Retail Sales Forecasting: Beating the Holiday Rush with SARIMAX

I built this project to tackle a classic, retail problem: how to accurately **forecast sales**. Simple models just don't work when you have to account for holiday sales surges, marketing promos, and long-term growth all at once.

My goal was to create a tool that could actually be used by an operations or marketing team. The final model, a fine-tuned **SARIMAX**, doesn't just guess—it **quantifies the impact of promotions** and gives a clear picture of the months ahead, helping teams optimize inventory and stop losing money.

The Business Problem: Why This Matters

Every retailer knows the pain of getting the forecast wrong. It's a constant balancing act that leads to two costly mistakes:

- **Overstocking:** You buy too much, and your cash gets tied up in products that will eventually be sold at a steep discount.
- **Stockouts:** You buy too little, miss out on sales during your busiest season, and send customers to your competitors.

This project was all about building a **reliable solution** to avoid those problems.

What This Project Can Do

- **Analyzes the Full Picture:** It breaks down the sales data to see the real trend, the 12-month seasonal patterns, and the impact of business decisions.

- **Quantifies What Matters:** It measures the exact sales lift from things like **marketing promotions**.
- **Compares Models Head-to-Head:** I didn't just pick one model. I tested **ARIMA, SARIMAX, and Prophet** to find the undisputed winner.
- **Delivers Actionable Forecasts:** The final model gives a **6-month forecast with a 95% confidence range**, so you're not just getting a single number—you're getting a clear guide for managing risk.
- **Includes the Data:** I've included the Python script I used to generate the realistic 4-year (48-month) **synthetic dataset**, complete with trends, seasonality, and promo flags.

My Workflow: From Data to Forecast

I followed a structured process to make sure the model was built on a solid foundation.

1. Data Generation & Prep

First, I needed good data. I built a 4-year synthetic dataset from scratch using pandas and numpy. This let me create a realistic baseline (retail_sales_mock_data.csv) that included:

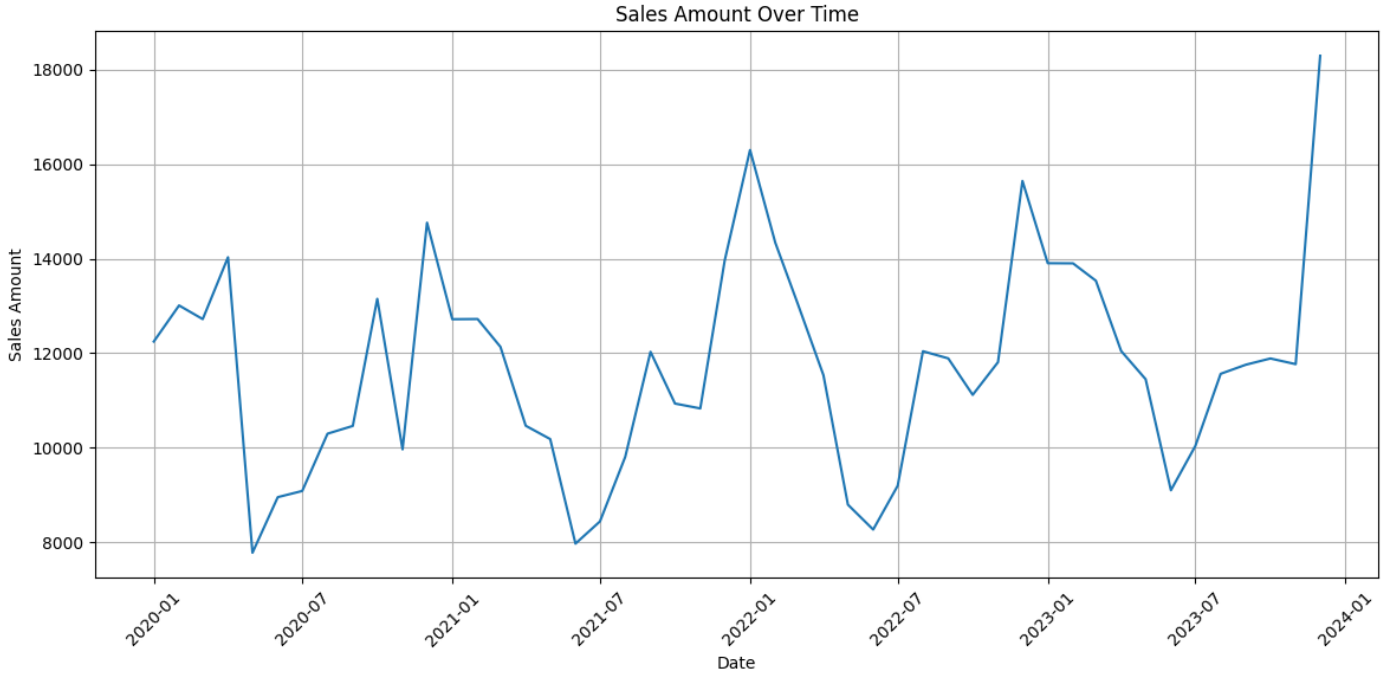
- A steady **upward trend** through "base_sales = 10000 + np.arange(num_months) * 50" (depicts the company growth).
- Strong **12-month seasonality** (the holiday rush).
- Binary flags for **Promotions** and **Holidays**.
- A bit of random **noise** through np.random.normal (like tiny unpredictable changes in sales such as weather, supply hiccups, customer behavior).

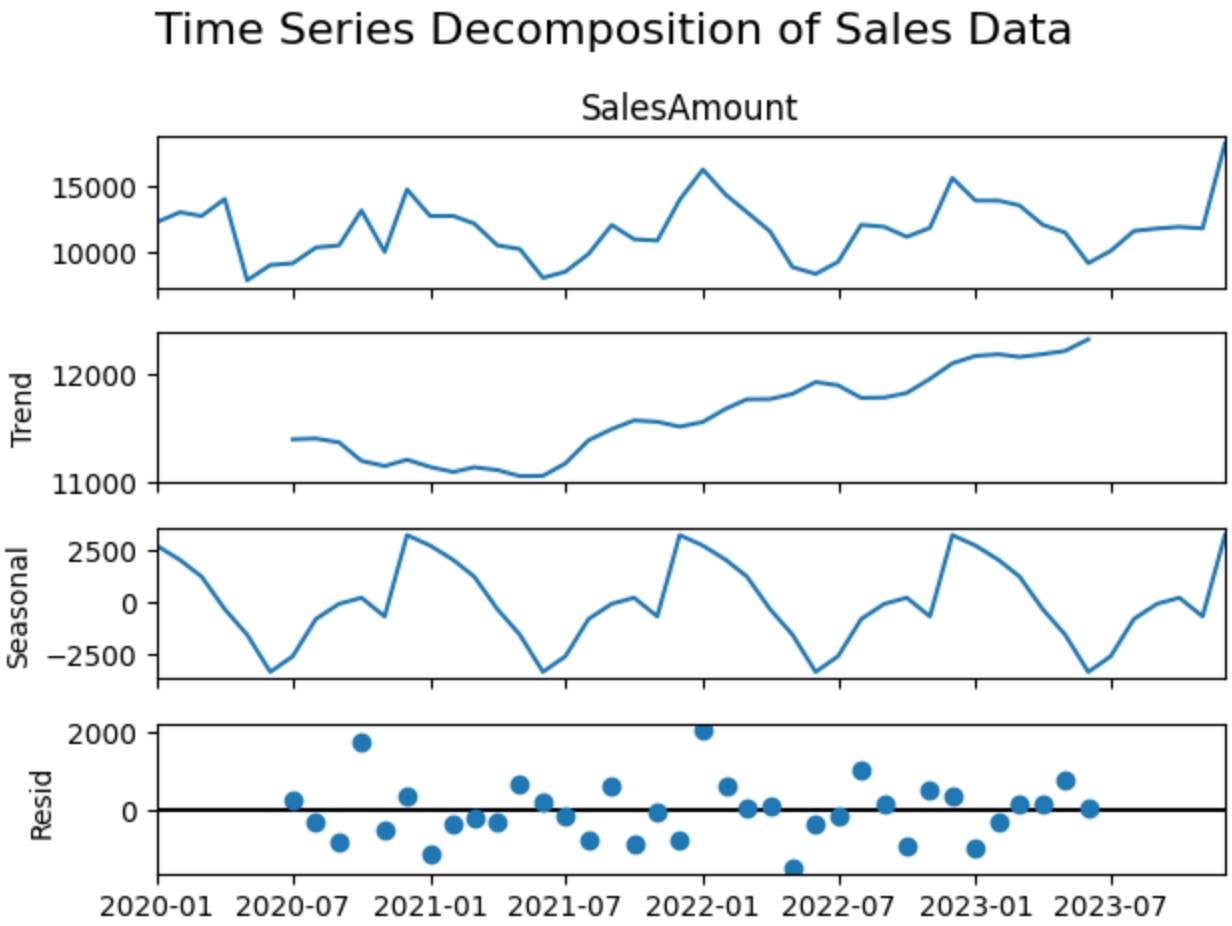
After generating it, I loaded, cleaned, and set up the Date index for time series analysi

