2025

SMART INVENTORY OPTIMIZER

SMART INVENTORY OPTIMIZER FOR LOCAL RETAIL SHOPS

MD SHARJEEL YAZDANI

FEYNN LABS | ASSAM

Table of Contents

SMART INVENTORY OPTIMIZER FOR LOCAL RETAIL SHOPS

[MD SHARJEEL YAZDANI]

[04/05/2025]

ABSTRACT

Small retail shop owners, including kirana stores and local food chains, often struggle with inefficient inventory management, leading to frequent issues such as overstocking, understocking, and losses due to expired goods. These challenges not only impact profitability but also reduce customer satisfaction due to inconsistent product availability. The Smart Inventory Optimizer is a machine learning-driven solution designed to address these issues by providing real-time demand forecasting, automated restocking recommendations, and seamless supplier synchronization. By leveraging predictive analytics, this system minimizes inventory waste, prevents stockouts, and ensures timely replenishment, enabling small retailers to operate more efficiently and profitably.

1. Problem Statement

Small retail shop owners (kirana stores and local food chains) face frequent overstocking, understocking, and expiry-related losses due to lack of real-time demand forecasting and automated restocking. This hurts profitability and customer satisfaction. The Smart Inventory Optimizer leverages machine learning to predict demand, optimize stock levels, and synchronize with suppliers, reducing waste and missed sales.

2. Market/Customer/Business Need Assessment

- **Massive Retail Base:**

  India has over **12 million kirana stores**, which form the backbone of the country's FMCG and grocery distribution. Yet, **70% still rely on manual processes or basic digital tools**, making inventory decisions largely guesswork.

- **High Loss Margins:**

  Small retailers frequently incur **5–10% losses due to expired perishable goods**, directly impacting already thin profit margins. Overstocking also ties up working capital and shelf space, while understocking leads to missed sales.

- **Growing Competition:**

  Organized retail chains and **quick commerce platforms** (e.g., Blinkit, Zepto) are rapidly capturing market share by offering better stock availability and faster delivery, putting pressure on small shops to become more efficient.

- **Demand for Simple AI Tools:**

  There is a clear gap in the market for **affordable, localized AI-powered inventory systems** that work in low-resource environments and require **minimal technical expertise** to use.

3. Target Specifications and Characterization

- **Retail Size & Inventory Complexity:**

  Designed for shops handling anywhere from **100 to 5,000 SKUs (stock keeping unit)**, covering both fast-moving staples and slower-turnover specialty items.

- **Device Ecosystem:**

  Compatible with **low-cost Android smartphones**, entry-level tablets, and desktops commonly available in small stores—no need for high-end infrastructure.

- **User Profile:**

  The target user is typically a **non-technical shop owner or worker** with limited digital literacy. The system provides a **mobile-first, intuitive UI** with regional language support and minimal data entry.

- **Connectivity & Usage Environment:**

  Supports **offline-first operation** with data syncing when internet is available. Features are optimized for **low-bandwidth environments**, ensuring smooth functionality even in semi-urban or rural areas.

---

4. External Search

- Economic Times Retail (https://retail.economictimes.indiatimes.com)
- IBEF Retail India (https://www.ibef.org/industry/retail-india)
- Statista: India Retail Industry Size
- Tracxn: Inventory Management Startups in India

---

5. Benchmarking Alternate Products

Product	Features	Limitations
KhataBook	Billing & basic khata management	No ML-based forecasting
GoFrugal	POS & inventory modules	High cost; complex for small shops
Zoho Inventory	SaaS inventory solution	Not mobile-first; limited local support

---

6. Applicable Patents

- US20200326104A1: AI-driven inventory prediction
- IN201821009556: Automated inventory control system

---

7. Applicable Regulations

• Consumer Goods Packaging & Labeling Rules (India):

Retailers using the Smart Inventory Optimizer often deal with packaged consumer goods. The app must ensure that all inventory inputs and supplier data reflect compliance with **Legal Metrology (Packaged Commodities) Rules**, including correct display of **MRP, expiry dates, net weight/volume, and manufacturer details**. This is critical for accurate inventory tracking and avoiding regulatory penalties during inspections.

• India's Data Protection Bill (Privacy Compliance):

As the platform handles sensitive user data such as store sales history, customer trends, and possibly personal identifiers, it must comply with the provisions of India's **Digital Personal Data Protection (DPDP) Act**. The system incorporates **data encryption, consent-based data sharing, and anonymization techniques**, ensuring that all user data is processed with transparency and integrity, while also providing users the right to access and delete their data.

