determine if we can make further improvements to this model.

Model Evaluation

Residual Analysis

Residuals are the differences between the observed values and the values predicted by the model. Residual analysis plays a crucial role as it helps us evaluate the quality of our model's fit to the data and identify potential issues or patterns that might have been missed during the modeling process. We will conduct 2 tests: Normality Test Using QQ Plot, Normality Test Using Shapiro-Wilk Test and Autocorrelation Test Using Ljung-Box Test

In this case we are using the SARIMA model with a weekly pattern, specifically SARIMA(1, 1, 1)(0, 1, 1, 7), since it showed the best fit according to the AIC and BIC criteria.

Normality Test Using QQ Plot

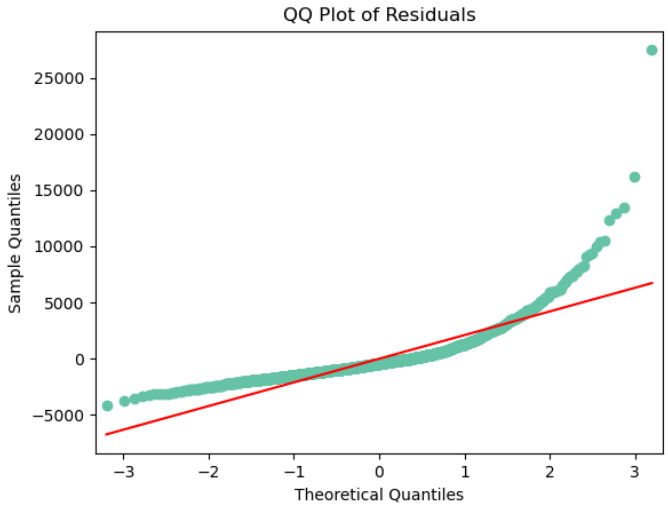
QQ plot (Quantile-Quantile plot) is a graphical tool used to assess whether a dataset follows a specific theoretical distribution, such as the normal distribution. In the context of time series analysis, it helps us check if the residuals of our model are normally distributed. If the points on the QQ plot roughly follow a straight line, it suggests that the residuals are approximately normally distributed.

Analysis of Normality Test Using QQ Plot

The x-axis represents the theoretical quantiles of a standard normal distribution. On the y-axis we can see the actual residuals from the model.

The actual residuals follow the 45-degree line up to 2 (second quartile). This suggests that the residuals mostly follow a normal distribution in the central part.

However, it is dispersed after 2. There are several severe outliers at 3 and afterwards. The dispersion and outlier indicate that the residuals have heavier tails than a normal distribution, showing some deviation from normality.



Normality Test Using Shapiro-Wilk Test

The **Shapiro-Wilk test** is a statistical test that quantifies the departure of a dataset from a normal distribution. The null hypothesis of the test is that the data are normally distributed. If the p-value obtained from the test is below a certain significance level (e.g., 0.05), we reject the null hypothesis and conclude that the residuals do not follow a normal distribution.

```
Shapiro-Wilk Test Statistic: 0.7574242949485779 P-value: 6.925217010693246e-42
```

Analysis of Normality Test Using Shapiro-Wilk Test

The low p-value indicates a rejection of the null hypothesis that the residuals are normally distributed. This aligns with the results we found above using the QQ-Plot.

Autocorrelation Test Using Ljung-Box Test

The **Ljung-Box test** is used to assess the presence of autocorrelation in a time series dataset. The Ljung-Box test is applied to the residuals to check for residual autocorrelation. The test statistic compares the autocorrelations of the residuals at different lags to the expected values under the assumption of no autocorrelation. If the test statistic is significantly different from what would be expected under the null hypothesis, it may indicate the presence of autocorrelation in the residuals.

The **lag value** is chosen based on the frequency of the time series data. In this case, the lag value was chosen to be 7 since the time series data exhibits clear weekly seasonal patterns.

Analysis of Autocorrelation Test Using Ljung-Box Test

Since the p-value is greater than 0.05, we fail to reject the null hypothesis that the residuals are independently distributed. This means that there is no evidence of autocorrelation in the residuals at lag 7, which is a good indication that the model has captured the underlying patterns in the data well.

This indicated that the residuals are behaving like white noise, and that the model has accounted for the underlying autocorrelation structure.

	lb_stat	lb_pvalue
7	6.455147	0.487718

Improving Model Performance- Addressing Non-Normality

Non-normality in the residuals may indicate that some underlying patterns in the data have not been adequately captured by the model.

