A Major Project Report

On

SMART PRICING AND FORECASTING TOOL FOR SMALL VENDORS

Submitted to CMREC (UGC Autonomous)

In Partial Fulfilment of the requirements for the Award of Degree

of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING (AI&ML)

Submitted By

KANDUKURU AKASH	**(228R1A6666)**
KUMMARI JITHENDHAR	**(238R5A6610)**
GUGULOTHU JAGAN	**(228R1A6686)**
BELLAMKONDA RAGHU VARMA	**(228R1A6670)**

Under the Esteemed guidance of

Mr. K. ANIL

Assistant Professor, Department of CSE (AI & ML)

Department of Computer Science & Engineering (AI&ML)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi, Affiliated to JNTU, Hyderabad)

(Kandlakoya, Medchal Road, Medchal-Malkajgiri Dist. Hyderabad-501 401)

2025-2026

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Accredited by NAAC & NBA, Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad, Kandlakoya, Medchal Road, Hyderabad-501 401)

Department of Computer Science & Engineering (AI & ML)

CERTIFICATE

This is to certify that the project entitled **"SMART PRICING AND FORECASTING TOOL FOR SMALL VENDORS"** is a bonafide work carried out by

KANDUKURU AKASH	**(228R1A6666)**
KUMMARI JITHENDHAR	**(238R5A6610)**
GUGULOTHU JAGAN	**(228R1A6686)**
BELLAMKONDA RAGHU VARMA	**(228R1A6670)**

in partial fulfillment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING (AI&ML)** from CMR Engineering College, under our guidance and supervision.

The results presented in this project have been verified and are found to be satisfactory. The results embodied in this project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Coordinator**	**Head of the Department**
Mr. K. Anil	Mr. G. Venkateswarlu	Dr. Madhavi Pingili
Assistant Professor	Assistant Professor	Professor & HOD
Department of CSE (AI&ML)	Department of CSE (AI&ML)	Department of CSE (AI&ML)

External Examiner: _____

DECLARATION

This is to certify that the work reported in the present Major project entitled **"SMART PRICING AND FORECASTING TOOL FOR SMALL VENDORS"** is a record of bonafide work done by us in the Department of Computer Science and Engineering (AI&ML), CMR Engineering College. The reports are based on the project work done entirely by us and not copied from any other source. We submit our project for further development by any interested students who share similar interests to improve the project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

KANDUKURU AKASH (228R1A6666)
KUMMARI JITHENDHAR (238R5A6610)
GUGULOTHU JAGAN (228R1A6686)
BELLAMKONDA RAGHU VARMA (228R1A6670)

ACKNOWLEDGMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, HOD, **Department of CSE (AI & ML), CMR Engineering College** for their constant support.

We are extremely thankful to **Mr. K. Anil,** Assistant Professor, Internal Guide, Department of CSE (AI & ML), for his constant guidance, encouragement and moral support throughout the project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We thank **Mr. G. Venkateswarlu,** Major Project Coordinator for his constant support in carrying out the project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this project successfully in time.

Finally, We are very much thankful to our parents who guided us for every step.

KANDUKURU AKASH (228R1A6666)
KUMMARI JITHENDHAR (238R5A6610)
GUGULOTHU JAGAN (228R1A6686)
BELLAMKONDA RAGHU VARMA (228R1A6670)

CONTENTS

ABSTRACT

The Smart Pricing and Forecasting Tool for Small Vendors is an intelligent, data-driven system designed to assist small-scale businesses in making accurate, timely, and profitable pricing decisions using historical data and advanced machine learning techniques. In rapidly changing market conditions, small vendors often struggle to analyze large volumes of sales data and identify optimal pricing strategies that balance customer demand, competition, and profit margins. To address this challenge, the proposed system integrates statistical forecasting models such as Rolling Average and Moving Average with a state-of-the-art deep learning approach, Temporal Convolutional Networks (TCN), to provide reliable and precise future price predictions. The architecture begins with the collection of essential inputs, including sales data, product information, and historical prices, which are then preprocessed through cleaning, sorting, and time-series transformation. Forecasting models analyze patterns, trends, and fluctuations to generate multi-model predictions. Further, the system performs price analysis to suggest ideal price ranges while evaluating stability and market behavior. The results are presented through a rich output module offering forecast graphs, comparative model insights, and summary metrics that help vendors make informed business decisions. By combining statistical methods, deep learning, and actionable price analytics, this project provides a cost-effective and scalable solution that empowers small vendors to enhance profitability, reduce losses, and improve market competitiveness through automated smart pricing.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

1.1 Introduction

Small vendors often depend on intuition, personal experience, or simple handwritten sales records to make important decisions related to pricing and product demand. While this traditional approach may work to some extent in stable environments, it becomes highly unreliable when customer preferences fluctuate frequently or when competition increases in the local market. In today's fast-changing retail environment, customer buying patterns can vary due to seasonal changes, festival periods, market trends, and even external factors such as economic conditions. When vendors rely only on manual judgment without analyzing historical data, they face challenges such as stocking too much inventory, running out of popular items, or failing to adjust prices at the right time. These issues directly affect profit margins and customer satisfaction, making it difficult for small vendors to compete effectively.

A smart forecasting system can play a crucial role in helping small vendors overcome these limitations by offering a structured, data-driven approach. By analyzing previous sales records, identifying purchasing patterns, and understanding how demand changes over time, the system can generate reliable forecasts for upcoming days or weeks. This allows vendors to plan inventory more accurately, reduce wastage, avoid unnecessary storage costs, and ensure that the right products are available when customers need them. Such a system transforms raw sales data into meaningful insights that support better decision-making.

The "Smart Pricing and Forecasting Tool for Small Vendors" has been specifically developed to simplify this process and make forecasting accessible even to non-technical users. The tool uses simple yet effective forecasting techniques such as Rolling Average (RA), Moving Average (MA), and Temporal Convolutional Network (TCN) to estimate future product demand. RA and MA smooth out sales variations and capture short-term trends, while TCN helps identify deeper temporal patterns in the data, resulting in more accurate predictions. The system processes historical sales and pricing data, detects patterns such as peak demand days, seasonal fluctuations, or sudden drops in sales, and provides short-term demand forecasts that vendors can trust.

In addition to forecasting demand, the system also offers smart pricing suggestions, enabling vendors to set prices based on demand levels, previous price trends, and market behavior. This prevents underpricing or overpricing and ensures competitive pricing that maximizes profit. The tool presents all results through clean visual graphs, including actual sales trends, rolling averages, and forecast curves, making it easier for vendors to understand their business performance at a glance. Through automated analysis, real-time feedback, and user-friendly visualizations, the system empowers small vendors to move from guesswork to data-driven decision-making, ultimately improving stock planning, reducing losses, and optimizing pricing strategies for better business growth.

1.2 Project Objectives

Small vendors traditionally manage their daily sales operations manually, without the help of advanced analytical tools or automated decision-making systems. Unlike large enterprises that rely on sophisticated forecasting platforms, small vendors often lack the financial and technical resources required to access such technologies. As a result, their decisions regarding stock management, pricing, and demand prediction are mostly based on intuition or basic sales registers. This makes the business prone to errors such as overstocking, stock shortages, inconsistent pricing, and missed sales opportunities.

To address these challenges, this project introduces a simplified, vendor-friendly forecasting platform that converts raw historical sales data into useful and actionable insights. The platform integrates simple statistical methods along with a deep learning approach to provide short-term predictions that can support everyday decision-making. By using Rolling Average (RA), Moving Average (MA), and Temporal Convolutional Network (TCN), the system is capable of understanding past demand behavior and predicting future sales trends with improved accuracy. The use of these models ensures that the tool remains both effective and computationally lightweight, making it suitable for small-scale vendors.

The system is built with a user-friendly design in mind. It includes an easy-to-use front-end interface that allows users to select categories, products, and timeframes without any technical expertise. The backend forecasting engine processes the data and returns real-time insights, making the platform appropriate for small shops, local grocery stores, rural vendors, and market-based sellers. The system is intentionally kept simple to ensure accessibility while still offering meaningful analytical support.

The primary objective of the proposed system is to analyze historical sales data and identify meaningful patterns such as peak purchasing periods, seasonal changes, and fluctuations in customer demand. By studying past trends, the system aims to provide a strong foundation for reliable demand forecasting. Another objective is to generate short-term demand forecasts for upcoming days using techniques such as Rolling Average (RA), Moving Average (MA), and Temporal Convolutional Network (TCN). These models are applied to forecast demand for 7-day, 15-day, and 30-day intervals, enabling vendors to plan inventory according to their business needs. A further objective of the system is to recommend an optimal price range for the product based on previous pricing trends and the predicted demand levels. This helps vendors adjust their prices smartly and remain competitive in the local market.

The system also aims to present the forecast outputs through clear visualizations such as trend graphs, rolling-average plots, and forecast curves. These visuals help users understand the data more easily, even if they have no technical background. Finally, the system aims to support small vendors in shifting from guesswork to data-driven decision-making. By providing accurate demand estimates and pricing insights, the tool helps improve inventory planning, reduce losses, and enable better pricing strategies for sustainable business growth.

1.3 Purpose Of The Project

The purpose of this project is to develop a simple, efficient, and affordable forecasting tool specifically designed for small vendors who typically lack access to advanced data analytics and prediction systems. Many existing commercial forecasting platforms are either too expensive, too complex, or designed for large enterprises with dedicated technical teams. As a result, micro-businesses and small vendors remain dependent on manual judgment, basic registers, or intuition-based decision-making. This project aims to bridge this gap by delivering an easy-to-use platform that provides reliable demand and pricing forecasts without requiring any technical expertise from the user.

One of the core purposes of this system is to enhance decision-making for small vendors by offering accurate and timely demand predictions. By analyzing historical sales patterns, the tool helps users determine the right quantity of stock to purchase, when to restock, and how much inventory is necessary for upcoming days or weeks. This data-driven support helps vendors avoid risky assumptions about customer demand.

Another major purpose is to reduce financial losses caused by stock mismanagement. Overstocking, especially for perishable or fast-moving items, leads to wastage and reduced profit margins. On the other hand, understocking results in missed sales opportunities and dissatisfied customers. The forecasting system minimizes both risks by suggesting balanced stocking levels based on real sales behavior.

The project also aims to improve vendor profitability through smarter pricing decisions. By examining past pricing data and sales trends, the system suggests an optimal price range for each product. Vendors can adjust their selling price according to market demand, seasonal changes, and peak periods, ensuring better revenue and improved competitiveness. This empowers small vendors to use the same data-driven pricing strategies often used by larger businesses.

Another important purpose is to simplify the interpretation of sales data. Many small vendors do not have the expertise to analyze charts or complex datasets. This system converts raw numbers into visually clear graphs and easy-to-understand insights. Line charts, trend plots, and summary visuals help users understand demand cycles, peak sales days, and future forecasts without difficulty.

Finally, the project aims to automate manual forecasting efforts, reducing the burden on vendors who previously calculated averages or monitored sales manually. The system processes the uploaded data automatically and generates results instantly, saving time and effort. This automation makes forecasting faster, more accurate, and accessible for even the smallest shops.

1.4 Problem Statement

Small vendors and local businesses often depend on manual record-keeping, experience, and guesswork to estimate product demand and set selling prices. This approach lacks accuracy and consistency, especially in dynamic market conditions where customer demand changes frequently due to factors such as seasonality, trends, and purchasing behavior. As a result, vendors commonly face issues like overstocking, which leads to wastage and financial loss, or understocking, which results in missed sales opportunities and customer dissatisfaction.

Moreover, small vendors usually do not have access to advanced forecasting tools or data analytics platforms used by large enterprises, as these systems are often expensive, complex, and require technical expertise. Even when sales data is available, vendors are unable to effectively analyze it to extract meaningful insights or predict future demand. This lack of data-driven decision-making limits their ability to plan inventory, optimize pricing, and improve profitability.

Therefore, there is a strong need for a simple, affordable, and user-friendly system that can analyze historical sales data, generate accurate demand forecasts, and provide pricing recommendations. Such a solution should present results in an easy-to-understand format, enabling small vendors to make informed decisions, reduce losses, and improve overall business efficiency without requiring technical knowledge.

1.5 Existing System With Disadvantages

In the current scenario, most small vendors, street sellers, and micro-business owners rely heavily on traditional and manual methods for managing their daily sales and predicting future demand. These methods generally include writing sales numbers in notebooks, maintaining simple registers, or keeping data loosely in spreadsheets without any analytical processing. Because of the lack of proper tools, decisions regarding stock purchases, pricing, and demand estimation are mostly based on experience, assumptions, and guesswork rather than data-driven insights. While this approach may work for very small volumes, it becomes unreliable and inconsistent when the business grows or when market conditions fluctuate.

Another limitation of the existing system is that it does not support long-term planning. Vendors cannot easily identify seasonal patterns, peak sales hours, frequently purchased items, or low-demand periods. As a result, they often end up purchasing stock either in excess or in insufficient quantities. Overstocking creates financial loss because unsold goods—especially perishable food items—get damaged or wasted. Understocking, on the other hand, leads to missed opportunities, customer dissatisfaction, and loss of profits. Since there is no predictive model, the vendor has no clear way to adjust pricing according to demand or competition.

Additionally, most of the available forecasting applications in the market are too expensive, too complex, or built for large-scale enterprises, which makes them unsuitable for small vendors. These tools often require advanced computer skills, license payments, or large datasets, which small businesses do not have. Many sellers also face challenges such as lack of internet connectivity, limited access to smartphones or computers, and no proper understanding of analytics dashboards. Because of these barriers, small vendors continue using outdated manual systems even though they are time-consuming and error-prone.

Overall disadvantages of the existing system include:

- Heavy dependence on manual calculations and human memory
- No accurate way to forecast future demand and plan stock
- High chances of errors, miscalculations, or missing data
- No automated insights, visualizations, or trend identification
- Difficulty in tracking daily, weekly, or monthly sales patterns
- Significant stock wastage due to over-purchasing
- Loss of customers and revenue due to understocking
- No data-driven pricing strategy
- Current digital tools are expensive, complex, or not vendor-friendly

1.6 Proposed System With Advantages

The proposed The proposed system introduces a smart, lightweight, and user-friendly sales forecasting application specifically designed for small vendors and micro-business owners. Unlike traditional market tools that require complex setup or technical skills, this solution focuses on simplicity and practicality. The system uses machine learning algorithms to automatically analyze historical sales data, identify hidden patterns, and generate highly accurate demand predictions for upcoming days, weeks, or seasons. By reducing the dependency on manual calculations and guesswork, the proposed system empowers vendors to make informed business decisions with confidence.

One of the core strengths of the proposed system is its ability to automatically process sales data and present results in a clean, easy-to-understand visual format. The platform converts raw numbers into graphs, charts, and summary insights, enabling even non-technical users to understand customer buying trends instantly. The dashboard provides essential information such as peak selling times, fast-moving products, slow-moving products, and expected demand levels. This helps vendors plan their stock purchases more effectively, reducing wastage and ensuring that sufficient inventory is available during high-demand periods. With better forecasting, vendors can prevent both overstocking and understocking, thus maintaining a healthy balance between supply and demand.

The system also incorporates features that assist vendors in optimizing their pricing strategies. By

understanding forecasted demand, the vendor can increase prices during high-demand seasons or offer discounts when demand is predicted to be low. Such dynamic pricing helps improve profitability while maintaining customer satisfaction. Additionally, the system reduces operational effort by automating routine tasks such as generating reports, calculating averages, and storing sales records. Vendors only need to input their daily sales, and the system handles the rest automatically. This significantly minimizes manual workload and reduces the chances of errors that are common in handwritten records.

