Strategic Demand Forecasting in the Automotive Sector: A Custom Multiplicative SARIMA Implementation

Diego Coglievina Díaz
Juan Jesús Orihuela Torres

December 13, 2025

Abstract

This technical report documents the end-to-end development of a proprietary forecasting engine designed to optimize inventory management and negotiation strategies within the automotive supply chain. Departing from standard library implementations, this project introduces a custom **Multiplicative Seasonal Autoregressive Integrated Moving Average (SARIMA)** model optimized via Gradient Descent. The methodology integrates a rigorous grid search with rolling-origin cross-validation to select optimal hyperparameters for each customer segment. The solution architecture includes a robust data engineering pipeline, reproducible experiment tracking via MLflow, and a scalable Flask API for real-time inference. Results demonstrate that the model successfully captures complex seasonal dynamics and outperforms seasonal baselines across most key segments.

Contents

1 Business Context and Problem Definition

1.1 Operational Landscape

The organization operates within the high-stakes ecosystem of automotive parts manufacturing and distribution. This sector is characterized by a multi-echelon supply chain with diverging demand patterns across its primary channels. Specifically, the business manages three critical segments:

- **Original Equipment Manufacturers (OEM):** Characterized by contractual, high-volume production schedules where failure to supply results in severe line-down penalties.
- **Dealer Network (Aftermarket):** A highly volatile channel driven by end-consumer behavior, break-fix cycles, and seasonality, representing the highest risk for forecast error.
- **Professional End-Users:** A recurrent, service-oriented channel requiring consistent availability.

1.2 Problem Statement

The core business challenge is the **inefficiency of capital allocation due to demand volatility**. Historically, reliance on reactive heuristics or static baselines (e.g., Seasonal Naive) has led to a dichotomy of operational risks:

1. **Capital Immobilization (Overstocking):** To buffer against volatility in the Dealer segment, the organization maintains excessive safety stock, tying up liquidity that could be deployed elsewhere.
2. **Margin Erosion via Reverse Logistics:** A lack of visibility into future *Returns* volume creates financial blindsides. Returns in this industry are not merely logistical events but fiscal reversals involving tax credits and inventory depreciation.

1.3 Solution Value and Strategic Justification

This project implements a proprietary forecasting engine designed to transition the organization from reactive planning to predictive optimization. The value of this custom solution rests on three pillars:

1.3.1 1. Precision in Volatile Markets via Regularization

Off-the-shelf statistical packages often fail to converge or overfit when applied to highly volatile series like the Dealer segment. By building the SARIMA algorithm from scratch using **Gradient Descent with L2 Regularization**, we enforce stability in the model parameters. This directly translates to more reliable long-term forecasts (12-month horizon), allowing the supply chain team to reduce safety stock levels without compromising service rates.

1.3.2 2. Net Margin Visibility (Dual-Target Prediction)

Unlike traditional systems that forecast only "Net Sales," this solution treats **Sales** and **Returns** as independent, competing stochastic processes. By forecasting them separately and computing the net flow at inference time, the Finance department gains:

- **Fiscal Anticipation:** Accurate projections of tax liabilities and credit issuance related to returns.
- **Real Revenue Forecasting:** A true picture of cash flow, filtering out the noise of high-volume return periods (e.g., end-of-year inventory adjustments by dealers).

1.3.3 3. Automated Negotiation Baseline

The deployment of this model via a REST API allows for the integration of unbiased, data-driven baselines into contract negotiations. When negotiating annual volume commitments with OEMs or Dealers, sales executives can rely on a statistically rigorous projection rather than intuition, shifting the negotiation leverage in favor of the organization.

2 Exploratory Data Analysis (EDA)

Before modeling, a comprehensive analysis of the raw data was conducted to understand distributions, trends, and seasonal patterns. This step is critical for determining the necessary transformations and the order of the SARIMA model.

2.1 Distribution Analysis

The distribution of transaction amounts was analyzed to identify skewness and potential outliers. As shown in Figure 1, the data is highly right-skewed, indicating that while most transactions are of lower value, there are significant high-value outliers that could bias a linear model if not addressed via transformation.

