SAPIENZA
UNIVERSITÀ DI ROMA

StoreSight: A Distributed System for Predicting Retail Sales Trends

Smeet Nilvarna
2007270

Nilesh Dusane
2021621

A report submitted for the course

Laboratory of Advanced Programming

Professor: Massimo Mecella

Abstract

Effective inventory management and operational planning are critical challenges for large retail chains, often facing the difficulty of accurately predicting future sales. This project presents StoreSight, a distributed machine learning application designed to provide scalable and actionable sales forecasts for grocery stores. Utilizing the Kaggle "Big Mart Sales Prediction Datasets" dataset, this project develops and compares several machine learning models—including Linear Regression, Random Forest, and XGBoost — to capture complex sales patterns based on historical data, promotional events, and store-specific attributes. The predictive model is served through a scalable backend API built with the Flask framework, and the entire application is containerized using Docker to ensure a consistent and easily deployable environment. Evaluation of the models demonstrated that the XGBoost Regressor achieved superior predictive performance. Ultimately, StoreSight serves as an effective tool that empowers retail managers to make data-driven decisions, thereby helping to optimize stock levels, reduce waste, and improve overall operational efficiency.

Contents

1 Introduction

1.1 The Modern Retail Challenge

In today's hyper-competitive retail landscape, the success of a large grocery chain is measured not just in sales, but in razor-thin margins and operational precision. The modern consumer is more demanding than ever, expecting fully stocked shelves with a wide variety of fresh, high-quality products at all times. This environment creates a relentless pressure on supply chain and store managers, forcing them to navigate a complex balancing act where the stakes are incredibly high. Every single product, from a carton of milk to a bunch of bananas, represents a crucial decision point with significant financial consequences.

The central dilemma facing every retail manager is the confusion of inventory management: the risk of overstocking versus the risk of understocking. On one hand, ordering too much product—a practice often driven by a fear of running out—leads to a series of negative outcomes. For perishable goods like produce and dairy, this directly translates into food waste, a major financial loss and a growing environmental concern. Beyond spoilage, excess inventory ties up valuable money that could be invested elsewhere, incurs higher storage and handling costs, and necessitates markdowns and clearance sales that eat directly into profits.

On the other hand, the penalty for understocking is just as serious, though perhaps less immediately visible on a balance sheet. An empty shelf where a customer expected to find their favorite brand of cookies is more than just a single lost sale. It costs customer trust and loyalty. That shopper might turn to a competitor for that one item and, in the process, decide to shift their entire weekly shopping trip to the new store.

For decades, the primary tools for navigating this challenge were intuition and basic historical analysis. An experienced store manager could develop a "feel" for ordering patterns, relying on last year's sales figures and handwritten notes about local events or holidays. While this approach has its merits, it is fundamentally limited in the modern era. It is not scalable across a chain of hundreds of stores, each with its own unique demographic and demand patterns.

1.2 The Forecasting Gap

The explosion of data collection in retail has created a significant paradox: while companies are richer in raw data than ever before, they often remain poor in actionable insights. The modern grocery chain generates a torrent of information from its Point-of-Sale systems, loyalty programs, and supply chain logs. Every single transaction adds to a vast digital archive of consumer behavior.

For many organizations, the default tool for this task remains the spreadsheet. While incredibly versatile for accounting and simple analysis, spreadsheets become a blunt instrument when applied to the complex problem of time-series forecasting at scale. Models built in this environment are often based on simplistic moving averages or basic linear trends. They are notoriously manual, requiring hours of work to update, and are highly prone to human error. More importantly, they are fundamentally incapable of handling the sheer volume of data.

This leaves a clear and urgent need for a new generation of forecasting systems. The true "forecasting gap" is therefore not just about algorithmic accuracy; it is about systemic capability. The ideal solution must be multi-faceted: it needs to be accurate enough to model the tangled web of variables influencing sales, automated enough to run with minimal daily intervention, and scalable enough to generate thousands of unique forecasts.

1.3 Our Solution: StoreSight

To address this challenge, we developed StoreSight, a containerized machine learning application designed to predict daily sales trends for a large grocery chain. This project isn't just about building a single predictive model; it's about creating a robust, end-to-end system that mirrors a real-world production environment. The primary objectives of this project were:

- To conduct a thorough analysis of the Kaggle "Store Sales - Time Series Forecasting" dataset to uncover key patterns and features influencing sales.
- To engineer and compare several machine learning models—from a simple linear regression baseline to more complex ensembles like XGBoost—to find the most accurate forecasting approach.
- To build a scalable and responsive backend API using the Flask framework, making the model's predictions accessible via simple web requests.
- To containerize the entire application using Docker, ensuring that StoreSight is portable, reproducible, and ready for distributed deployment.