Log Transformations

A log transformation is commonly used to stabilize variance and make a dataset more closely approximate normality. The transformation $log(1+x)$ is a variation of the log transformation that can handle zero values. By adding 1 to each value before taking the logarithm, you avoid the issue of taking the log of zero, which is undefined.

```
Shapiro-Wilk Test Statistic: 0.9751886129379272 P-value: 3.696146002923808e-15
```

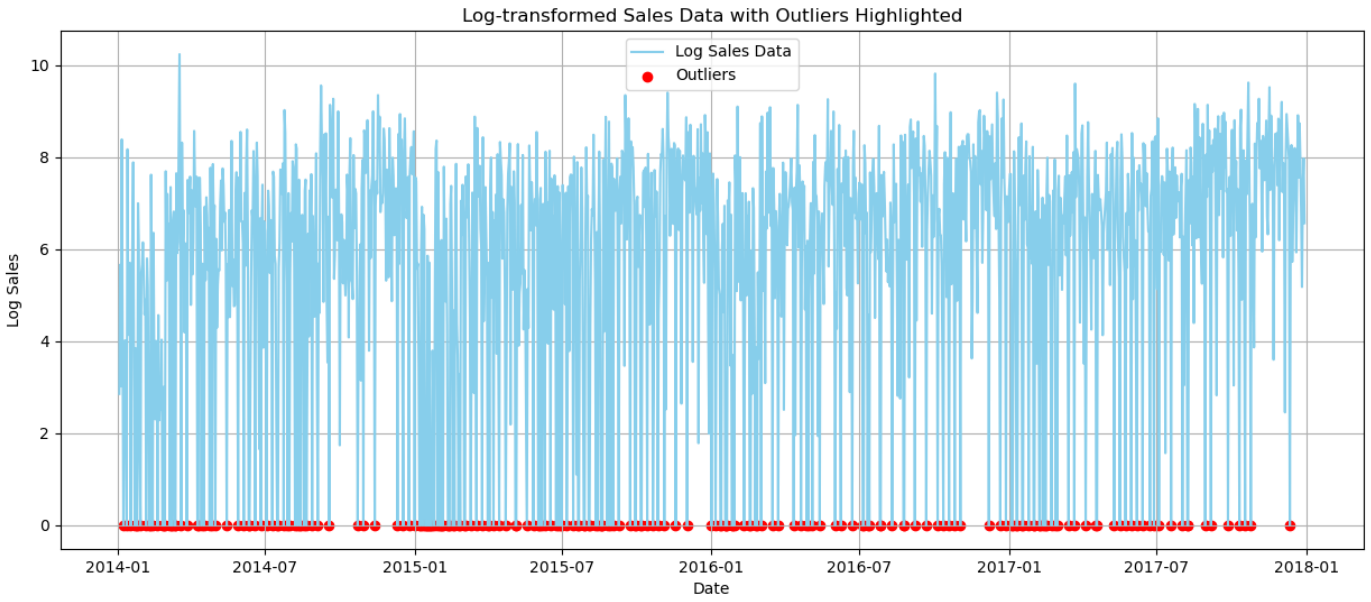
Analysis of Applying Log Transformation

The low p-value indicates a rejection of the null hypothesis that the residuals are normally distributed. The Normality test using Shapiro-Wilk test on the log-transformed residuals performed better than the previous model (relatively); however it still failed. Therefore further improvements should be made.

Model Performance - Model performed significantly better than previous SARIMA models. Will explore this further when comparing new SARIMA models.

Handling Outliers

Identifying and treating outliers can reduce the heavy tails in the residuals. As mentioned previously; the QQ plot showed that the deviations occur at higher theoretical quantiles (e.g., 3) and reach values around 25000. However there was also deviation at the lower theoretical quantiles as well. To address this issue we will inspect the outliers after applying log transformation to the dataset.



Analysis of Outliers

The random sampling of outliers; as well as the Log-Transformed Sales Data plot above indicate that all the outliers occur at 0. The extreme outliers that were present at the extremes ar no longer present after applying the log transformation on the data.

Replacing Outliers with Adjacent Averages / Weekly Averages

This strategy involves identifying outliers and replacing them with the adjacent averages. In the case of missing adjacent data; the data is replaced with the weekly average instead. This can be effective for preserving data and allowing the model to perform better.

```
Shapiro-Wilk Test Statistic: 0.9751886129379272 P-value: 3.696146002923808e-15
```

The low p-value indicates a rejection of the null hypothesis that the residuals are normally distributed. The Normality test using Shapiro-Wilk test on the log-transformed residuals performed identically as before.

