Advanced Programming 2025

Retail Store Inventory Forecasting ML System

Final Project Report

Gauthier Loyer
gauthier.loyer@unil.ch
Student ID: 20411666

January 29, 2026

Abstract

This project builds a retail demand forecasting system that predicts daily units sold and revenue and converts predictions into order recommendations with economic cost implications. Using the provided retail inventory dataset, the pipeline cleans and validates records, aggregates daily sales by store, product, category, and region, and engineers time, lag, moving-average, and categorical features. Multiple supervised models are trained (linear regression, random forest, and gradient boosting), and baselines are included for comparison. Evaluation uses MAE and RMSE, alongside finance-relevant cost estimates that quantify holding and stockout costs. The best-performing model in the latest run is a random forest, which yields the lowest RMSE and supports inventory decisions through recommendation tables and visual diagnostics. The system emphasizes time-aware splitting to reduce look-ahead bias and documents potential data leakage risks around inventory and ordering features. The main contribution is an end-to-end, reproducible workflow that connects forecasting accuracy to inventory cost decisions, supported by diagnostics and per-product performance analysis.

Keywords: demand forecasting, inventory optimization, time series, machine learning, retail analytics, cost-sensitive evaluation

Contents

1 Introduction

This project addresses retail inventory forecasting, a problem with direct financial impact through overstocking (holding costs) and stockouts (lost revenue). The repository implements an end-to-end workflow that predicts daily units sold (primary target) and revenue (secondary target) and converts forecasts into product-level reorder recommendations.

1.1 Background and Motivation

This project is motivated by practical consulting use cases where machine learning models support operational decision-making in retail. Retail stores regularly face ordering and inventory planning decisions; the goal is to transform historical sales data into actionable insights by predicting how much of each product will sell and converting those predictions into reorder suggestions (e.g., "order X units if predicted sales exceed current stock").

1.2 Problem Statement

Given daily sales records with contextual features (e.g., Weather Condition, Seasonality, Region), predict future daily demand (units sold) per product and store context, and produce reorder quantities and urgency indicators.

Research question: How accurately can machine learning models forecast daily retail demand using historical sales and contextual features, and how can these forecasts be translated into cost-aware inventory ordering decisions?

1.3 Objectives and Goals

- Build a reproducible pipeline from data loading to recommendations.
- Compare multiple models and simple baselines using time-aware validation.
- Report both ML accuracy and finance-relevant cost impact.
- Provide diagnostics and per-product performance analysis.

1.4 Report Organization

Section 2 covers the research question and related work, Section 3 describes data, features, and modeling choices, Section 4 reports the experimental setup, metrics, and figures, Section 5 interprets results and limitations, Section 6 concludes with future work, and Section 7 summarizes codebase and reproducibility.

2 Research Question and Related Work

This section situates the research question within existing methodological approaches to retail demand forecasting and motivates the modeling choices used in this project. The repository does not include external papers/books to cite, so what follows summarizes relevant prior *approach families* using only what is implemented in the project (and flags missing external citations explicitly). Tree-based ensembles are suitable here because they capture non-linear relationships and interactions between features.

2.1 Previous Approaches to Similar Problems

The codebase includes simple forecasting baselines (last value, mean, seasonal naive) and supervised ML models trained on engineered temporal features (lags and moving averages). These represent standard benchmark families for demand forecasting: naive/statistical heuristics versus feature-based supervised learning.

2.2 Relevant Algorithms or Methodologies

Implemented models include linear regression (interpretable baseline), random forest regression, and gradient boosting regression. The validation strategy uses a chronological split and time-series cross-validation (TimeSeriesSplit) to limit look-ahead bias.

2.3 Datasets Used in Related Studies

The repo contains a single local dataset archive (`data/raw/Retail_store_inventory_forecasting_dataset.zip`). The external provenance for this dataset is the Kaggle dataset page provided during project finalization: `https://www.kaggle.com/datasets/alexhuitron/supermarket-sales?resource=download`. The dataset is downloaded and stored locally in the repository as a ZIP archive to support reproducibility.

3 Methodology

3.1 Data Description

Raw and processed datasets are stored locally to ensure reproducibility.[1]

3.1.1 Source and Collection Method

The dataset is shipped as a local ZIP file inside the repository. The external provenance is the Kaggle dataset page: `https://www.kaggle.com/datasets/alexhuitron/supermarket-sales?resource=download`. The ZIP is used as the local, versioned input artifact for the pipeline.

3.1.2 Size and Characteristics

The processed table has 73,100 rows and 68 columns with a daily date field spanning 2022-01-01 through 2024-01-01. Observations are keyed by date and retail dimensions (Store ID, Product ID, Category, Region).