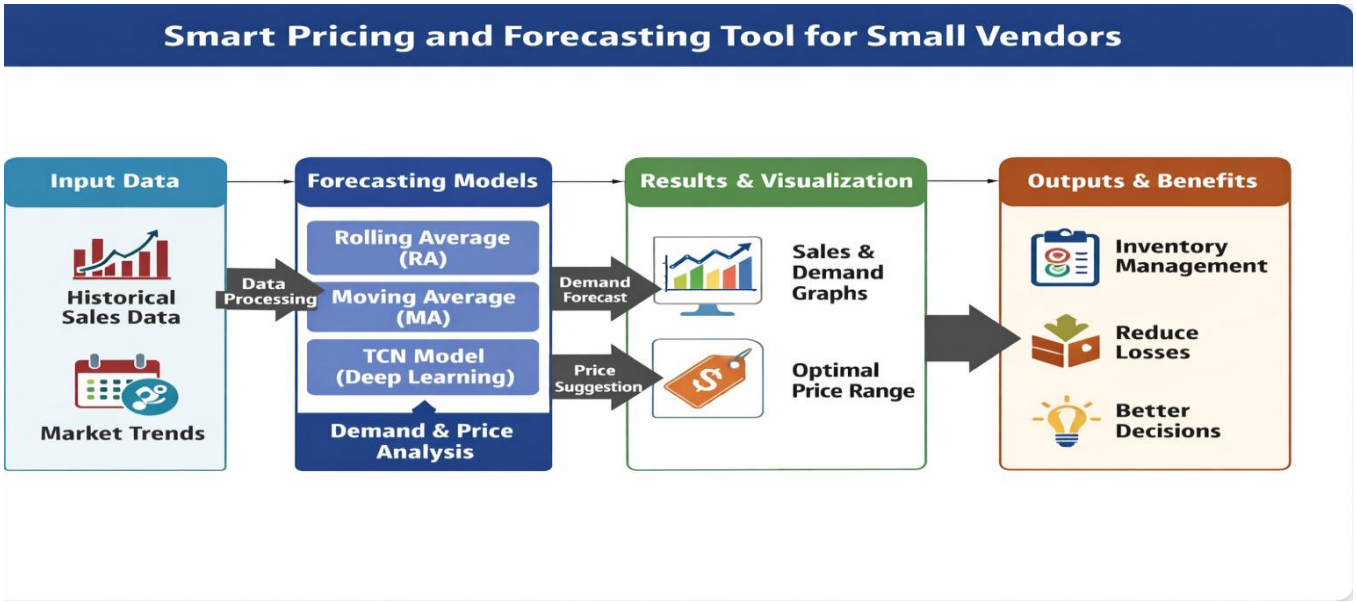

Fig. 1.5.1 Block diagram of proposed system

Another major feature of the proposed system is its high accessibility and affordability. It does not require powerful hardware, complicated installations, or high bandwidth. The application can run on basic smartphones or computers, making it accessible to small vendors with limited resources. Its interface is designed to be intuitive, with clear buttons, simple menus, and guided instructions so that even first-time users can easily operate it. Unlike enterprise software that requires monthly fees or licenses, this system offers a cost-effective solution that fits within the financial limits of small business owners. This makes it a practical tool not only for retail vendors but also for street food sellers, grocery shops, fruit and vegetable vendors, small kirana stores, and other micro-entrepreneurs. Furthermore, the proposed system provides secure data storage and backup, ensuring that the vendor's historical sales records are never lost. With cloud-based storage support, users can access their data anytime and from anywhere. The system also

supports exporting reports in different formats, allowing vendors to share or print their monthly performance summaries. These features enhance transparency and help users track their business growth over time. In the future, the system can also be expanded with advanced modules such as customer behavior analysis, sales recommendations, inventory alerts, and automated notifications.

Advantages:

• Supports better pricing decisions to increase profit

• Improves demand prediction accuracy using data-driven methods

• Helps reduce losses by avoiding overstocking and stock shortages

1.7 Input and Output Design

1.7.1 Input Design

The input module defines the structure, format, and flow of sales data processed by the Smart Pricing and Forecasting Tool for Small Vendors. Since the system operates on historical sales records, the input layer is designed to handle structured datasets containing product details, dates, quantities sold, and pricing information. The system accepts data in CSV format, which may include variations such as missing values, inconsistent date formats, or irregular entries.

To ensure consistency and accuracy, the input pipeline performs several preprocessing steps before the data is used for forecasting. These steps include data cleaning, handling missing values, converting date formats into standard time-series structure, and aggregating daily sales data. The system transforms raw input into a structured format suitable for statistical and deep learning models such as Rolling Average, Moving Average, and TCN.

The input module supports both existing historical datasets for model training and new incoming sales data for prediction. By defining clear input formats and preprocessing rules, the system ensures reliable data flow, reduces inconsistencies, and improves the accuracy of forecasting models.

Objectives

• The input pipeline removes inconsistencies and prepares clean data for forecasting models.

• The system supports continuous data updates, allowing new sales records to be added for improved prediction accuracy.

• A structured format is defined for capturing historical sales data, ensuring all required fields such as date, product, quantity, and price are properly maintained.

1.7.2 Output Design

The output module defines the structure, format, and presentation of the results generated by the forecasting system. Since the system performs demand prediction and pricing analysis, the outputs are designed to be clear, interpretable, and useful for decision-making by small vendors. The primary output includes predicted demand values generated using Rolling Average, Moving Average, and TCN models. In addition to demand forecasting, the system provides a suggested price range based on historical pricing trends and demand behavior. The outputs are presented in both numerical and visual formats to improve usability.

To enhance understanding, the system displays results through structured dashboards that include summary metrics such as total units sold, average daily sales, and average price. It also generates separate graphical visualizations for actual sales trends, rolling averages, moving average forecasts, and TCN predictions. These graphs help vendors easily interpret patterns and make informed decisions. The system ensures that outputs are consistent, easy to read, and visually intuitive, allowing even non-technical users to understand forecasting results without difficulty. By providing both textual and graphical outputs, the system supports effective decision-making and better business planning.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

[1] Kong et al., 2026 – "Deep Learning for Time Series Forecasting: A Survey"

Kong et al. (2026) presented a comprehensive review of modern deep learning approaches for time-series forecasting, including architectures such as Temporal Convolutional Networks (TCN). Their survey summarizes various models, benchmark datasets, and performance evaluations across multiple domains. The study highlights the effectiveness of deep learning techniques in capturing complex temporal patterns and emphasizes the importance of selecting appropriate architectures based on application requirements.

[2] Kiram et al., 2026 – "A Comparative Study of Temporal Convolutional Network and Gated Recurrent Unit for Predicting Ethereum Prices"

Kiram et al. (2026) conducted a comparative analysis between Temporal Convolutional Networks (TCN) and Gated Recurrent Units (GRU) for cryptocurrency price prediction. Their findings demonstrate that TCN models outperform GRU in capturing long-term dependencies in Ethereum price data. The study concludes that TCN is a more suitable choice for financial time-series forecasting due to its stability and parallel processing capability.

[3] Huang et al., 2026 – "TSCND: Temporal Subsequence-Based Convolutional Network with Difference for Time Series Forecasting"

Huang et al. (2026) proposed a novel Temporal Convolutional Network variant called TSCND, which enhances feature extraction by focusing on recent subsequences and incorporating difference operations. Their model improves forecasting accuracy by effectively capturing short-term variations while preserving temporal dependencies. The results show significant performance gains compared to standard TCN models.

[4] Anonymous, 2025 – "A Novel Dual-Channel Temporal Convolutional Network for Photovoltaic Power Forecasting"

This study (2025) introduced a dual-channel Temporal Convolutional Network architecture integrated with attention mechanisms for photovoltaic power forecasting. The model processes data through parallel channels to capture diverse temporal features and applies attention to enhance important patterns. The results indicate improved prediction accuracy compared to traditional TCN approaches.

[5] Chen & Liu, 2025 – "Short-Term Demand Forecasting Using Machine Learning Techniques"
Chen and Liu (2025) explored various machine learning techniques for short-term demand forecasting. Their study evaluates multiple models and demonstrates how machine learning methods can effectively handle complex demand patterns. The findings highlight the importance of feature engineering and model selection in improving forecasting performance.

[6] Zhang et al., 2025 – "Improving Time Series Forecasting Using Attention-Based DeepLearning Models"
Zhang et al. (2025) investigated the use of attention mechanisms in deep learning models for time-series forecasting. Their work shows that attention-based architectures can better capture relevant temporal dependencies by focusing on important time steps. The study reports improved forecasting accuracy and interpretability compared to traditional deep learning models.

[7] Patel & Sharma, 2025 – "Sales Forecasting Using Random Forest Algorithm in Retail Industry"
Patel and Sharma (2025) applied the Random Forest algorithm to sales forecasting in the retail sector. Their study demonstrates that ensemble learning methods can effectively model non-linear relationships in sales data. The results indicate that Random Forest provides reliable and accurate predictions, making it suitable for retail demand forecasting.

[8] Zhang et al., 2025 – "Advance Prediction of Coastal Groundwater Levels with Temporal Convolutional and Long Short-Term Memory Networks"
Zhang et al. (2025) proposed a hybrid model combining Temporal Convolutional Networks (TCN) and Long Short-Term Memory (LSTM) networks for groundwater level prediction. Their approach leverages TCN for short-term feature extraction and LSTM for long-term dependency modeling. The study shows that the hybrid model significantly improves forecasting accuracy in hydrological applications.

[9] Reddy & Kumar, 2024 – "Product Demand Prediction Using Support Vector Machines"
Reddy and Kumar (2024) utilized Support Vector Machines (SVM) for product demand prediction. Their study highlights the effectiveness of SVM in handling high-dimensional data and capturing complex patterns. The results demonstrate that SVM can be a reliable method for demand forecasting when properly tuned.

[10] Anonymous, 2024 – "A Hybrid Temporal Convolutional Network and Prophet Model for Power Load Forecasting"

This study (2024) proposed a hybrid forecasting approach combining Temporal Convolutional Networks (TCN) with the Prophet model. The integration leverages TCN's ability to capture temporal dependencies and Prophet's strength in handling seasonality and trends. The results show enhanced accuracy and robustness compared to using either model independently.

S. No	Author & Year	Method / Model	Key Idea	Contribution
1	Kong et al., 2026	DL (TCN, CNN, RNN)	Survey of DL models	Comprehensive review; DL captures complex temporal patterns effectively
2	Kiram et al., 2026	TCN vs GRU	Model comparison	TCN outperforms GRU in long-term dependency modelling for crypto prediction
3	Huang et al., 2026	TSCND (TCN variant)	Feature extraction improvement	Focuses on recent Sub sequences; improves prediction accuracy
4	Anonymous, 2025	Dual-channel TCN + Attention	Multi-channel learning	Enhances feature learning; improves photovoltaic forecasting accuracy
5	Chen & Liu, 2025	ML Techniques	Demand forecasting	ML models effectively capture demand patterns; importance of feature selection

S. No	Author & Year	Method / Model	Key Idea	Contribution
6	Zhang et al., 2025	Attention-based DL	Temporal attention	Improves accuracyby focusingonimportanttime steps
7	Patel & Sharma, 2025	Random Forest	Ensemble learning	Handles non-linear data; improves prediction accuracy in retail sales
8	Zhang et al., 2025	TCN + LSTM (Hybrid)	Hybrid modeling	Captures both short- and long-term dependencies effectively
9	Reddy & Kumar, 2024	SVM	Regression & classification	Works well for complex relationships; requires parameter tuning
10	Anonymous, 2024	TCN + Prophet	Hybrid forecasting	Improves robustness and accuracy by combining trend + temporal modeling

Table no. 2.1 Literature Survey

CHAPTER 3

SOFTWARE REQUIEMENTS ANALYSIS

3. SOFTWARE REQUIREMNET ANALYSIS

Software Requirement Analysis is a crucial phase in the development process, as it defines the functional, non-functional, and operational expectations of the proposed system. For small vendors, the forecasting system must be simple, fast, and reliable while supporting core functionalities such as data input, prediction generation, graph visualization, and reporting. This chapter outlines the major modules of the system and explains how each module contributes to the overall workflow. It also presents the feasibility analysis to evaluate whether the system is practical, usable, and beneficial within the constraints of resources, technology, cost, and time.

3.1 Modules And Their Functionalities

The proposed forecasting tool has been designed by dividing the entire system into modular components to ensure easier development, testing, and maintenance. Each module serves a specific purpose and collectively supports the complete forecasting and pricing functionalities required by small vendors.

1. Data Input and Upload Module

This module allows users to enter their daily sales and pricing values. Vendors can input data manually or upload simple CSV files containing historical sales records. The module validates the data to ensure correct formatting, numeric values, and consistency. Incorrect entries are flagged, helping users avoid data errors.

2. Data Preprocessing Module

Once the raw data is uploaded, the preprocessing module cleans and organizes it for forecasting. Tasks include handling missing values, removing outliers, normalizing data, generating time-based features, and converting data into a forecasting-ready structure. This ensures the accuracy of prediction models and improves the quality of insights.

3. Forecasting Models Module (RA, MA, TCN)

This module contains the core forecasting logic. It supports:

Rolling Average (RA) for simple pattern detection

Moving Average (MA) for trend analysis

Temporal Convolutional Network (TCN) for deep learning-based forecasting

The module automatically selects parameters, executes model calculations, and generates 7-day, 15-day, and 30-day demand predictions. The results are generated quickly and displayed in an organized format.

4. Pricing Recommendation Module

Based on historical trends and predicted demand, this module suggests suitable price ranges. When demand is expected to rise, the system recommends a higher price range; during low-demand periods, it suggests a competitive or discounted price. This helps vendors maximise profit and maintain stock flow.

5. Visualization and Insights Module

The visual module converts raw numerical outputs into easy-to-understand graphical representations. It displays line charts, bar graphs, trend curves, and summary insights. Users can visually compare past performance with forecasted values and understand buying patterns without needing technical knowledge.

6. Reporting and Export Module

This module allows users to generate downloadable reports containing forecasts, pricing suggestions, and statistical summaries. Reports can be exported as PDF, CSV, or image files. This is especially helpful for vendors who want to maintain monthly records or share business updates.

7. Authentication and User Management Module

This module ensures secure login and personalized dashboards. Users can create accounts, store data securely, and access their history from any device.

8. System Settings and Notifications Module

Users can customize settings such as prediction duration, currency units, notification alerts, and data refresh schedules. The module also sends reminders for daily data entry and alerts during forecasted

high-demand periods.

3.2 Functional Requirements

Functional requirements define what the system should do and the services it must provide to users. The Smart Pricing and Forecasting Tool for Small Vendors includes the following functionalities:
The system allows users to select product categories and specific products for analysis. It accepts historical sales data and processes it to generate demand forecasts for different time periods such as 7, 15, and 30 days. The system performs data preprocessing including cleaning, formatting, and organizing time-series data before applying forecasting techniques.

The system generates demand predictions using Rolling Average, Moving Average, and Temporal Convolutional Network (TCN) models. It also provides price suggestions based on past pricing trends and demand patterns. The results are displayed in both numerical and graphical formats, including sales trends, forecast graphs, and summary metrics.

The system enables dynamic interaction through the web interface, where users can request forecasts and instantly view results. It also generates and displays separate graphs such as actual sales trend, rolling average, moving average forecast, and TCN forecast. All outputs are updated in real-time based on user input.

3.3 Non-Functional Requirements

Non-functional requirements define how the system performs and the quality attributes it must satisfy. These requirements ensure that the system is reliable, efficient, and user-friendly. The system must provide fast response time, generating forecasts and displaying results within a few seconds to support real-time decision-making. It should be scalable, allowing new sales data to be added continuously without affecting performance. The system must ensure accuracy and consistency in predictions by properly handling input data and model execution.
The user interface should be simple, intuitive, and easy to use, even for non-technical users such as small vendors. The system must be reliable and operate without failures under normal usage conditions. It

should also be portable, meaning it can run on different devices such as laptops, desktops, and mobile browsers. Additionally, the system must maintain data integrity by correctly processing and storing input data. It should be maintainable and extensible, allowing future enhancements such as adding new forecasting models or additional features without major changes to the system architecture.

3.4 Feasibility Study

A feasibility study evaluates whether the proposed system is realistic, beneficial, and executable within available resources. It examines technical, economic, operational, time-based, and social feasibility. This ensures that the system is practical for small vendors who have limited financial and technological resources.

Technical Feasibility

The system uses lightweight statistical and machine learning models such as Rolling Average, Moving Average, and TCN, which require minimal computing power. The application can run on low-end laptops, mobile devices, and basic web browsers without needing high-performance hardware. The technologies used are stable, well-supported, and easy to integrate, ensuring long-term sustainability. Since vendors only need to provide simple sales data, the technical burden is very low. Thus, the system is technically feasible.

Economic Feasibility

The proposed solution is designed to be cost-effective. Unlike enterprise forecasting platforms that require subscriptions or licenses, this tool runs with minimal expenses. The development uses open-source libraries and low-cost deployment platforms. For vendors, the system eliminates manual record-keeping, reduces stock wastage, and improves profitability through better planning and pricing. The return on investment (ROI) is high because even small improvements in demand prediction result in increased sales and reduced losses. Therefore, the system is economically feasible.

Operational Feasibility

The user interface is simple, intuitive, and suitable for non-technical users. Small vendors, who often lack formal technical training, can easily navigate the system with minimal guidance. Daily operations such

as entering data, viewing predictions, and reading graphs are automated and require minimal manual effort. The forecasting output is provided in a clear, visual manner, enabling better decision-making. Since the system fits the working style and needs of small vendors, operational feasibility is strong.