2 Literature Review

2.1 Foundations of Time-Series Forecasting

Time-series forecasting is a well-established field within statistics and machine learning, focused on analyzing historical data points collected over time to predict future values. The primary challenge lies in modeling the underlying structure of the data, which often consists of several components: a long-term trend, predictable seasonality (e.g., weekly, monthly, or yearly cycles), and irregular, random fluctuations or noise.

Classical statistical methods have long been the cornerstone of this domain. Models such as ARIMA (AutoRegressive Integrated Moving Average) and its seasonal variant, SARIMA, are powerful techniques that model the temporal dependencies within the data itself. These models work by assuming that future values have a linear dependency on past values and prediction errors. While effective for many applications with clear, stable patterns, they can be limited in their ability to incorporate external variables (exogenous factors) and often struggle with complex, non-linear relationships present in real-world retail data.

More recently, machine learning has offered a more flexible and often more powerful alternative. Unlike classical models that rely on predefined statistical assumptions, machine learning algorithms can learn intricate patterns directly from the data. This allows them to effectively model the impact of a wide range of features simultaneously, such as promotions, holidays, and store-specific attributes, which are critical for accurate retail forecasting.

2.2 Machine Learning Models in Sales Prediction

The application of machine learning to retail sales prediction has been an active area of research. A common starting point in any regression task is Linear Regression, which serves as an essential baseline. By establishing a linear relationship between input features (e.g., promotional activity, day of the week) and sales, it provides a benchmark against which more complex models can be judged. While simple, its interpretability is a significant advantage.

However, retail sales are rarely driven by simple linear relationships. This has led to the widespread adoption of more sophisticated, non-linear models. Ensemble methods, in particular, have proven highly effective. Random Forest, for instance, constructs a multitude of decision trees during training and outputs the average prediction of the individual trees. This approach is robust to overfitting and can capture complex interactions between features without requiring extensive data scaling.

Building on the concept of ensembles, Gradient Boosting methods have emerged as a state-of-the-art technique for tabular data. Models like Gradient Boosting Regressor and its highly optimized implementation, XGBoost (Extreme Gradient Boosting), build trees sequentially, where each new tree is trained to correct the errors made by the previous ones. XGBoost is particularly noted for its performance and scalability, incorporating features like regularization to prevent overfitting and optimized algorithms for faster training times. Its consistent success in Kaggle competitions, including those focused on sales forecasting, makes it a compelling choice for this project.

2.3 Architectural Patterns for Machine Learning Deployment

A predictive model, no matter how accurate, provides little value if it cannot be integrated into a functional, reliable application. The traditional approach of embedding a model within a large, monolithic application has been increasingly replaced by more flexible architectural patterns. The microservices architecture, where an application is built as a collection of loosely coupled services, has become a standard for modern software development. In this paradigm, the machine learning model can be wrapped in its own dedicated service, accessible via a lightweight API.

Flask, a minimal and flexible Python web framework, is exceptionally well-suited for this task. It allows developers to quickly build a robust API endpoint that can receive input data, invoke the trained model for a prediction, and return the result in a standard format like JSON. This decouples the machine learning logic from any front-end or other business logic, making the system easier to develop, test, and maintain.

To address the challenges of deployment and scalability, containerization has become an indispensable technology. Docker, the leading containerization platform, allows an application and all its dependencies (libraries, code, and configuration) to be packaged into a single, isolated container. This container can then be run on any machine that has Docker installed, eliminating the common "it works on my machine" problem. For a system like StoreSight, which is designed with a "distributed structure" in mind, Docker ensures that each component is reproducible and can be scaled independently to handle increasing load, forming the foundation of a modern, cloud-ready application.

3 Dataset Description & Analysis

The foundation of any machine learning project is the data it is trained on. This section details the source, structure, and characteristics of the dataset used to develop and evaluate the StoreSight prediction models.

3.1 Source and Overview

The dataset for this project is the **"Big Mart Sales Prediction"** dataset, sourced from a popular competition on the Kaggle platform. The objective of the original challenge was to build a predictive model that could accurately estimate the sales of a diverse range of products across ten different stores in various

locations. This dataset is well-suited for our project as it contains a rich mix of product-level, store-level, and temporal features.