The test statistic is close to 1, which would usually indicate that the distribution is close to normal. However, the extremely small p-value is evidence against the null hypothesis that the data follows a normal distribution. To further evaluate we will conduct a Normality Test Using QQ Plot below:

Model Performance - Model performed significantly better than previous SARIMA models. However performed identical to previous Log Transformation model indicating that the performance of the model was not compromised by replacing outliers with adjacent Averages and Weekly Averages.

Here we can see that the log transformation helped stabilize the variance across time, making the data more suitable for modeling.

By averaging out the Outliers it helped make the residuals more normally distributed and reduce any abrupt changes in the series, which could improve the fit of time series models.

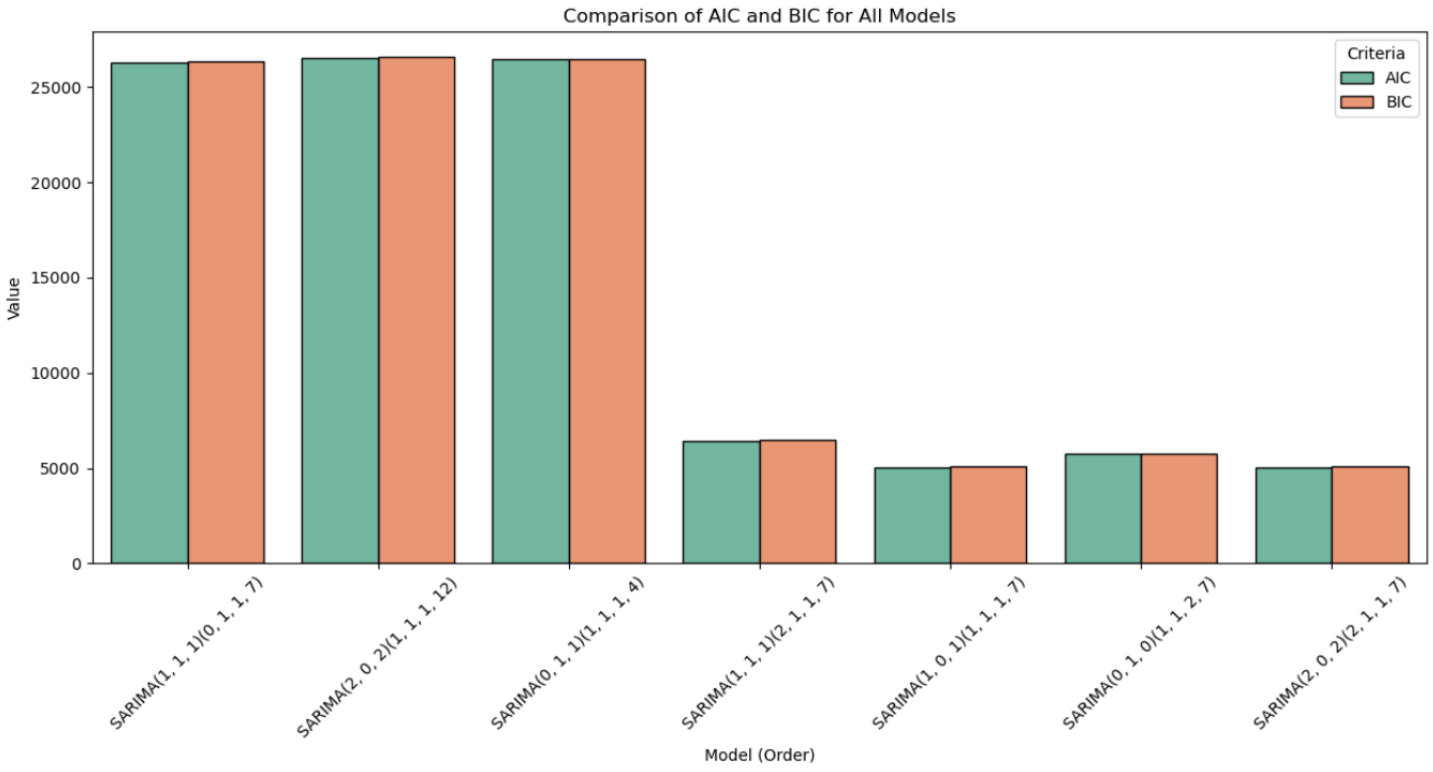
Model Re-Specification

Model 1: SARIMA(1, 0, 1)(1, 1, 1, 7) - This model includes both AR and MA terms with a seasonal differencing order of 1 and a seasonal period of 7, reflecting the weekly cycle. The presence of both AR and MA terms allows the model to capture complex autocorrelation patterns.