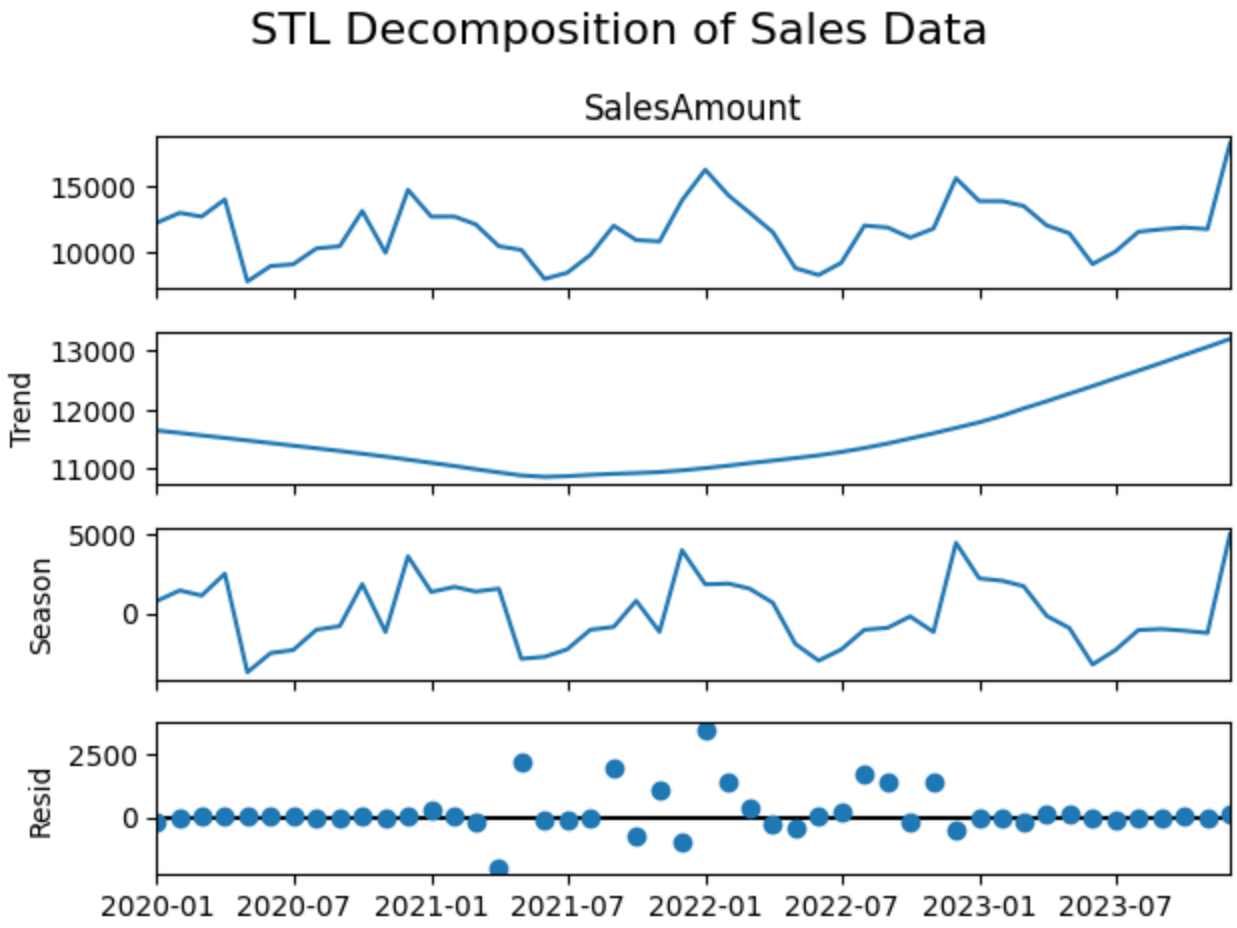
2. Exploratory Data Analysis (EDA)

Before trying to model anything, I had to understand the data.

- **Decomposition :** To properly understand the sales data, I needed to break it down into its main components:
    i. **Trend**: A long-term increase or decrease in the data over time,
    ii. **Seasonality**: A pattern that repeats at regular intervals (e.g., weekly, monthly, yearly)
    iii. **Residuals**: The leftover noise




Time Series Decomposition of Sales Data
SalesAmount
15000
10000
Trend
12000
11000
Seasonal
2500
0
−2500
Resid
2000
0
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07

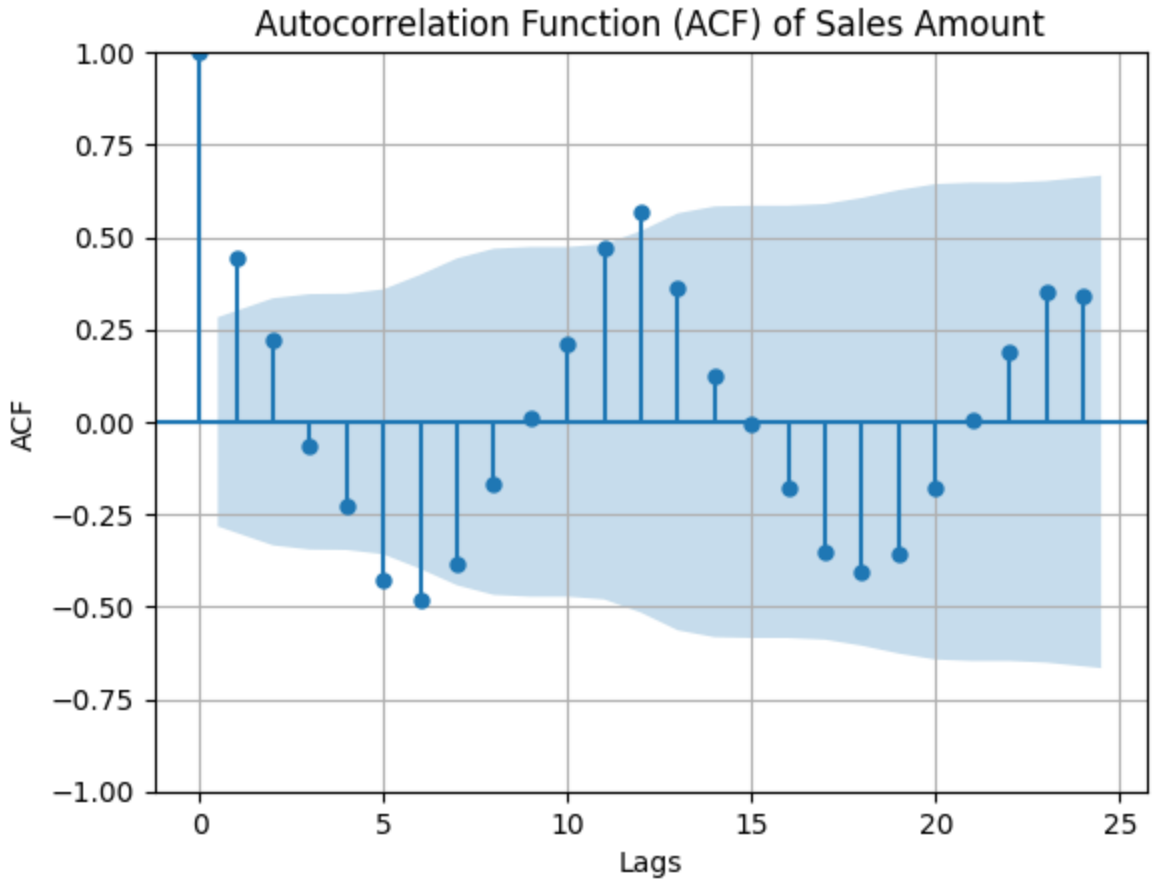


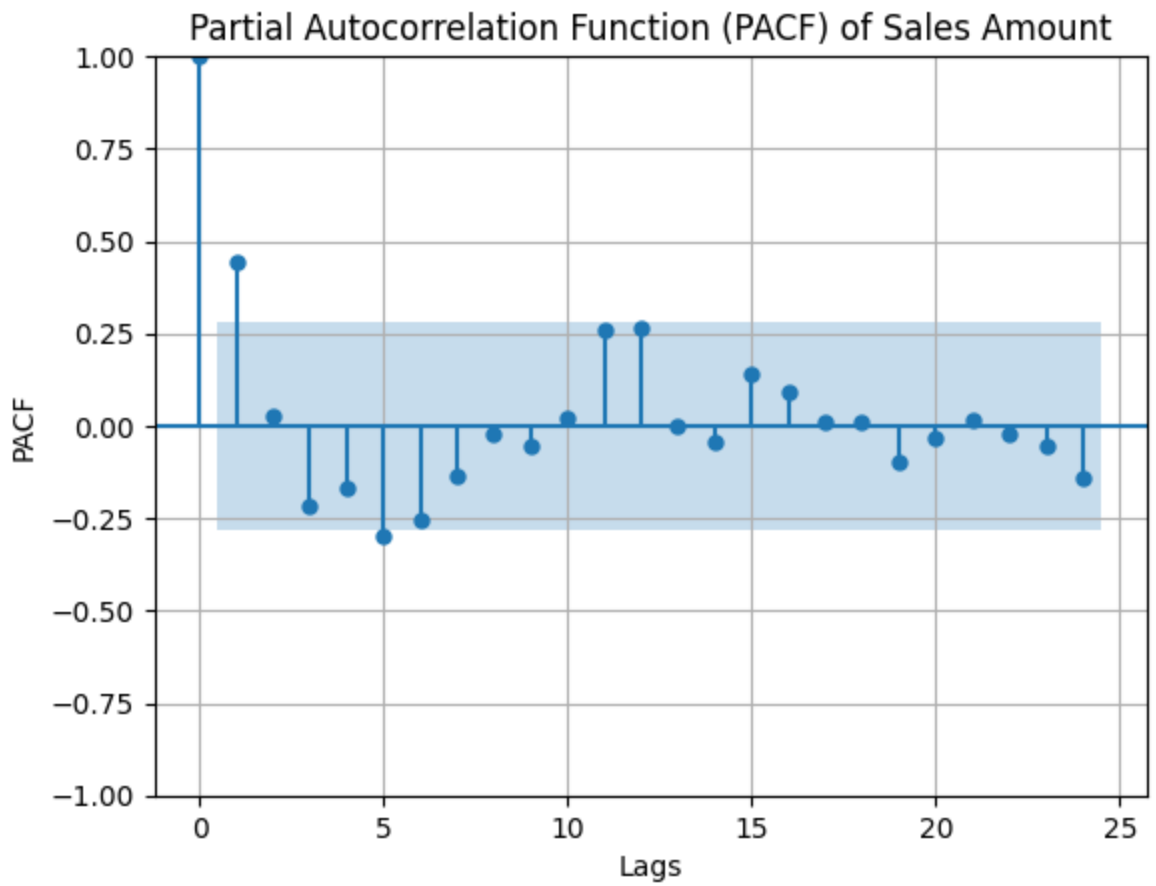
- I generated two decomposition plots (shown above) —one classical and STL method. Both plots showed a consistent upward trend, which suggests that my sales are experiencing steady, long-term growth. Critically, the seasonal component in both decompositions revealed significant, repeating yearly patterns, peaking strongly towards the end of the year which is entirely typical for retail holiday rushes. The residuals from the STL decomposition looked even cleaner, appearing randomly distributed around zero. **This robustness of the STL method is important because it confirms the model effectively captured the trend and seasonality, leaving less unexplained noise**

```
ADF Statistic: -4.333890747860533
p-value: 0.0003878661736017086
Number of lags used:  5
Number of observations used for ADF regression and critical values calculation: 42
Critical Values:
   1%: -3.596635636000432
   5%: -2.933297331821618
   10%: -2.6049909750566895
```

.

