**Inventory Product Demand Forecasting**

A Mini Project Report

Submitted by

**Nithenkumar S 231801119**

**Nithun kumar R S 231801121**

In partial fulfilment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

**RAJALAKSHMI ENGINEERING COLLEGE,**

**ANNA UNIVERSITY : CHENNAI 600025**

**2024 - 2025**

**BONAFIDE CERTIFICATE**

Certified that this project report **“INVENTORY PRODUCT DEMAND FORECASTING”** is the Bonafide work of **“NITHENKUMAR S(231801119), NITHUNKUMAR R S(231801121)”** who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**Dr. Sekar K, M.E ., Ph.D., Mrs. Y. NIRMALA ANANDHI**

**HEAD OF THE DEPARTMENT SUPERVISOR**

**AND PROFESSOR AND ASSISTANT PROFESSOR**

Department of Artificial Intelligence Department of Artificial Intelligence

and Data Science and Data Science

Rajalakshmi Engineering College, Rajalakshmi Engineering College,

Thandalam, Chennai – 602 105 Thandalam, Chennai – 602 105

Submitted for Project Viva-Voce Examination held on**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**TABLE OF CONTENT**

**CHAPTER NO. TITLE PAGE NO  
   
 ACKNOWLEDGEMENT**

**ABSTRACT   
   
 1 INRODUCTION** 1.1 GENERAL  
  
 1.2 NEED FOR THE STUDY  
  
 1.3 OVERVIEW FOR THE PROJECT  
  
 1.4 OBJECTIVES OF THE STUDY  
  
 **2 SYSTEM REQUIREMENTS** 2.1 HARDWARE REQUIREMENTS  
  
 2.2 SOFTWARE REQUIREMENTS  
  
 **3 SYSTEM OVERVIEW** 3.1 MODULE 1-DATA COLLECTION  
 AND PREPROCESSING  
  
 3.2 MODULE 2-MODEL DEVELOPMENT  
 TRAINING AND EVALUATION  
  
 **4**  **RESULT AND DISCUSSION  
  
 5 CONCLUSION**   
  
 **6 APPENDIX  
  
 7 REFERENCES**

**ACKNOWLEDGEMENT**

We begin by expressing our deepest gratitude to the Almighty for bestowing upon us the strength, knowledge, and perseverance to successfully complete this mini-project on inventory product demand forecasting.

We are profoundly thankful to our Chairman, Mr. S. MEGANATHAN, B.E., F.I.E., our Vice Chairman, Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S., and our respected Chairperson, Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D., for creating a platform for learning and innovation through their premier institution. Their vision and commitment to excellence have provided us with the infrastructure and opportunities needed for academic and professional growth.

We extend our heartfelt thanks to our Principal, Dr. S.N. MURUGESAN, M.E., Ph.D., for his continuous support and the facilities made available to ensure the smooth execution of this project. Our sincere gratitude also goes to Dr. K. SEKAR, M.E., Ph.D., Head of the Department of Artificial Intelligence and Machine Learning, for his invaluable guidance and encouragement throughout this endeavor.

We owe special thanks to our internal guide, Mrs. Y. NIRMALAANANDHI, M.E., Assistant Professor, Department of Artificial Intelligence and Machine Learning, Rajalakshmi Engineering College, for her steadfast support and expert guidance, which have been pivotal in the successful execution of this project. We also wish to express our sincere gratitude to our Project Coordinator, Mr. K. GOPINATH, M.E., Assistant Professor, for his insightful suggestions and constructive feedback during our review sessions, which helped refine our work significantly.

Finally, we are deeply grateful to the retail store owner whose data served as the backbone of this project. Their cooperation in providing access to essential sales data enabled us to build and validate our forecasting model. This project has been a collaborative effort, and we remain indebted to everyone who contributed to its successful completion.

**ABSTRACT**

Effective inventory management is critical for multi-channel retail stores to meet customer demand while minimizing costs associated with overstocking or understocking. This study explores the use of machine learning algorithms to develop a demand forecasting system tailored for a retail store in Thailand, with sales data collected over five years (2017–2021) from both online and offline channels. The research focuses on building and comparing forecasting models using the CatBoost algorithm, XGBoost algorithm, and Linear Regression, evaluating their performance over 7-day and 30-day forecasting periods.

Through rigorous data preprocessing and feature engineering, the study addresses challenges such as inconsistent sales trends and seasonal variations. The results show that XGBoost outperforms in short-term (7-day) forecasting with lower error metrics, while CatBoost demonstrates superior performance in long-term (30-day) forecasting. Both models show high accuracy, with SMAPE values of 24.13% and 24.47%, respectively.

This demand forecasting system, when integrated into a web application using the Flask API framework, provides retail stores with actionable insights to optimize inventory levels, reduce costs, and improve customer satisfaction. The study highlights the effectiveness of machine learning in solving real-world inventory challenges in the retail sector.

The proposed system highlights the effectiveness of machine learning in addressing inventory challenges, emphasizing the role of data preprocessing and feature engineering in improving forecast accuracy. By leveraging historical sales data and integrating features like seasonal trends and lag indicators, the models provide reliable predictions for inventory optimization and strategic planning. This approach transforms traditional inventory management into a predictive, data-driven solution, offering valuable insights for supply chain efficiency and targeted marketing.

**CHAPTER 1**

**1. INRODUCTION**

**1.1 GENERAL**

In the modern retail landscape, managing inventory efficiently is crucial for maintaining customer satisfaction while minimizing operational costs. Retail stores, particularly those with multi-channel operations, face significant challenges in balancing product demand across various sales channels, both online and offline. These challenges often lead to issues such as overstocking or stockouts, which not only affect profitability but also impact customer experience.

To address these issues, demand forecasting has emerged as a critical tool for predicting future product needs and optimizing inventory levels. Machine learning techniques, such as CatBoost, XGBoost, and Linear Regression, offer powerful methods to analyze historical sales data and predict future demand with higher accuracy. In this study, a demand forecasting system is developed using data from a retail store in Thailand, spanning from 2017 to 2021, to forecast product demand for both short-term (7 days) and long-term (30 days) periods. The paper explores the effectiveness of these machine learning algorithms in improving inventory management by reducing costs associated with overstocking, storage, and labor, while also ensuring that products are available to meet customer demand.