Model 2: SARIMA(0, 1, 0)(1, 1, 2, 7) - This model uses only seasonal AR and MA terms with a non-seasonal differencing order of 1. The absence of non-seasonal AR and MA terms might allow the model to focus on the weekly seasonal patterns.

Model 3: SARIMA(2, 0, 2)(2, 1, 1, 7) - This model has higher-order non-seasonal and seasonal AR and MA terms, allowing for a more flexible fit to the data. The seasonal differencing order and period remain consistent with the weekly cycle.

Comparing New SARIMA Models with Previous Models

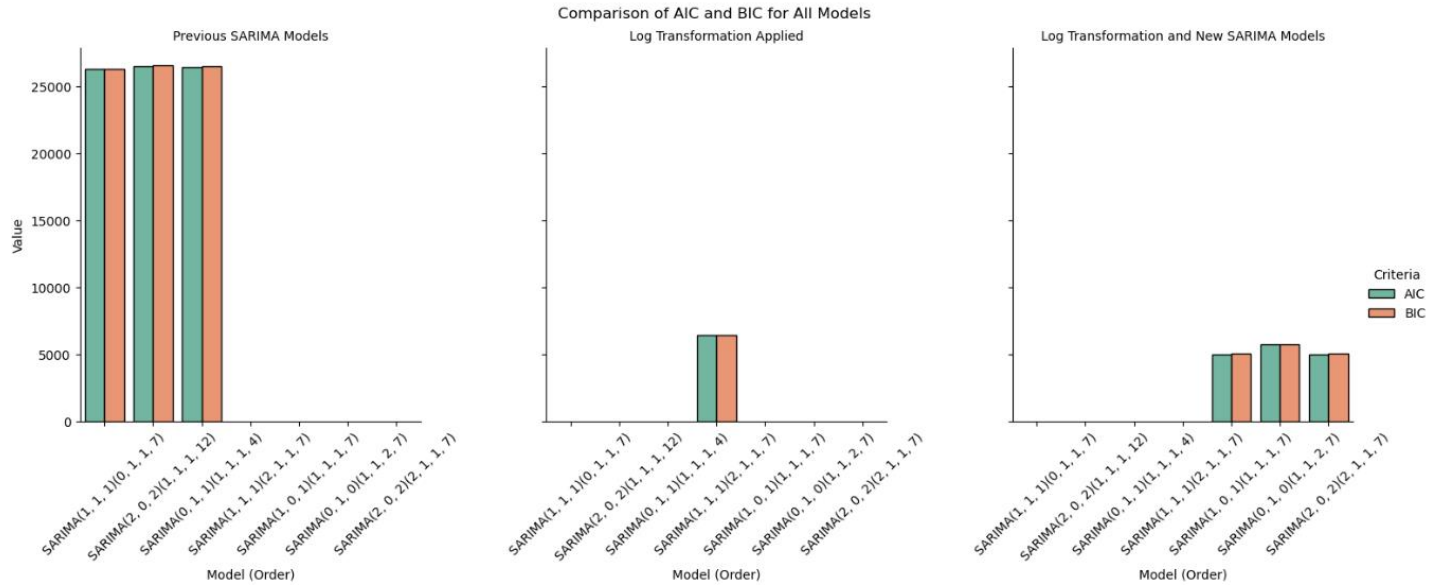


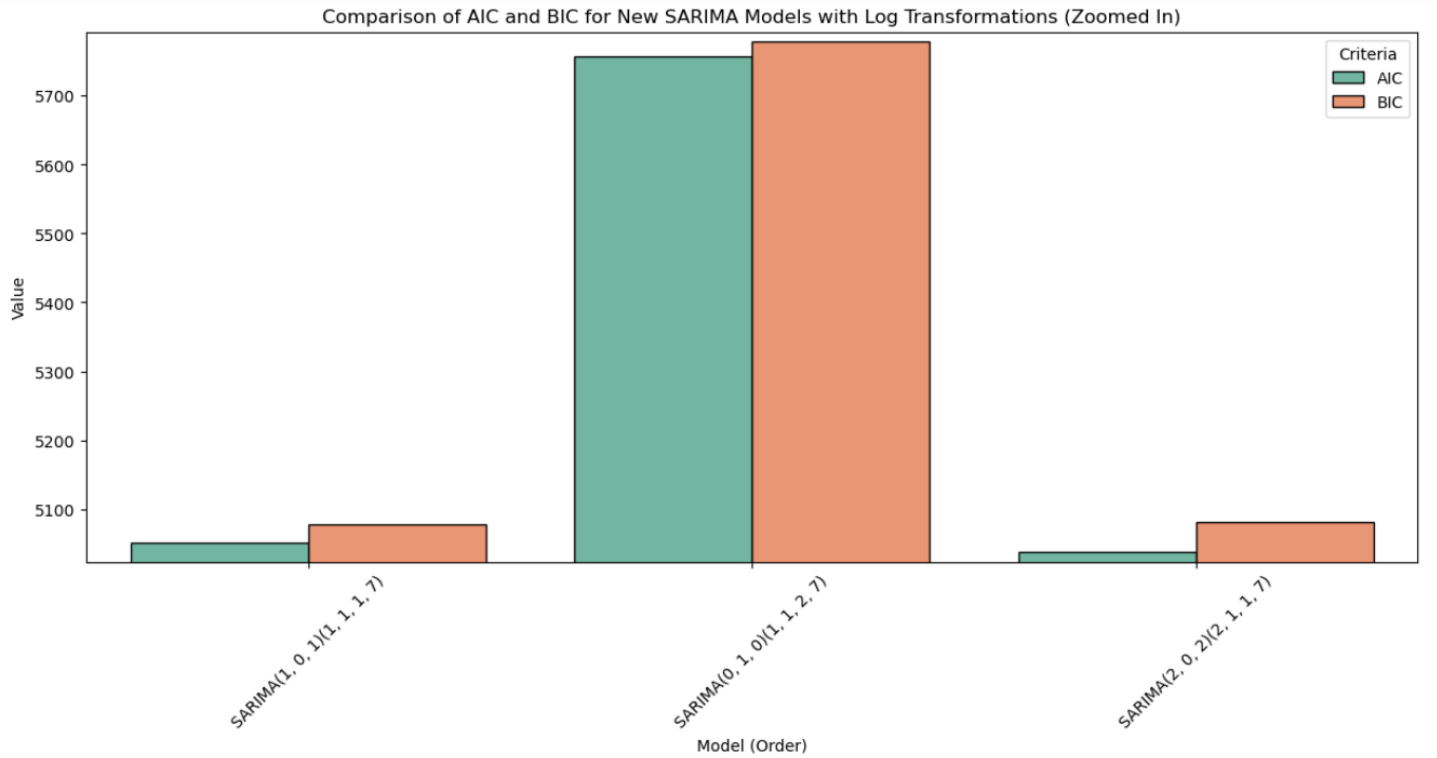
Analysis of New SARIMA Models with Previous Models

The three on the right were the SARIMA models created originally. In the previous graph we had zoomed in to compare these three models; however compared to the other four models present these three significantly under perform.

The one in the middle is the best performing SARIMA model (which took into account weekly cyclical pattern); and had the Log Transformation applied. As mentioned previously; the log transformation created significant improvement to model performance.

The three on the right assess different parameter for weekly cyclical SARIMA models. In order to clearly visualize all three different types of models the plots below separate them accordingly below:





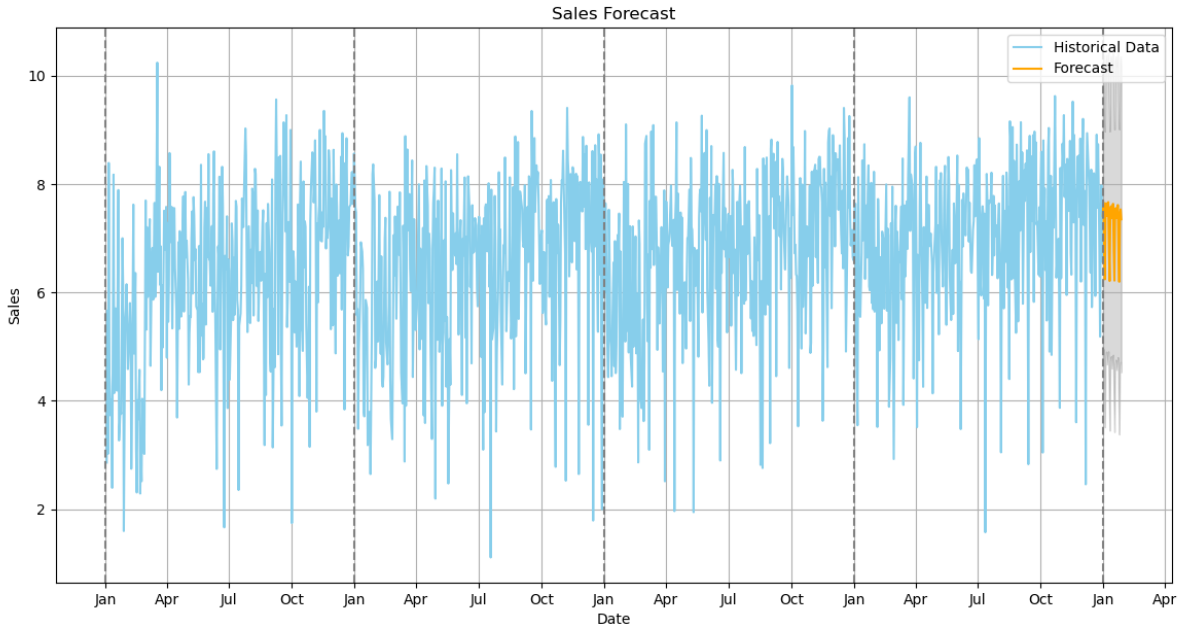
Analysis of New SARIMA Models

Model 1 and Model 3 performed significantly better than Model 2.

The absence of non-seasonal AR and MA terms in Model 2 led to its underperformance, as it could not capture some underlying patterns.

Model 1 and Model 3's inclusion of both AR and MA terms with seasonal differencing allowed them to capture more complexities.

Model Forecasting



Conclusion

Technical Application Explained

- **Improved Forecasting Models**: By enhancing the prediction models, the store manager can better anticipate weekly sales patterns. This allows for more effective inventory management, reducing overstocking or understocking issues.
- **Log Transformation and Outlier Handling**: These techniques helped make the forecasts more reliable, reducing sudden and unexpected changes that could lead to poor decision-making. This has a direct benefit in planning promotions, staffing, and other store operations.
- **Emphasis on Weekly Cycles**: The models that performed best paid attention to the weekly cycles in the data. For the store manager, this is vital in capturing regular shopping patterns and could lead to more efficient scheduling and targeted marketing.

Future Improvements

- **Model Specificity for Different Products or Categories**: While the current models were focused on overall trends, breaking down the forecasts by product category or even specific items could provide more nuanced

insights. Future investigations could tailor models to individual store departments, leading to more personalized strategies.

- **Integration with External Factors**: The current models may not take into account external factors like holidays, local events, or economic conditions. Future research could explore how these factors might be integrated into the models for even more accurate predictions.
- **Real-time Adaptation**: The existing models may need periodic re-evaluation and tuning. Creating a system that can adapt in real time to changes in shopping patterns or other influencing factors could be a valuable future development.

Final Thoughts

The project's success in improving the forecasting models offers tangible benefits to the store manager, primarily in inventory management and strategic planning. The focus on weekly cycles and data stabilization techniques has immediate applicability in daily store operations. However, future investigations could further refine the models, making them even more valuable by addressing specific product categories and incorporating external influences. Such advancements could lead to even more targeted and effective decision-making, aligning closely with the unique needs and challenges of managing a retail store.

Code Index

- *For a more in-depth analysis please refer to the Jupyter Notebook. Below are just what I consider important snippets of code that were utilized in this report.*