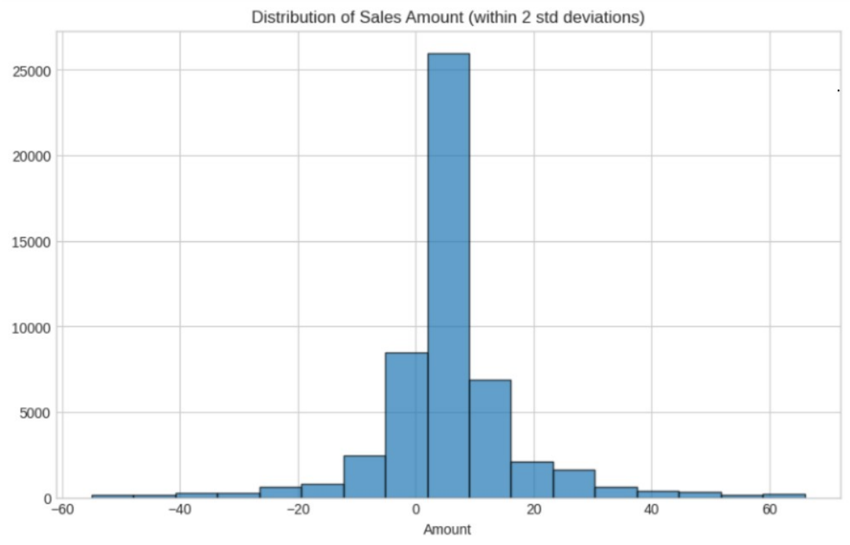


Figure 1: Distribution of Sales Amounts showing significant right-skewness.

2.2 Time Series Structure

The aggregated sales time series (Figure 2) reveals non-stationary behaviors, specifically a changing mean (trend) and variance over time.

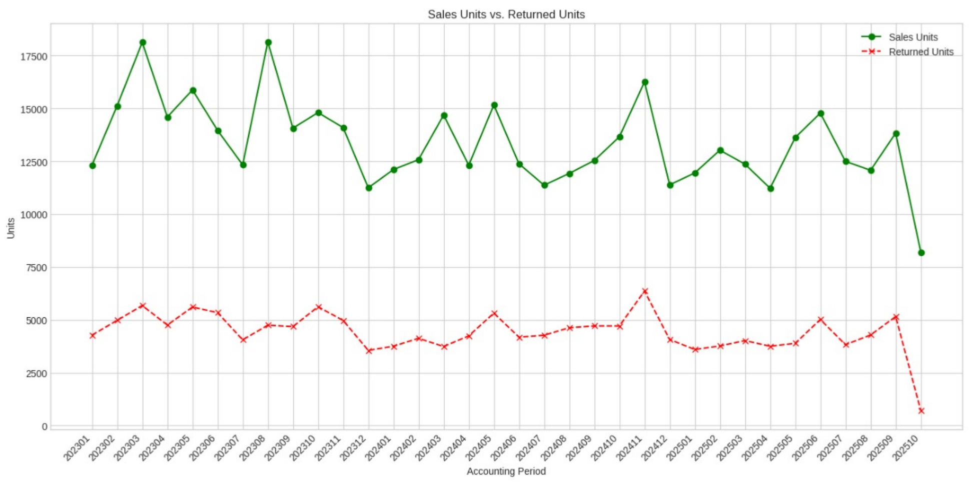
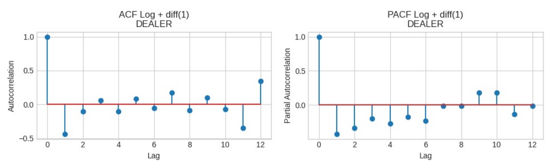


Figure 2: Aggregated Sales Time Series. Note the variance instability over time.

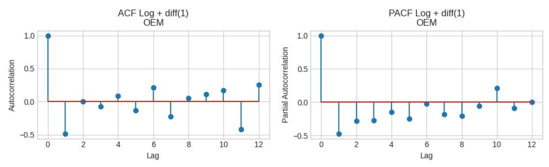
2.3 Autocorrelation Analysis (ACF)

The Autocorrelation Function (ACF) is the primary tool for identifying seasonality and the order of Moving Average (MA) terms. Theoretically, a seasonal series will show spikes at lags that are multiples of the seasonal period s.

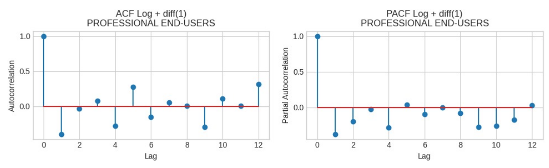
Figures 3 displays the ACF for the three key segments. In all cases, we observe significant correlations at regular intervals, confirming the presence of seasonality.



(a) ACF: Dealer Segment



(b) ACF: OEM Segment



(c) ACF: Professional End Users

Figure 3: Autocorrelation Functions per Segment showing seasonal patterns.

Finally, Figure 4 shows the ACF of the total sales after a log transformation. This step helps visualize the persistence of the series after variance stabilization.

