**WESTERN MICHIGAN UNIVERSITY**

****

**SPRING 2025**

**CS 5821 – MACHINE LEARNING**

**INVENTORY DEMAND FORECASTING USING MACHINE LEARNING.**

**Abstract:**

In the retail and supply chain landscape, inventory demand forecasting is essential for optimizing stock levels, reducing waste, and improving customer satisfaction. This project aims to build an intelligent forecasting system using machine learning to predict weekly product demand at individual store levels. By leveraging a multi-year transactional sales dataset, we developed a data pipeline that includes data enrichment, exploratory analysis, feature engineering, and model training. We compared multiple models and selected XGBoost for its superior accuracy (MAE: 10.73, RMSE: 13.92). The final model was integrated into two role-specific web applications built using Streamlit — one for sales representatives to manage orders across stores, and one for store owners to monitor performance and forecast needs. This end-to-end solution demonstrates how machine learning can directly empower business users with actionable insights for inventory planning and operational efficiency.

**Group Information: Team IPL**

**Vamshi Saini (WIN: 527864349).**

**Hardik Mahto (WIN: 129569035).**

**Rohan Patil (WIN: 295879943).**

**Submitted to:**

**Prof. A.C. Fong.**

**Table of Contents**

1. **Introduction** 3
2. **Dataset Overview and Initial Setup** 3
3. **Exploratory Data Analysis (EDA)** 4
4. **Feature Engineering** 5
5. **Machine Learning Modeling** 7
6. **Application Integration** 8
7. **Conclusion** 8
8. **References** 9

# 1. Introduction

In the competitive landscape of retail and supply chain management, inventory forecasting plays a critical role in optimizing operations and improving customer satisfaction. Businesses need reliable systems to anticipate future demand, avoid stockouts or overstocking, and manage logistics proactively.  
  
This project focuses on building an intelligent forecasting system using machine learning techniques to predict product demand at individual store levels. Our objective is to use historical sales data across multiple stores and products to forecast how much quantity of each product will be required in the upcoming week. These predictions help both sales representatives and store owners in decision-making and inventory planning.  
  
In addition to model development, we also aimed to deploy the forecasting pipeline through two user-facing applications:  
- One for sales reps to monitor and plan stock orders for their assigned stores  
- One for store owners to track their store performance, analyze trends, and view predicted needs  
  
This report outlines the complete pipeline — from raw data ingestion, feature engineering, machine learning model development, model evaluation, to application integration.

# 2. Dataset Overview and Initial Setup

**2.1 Raw Dataset**  
The core of our analysis and model development started with a historical sales dataset named train.csv. This dataset contained over 9 million rows of daily-level sales information spanning from January 1, 2013 to December 31, 2017. Each row recorded:  
- date: The date of the sale.  
- store: Store ID (1–10).  
- item: Product ID (1–50).  
- sales: Units sold on that date.  
  
**2.2 Supporting Metadata Creation**  
We manually created three supporting datasets:  
- products\_table.csv: Assigned item IDs unique whiskey names.  
- stores\_table.csv: 10 stores with names, locations, and sales rep IDs.  
- sales\_reps\_table.csv: 10 mock sales reps with names and contact info.  
  
**2.3 MySQL Database Setup**  
We created a structured database named Saini\_Distributions in MySQL, populating tables from the above CSVs using Python’s pandas and SQLAlchemy.  
  
**2.4 Merging Datasets**  
We merged all datasets into a single DataFrame that included contextual data for every sale: product, store, sales rep, location, and date. This was saved as merged\_sales\_data.csv.

# 3. Exploratory Data Analysis (EDA)

EDA was carried out to explore store performance, product trends, seasonal effects, and overall demand patterns. This helped in guiding our feature engineering and understanding the business scenario.

**3.1 Objectives**

* Understand store-level performance.
* Identify best and least selling products.
* Explore rep impact.
* Detect seasonal patterns.
* Quantify demand consistency.

**3.2 Visual Insights**

* Store\_2, Store\_8, and Store\_3 led in total sales.
* Sales rep “Avery Blake” managed high-performing stores.
* Weller Special Reserve, Calumet Farm were top sellers.
* Seasonal peaks observed in the mid of the year.
* Some products showed steady demand, others high volatility.