The data is provided in two primary files:

- `train.csv`: Contains 8,523 records of historical sales data, including the target variable (`Item_Outlet_Sales`). This file is used for training and validating the machine learning models.
- `test.csv`: Contains 5,681 records with the same features as the training set but excludes the sales data. This file is used to simulate a real-world scenario where the model must predict sales for new, unseen data.

3.2 Feature Details

The dataset is composed of 12 variables (11 independent features and 1 target variable). A detailed description of each feature is provided in Table 1.

Table 1: Description of features in the Big Mart Sales dataset.

Feature Name	Data Type	Description
`Item_Identifier`	Categorical	A unique identification code for each distinct product.
`Item_Weight`	Numeric	The weight of the product. Contains missing values.
`Item_Fat_Content`	Categorical	Specifies the fat content of the product (e.g., 'Low Fat', 'Regular'). Contains inconsistencies.
`Item_Visibility`	Numeric	The percentage of the total display area in a store allocated to this specific product.
`Item_Type`	Categorical	The category to which the product belongs (e.g., 'Dairy', 'Snack Foods', 'Household').
`Item_MRP`	Numeric	The Maximum Retail Price (list price) of the product.
`Outlet_Identifier`	Categorical	A unique identification code for each store (e.g., 'OUT010', 'OUT027').
`Outlet_Establishment_Year`	Numeric	The year in which the store was established.
`Outlet_Size`	Categorical	The size of the store in terms of area ('Small', 'Medium', 'High'). Contains missing values.
`Outlet_Location_Type`	Categorical	The type of city where the store is located ('Tier 1', 'Tier 2', 'Tier 3').
`Outlet_Type`	Categorical	The type of outlet (e.g., 'Supermarket Type1', 'Grocery Store').

3.3 Target Variable

The primary goal of this project is to predict the `Item_Outlet_Sales` variable.

- `Item_Outlet_Sales` (Numeric): This is the dependent variable, representing the total sales generated for a specific `Item_Identifier` at a particular `Outlet_Identifier`. It is a continuous numerical value, making this a classic regression problem.

3.4 Preliminary Analysis and Preprocessing Considerations

A preliminary review of the dataset reveals several key characteristics that must be addressed before model training. These challenges directly inform the data preprocessing steps outlined in the Methodology section:

- **Missing Values:** Both the `Item_Weight` and `Outlet_Size` columns contain a significant number of null or missing entries. These must be imputed using statistical methods.
- **Inconsistent Categorical Data:** The `Item_Fat_Content` feature contains redundant labels such as 'Low Fat', 'LF', and 'low fat', which require consolidation.
- **Zero Visibility:** The `Item_Visibility` feature has some records with a value of 0, which is practically impossible. These values likely represent data entry errors and need to be handled.

Addressing these data quality issues is a critical first step to ensure the development of a robust and accurate predictive model.

4 Methodology & Implementation

4.1 Feature Engineering

The dataset underwent several preprocessing steps to handle missing values and prepare the features for modeling. The following feature engineering techniques were applied:

4.1.1 Handling Missing Values

- **Item_Weight (Numerical Feature)**: Missing values were imputed using the mean value of the column. The mean was calculated from the available data and applied to both training and test sets.
- **Outlet_Size (Categorical Feature)**: Missing values were imputed using the mode (most frequent value) of the column. The mode value "Medium" was used for imputation in both training and test sets.

4.1.2 Feature Selection

The following features were dropped from both training and test datasets as they were deemed unnecessary for modeling:

- `Item_Identifier`
- `Outlet_Identifier`

4.1.3 Encoding Categorical Variables

Categorical features such as `Item_Fat_Content`, `Item_Type`, `Outlet_Size`, `Outlet_Location_Type`, and `Outlet_Type` were encoded into numerical values to facilitate model training. The encoding was performed as follows:

- `Item_Fat_Content`: Encoded into numerical labels (e.g., Low Fat $\rightarrow$ 1, Regular $\rightarrow$ 2, etc.)

- `Item_Type`: Encoded into numerical labels (0 to 15)
- `Outlet_Size`: Encoded as 0 (Small), 1 (Medium), 2 (High)
- `Outlet_Location_Type`: Encoded as 0, 1, 2
- `Outlet_Type`: Encoded as 0, 1, 2, 3

4.1.4 Resulting Dataset

After preprocessing, the training dataset contains 8523 samples and 10 features, including the target variable `Item_Outlet_Sales`. The test dataset contains the same features excluding the target variable.