**Source for Inventory Product Management**  
For a broader understanding of inventory product management, various research papers and industry reports suggest that effective demand forecasting models, powered by machine learning, can significantly enhance the accuracy of inventory planning. As outlined in the IEEE paper on demand forecasting for multi-channel retailing stores, machine learning algorithms like CatBoost and XGBoost have proven to outperform traditional methods by offering more precise and timely forecasts, ultimately improving product availability and reducing waste. This integration of advanced technologies not only helps in managing current stock levels but also aids in strategic long-term inventory planning

**1.2 Need for the Study**

Efficient inventory management is a critical challenge faced by retail stores, especially those operating across multiple sales channels, including both online and offline platforms. Inaccurate inventory management often leads to issues such as stockouts, overstocking, and delayed shipments, all of which can negatively impact customer satisfaction, increase operational costs, and reduce profitability. Traditional inventory management practices, which rely heavily on manual processes and historical judgment, may no longer be sufficient in today’s fast-paced retail environment.

With the advent of e-commerce and multi-channel retailing, there is a growing need for more sophisticated and data-driven approaches to forecast product demand accurately. Predicting future demand allows retailers to optimize inventory levels, minimize costs related to excess stock or lost sales, and improve overall supply chain efficiency. Machine learning algorithms, such as CatBoost, XGBoost, and Linear Regression, have proven to be highly effective in demand forecasting by analyzing large datasets and recognizing complex patterns that human judgment may miss.

This study is essential as it explores the potential of machine learning models to address these challenges. By leveraging historical sales data from a retail store in Thailand, the study aims to develop accurate demand forecasting models for both short-term and long-term periods, helping retail stores make informed decisions about inventory management. The outcomes of this study will provide valuable insights into how machine learning can be integrated into inventory management systems, thus improving operational efficiency, reducing costs, and enhancing customer satisfaction.

**1.3 Overview of the Project**

This project focuses on developing a demand forecasting system for inventory management in a multi-channel retailing environment. With increasing competition and the complexity of managing both online and offline sales channels, accurate demand forecasting has become essential for ensuring that retail stores maintain optimal inventory levels while minimizing costs associated with stockouts or overstocking.

The project utilizes historical sales data from a retail store in Thailand, covering a period from 2017 to 2021. This data, which includes sales transactions across multiple branches and distribution channels, is preprocessed and analyzed to generate key features such as seasonal trends, lag values, and moving averages. The goal is to build machine learning models that can accurately predict product demand for both short-term (7-day) and long-term (30-day) forecasting periods.

Three machine learning algorithms—CatBoost, XGBoost, and Linear Regression—are compared for their ability to forecast demand. The project evaluates the performance of these models using metrics such as Root Mean Squared Error (RMSE), Mean Absolute Deviation (MAD), and Symmetric Mean Absolute Percentage Error (SMAPE), providing a quantitative assessment of their accuracy.

Ultimately, this project aims to create a reliable demand forecasting system that can be integrated into the retail store’s inventory management system. The system will help the store optimize stock levels, reduce costs, and improve customer satisfaction by ensuring the availability of products while minimizing waste. The findings of this project demonstrate the practical application of machine learning in solving real-world challenges faced by retailers in managing their inventory across multiple channels.

**1.4 Objectives of the Study**

The main objectives of this study are as follows:

1. **To Develop a Demand Forecasting Model:**  
   The primary goal of this study is to build accurate demand forecasting models using machine learning algorithms such as CatBoost, XGBoost, and Linear Regression. These models aim to predict product demand for both short-term (7-day) and long-term (30-day) periods to assist in efficient inventory management.
2. **To Compare the Performance of Different Algorithms:**  
   The study aims to evaluate and compare the effectiveness of three machine learning algorithms—CatBoost, XGBoost, and Linear Regression—by assessing their performance based on various metrics such as Root Mean Squared Error (RMSE), Mean Absolute Deviation (MAD), and Symmetric Mean Absolute Percentage Error (SMAPE). This comparison will identify the most suitable model for demand forecasting in a multi-channel retailing environment.
3. **To Optimize Inventory Management:**  
   By accurately forecasting product demand, the study seeks to help retail stores optimize their inventory levels, ensuring that products are neither overstocked nor understocked. This will reduce inventory holding costs and minimize the risk of stockouts, leading to better customer satisfaction.
4. **To Analyze the Impact of Data Preprocessing and Feature Engineering:**  
   The study aims to demonstrate the importance of data preprocessing and feature engineering techniques, such as handling missing values, removing outliers, and generating meaningful features (seasonal trends, lag values, moving averages), in improving the accuracy of demand forecasting models.
5. **To Provide Insights for Retail Store Decision-Making:**  
   The study intends to offer actionable insights to retail stores by providing a demand forecasting system that can be integrated into their existing inventory management systems. These insights will help store owners and managers make informed decisions regarding product procurement, stocking, and distribution.
6. **To Contribute to the Field of Machine Learning in Retail Analytics:**  
   This study aims to contribute to the growing body of research in applying machine learning techniques to solve real-world problems in retail analytics, specifically in demand forecasting and inventory management. The findings will help in the adoption of machine learning solutions by small and medium-sized retail businesses.
7. **To Enhance Supply Chain Efficiency:**  
   The study aims to enhance the overall efficiency of the retail store’s supply chain by providing more accurate demand predictions. This will enable better coordination between procurement, inventory, and distribution, reducing delays and minimizing the risk of both stockouts and excess inventory.
8. **To Foster Data-Driven Decision Making:**  
   By implementing machine learning-driven demand forecasting, the study seeks to foster a data-driven approach to decision-making in retail operations. This will help retailers transition from reliance on traditional methods to more scientific and automated systems that can adapt to dynamic market conditions and customer behavior.

**CHAPTER 2**

**2. SYSTEM REQUIREMET**

**2.1 HARDWARE REQUIREMENTS**

For efficient data processing and machine learning algorithm execution, a capable hardware setup is essential. This project requires a well-configured system to handle large datasets and complex models, ensuring smooth and efficient operations. Below are the key hardware requirements:

**Processor (CPU):**  
A multi-core processor, such as the **Intel Core i5** or an equivalent with a clock speed of 2.5 GHz or higher, is recommended for efficient data processing and running machine learning algorithms.

**Memory (RAM):**  
At least **8 GB of RAM** is necessary for handling large datasets and performing the computations involved in training machine learning models. For more complex datasets or larger models, **16 GB or more** is preferred.

**Storage (Hard Drive):**  
A minimum of **100 GB of free storage** is required to store datasets, preprocessed data, and model outputs. Using a **Solid State Drive (SSD)** is recommended for faster data access and processing speeds.

**Graphics Processing Unit (GPU):**  
A **dedicated GPU** may be used to accelerate model training, particularly for larger datasets or deep learning tasks. A GPU with at least **4 GB of VRAM** (such as the NVIDIA GTX or RTX series) will improve performance.

**Display:**  
A monitor with a minimum resolution of **1080p (1920 x 1080)** is recommended to facilitate data analysis and visualization.