**3.3 Clean-Up**  
Charts like 'Average Daily Sales' were excluded due to unclear trends.  
  
**3.4 Takeaways**

* Seasonality is strong.
* Some products are reliably consistent.
* Useful features were identified for modeling.

**Source file: Data\_modeling.ipynb**

A graph of sales by store

AI-generated content may be incorrect.A graph of a graph showing different colored lines

AI-generated content may be incorrect.

A graph with red squares

AI-generated content may be incorrect. A graph of a product

AI-generated content may be incorrect. A graph of a product

AI-generated content may be incorrect.A graph of sales by store

AI-generated content may be incorrect.A graph of a product

AI-generated content may be incorrect. A graph of red bars

AI-generated content may be incorrect.

# 4. Feature Engineering

To make the raw time-series data suitable for supervised learning, we engineered several types of features. This resulted in a structured tabular dataset where each row represented a specific product in a store on a given day, enriched with historical context and metadata.

**4.1 Lag Features**

* **lag\_1 (Sales one day ago)**:  
  This feature tells the model what the sales were *yesterday* for a specific product at a specific store. It’s useful for identifying very short-term trends or momentum in demand (e.g., a sudden spike or drop).
* **lag\_7 (Sales one week ago)**:  
  This captures the sales from *exactly one week ago*, helping the model detect *weekly seasonality*. For example, if sales tend to spike on weekends, this feature helps the model anticipate similar patterns this week.
* **lag\_14 (Sales two weeks ago)**:  
  This goes even further back and reinforces the *repetition of patterns* across weeks. If a product performs similarly every two weeks (e.g., due to bi-weekly promotions), this helps the model pick that up.

These lag features serve as historical memory. Time-series patterns, especially short-term trends and weekly cycles can’t be captured unless we explicitly include past values as inputs.

**4.2 Rolling Statistics**

* **rolling\_mean\_7 (7-day average sales)**:  
  This shows the average daily sales for the product over the last 7 days. It smooths out short-term fluctuations and gives the model a sense of the *recent sales trend*. It’s like saying, “On average, how has this item performed lately?”
* **rolling\_std\_7 rolling\_std\_7 (7-day standard deviation of sales)**:  
  This tells us how *stable or volatile* the sales have been over the past week. A low value means sales are consistent (e.g., a product with loyal demand), while a high value suggests unpredictable behavior (e.g., promotional or impulse-driven purchases).

These rolling metrics help the model balance its confidence. If recent sales are steady, it can trust the lag features more. If they’re volatile, the model might rely more on other signals like seasonal patterns or product type.

**4.3 Temporal Features**

* month: Extracted from the date (1 to 12). Captures **seasonal trends** like high sales in December (holiday season) or summer months for specific products.
* week: Week number of the year (1 to 52). Helps track broader demand cycles, promotions, or events occurring on a weekly basis (e.g., Week 26 may be summer peak).
* day\_of\_week: Values from 0 (Monday) to 6 (Sunday). This is crucial for capturing **weekly cyclic behavior** — many stores have more traffic on weekends, or promotions may happen mid-week.

These features help the model align its predictions with calendar-based events and shopper behavior. Without them, it wouldn't know that Friday sales tend to be higher than Monday’s, or that November usually brings a surge in demand.  
 **4.4 One-Hot Encoding**

One-hot encoding expanded categorical features into multiple binary columns, enabling the model to learn store and product-specific behaviors. We applied to store name, product name, and sales rep name for categorical representation.  
  
**4.5 Target Variable**  
We defined the target as total product sales over the next 7 days. The final dataset had columns like:  
- store, item, lag features, rolling stats, date parts, and binary columns for categories.  
  
This setup allowed the model to capture trends, seasonality, and store-product-specific behavior effectively.

# 5. Machine Learning Modeling

The core objective of our project was to forecast the product demand for the upcoming week. To achieve this, we formulated the problem as a **supervised regression task**, where the model learns from historical data and predicts the expected quantity of sales for each (store, product) pair. We engineered a range of features capturing time-based trends, store-level and product-level characteristics, and sales representative information. The **target variable** was defined as the **total sales over the next 7 days** for each product in each store.