- **Stationarity (ADF Test):** I ran the Augmented Dickey-Fuller (ADF) test (shown above) to see if the data was stable (stationary) or if it needed transformations. I ran the adfuller test on the SalesAmount column, and the results were great! The ADF Statistic came out very negative at -4.334. More importantly, the p-value was extremely low at 0.000388, which is far below the standard 0.05 significance threshold. Because the p-value is so small, I can confidently reject the null hypothesis that the data is non-stationary (has a unit root). This strong evidence tells me that the sales data is already stationary. This is a huge win because it means I won't have to apply complicated transformations like differencing to stabilize the series before I start building the ARIMA or SARIMA model. The data is ready for modeling
- **Analyzing Time Dependency with ACF and PACF**: To prepare for building an effective forecasting model, I needed to understand the time-based dependencies in the sales data. Instead of looking at simple correlation between two different variables, I focused on autocorrelation, which measures how sales at the current month are related to sales from previous months.
- I generated two key plots for this checking up to 24 months (lags) back:
- a. the Autocorrelation Function (ACF)
- b. Partial Autocorrelation Function (PACF),


Autocorrelation Function (ACF) of Sales Amount
ACF
Lags
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25



- The Autocorrelation Function (ACF) plot shows how the sales amount at different time lags correlates with the current sales value. At lag 0, the autocorrelation is 1, indicating perfect correlation with itself, as expected. The visible peaks around lag 6 and 12 suggest that there's a repeating pattern roughly every 6 and 12 months, which points to seasonality — for example, a holiday or shopping trend that happens once a year

- Partial Autocorrelation Function (PACF) plot isolates the direct effect of past months on the current month's sales, removing the "middle steps." The biggest drop after lag 1 shows that last month's sales have the most direct influence, while later months have a weaker or indirect impact.
- In short: ACF = "Everything that might have an effect, directly or through other months." PACF = "Only the immediate, direct impact from specific past months."

3. Feature Engineering: Introducing Lagged Sales

- Sales today are often influenced by sales from the last few months. For example, if sales were high last month due to a promotion, that momentum might carry forward.
- My model needs those past values to "see" that pattern.
- I wrote a Python function called create_lag_features specifically to handle this.
- Inside, I used a loop to iterate through a list of specified lag periods (1,2,3,6,12), employing the pandas.shift() method to slide the SalesAmount values down by the appropriate number of months.

  The column "SalesAmount_lag_1" represents sales from 1 month ago
  The column "SalesAmount_lag_2" represents sales from 2 months ago
  The column "SalesAmount_lag_3" represents sales from 3 months ago
  The column "SalesAmount_lag_6" represents sales from 6 months ago
  The column "SalesAmount_lag_12" represents sales from 12 month ago

The lags collectively let my model “look back” over different time horizons — short, medium, and long-term — to learn repeating sales patterns more accurately

After running the function, my new DataFrame, "sales_data_with_lags", now has **8 columns**, including the **5 brand-new lag features**.

As expected, looking at the .info() output, the total number of entries dropped from 48 to 36 because the first 12 rows were removed due to the 12-month lag.

This new, richer dataset, starting from January 2021, was saved to **retail_sales_with_lags.csv**.

Finally, I ran the **Augmented Dickey-Fuller (ADF) test one more time** on the new dataset's SalesAmount column.

```
ADF Statistic: -3.679957551985488
p-value: 0.004404066055334108
Number of lags used:  5
Number of observations used for ADF regression and critical values calculation: 30
Critical Values:
   1%: -3.6699197407407405
   5%: -2.9640707407407407
   10%: -2.621171111111111
```

.

The resulting p-value of 0.0044 is still well below 0.05, confirming that the series remains stationary even after adding the lag features. This successful feature engineering step means the dataset is now fully prepared for advanced predictive modeling.

4. The Model: Finding a Winner

I split the data into a **30-month training set** and a **6-month validation set** to see which model could perform best on unseen data.

- **Attempt 1: ARIMA**

> **ARIMA(1,0,1)**

- I started with a simple ARIMA(1, 0, 1) model, which incorporates one previous month's sales (AR=1) and one previous month's prediction error (MA=1). The I=0 part was crucial, as I already proved the data was stationary and didn't need differencing.
- The arima_model.fit() function executed the machine learning, finding the optimal parameters to minimize the prediction error on the training data. The model summary looked solid, confirming the parameters were statistically significant. I then immediately used the trained model to forecast sales for the next 6 steps, corresponding directly to the unseen data in the validationSet.
- Comparing the actual sales to the forecast showed a serious problem, especially in December 2023: the actual sales spiked to 18,289, but the model only predicted 11,590, completely missing the huge seasonal holiday lift.

Date	Actual	Forecast
2023-07-01	10042.0	9404.249277
2023-08-01	11566.0	10522.267519
2023-09-01	11759.0	11087.363305
2023-10-01	11890.0	11372.987668
2023-11-01	11770.0	11517.354833
2023-12-01	18289.0	11590.324373

- The model struggles significantly with the last data point (December 2023), **under-forecasting the actual sales by nearly $6,700**. This massive error suggests a strong seasonal spike in December that the simple ARIMA(1,0,1) model failed to capture. The model's forecasts for July through November are much closer to the actual values.
- This discrepancy has immediate real-world implications; if a business relies on this forecast, they would drastically under-order inventory, leading to missed sales opportunities and poor customer satisfaction during the peak holiday season.

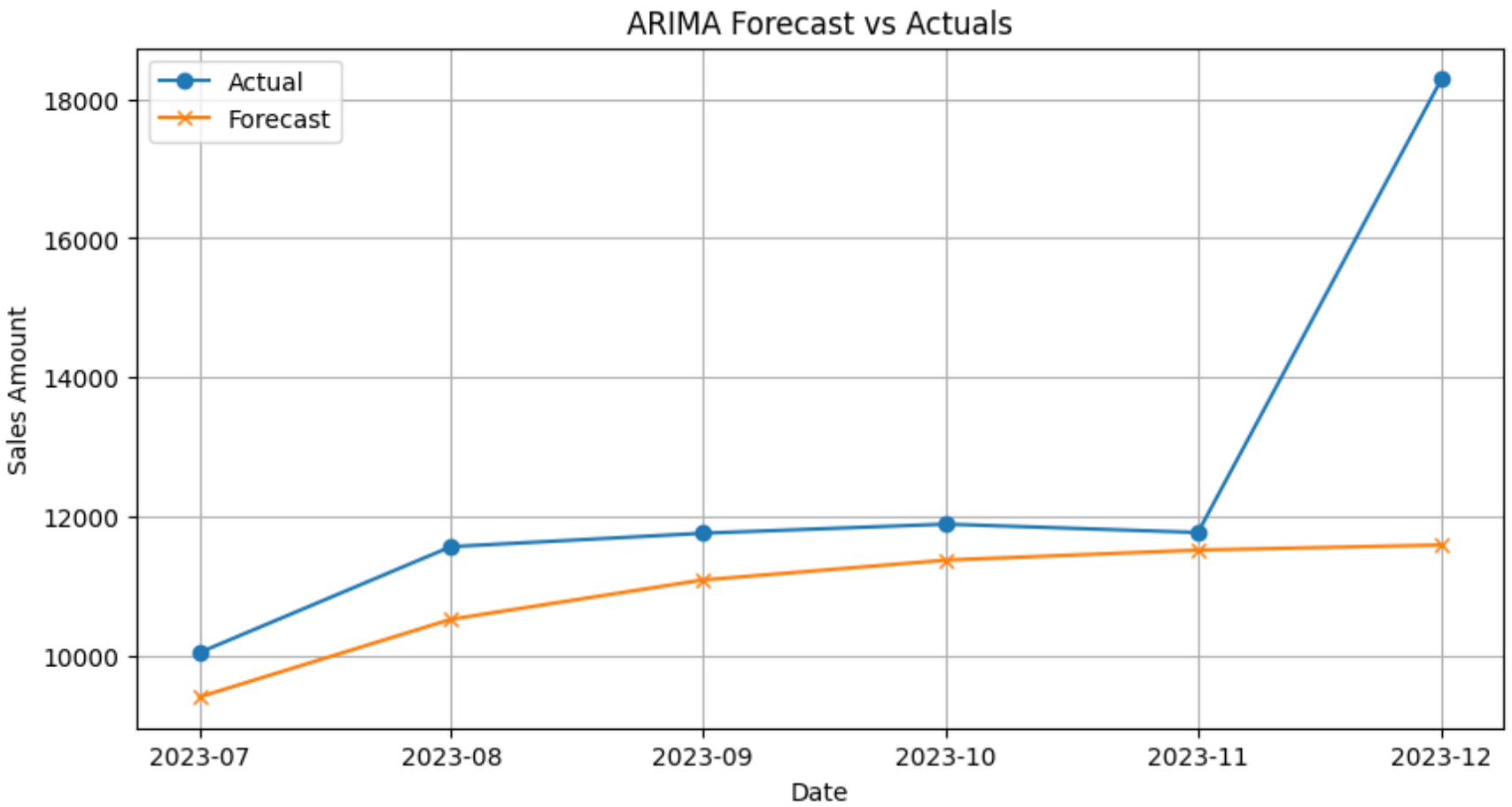
- Finally, I quantified this performance using several metrics

```
Mean Absolute Error (MAE) ARIMA 1,0,1:  1636.9088375517365
Mean Squared Error (MSE) ARIMA 1,0,1: 7858430.97
Root Mean Squared Error (RMSE) ARIMA 1,0,1: 2803.29
Mean Absolute Percentage Error (MAPE) ARIMA 1,0,1: 0.11%
```