**Internet Connection:**  
A stable **internet connection** is required to access cloud resources, download datasets, and install necessary machine learning libraries.

**Backup and Storage Devices:**  
It is advised to use **external storage** (e.g., USB drives or cloud storage) for backing up project data and model results to ensure data security and easy recovery.

These hardware specifications will ensure smooth execution of the data processing, machine learning model training, and overall project development.

**2.2 Software Requirements**

To implement and run the demand forecasting system for inventory management efficiently, the following software tools and platforms are essential. These tools are crucial for the data preparation, model development, and deployment phases of the project.

1. **Operating System:**
   * **Windows 10/11** or **Linux (Ubuntu):**  
     The operating system serves as the base platform for running the development environment and all the necessary tools. Both Windows and Linux support all required libraries and packages for machine learning tasks. Windows provides ease of use, especially for users who are familiar with GUI-based environments, while Linux (specifically Ubuntu) offers a stable and open-source platform with advanced command-line tools, making it popular for development and deployment in machine learning projects.
2. **Programming Languages:**
   * **Python 3.x:**  
     Python is the primary programming language for this project due to its simplicity and vast ecosystem of libraries and frameworks tailored to data science, machine learning, and data visualization. Python is used for tasks such as:
     + Data manipulation and cleaning using libraries like **Pandas**.
     + Developing machine learning models using **Scikit-learn**, **CatBoost**, and **XGBoost**.
     + Visualizing results with **Matplotlib** and **Seaborn**. Python 3.x is preferred over earlier versions because of its support for newer features and libraries.
3. **Integrated Development Environment (IDE):**
   * **PyCharm** or **Visual Studio Code:**  
     These are the most popular IDEs for Python development. They offer features like:
     + Code completion, syntax highlighting, and error checking.
     + Integrated version control with Git.
     + Debugging tools for identifying and solving coding issues.
     + PyCharm, a dedicated IDE for Python, offers powerful features, while **Visual Studio Code** is lightweight and highly customizable with a variety of extensions for Python.
   * **Jupyter Notebook** or **Google Colab:**  
     Jupyter Notebook is essential for interactive coding, where data exploration, analysis, and visualizations are carried out. It allows you to run code in cells and immediately see the output, which is particularly useful for data science tasks and machine learning model evaluation.  
     **Google Colab** is a cloud-based alternative to Jupyter Notebook that provides free access to GPUs and TPUs, making it useful for training large models or datasets. It's particularly useful when resources are limited on local machines.
4. **Machine Learning Libraries:**
   * **Scikit-learn:**  
     **Scikit-learn** is one of the most commonly used libraries for machine learning in Python. It provides a wide range of tools for building models, preprocessing data, and evaluating model performance. It includes algorithms like **Linear Regression** and tools for splitting datasets, feature scaling, and model evaluation.
   * **CatBoost:**  
     **CatBoost** is a gradient boosting library developed by Yandex. It is highly effective for structured data and is known for its excellent performance without requiring significant hyperparameter tuning. This library will be used to build the demand forecasting model, particularly for its ability to handle categorical data efficiently.
   * **XGBoost:**  
     **XGBoost** is another gradient boosting algorithm widely used for structured data. It is highly efficient and optimized for performance. Like CatBoost, it will be used for building predictive models, with XGBoost often offering high accuracy in demand forecasting tasks.
   * **Pandas:**  
     **Pandas** is the go-to library for data manipulation in Python. It allows for efficient handling and analysis of structured data, particularly in the form of DataFrames. It will be used to clean, preprocess, and manipulate the sales data before it is fed into machine learning models.
   * **NumPy:**  
     **NumPy** provides support for large, multi-dimensional arrays and matrices, and it is used in conjunction with other libraries to perform numerical operations. It is often a foundation for other data science libraries, and it will be crucial for handling numerical data and performing statistical operations.
   * **Matplotlib/Seaborn:**  
     **Matplotlib** and **Seaborn** are the primary libraries used for data visualization. **Matplotlib** provides basic plotting functionalities, while **Seaborn** offers more advanced and aesthetically pleasing visualizations. These will be used for visualizing model performance, data trends, and other key aspects of the demand forecasting system.
   * **TensorFlow/Keras (optional):**  
     While not necessarily required for this project, **TensorFlow** or **Keras** can be employed if deep learning models are introduced in the future for more complex forecasting tasks. These libraries are popular for training neural networks and handling large-scale data problems.
5. **Data Management Tools:**
   * **SQL Database (optional):**  
     For retail stores that use databases to store their sales data, **SQL** or **NoSQL** databases such as **MySQL**, **SQLite**, or **MongoDB** may be used to retrieve and manage data. These databases will be helpful if the dataset is stored in relational tables or requires querying from multiple sources.
   * **Excel:**  
     **Excel** is an excellent tool for initial data exploration, cleaning, and simple visualization. It is particularly useful for small datasets, or for inspecting the raw data before importing it into Python for further analysis.
6. **Version Control:**
   * **Git:**  
     **Git** is essential for version control, enabling you to track changes in your codebase and collaborate with others. It allows you to revert back to previous versions of your code and manage different development branches efficiently.
   * **GitHub** or **GitLab:**  
     These platforms provide cloud-based repositories for storing and sharing code. They also support collaborative development, enabling multiple team members to contribute to the project, and allowing access to the project from any location.
7. **Cloud Platforms (optional):**
   * **Google Cloud Platform (GCP), Amazon Web Services (AWS), or Microsoft Azure:**  
     These platforms offer cloud-based storage and compute resources, especially useful for handling large datasets and executing models that require more computational power than a local machine can provide. Using cloud resources can speed up model training and allow for scalability when working with bigger data.
8. **Web Framework (for deployment):**
   * **Flask:**  
     **Flask** is a lightweight Python web framework used for deploying applications. In this project, Flask will be used to build a web-based interface where the demand forecasting models can be deployed and accessed. It will integrate the machine learning model into a user-friendly web application, allowing retail store managers to interact with the forecasting system.

These software tools and libraries form the backbone of the project, providing the necessary infrastructure for data manipulation, model training, evaluation, and deployment. They ensure that the project is scalable, efficient, and capable of handling large datasets and complex forecasting tasks.