Time Feasibility

The development timeline of this system is realistic and efficient. Each module can be implemented and tested individually, reducing delays. Forecasting models generate results within seconds, allowing real-time usage. The system is designed to provide immediate value to vendors without requiring long setup periods. The modular structure ensures quick updates and easy enhancements. Thus, the project is highly feasible in terms of time.

Legal and Social Feasibility

The system does not collect sensitive personal information and follows basic data privacy practices. It only stores sales figures and vendor account details, which makes it socially acceptable and legally safe.

CHAPTER 4

SOFTWARE AND HARDWARE REQUIREMENTS

4. SOFTWARE AND HARDWARE REQUIREMENTS

Every software project requires a well-defined set of resources to ensure efficient development, smooth deployment, and reliable operation. For a forecasting tool designed specifically for small vendors, the system must be lightweight yet powerful enough to process sales data, run forecasting models, generate graphs, and display results in real time. This chapter describes the essential software and hardware requirements needed to implement and operate the proposed "Smart Pricing and Forecasting Tool for Small Vendors." The requirements listed here ensure the system performs efficiently while remaining cost-effective and easily accessible even to users with limited technical infrastructure.

4.1 Software Requirements

The software requirements describe all the tools, technologies, frameworks, and platforms used for both backend processing and frontend interface operations. These tools are chosen based on availability, scalability, ease of use, and compatibility with simple devices used by small-scale vendors.

1. Operating System

The project can run on commonly used operating systems such as:

- **Windows 10 or above**
- **Linux distributions (Ubuntu preferred)**
- **macOS Monterey or above**

These platforms support the required dependencies and allow easy deployment of Python-based backend services, making them suitable for both development and usage.

2. Programming Languages

Python 3.10+ (Primary backend language)

Used for implementing forecasting models (RA, MA, TCN), preprocessing, evaluation, and graph generation. Python offers strong libraries for machine learning, numerical computation, and visualization.

JavaScript(Frontendscripting)

Used to handle UI interactions, fetch API results, update charts, and improve responsiveness.

3. Backend Frameworks and Libraries

Flask / FastAPI

Lightweight Python web frameworks used for building REST APIs that process user data and return prediction results.

NumPy & Pandas

For data handling, cleaning, preprocessing, and generating statistical outputs.

Matplotlib / Plotly

Graphing libraries used for generating line plots, comparison charts, and forecast visualizations.

TensorFlow / PyTorch

Deep learning frameworks required for implementing the TCN model and managing trained model weights.

4. Frontend Frameworks and Tools

HTML5, CSS3

Used for structuring and styling the user interface in a simple and visually clear manner.

Bootstrap / TailwindCSS

Makes the frontend responsive and suitable for mobile devices used by vendors.

JavaScript Libraries (Chart.js / Plotly.js)

Used to display interactive graphs, forecast charts, and summary visuals directly on the webpage.

5. Database Requirements

For storage of user details, login data, and uploaded sales files:

SQLite (lightweight option, ideal for small vendors)
or

MySQL / PostgreSQL (for scalable deployment)

The database stores only non-sensitive business data ensuring privacy and usability.

6. Additional Tools and Platforms

Jupyter Notebook
Used during development for testing and verifying forecasting models.

Git / GitHub
For version control and team collaboration.

XAMPP / WAMP (optional)
Can be used for local hosting and backend testing during development.

Cloud Deployment (optional)
If vendors need online access:

- AWS / Azure / Google Cloud for API hosting
- Firebase for real-time database if required

7. Browser Requirements

The system runs fully in a browser environment, supporting:

- Google Chrome

- Mozilla Firefox
- Microsoft Edge

Ensuring maximum accessibility for small vendors with simple devices.

4.2 Hardware Requirements

The hardware requirement section focuses on both development-level and end-user-level devices. Since the goal is to support small vendors, the system must be functional even on minimal hardware.

1. For Developers (During System Development)

Developers require slightly higher system specifications to train forecasting models and handle computational tasks.

Minimum Development System Requirements:

- **Processor:** Intel Core i5 (8th generation or above) / AMD Ryzen 5
- **RAM:** 8 GB (16 GB recommended for deep learning models)
- **Storage:** 256 GB SSD (for faster data access and model training)
- **Graphics:** Integrated GPU is sufficient (Dedicated GPU optional for faster TCN training)
- **Display:** 15-inch or larger monitor for coding and visualization
- **Internet:** Stable connection for dependency installation and cloud integration

2. For End Users (Small Vendors)

The system is intentionally designed to run on low-spec hardware so that small vendors can access forecasting without expensive equipment.

Minimum User System Requirements:

- **Processor:** Any dual-core processor (Intel Pentium / Core i3 / mobile processor)

- **RAM:** 4 GB
- **Storage:** 2 GB free space for local files
- **Device Type:**
  - Desktop computer
  - Laptop
  - Tablet
  - Android mobile phone with browser access
- **Operating System:** Android / Windows / Linux
- **Internet Connectivity:** Basic mobile data connection to load the web app
- **Browser:** Chrome or any modern browser supporting JavaScript

Since the system uses lightweight models like RA and MA and pre-trained TCN models run on the backend, the user device only receives final predictions. This makes the system extremely efficient even on older or low-budget devices.

3. Additional Optional Hardware for Vendors

- **Thermal printers** (for printing daily sales summaries)
- **Barcode scanners** (if vendors maintain inventory systems)
- **External drives** for backup of sales data
- **Smartphones or Tablets** for portable usage

These are not mandatory but can improve usability and convenience.

CHAPTER 5

SYSTEM ARCHITECTURE

5. SYSTEM ARCHITECTURE

5.1 System Architecture

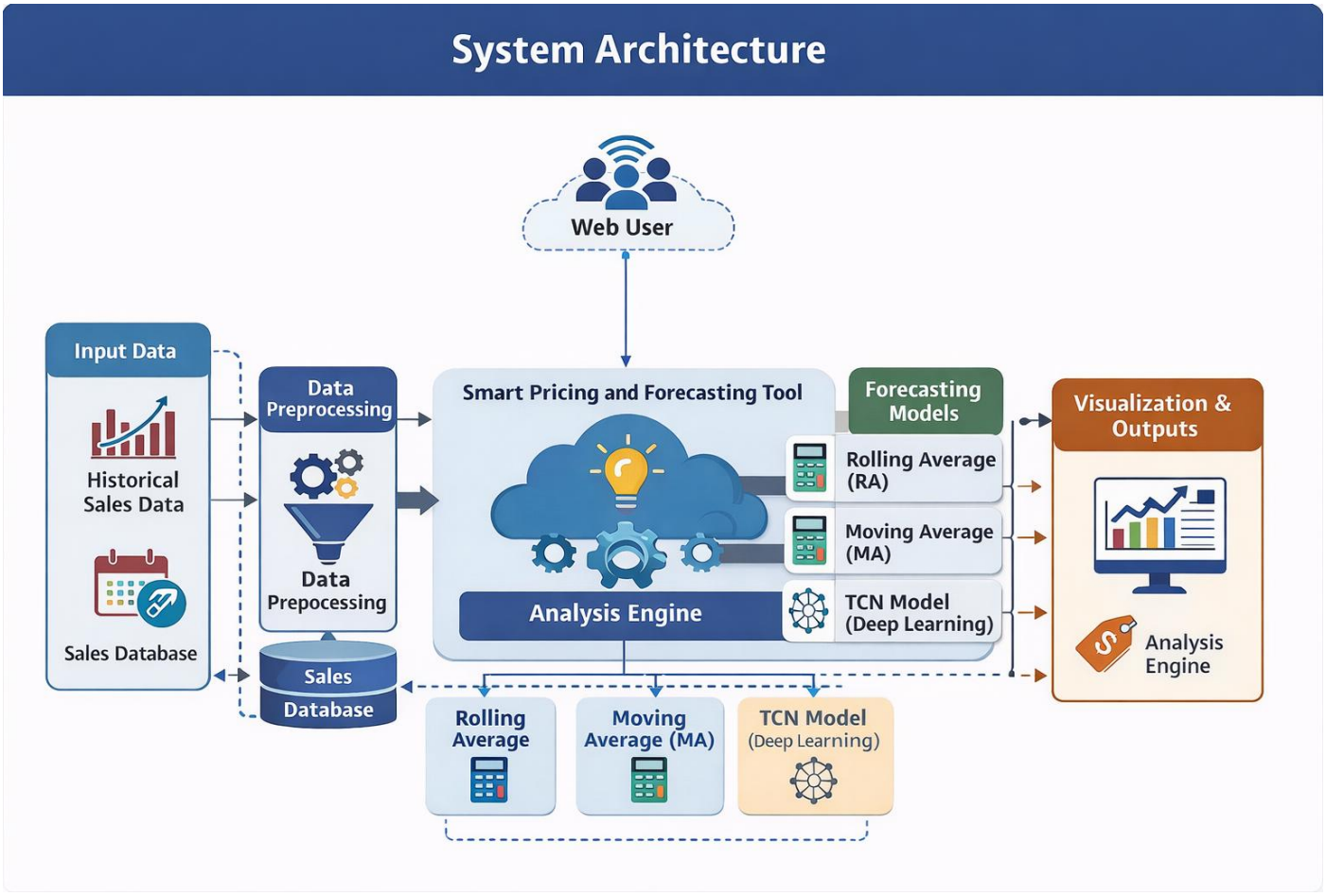

Fig. 5.1.1 System Architecture

The architecture of the proposed system is developed with a modular, layered approach to ensure smooth data flow from user input to final forecasting output. The system is divided into four primary layers:

1. User Interface Layer (Frontend)

This layer provides a simple and interactive interface through which vendors upload sales files, view generated forecasts, track sales trends, and observe price suggestions.

Key functions include:

- File upload (CSV/Excel sales data)
- Graph visualization (Actual trend, Rolling Average, Moving Average, TCN Forecast)
- Display of forecast results
- Display of price range insights
- Summary statistics

Technologies used: HTML, CSS, JavaScript, Chart.js, Bootstrap/Tailwind.

2. Application Logic Layer (Backend API)

This layer contains all computational logic and acts as a bridge between the interface and the forecasting models.

Core functionalities:

- Receive and validate input data
- Preprocess sales data
- Generate RA, MA, and TCN forecasts
- Calculate rolling averages
- Estimate pricing insights
- Return processed results to the frontend

Framework used: Flask / FastAPI (Python).

3. Forecasting & Analytics Engine

This is the heart of the system where mathematical and deep learning techniques are applied.

Components include:

- **Rolling Average Module (RA)** → Smooths sales fluctuations
- **Moving Average Module (MA)** → Predicts upcoming demand based on historical windows
- **TCN Model** → Provides accurate deep learning forecasts for 7/15/30 days
- **Pricing Insight Module** → Suggests price range using historical price behavior

The forecasting engine outputs structured results that are passed back to the API layer.

4. Data Storage Layer

Stores all required data:

- Historical sales and price records
- Uploaded CSV/Excel files
- Intermediate preprocessed datasets
- Forecast outputs and graphs

Database: SQLite/MySQL (depending on deployment).

File Storage: Local directory for graphs and uploaded files.

Architectural Flow

1. User uploads sales data.
2. Backend preprocesses data and extracts dates, quantities, and prices.
3. RA & MA models compute statistical forecasts.
4. TCN model generates deep learning forecasts.
5. Pricing module analyzes price stability and trends.
6. System generates multiple graphs (Actual, Rolling Avg, MA Forecast, TCN Forecast).
7. Frontend displays results, graphs, and recommendations.

This architecture ensures a smooth flow from input to output, enabling small vendors to access demand forecasting easily.

5.2 Data Flow Diagrams

The Data Flow Diagram (DFD) of the *Smart Pricing and Forecasting Tool for Small Vendors* illustrates how data moves through different stages of the system to generate demand forecasts and pricing insights. The process begins with the user, who provides input in the form of historical sales data and selects the required product and timeframe. This data is then passed to the data collection and preprocessing module, where it is cleaned, structured, and converted into a proper time-series format by handling missing values and organizing daily sales records.

After preprocessing, the processed data is forwarded to the forecasting module, which includes Rolling Average (RA), Moving Average (MA), and Temporal Convolutional Network (TCN) models. These models analyze the data and generate demand predictions for the selected time period. At the same time, the pricing analysis module evaluates historical price trends and calculates a suggested price range based on demand behavior.

The outputs from these modules are then sent to the visualization layer, where the system generates graphs such as actual sales trends, rolling averages, moving average forecasts, and TCN forecasts. Finally, the results—including predicted demand, price suggestions, and summary metrics—are displayed to the user through an interactive dashboard. This structured data flow ensures efficient processing, accurate predictions, and easy interpretation for small vendors.

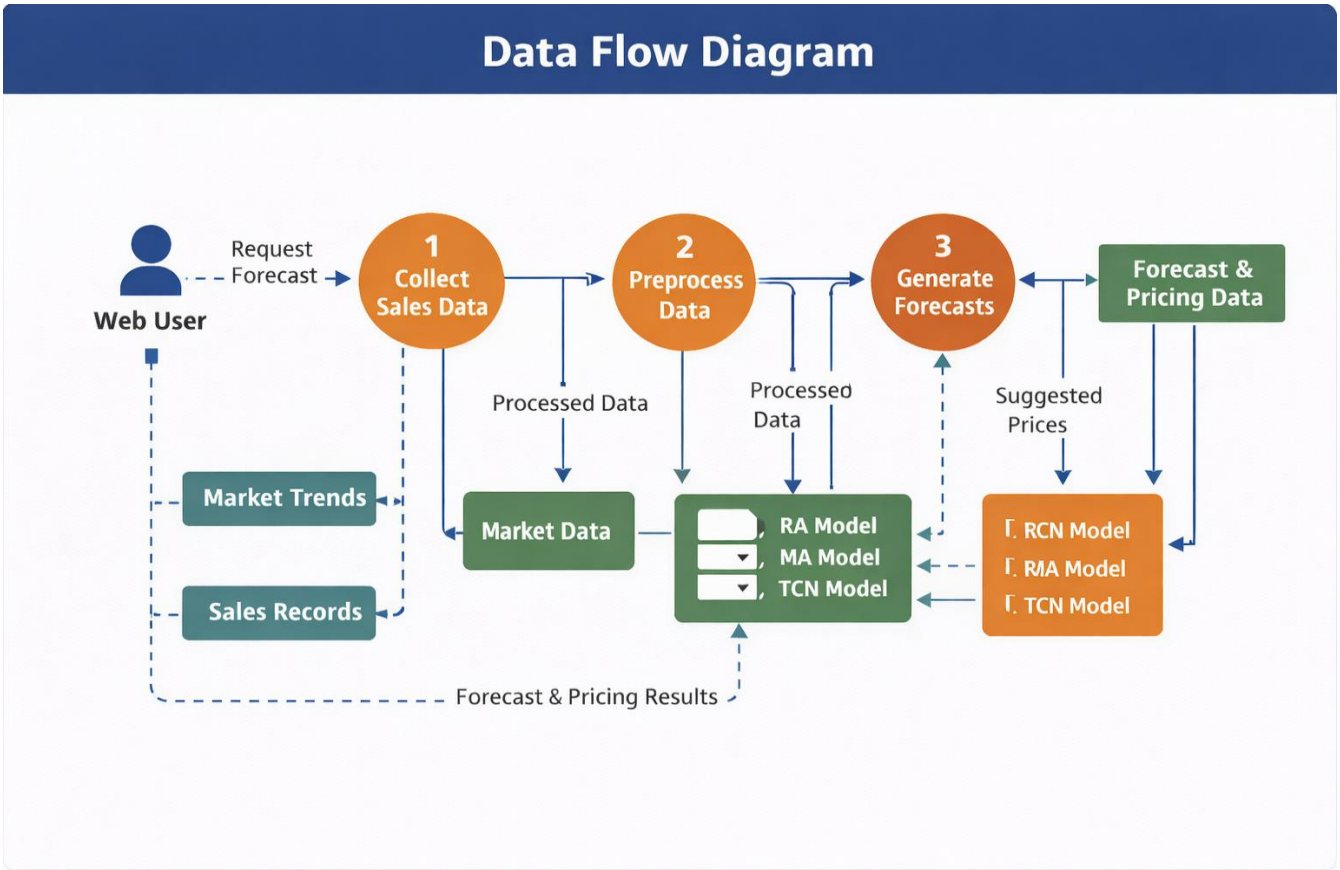

Fig 5.2.1 Data Flow diagram

5.3 Uml Diagrams

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

The two broadest categories that encompass all other types are:

1. Behavioural UML diagram and

2. Structural UML diagram.

As the name suggests, some UML diagrams try to analyses and depict the structure of a system or process, whereas other describe the behavior of the system, its actors, and its building components.