Mean Absolute Error (MAE): 1636.91 means on average, the model`s forecasts are off by approximately $1,637 per month.
Root Mean Squared Error (RMSE): 2803.29

This metric is larger than the MAE (2803.29 > 1636.91) because the squaring of the large December error heavily penalizes the RMSE. It indicates that large errors are a significant problem for this model

Mean Absolute Percentage Error (MAPE): 0.11%
The average forecast is off by about 11% of the actual sales value. An 11% average error is generally considered acceptable for many real-world sales forecasts, except for the clear failure in December.
ARIMA_101_Graph



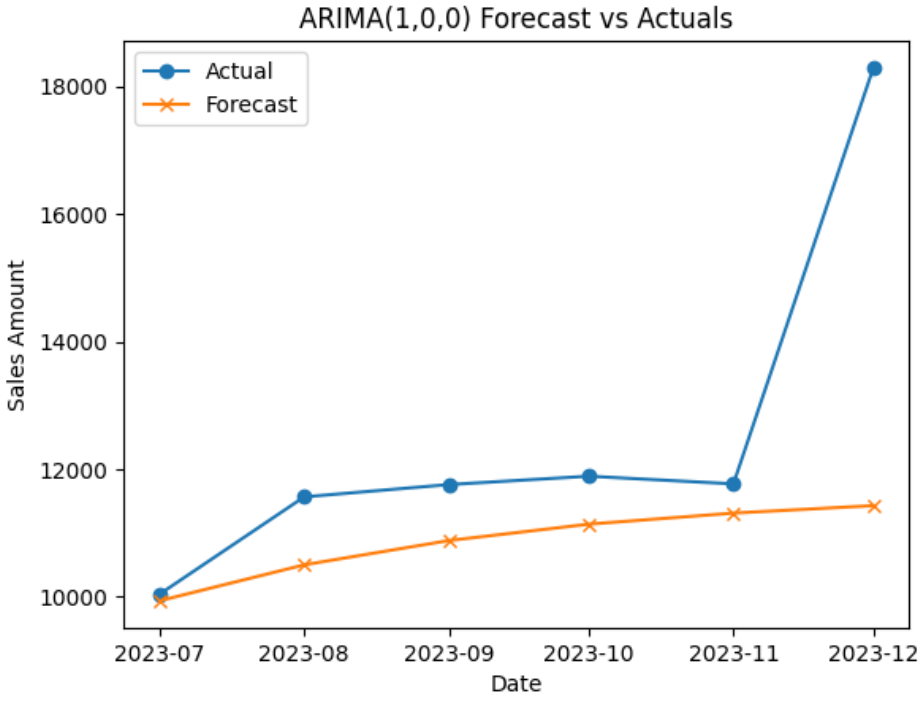
The forecasts, the errors and the above graph clearly demonstrate that the basic ARIMA model is insufficient

Attempting and Failing with Multiple ARIMA Variants

> **ARIMA(1,0,0)**

Date	Actual	Forecast
2023-07-01	10042.0	9938.620311
2023-08-01	11566.0	10501.954242
2023-09-01	11759.0	10882.181171
2023-10-01	11890.0	11138.818505
2023-11-01	11770.0	11312.038011
2023-12-01	18289.0	11428.953959

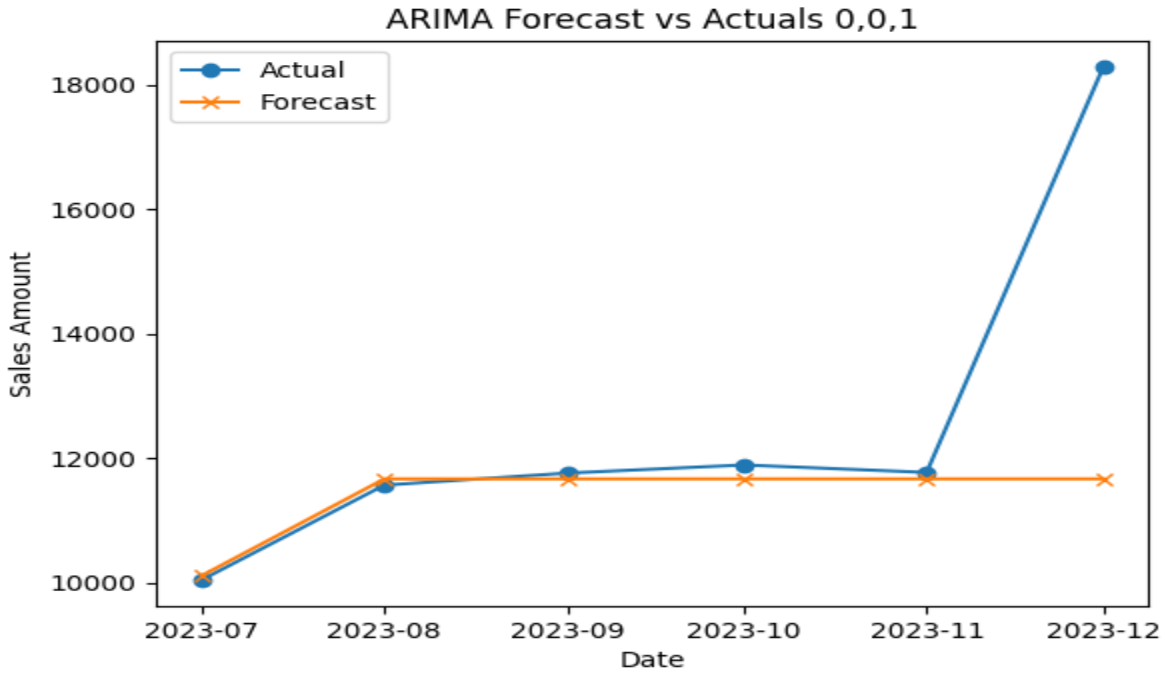
```
Mean Absolute Error (ARIMA(1,0,0)): 1685.57
Mean Squared Error (ARIMA(1,0,0)): 8290987.75
Root Mean Squared Error (ARIMA(1,0,0)): 2879.41
Mean Absolute Percentage Error (ARIMA(1,0,0)): 0.11%
```



> **ARIMA(0,0,1)**

Date	Actual	Forecast
2023-07-01	10042.0	10109.989189
2023-08-01	11566.0	11665.917189
2023-09-01	11759.0	11665.917189
2023-10-01	11890.0	11665.917189
2023-11-01	11770.0	11665.917189
2023-12-01	18289.0	11665.917189

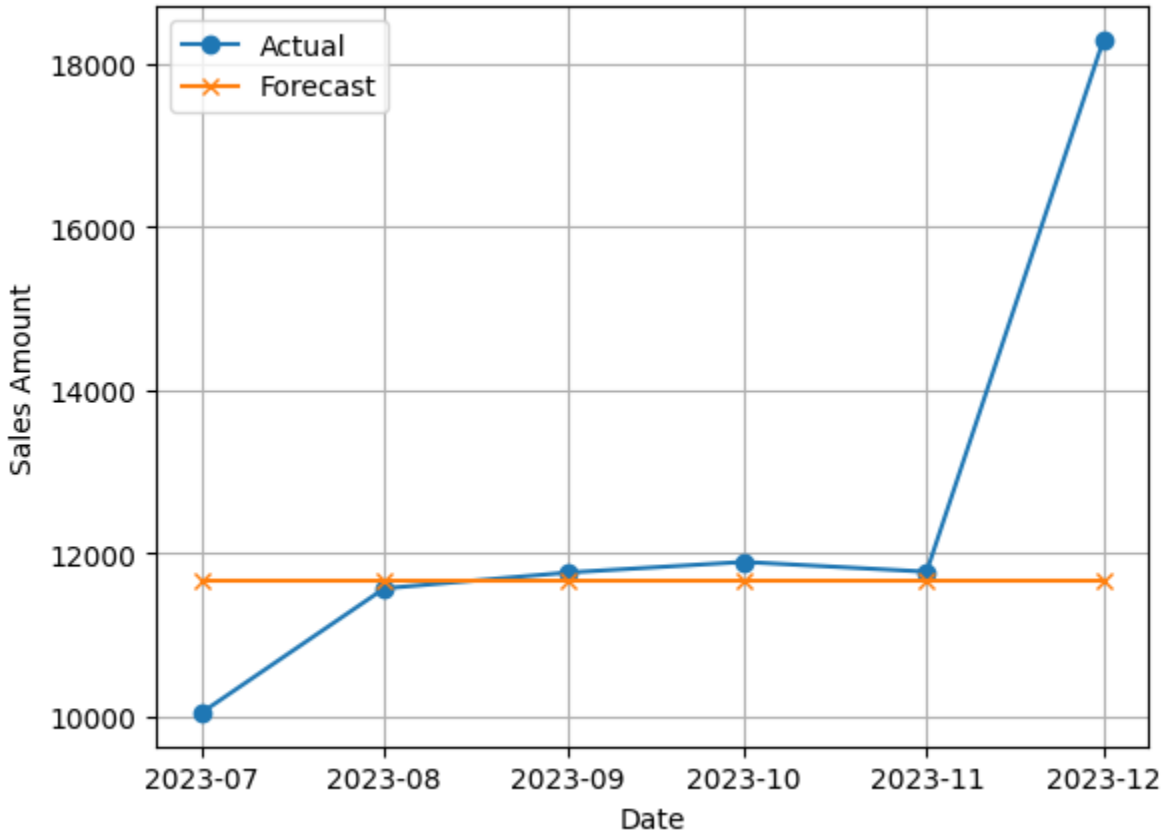
```
Mean Absolute Error (ARIMA(0,0,1)): 1202.04
Mean Squared Error (ARIMA(0,0,1)): 7324923.77
Root Mean Squared Error (ARIMA(0,0,1)): 2706.46
Mean Absolute Percentage Error (ARIMA(0,0,1)): 0.07%
```



