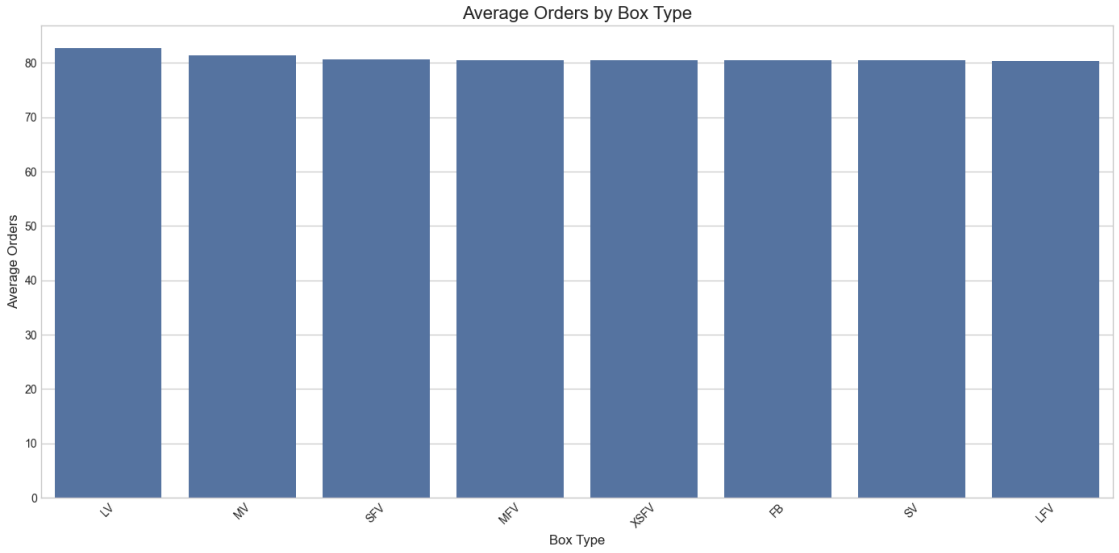
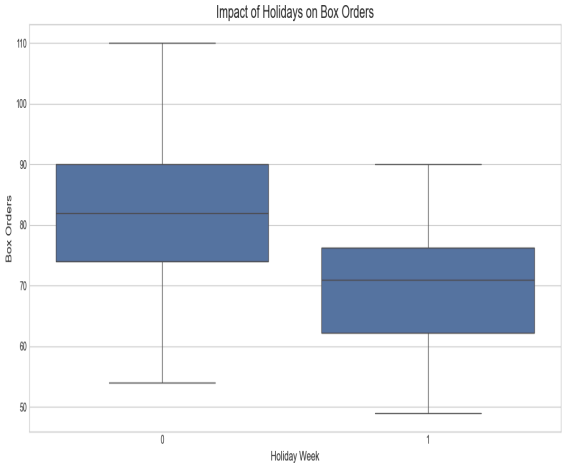
Data Analysis and Feature Engineering

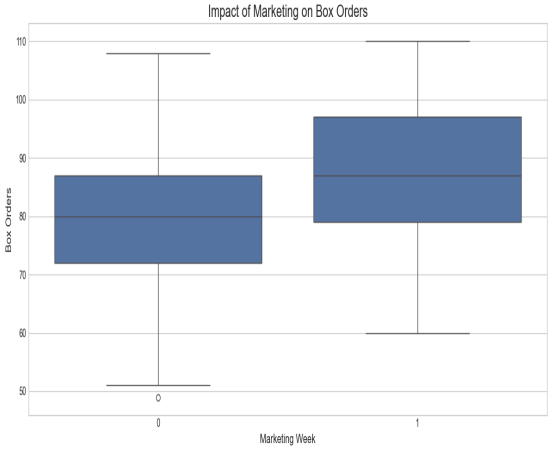
https://github.com/Tofu0142/Oddbox

The data analysis process begins with exploring the dataset, which contains:

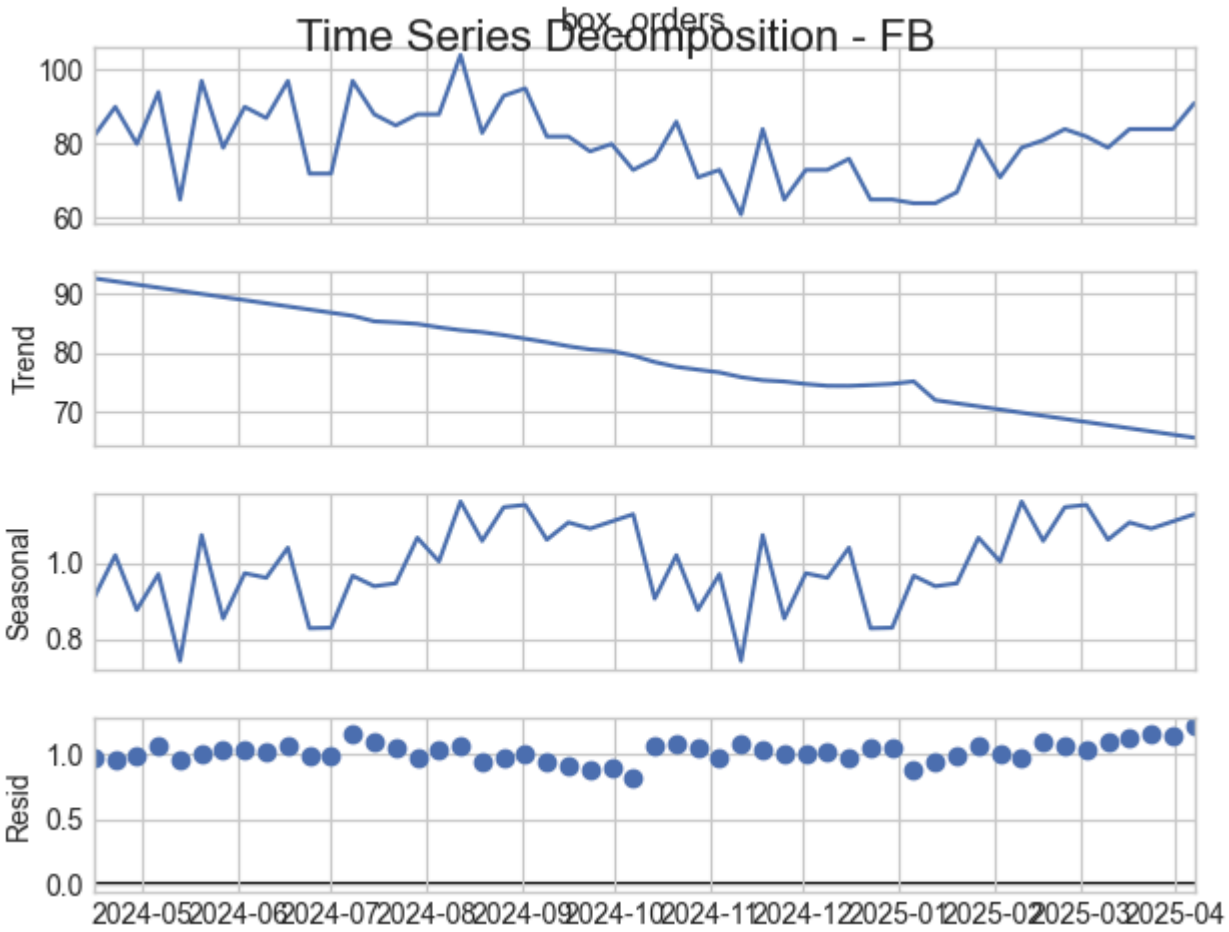
- Weekly box orders for different box types (FB, LFV, LV, etc.)
- Marketing week indicators
- Holiday week indicators
- Subscriber information (weekly and fortnightly)
- Temporal features (month, week of year, etc.)
- Key feature engineering steps include:
- Creating time-based features (month, week_of_year, day_of_year, quarter)
- Generating lag features (previous 1-4 weeks of orders)
- Creating rolling statistics (mean, std over 4-week windows)
- Adding seasonality indicators (is_summer, is_winter, etc.)
- One-hot encoding categorical variables like box_type
- The analysis also includes visualization of:
- Total box orders over time
- Average orders by box type
- Impact of marketing and holiday weeks on orders
- Correlation matrix of features







From the above two figures, marketing and holiday_week, we can find the marketing_week (promotion) and non_holiday week, the order is higher. We can suppose the marketing week and holiday week will affect box order.



Take one box type as an example to show the season trend:

Observed Data (Top Panel)

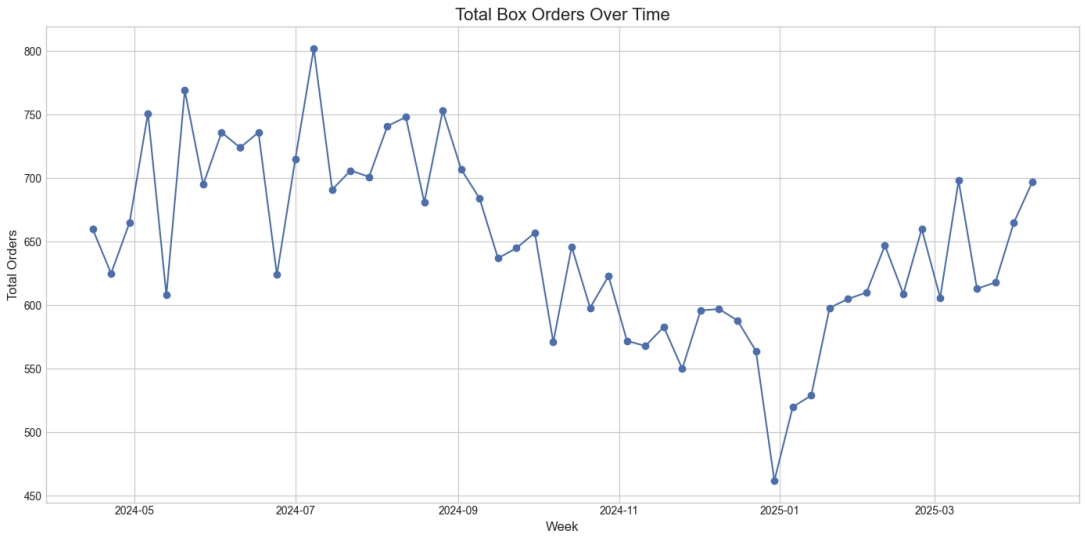
The raw data shows considerable fluctuations in box orders, ranging from approximately 60 to 100 units, with significant week-to-week variability.