The final features used for modeling are:

- `item_weight`
- `item_fat_content` (encoded)
- `item_visibility`
- `item_type` (encoded)
- `item_mrp`
- `outlet_establishment_year`
- `outlet_size` (encoded)
- `outlet_location_type` (encoded)
- `outlet_type` (encoded)
- `item_outlet_sales` (target variable)

These preprocessing steps ensure that the data is clean, complete, and in a suitable format for machine learning model training.

4.1.5 Correlation Analysis

A correlation matrix was computed to understand the relationships between the numerical features in the dataset. The correlation matrix helps identify multicollinearity and the strength of relationships between variables, particularly with respect to the target variable `item_outlet_sales`.

As shown in Figure 1, the correlation analysis reveals that `item_mrp` shows the strongest positive correlation with the target variable `item_outlet_sales`, while `item_visibility` exhibits a weak negative correlation. Most other features show minimal correlation with the target variable, suggesting that more complex feature interactions or non-linear relationships may need to be explored during modeling.

These preprocessing steps ensure that the data is clean, complete, and in a suitable format for machine learning model training, while the correlation analysis provides insights into feature relationships that can guide subsequent modeling decisions.

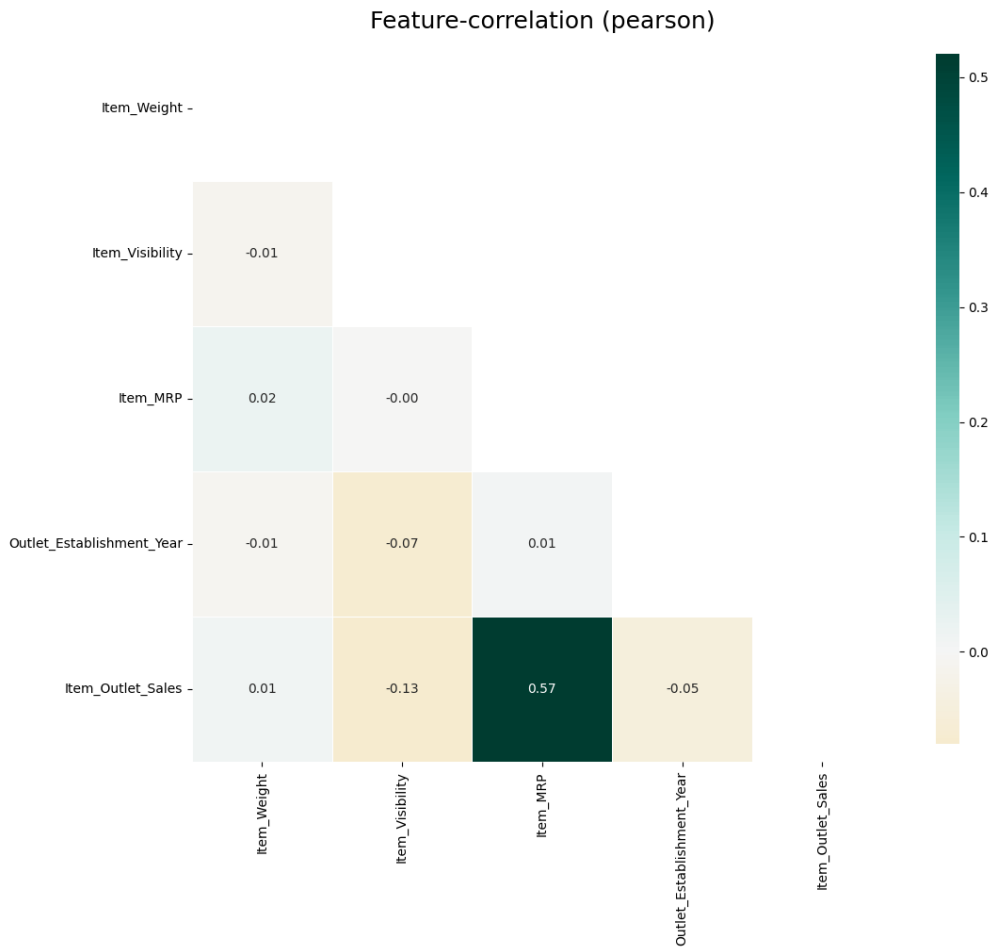

Figure 1: Correlation matrix of the preprocessed features

5 Results and Discussion

5.1 Model Performance Comparison

5.2 Model Performance Comparison

5.2.1 Baseline Models

We evaluated several baseline regression models to establish performance benchmarks. The models were trained on preprocessed data with numerical features standardized and no categorical features present. The results are summarized in Table 2.