A.) Time Series Decomposition for 2016

```
from statsmodels.tsa.seasonal import seasonal_decompose

# Preparing the daily sales data for 2016
subset_2016 = data[data['Year'] == 2016]
daily_sales_2016 = subset_2016.groupby('Order Date')['Sales'].sum()

# Performing time series decomposition with a weekly frequency (7-day cycle)
decomposition_2016 = seasonal_decompose(daily_sales_2016, period=7)
```

```
# Setting up the subplots for trend, seasonal, and residual components
fig, axes = plt.subplots(3, 1, figsize=(15, 15), sharex=False)

# Plotting the decomposed components
components = {'Trend': decomposition_2016.trend,
              'Seasonal': decomposition_2016.seasonal,
              'Residual': decomposition_2016.resid}

for ax, component in zip(axes, components.items()):
    name, series = component
    sns.lineplot(x=series.index, y=series, color='skyblue', ax=ax)
    ax.set_title(f'{name} Component for 2016')
    ax.set_ylabel(name)
    ax.xaxis.set_major_locator(mdates.MonthLocator())  # Displaying only months on x-axis
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%b'))  # Displaying month name

plt.xlabel('Month')
plt.tight_layout()

# Save the figure to the specified path
plt.savefig('Time Series Decomposition for 2016.png')

plt.show()
```

```
# Importing the Augmented Dickey-Fuller test function
from statsmodels.tsa.stattools import adfuller

# Performing the Dickey-Fuller test
adf_result = adfuller(sales_data)

# Extracting the p-value
p_value = adf_result[1]

# Printing the result
p_value
```

```
2.9148790727674486e-05
```

```
# Importing the ACF and PACF plotting functions
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Plotting the ACF
plt.figure(figsize=(15, 8))
plot_acf(sales_data, lags=40)
plt.title('Auto-Correlation Function (ACF) Plot')

# Save the figure to the specified path
plt.savefig('ACF Plot.png')

plt.show()
```

```python
from statsmodels.tsa.arima.model import ARIMA

# Function to test AR models of different orders
def test_ar_models(data, orders=[1, 3, 5]):
    results = []
    for order in orders:
        # Fitting the AR model
        ar_model = ARIMA(data, order=(order, 0, 0))
        ar_fit = ar_model.fit()
        # Storing the AIC and BIC
        results.append((order, ar_fit.aic, ar_fit.bic))
    return results

# Testing AR models on the sales data
ar_results = test_ar_models(sales_data)

ar_results
```

```
[(1, 26594.205758160624, 26610.06022089833),
 (3, 26582.544412672796, 26608.96851723564),
 (5, 26583.003065472632, 26619.996811860612)]
```

```python
import warnings
warnings.filterwarnings('ignore')

# Defining the orders for ARIMA models
arima_orders = [
    (1, 1, 1),
    (2, 0, 2),
    (1, 1, 2),
    (2, 1, 1),
    (2, 1, 2),
    (3, 1, 1),
    (1, 1, 3),
    (3, 1, 2),
]

# Function to test ARIMA models of different orders
def test_arima_models(data, orders):
    results = []
    for order in orders:
        # Fitting the ARIMA model
        arima_model = ARIMA(data, order=order)
        arima_fit = arima_model.fit()
        # Storing the AIC and BIC
        results.append((order, arima_fit.aic, arima_fit.bic))
    return results

# Testing ARIMA models on the sales data using the previously defined function
arima_results = test_arima_models(sales_data, arima_orders)

arima_results
```

```
[((1, 1, 1), 26521.269577209903, 26537.12198162849),
 ((2, 0, 2), 26536.866074896316, 26568.575000371726),
 ((1, 1, 2), 26519.90737573874, 26541.043914963524),
 ((2, 1, 1), 26523.19346322155, 26544.33000244633),
 ((2, 1, 2), 26527.415532257524, 26553.8362062885),
 ((3, 1, 1), 26525.189824877754, 26551.61049890873),
 ((1, 1, 3), 26528.573898673552, 26554.994572704527),
 ((3, 1, 2), 26529.400341460845, 26561.105150298015)]
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Function to test SARIMA models of different orders
def test_sarima_models(data, orders):
    results = []
    for order, seasonal_order in orders:
        # Fitting the SARIMA model
        sarima_model = SARIMAX(data, order=order, seasonal_order=seasonal_order)
        sarima_fit = sarima_model.fit(disp=False)
        # Storing the AIC and BIC
        results.append((order, seasonal_order, sarima_fit.aic, sarima_fit.bic))
    return results

# Defining the orders for SARIMA models
sarima_orders = [
    ((1, 1, 1), (0, 1, 1, 7)),  # Weekly seasonality
    ((2, 0, 2), (1, 1, 1, 12)), # Monthly seasonality
    ((0, 1, 1), (1, 1, 1, 4)),  # Quarterly seasonality
]

# Testing SARIMA models on the sales data
sarima_results = test_sarima_models(sales_data, sarima_orders)

sarima_results
```

```
[((1, 1, 1), (0, 1, 1, 7), 26316.341673192757, 26337.458948534415),
 ((2, 0, 2), (1, 1, 1, 12), 26536.346404882803, 26573.282299701834),
 ((0, 1, 1), (1, 1, 1, 4), 26477.099293187355, 26498.224835841636)]
```

```
from scipy.stats import shapiro
from statsmodels.graphics.gofplots import qqplot
from statsmodels.stats.diagnostic import acorr_ljungbox

# Fitting the selected SARIMA model
sarima_order = (1, 1, 1)
seasonal_order = (0, 1, 1, 7)
sarima_model = SARIMAX(sales_data, order=sarima_order, seasonal_order=seasonal_order)
sarima_fit = sarima_model.fit(disp=False)
# Extracting the residuals
residuals = sarima_fit.resid
```

```
from statsmodels.graphics.gofplots import qqplot
```

```
# Normality test using QQ plot
qqplot(residuals, line='s')
plt.title('QQ Plot of Residuals')

# Save the figure to the specified path
plt.savefig('Normality Test Using QQ Plot.png')

plt.show()
```

```
# Normality test using Shapiro-Wilk test
shapiro_test_stat, shapiro_p_value = shapiro(residuals)
print("Shapiro-Wilk Test Statistic:", shapiro_test_stat, "P-value:", shapiro_p_value)
```

```
Shapiro-Wilk Test Statistic: 0.7574242949485779 P-value: 6.925217010693246e-42
```

```
# Autocorrelation test using Ljung-Box test
ljung_box_result = acorr_ljungbox(residuals, lags=[7])
print(ljung_box_result)
```

```
    lb_stat  lb_pvalue
7  6.455147   0.487718
```