3.1.3 Features/Variables

Key predictive features include Weather Condition, Seasonality, and Region, plus Store ID, Product ID, and Category. Numerical features include Price, Inventory Level, Units Ordered, Discount, and Competitor Pricing. Time-derived features include day-of-week, month, quarter, weekend flag, and engineered lag and moving-average features. The primary target is Units Sold (standardized as `quantity`); revenue is derived as Price $\times$ Units Sold and stored as `revenue`.

[1]Raw: `data/raw/Retail_store_inventory_forecasting_dataset.zip`; processed: `data/processed/processed_data.csv`. The processed dataset contains 73,100 rows and 68 columns, with dates from 2022-01-01 to 2024-01-01.

3.1.4 Data Quality Issues

The pipeline performs schema validation, missing value handling, duplicate removal, negative value correction, and Product ID–Category consistency checks. A precomputed `Demand Forecast` column is present in the dataset and is explicitly excluded from features to reduce target leakage risk. Potential leakage risks remain for `Inventory Level` and `Units Ordered` if their timestamps are not aligned with decision time (discussed further in Section 5).

3.2 Modeling and Evaluation Approach

Linear Regression, Random Forest Regressor, and Gradient Boosting Regressor are trained for demand forecasting. The project also evaluates naive baselines (last value, mean, seasonal naive) for context. All models are classical ML estimators (no neural networks); the tree ensembles capture non-linearities while linear regression provides an interpretable baseline. Raw data are loaded from ZIP/CSV, parsed into a datetime index, cleaned/validated, aggregated to daily sales by dimensions, filtered for minimum history, enriched with time features, lag features, moving averages, and one-hot encoded categorical features. A time-aware split is applied to avoid mixing future observations into training. The forecasting task predicts units sold for day t using features available up to day $t-1$. Model performance is evaluated using MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error). The pipeline also computes an economic cost proxy (holding vs. stockout cost) to interpret forecasting error in financial terms. The cost metric is a simplified proxy intended to compare models under consistent assumptions rather than represent a full accounting-grade inventory cost model. Holding cost is set to \$0.1 per unit per day and stockout cost to \$10 per unit; costs are asymmetric. Total cost sums these penalties across the test period.

3.3 Implementation

Key implementation decisions include modularizing the pipeline into data loading, modeling, and evaluation components to ensure reproducibility and clarity. Particular attention was given to time-aware splitting to prevent look-ahead bias and to feature exclusion rules to mitigate data leakage risks.

3.3.1 Programming Languages and Libraries

Python 3.9 is used with pandas/numpy for data handling, scikit-learn for modeling, and matplotlib/seaborn for visualization. Models and scalers are persisted with joblib.

3.3.2 System Architecture

The primary entry point `main.py` orchestrates data loading `src/data_loader.py`, model training and cross-validation `src/models.py`, and evaluation/visualization/recommendations `src/evaluation.py`. Configuration lives in `src/config.py`.

3.3.3 Key Code Components

(i) preprocessing and feature engineering; (ii) model training (including baselines); (iii) evaluation metrics (MAE/RMSE) and cost computation; (iv) recommendation generation and final reporting; (v) diagnostic plots and per-product analysis.

The following function illustrates the metric computation used across evaluations. It uses `mean_absolute_error` and `mean_squared_error` from `sklearn.metrics`, and `numpy` (`np`) and `pandas`.

```python
def calculate_metrics(y_true: pd.Series, y_pred: pd.Series) -> Dict[str, float]:
    """Calculate MAE and RMSE metrics."""
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    return {"MAE": mae, "RMSE": rmse}
```

Listing 1: Metric computation with MAE and RMSE

4 Results

4.1 Experimental Setup

- **Hardware specifications**: MacBook Air (Model Identifier: Mac16,13), Apple M4 (10 cores: 4 performance + 6 efficiency), 16 GB RAM, macOS 15.7.3 (Build 24G419).
- **Software versions**: Python 3.9.6 and dependencies snapshot saved at `report/dependencies_snapshot.txt`.
- **Hyperparameters**: Key configuration in `src/config.py` includes test size 0.2, minimum history 30 days, lag windows (1, 7, 30), moving average windows (7, 30), random state 42, and 5-fold time-series cross-validation. Hyperparameter tuning and XGBoost are disabled by default.

4.2 Performance Evaluation

Table 1 reports model metrics from `results/metrics/model_comparison.csv`. Random forest achieves the lowest RMSE in the latest run and is used for recommendations. The random forest outperforms the baseline by capturing non-linear effects and interactions between temporal features, pricing, and categorical variables. Linear regression underfits these relationships, while gradient boosting did not provide additional gains under the default hyperparameter configuration.