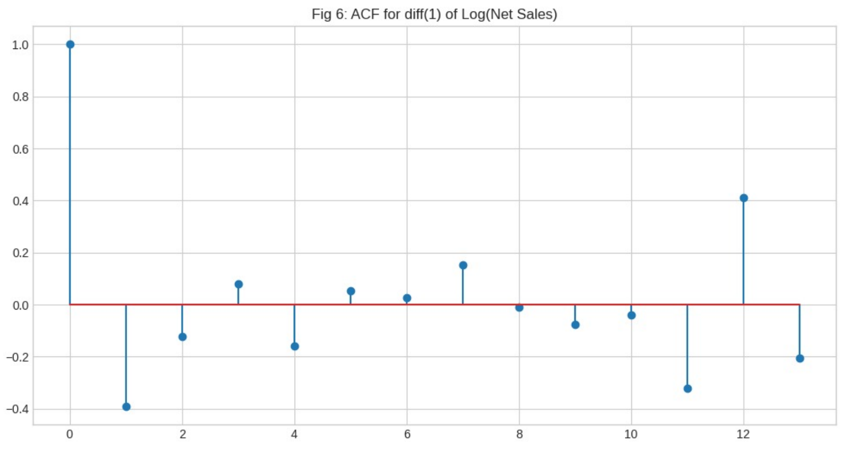


Figure 4: ACF of Log-Transformed Total Sales.

3 Data Engineering and Transformation Pipeline

The raw source is a transactional table at **daily granularity**. Each record represents units posted for a given business segment and accounting period. Since ARIMA-family models assume a **regularly spaced time index** and behave best when the input is close to **stationary** after transformation, the project implements a deterministic pipeline that converts raw transactions into clean **monthly** time series. This logic is implemented in `01_data_transformation.py` and validated in `00_data_exploration.py`.

3.1 Monthly panel construction

The first stage converts daily events into a monthly panel per segment. Transactions are aggregated into monthly totals and split into two distinct targets:

- **Net Sales** (`net_sales`): the monthly sum of all positive unit movements.
- **Returns** (`returns`): the monthly sum of the absolute value of negative unit movements, expressed as a positive magnitude.

This separation is intentional because sales and returns often follow different dynamics and seasonal patterns. Treating them as independent targets improves interpretability and prevents cancellation effects that can hide structure when modeling a single net series.

Finally, the pipeline enforces a complete monthly calendar for each segment. A continuous monthly index is created from the first to the last observed accounting period, and any missing month is filled with zero units. This step is operationally critical: seasonal differencing and SARIMA recursion require that all lagged months exist (notably at lags 12, 24, etc.). Without a complete grid, differencing either fails or implicitly changes the effective time axis.

3.2 Stationarity-oriented transformations

After aggregation, each monthly series is transformed to better satisfy ARIMA assumptions.

First, a **shifted log transform** is applied to stabilize variance in the presence of level-dependent volatility. Because returns and low-volume months can produce non-positive values, a data-dependent shift constant c is used to ensure the log argument is strictly positive:

$$y'_t = \log(y_t + c). \tag{1}$$

The shift is chosen per series so that the smallest observed value becomes at least 1 prior to applying $\log(\cdot)$, avoiding undefined values while preserving relative differences on the transformed scale.

Second, **differencing** removes predictable mean structure. ACF inspection supports an annual seasonal span $S = 12$ for monthly data, so a seasonal difference is applied:

$$z_t = y'_t - y'_{t-12}. \tag{2}$$

If residual trend remains, a first regular difference is applied to the seasonally adjusted series:

$$w_t = z_t - z_{t-1}. \tag{3}$$

The resulting working series $\{w_t\}$ is the stationary signal used by the SARIMA recursion and estimation routines. Inverse transformations are applied after inference to return forecasts to business units.

3.3 Diagnostics and parameter selection

Transformations are validated and the differencing orders are chosen using standard time-series diagnostics:

- **ACF/PACF** to confirm the seasonal structure and to verify that differencing removes low-frequency dependence,
- **ADF and KPSS** tests as complementary stationarity checks, since their null hypotheses point in opposite directions and reduce the risk of over-differencing or under-differencing.

This stage determines the seasonal span S and the differencing orders (d, D) used later when specifying the SARIMA model.

4 Mathematical Framework: ARIMA and Multiplicative SARIMA

This section defines the full stochastic model used in this project. We start from non-seasonal ARIMA and then extend the same construction to the multiplicative seasonal model SARIMA. Throughout, we are modeling a *univariate discrete-time stochastic process*, not a deterministic sequence.

4.1 Objects and notation

Let $\{Y_t\}_{t\in\mathbb{Z}}$ be a real-valued stochastic process. Each Y_t is a random variable, and the observed time series $\{y_t\}$ is a single realization of $\{Y_t\}$.