The different types are broken down as follows:

1. Sequence diagram
2. Use case Diagram
3. Activity diagram

1. Sequence Diagram

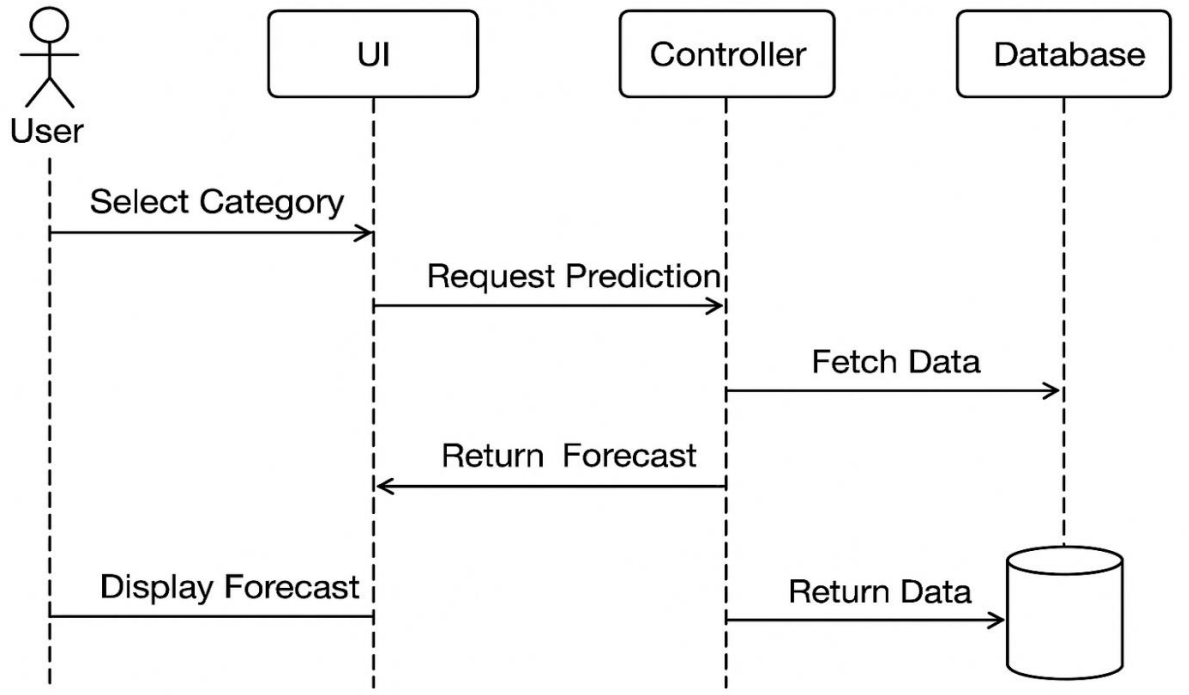

Fig 5.3.1 Sequence diagram

The sequence diagram describes the step-by-step communication between different system components. When the vendor uploads a file, the frontend sends it to the backend API, which forwards it to the data preprocessing unit. Once cleaned, the data is passed to the forecasting engine, where RA, MA, and TCN models generate their respective outputs. These results are then sent to the graph generation unit, which creates the visual charts. All processed data and images are collected by the result manager and returned to the frontend interface. The vendor then views the complete dashboard

containing forecasts and price insights. The sequence diagram emphasizes the order and timing of interactions among modules.

2. Use Case Diagram

The use case diagram represents the interaction between the small vendor and the forecasting system. It shows how the vendor uploads the sales data, requests forecasts, and views the analytical outputs. The system acts as the main service provider, responding to user inputs by communicating with the forecasting engine. Once the vendor selects a category, product, and timeframe, the system processes the request and generates the rolling average, moving average, and TCN forecasts. It also produces price recommendations and summary metrics, which are displayed to the vendor. The use case diagram highlights the system's core goal—to deliver a simple and efficient forecasting experience with minimal user involvement.

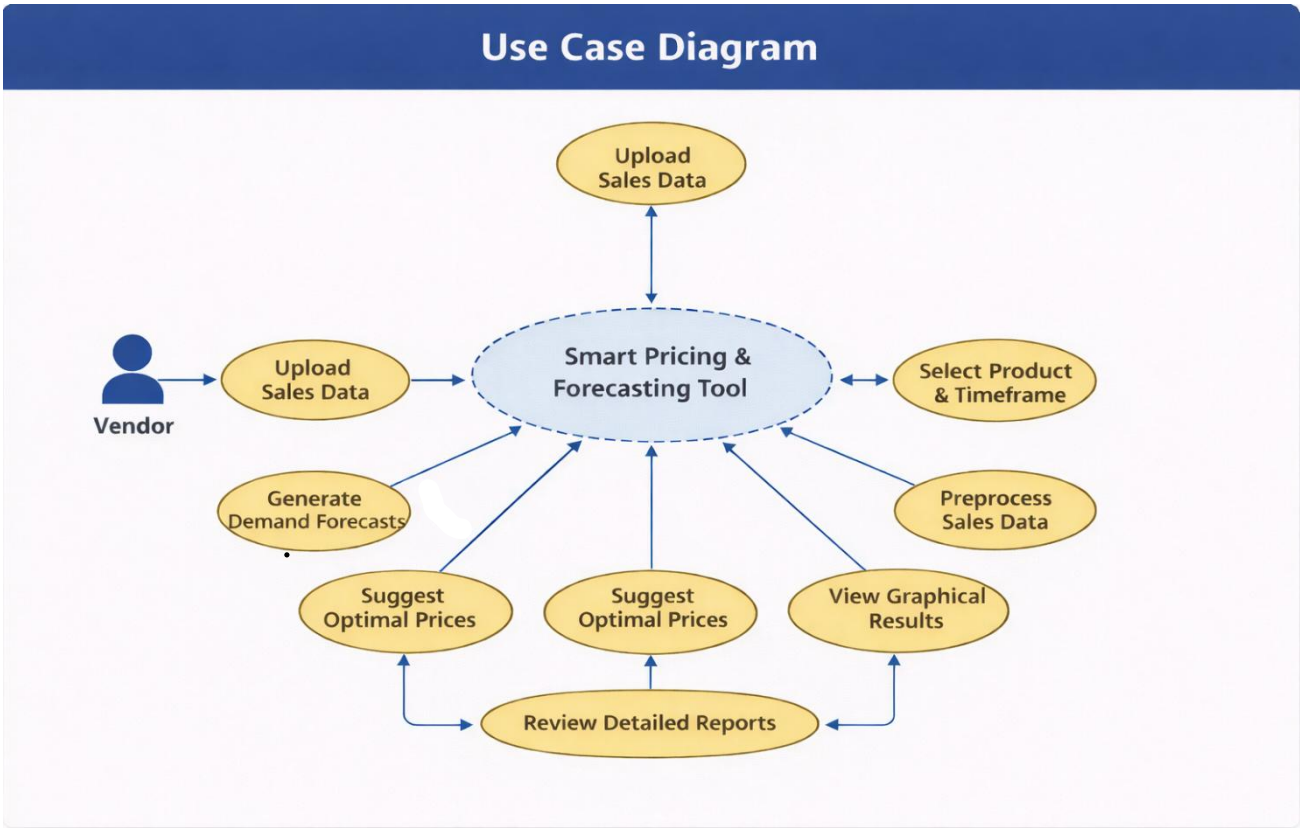

Fig 5.3.2 Use case diagram

3. Activity Diagram

The activity diagram illustrates the workflow of the forecasting process from start to finish. The process

begins when the vendor uploads the required sales data. The system first validates and preprocesses the data before sending it to the forecasting modules. It calculates rolling averages, moving averages, and runs the TCN model to generate predictions. After this, the pricing analysis module evaluates historical price trends and produces suggested price ranges. The graph generator then creates visual representations such as actual trends, RA, MA, and TCN graphs. Finally, all results—numerical outputs, graphs, and insights—are combined and presented to the vendor on the dashboard. The flow clearly shows how the system transitions smoothly between stages.

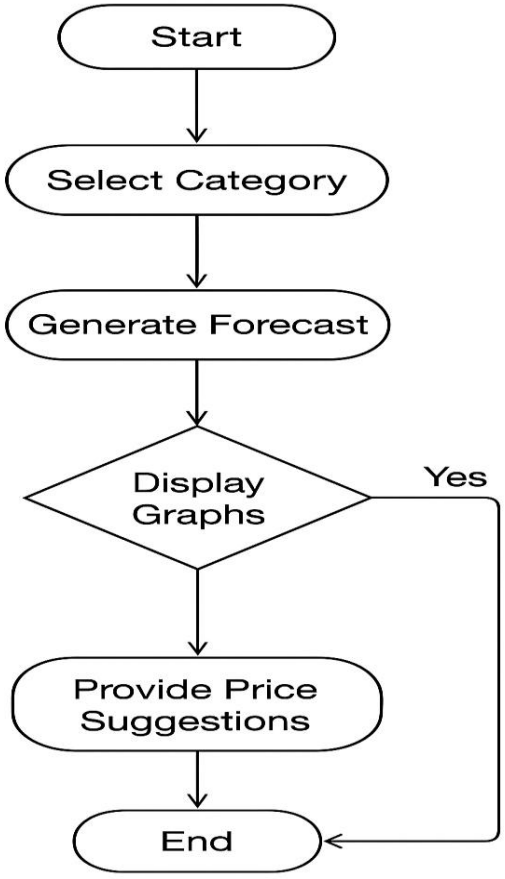

Fig 5.3.3 Activity diagram

4. Class Diagram

The class diagram of the Smart Pricing and Forecasting Tool for Small Vendors represents the structure of the system by showing its main components, their attributes, methods, and relationships. The Vendor class represents the user interacting with the system, allowing actions such as selecting products and viewing reports. The Sales Data class stores important information like product ID, date, quantity sold, and price,

which acts as the primary input for forecasting.

The Forecasting Module is the core component of the system, where different models such as Rolling Average, Moving Average, and TCN are implemented to generate demand predictions. This module takes processed sales data and produces forecast results. The Pricing Analysis Module analyzes historical price trends and suggests optimal pricing based on demand patterns. The Visualization class is responsible for displaying results through graphs, summaries, and reports, making the output easy to understand. The relationships between these classes show how data flows from sales data to forecasting and pricing modules, and finally to visualization. Overall, the class diagram clearly defines the system structure and ensures smooth interaction between all components for efficient forecasting and decision-making.

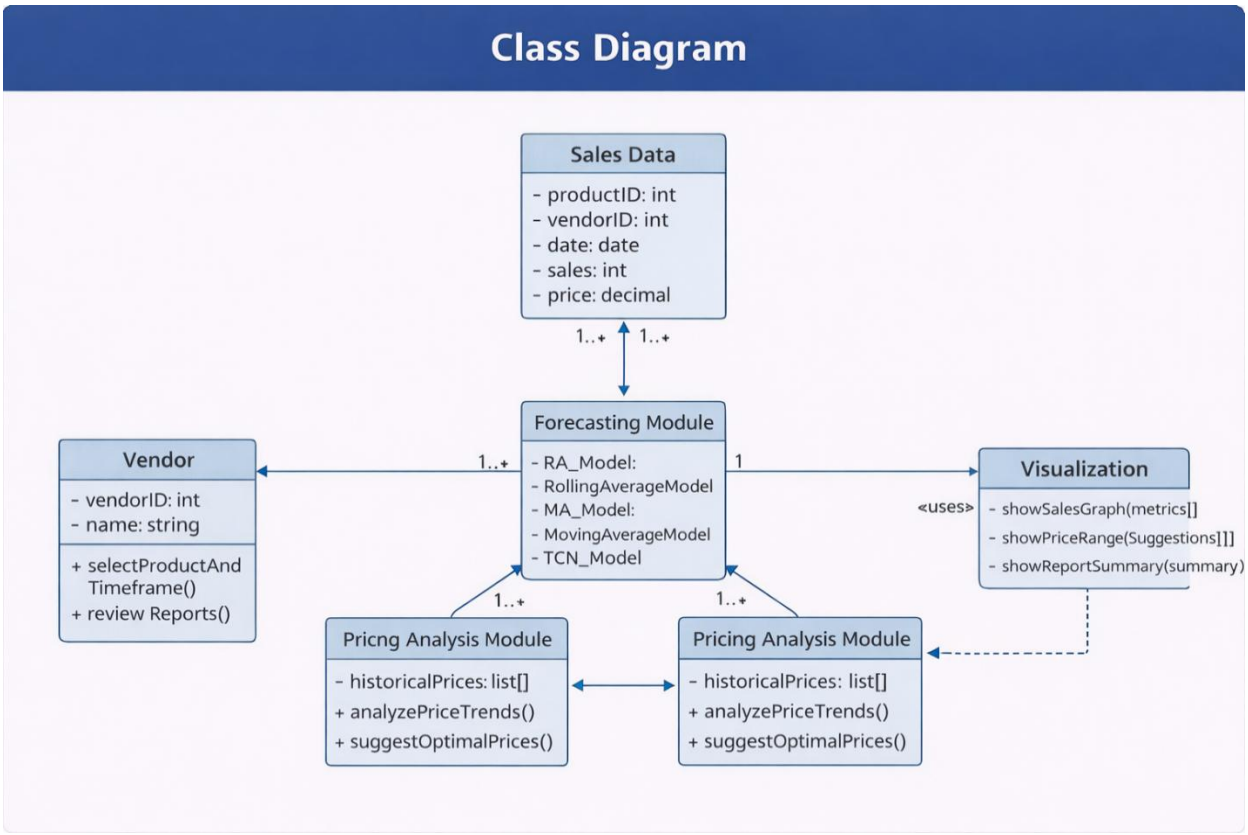

Fig 5.3.4 Class diagram

CHAPTER 6

CODING AND IMPLEMENTATION

6. CODING AND IMPLEMENTATION

6.1 Source Code

Frontend File Code: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Smart Pricing Tool</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body class="bg-light">
<nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-4">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Smart Pricing and Forecasting Tool for Small Vendors</a>
    </div>
</nav>
<div class="container">
    <div class="card shadow-sm">
        <div class="card-body">
            <h4 class="card-title mb-4">Forecast Your Product Demand</h4>
            <form id="product-form" class="row g-3">
```

```
<div class="col-md-4">
    <label for="category" class="form-label">Category</label>
    <select id="category" name="category" class="form-select">
        <option value="">-- Choose Category --</option>
        {% for cat in categories %}
            <option value="{{ cat }}">{{ cat }}</option>
        {% endfor %}
    </select>
</div>
<div class="col-md-4">
    <label for="product" class="form-label">Product</label>
    <select id="product" name="product" class="form-select">
        <option value="">-- Choose Product --</option>
    </select>
</div>
<div class="col-md-4">
    <label for="timeframe" class="form-label">Time Frame</label>
    <select id="timeframe" name="timeframe" class="form-select">
        <option value="">-- Choose --</option>
        <option value="7">Next 7 Days</option>
        <option value="15">Next 15 Days</option>
        <option value="30">Next 30 Days</option>
        <option value="custom">Custom Range</option>
    </select>
</div>
```

```
            <div id="custom-range" class="col-12 row gx-2" style="display:none;">
              <div class="col-md-6">
                <label for="start_date" class="form-label">From</label>
                <input type="date" id="start_date" class="form-control">
              </div>
              <div class="col-md-6">
                <label for="end_date" class="form-label">To</label>
                <input type="date" id="end_date" class="form-control">
              </div>
            </div>
            <div class="col-12 text-center">
              <div id="loading-spinner" class="spinner-border text-primary" role="status"
style="display:none;">
                <span class="visually-hidden">Loading...</span>
              </div>
            </div>
            <div class="col-12">
              <button type="button" id="forecast-btn" class="btn btn-success">Get Forecast</button>
            </div>
          </form>
        </div>
      </div>
      <div id="results" class="card mt-4 shadow-sm" style="display:none;">
        <div class="card-body">
```