**CHAPTER 3**

**3. SYSTEM OVERVIEW**

**3.1 MODULE 1-DATA COOLECTION AND PREPROCESSING**

**3.1 Module 1: Data Collection and Preprocessing**

The first module focuses on collecting and preparing data for the demand forecasting model. Proper data preprocessing ensures that the raw data is cleaned, structured, and ready for analysis and model building. The steps in this module include:

* **Data Collection**:  
  The sales data is collected from various sources, including retail stores and online channels. The dataset spans from **2017 to 2021** and includes transaction details, product quantities, sales prices, and information from multiple branches. This data is consolidated into a single dataset for easier analysis.
* **Data Cleaning**:  
  The raw data often contains missing values, irrelevant information, or errors. Techniques such as **imputation**, **removal of duplicate records**, and filtering out irrelevant columns (e.g., customer data, shipping details) are applied. Missing values are addressed using methods such as replacing them with the mean, mode, or removing rows where the data is crucial.
* **Feature Engineering**:  
  To enhance the accuracy of demand forecasting, several features are created, such as:
  + **Date-related features**: Extracting features like day of the week, month, year, and seasonality to capture trends.
  + **Lag features**: Including past sales data (e.g., sales from 7 days ago) to capture temporal dependencies.
  + **Moving Averages**: Calculating rolling averages over specific periods to smooth out fluctuations and reveal trends.
* **Data Transformation**:  
  The data is transformed into a consistent format. Dates are standardized, and categorical variables like product IDs and branch locations are encoded into numerical values using **Label Encoding** or **One-Hot Encoding**. This step ensures that all variables are appropriately formatted for machine learning models.
* **Data Scaling and Normalization**:  
  The numerical features (e.g., sales quantities and prices) are scaled to a common range using techniques like **Min-Max Scaling** or **Standardization** to ensure that no feature dominates during model training.
* **Data Splitting**:  
  The dataset is divided into **training**, **validation**, and **testing** sets to evaluate the performance of the machine learning models and prevent overfitting.

The outcome of this module is a clean, structured dataset ready for training machine learning models.

**3.2 Module 2: Model Development, Training, and Testing**

After the data has been collected and preprocessed, the next step is to develop and train the forecasting models. This module focuses on building, training, and testing the models to predict product demand. The process is broken down into the following steps:

* **Model Selection**:  
  Several machine learning algorithms are selected for comparison. For this project, the following models will be used:
  + **CatBoost**: A gradient boosting algorithm known for its efficiency and performance with categorical data.
  + **XGBoost**: Another gradient boosting algorithm that has been widely used in predictive modeling tasks due to its speed and performance.
  + **Linear Regression**: A traditional statistical model that will serve as a baseline for comparison.
* **Model Training**:  
  The models are trained using the preprocessed training dataset. The **training set** contains historical sales data, and the model learns the patterns and relationships between features (such as product, time, and sales quantity) and target variables (demand forecasts). Hyperparameters may be tuned to improve performance.
* **Model Evaluation**:  
  After training the models, their performance is evaluated using the **validation dataset**. Metrics such as **Root Mean Squared Error (RMSE)**, **Mean Absolute Deviation (MAD)**, and **Symmetric Mean Absolute Percentage Error (SMAPE)** are used to assess the accuracy of the models. These metrics provide insight into how well the models are predicting product demand.
* **Model Testing**:  
  The models are then tested using a **testing dataset** that the models have never seen before. This helps evaluate the generalization ability of the models and ensures that they are not overfitting to the training data.
* **Model Comparison**:  
  Once all models are trained and tested, their performance is compared to determine the best performing model. The model with the lowest RMSE and SMAPE values, along with the highest prediction accuracy, is selected as the final model for demand forecasting.
* **Model Deployment Preparation**:  
  Once the best model is selected, it is further tuned and optimized for real-world deployment. The model is then integrated into a web application or inventory management system for use by retail store managers to forecast demand.

The outcome of this module is the development of a reliable demand forecasting model that accurately predicts product demand and can be used to optimize inventory management.

**CHAPTER 4**

**4. RESULT AND DISCUSSION**

**4.1 RESULT**

The following are the results of the demand forecasting models for both short-term (7-day) and long-term (30-day) periods based on the evaluation metrics: Root Mean Squared Error (RMSE), Mean Absolute Deviation (MAD), and Symmetric Mean Absolute Percentage Error (SMAPE).

For 7-Day Demand Forecasting:

* CatBoost:
  + RMSE: 0.331136
  + MAD: 0.594066
  + SMAPE: 24.13%
* XGBoost:
  + RMSE: 0.324897
  + MAD: 0.541571
  + SMAPE: 23.48%
* Linear Regression:
  + RMSE: 0.339751
  + MAD: 0.511551
  + SMAPE: 25.12%

For 30-Day Demand Forecasting:

* CatBoost:
  + RMSE: 0.332399
  + MAD: 1.328
  + SMAPE: 24.47%
* XGBoost:
  + RMSE: 0.334674
  + MAD: 1.202771
  + SMAPE: 24.85%
* Linear Regression:
  + RMSE: 0.343910
  + MAD: 1.498935
  + SMAPE: 26.14%

**4.2 Discussion**

The results of the demand forecasting models are analyzed to determine their performance and suitability for predicting product demand.

Model Comparison:

* XGBoost performed the best for short-term demand forecasting (7-day period) with the lowest RMSE (0.324897) and SMAPE (23.48%). This suggests that XGBoost is highly effective at capturing the immediate patterns and fluctuations in product demand. Its ability to capture short-term trends makes it a suitable choice for short-term inventory management.
* CatBoost, while slightly less accurate than XGBoost for the 7-day forecast, showed superior performance for long-term demand forecasting (30-day period). With an RMSE of 0.332399 and SMAPE of 24.47%, it handled the long-term demand trends better. This indicates that CatBoost is more robust in handling extended periods, capturing the broader patterns and seasonality in demand.
* Linear Regression, the simplest model used in this study, showed relatively poor performance when compared to the advanced gradient boosting models. It had higher RMSE and SMAPE values in both the 7-day and 30-day forecasting tasks, indicating that Linear Regression struggles to capture the complexities in the data, such as seasonal variations and trends over time.

Importance of Feature Engineering and Preprocessing:

* The preprocessing techniques, such as handling missing values, feature scaling, and feature engineering (e.g., generating date-related features, lag features, and moving averages), were crucial for improving the models' performance. By incorporating time-related features like the day of the week, month, and seasonality, as well as lag features to capture historical demand, the models could better predict future sales.
* The transformations applied to categorical variables (via One-Hot Encoding or Label Encoding) and the normalization of numerical features ensured that the data was in an appropriate format for the machine learning models to process effectively.

Practical Implications:

* The high performance of XGBoost and CatBoost in both short-term and long-term forecasting demonstrates their potential in real-world applications for inventory management. These models can help retailers forecast demand accurately, leading to better inventory planning, reduced excess stock, and minimized stockouts.
* The SMAPE values, which are around 24-25%, indicate that the models are reasonably accurate but could still be improved. Future work could explore advanced techniques, such as neural networks or ensemble learning, to further improve the forecast accuracy and adapt to changing market conditions.