A central tool is the **backshift operator** B, which acts on time-indexed random variables by shifting them backward in time:

$$BY_t = Y_{t-1}, \qquad B^k Y_t = Y_{t-k}. \tag{4}$$

The operator acts on the time index, not on coefficients (coefficients are fixed scalars). This matches the standard notational conventions used in time-domain ARIMA modeling. *(See the definition and remarks on B in the course notes [1].)*

4.2 AR, MA, and ARMA as stochastic difference equations

We define an **innovation** (or **white noise**) sequence $\{w_t\}$ as a sequence of i.i.d. random variables with

$$\mathbb{E}[w_t] = 0, \qquad \mathrm{Var}(w_t) = \sigma_w^2, \tag{5}$$

often modeled as Gaussian $w_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_w^2)$ for likelihood-based inference [2].

Autoregressive model AR(p). An AR(p) model states that the current value depends linearly on its last p values plus an innovation:

$$X_t = \mu + \phi_1(X_{t-1} - \mu) + \cdots + \phi_p(X_{t-p} - \mu) + w_t, \tag{6}$$

where $\mu \in \mathbb{R}$ is a constant mean level and $\phi_i \in \mathbb{R}$ are AR coefficients.

Using operator notation, define the **AR polynomial**

$$\phi_p(B) = 1 - \phi_1 B - \cdots - \phi_p B^p. \tag{7}$$

Then the AR(p) model can be written compactly as

$$\phi_p(B)(X_t - \mu) = w_t. \tag{8}$$

This polynomial form is the standard entry point for ARIMA notation [1].

Moving-average model MA(q). An MA(q) model states that the current value is a linear combination of the current and past q innovations:

$$X_t = \mu + w_t + \theta_1 w_{t-1} + \cdots + \theta_q w_{t-q}, \tag{9}$$

with MA coefficients $\theta_k \in \mathbb{R}$.

Define the **MA polynomial**

$$\theta_q(B) = 1 + \theta_1 B + \cdots + \theta_q B^q, \tag{10}$$

so that

$$X_t - \mu = \theta_q(B) w_t. \tag{11}$$

Many references and software packages use a sign-flipped MA convention; the model class is the same, but coefficient signs change accordingly, so the chosen convention must be stated explicitly [2].

ARMA(p, q). Combining both mechanisms yields ARMA(p, q):

$$\phi_p(B)(X_t - \mu) = \theta_q(B) w_t. \tag{12}$$

Invertibility restrictions are typically enforced by software when estimating MA terms, because they ensure a stable and identifiable representation [2].

4.3 Differencing and ARIMA(p, d, q)

ARMA assumes (weak) stationarity in the mean. When the observed process $\{Y_t\}$ is nonstationary due to trend, we apply differencing.

Define the **non-seasonal difference operator** ∇ by

$$\nabla = 1 - B, \qquad \nabla Y_t = (1 - B)Y_t = Y_t - Y_{t-1}. \tag{13}$$

Higher-order differencing is repeated application, $\nabla^d = (1 - B)^d$ [3].

Let the differenced process be

$$X_t = \nabla^d Y_t. \tag{14}$$

Then an ARIMA(p, d, q) model is defined as an ARMA(p, q) model for the differenced series $\{X_t\}$:

$$\phi_p(B)(X_t - \mu) = \theta_q(B) w_t \qquad \text{with} \qquad X_t = \nabla^d Y_t. \tag{15}$$

4.4 Seasonality and SARIMA$(p,d,q) \times (P,D,Q)_S$

Now assume the data have a seasonal span S (for example, $S = 12$ for monthly data with yearly seasonality). Seasonal nonstationarity is handled via **seasonal differencing**:

$$\nabla_S = 1 - B^S, \qquad \nabla_S Y_t = (1 - B^S)Y_t = Y_t - Y_{t-S}. \tag{16}$$

When both trend and seasonality are present, both operators may be applied:

$$X_t = \nabla^d \nabla_S^D Y_t. \tag{17}$$

The course notes emphasize this combined differencing as the standard preparation step before fitting seasonal AR and MA terms [4].

Seasonal and non-seasonal polynomials. Define the seasonal AR and MA polynomials as functions of B^S:

$$\Phi_P(B^S) = 1 - \Phi_1 B^S - \cdots - \Phi_P B^{PS}, \qquad \Theta_Q(B^S) = 1 + \Theta_1 B^S + \cdots + \Theta_Q B^{QS}. \tag{18}$$

Multiplicative SARIMA. A multiplicative seasonal ARIMA model applies both the non-seasonal and seasonal polynomials, multiplying them on the AR side and on the MA side:

$$\Phi_P(B^S)\, \phi_p(B)\, (X_t - \mu) = \Theta_Q(B^S)\, \theta_q(B)\, w_t, \qquad X_t = \nabla^d \nabla_S^D Y_t. \tag{19}$$

This multiplicative structure is the defining feature of SARIMA notation [4].