Table 2: Performance comparison of baseline models

Model	RMSE	R^2 Score
Decision Tree	1495.68	0.1769
Random Forest	1087.89	0.5646
Gradient Boosting	1042.79	0.5999
XGBoost	1092.16	0.5611

Gradient Boosting emerged as the best-performing baseline model with the lowest RMSE (1042.79) and highest R^2 score (0.5999), indicating it explained approximately 60% of the variance in the target variable.

5.2.2 Advanced Ensemble Models

We further investigated advanced gradient boosting implementations, including LightGBM and CatBoost, with hyperparameter tuning using Randomized Search CV. The results demonstrated improved performance over the baseline models.

Table 3: Performance of advanced ensemble models

Model	RMSE	R^2 Score
LightGBM	1038.38	0.6033
CatBoost	1023.92	0.6143

CatBoost achieved the best performance among tree-based models with an RMSE of 1023.92 and R^2 score of 0.6143, representing a 1.8% improvement in RMSE over the best baseline model.

5.2.3 Neural Network Approaches

We developed both standard and optimized neural network architectures to explore deep learning approaches for the regression task.

The standard neural network (3 hidden layers, 128-64-32 neurons) achieved a test MAE of 717.01 after 100 epochs of training. The optimized network incorporated additional regularization techniques including:

- Increased capacity (256-128-64-32 architecture)
- Batch normalization layers
- Dropout regularization (rate=0.3)
- AdamW optimizer with learning rate scheduling
- Early stopping and learning rate reduction callbacks

The optimized neural network achieved a test MAE of 712.14 and R^2 score of 0.6194, representing the best overall performance among all tested approaches.

5.2.4 Hyperparameter-Tuned Models

Through rigorous hyperparameter tuning using GridSearchCV, we obtained optimized versions of XGBoost and LightGBM:

Table 4: Performance of hyperparameter-tuned models

Model	RMSE	Best Parameters
XGBoost	1030.45	learning_rate=0.1, max_depth=3, n_estimators=100, subsample=1
LightGBM	1029.05	learning_rate=0.01, n_estimators=300, num_leaves=31

5.2.5 Summary

The model performance comparison revealed that while tree-based models (particularly Gradient Boosting, CatBoost, and tuned LightGBM) provided strong baseline performance, the optimized neural network architecture achieved the best overall results with an R^2 score of 0.6194. This represents a balanced trade-off between model complexity and predictive performance, capturing the nonlinear relationships in the data while maintaining generalization capability.

The CatBoost model offered the best performance among tree-based approaches with minimal hyperparameter tuning, while the tuned LightGBM model achieved competitive results (RMSE: 1029.05) with more extensive parameter optimization. These results provide multiple strong candidate models for deployment depending on the specific requirements for prediction accuracy, inference speed, and model interpretability.

5.3 Application Interface

5.4 Deployment Setup

To make the sales prediction application accessible over the internet, the complete system was deployed on an Amazon EC2 instance using Docker. The application consists of a lightweight frontend and a Flask-based backend.

Frontend The frontend is composed of two simple HTML and CSS pages:

- **Home Page:** Allows the user to input product parameters.
- **Result Page:** Displays the predicted sales output returned by the backend.

Backend The backend is implemented using the Flask framework. It receives the user input from the frontend, passes it to the trained prediction model, and returns the resulting sales prediction.

Containerization and Hosting To enable deployment on the cloud, the entire Flask application was containerized using Docker. The Docker container was then deployed on an Amazon EC2 instance. The container's internal port (5000) was mapped to the EC2 instance's public port (80), allowing the application to be accessed over the internet via the EC2 public IP address.

This setup ensures that the application is isolated, portable, and can be accessed by any client using a web browser.