Table 1: Model Performance Metrics (Quantity Forecasting)

Model	**MAE**	**RMSE**
Baseline (mean)	88.80	108.25
Random forest	61.87	79.89

Only the top baseline and best model are shown; the full comparison is in `results/metrics/model_comparison.`

Cost-sensitive evaluation computes total economic cost combining holding and stockout costs; the best model run reports a total cost of approximately \$4.54M. (Recall that the cost metric is a simplified proxy for model comparison, not an accounting-grade estimate.)

Additionally, the repository runs a revenue forecasting track (enabled in configuration), producing `results/metrics/revenue_model_comparison.csv`. In the latest run, `random_forest_revenue` shows the lowest RMSE among the revenue models.

4.3 Visualizations

Figure 1 shows prediction vs. actual for the best model, Figure 2 shows residual diagnostics, and Figure 3 summarizes per-product error. Figure 4 shows historical sales trends.

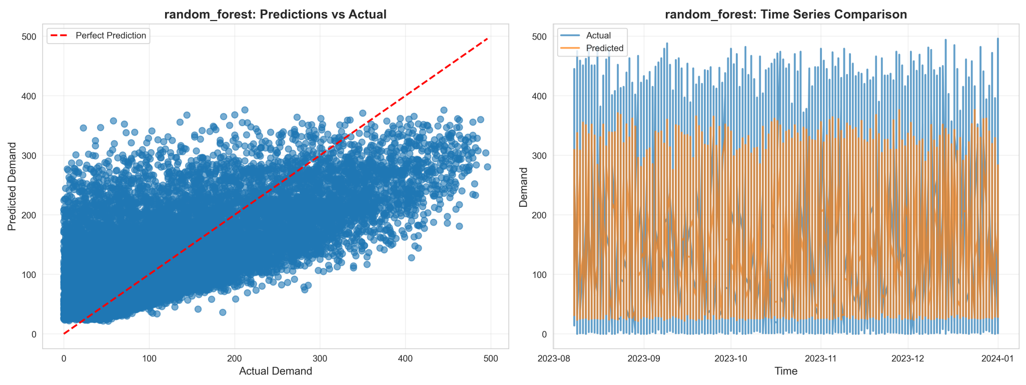

Figure 1: Random forest predictions vs. actuals

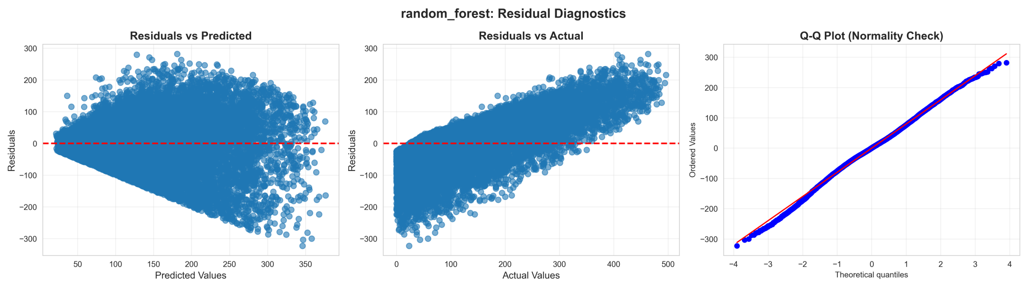

Figure 2: Random forest residual diagnostics

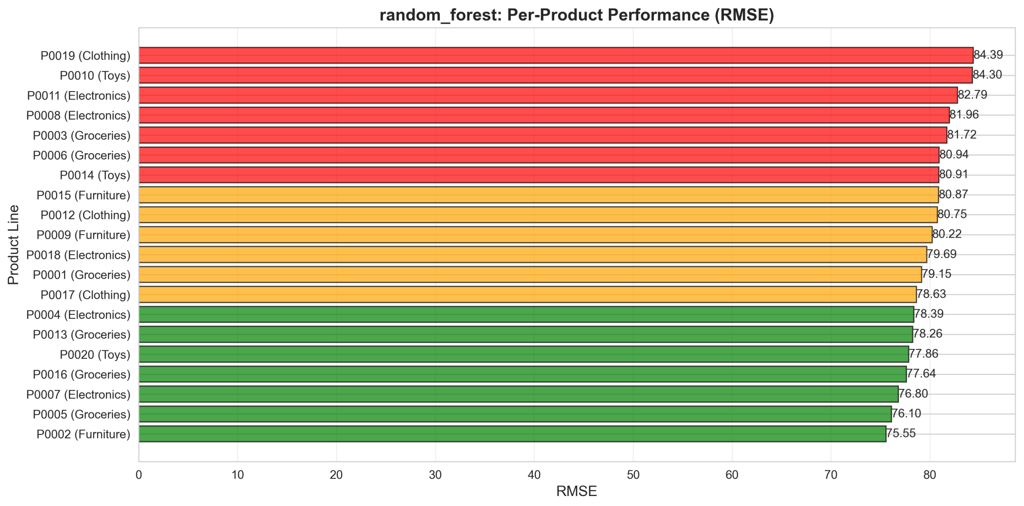

Figure 3: Per-product performance (RMSE)