In conclusion, XGBoost and CatBoost outperform Linear Regression in both short-term and long-term demand forecasting. These models provide reliable predictions that can be integrated into inventory management systems to optimize stock levels, enhance supply chain management, and improve customer satisfaction.

**CHAPTER 5**

**5. CONCLUSION**

This study focused on developing demand forecasting models for inventory management in multi-channel retail stores using machine learning algorithms. The goal was to create a system that could accurately predict product demand for both short-term (7-day) and long-term (30-day) periods, helping retail stores optimize inventory levels and reduce costs associated with overstocking and stockouts.

The results demonstrated that both XGBoost and CatBoost outperformed Linear Regression, with XGBoost excelling in short-term forecasts (7-day) and CatBoost providing superior performance for long-term forecasts (30-day). These models achieved promising results with SMAPE values in the range of 24-25%, indicating good forecast accuracy.

The importance of feature engineering and data preprocessing was clearly demonstrated, as transforming raw data into meaningful features—such as lag values, seasonal indicators, and moving averages—significantly improved the models' performance. Additionally, proper handling of missing values, encoding of categorical variables, and scaling of numerical features ensured that the models could learn from the data effectively.

In conclusion, the demand forecasting models developed in this study show strong potential for real-world applications in retail inventory management. By accurately predicting product demand, these models can help retailers optimize their inventory, reduce operational costs, and improve customer satisfaction. Future work may focus on refining the models further or integrating them into automated systems for real-time forecasting and decision-making.

**CHAPTER 6**

**APPENDIX**

**A. Libraries and Tools Used**

* **Libraries:**
  + pandas for data manipulation and cleaning.
  + numpy for numerical computations.
  + xgboost for implementing the forecasting model.
  + joblib for model persistence.
  + sklearn for evaluation metrics and dataset splitting.
* **Software/Environment:**
  + Python 3.x
  + Jupyter Notebook or a similar IDE.

**B. Data Preprocessing**

1. **Loading the Dataset:**
   * The dataset (data\_2013\_2016.csv) was loaded using pandas.
   * The date column was converted into a datetime format for time-based operations.
2. **Feature Engineering:**  
   Derived features were created to enhance the model's ability to capture demand patterns:
   * **Moving Averages:** Calculated 7-day, 30-day, and 90-day averages for historical sales trends.
   * **Seasonality Features:** Extracted month, quarter, and year to account for seasonal variations.
   * **Lag Features:** Created lagged sales data for 1-day, 7-day, and 30-day periods to model dependencies.
   * **Stock Dynamics:** Included stock\_turnover\_ratio, days\_until\_stockout, and stock\_to\_sales\_ratio to understand inventory behavior.
   * **Supplier Reliability:** Added supplier\_lead\_time\_variance to account for lead time variability.
3. **Handling Missing Values:**
   * Rows with missing values generated due to lagged features or rolling windows were dropped to maintain dataset integrity.
4. **Feature Selection:**
   * Key features for the model were identified, including sales trends, stock levels, seasonality indicators, and supplier metrics.
5. **Data Splitting:**
   * The data was split into training (80%) and testing (20%) sets using train\_test\_split to evaluate the model's performance on unseen data.

**C. Model Development**

1. **Algorithm Selection:**
   * The XGBoost regressor was chosen for its efficiency and robustness in handling structured data.
2. **Model Configuration:**
   * Hyperparameters included:
     + n\_estimators: 500
     + learning\_rate: 0.1
     + max\_depth: 8
     + subsample: 0.8
     + colsample\_bytree: 0.8
   * The model was trained with the objective reg:squarederror.
3. **Model Training:**
   * The training set was used to fit the XGBoost model.
4. **Model Saving:**
   * The trained model was saved as xgboost\_inventory\_model.joblib using the joblib library for future use.

5. Sample code

PYTHON

# Import required libraries

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

import xgboost as xgb

import joblib

# Step 1: Load the dataset

data = pd.read\_csv('data\_2013\_2016.csv') # Replace with your dataset file path

# Step 2: Data preprocessing

# Convert 'date' column to datetime format for easier date manipulations

data['date'] = pd.to\_datetime(data['date'])

# Step 3: Feature Engineering

# Create moving averages for sales over 7, 30, and 90 days

data['7\_day\_avg\_sales'] = data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=7, min\_periods=1).mean())

data['30\_day\_avg\_sales'] = data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=30, min\_periods=1).mean())

data['90\_day\_avg\_sales'] = data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=90, min\_periods=1).mean())

# Extract seasonality features: month, quarter, and year

data['month'] = data['date'].dt.month

data['quarter'] = data['date'].dt.quarter

data['year'] = data['date'].dt.year

# Create lag features: 1 day, 7 days, and 30 days lagged sales

data['sales\_lag\_1'] = data.groupby(['store', 'item'])['sales'].shift(1)

data['sales\_lag\_7'] = data.groupby(['store', 'item'])['sales'].shift(7)

data['sales\_lag\_30'] = data.groupby(['store', 'item'])['sales'].shift(30)

# Stock-related features

data['stock\_turnover\_ratio'] = data['sales'] / (data['stock\_on\_hand'] + 1e-5) # Avoid division by zero

data['days\_until\_stockout'] = data['stock\_on\_hand'] / (data['7\_day\_avg\_sales'] + 1e-5)

data['days\_since\_last\_restock'] = (data['date'] - data.groupby(['store', 'item'])['date'].shift()).dt.days.fillna(0)

data['stock\_to\_sales\_ratio'] = data['stock\_on\_hand'] / (data['sales'] + 1e-5)

# Supplier reliability feature

data['supplier\_lead\_time\_variance'] = data.groupby('supplier\_id')['supplier\_lead\_time'].transform('std').fillna(0)

# Weekend indicator

data['is\_weekend'] = data['date'].dt.weekday.apply(lambda x: 1 if x >= 5 else 0)

# Cumulative sales within a month

data['cumulative\_sales'] = data.groupby(['store', 'item', 'year', 'month'])['sales'].cumsum()

# Stock shortage indicator

data['stock\_shortage'] = ((data['stock\_on\_hand'] < data['stock\_reorder\_level']) | (data['stock\_on\_hand'] <= 0)).astype(int)

# Drop rows with missing values generated due to lagging or rolling operations

data.dropna(inplace=True)

# Step 4: Define features (X) and target variable (y)

features = [

'store', 'item', 'supplier\_id', 'supplier\_lead\_time', 'stock\_on\_hand',

'stock\_reorder\_level', 'replenishment\_lead\_time', 'month', 'quarter', 'year',