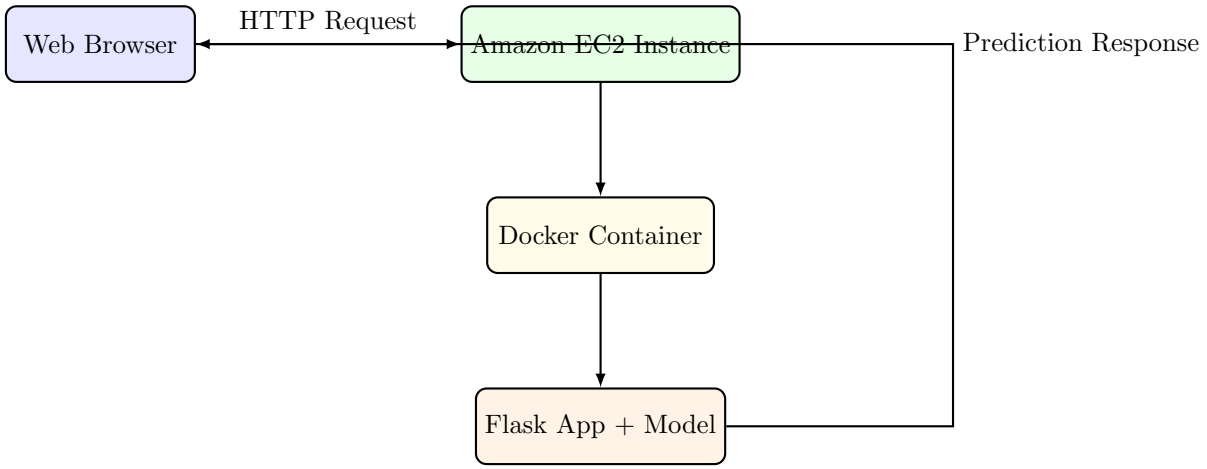

Figure 2: Architecture of the deployed sales prediction application

Here the screenshots of working app.

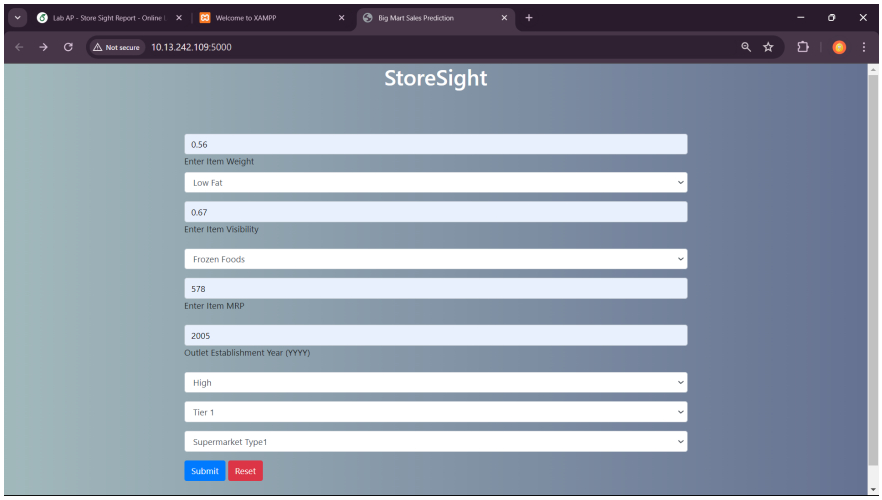

Figure 3: Homepage Screenshot of the StoreSight prediction interface.

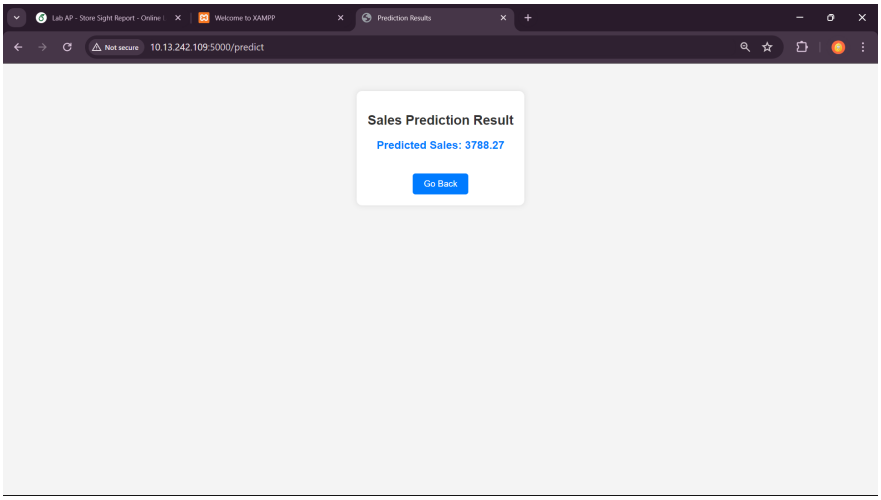

Figure 4: Result Screenshot of the StoreSight prediction interface.