```
        <h5 class="card-title">Forecast Results</h5>
        <p><strong>Rolling Average Forecast:</strong> <span id="rolling"></span></p>
        <p><strong>Weighted Moving Average Forecast:</strong> <span id="weighted"></span></p>
        <p><strong>Temporal Convolutional Network Forecast:</strong> <span id="tcn"></span></p>
        <p><strong>Tip:</strong> <span id="tip"></span></p>
        <hr>
        <h6> Price Insights</h6>
        <p><strong>Suggested Price Range:</strong> <span id="price-range"></span></p>
        <p><strong>Price Tip:</strong> <span id="price-tip"></span></p>
    </div>
</div>
<div id="summary-cards" class="row mt-4" style="display:none;">
    <div class="col-md-4">
        <div class="card text-white bg-primary mb-3">
            <div class="card-header"> Total Units (Last 30 Days)</div>
            <div class="card-body">
                <h5 class="card-title" id="total-units"></h5>
            </div>
        </div>
    </div>
    <div class="col-md-4">
        <div class="card text-white bg-success mb-3">
            <div class="card-header"> Avg Daily Sales</div>
            <div class="card-body">
                <h5 class="card-title" id="avg-sales"></h5>
```

```
            </div>
          </div>
        </div>
        <div class="col-md-4">
          <div class="card text-white bg-warning mb-3">
            <div class="card-header"> Avg Price (Last 30 Days)</div>
            <div class="card-body">
              <h5 class="card-title" id="avg-price"></h5>
            </div>
          </div>
        </div>
      </div>
      <div id="image-results" class="row mt-4" style="display:none;">
        <!-- Actual Sales Trend -->
        <div class="col-md-6 mb-4">
          <h6>Actual Sales Trend (30 Days)</h6>
          <img id="actual-image" src="" class="img-fluid border rounded shadow-sm" alt="Actual Sales">
        </div>
        <!-- Rolling Average -->
        <div class="col-md-6 mb-4">
          <h6>Rolling Average (7 Days)</h6>
          <img id="rolling-image" src="" class="img-fluid border rounded shadow-sm"
alt="Rolling Average">
        </div>
        <!-- Moving Average Forecast -->
```

```
        <div class="col-md-6 mb-4">
            <h6>Moving Average Forecast</h6>
            <img id="ma-image" src="" class="img-fluid border rounded shadow-sm" alt="MA Forecast">
        </div>
        <!-- TCN Forecast -->
        <div class="col-md-6 mb-4">
            <h6>TCN Forecast</h6>
            <img id="tcn-image" src="" class="img-fluid border rounded shadow-sm" alt="TCN Forecast">
        </div>
        <div class="col-md-6">
            <h6> Summary Metrics</h6>
            <img id="summary-image" src="" class="img-fluid border rounded shadow-sm"
alt="Summary chart">
        </div>
    </div>
</div>
<script>
    $(document).ready(function () {
        $("#category").change(function () {
            let selectedCategory = $(this).val();
            if (selectedCategory) {
                $.ajax({
                    url: "/get-products",
                    type: "POST",
                    contentType: "application/json",
```

```
            data: JSON.stringify({ category: selectedCategory }),
            success: function (response) {
                let productDropdown = $("#product");
                productDropdown.empty();
                productDropdown.append('<option value="">-- Choose Product --</option>');
                response.products.forEach(function (prod) {
                    productDropdown.append(`<option value="${prod}">${prod}</option>`);
                });
            }
        });
    } else {
        $("#product").empty().append('<option value="">-- Choose Product --</option>');
    }
});
$("#timeframe").change(function () {
    $("#custom-range").toggle($(this).val() === "custom");
});
$("#forecast-btn").click(function () {
    let category = $("#category").val();
    let product = $("#product").val();
    let timeframe = $("#timeframe").val();
    let start_date = $("#start_date").val();
    let end_date = $("#end_date").val();
    if (!category || !product || !timeframe) {
        alert("Please select category, product, and timeframe.");
```

```
    return;
}
if (timeframe === "custom" && (!start_date || !end_date)) {
    alert("Please select both start and end dates.");
    return;
}
const payload = {
    category: category,
    product: product,
    timeframe: timeframe,
    start_date: start_date,
    end_date: end_date
};
$("#loading-spinner").show();
$.ajax({
    url: "/get-forecast",
    type: "POST",
    contentType: "application/json",
    data: JSON.stringify(payload),
    success: function (data) {
        $("#rolling").text(data.rolling_forecast);
        $("#weighted").text(data.weighted_forecast);
        $("#tcn").text(data.tcn_forecast);
        $("#tip").text(data.tip);
        $("#price-range").text(data.price_suggestion);
```

```
            $("#price-tip").text(data.price_change);
            $("#total-units").text(data.summary.total_units);
            $("#avg-sales").text(data.summary.avg_daily_sales);
            $("#avg-price").text(data.summary.avg_price);

            $("#results").show();
            $("#summary-cards").show();
            const ts = new Date().getTime();
            $("#actual-image").attr("src", "/static/uploads/actual_trend.png?t=" + ts);
            $("#rolling-image").attr("src", "/static/uploads/rolling_trend.png?t=" + ts);
            $("#ma-image").attr("src", "/static/uploads/ma_forecast.png?t=" + ts);
            $("#tcn-image").attr("src", "/static/uploads/tcn_forecast.png?t=" + ts);
            $("#summary-image").attr("src", "/static/uploads/summary.png?t=" + ts);
            $("#image-results").show();
        },
        error: function (xhr) {
            alert("Error: " + (xhr.responseJSON?.error || "Unable to get forecast"));
        },
        complete: function () {
            $("#loading-spinner").hide();
        }
      });
    });
  });
</script>
```

```
</body>
</html>
```

Backend File Code: app.py

```
from flask import Flask, render_template, request, jsonify
import pandas as pd
from datetime import datetime, timedelta
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import seaborn as sns
import os
import numpy as np
# --- Optional: deep learning imports ---
try:
    import torch
    import torch.nn as nn
    from torch.utils.data import DataLoader, TensorDataset
except Exception as e:
    torch = None
    nn = None
    DataLoader = None
    TensorDataset = None
    print("Warning: torch not available. TCN functionality will be disabled.", e)
```

```
app = Flask(__name__)
# Ensure upload folder exists
UPLOAD_FOLDER = 'static/uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# Ensure models folder exists
MODEL_FOLDER = "models"
os.makedirs(MODEL_FOLDER, exist_ok=True)
# Load and preprocess data
df_all = pd.read_csv("data/trice_cleaned.csv", encoding='ISO-8859-1')
df_all["order_date"] = pd.to_datetime(df_all["order_date"], errors="coerce")
def get_categories():
    return sorted(df_all["category"].dropna().unique())
def get_products_by_category(category):
    filtered = df_all[df_all["category"] == category]
    return sorted(filtered["Product_or_Dish"].dropna().unique())
# ----------------------------
# --- TCN implementation -----
# ----------------------------
# Only define TCN classes if torch is available
if torch is not None:
    DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    class TCNBlock(nn.Module):
        def __init__(self, in_channels, out_channels, dilation):
            super().__init__()
```

```
        # causal convolution implemented via padding=dilation, then cropping is avoided by
         using last output
        self.conv1 = nn.Conv1d(in_channels, out_channels, kernel_size=3,
                               padding=dilation, dilation=dilation)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv1d(out_channels, out_channels, kernel_size=3,
                               padding=dilation, dilation=dilation)
        self.relu2 = nn.ReLU()
        self.downsample = nn.Conv1d(in_channels, out_channels, 1) if in_channels != out_channels
else None
    def forward(self, x):
        out = self.relu1(self.conv1(x))
        out = self.relu2(self.conv2(out))
        res = x if self.downsample is None else self.downsample(x)
        return out + res
class TCNForecaster(nn.Module):
    def __init__(self, input_channels=1, num_channels=[16, 32, 64]):
        super().__init__()
        layers = []
        for i, out_ch in enumerate(num_channels):
            dilation = 2 ** i
            in_ch = input_channels if i == 0 else num_channels[i - 1]
            layers.append(TCNBlock(in_ch, out_ch, dilation))
        self.network = nn.Sequential(*layers)
        self.linear = nn.Linear(num_channels[-1], 1)
```

```
    def forward(self, x):
        # x: (batch, channels, seq_len)
        out = self.network(x)           # (batch, channels_out, seq_len)
        out = out[:, :, -1]             # take last time-step features
        return self.linear(out)         # (batch, 1)


def create_sequences(values, window=30):
    """
    values: 1D numpy array
    returns X of shape (n_samples, 1, window) and y of shape (n_samples, 1)
    """
    X, y = [], []
    if len(values) <= window:
        return np.empty((0, window)), np.empty((0,))
    for i in range(len(values) - window):
        X.append(values[i:i + window])
        y.append(values[i + window])
    X = np.array(X, dtype=float)
    y = np.array(y, dtype=float)
    return X, y
def train_tcn_for_product(series_values, model_path, window=30, epochs=20,
batch_size=32, lr=1e-3):
    """
    series_values: 1D numpy array of daily quantities (continuous daily index expected)
    model_path: where to save the trained model
```

```
Trains a small TCN and saves state_dict to model_path
"""
X, y = create_sequences(series_values, window=window)
if X.shape[0] == 0:
    # Not enough data to train
    return False

X = X.reshape(X.shape[0], 1, window)  # (n, 1, window)
y = y.reshape(-1, 1)
dataset = TensorDataset(torch.tensor(X, dtype=torch.float32),
                        torch.tensor(y, dtype=torch.float32))
loader = DataLoader(dataset, batch_size=min(batch_size, len(dataset)), shuffle=True)
model = TCNForecaster().to(DEVICE)
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
loss_fn = nn.MSELoss()
model.train()
for epoch in range(epochs):
    epoch_loss = 0.0
    for batch_x, batch_y in loader:
        batch_x = batch_x.to(DEVICE)
        batch_y = batch_y.to(DEVICE)
        pred = model(batch_x)
        loss = loss_fn(pred, batch_y)
        optimizer.zero_grad()
        loss.backward()
```

```
            optimizer.step()
            epoch_loss += loss.item() * batch_x.size(0)
        # simple early exit for small data if loss stagnates (helps speed)
        # (we avoid complex schedulers to keep the file self-contained)
    # save model
    torch.save(model.state_dict(), model_path)
    return True

def tcn_predict_next_days_from_series(series_values, model_path, days=7, window=30):
    """
    Returns list of predicted values for next `days` days using the TCN model at model_path.
    If model_path not present, returns empty list.
    """
    if not os.path.exists(model_path):
        return []
    model = TCNForecaster().to(DEVICE)
    try:
        model.load_state_dict(torch.load(model_path, map_location=DEVICE, weights_only=True))
    except Exception:
        # incompatible model or corrupt
        return []
    model.eval()
    seq = series_values.copy().astype(float)
    # if sequence shorter than window, pad with zeros at front
    if len(seq) < window:
```

```
            seq = np.concatenate([np.zeros(window - len(seq)), seq])
        preds = []
        with torch.no_grad():
            for _ in range(days):
                x = torch.tensor(seq[-window:].reshape(1, 1, -1), dtype=torch.float32).to(DEVICE)
                next_val = model(x).cpu().numpy().ravel()[0]
                # clip to >= 0
                if np.isnan(next_val) or np.isinf(next_val):
                    next_val = 0.0
                next_val = max(0.0, float(next_val))
                preds.append(round(next_val, 2))
                seq = np.append(seq, next_val)
        return preds
# ----------------------------
# --- End TCN section -------
# ----------------------------
@app.route("/")
def index():
    categories = get_categories()
    return render_template("index.html", categories=categories)
@app.route("/get-products", methods=["POST"])
def get_products():
    category = request.json.get("category")
    if not category:
        return jsonify({"products": []})
```

```
    products = get_products_by_category(category)
    return jsonify({"products": products})
@app.route("/get-forecast", methods=["POST"])
def get_forecast():
    data = request.json
    category = data.get("category")
    product = data.get("product")
    timeframe = data.get("timeframe")
    start_date = data.get("start_date")
    end_date = data.get("end_date")
    if timeframe == "custom" and start_date and end_date:
        try:
            start_dt = datetime.strptime(start_date, "%Y-%m-%d")
            end_dt = datetime.strptime(end_date, "%Y-%m-%d")
            forecast_days = (end_dt - start_dt).days + 1
            if forecast_days <= 0:
                return jsonify({"error": "Invalid custom date range."}), 400
        except ValueError:
            return jsonify({"error": "Invalid date format."}), 400
    else:
        forecast_days = int(timeframe or 7)
    filtered_df = df_all[
        (df_all["category"] == category) &
        (df_all["Product_or_Dish"] == product)
    ].copy()
```

```
today = df_all["order_date"].max()
last_30_days = today - timedelta(days=30)
same_range_last_year_start = last_30_days - timedelta(days=365)
same_range_last_year_end = today - timedelta(days=365)
recent_df = filtered_df[filtered_df["order_date"] >= last_30_days]
past_year_df = filtered_df[
    (filtered_df["order_date"] >= same_range_last_year_start) &
    (filtered_df["order_date"] <= same_range_last_year_end)
]
# --- Rolling Average ---
total_qty = recent_df["quantity"].sum()
num_days = recent_df["order_date"].nunique() or 1
daily_avg = total_qty / num_days
rolling_forecast = round(daily_avg * forecast_days)
# --- Weighted Moving Average ---
weights = [0.5, 0.3, 0.2]
weighted_sum = 0
for i, weight in enumerate(weights):
    start = today - timedelta(days=7 * (i + 1))
    end = today - timedelta(days=7 * i)
    week_df = filtered_df[
        (filtered_df["order_date"] >= start) &
        (filtered_df["order_date"] < end)
    ]
    week_qty = week_df["quantity"].sum()
```

```
        week_days = week_df["order_date"].nunique() or 1
        weighted_sum += (week_qty / week_days) * weight
    weighted_forecast = round(weighted_sum * forecast_days)
    # --- Price Suggestion ---
    recent_avg_price = recent_df["price_after_discount"].mean()
    past_avg_price = past_year_df["price_after_discount"].mean()
    price_change_percent = 0
    if past_avg_price and past_avg_price > 0:
        price_change_percent = ((recent_avg_price - past_avg_price) / past_avg_price) * 100
    suggested_low = round(recent_avg_price * 0.975, 2) if not np.isnan(recent_avg_price) else None
    suggested_high = round(recent_avg_price * 1.025, 2) if not np.isnan(recent_avg_price) else None
    price_range = f"₹{suggested_low} – ₹{suggested_high}" if suggested_low is not None and
suggested_high is   not None else "N/A"
    price_tip = "Price data not available."
    if price_change_percent > 0:
        price_tip = f"Prices have increased {price_change_percent:.1f}% compared to the
 same period last year."
    elif price_change_percent < 0:
        price_tip = f"Prices have dropped {abs(price_change_percent):.1f}% compared to the
 same period last year."
    else:
        price_tip = "Prices have remained stable compared to last year."
    # --- Summary Metrics ---
    total_units = int(total_qty)
    avg_daily_sales = round(daily_avg, 2)
```

```
    avg_price = round(recent_avg_price, 2) if recent_avg_price and not np.isnan(recent_avg_price)
else None
    chart_dates = []
    chart_sales = []
    recent_trend_df = recent_df.groupby("order_date").agg({"quantity": "sum"}).reset_index()
    recent_trend_df = recent_trend_df.sort_values("order_date")
    for _, row in recent_trend_df.iterrows():
        chart_dates.append(row["order_date"].strftime("%b-%d"))
        chart_sales.append(row["quantity"])
    forecast_dates = []
    forecast_values = []
    # avoid division by zero
    if forecast_days > 0:
        avg_forecast_per_day = round((rolling_forecast + weighted_forecast) / 2 / forecast_days, 2)
    else:
        avg_forecast_per_day = 0
    for i in range(forecast_days):
        future_date = today + timedelta(days=i + 1)
        forecast_dates.append(future_date.strftime("%b-%d"))
        forecast_values.append(avg_forecast_per_day)
    all_dates = chart_dates + forecast_dates
    all_sales = chart_sales + forecast_values
    is_forecast = [False] * len(chart_dates) + [True] * len(forecast_dates)
    # --- TCN Forecast integration ---
    tcn_forecast_values = []
```