'7\_day\_avg\_sales', '30\_day\_avg\_sales', '90\_day\_avg\_sales',

'sales\_lag\_1', 'sales\_lag\_7', 'sales\_lag\_30',

'stock\_turnover\_ratio', 'days\_until\_stockout', 'days\_since\_last\_restock',

'stock\_to\_sales\_ratio', 'supplier\_lead\_time\_variance', 'is\_weekend',

'cumulative\_sales', 'stock\_shortage'

]

X = data[features]

y = data['sales']

# Step 5: Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 6: Initialize and train the XGBoost regressor

xgb\_regressor = xgb.XGBRegressor(

n\_estimators=500,

learning\_rate=0.1,

max\_depth=8,

subsample=0.8,

colsample\_bytree=0.8,

random\_state=42,

objective='reg:squarederror'

)

xgb\_regressor.fit(X\_train, y\_train)

# Step 7: Make predictions on the test set

y\_pred = xgb\_regressor.predict(X\_test)

# Step 8: Evaluate the model

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

mae = mean\_absolute\_error(y\_test, y\_pred)

print(f'RMSE: {rmse}')

print(f'MAE: {mae}')

# Step 9: Save the trained model for future use

joblib.dump(xgb\_regressor, 'xgboost\_inventory\_model.joblib')

**Predicting Sales Using a Trained XGBoost Model**

# Import necessary libraries

import pandas as pd

import joblib

# Step 1: Load the trained model

xgb\_regressor = joblib.load('xgboost\_inventory\_model.joblib') # Load the saved XGBoost model

# Step 2: Load the new data for prediction

new\_data = pd.read\_csv('data\_2017.csv') # Replace with the file path to your new dataset

# Step 3: Data Preprocessing on New Data

# Convert 'date' column to datetime format

new\_data['date'] = pd.to\_datetime(new\_data['date'])

# Re-create derived features as in the training phase

new\_data['7\_day\_avg\_sales'] = new\_data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=7, min\_periods=1).mean())

new\_data['30\_day\_avg\_sales'] = new\_data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=30, min\_periods=1).mean())

new\_data['90\_day\_avg\_sales'] = new\_data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=90, min\_periods=1).mean())

new\_data['month'] = new\_data['date'].dt.month

new\_data['quarter'] = new\_data['date'].dt.quarter

new\_data['year'] = new\_data['date'].dt.year

new\_data['sales\_lag\_1'] = new\_data.groupby(['store', 'item'])['sales'].shift(1)

new\_data['sales\_lag\_7'] = new\_data.groupby(['store', 'item'])['sales'].shift(7)

new\_data['sales\_lag\_30'] = new\_data.groupby(['store', 'item'])['sales'].shift(30)

new\_data['stock\_turnover\_ratio'] = new\_data['sales'] / (new\_data['stock\_on\_hand'] + 1e-5)

new\_data['days\_until\_stockout'] = new\_data['stock\_on\_hand'] / (new\_data['7\_day\_avg\_sales'] + 1e-5)

new\_data['days\_since\_last\_restock'] = (new\_data['date'] - new\_data.groupby(['store', 'item'])['date'].shift()).dt.days.fillna(0)

new\_data['stock\_to\_sales\_ratio'] = new\_data['stock\_on\_hand'] / (new\_data['sales'] + 1e-5)

new\_data['supplier\_lead\_time\_variance'] = new\_data.groupby('supplier\_id')['supplier\_lead\_time'].transform('std').fillna(0)

new\_data['is\_weekend'] = new\_data['day\_of\_week'].apply(lambda x: 1 if x >= 6 else 0)

new\_data['cumulative\_sales'] = new\_data.groupby(['store', 'item', 'year', 'month'])['sales'].cumsum()

new\_data['reorder\_frequency'] = new\_data.groupby(['store', 'item'])['restocked\_inventory'].transform(lambda x: x.rolling(window=30, min\_periods=1).sum())

new\_data['stock\_shortage'] = ((new\_data['stock\_on\_hand'] < new\_data['stock\_reorder\_level']) | (new\_data['stock\_on\_hand'] <= 0)).astype(int)

# Drop rows with missing values generated during lag or rolling window calculations

new\_data.dropna(inplace=True)

# Step 4: Select features for prediction

features = [

'store', 'item', 'supplier\_id', 'supplier\_lead\_time', 'stock\_on\_hand',

'stock\_reorder\_level', 'replenishment\_lead\_time', 'day\_of\_week', 'restocked\_store',

'restocked\_inventory', '7\_day\_avg\_sales', '30\_day\_avg\_sales', '90\_day\_avg\_sales',

'month', 'quarter', 'year', 'sales\_lag\_1', 'sales\_lag\_7', 'sales\_lag\_30',

'stock\_turnover\_ratio', 'days\_until\_stockout', 'days\_since\_last\_restock',

'stock\_to\_sales\_ratio', 'supplier\_lead\_time\_variance', 'is\_weekend',

'cumulative\_sales', 'reorder\_frequency', 'stock\_shortage'

]

# Ensure the new data contains all required features

X\_new = new\_data[features]

# Step 5: Make predictions using the trained model

new\_data['predicted\_sales'] = xgb\_regressor.predict(X\_new)

# Step 6: Save the results

new\_data.to\_csv('data\_2017\_with\_predictions.csv', index=False)

print("Prediction completed and saved to 'data\_2017\_with\_predictions.csv'.")

**Prediction Code**

# Import required libraries

import pandas as pd

import joblib

# Step 1: Load the trained model

xgb\_regressor = joblib.load('xgboost\_inventory\_model.joblib') # Load the saved XGBoost model

# Step 2: Load the new data for prediction

new\_data = pd.read\_csv('data\_2017.csv') # Replace with the file path to your new dataset

# Step 3: Convert 'date' column to datetime format

new\_data['date'] = pd.to\_datetime(new\_data['date'])