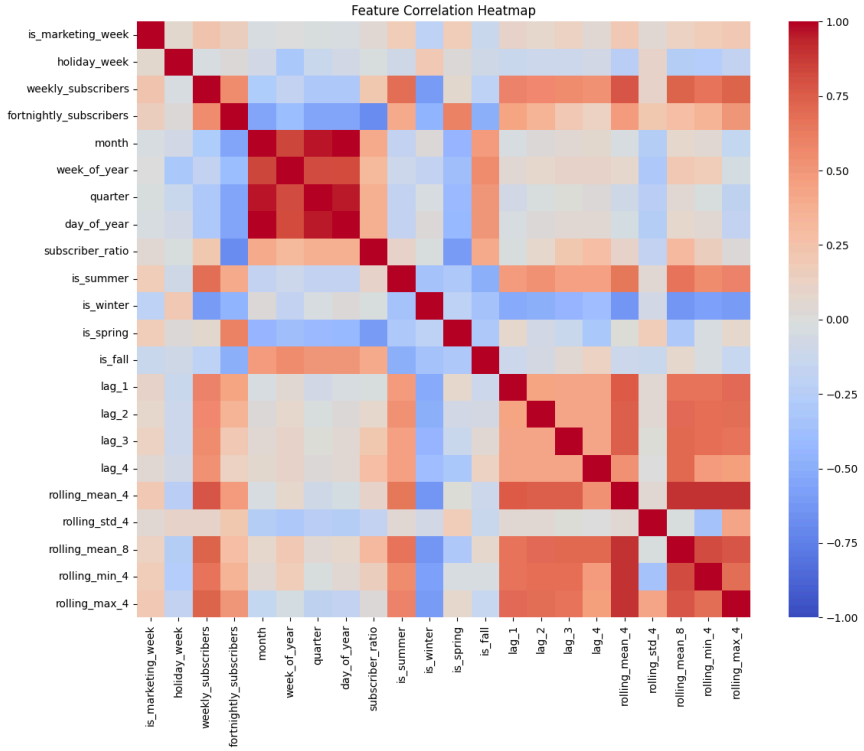
Trend Component (Second Panel)

The trend shows a clear downward trajectory over the entire period, declining from about 90 units to 65 units. This indicates a long-term decrease in demand for FB boxes, suggesting potential market saturation, increased competition, or changing customer preferences.

Seasonal Component (Third Panel)

The seasonal pattern reveals consistent cyclical behavior that repeats approximately every 3-4 months. The peaks (values above 1.0) represent periods when orders typically exceed the trend, while troughs (values below 1.0) show periods of lower seasonal demand.





Forecasting Methods

Multiple forecasting approaches were implemented:

- Machine Learning Models:
    a. Gradient Boosting
    b. XGBoost
- Linear Regression
- K-Nearest Neighbors
- Time Series Models:
    a. ARIMA
    b. Prophet

The prediction of box orders for each box type for the next 4 weeks is implemented through the `generate_forecasts` function. Here's how it works:

Input Requirements:

1. A trained model (ML model or time series model)
2. Processed data containing all necessary features
3. List of box types to generate forecasts for
4. Number of forecast periods (default is 4 weeks)

For Time Series Models (Prophet, ARIMA):

- The function generates a total forecast for all box types combined
- It then distributes this total to individual box types based on their historical proportions
- For Prophet: Creates a future dataframe and uses the model's predict method
- For ARIMA: Uses the forecast method directly

For Machine Learning Models:

- The function forecasts each box type separately
- For each box type:
- It starts with the most recent data point for that box type
- It then iteratively predicts one week at a time, for 4 weeks
- After each prediction, it updates time-based features for the next week
- It also updates lag features and rolling statistics using previous predictions
- This creates a chain of predictions where each forecast influences the next

4. Feature Updates:

- Date features (month, day_of_year, week_of_year, quarter)
- Cyclical features (sine/cosine transformations)
- Lag features (previous 1-4 weeks of orders)
- Rolling statistics (mean, std, min, max of previous 4 weeks)

According to model comparison, the RandomizedSearchCV is the best model, Therefore, I give hyperparameter optimization using RandomizedSearchCV

Parameters tuned: n_estimators, learning_rate, max_depth, min_samples_split, min_samples_leaf, subsample, max_features

Model Evaluation

Models are evaluated using multiple metrics:

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Percentage Error (MAPE)

The best model is selected based on the lowest MAE on the test set.