```
    tcn_model_key = f"{category}_{product}".replace(" ", "_").replace("/", "_")
    tcn_model_path = os.path.join(MODEL_FOLDER, f"{tcn_model_key}_tcn.pth")
    # Build a continuous daily series for this product (fill missing dates with 0)
    # We use the entire available history for training/prediction
    if not filtered_df.empty:
        grouped = filtered_df.groupby("order_date").agg({"quantity":
"sum"}).reset_index().sort_values("order_date")
        # create continuous date index between min and max
        min_date = grouped["order_date"].min()
        max_date = grouped["order_date"].max()
        all_dates_index = pd.date_range(start=min_date, end=max_date, freq='D')
        daily_series = grouped.set_index("order_date").reindex(all_dates_index,
fill_value=0)["quantity"].astype(float).values
    else:
        daily_series = np.array([], dtype=float)
    # If torch is available, ensure model exists, otherwise train quick model, then predict
    if torch is not None and len(daily_series) > 0:
        # If model doesn't exist, train quickly (small epochs) and save
        if not os.path.exists(tcn_model_path):
            # try training; use smaller epochs for small datasets to keep it fast
            try:
                # epochs scaled down for small datasets
                train_epochs = 30 if len(daily_series) > 200 else 10
                train_tcn_for_product(daily_series, tcn_model_path, window=30, epochs=train_epochs,
batch_size=16, lr=1e-3)
            except Exception as e:
```

```
            print("TCN training failed:", e)
        # predict next days if model exists
        if os.path.exists(tcn_model_path):
            try:
                tcn_preds = tcn_predict_next_days_from_series(daily_series, tcn_model_path,
days=forecast_days, window=30)
                # ensure length matches forecast_days (pad with zeros if necessary)
                if len(tcn_preds) < forecast_days:
                    tcn_preds = tcn_preds + [0.0] * (forecast_days - len(tcn_preds))
                tcn_forecast_values = tcn_preds
            except Exception as e:
                print("TCN prediction failed:", e)
                tcn_forecast_values = []
    else:
        # Torch not available or no data -- TCN forecast not possible
        tcn_forecast_values = []
    # compute TCN totals
    if tcn_forecast_values:
        tcn_forecast_total = round(sum(tcn_forecast_values))
        tcn_daily_avg = round(sum(tcn_forecast_values) / len(tcn_forecast_values), 2)
    else:
        tcn_forecast_total = None
        tcn_daily_avg = None
    # --- Visualization: Line Chart (overlay actual + MA forecast + TCN forecast if available) ---
    # ============================
```

```
# 1️⃣ Actual Sales Trend Graph
# ============================
actual_img_path = os.path.join(UPLOAD_FOLDER, "actual_trend.png")
plt.clf()
plt.figure(figsize=(15, 10))
# convert dates to string for plotting
actual_dates = [str(d) for d in chart_dates]
plt.plot(actual_dates, chart_sales, marker='o', color='blue', label="Actual Sales")
plt.xticks(rotation=45)
plt.title("Actual Sales Trend (Last 30 Days)")
plt.xlabel("Date")
plt.ylabel("Units Sold")
plt.legend()
plt.tight_layout()
plt.savefig(actual_img_path)
plt.close()
# ============================
# 2️⃣ Rolling Average Trend Graph
# ============================
rolling_img_path = os.path.join(UPLOAD_FOLDER, "rolling_trend.png")
plt.clf()
plt.figure(figsize=(15, 10))
rolling_window = 7
rolling_avg_series = (
    recent_trend_df["quantity"]
```

```
    .rolling(rolling_window)
    .mean()
    .fillna(method='bfill')
    .fillna(0)
)
rolling_avg_dates = recent_trend_df["order_date"].dt.strftime("%b-%d").tolist()
plt.plot(rolling_avg_dates, rolling_avg_series, linestyle="--", marker='o',color='green',
        label=f"Rolling Avg ({rolling_window} days)")
plt.xticks(rotation=45)
plt.title(f"Rolling Average Trend ({rolling_window} Days)")
plt.xlabel("Date")
plt.ylabel("Avg Units Sold")
plt.legend()
plt.tight_layout()
plt.savefig(rolling_img_path)
plt.close()
# ============================
# 3️⃣ Moving Average Forecast Graph
# ============================
ma_img_path = os.path.join(UPLOAD_FOLDER, "ma_forecast.png")
plt.clf()
plt.figure(figsize=(15, 10))
if forecast_dates and forecast_values:
    ma_dates = [str(d) for d in forecast_dates]
    plt.plot(ma_dates, forecast_values, marker='o',color='orange', label="Moving Avg Forecast")
```

```
else:
    plt.text(0.3, 0.5, "No MA Forecast Available", fontsize=14)
plt.xticks(rotation=45)
plt.title(f"Moving Average Forecast ({forecast_days} Days)")
plt.xlabel("Date")
plt.ylabel("Forecasted Units")
plt.legend()
plt.tight_layout()
plt.savefig(ma_img_path)
plt.close()
# ============================
# 4️⃣ TCN Forecast Graph
# ============================
tcn_img_path = os.path.join(UPLOAD_FOLDER, "tcn_forecast.png")
plt.clf()
plt.figure(figsize=(15, 10))
if tcn_forecast_values:
    tcn_dates = [str(d) for d in forecast_dates]
    plt.plot(tcn_dates, tcn_forecast_values, marker='o',color='red', label="TCN Forecast")
else:
    plt.text(0.3, 0.5, "TCN Forecast Not Available", fontsize=14)
plt.xticks(rotation=45)
plt.title(f"TCN Forecast ({forecast_days} Days)")
plt.xlabel("Date")
plt.ylabel("Units")
```

```
plt.legend()
plt.tight_layout()
plt.savefig(tcn_img_path)
plt.close()
# --- Visualization: Summary Bar Chart ---
summary_img_path = os.path.join(UPLOAD_FOLDER, "summary.png")
plt.figure(figsize=(15, 10))
summary_labels = ["Total Units", "Avg Daily Sales", "Avg Price"]
summary_values = [total_units, avg_daily_sales, avg_price if avg_price else 0]
sns.barplot(x=summary_labels, y=summary_values, palette="pastel")
plt.title("Sales Summary (Last 30 Days)")
for index, value in enumerate(summary_values):
    try:
        plt.text(index, value, f"{value:.2f}", ha='center', va='bottom')
    except Exception:
        plt.text(index, value, f"{value}", ha='center', va='bottom')
plt.tight_layout()
plt.savefig(summary_img_path)
plt.close()
response = {
    "rolling_forecast": f"{rolling_forecast} units",
    "weighted_forecast": f"{weighted_forecast} units",
    "tip": f"Forecast is for {forecast_days} days.",
    "price_suggestion": price_range,
    "price_change": price_tip,
```