# Step 4: Generate required derived features for prediction

new\_data['7\_day\_avg\_sales'] = new\_data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=7, min\_periods=1).mean())

new\_data['30\_day\_avg\_sales'] = new\_data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=30, min\_periods=1).mean())

new\_data['90\_day\_avg\_sales'] = new\_data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=90, min\_periods=1).mean())

new\_data['month'] = new\_data['date'].dt.month

new\_data['quarter'] = new\_data['date'].dt.quarter

new\_data['year'] = new\_data['date'].dt.year

new\_data['sales\_lag\_1'] = new\_data.groupby(['store', 'item'])['sales'].shift(1)

new\_data['sales\_lag\_7'] = new\_data.groupby(['store', 'item'])['sales'].shift(7)

new\_data['sales\_lag\_30'] = new\_data.groupby(['store', 'item'])['sales'].shift(30)

new\_data['stock\_turnover\_ratio'] = new\_data['sales'] / (new\_data['stock\_on\_hand'] + 1e-5)

new\_data['days\_until\_stockout'] = new\_data['stock\_on\_hand'] / (new\_data['7\_day\_avg\_sales'] + 1e-5)

new\_data['days\_since\_last\_restock'] = (new\_data['date'] - new\_data.groupby(['store', 'item'])['date'].shift()).dt.days.fillna(0)

new\_data['stock\_to\_sales\_ratio'] = new\_data['stock\_on\_hand'] / (new\_data['sales'] + 1e-5)

new\_data['supplier\_lead\_time\_variance'] = new\_data.groupby('supplier\_id')['supplier\_lead\_time'].transform('std').fillna(0)

new\_data['is\_weekend'] = new\_data['date'].dt.weekday.apply(lambda x: 1 if x >= 5 else 0)

new\_data['cumulative\_sales'] = new\_data.groupby(['store', 'item', 'year', 'month'])['sales'].cumsum()

new\_data['reorder\_frequency'] = new\_data.groupby(['store', 'item'])['restocked\_inventory'].transform(lambda x: x.rolling(window=30, min\_periods=1).sum())

new\_data['stock\_shortage'] = ((new\_data['stock\_on\_hand'] < new\_data['stock\_reorder\_level']) | (new\_data['stock\_on\_hand'] <= 0)).astype(int)

# Step 5: Drop rows with missing values generated during lag or rolling operations

new\_data.dropna(inplace=True)

# Step 6: Select feature columns for prediction

features = [

'store', 'item', 'supplier\_id', 'supplier\_lead\_time', 'stock\_on\_hand',

'stock\_reorder\_level', 'replenishment\_lead\_time', 'day\_of\_week', 'restocked\_store',

'restocked\_inventory', '7\_day\_avg\_sales', '30\_day\_avg\_sales', '90\_day\_avg\_sales',

'month', 'quarter', 'year', 'sales\_lag\_1', 'sales\_lag\_7', 'sales\_lag\_30',

'stock\_turnover\_ratio', 'days\_until\_stockout', 'days\_since\_last\_restock',

'stock\_to\_sales\_ratio', 'supplier\_lead\_time\_variance', 'is\_weekend',

'cumulative\_sales', 'reorder\_frequency', 'stock\_shortage'

]

# Ensure the dataset contains all required features

X\_new = new\_data[features]

# Step 7: Make predictions using the trained model

new\_data['predicted\_sales'] = xgb\_regressor.predict(X\_new)

# Step 8: Save the results (original columns + predictions) to a new CSV file

new\_data[['date', 'store', 'item', 'sales', 'predicted\_sales']].to\_csv('data\_2017\_with\_predictions.csv', index=False)

print("Prediction completed and saved to 'data\_2017\_with\_predictions.csv'.")

Dual Model Implementation for Sales and Understock Prediction

import pandas as pd

import numpy as np

import joblib

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, accuracy\_score

import xgboost as xgb

# Load dataset

data = pd.read\_csv('data\_2013\_2016.csv')

data['date'] = pd.to\_datetime(data['date'])

# Feature Engineering

data['month'] = data['date'].dt.month

data['year'] = data['date'].dt.year

data['7\_day\_avg\_sales'] = data.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=7, min\_periods=1).mean())

data['sales\_lag\_1'] = data.groupby(['store', 'item'])['sales'].shift(1)

data['stock\_turnover\_ratio'] = data['sales'] / (data['stock\_on\_hand'] + 1e-5)

data['stock\_shortage'] = ((data['stock\_on\_hand'] < data['stock\_reorder\_level']) | (data['stock\_on\_hand'] <= 0)).astype(int)

data.dropna(inplace=True)

# Features and targets

features = ['store', 'item', 'month', 'year', '7\_day\_avg\_sales', 'sales\_lag\_1', 'stock\_turnover\_ratio']

X\_sales = data[features]

y\_sales = data['sales']

X\_understock = data[features]

y\_understock = data['stock\_shortage']

# Train-test split

X\_train\_sales, X\_test\_sales, y\_train\_sales, y\_test\_sales = train\_test\_split(X\_sales, y\_sales, test\_size=0.2, random\_state=42)

X\_train\_understock, X\_test\_understock, y\_train\_understock, y\_test\_understock = train\_test\_split(X\_understock, y\_understock, test\_size=0.2, random\_state=42)

# Sales Forecasting Model

sales\_model = xgb.XGBRegressor(objective='reg:squarederror', n\_estimators=100, max\_depth=10, learning\_rate=0.1)

sales\_model.fit(X\_train\_sales, y\_train\_sales)

joblib.dump(sales\_model, 'xgboost\_sales\_model.joblib')

sales\_predictions = sales\_model.predict(X\_test\_sales)

print(f"Sales Forecasting - RMSE: {np.sqrt(mean\_squared\_error(y\_test\_sales, sales\_predictions))}, MAE: {mean\_absolute\_error(y\_test\_sales, sales\_predictions)}")

# Understock Prediction Model

understock\_model = xgb.XGBClassifier(n\_estimators=100, max\_depth=10, learning\_rate=0.1)

understock\_model.fit(X\_train\_understock, y\_train\_understock)

joblib.dump(understock\_model, 'xgboost\_understock\_model.joblib')

understock\_predictions = understock\_model.predict(X\_test\_understock)

print(f"Understock Prediction - Accuracy: {accuracy\_score(y\_test\_understock, understock\_predictions)}")

Code for Sales and Understock Prediction with Visualization

import pandas as pd

import numpy as np

import joblib

import xgboost as xgb

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, accuracy\_score

# Load saved sales and understock models

sales\_model = joblib.load('xgboost\_sales\_model.joblib')

understock\_model = joblib.load('xgboost\_understock\_model.joblib')

# Load the new dataset

data\_2017 = pd.read\_csv('data\_2017.csv')

data\_2017['date'] = pd.to\_datetime(data\_2017['date'])

