CV Prep

Project Analysis: Demand Forecasting for Cement Industry

## 🧩 1. Problem Statement

**Business Objective**:
Forecast cement demand at multiple levels — **dealer**, **district**, and **regional** — to optimize the **cement supply chain** and reduce overstocking, understocking, and delays.

**Your Framing**:
I aimed to enable proactive supply chain planning using **accurate time series forecasting models**.

## 2. Techniques Used

I combined **classical time series** and **machine learning models**:

- **Lasso & Ridge Regression**: Regularized linear models to prevent overfitting and handle multicollinearity.
- **Random Forest**: Captures nonlinearities and interactions between variables.
- **Ensemble Stacking**: Smartly combines predictions from multiple base models, enhancing generalization and reducing model variance.

Additional implied techniques:

- Feature engineering on temporal & external variables (likely holidays, price fluctuations, weather, etc.)
- Handling non-stationarity or seasonality in demand patterns (even if not explicitly said)

## Evaluation & Metrics

Used **MAPE (Mean Absolute Percentage Error)**, which is appropriate for demand forecasting.

Results (very realistic and solid):

- **Regional MAPE: 6.7%**
- **District MAPE: 19.75%**
- **Dealer MAPE: 27.90%**

Interpretation: The model is **more accurate at aggregate levels** (regional) and **noisier at the micro level** (dealer). This is normal in hierarchical time series forecasting.

### Impact

- **Supply Chain Responsiveness improved by 20%**: This quantifies business value.
- Helped to reduce holding costs, prevent stockouts, and plan logistics better.
- Forecasting at different levels allows **granular decision-making**, especially useful in a fragmented distribution system like cement.

### Why did you used ridge and lasso while both are regularization techniques?

I used both Ridge and Lasso because they serve slightly different purposes in regularization, and combining them allowed me to evaluate which regularization behavior worked better for different levels of granularity in the forecasting.

- **Ridge (L2)** helps when all features are useful and collinear, as it **shrinks coefficients** but doesn't eliminate them.

- **Lasso (L1)** helps with **feature selection**, as it can shrink some coefficients exactly to zero, making the model more interpretable.

In my project, Lasso was particularly useful at **regional level**, where fewer, stronger signals mattered. Ridge was helpful at the **dealer level**, where data was noisy and we wanted to preserve weaker signals without overfitting.

I also included both in the **ensemble stacking**, since their different behaviors gave diversity to the base models — improving final performance through model blending.

## ✅ Extended Technical Version (if they dig deeper):

### 🔹 Why both are useful:

- Ridge and Lasso **penalize weights differently**:
    - **Ridge (L2)** penalty: distributes shrinkage across all coefficients → keeps all features, good when many features contribute weakly.
    - 
    - **Lasso (L1)** penalty: forces some weights to zero → useful when only a few features are strongly predictive.

### 🔹 In forecasting:

- **Lasso** works well when temporal/external features are sparse or only a few are influential (e.g., maybe only price and past 1-week demand matter).
- **Ridge** works better when multicollinearity exists across lagged features (e.g., t-1, t-2, ..., t-12).

### 🔹 In ensemble:

Using both improves model diversity in stacking:

- Lasso gives **sparse**, interpretable signals.
- Ridge captures **distributed patterns** that Lasso might ignore.
  This diversity leads to **better final predictions** when blended.

---

# ✅ Q1: Why did you use ensemble stacking?

I used ensemble stacking to **combine the strengths of multiple models** and minimize their weaknesses.

- Some models like Ridge/Lasso are **stable and interpretable**, but may underperform in capturing non-linear patterns.
- Others like Random Forests handle **non-linearities and interactions** well but can be prone to overfitting or lack interpretability.

By stacking them — especially with a **meta-learner** (like a simple linear model or gradient boosting on top) — I was able to create a final model that:

- Outperformed individual models

- Generalized better across dealer, district, and region levels
- Reduced error (MAPE) consistently

The diversity of model types was key to ensemble success, and it also ensured **scalability and robustness** across granular demand levels.