```
        "summary": {
            "total_units": total_units,
            "avg_daily_sales": avg_daily_sales,
            "avg_price": f"₹{avg_price}" if avg_price else "N/A"
        },
        "images": {
            "actual": "/static/uploads/actual_trend.png",
            "rolling": "/static/uploads/rolling_trend.png",
            "ma": "/static/uploads/ma_forecast.png",
            "tcn": "/static/uploads/tcn_forecast.png",
            "summary": "/static/uploads/summary.png"
        }
    }
    # Add deep learning results (TCN)
    if tcn_forecast_total is not None:
        response["tcn_forecast"] = f"{tcn_forecast_total} units"
        response["deep_learning_forecast"] = f"{tcn_forecast_total} units"
        response["deep_learning_daily"] = tcn_daily_avg
    else:
        response["tcn_forecast"] = "N/A"
        response["deep_learning_forecast"] = "N/A"
        response["deep_learning_daily"] = "N/A"
    return jsonify(response)
if __name__ == "__main__":
    app.run(debug=True)
```

6.2 Implementation

6.2.1 Front-End Implementation

The front-end of the Smart Pricing and Forecasting Tool for Small Vendors provides a simple, responsive, and user-friendly interface. The main modules include Category Selection, Product Selection, Timeframe Selection, Forecast Generation, and Result Visualization.

The interface allows users to select a product category and corresponding product from dropdown menus. Users can choose a forecast period such as 7, 15, or 30 days, or specify a custom date range. Input validation is applied to ensure that all required fields are selected before generating forecasts.

Once the user submits the request, the input data is sent to the backend using AJAX. The results, including demand forecasts and price suggestions, are displayed clearly on the dashboard. The system presents outputs in both numerical and graphical formats, including sales trends and forecast graphs, ensuring easy interpretation for non-technical users. The use of Bootstrap ensures responsive design and smooth navigation across devices.

6.2.2 Backend Implementation (Flask)

The backend is implemented using Flask, which provides a lightweight and efficient framework for handling web requests and processing data. The backend manages routing, data processing, forecasting logic, and result generation.

- Flask routes handle requests for product selection and forecasting.
- Pandas is used for data loading, cleaning, and preprocessing.
- NumPy supports numerical computations required for forecasting.
- APIs are defined to process user inputs and return results in JSON format.

The backend ensures proper validation of inputs, efficient data handling, and seamless communication with

the frontend.

6.2.3 Model Integration and Processing Workflow

The forecasting module is integrated within the backend and processes data through multiple stages. When a request is received, the system first preprocesses the sales data by handling missing values, organizing it into a time-series format, and aggregating daily sales.

Forecasting is performed using three methods: Rolling Average (RA), Moving Average (MA), and Temporal Convolutional Network (TCN). The statistical models capture short-term trends, while the deep learning model identifies complex temporal patterns.

The system also includes a pricing analysis module that evaluates historical pricing trends and suggests an optimal price range. The final outputs, including demand forecasts and price insights, are returned to the frontend in structured JSON format and displayed to the user.

6.2.4 Deployment and Reliability

The system is deployed on a local server environment using Flask. The application runs on a specified port (127.0.0.1:5000) and supports real-time forecasting requests.

Static files such as generated graphs are stored and served efficiently. The system is tested under multiple scenarios to ensure stable performance, quick response time, and accurate predictions. Proper error handling mechanisms are implemented to manage invalid inputs and system exceptions.

6.2.5 Libraries installed in Project

1. pip install pandas
2. pip install numpy
3. pip install matplotlib
4. pip install seaborn
5. pip install pyplot
6. pip install jsonify
8. pip install Flask

9. pip install render_template
10. pip install torch
11. pip install request
12. pip install datetime
13. pip install timedata
14. pip install os
15. pip install DataLoader
16. pip install TensorDataSet

6.2.6 Conclusion

The implementation successfully integrates a user-friendly frontend, a robust Flask backend, and efficient forecasting models into a unified system. The tool provides accurate demand predictions, pricing suggestions, and clear visual insights in real time. Its simple design makes it suitable for small vendors without technical expertise. The modular architecture allows future enhancements such as improved models, cloud deployment, and advanced analytics features.

CHAPTER 7

SYSTEM TESTING

7. SYSTEM TESTING

System testing is a critical activity that ensures the developed Smart Pricing and Forecasting Tool for Small Vendors performs accurately, consistently, and reliably under real operating conditions. The primary purpose of testing is to identify potential defects, validate system behavior, and confirm that all functional and non-functional requirements have been satisfied.

In this project, system testing focuses on validating all major modules, including the frontend interface, Flask backend services, data preprocessing components, and forecasting models such as Rolling Average (RA), Moving Average (MA), and Temporal Convolutional Network (TCN). Testing ensures that user inputs, data handling, forecasting logic, and graphical outputs operate correctly without errors or inconsistencies.

System testing was conducted across different datasets and scenarios to ensure correctness, robustness, and usability. Special emphasis was placed on forecasting accuracy, handling missing or irregular data, and system stability during repeated forecasting requests.

7.1 Types of System Testing

7.1.1 Unit Testing

Unit testing was performed to validate individual components of the system independently. Each Flask route, preprocessing function, and forecasting model was tested with controlled inputs to verify expected outputs.

Key focus areas included:

• data cleaning and preprocessing logic

• correct computation of Rolling Average and Moving Average

• TCN model prediction behavior

• graph generation and file saving

Unit testing ensured that each internal component worked correctly without unexpected errors before integration.

7.1.2 Integration Testing

Integration testing verified whether different modules worked correctly when combined.

Modules tested in combination included:

• frontend form submission with backend API responses

• preprocessing pipeline connected with forecasting models

• interaction between RA, MA, and TCN models

• generation and display of graphs in the frontend

This testing ensured smooth data flow and correct interaction between all system components.

7.1.3 Functional Testing

Functional testing validated that each system feature worked according to user requirements.

Major validation rules included:

• valid inputs generate correct forecasts

• invalid or missing inputs are handled properly

• correct demand forecast values are displayed

• price suggestions are generated accurately

• graphs and summary metrics are displayed correctly

This testing ensured that all features were usable, accurate, and responsive.

7.1.4 System Testing

System testing evaluated the complete application as a whole under real conditions.

Tests verified:

• overall coordination between frontend and backend

• correct processing of large datasets

• accuracy of forecasting results

• stability during multiple forecast requests

The system demonstrated consistent and reliable performance under different usage scenarios.

7.1.5 White-Box Testing

White-box testing was applied to internal system components such as preprocessing logic and forecasting

algorithms. Internal code execution, data flow, and calculations were analyzed to ensure correctness in model computations and data transformations.

7.1.6 Black-Box Testing

Black-box testing evaluated the system from the user's perspective without examining internal code. Inputs were provided through the user interface, and outputs such as forecasts, price suggestions, and graphs were verified for correctness and usability.

This ensured that the system behaved as expected for end users.

7.1.7 Acceptance Testing

Acceptance testing confirmed that the system met all project requirements and user expectations. The system was evaluated based on usability, forecasting accuracy, output clarity, and overall performance.

Feedback indicated that the system is simple to use, responsive, and effective for small vendors in making data-driven decisions.

7.2 Test Strategies

A structured testing strategy was followed throughout the development of the Smart Pricing and Forecasting Tool for Small Vendors. Testing progressed systematically from validating individual components to verifying the complete system to ensure accuracy, reliability, and performance.

7.2.1 Test Strategy and Approach

Testing was performed both manually and programmatically. Dataset-based test cases and execution logs were used to validate the correctness of forecasting models and system behavior.

Primary strategic objectives included:

- verifying correctness of data preprocessing and time-series input generation
- ensuring accurate output from Rolling Average, Moving Average, and TCN models
- verifying smooth interaction between frontend and backend services
- validating system performance under different datasets and repeated requests

Real-time testing simulated actual user interactions, while controlled test cases ensured logical correctness of forecasting and pricing outputs.

7.2.2 Test Objectives

The following objectives guided the testing process:

- all input fields and selections must work correctly
- system should respond quickly without delays
- invalid or missing inputs must be handled properly
- forecasting results should be consistent with expected trends
- navigation and page transitions should function smoothly

7.2.3 Features Tested

The major system features tested include:

- product and category selection functionality
- correct data preprocessing and handling of missing values
- accuracy of demand forecasting using RA, MA, and TCN models
- generation of price suggestions based on demand trends
- correct display of graphs and summary metrics
- proper working of frontend-backend communication

7.2.4 Integration Testing Strategy

Integration testing focused on ensuring proper interaction between different modules and preventing data mismatches.

Testing confirmed correct:

- connection between frontend input and backend processing
- data flow from preprocessing to forecasting models
- synchronization of outputs from RA, MA, and TCN models
- correct rendering of graphs in the frontend

This strategy ensured smooth system operation and prevented errors during execution.

7.2.5 Acceptance Criteria

The system was accepted only when it satisfied the following conditions:

• accurate generation of demand forecasts

• stable and fast system performance

• error-free input handling and navigation

• clear and understandable output display

• compliance with all functional requirements

7.2.6 Overall Test Results

All planned test cases were executed successfully. The system demonstrated stable performance, accurate forecasting results, and reliable operation across different scenarios. The combination of statistical and deep learning models provided consistent and meaningful predictions.

7.2.7 Conclusion

System testing confirmed that the developed forecasting system meets all functional requirements and operates reliably under various conditions. The integration of preprocessing, forecasting models, and visualization modules ensures smooth performance. The system is user-friendly, accurate, and suitable for helping small vendors make data-driven decisions effectively.

7.3 Sample Test Cases

S No.	Test Case	Expected Result	Result	Remarks (if any)
01	Category Selection	Selected category loads corresponding products	Pass	Ensure dropdown updates correctly
02	Product Selection	Selected product is accepted for forecasting	Pass	Validate product exists in dataset
03	Timeframe Selection	System accepts 7/15/30 days or custom range	Pass	Validate proper date range input
04	Data Preprocessing	Sales data is cleaned and formatted correctly	Pass	Check handling of missing values
05	Rolling Average Forecast	System generates correct RA forecast	Pass	Verify calculation accuracy
06	Moving Average Forecast	System generates correct MA forecast	Pass	Ensure weighted logic works properly
07	TCN Forecast	System generates deep learning-based prediction	Pass	Check model loading and output validity
08	Price Suggestion	System generates correct price range	Pass	Validate price trend calculation
09	Graph Generation	Graphs are created and saved successfully	Pass	Check all 4 graphs are generated

S No.	Test Case	Expected Result	Result	Remarks (if any)
10	Result Display	Forecast results and graphs are displayed correctly	Pass	Verify UI rendering
11	Invalid Input Handling	System shows error for missing selections	Pass	Ensure proper validation messages
12	Multiple Requests	System handles repeated requests without crash	Pass	Check system stability

Table No 7.3.1 Test Cases

Test Case 1:

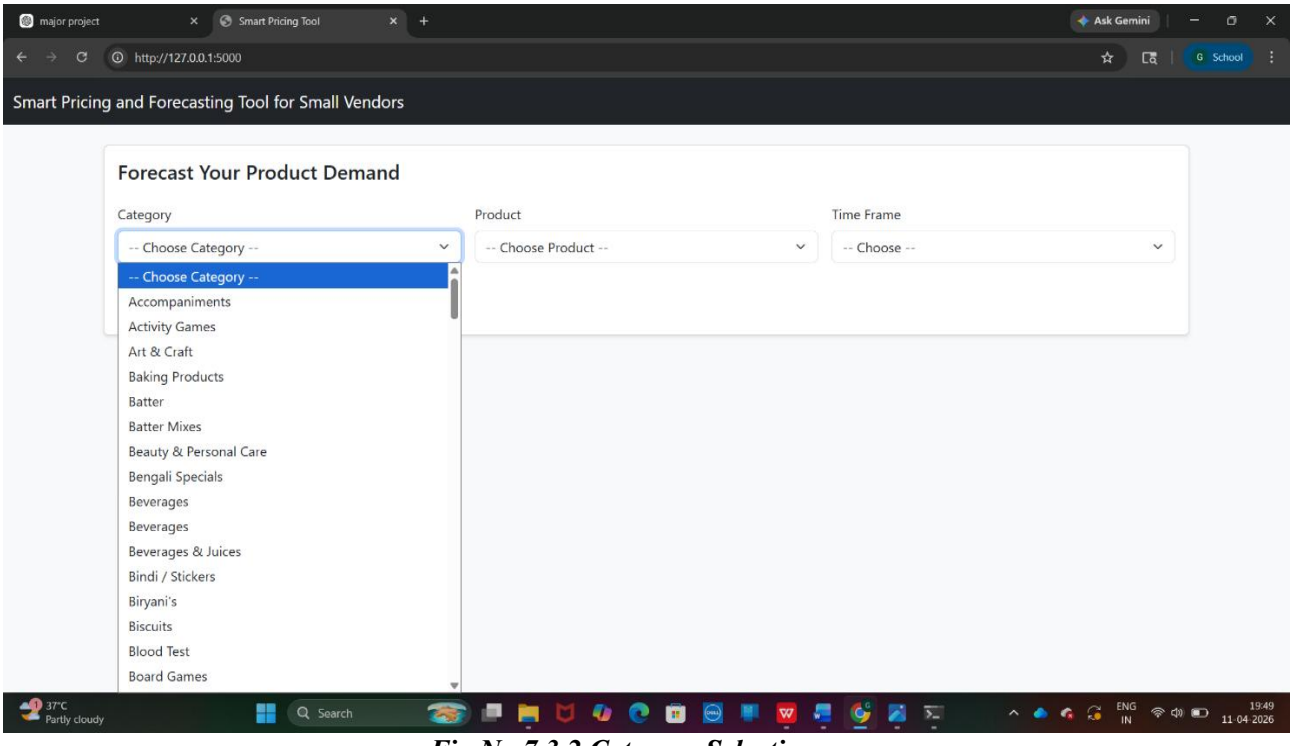

Fig No 7.3.2 Category Selection