4.5 Why multiplication creates interaction lags

Because the seasonal and non-seasonal polynomials multiply, expanding them produces **interaction lags** of the form $i + jS$.

For example, the AR-side expansion of a $(1,0,0) \times (1,0,0)_{12}$ structure is

$$(1 - \phi_1 B)(1 - \Phi_1 B^{12}) = 1 - \phi_1 B - \Phi_1 B^{12} + \phi_1 \Phi_1 B^{13}. \tag{20}$$

The appearance of B^{13} shows that the multiplicative model induces dependence at lag 13 even though neither component individually included lag 13. The seasonal notes explicitly highlight that seasonal and non-seasonal components multiply, which is exactly what generates these cross-lag terms [4].

A symmetric phenomenon occurs on the MA side. For a $(0,0,1) \times (0,0,1)_{12}$ model, multiplication yields MA terms at lags 1, 12, and 13, and the notes use this as a canonical example [4].

4.6 Parameter estimation as regularized least squares with SGD

Let $\boldsymbol{\Omega} \in \mathbb{R}^K$ denote the parameter vector that contains all coefficients being optimized (non-seasonal, seasonal, and any explicitly implemented interaction-lag coefficients).

At each time t, define a feature vector $\mathbf{x}_t \in \mathbb{R}^K$ that collects the relevant lagged predictors implied by the expanded SARIMA recursion (lagged X_{t-i} terms and lagged innovations or residual proxies). The one-step-ahead conditional predictor is written as a linear form

$$\widehat{X}_t = \mu + \boldsymbol{\Omega}^\top \mathbf{x}_t, \tag{21}$$

and the one-step-ahead error (innovation proxy) is

$$e_t(\boldsymbol{\Omega}) = X_t - \widehat{X}_t. \tag{22}$$

We estimate $\mathbf{\Omega}$ by minimizing a regularized sum of squared one-step errors:

$$J(\mathbf{\Omega}) = \frac{1}{2}\sum_{t=1}^{T} e_t(\mathbf{\Omega})^2 + \frac{\lambda}{2}\|\mathbf{\Omega}\|_2^2, \tag{23}$$

where $\lambda \geq 0$ is the L2 regularization strength.

A stochastic-gradient step (using one time index t_k at iteration k) takes the form

$$\mathbf{\Omega}^{(k+1)} = \mathbf{\Omega}^{(k)} - \eta\left(\nabla_{\mathbf{\Omega}}\frac{1}{2}e_{t_k}^2 + \lambda\mathbf{\Omega}^{(k)}\right) = (1-\eta\lambda)\mathbf{\Omega}^{(k)} + \eta\, e_{t_k}\,\mathbf{x}_{t_k}, \tag{24}$$

with learning rate $\eta > 0$.

Regularization shrinks coefficients toward zero, which helps control numerical instability in long recursive forecasting, especially when MA-style recursion uses estimated past errors that can amplify sensitivity to parameter changes.

5 Model Selection and MLOPS Architecture

5.1 Hyperparameter Tuning Strategy: Grid Search

Selecting the correct model order $(p, d, q) \times (P, D, Q)$ is non-trivial. While ACF/PACF plots provide a heuristic starting point, they are often ambiguous for complex seasonal data. To address this, we implemented a **Grid Search** from scratch.

This search algorithm iterates through a manually specified subset of the hyperparameter space, building a "grid" of all possible combinations of parameters and training a distinct model for each point on the grid.

In our implementation, we fixed the differencing parameters $(d = 1, D = 1)$ based on stationarity tests and defined the search space for the remaining terms:

- Non-seasonal AR (p) and MA (q): Ranges $\{0, 1\}$ and $\{0, 1, 2\}$
- Seasonal AR (P) and MA (Q): Ranges $\{0, 1\}$

For each combination, the model is evaluated using **Rolling-Origin Cross-Validation**. This technique respects the temporal order of data, training on $t_0 \dots t_k$ and predicting $t_{k+1} \dots t_{k+h}$, then sliding the window forward. The configuration with the lowest average RMSE across all folds is selected for the final production model.

5.2 Deployment (Flask REST API)

The trained forecasting models are exposed through a RESTful inference service implemented with **Flask**. Conceptually, the API has two responsibilities: (i) *model management* (discover, select, and load MLflow artifacts into memory), and (ii) *forecast generation* (serve predictions as a time-indexed vector for downstream consumers).