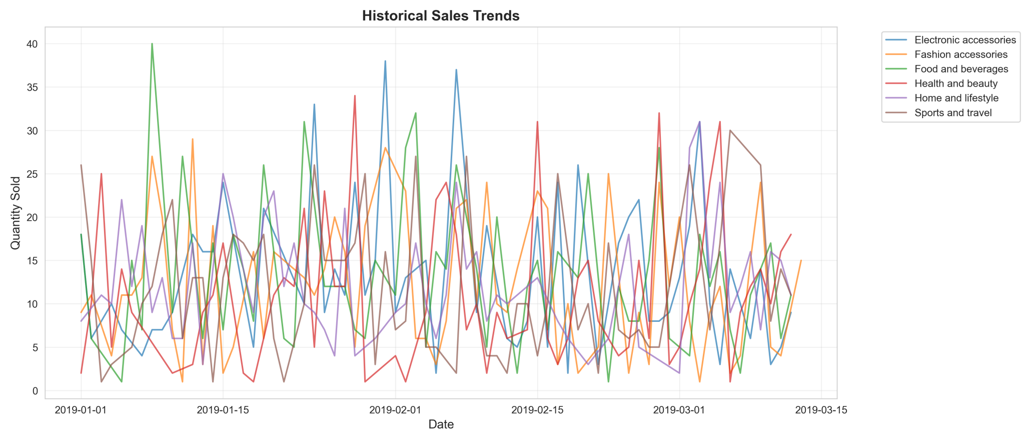

Figure 4: Sales trends over time

5 Discussion

The random forest model provides the best quantitative accuracy across MAE and RMSE, and supports an economically grounded recommendation system with explicit cost calculations. The evaluation pipeline includes time-aware splitting and cross-validation, which are appropriate for temporal retail data. A key challenge is potential leakage for features like Inventory Level and Units Ordered; the code documents these risks and includes validation checks, but the report cannot confirm the data collection semantics without external documentation. In a production setting, this risk could be mitigated by shifting inventory-related features by one decision period or restricting features to information available strictly at order time. Another limitation is the absence of external baselines such as ARIMA or exponential smoothing in the repo, limiting direct comparison to classical time-series methods. The diagnostics show variability across products, indicating heterogeneous demand patterns and opportunities for targeted improvements.

6 Conclusion and Future Work

6.1 Summary

This project delivers a reproducible retail demand forecasting pipeline that links prediction accuracy to inventory cost decisions. It integrates data validation, feature engineering, multiple ML models, and cost-aware evaluation, producing actionable order recommendations. The best-performing model is a random forest with the lowest RMSE in the latest run.

6.2 Future Directions

- Add classical time-series baselines (e.g., ARIMA or exponential smoothing) for stronger benchmarking.
- Expand diagnostics to quantify uncertainty and stability across temporal regimes.
- Record and report hardware and runtime metrics for full experimental transparency.
- Incorporate external business constraints (lead times, service levels) into recommendations.

7 Codebase and Reproducibility

The project is fully reproducible from the provided codebase. Dependencies are listed in `requirements.txt`, and the full pipeline can be executed using `python main.py`, which generates trained models, evaluation metrics, figures, and inventory recommendations. Detailed repository structure and setup steps are given in Appendix B.

References

1. Project README (local repository). *README.md*.
2. Project Proposal (local repository). *PROPOSAL.md*.
3. Project Review (local repository). *MASTERS_REVIEW.md*.
4. Retail Store Inventory Forecasting Dataset (local file). *data/raw/Retail_store_inventory_forecasting_dat*
5. Kaggle dataset provenance (provided during report finalization). Available at: `https://www.kaggle.com/datasets/alexhuitron/supermarket-sales?resource=download`.

A Additional Figures

Additional figures are stored in `results/figures/` and include feature importance, residual diagnostics, and revenue forecasting plots.

B Code Repository

GitHub Repository: `https://github.com/gauthierloyerg3-sketch/data_proj`
Repository structure and reproduction steps:

- **Structure**: `src/` for code, `data/` for datasets, `results/` for metrics and figures, `models/` for trained artifacts.
- **Installation**: `pip install -r requirements.txt` (or conda environment per `environment.yml`).
- **Reproduction**: Run `python main.py` to generate metrics, figures, and recommendations.

C Use of AI Tools

AI tools were used as supportive assistants during the development of this project. ChatGPT was used to help refine explanations, improve clarity of written sections, and assist with debugging isolated code issues. Cursor AI was used to support coding tasks and refactoring. All modeling choices, implementation decisions, experimental design, and result interpretation were made and validated by me.