• GST Transaction Record-Keeping Regulations:

Since the platform may assist retailers in recording purchase and sale transactions, it must support **Goods and Services Tax (GST)** compliance in India. This includes **generating transaction logs, supporting invoice-ready exports**, and maintaining secure digital records that can be readily accessed for audits or filings. Future integration with government e-invoicing portals may further streamline compliance.

---

8. Applicable Constraints

• Low-Connectivity Regions:

The Smart Inventory Optimizer is designed with the realities of small-town and rural retail in mind. Many target users operate in areas with unreliable or limited internet access. To ensure uninterrupted usability, the app supports **offline-first functionality**—allowing shopkeepers to view past data, update stock manually, and receive locally cached recommendations. Data syncs automatically once a stable internet connection is detected, ensuring the system remains functional without requiring constant connectivity.

• Budget Constraint (Target < ₹300/month):

Cost sensitivity is a key factor among small and medium retailers. The solution is built to provide high value at a low cost, with the **premium plan priced at ₹299/month**, aligning with the affordability range of the target segment. The core functionality is accessible for free, while the premium tier offers enhanced features like demand forecasting and supplier sync, allowing retailers to scale usage based on their comfort and business growth.

• Minimal In-House Technical Expertise:

Most of the intended users do not have dedicated IT teams or advanced technical knowledge. Therefore, the app is designed to be **intuitive, low-touch, and highly visual**, with minimal setup or configuration required. Features such as **automated onboarding, guided workflows, voice-assisted inputs**, and **regional language support** ensure usability for non-technical users, making adoption easy and sustainable.

---

9. Business Model

• Freemium Subscription:

The app follows a freemium model to maximize adoption among small retailers.

- **Free Tier:** Offers basic inventory tracking, manual stock entry, and sales monitoring features, supported by in-app advertisements. This tier is ideal for local shopkeepers who are just starting to digitize their operations.
- **Premium Tier (₹299/month):** Unlocks advanced features including AI-powered **demand forecasting**, **automated supplier synchronization**, restocking recommendations, and an **ad-free user interface (UI)** for a smoother, distraction-free experience. This subscription is targeted at growth-oriented retailers looking to optimize operations and increase margins.

• Ad Revenue (Free Tier):

Retailers on the free plan will see **targeted promotional ads** from **Fast-Moving Consumer Goods (FMCG)** brands and distributors etc. These ads are contextually relevant based on store inventory and sales trends, offering suppliers a direct advertising channel to active shopkeepers while generating a steady revenue stream for the platform.

• Commission on Supplier Orders:

When a retailer places an order through the app's **integrated supplier portal**, a **small commission** is charged to the supplier or distributor. This creates a transactional revenue stream tied directly to usage, while also encouraging more suppliers to join the platform in exchange for digital access to a wide retailer base.

---

10. Concept Generation

Research and Problem Identification:

Online sources like articles and case studies were reviewed to understand inventory challenges faced by small retailers—such as overstocking, stockouts, and lack of forecasting.

Idea Development:

A mobile/web app using machine learning for sales analysis and automated restocking emerged as the most practical, user-friendly, and cost-effective solution.

Concept Visualization:

The app's key features and user interface were outlined, focusing on simplicity, clarity, and ease of use.