> **ARIMA(0,0,0)**

Date	Actual	Forecast
2023-07-01	10042.0	11671.733333
2023-08-01	11566.0	11671.733333
2023-09-01	11759.0	11671.733333
2023-10-01	11890.0	11671.733333
2023-11-01	11770.0	11671.733333
2023-12-01	18289.0	11671.733333

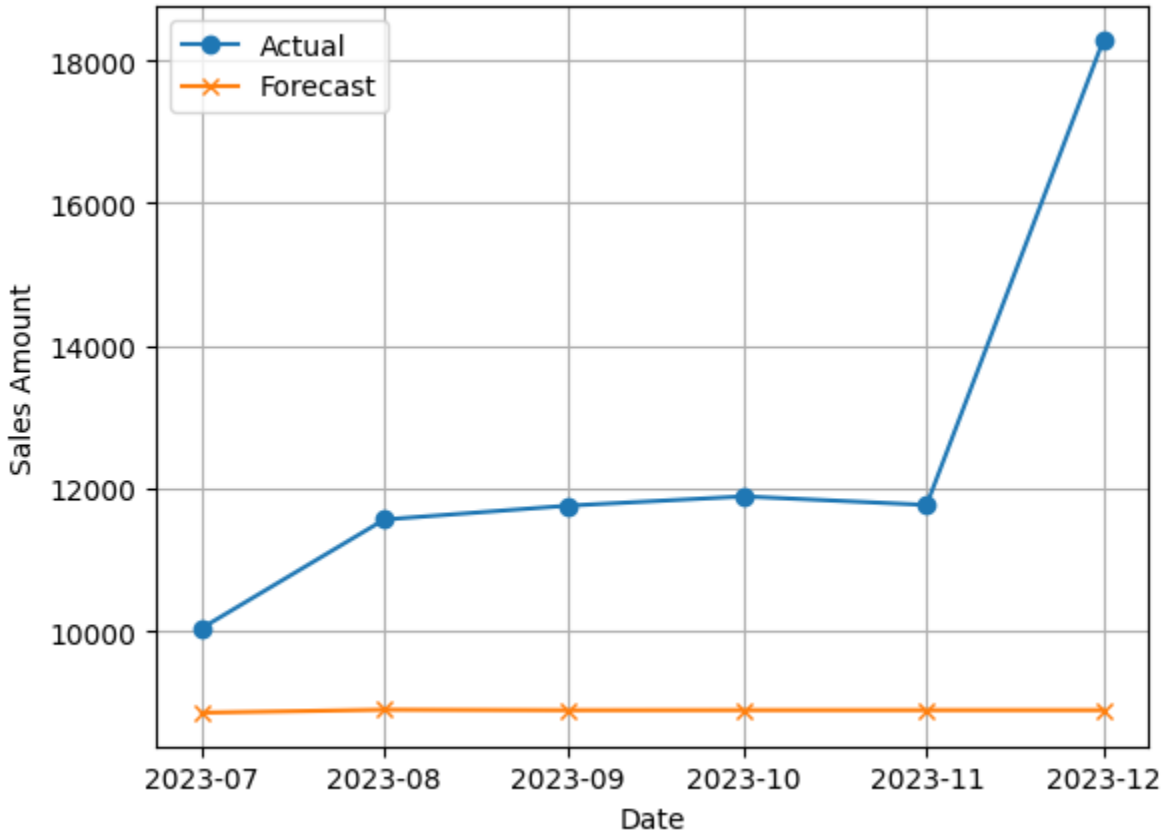
```
Mean Absolute Error ARIMA(0,0,0):  1459.42
Mean Squared Error ARIMA(0,0,0): 7753390.09
Root Mean Squared Error ARIMA(0,0,0): 2784.49
Mean Absolute Percentage Error ARIMA(0,0,0): 0.09%
```



> **ARIMA(1,1,1)**

Date	Actual	Forecast
2023-07-01	10042.0	8858.132869
2023-08-01	11566.0	8899.493398
2023-09-01	11759.0	8892.535602
2023-10-01	11890.0	8893.706064
2023-11-01	11770.0	8893.509165
2023-12-01	18289.0	8893.542288

```
Mean Absolute Error (ARIMA(1,0,0)): 3664.18
Mean Squared Error (ARIMA(1,0,0)): 20375836.58
Root Mean Squared Error (ARIMA(1,0,0)): 2879.41
Mean Absolute Percentage Error (ARIMA(1,0,0)): 0.27%
```



- *Result:* A total failure. I tried multiple ARIMA setups, and they all fell flat. They were completely **blind to the 12-month seasonality** and couldn't predict the December sales spike to save their lives. This proved non-seasonal models were useless for this problem.

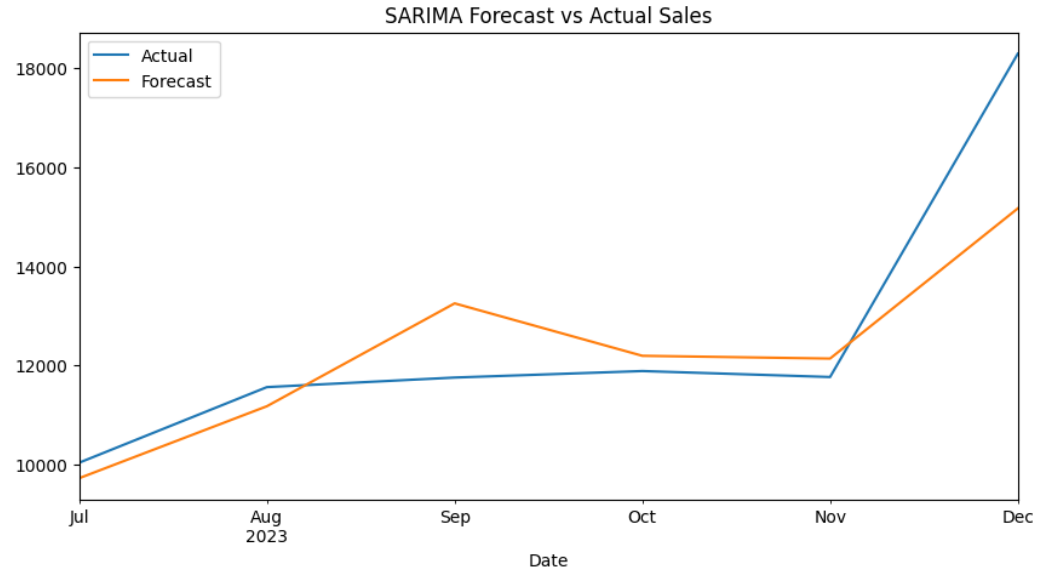
- **Attempt 2: SARIMA**
  - ARIMA models were failing to capture the retail seasonality, I pivoted to the Seasonal ARIMA (SARIMA) model.
  - **Attempting with Multiple SARIMA Variants**

    ** > **SARIMA (1,1,1)(1,1,1,12)**

    The first SARIMA model was a significant improvement, with the RMSE dropping to 1,440.38 and the critical December forecast jumping to 15,169, making it far more accurate than any ARIMA model. **This increased accuracy means supply chain teams can now forecast inventory with much higher confidence, preventing costly stockouts and overstocking.**

Date	Actual	Forecast
2023-07-01	10042.0	9729.294530
2023-08-01	11566.0	11180.591319
2023-09-01	11759.0	13254.831948
2023-10-01	11890.0	12197.537065
2023-11-01	11770.0	12141.895211
2023-12-01	18289.0	15169.478291

```
Mean Absolute Error for SARIMA (1,1,1)(1,1,1,12): , 998.82
Mean Squared Error for SARIMA (1,1,1)(1,1,1,12): , 2074689.76
Root Mean Squared Error for SARIMA (1,1,1)(1,1,1,12): , 1440.38
Mean Absolute Percentage Error (MAPE) SARIMA (1,1,1)(1,1,1,12): 0.07
```