5.2.1 Objects and service state

The API manipulates the following objects:

- **Segments (categorical identifiers).** A segment selects a business partition (e.g., Dealer, OEM, Professional End-Users). Segments are treated as *strings* and are validated against a fixed allow-list.

- **Targets (categorical identifiers).** A target selects which time series is being forecast (e.g., `net_sales`, `returns`). Targets are treated as *strings* and are validated against a fixed allow-list.
- **MLflow runs (model artifacts).** A run is identified by a `run_id`. Each run stores parameters (e.g., SARIMA orders), metrics (e.g., RMSE), and a serialized model artifact.
- **Loaded model registry (in-memory mapping).** The server maintains a dictionary keyed by `model_key := segment_target`. Each entry contains: the loaded model object, the `run_id`, and a `loaded_at` timestamp. This is the state that inference endpoints depend on.

5.2.2 Cross-cutting concerns

JSON contract. All inference and model-management interactions use JSON request bodies and JSON responses. Requests that omit a body, omit required fields, or fail validation return `HTTP 400`.

Rate limiting (per-client, in-memory). To prevent resource exhaustion, requests are throttled per client IP. The server tracks recent request timestamps in a sliding time window, and rejects excess requests with `HTTP 429` plus a `Retry-After` header (in seconds). Because this limiter is in-memory, it resets on server restart and does not coordinate across multiple replicas.

5.2.3 Endpoint catalog and detailed contracts

`GET /health` **(liveness and service metadata).** This endpoint answers whether the server is running and reports how many models are currently loaded into memory.

```
GET /health
```

Response (200):

```
{
  "status": "healthy",
  "timestamp": "YYYY-MM-DDTHH:MM:SS.ssssss",
  "loaded_models_count": <integer>
}
```

`GET /models` **(discover available MLflow runs).** This endpoint queries MLflow for finished runs, extracts run parameters and metrics, and returns them as a list. Optional query parameters allow filtering by segment and/or target.

```
GET /models
GET /models?segment=DEALER
GET /models?target=returns
GET /models?segment=DEALER&target=returns
```

Response (200):

```
{
  "total_count": <integer>,
  "filters_applied": { "segment": "<string-or-null>", "target": "<string-or-null>" },
  "models": [
    {
      "run_id": "<string>",
      "run_name": "<string>",
      "start_time": "YYYY-MM-DDTHH:MM:SS",
      "segment": "<string>",
      "target": "<string>",
      "sarima_order": "(p,d,q)(P,D,Q)[s]",
      "metrics": {
        "sarima_RMSE": <number-or-null>,
        "sarima_MAPE": <number-or-null>,
        "beats_baseline": <number-or-null>
      }
    }
  ]
}
```

`GET /models/loaded` **(inspect in-memory loaded models).** This endpoint returns the server-side registry of loaded models, keyed by `model_key := segment_target`.

```
GET /models/loaded
```

Response (200):

```
{
  "loaded_models": {
    "DEALER_net_sales": {
      "run_id": "<string>",
      "segment": "DEALER",
      "target": "net_sales",
      "loaded_at": "YYYY-MM-DDTHH:MM:SS.ssssss"
    }
  },
  "count": <integer>
}
```

`POST /models/load` **(load a specific run for inference).** This endpoint materializes a model artifact from MLflow into server memory. After loading, inference can be requested for the corresponding `segment` and `target`.

```
POST /models/load
Content-Type: application/json
```

Request body:

```
{
  "run_id": "<string>",
  "segment": "<string>",
  "target": "<string>"
}
```

Responses:

- **200** on success:

  ```
  { "success": true, "message": "<string>", "model_key": "SEGMENT_TARGET" }
  ```

- **400** if any field is missing or invalid.
- **500** if MLflow loading fails (artifact missing, incompatible environment, corrupted run, etc.).

`POST /models/auto-load` **(bulk load "best" runs for all segment-target pairs).** This endpoint loads one model per admissible `(segment, target)` pair. In this implementation, "best" is operationally defined as the *first* matching run returned by the server's MLflow search ordering; therefore, the selection criterion is only as strong as the ordering rule used when fetching runs.

```
POST /models/auto-load
```

Response (200):

```
{
  "success": true,
  "message": "Auto-loaded <integer> models",
  "loaded_models": { ...same structure as GET /models/loaded... }
}
```

`POST /predict` **(single-series forecast).** This is the primary inference endpoint. It returns a monthly forecast vector for a specific `segment` and `target`. Forecast length is `num_periods` and is constrained to a bounded horizon. The server constructs a monthly date index and pairs each predicted value with `YYYY-MM` identifiers.

```
POST /predict
Content-Type: application/json
```

Request body:

```
{
  "segment": "<string>",
  "target": "<string>",
  "num_periods": <integer, default 12>,
  "start_date": "YYYY-MM-DD"   (optional)
}
```

Semantics and validation:

- `segment` and `target` are required and must belong to the allow-lists.
- `num_periods` must be an integer in a fixed interval (bounded to avoid unbounded recursion and latency).
- If `start_date` is omitted (or fails to parse), the server anchors the forecast at the first day of the next calendar month.
- Predictions are returned as non-negative integers (post-processed via rounding and clipping).