#### 5.1 Models Evaluated

We trained and evaluated multiple models to identify the most suitable one for our use case:

* **Linear Regression**: This served as our baseline model. It is simple, interpretable, and assumes a linear relationship between input features and the target variable. While it provides a benchmark for comparison, it typically fails to capture complex patterns in real-world sales data.
* **Random Forest Regressor**: An ensemble method that builds multiple decision trees and aggregates their predictions. This model can capture non-linear relationships and feature interactions, and it usually improves performance over linear models by reducing variance.
* **XGBoost Regressor**: This gradient boosting algorithm produced the best results in our project. XGBoost is known for its ability to handle missing values, capture intricate non-linearities, and prevent overfitting through built-in regularization. Despite minimal hyperparameter tuning, it delivered superior performance across all evaluation metrics.

The final XGBoost model was trained on the full dataset and saved in a serialized format (xgboost\_model.pkl) for use in both deployed applications.

#### 5.2 Evaluation Metrics

To compare the models quantitatively, we used three standard metrics for regression:

* **Mean Absolute Error (MAE)**: This metric represents the average absolute difference between actual and predicted sales values. It is easy to interpret — a lower MAE means the model is more accurate on average.
* **Root Mean Squared Error (RMSE)**: RMSE penalizes larger errors more than MAE by squaring the differences before averaging. It’s a good indicator of how sensitive the model is to outliers.
* **Mean Absolute Percentage Error (MAPE)**: MAPE expresses the prediction error as a percentage of the actual value, making it easier to interpret across different scales. It is particularly useful for comparing models when the magnitude of values varies significantly.

These metrics together provided a holistic understanding of model performance, helping us select XGBoost as the most reliable and accurate option for deployment.  
  
**5.3 Results**

|  |  |  |  |
| --- | --- | --- | --- |
| Model | MAE | RMSE | MAPE |
| Linear Regression | 11.26 | 14.64 | 3.19% |
| Random Forest | 11.22 | 14.54 | 3.18% |
| XGBoost | 10.73 | 13.92 | 3.04% |

We selected XGBoost for deployment due to its superior accuracy, efficiency, and built-in regularization.

# 6. Application Integration

We developed two Streamlit web applications to serve different users:  
  
**6.1 Sales Rep Forecast App**  
- Lets sales reps log in and see forecasted product demand for their assigned stores.  
- Shows most sold (Top 5) and least sold (Bottom 5) products across stores.  
  
**6.2 Store Owner App**  
- Allows store managers to view weekly forecasts and analyze sales trends.  
- Application includes charts, tables, and product-level summaries for evaluation and monitoring of the store performance.  
  
**6.3 Backend Design**  
Both apps load the trained XGBoost model and pull latest data from MySQL. The pre saved model will perform prediction on-the-fly and send the results to the application which are then displayed dynamically respective to sales reps and stores.  
  
The system is modular and supports easy data updates and user-specific interfaces.

**Source files: sales\_forecast\_app.py & store\_owner\_app.py**

# 7. Conclusion

This project demonstrated the application of machine learning to solve a real-world demand forecasting problem.  
We cleaned and enriched raw sales data, performed detailed EDA, engineered relevant features, trained multiple ML models, and built lightweight applications to empower non-technical users with predictive insights.  
  
XGBoost was chosen for deployment due to its high accuracy (MAPE of 3.04%) and robustness. The Streamlit applications provide personalized dashboards for both sales reps and store owners, helping them make timely and data-driven inventory decisions.  
  
Future work could include incorporating external data like holidays or promotions, providing more options for sales reps to order items directly hassle free in the application. Other possible work includes, fine-tuning the model with hyperparameter optimization, and implementing long-term forecasts.

# References

[1] S. Shekhar, "Types of Demand Forecast," GeeksforGeeks, Sep. 6, 2022. [Online]. Available: <https://www.geeksforgeeks.org/types-of-demand-forecast/>  
[2] Y. Gülcan, "Linear Regression vs XGBoost Regressor," Kaggle Notebooks, [Online]. Available: <https://www.kaggle.com/code/yusufglcan/linear-regression-vs-xgboost-regressor>  
[3] XGBoost Developers, “XGBoost Documentation,” XGBoost, [Online]. Available: <https://xgboost.readthedocs.io/en/stable/python/python_intro.html>  
[4] Streamlit Inc., “Streamlit — The fastest way to build and share data apps,” Streamlit.io, [Online]. Available: <https://streamlit.io/>