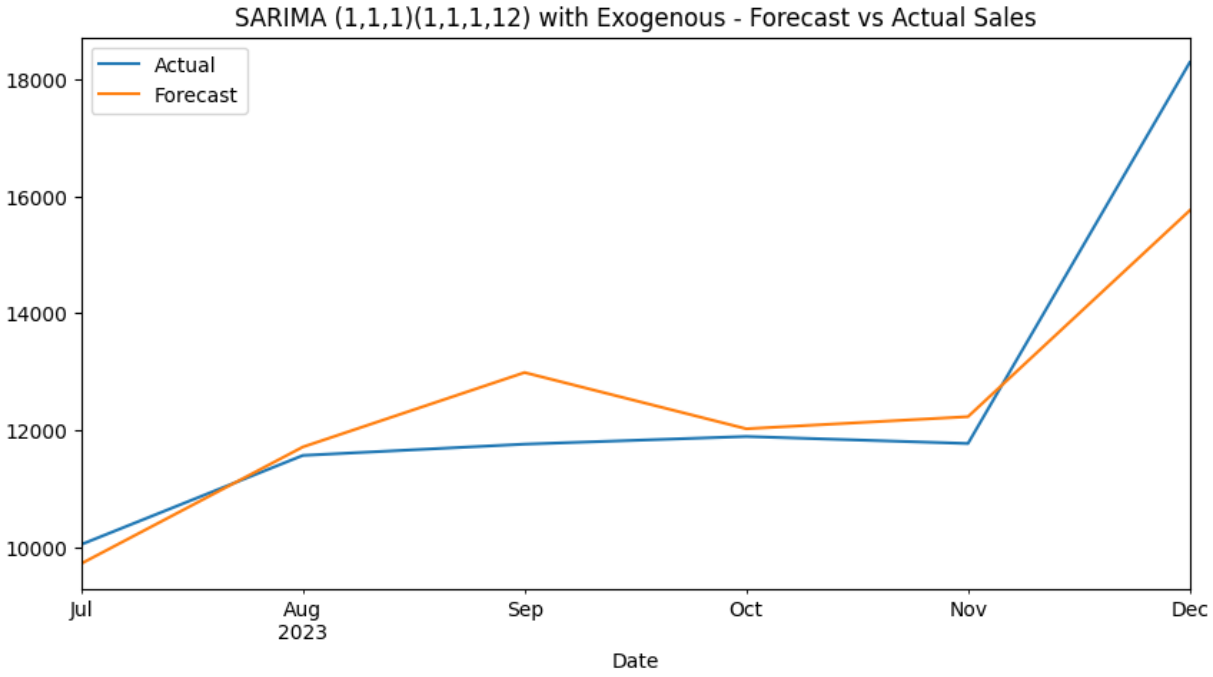
> **SARIMA (1,1,1)(1,1,1,12) with Exogenous Features**

Next, I enhanced the model by adding the Promotion and HolidayMonth flags as exogenous features to create a SARIMAX model.

The SARIMAX(1, 1, 1)(1, 1, 1, 12) model with exogenous features yielded the best result yet, achieving an RMSE of 1,174.10 and pushing the December forecast even closer to the actual value at 15,756.

Date	Actual	Forecast
2023-07-01	10042.0	9714.319019
2023-08-01	11566.0	11710.925311
2023-09-01	11759.0	12984.472151
2023-10-01	11890.0	12021.979361
2023-11-01	11770.0	12228.715447
2023-12-01	18289.0	15756.587852

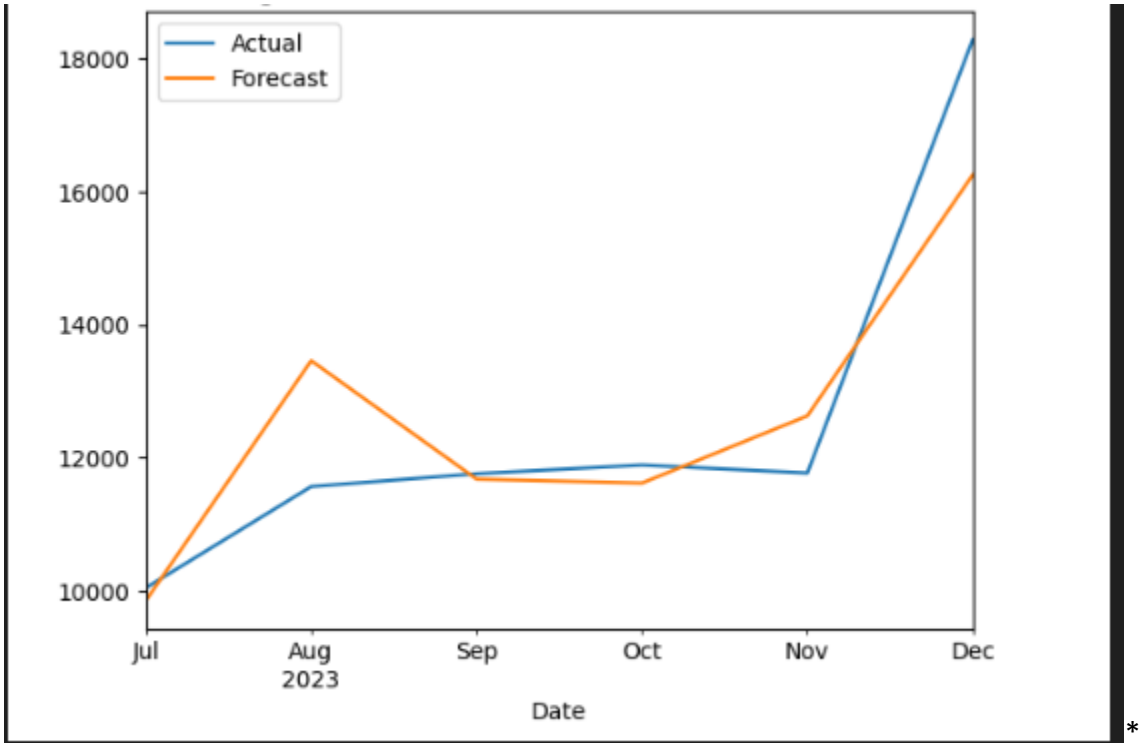
```
Mean Absolute Error for SARIMA exog (1,1,1)(1,1,1,12): 803.53
Mean Squared Error for SARIMA exog (1,1,1)(1,1,1,12): , 1378518.31
Root Mean Squared Error for SARIMA exog (1,1,1)(1,1,1,12): , 1174.10
Mean Absolute Percentage Error (MAPE) SARIMA exog (1,1,1)(1,1,1,12): 0.06
```



> **SARIMA (1,1,2)(1,0,1,12) with Exogenous Features**

Date	Actual	Forecast
2023-07-01	10042.0	9849.459562
2023-08-01	11566.0	13459.884260
2023-09-01	11759.0	11677.780443
2023-10-01	11890.0	11618.324933
2023-11-01	11770.0	12628.918377
2023-12-01	18289.0	16255.703842

```
Mean Absolute Error for SARIMA exog (1,1,2)(1,0,1,12): 888.59
Mean Squared Error for SARIMA exog (1,1,2)(1,0,1,12): , 1429384.57
Root Mean Squared Error for SARIMA exog (1,1,2)(1,0,1,12): 1195.57
Mean Absolute Percentage Error (MAPE) SARIMA exog (1,1,2)(1,0,1,12): 0.07%
```



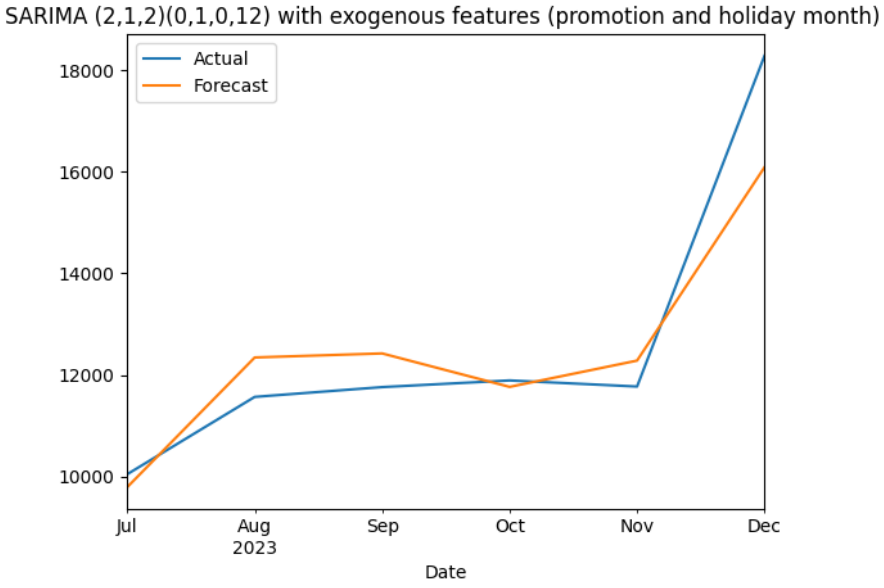
**

> **SARIMA (2,1,2)(0,1,0,12) with Exogenous Features**

This led me to explore SARIMAX(2, 1, 2)(0, 1, 0, 12), which gave the best performance metrics overall with a Mean Absolute Error (MAE) of 754.88 and an RMSE of 1,017.34.
This final model's December forecast was 16,092, very close to the actual 18,289 peak.
Achieving this level of accuracy is a massive win for the business, translating directly into
> optimized inventory levels,
> tighter budget adherence,
> maximum profit
captured during the all-important holiday season.
The use of exogenous features in SARIMAX was the key to moving beyond simplistic time trends to actionable, high-impact business forecasting.

Date	Actual	Forecast
2023-07-01	10042.0	9787.624352
2023-08-01	11566.0	12343.105438
2023-09-01	11759.0	12422.548438
2023-10-01	11890.0	11763.236731
2023-11-01	11770.0	12281.223179
2023-12-01	18289.0	16092.736455

```
Mean Absolute Error for SARIMA exog (2,1,2)(0,1,0,12): 754.88
Mean Squared Error for SARIMA exog (2,1,2)(0,1,0,12): , 1034981.33
Root Mean Squared Error for SARIMA exog (2,1,2)(0,1,0,12): , 1017.34
Mean Absolute Percentage Error (MAPE) for SARIMA exog (2,1,2)(0,1,0,12): 0.05
```



**

* *Result:* This is where the magic happened. SARIMAX is built for this. It combines seasonal components (S) with the ability to add **external regressors** (X). By feeding it the **Promotion** and **HolidayMonth** flags, the model could finally see the *why* behind the numbers. * The final tuned model, **SARIMAX(2, 1, 2)(0, 1, 0, 12) + Exogenous**, blew the others away.