	GradientBoosting	XGBoost	Linear	KNN	ARIMA
MAE	**7.59**	7.74	7.70	8.66	86.28
RMSE	**9.23**	9.53	9.48	10.38	92.72

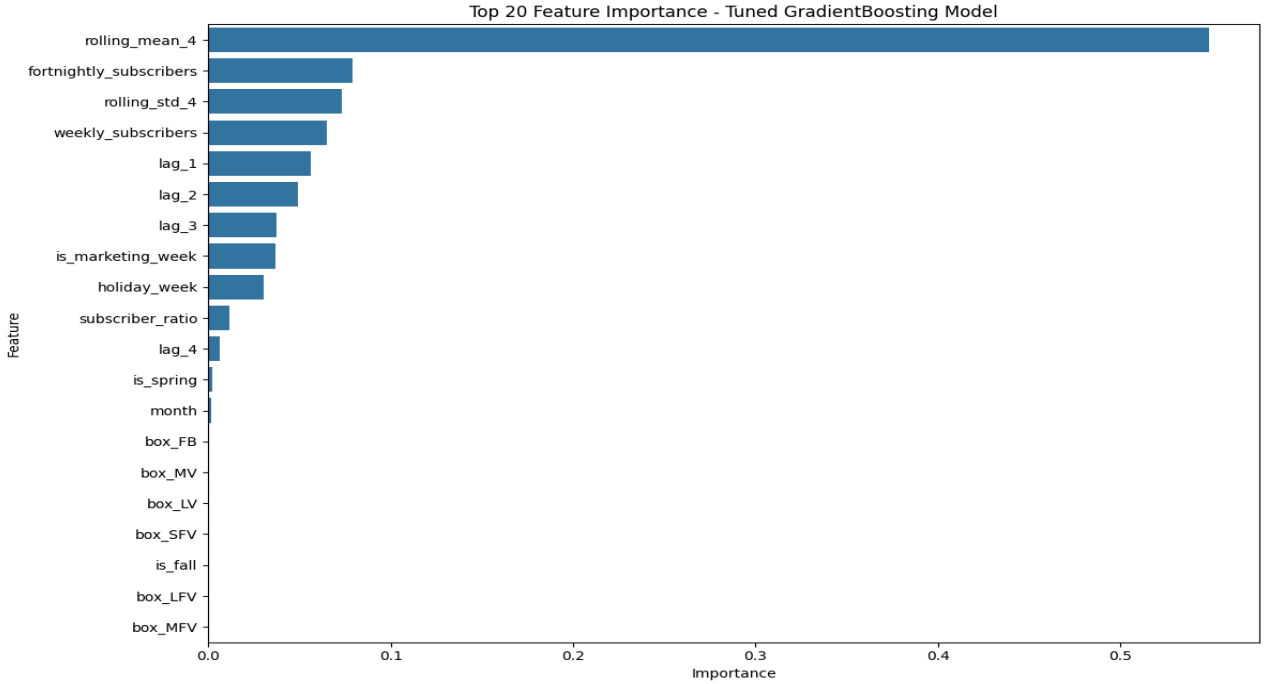
MAPE	**9.68%**	9.93%	9.78%	11.16%	13.24%

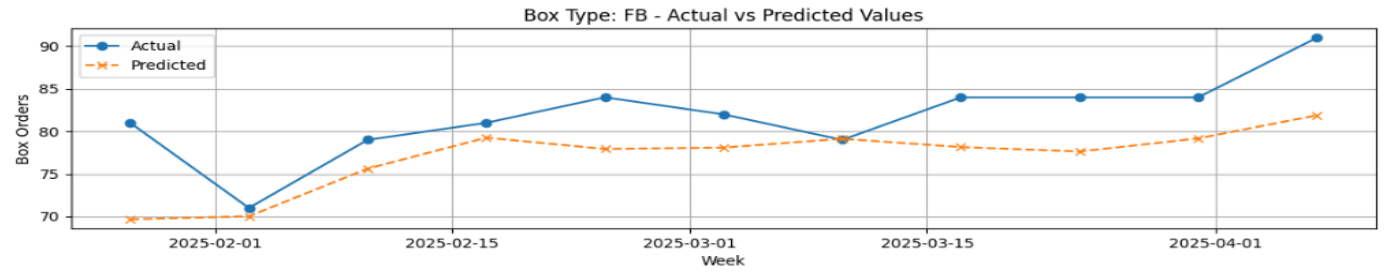
After Fine-tuning:

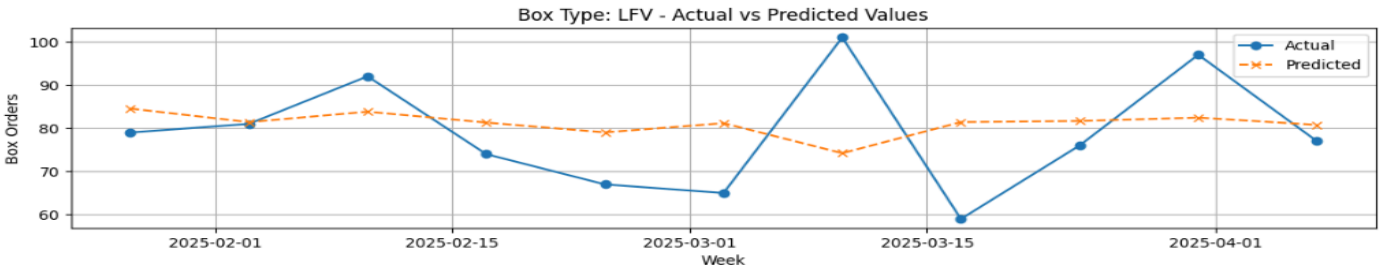
Best MAE: 7.25

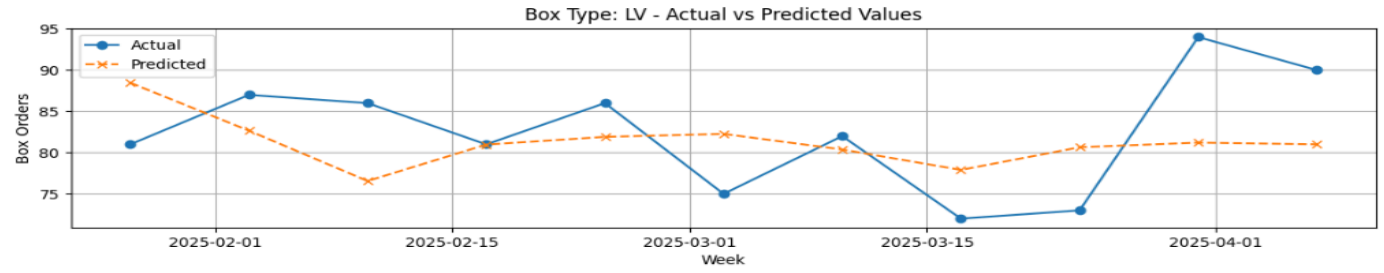
Best RMSE: 8.96

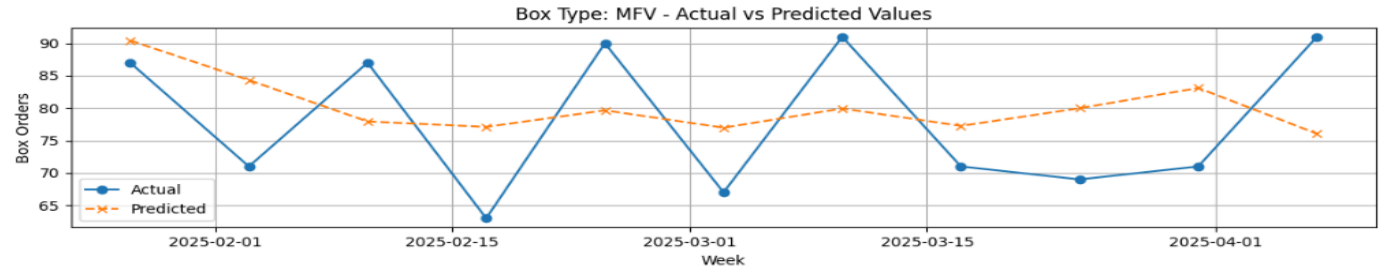
Best MAPE: 9.31%

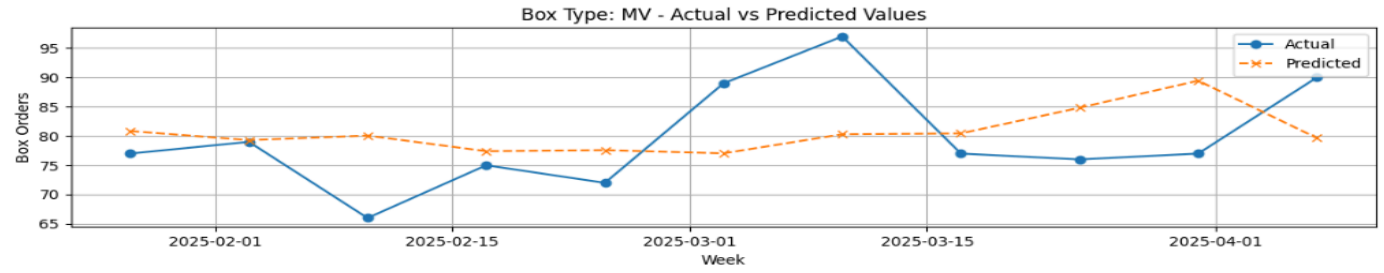


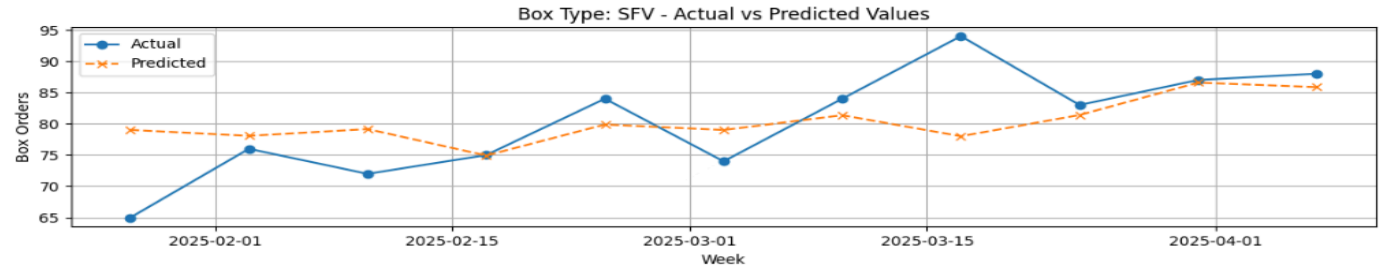

Box Type: FB - Actual vs Predicted Values
Actual
Predicted
Box Orders
Week
2025-02-01
2025-02-15
2025-03-01
2025-03-15
2025-04-01

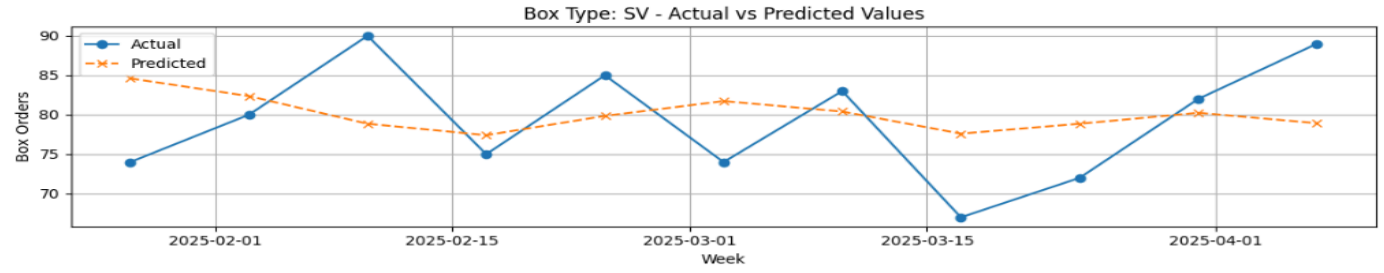

Box Type: LFV - Actual vs Predicted Values
Actual
Predicted
Box Orders
Week
2025-02-01
2025-02-15
2025-03-01
2025-03-15
2025-04-01

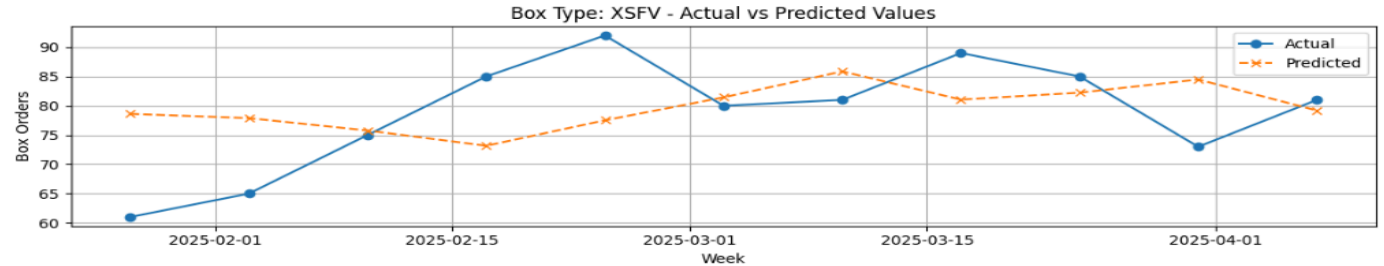

Box Type: LV - Actual vs Predicted Values
Actual
Predicted
Box Orders
Week
2025-02-01
2025-02-15
2025-03-01
2025-03-15
2025-04-01


Box Type: MFV - Actual vs Predicted Values
Actual
Predicted
Box Orders
Week
2025-02-01
2025-02-15
2025-03-01
2025-03-15
2025-04-01


Box Type: MV - Actual vs Predicted Values
Actual
Predicted
Box Orders
Week
2025-02-01
2025-02-15
2025-03-01
2025-03-15
2025-04-01


Box Type: SFV - Actual vs Predicted Values
Actual
Predicted
Box Orders
Week
2025-02-01
2025-02-15
2025-03-01
2025-03-15
2025-04-01


Box Type: SV - Actual vs Predicted Values
Actual
Predicted
Box Orders
Week
2025-02-01
2025-02-15
2025-03-01
2025-03-15
2025-04-01