Description: This image represents a test case for category selection functionality in the Smart Pricing and Forecasting Tool. The system successfully displays multiple categories in the dropdown and allows the user to select one without errors. The test verifies that category filtering works correctly and the feature passes as expected.

Test Case 2:

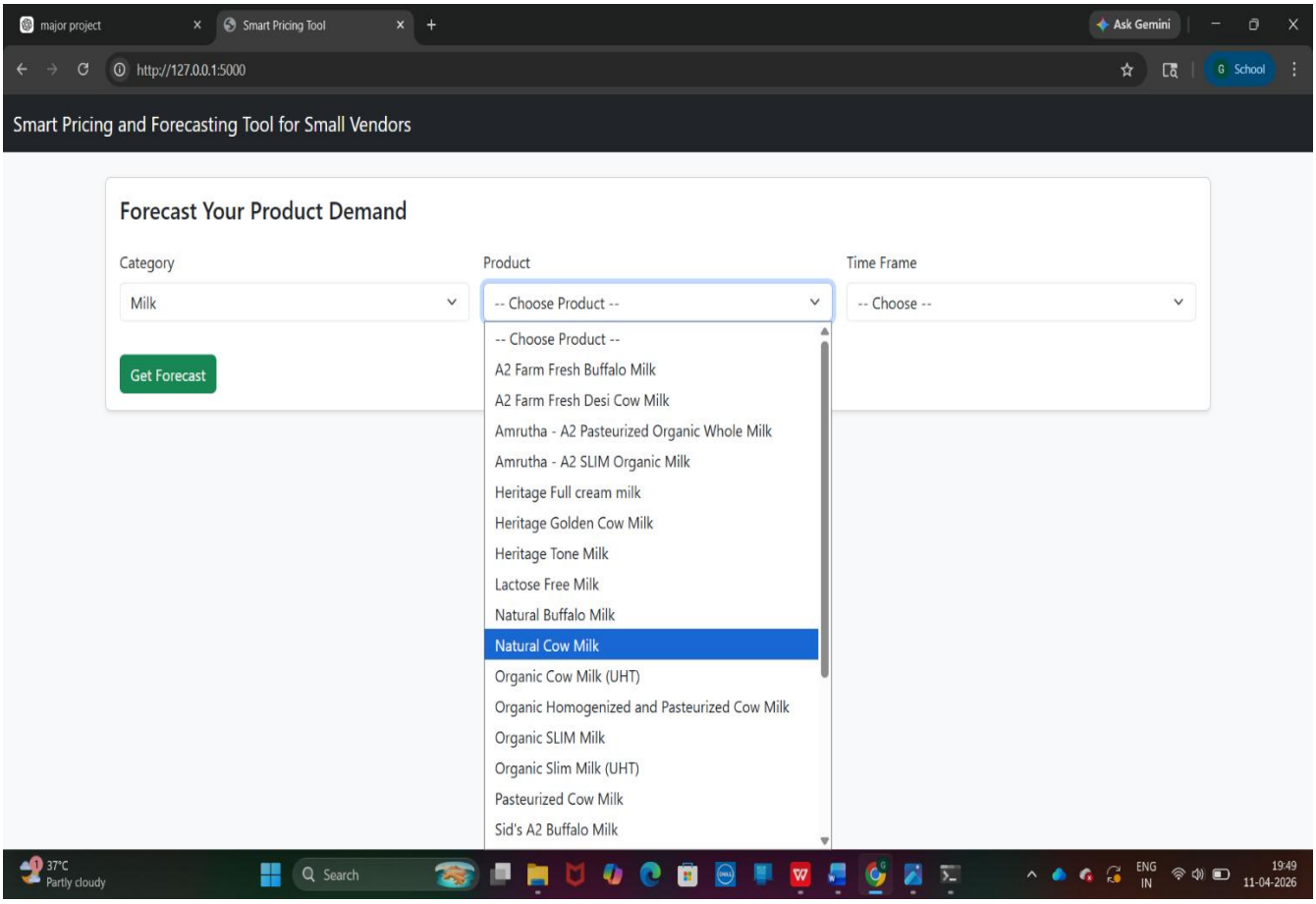

Fig No 7.3.3 Product Selection

Description: This image represents a test case for product selection based on chosen category in the Smart Pricing and Forecasting Tool. The system correctly filters and displays relevant products under the selected category (Milk) in the dropdown. The test confirms proper dynamic loading of products and passes successfully.

Test Case 3:

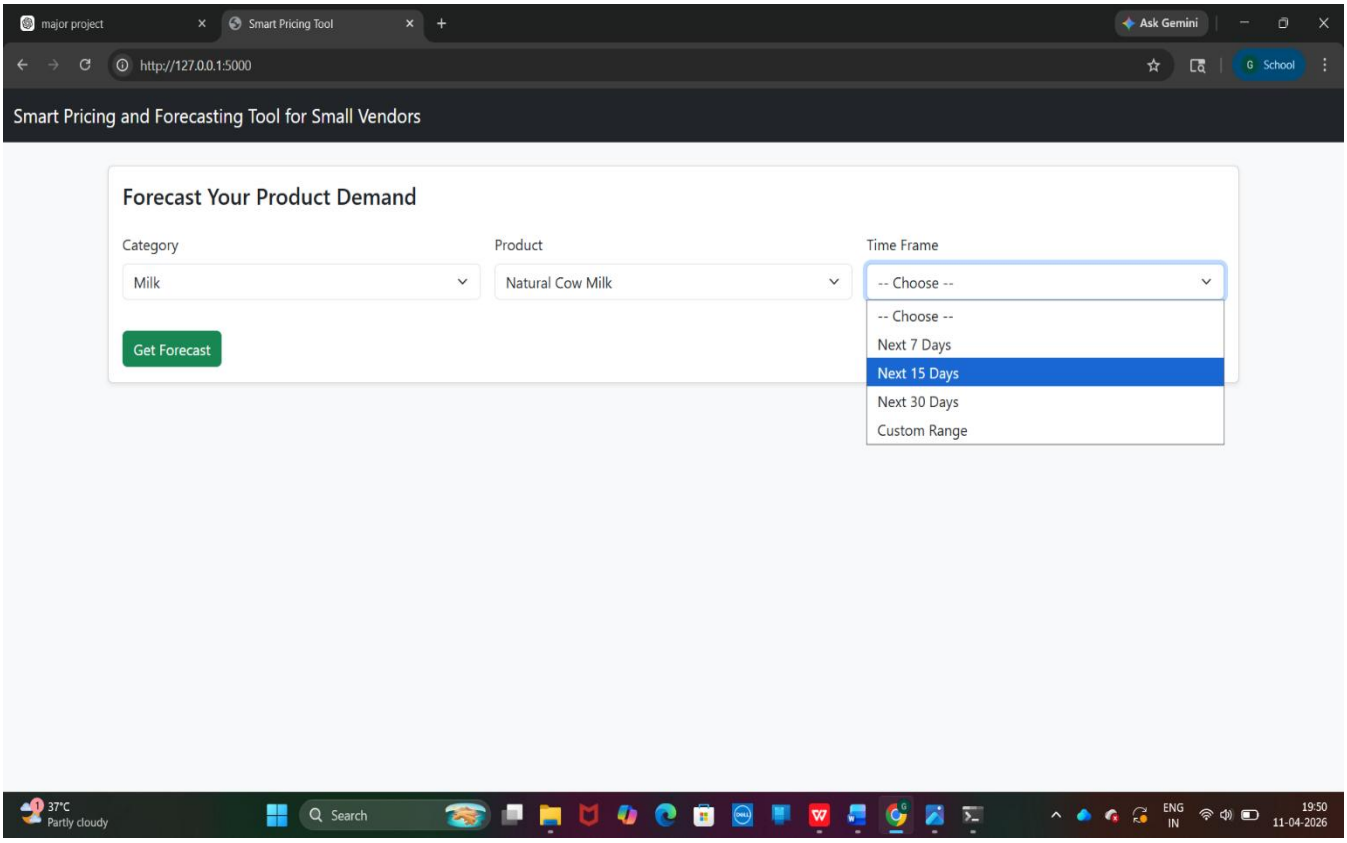

Fig No 7.3.4 Time Frame Selection

Description: This image represents a test case for timeframe selection functionality in the Smart Pricing and Forecasting Tool. The system correctly displays multiple timeframe options (7, 15, 30 days, and custom range) and allows user selection without errors. The test verifies that timeframe input is properly captured for forecasting, and the feature passes successfully.

CHAPTER 8

RESULTS

8. RESULTS

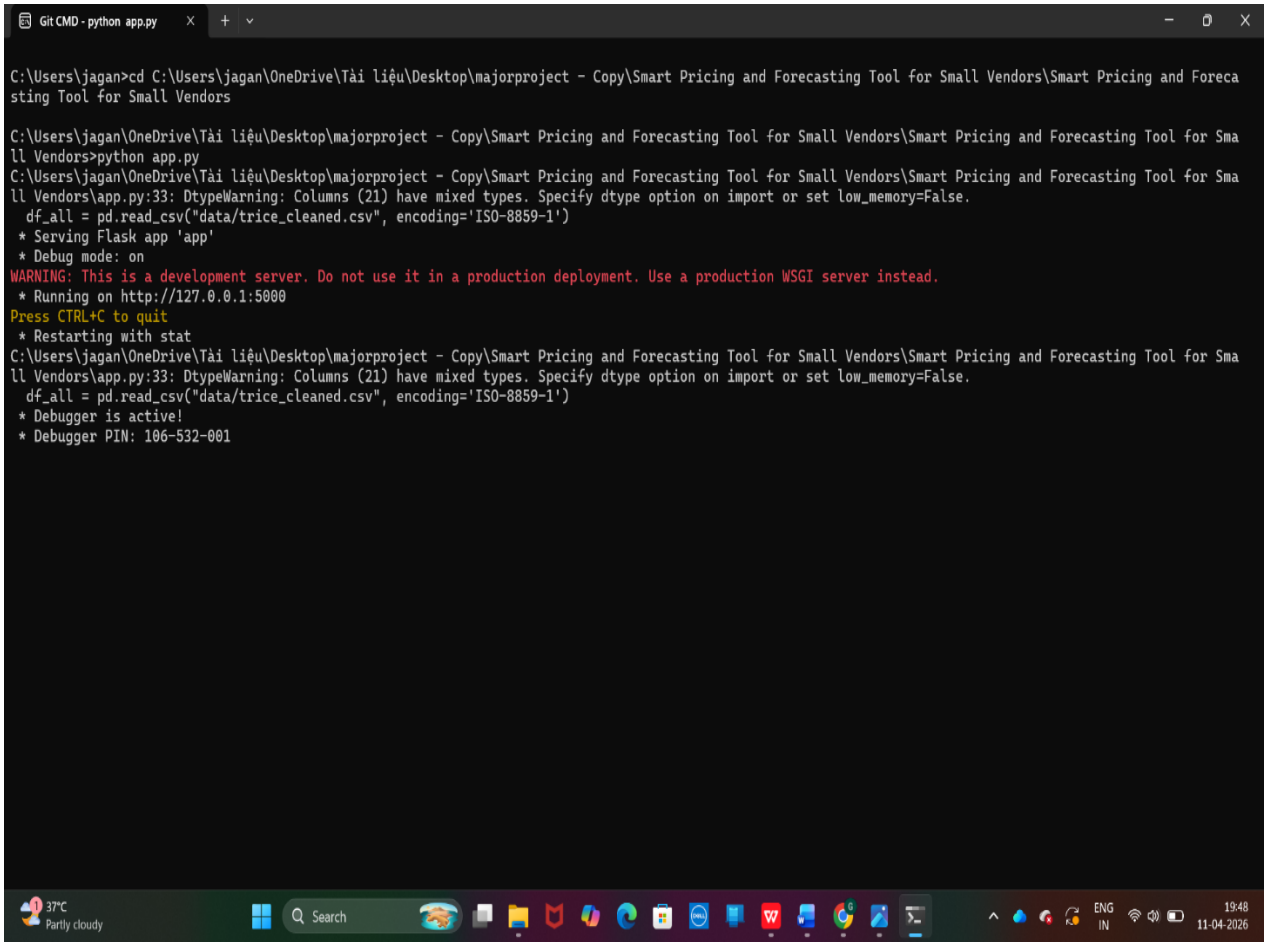

Fig No 8.1 Command Prompt Execution

Description: This image shows the project path being pasted and executed in the Command Prompt to run the Smart Pricing and Forecasting Tool. The Flask server starts successfully after loading the sales dataset from the specified directory. The application runs on a local server (127.0.0.1:5000), ready to process forecasting. requests.

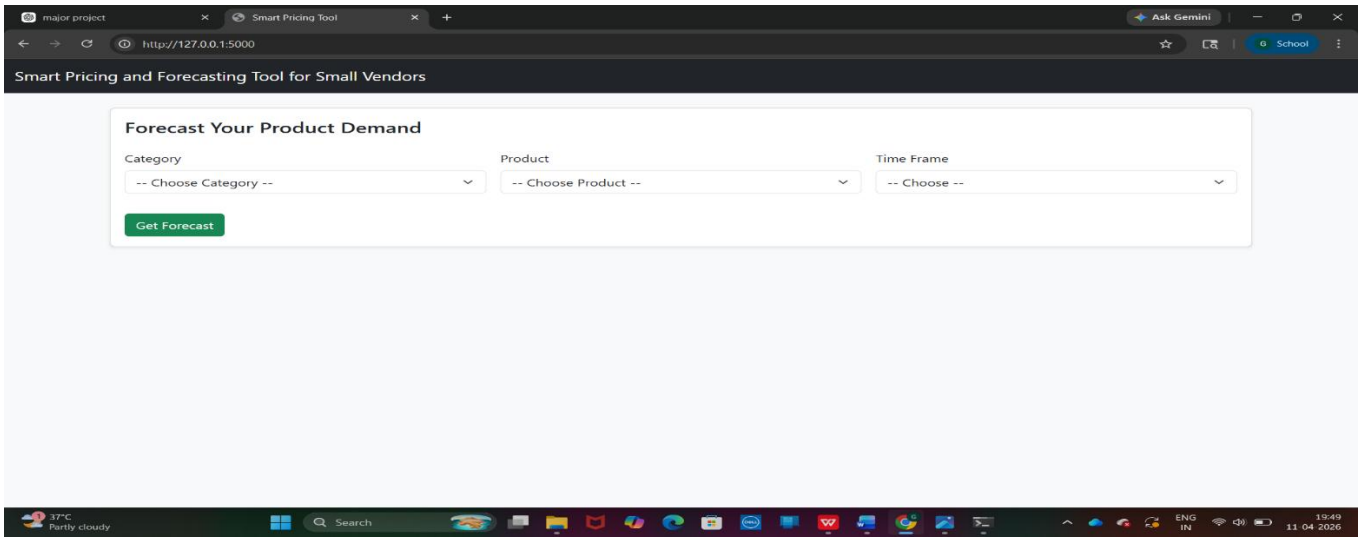

Fig No 8.2 User Interface

Description: This image shows the frontend interface of the Smart Pricing and Forecasting Tool running in a web browser. Users can select category, product, and timeframe to generate demand forecasts. The interface provides a simple and user-friendly way to request and view forecasting results.

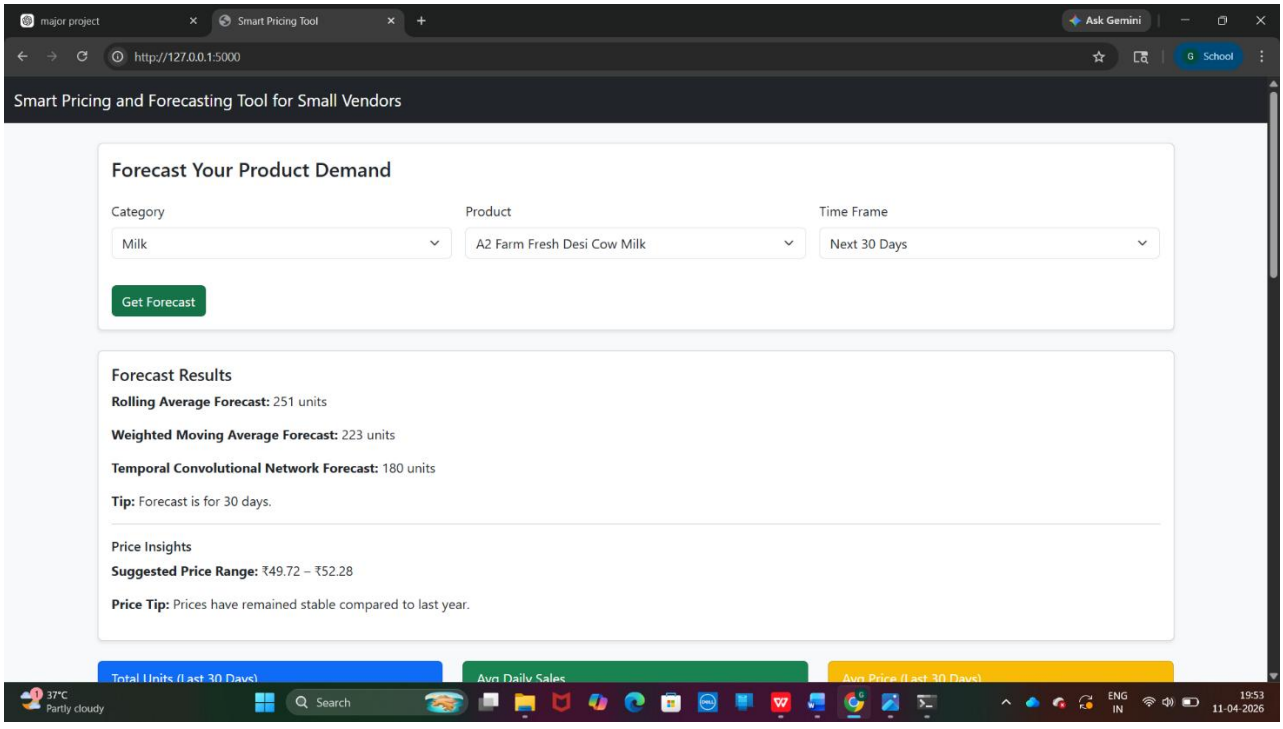

Fig No 8.3 Result Display

Description: This image shows the forecast results generated by the Smart Pricing and Forecasting Tool after selecting product and timeframe. The system displays demand predictions using Rolling Average, Moving Average, and TCN models along with price suggestions. It provides clear insights to help vendors make better inventory and pricing decisions.

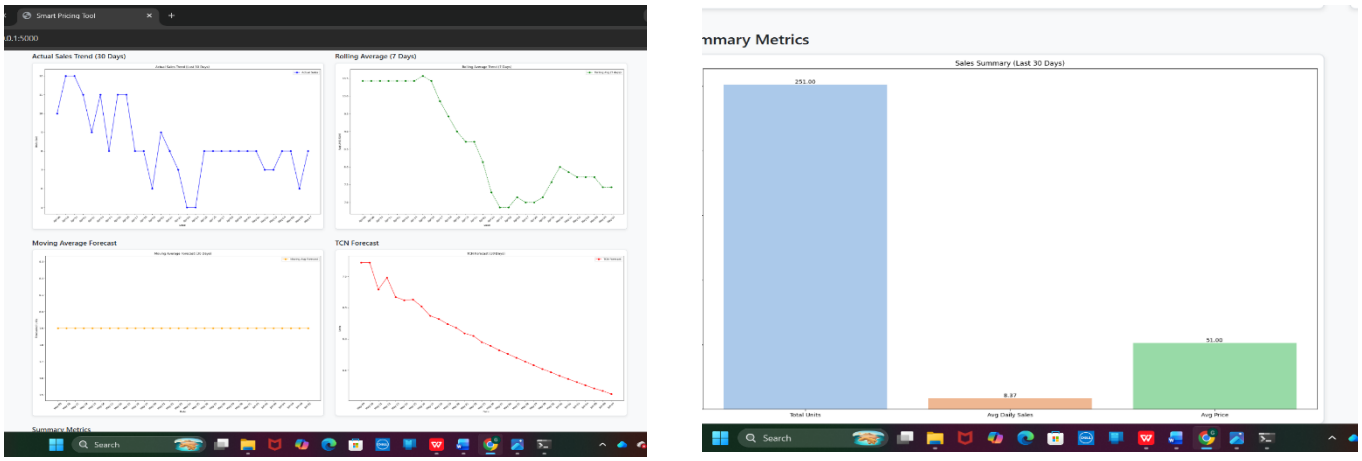

Fig No 8.4 Output Graphs

Description: These images show the graphical outputs generated by the Smart Pricing and Forecasting Tool. They include separate visualizations for actual sales trends, rolling average, moving average forecast, and TCN forecast. The summary metrics chart provides an overview of total sales, average sales, and average price for better analysis.

CHAPTER 9

CONCLUSION

9. CONCLUSION

In conclusion, the Smart Pricing and Forecasting Tool for Small Vendors represents a significant advancement in applying data-driven techniques to real-world business challenges faced by small-scale vendors. The project effectively demonstrates how historical sales and pricing data can be transformed into meaningful insights through systematic data collection, preprocessing, and analysis. By ensuring that the dataset is clean, structured, and consistent, the system lays a strong foundation for accurate forecasting and reliable decision-making.

The integration of statistical models such as Rolling Average (RA) and Weighted Moving Average (WMA) enables efficient short-term forecasting and trend analysis. These models help smooth fluctuations, reduce noise, and provide interpretable results that are easy for small vendors to understand. At the same time, the inclusion of the Temporal Convolutional Network (TCN) introduces a powerful deep learning component capable of capturing complex temporal patterns and long-range dependencies in time-series data. Unlike traditional models, TCN uses causal and dilated convolutions to ensure that predictions are both accurate and logically consistent without relying on future data.

By combining these approaches, the system achieves a balanced framework that leverages both simplicity and advanced learning capabilities. This hybrid structure enhances forecasting accuracy while maintaining interpretability, which is crucial for real-world adoption among non-technical users. Additionally, the project successfully implements a complete machine learning pipeline, including data preprocessing, model training, prediction generation, and result visualization. The use of modern tools such as Pandas, NumPy, TensorFlow, and Scikit-learn ensures efficient computation, scalability, and reproducibility.

From a practical perspective, the system provides valuable support for small vendors by enabling better inventory planning, reducing losses caused by overstocking or understocking, and assisting in optimal pricing decisions. The user-friendly interface and graphical visualizations make it easy to interpret results, allowing vendors to make informed decisions without requiring technical expertise. This

highlights the real-world applicability and impact of the project in improving business efficiency and profitability.

Furthermore, the project demonstrates how machine learning and deep learning can be integrated into simple, accessible applications tailored for small businesses. It bridges the gap between advanced analytical techniques and practical usability, making forecasting tools more inclusive and affordable. The modular design of the system also allows for easy extension and scalability.

Looking ahead, the system can be further enhanced by incorporating real-time data streams, enabling dynamic and continuous forecasting updates. Advanced features such as automated price adjustment, integration with cloud platforms, and connection with enterprise systems like ERP can significantly improve scalability and usability. Additionally, incorporating external factors such as seasonal trends, market conditions, and customer behavior can further improve prediction accuracy.

In conclusion, the Smart Pricing project not only serves as an effective forecasting and decision-support tool but also showcases the potential of combining statistical and deep learning techniques in solving practical business problems. It provides a scalable, efficient, and user-friendly solution that empowers small vendors to adopt data-driven strategies and remain competitive in an evolving market environment.

CHAPTER 10

FUTURE ENHANCEMENTS

10. FUTURE ENHANCEMENTS

One major future enhancement is the integration of real-time data into the Smart Pricing system. While the current implementation relies on historical sales and pricing data, incorporating live data from e-commerce platforms, point-of-sale systems, or online marketplaces would enable dynamic and immediate price adjustments. This would allow businesses to respond quickly to sudden changes in demand, stock levels, or competitor actions, making the pricing strategy more adaptive and effective.

Another area of improvement is the inclusion of external factors such as competitor pricing, seasonal trends, promotional campaigns, and economic indicators. By considering these variables, the model can generate more holistic and realistic price forecasts, ensuring that suggested prices reflect the complex dynamics of the market rather than relying solely on historical trends.

The system can also be enhanced to support automated dynamic pricing. In such a setup, product prices would be adjusted automatically in real time based on forecasted demand, inventory levels, and competitor behavior. This would not only maximize revenue but also optimize inventory turnover, reduce overstocking, and improve overall operational efficiency.

From a technical perspective, future work could involve hybrid and ensemble models, combining TCN with other advanced forecasting techniques like LSTM, GRU, or Prophet. This approach would leverage the strengths of multiple models, improving prediction accuracy and making the system more robust in handling complex and noisy datasets. Additionally, incorporating explainable AI (XAI) features using methods like SHAP or LIME could help users understand the reasoning behind suggested prices, increasing transparency and trust in the system.

Scalability and cloud deployment represent another enhancement path. Hosting the system on cloud platforms would allow it to handle large datasets and multiple product lines efficiently, enable multi-user access, and support continuous updates to the predictive models. Alongside this, developing a

user-friendly dashboard with interactive visualizations, alerts, and scenario simulations would make the system more accessible to business managers, helping them make faster and more informed pricing decisions.

Finally, advanced techniques like reinforcement learning can be explored to create adaptive pricing strategies, where the system learns over time to optimize revenue or profit based on customer response and competitor behavior. Additional expansions could include mobile integration for on-the-go decision-making and multi-product or multi-market optimization to manage pricing strategies across various categories and regions simultaneously. Together, these enhancements would transform the Smart Pricing project into a fully intelligent, scalable, and highly adaptive solution for modern business challenges.

CHAPTER 11

REFERENCES

11. REFERENCES

[1] “Deep Learning for Time Series Forecasting: A Survey” by Kong et al. (2026) provides a comprehensive review of modern deep learning methods, including Temporal Convolutional Networks, for time-series forecasting, summarizing architectures, datasets, and performance across multiple domains.

[2] “A Comparative Study of Temporal Convolutional Network and Gated Recurrent Unit for Predicting Ethereum Prices” by Kiram et al. (2026) compares TCN and GRU models for financial time-series forecasting, demonstrating the advantages of TCN in capturing long-term dependencies in cryptocurrency price prediction.

[3] “TSCND: Temporal Subsequence-Based Convolutional Network with Difference for Time Series Forecasting” by Huang et al. (2026) proposes a variant of TCN that improves feature extraction by focusing on recent subsequences, resulting in more accurate time-series predictions.

[4] “A Novel Dual-Channel Temporal Convolutional Network for Photovoltaic Power Forecasting” (2025) introduces a dual-channel TCN architecture with attention mechanisms to forecast photovoltaic power generation, achieving improved accuracy compared to standard TCN models.

[5] R. Chen and H. Liu, “Short-Term Demand Forecasting Using Machine Learning Techniques,” Expert Systems with Applications, 2025.

[6] M. Zhang, Q. Li, and J. Wang, “Improving Time Series Forecasting Using Attention-Based Deep Learning Models,” IEEE Transactions on Neural Networks and Learning Systems, 2025.

[7] S. Patel and A. Sharma, “Sales Forecasting Using Random Forest Algorithm in Retail Industry,” Procedia Computer Science, 2025.

[8] “Advance Prediction of Coastal Groundwater Levels with Temporal Convolutional and Long Short-Term Memory Networks” by Zhang et al. (2025) demonstrates a hybrid TCN-LSTM model for hydrological forecasting, capturing both short-term and long-term temporal dependencies effectively.

[9] K. Reddy and P. Kumar, “Product Demand Prediction Using Support Vector Machines,” International Journal of Data Science, 2024.

[10] “A Hybrid Temporal Convolutional Network and Prophet Model for Power Load Forecasting” (2024) combines TCN with the Prophet forecasting model to enhance power load predictions, showing improved accuracy and robustness compared to using either method alone.

[11] L. Zhao et al., “Time Series Forecasting with Convolutional Neural Networks: A Review,” IEEE Access, 2024.

[12] J. Torres, D. Martinez, and P. Garcia, “Forecasting Retail Sales Using Machine Learning Models,” Journal of Retail Analytics, 2024.

[13] “Deep Temporal Convolution Network for Time Series Classification” (2023) presents a TCN-basedarchitecture for multivariate time-series classification, emphasizing multi-scale feature extraction and eliminating the need for manual feature engineering.

[14] “TCCT: Tightly-Coupled Convolutional Transformer on Time Series Forecasting” by Li and Wang (2023) introduces a hybrid model that combines convolutional layers and transformer mechanisms to capture both local and long-range dependencies in time-series data.

[15] “SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction” by Liu et al. (2023) proposes SCINet, a convolution-based architecture for time-series forecasting, demonstrating superior performance over traditional convolutional and transformer-based models.

[16] A. Singh and R. Gupta, “A Comparative Study of Machine Learning Models for Demand Forecasting,” International Journal of Forecasting, 2023.

[17] “Temporal Convolutional Networks Applied to Energy-Related Time Series Forecasting” (2022) illustrates the effectiveness of TCN over recurrent models like LSTM in forecasting energy demand, highlighting both higher prediction accuracy and computational efficiency.

[18] B. Kumar and S. Verma, “Sales Prediction Using Regression and Machine Learning Techniques,” Procedia Computer Science, 2021.

[19] H. Nguyen and T. Pham, “Application of Deep Learning Models for Time Series Prediction,” *IEEE Access*, 2020.

[20] Y. Li, R. Yu, C. Shahabi, and Y. Liu, “Diffusion Convolutional Recurrent Neural Network for Traffic Forecasting,” IEEE Transactions on Knowledge and Data Engineering, 2020.