2.) Evaluating Log Transformations

```
# Fitting the selected SARIMA model to the log-transformed data
sarima_order = (1, 1, 1)
seasonal_order = (0, 1, 1, 7)
sarima_model_log = SARIMAX(log_sales_data, order=sarima_order, seasonal_order=seasonal_order)
sarima_fit_log = sarima_model_log.fit(disp=False)

# Extracting the residuals
residuals_log = sarima_fit_log.resid

# Normality test using Shapiro-Wilk test on the log-transformed residuals
shapiro_test_stat_log, shapiro_p_value_log = shapiro(residuals_log)
print("Shapiro-Wilk Test Statistic:", shapiro_test_stat_log, "P-value:", shapiro_p_value_log)
sarima_fit_log.aic, sarima_fit_log.bic,
```

```
Shapiro-Wilk Test Statistic: 0.9751886129379272 P-value: 3.696146002923808e-15
```

```
(6440.280648397795, 6461.397923739453)
```

1.) Creating New SARIMA Models

```
# Creating a list to store SARIMA results
new_sarima_results = []

# List of new SARIMA orders
new_sarima_orders = [
    ((1, 0, 1), (1, 1, 1, 7)),
    ((0, 1, 0), (1, 1, 2, 7)),
    ((2, 0, 2), (2, 1, 1, 7))
]

# Fitting the new SARIMA models
for order, seasonal_order in new_sarima_orders:
    sarima_model_new = SARIMAX(log_sales_data_replaced, order=order, seasonal_order=seasonal_order)
    sarima_fit_new = sarima_model_new.fit(disp=False)
    aic_new = sarima_fit_new.aic
    bic_new = sarima_fit_new.bic
    new_sarima_results.append((f"SARIMA{order}{seasonal_order}", aic_new, bic_new))
```

3. Model Forecasting

Forecasting using the final model:

```
# Fitting the specific SARIMA(2, 0, 2)(2, 1, 1, 7) model
final_model = SARIMAX(log_sales_data_replaced, order=(2, 0, 2), seasonal_order=(2, 1, 1, 7))
sarima_fit_specific = final_model.fit(disp=False)

# Forecasting future values (e.g., 28 periods ahead)
forecast_periods = 28
forecast_results = sarima_fit_specific.get_forecast(steps=forecast_periods)
forecast_values = forecast_results.predicted_mean
confidence_intervals = forecast_results.conf_int()

# Getting the index for the forecast
forecast_index = log_sales_data_replaced.index[-1] + pd.DateOffset(1)
forecast_index = pd.date_range(forecast_index, periods=forecast_periods, freq='D')

# Plotting the forecast
plt.figure(figsize=(14, 7))
plt.plot(log_sales_data_replaced, label='Historical Data', color='skyblue')
plt.plot(forecast_index, forecast_values, label='Forecast', color='orange')
plt.fill_between(forecast_index, confidence_intervals.iloc[:, 0], confidence_intervals.iloc[:, 1], color='k', alpha=.15)

# Setting the x-axis to display every third month
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(bymonth=range(1, 13, 3)))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b'))

# Adding vertical dashed lines to separate each year
for year in range(log_sales_data_replaced.index.year.min(), forecast_index.year.max() + 1):
    plt.axvline(pd.Timestamp(f'{year}-01-01'), color='gray', linestyle='--')
```