Box Type: XSFV - Actual vs Predicted Values
Actual
Predicted
Box Orders
Week
2025-02-01
2025-02-15
2025-03-01
2025-03-15
2025-04-01

System Architecture Design

API Endpoint Design

Although not explicitly required in the assignment, as an MLE, I designed API endpoints because:

- They enable the forecasting service to integrate seamlessly with other business systems (inventory management, order processing)
- They separate forecasting logic from consuming applications, allowing independent development and scaling
- The same forecasting service can be used by multiple applications, promoting code reusability
- API versioning enables gradual updates without breaking existing integrations

Single prediction

```
curl -X POST "http://localhost:8000/predict"
     -H "Content-Type: application/json" \
     -d '{
           "box_type": "box_type_A",
           "date": "2023-06-15",
           "additional_features": {
             "subscriber_ratio": 1.5
           }
     }'
```

```
(realtime_interview_copilot) xinzhang@Xins-MacBook-Pro Oddbox % >....
          "date": "2023-06-15",
          "additional_features": {
            "subscriber_ratio": 1.5
          }
        }'
{"box_type":"box_type_A","predicted_orders":78.32,"prediction_date":"2023-06-15","confidence_interval":{"lower_bound":70.48,"upper_
bound":86.15}}
```

Batch prediction

```
curl -X POST "http://localhost:8000/batch-predict" \
     -H "Content-Type: application/json" \
     -d '{
           "predictions": [
             {
               "box_type": "box_type_A",
               "date": "2023-06-15"
             },
             {
               "box_type": "box_type_B",
               "date": "2023-06-16"
             }
           ]
         }'
```

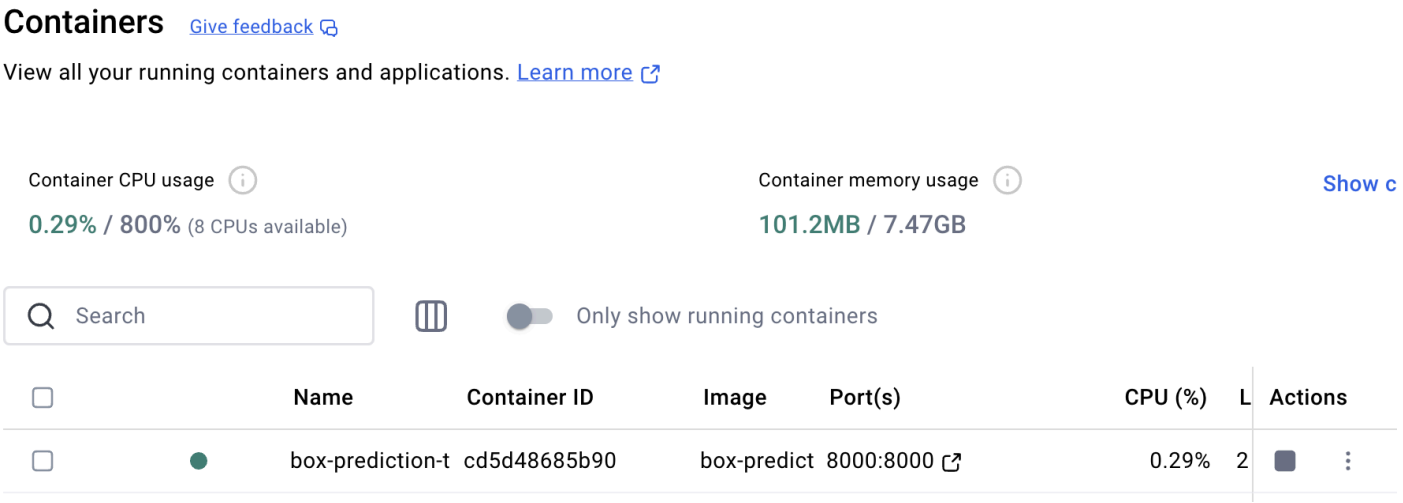
```
        }'
{"predictions":[{"box_type":"box_type_A","predicted_orders":78.32,"prediction_date":"2023-06-15","confidence_interval":{"lower_boun
d":70.48,"upper_bound":86.15}},{"box_type":"box_type_B","predicted_orders":78.92,"prediction_date":"2023-06-16","confidence_interva
l":{"lower_bound":71.03,"upper_bound":86.81}}],"model_version":"1.0.0","timestamp":"2025-04-16T10:05:04.108079"}
```

Docker Containerization

As an MLE position, I chose to containerize the model deployment using Docker because:

- It ensures the application runs consistently across development, testing, and production environments
- It packages all dependencies within the container, avoiding conflicts with other applications