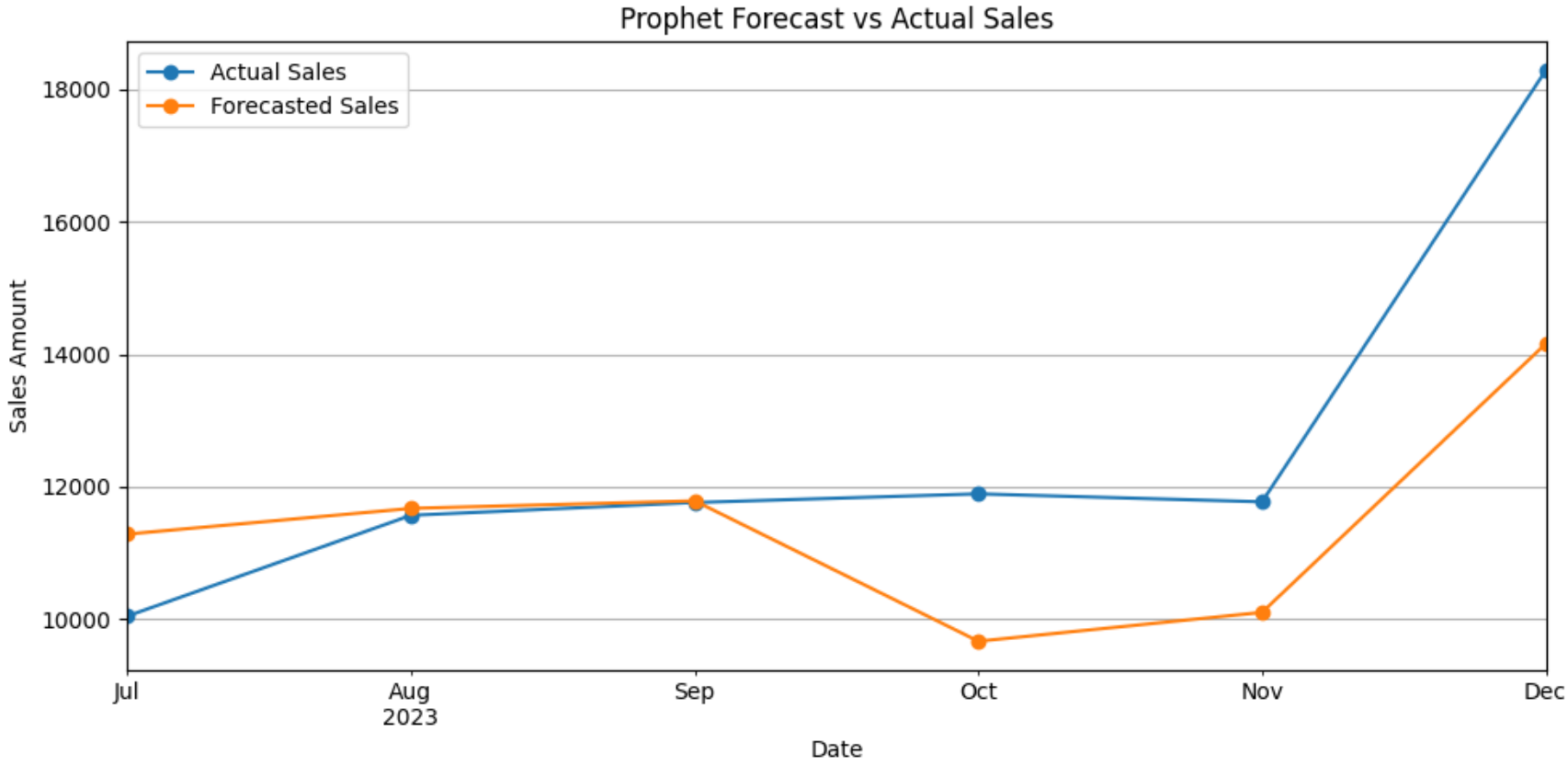
- **Attempt 3: Facebook Prophet**
    - After maximizing the performance of SARIMAX, I realized that I should explore one more class of models i.e. Facebook Prophet.
    - I prepared the data, making a clean copy and renaming my 'Date' and 'SalesAmount' columns to the required **'ds'** and **'y'** so the model could understand the input.
      As the December sales are uniquely high, I defined a custom holiday calendar specifically for those months to capture seasonal spike separately.
    - To prevent mistakes from skewing my predictions, I used the z-score method to detect any potential outliers.
    - Having cleaned and formatted the data, I configured the Prophet model, deliberately turning off the default daily and weekly seasonality since my data was recorded monthly.
    - I coded in such a way that the model looks for a **12-month** custom yearly cycle.
    - Finally, I added the external factors of 'Promotion' and 'HolidayMonth' as special regressors, ensuring the model would consider their influence when generating its future forecast.
- **Attempting with Multiple Prophet Variants**

  ** > **Prophet - Seasonality Mode Additive - Fourier_order = 5**

  My first attempt, using Additive Seasonality with a Fourier Order of 5, was immediately insightful, resulting in a Root Mean Squared Error (RMSE) of 2,096.18. However, the crucial December forecast of 14,158 drastically underestimated the actual peak of 18,289, which could lead to severe inventory shortages during the most profitable month of the year.

	ds	Actual	Forecast
0	2023-07-01	10042.0	11281.62
1	2023-08-01	11566.0	11670.38
2	2023-09-01	11759.0	11784.52
3	2023-10-01	11890.0	9662.71
4	2023-11-01	11770.0	10099.30
5	2023-12-01	18289.0	14158.20

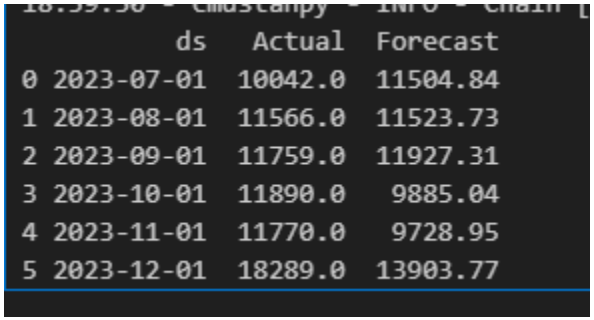
```
Mean Absolute Error for Prophet: 1566.39
Mean Squared Error for Prophet: 4393962.01
Root Mean Squared Error for Prophet: 2096.18
Mean Absolute Percentage Error (MAPE) for Prophet: 0.11%
```



**

> **Prophet - Seasonality Mode Additive - Fourier_order = 10**

I tried increasing the complexity by using a Fourier Order of 10, but this didn't significantly improve the December forecast.



	ds	Actual	Forecast
0	2023-07-01	10042.0	11504.84
1	2023-08-01	11566.0	11523.73
2	2023-09-01	11759.0	11927.31
3	2023-10-01	11890.0	9885.04
4	2023-11-01	11770.0	9728.95
5	2023-12-01	18289.0	13903.77

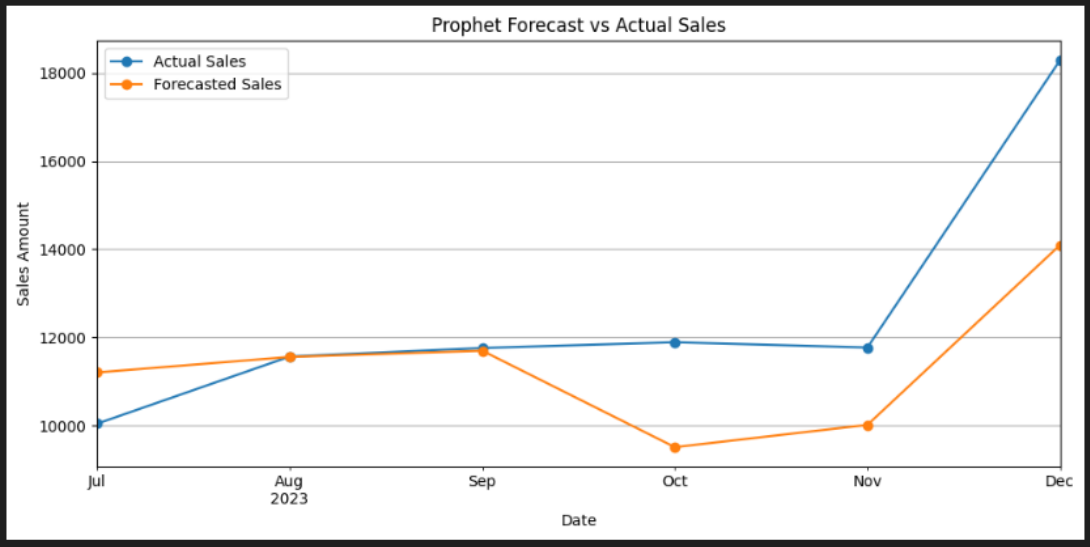
> **Prophet - Seasonality Mode: Multiplicative - Fourier_order = 5**

Next, I tested the Multiplicative Seasonality mode, which is better suited for sales data where the seasonal swing grows as the baseline sales

increased.

	ds	Actual	Forecast
0	2023-07-01	10042.0	11204.46
1	2023-08-01	11566.0	11556.99
2	2023-09-01	11759.0	11697.24
3	2023-10-01	11890.0	9508.41
4	2023-11-01	11770.0	10013.91
5	2023-12-01	18289.0	14094.59