# Feature Engineering on new data

data\_2017['month'] = data\_2017['date'].dt.month

data\_2017['quarter'] = data\_2017['date'].dt.quarter

data\_2017['year'] = data\_2017['date'].dt.year

data\_2017['7\_day\_avg\_sales'] = data\_2017.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=7, min\_periods=1).mean())

data\_2017['30\_day\_avg\_sales'] = data\_2017.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=30, min\_periods=1).mean())

data\_2017['90\_day\_avg\_sales'] = data\_2017.groupby(['store', 'item'])['sales'].transform(lambda x: x.rolling(window=90, min\_periods=1).mean())

data\_2017['sales\_lag\_1'] = data\_2017.groupby(['store', 'item'])['sales'].shift(1)

data\_2017['sales\_lag\_7'] = data\_2017.groupby(['store', 'item'])['sales'].shift(7)

data\_2017['sales\_lag\_30'] = data\_2017.groupby(['store', 'item'])['sales'].shift(30)

data\_2017['stock\_turnover\_ratio'] = data\_2017['sales'] / (data\_2017['stock\_on\_hand'] + 1e-5)

data\_2017['days\_until\_stockout'] = data\_2017['stock\_on\_hand'] / (data\_2017['7\_day\_avg\_sales'] + 1e-5)

data\_2017['days\_since\_last\_restock'] = (data\_2017['date'] - data\_2017.groupby(['store', 'item'])['date'].shift()).dt.days.fillna(0)

data\_2017['stock\_to\_sales\_ratio'] = data\_2017['stock\_on\_hand'] / (data\_2017['sales'] + 1e-5)

data\_2017['supplier\_lead\_time\_variance'] = data\_2017.groupby('supplier\_id')['supplier\_lead\_time'].transform('std').fillna(0)

data\_2017['is\_weekend'] = data\_2017['day\_of\_week'].apply(lambda x: 1 if x >= 6 else 0)

data\_2017['cumulative\_sales'] = data\_2017.groupby(['store', 'item', 'year', 'month'])['sales'].cumsum()

data\_2017['reorder\_frequency'] = data\_2017.groupby(['store', 'item'])['restocked\_inventory'].transform(lambda x: x.rolling(window=30, min\_periods=1).sum())

data\_2017['stock\_shortage'] = ((data\_2017['stock\_on\_hand'] < data\_2017['stock\_reorder\_level']) | (data\_2017['stock\_on\_hand'] <= 0)).astype(int)

# Drop rows with missing values due to lag features

data\_2017.dropna(inplace=True)

# Define the features used for prediction

features = [

'store', 'item', 'supplier\_id', 'supplier\_lead\_time', 'stock\_on\_hand',

'stock\_reorder\_level', 'replenishment\_lead\_time', 'day\_of\_week', 'restocked\_store',

'restocked\_inventory', '7\_day\_avg\_sales', '30\_day\_avg\_sales', '90\_day\_avg\_sales',

'month', 'quarter', 'year', 'sales\_lag\_1', 'sales\_lag\_7', 'sales\_lag\_30',

'stock\_turnover\_ratio', 'days\_until\_stockout', 'days\_since\_last\_restock',

'stock\_to\_sales\_ratio', 'supplier\_lead\_time\_variance', 'is\_weekend',

'cumulative\_sales', 'reorder\_frequency', 'stock\_shortage'

]

# Predict Sales using the saved model

X\_2017 = data\_2017[features]

data\_2017['predicted\_sales'] = sales\_model.predict(X\_2017)

# Predict Understock Risk using the saved model

data\_2017['understock\_risk'] = understock\_model.predict(X\_2017)

# Save the new data with predictions to a CSV file

data\_2017[['date', 'store', 'item', 'sales', 'predicted\_sales', 'understock\_risk']].to\_csv('data\_2017\_with\_predictions.csv', index=False)

print("Predicted data saved as 'data\_2017\_with\_predictions.csv'.")

# Summary and Visualization

# Calculate RMSE and MAE for sales predictions

rmse = np.sqrt(mean\_squared\_error(data\_2017['sales'], data\_2017['predicted\_sales']))

mae = mean\_absolute\_error(data\_2017['sales'], data\_2017['predicted\_sales'])

print(f"Sales Prediction - RMSE: {rmse}, MAE: {mae}")

# Calculate Understock/Overstock Analysis

understock\_cases = data\_2017[data\_2017['understock\_risk'] == 1]

overstock\_cases = data\_2017[data\_2017['stock\_on\_hand'] > data\_2017['predicted\_sales']]

optimal\_reorder\_points = data\_2017.groupby(['store', 'item'])['stock\_reorder\_level'].mean()

print(f"Understock Cases Count: {understock\_cases.shape[0]}")

print(f"Overstock Cases Count: {overstock\_cases.shape[0]}")

print(f"Optimal Reorder Points by Product/Store Combination:\n{optimal\_reorder\_points}")

# Visualization

# Plot original vs. predicted sales

plt.figure(figsize=(12, 8))

plt.plot(data\_2017['date'], data\_2017['sales'], label='Original Sales', color='blue', alpha=0.6)

plt.plot(data\_2017['date'], data\_2017['predicted\_sales'], label='Predicted Sales', color='red', linestyle='--')

plt.title('Original vs. Predicted Sales Over Time')

plt.xlabel('Date')

plt.ylabel('Sales')

plt.legend()

plt.show()

# Understock vs Overstock analysis

plt.figure(figsize=(10, 6))

sns.countplot(data=data\_2017, x='understock\_risk', palette='coolwarm')

plt.title('Understock Risk Analysis')

plt.xlabel('Understock Risk (1 = Risk)')

plt.ylabel('Count')

plt.show()

# Scatter plot of Stock vs Predicted Sales

plt.figure(figsize=(10, 6))

sns.scatterplot(data=data\_2017, x='stock\_on\_hand', y='predicted\_sales', hue='understock\_risk', style='understock\_risk', palette='coolwarm')

plt.title('Stock on Hand vs. Predicted Sales')

plt.xlabel('Stock on Hand')

plt.ylabel('Predicted Sales')

plt.legend(title='Understock Risk')

plt.show()

# Optimal Reorder Point Visualization

plt.figure(figsize=(12, 8))

sns.barplot(data=data\_2017, x='item', y='stock\_reorder\_level', hue='store', ci=None)

plt.title('Optimal Reorder Points by Store

**CHAPTER 7**

**REFERENCE**

**1.** **Pandas Documentation**:  
Used for data manipulation and cleaning throughout the project.  
URL: https://pandas.pydata.org

2. **NumPy Documentation**:  
Utilized for numerical operations and array manipulations.  
URL: <https://numpy.org>

3. **XGBoost Documentation**:  
Reference for implementing and tuning the XGBoost regressor for demand forecasting.  
URL: <https://xgboost.readthedocs.io>

4. **Joblib Documentation**:  
Used for saving and loading the trained model.  
URL: <https://joblib.readthedocs.io>

5. **Scikit-learn Documentation**:  
Employed for dataset splitting and evaluation metrics such as RMSE and MAE.  
URL: <https://scikit-learn.org>

6. **IEEE Research Paper**:  
Natthamonkan Kheawpeam and Sukree Sinthupinyo. *"Demand Forecasting Using Machine Learning to Manage Product Inventory for Multi-channel Retailing Store."* IEEE, 2023.

7. **Python Official Documentation**:  
Comprehensive guide to Python libraries and datetime module.  
URL: <https://www.python.org/doc/>