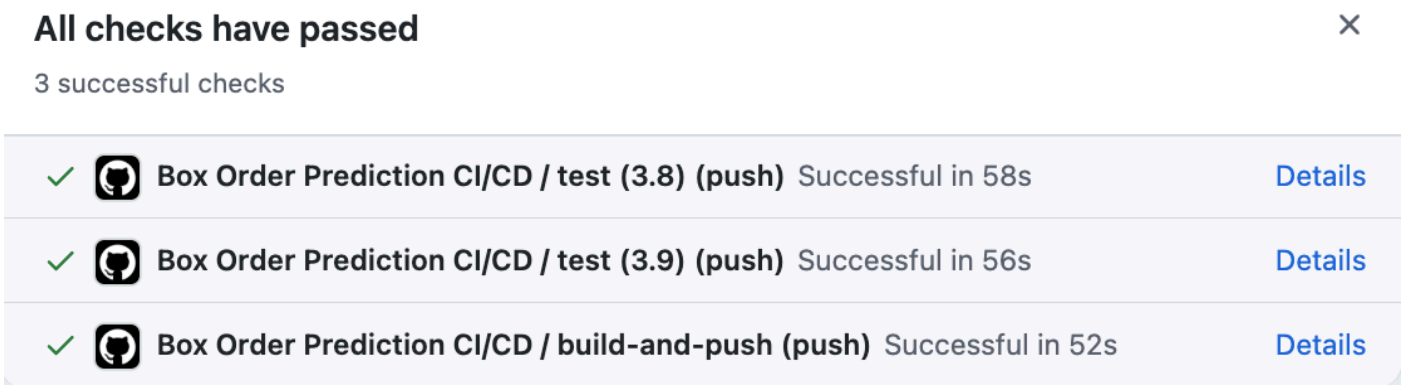
- It facilitates horizontal scaling by easily deploying multiple identical containers
- Containers are lightweight compared to virtual machines, allowing more efficient resource utilization
- Containers can run on any system with Docker installed, regardless of the underlying OS



GitHub CI/CD Pipeline

Implementing a CI/CD pipeline is a key responsibility for me as an MLE because:

- It automates testing, building, and deployment processes, reducing manual errors and saving time
- It ensures all tests pass before deployment, maintaining code quality
- It enables quick feedback on code changes and facilitates frequent, small updates
- It standardizes the deployment process, ensuring it's performed the same way every time
- Each deployment is tied to a specific commit, making it easy to track what changes were deployed
- It simplifies rolling back to previous versions if issues are detected



Why Go Beyond the Basic Requirements?

While the assignment may have only required basic data analysis and model building, as a professional MLE, I believe in providing complete end-to-end solutions. In a real work environment, a model's value is only realized when it can be reliably deployed and integrated into business processes.

Assumptions Made

1. Temporal Patterns: The approach assumes that past patterns in box orders will continue in the future.

Feature Relevance: The models assume that engineered features (lags, rolling statistics, seasonality) capture the relevant information for forecasting.

Stationarity: For time series models like ARIMA, there's an implicit assumption of stationarity after differencing.

Independence: The models assume that observations for different box types are independent.

5. Data Quality: The approach assumes that the historical data is representative and of good quality.

Improvements with More Time or Data

With more time or data, I would:

Explore More Advanced Models:

- Deep learning approaches (LSTM, Transformer-based models)
- Hybrid models combining statistical and ML approaches

Enhanced Feature Engineering:

- Create more interaction features
- Incorporate external data (weather, economic indicators)
- Develop more sophisticated lag structures

Hierarchical Forecasting:

- Implement hierarchical forecasting to ensure consistency across different box types
- Use reconciliation methods to improve forecast accuracy

Ensemble Methods:

- Create ensemble models combining different forecasting approaches
- Implement stacking or blending techniques

Uncertainty Quantification:

- Develop prediction intervals to quantify forecast uncertainty
- Implement probabilistic forecasting methods

Cross-Validation:

- Use more sophisticated time series cross-validation techniques
- Implement rolling-origin evaluation

Incorporating Future Known Events

To incorporate future known events:

Feature Engineering:

- Create binary indicators for future holidays, promotions, etc. Like I currently did
- Develop distance-based features (days until/since holiday)
- Create interaction features between events and other variables

Event Impact Modeling:

- Model the typical impact of different event types based on historical data
- Create event-specific models for major recurring events

Measuring Forecast Performance in Production

In production, I would measure forecast performance using:

Accuracy Metrics:

- MAE, RMSE, MAPE for overall accuracy
- Weighted metrics that prioritize high-volume box types

Bias Metrics:

- Mean Error to detect systematic over/under-forecasting
- Tracking bias over time to detect drift

Business Impact Metrics:

- Inventory costs associated with forecast errors
- Service level (percentage of demand met)
- Revenue impact of stockouts or overstock

Comparative Analysis:

- Compare against baseline methods (naive forecast, moving average)
- Compare against previous forecasting system

Monitoring System:

- Implement automated monitoring for forecast accuracy
- Set up alerts for significant deviations from expected performance

Visualization Dashboard:

- Create dashboards showing forecast vs. actual over time
- Visualize performance by box type, time period, etc.

Regular Retraining and Evaluation:

- Schedule regular model retraining
- Evaluate if model performance degrades over time