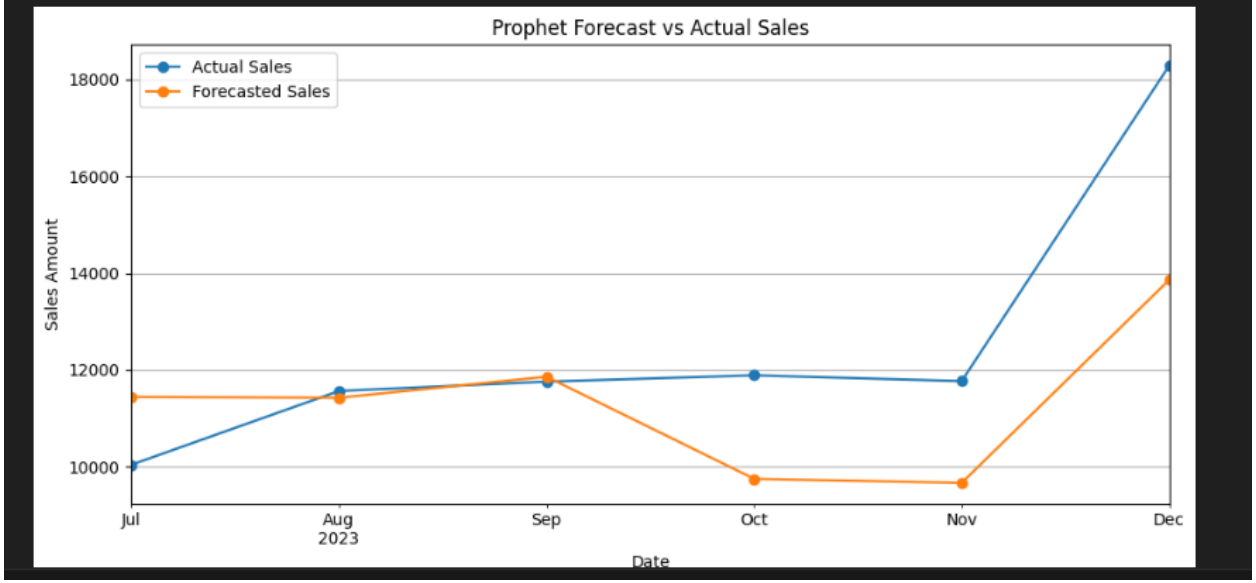
```
Mean Absolute Error for Prophet: 1594.22
Mean Squared Error for Prophet: 4617351.17
Root Mean Squared Error for Prophet: 2148.80
Mean Absolute Percentage Error (MAPE) for Prophet: 0.12%
```



> **Prophet - Seasonality Mode: Multiplicative - Fourier_order = 10**

	ds	Actual	Forecast
0	2023-07-01	10042.0	11443.15
1	2023-08-01	11566.0	11426.31
2	2023-09-01	11759.0	11860.93
3	2023-10-01	11890.0	9750.87
4	2023-11-01	11770.0	9666.50
5	2023-12-01	18289.0	13862.80

```
Mean Absolute Error for Prophet: 1718.60
Mean Squared Error for Prophet: 5097493.37
Root Mean Squared Error for Prophet: 2257.76
Mean Absolute Percentage Error (MAPE) for Prophet: 0.13%
```



Despite this logical shift, both Multiplicative models (with Fourier Orders 5 and 10) actually performed worse, yielding higher RMSEs and lower December forecasts. The best forecast for December came from the initial simple Additive model (Fourier Order 5), which provided a forecast of 14,158.20.

Comparison of Model Performance:

Model	Best Parameters	Best Metric (RMSE)	My Takeaway
ARIMA	(Multiple)	~2,700 - 2800	Inadequate. Fails to model seasonality.
Prophet	Manual Changepoints, prior_scale=0.5	2,096.18	Good and interpretable, but not the most accurate.
SARIMAX	**(2, 1, 2)(0, 1, 0, 12) + Exogenous**	**1,017.34**	**WINNER.** Best accuracy

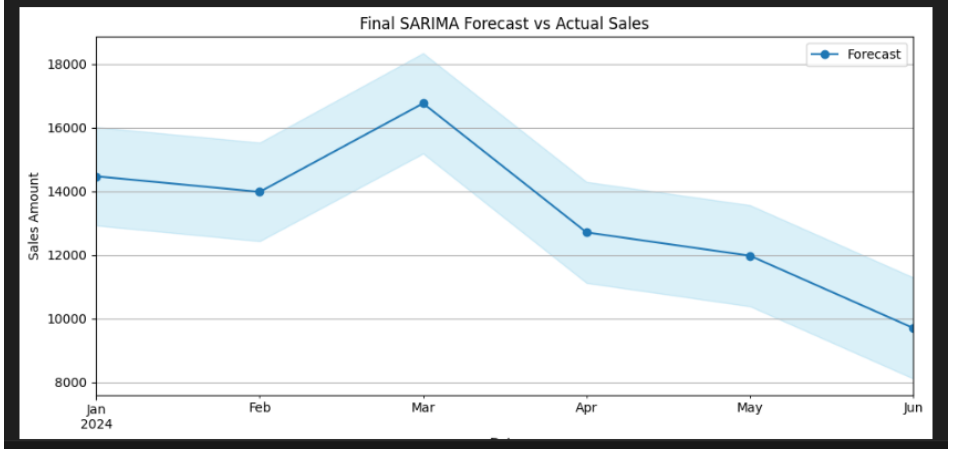
---

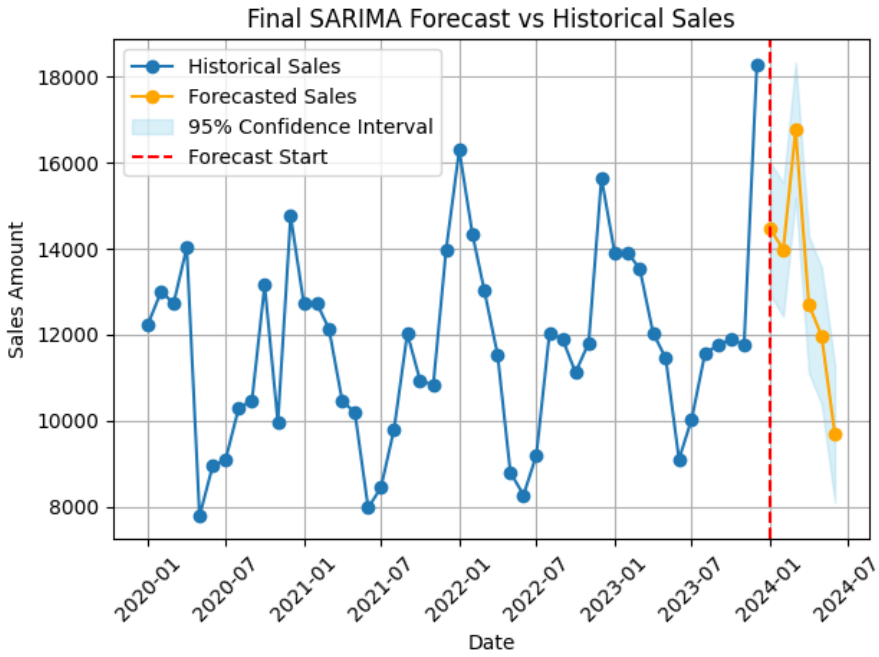
What This Model Delivers

After finding the winner, I retrained the SARIMAX(2, 1, 2)(0, 1, 0, 12) model on the entire 48-month dataset (Jan 2020 through Dec 2023).

- I then constructed a future scenario for the next six months (Jan 2024 to June 2024), inputting a planned schedule of promotions for March and May.
- I used the model to generate a **6-month forecast**.
- The output included a 95% confidence interval (CI), which is the most valuable output for a risk manager;
- For March, the forecast of 16,759 comes with a range of approximately 15,182 to 18,338. This CI tells the business that while 16,759 is the best guess, they must be prepared for sales to potentially reach over 18,000, **giving a clear, data-driven margin for inventory safety stock**.
- The forecast also predicts a sudden drop in in June with sales dropping to 9,698, allowing the purchasing department to **strategically decrease orders and conserve working capital** during the slower summer months.
- This final SARIMAX model, with its proven accuracy and ability to incorporate future marketing actions, is now ready to be deployed as the official tool for all medium-term sales planning, offering significant real-world benefits across the entire supply chain.

	Forecast	Lower CI	Upper CI
2024-01-01	14467.604032	12921.720036	16013.488027
2024-02-01	13978.548775	12430.016577	15527.080973
2024-03-01	16759.838614	15182.048693	18337.628535
2024-04-01	12705.069450	11117.239114	14292.899786
2024-05-01	11972.046325	10383.089915	13561.002736
2024-06-01	9698.501287	8109.820236	11287.182338





This project provides a clear path from reactive, gut-feel decisions to **proactive, data-driven planning.**

🛠️ Tech & Libraries Used

- **Python 3.12.0**
- **pandas:** For all the data wrangling and time series
- **numpy:** For numerical operations and building the dataset.
- **statsmodels:** For time series analysis (ADF, STL, ACF/PACF) , ARIMA, **SARIMAX model**.
- **Scikit-learn** For evaluation metrics
- **Prophet:** For building the Prophet comparison model.
- **matplotlib:** For all the visualizations.
- **missingno:** For a quick visual check on missing data.

Check out the work:

- The main analysis and model building is in the **Forecasting_for_a_Retail_Store.ipynb** notebook.
- The raw data I generated is **Data/retail_sales_mock_data.csv**.
- The data with engineered lag features is **Data/retail_sales_with_lags.csv**.