---

## ✅ Q2: How did you tune hyperparameters?

For each model, I used a **time-aware validation strategy** like **rolling window or expanding window cross-validation**, which respects temporal order — instead of random K-Fold CV.

- For **Ridge/Lasso**, I tuned the alpha (regularization strength) using `GridSearchCV` with a **TimeSeriesSplit** strategy.
- For **Random Forest**, I tuned `n_estimators`, `max_depth`, `min_samples_split` etc., using RandomizedSearch or Optuna for efficiency.

During stacking, I:

- Validated each base model's performance on a hold-out set
- Then used their predictions as inputs to a **meta-model**, which was also cross-validated

The key was to avoid **data leakage across time** and ensure **hyperparameters generalized** well to unseen future data.

---

## ✅ Q3: Why did you use Random Forest and not XGBoost?

I started with Random Forest because:

- It's fast to train, works well **out-of-the-box**, and handles **noisy, tabular, and high-dimensional data** like dealer-level demand.

- It's less sensitive to hyperparameters compared to XGBoost.

XGBoost is more **accurate but complex** — and it can overfit if not tuned properly. Given the **scale and frequency** of the data, RF gave me **faster iteration cycles** and robust results.

That said, XGBoost was on my radar — but for this project, RF gave a better **trade-off between performance and simplicity**, especially when used as a base learner in ensemble stacking.

I would explore XGBoost or LightGBM in follow-up phases if model performance plateaued or more non-linear interactions needed modeling.

---

# Optional Add-On (if asked): Did you try neural nets like LSTM?

I considered it, but LSTM/DeepAR require:

- A **lot of historical data per time series** (which wasn't available at the dealer level)

- High computational cost and complex tuning

Since my goal was also **interpretability + speed**, traditional + ensemble models were more practical and scalable.

---

## Why Not used ARIMA/SARIMA?

This is another **great question** interviewers ask to test your **model selection reasoning**, especially for time series.

Here's a strong way to answer **why you didn't use ARIMA/SARIMA** in your cement demand forecasting project:

---

## ✅ Interview-Ready Answer: Why not ARIMA or SARIMA?

I considered ARIMA and SARIMA, but chose not to use them for several reasons tied to the **data structure**, **forecasting goals**, and **scalability**:

1. **Multiple Time Series**:
   We had hundreds (possibly thousands) of separate time series — for each **dealer**, **district**, and **region**.
   ARIMA/SARIMA are **univariate models** and must be trained **individually per series**, which is not scalable or practical at this scale.

2. **Need for External Features (Exogenous Variables)**:
   We wanted to incorporate **external factors** like promotions, prices, seasonality markers, holidays, and maybe even weather.
   Traditional ARIMA/SARIMA don't handle these well — or require SARIMAX extensions, which complicates the setup without much gain.

3. **Data Non-Stationarity & Irregularity**:
   Many dealer-level time series were **noisy, intermittent, and non-stationary**.
   ARIMA needs careful **differencing, trend and seasonality modeling**, which would've required **manual tuning** per series.
   Instead, machine learning models (Ridge, Lasso, RF) handled these irregularities more gracefully.

4. **Performance and Flexibility**:

- ARIMA works well for **short-term, stable series**, but struggled with long-range or highly volatile patterns.

- Ensemble models gave **better accuracy**, handled **non-linear patterns**, and let us train one **global model** across all time series — leading to better **scalability** and **consistency**.

So while ARIMA/SARIMA are great for small-scale, interpretable forecasting tasks, they were not suitable for our **large-scale, multi-level demand forecasting problem**.

---

## 🔁 Optional Add-On (if they dig deeper):

That said, I've worked with ARIMA in other contexts and understand its strengths. If we had fewer time series or needed interpretable forecasts for specific districts, I would consider ARIMA or SARIMAX for a benchmark or audit.

---