Response (200):

```
{
  "segment": "<string>",
  "target": "<string>",
  "num_periods": <integer>,
  "forecast_start": "YYYY-MM-DD",
  "forecast_end": "YYYY-MM-DD",
  "forecasts": [
    { "period": 1, "date": "YYYY-MM-DD", "month": "YYYY-MM", "forecast_value": <integer> }
  ],
  "model_info": { "run_id": "<string-or-null>", "loaded_at": "<string-or-null>" }
}
```

`POST /predict/batch` **(multi-series forecast with server-side netting).** This endpoint produces forecasts for all targets for a given segment in a single call and performs server-side business logic when both `net_sales` and `returns` are available. Specifically, it computes:

$$\texttt{total\_net\_forecast}_t := \max\{\texttt{net\_sales\_forecast}_t - \texttt{returns\_forecast}_t,\ 0\}.$$

This shifts arithmetic and consistency rules to the server, so clients only consume one canonical vector.

```
POST /predict/batch
Content-Type: application/json
```

Request body:

```
{
  "segment": "<string>",
  "num_periods": <integer, default 12>,
  "start_date": "YYYY-MM-DD"   (optional)
}
```

Response (200):

```
{
  "segment": "<string>",
  "num_periods": <integer>,
  "forecast_start": "YYYY-MM-DD",
  "forecast_end": "YYYY-MM-DD",
  "forecasts": [
    {
      "period": 1,
      "date": "YYYY-MM-DD",
      "month": "YYYY-MM",
      "net_sales_forecast": <integer>,
      "returns_forecast": <integer>,
      "total_net_forecast": <integer>
    }
  ],
  "errors": { "net_sales": "<string>", "returns": "<string>" }  (null if none)
}
```

Error behavior: If no target forecast can be produced (e.g., no models are loaded for the requested segment), the endpoint returns `HTTP 400` and includes per-target error details.

5.2.4 Operational configuration

The server reads environment variables to control runtime behavior:

- `AUTO_LOAD_MODELS`: whether to auto-load one run per segment-target pair on startup.
- `API_HOST`, `API_PORT`, `API_DEBUG`: host binding, port, and debug mode.

This design enables the same codebase to run locally, on a LAN, or behind a reverse proxy, without modifying the implementation.

6 Results and Interpretation

6.1 Forecast Visualization

The model demonstrates a strong ability to capture the seasonality inherent in the automotive sector. Figure 5 illustrates the 12-month forecast horizon across all three segments.

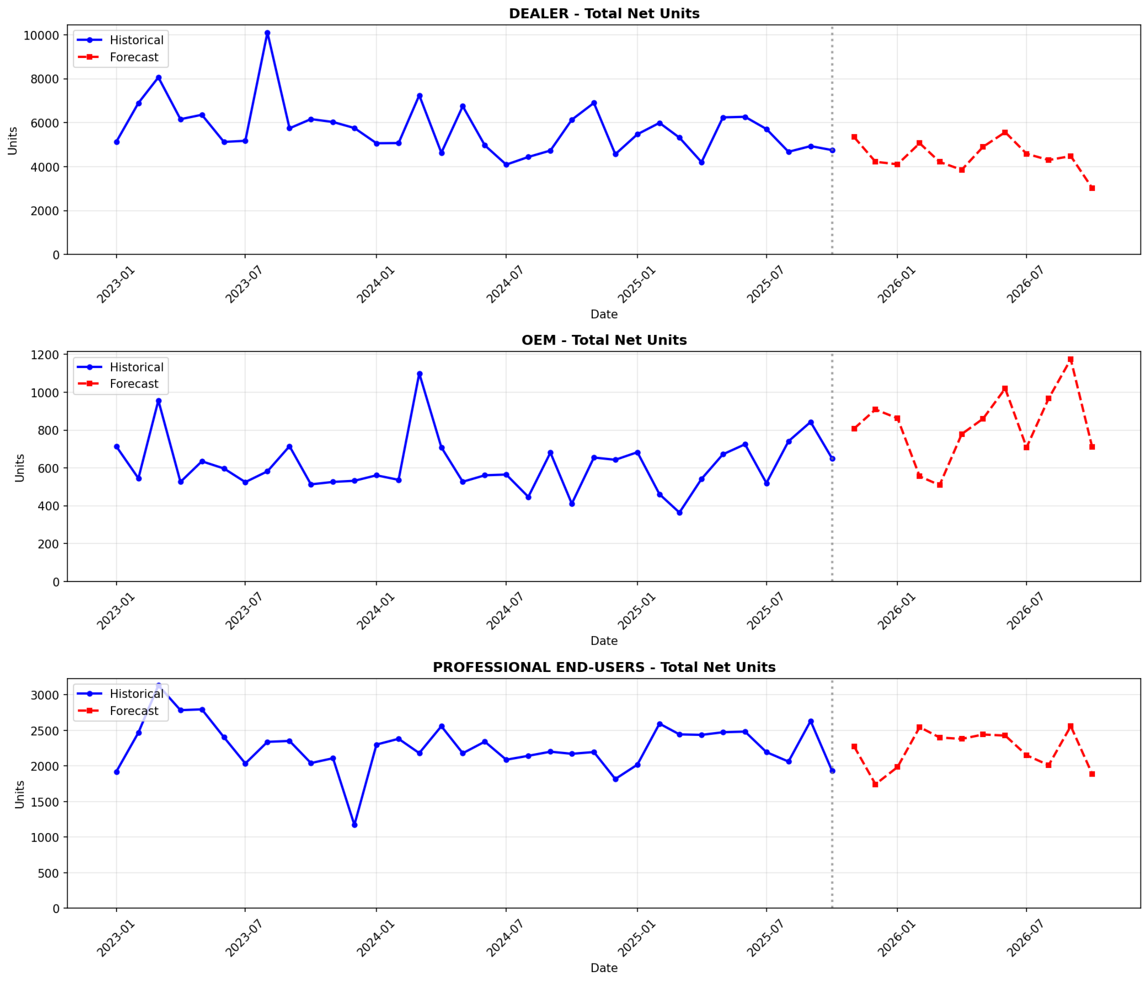


Figure 5: Global 12-Month Forecast for Net Sales across Segments. Note the capture of the end-of-year dip and Q2 recovery.

6.2 Performance Metrics

Table 1 summarizes the performance on the hold-out test set for all customer segments and targets. The Custom SARIMA model outperforms or remains competitive with the Seasonal Naive baseline across the board. Notably, the *Dealer* segment, which carries the highest volume and volatility, shows significant improvement in RMSE.

Segment	**Target**	**SARIMA RMSE**	**Baseline RMSE**	**Outcome**
Dealer	Net Sales	3672.19	3685.09	**Better**
Dealer	Returns	2470.63	2699.63	**Better**
OEM	Net Sales	291.86	143.26	Baseline Better
OEM	Returns	96.91	104.72	**Better**
Professional Users	Net Sales	430.48	449.39	**Better**
Professional Users	Returns	96.62	101.76	**Better**

Table 1: Comparative Accuracy Metrics (Test Set). The SARIMA model outperforms the baseline in 5 out of 6 targets.

7 Conclusion

This project delivered a robust forecasting platform for the automotive industry by implementing a Multiplicative SARIMA algorithm from scratch using Gradient Descent optimization. Model selection through Grid Search and Rolling-Origin Cross-Validation ensured that the chosen hyperparameters remain robust against temporal shifts. Integration with MLflow for experiment tracking and Flask for API deployment transforms this solution from an academic exercise into a production-ready asset. The ability to forecast Net Sales and Returns independently enables inventory optimization and more accurate fiscal margin projections.

Future iterations could extend this work in several directions. Incorporating exogenous variables through a SARIMAX formulation would allow the model to capture external demand drivers such as macroeconomic indicators or promotional events. Ensemble approaches combining SARIMA with machine learning models could further reduce forecast variance. Replacing Grid Search with Bayesian optimization would improve hyperparameter tuning efficiency. Finally, implementing prediction intervals and automated retraining pipelines would enhance both forecast interpretability and long-term model maintenance.

References

[1] Penn State University. (n.d.). *Lesson 1: Time Series Basics*. STAT 510: Applied Time Series Analysis.

[2] Penn State University. (n.d.). *Lesson 2: MA Models, Partial Autocorrelation, Notational Conventions*. STAT 510: Applied Time Series Analysis.

[3] Penn State University. (n.d.). *Lesson 3: Identifying and Estimating ARIMA models; Using ARIMA models to forecast future values*. STAT 510: Applied Time Series Analysis.

[4] Penn State University. (n.d.). *Lesson 4: Seasonal Models*. STAT 510: Applied Time Series Analysis.

[5] Penn State University. (n.d.). *Lesson 5: Smoothing and Decomposition Methods and More Practice with ARIMA Models*. STAT 510: Applied Time Series Analysis.