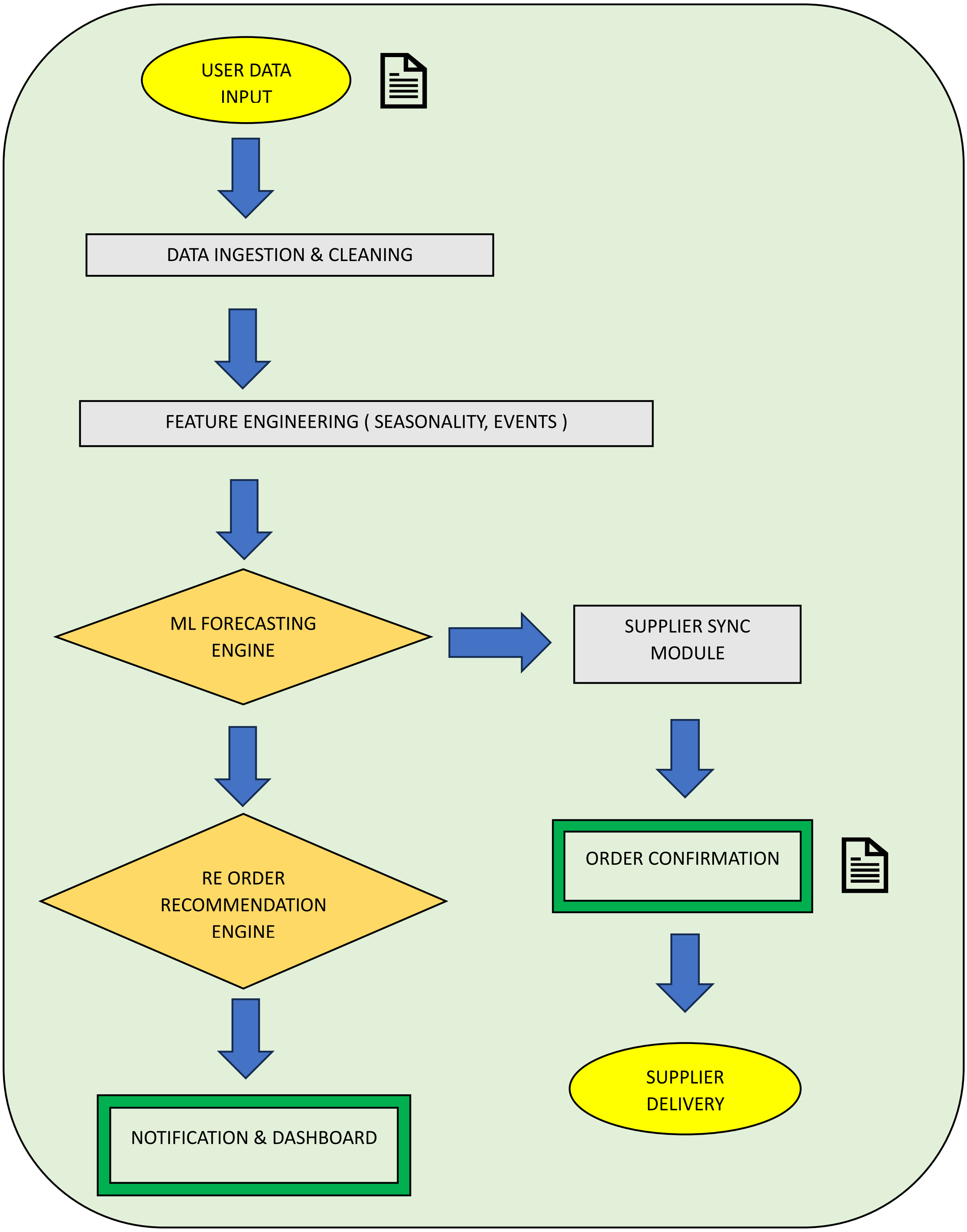
---

11. Concept Development

We are building a mobile and web-based application designed specifically for small and mid-sized retailers. The platform ingests data from Point of Sale (POS) systems or manually uploaded stock sheets, making it accessible to both tech-savvy and traditional store owners. Once the data is collected, it is processed through machine learning pipelines that analyze historical sales patterns, seasonality, and supplier lead times. The system then generates simple, actionable restocking recommendations that require minimal user input. By eliminating guesswork and manual inventory tracking, the app helps retailers maintain optimal stock levels, avoid lost sales due to stockouts, and reduce excess inventory costs—making inventory management smarter and more efficient.

---

12. Final Product Prototype (Abstract) with Schematic Diagram



12.1 FLOW EXPLANATION OF SMART INVENTORY OPTIMIZER

- **Data Ingestion & Cleaning:**

  Raw sales, inventory, and supplier data from POS systems or manual logs are imported and standardized. The system handles missing or inconsistent entries through imputation techniques, ensuring a clean and consistent dataset ready for analysis.

- **Feature Engineering:**

  The system enriches the dataset by incorporating external variables such as local festivals, weather conditions, and nearby events. These factors are encoded into the dataset as features that significantly influence customer demand, allowing the forecasting model to account for real-world context.

- **ML Forecasting Engine:**

  Advanced time-series forecasting models (e.g., ARIMA, Prophet, or LSTM) are applied to predict future demand for each SKU (Stock Keeping Unit) at daily or weekly intervals. The engine dynamically adjusts based on seasonality, trends, and external factors to provide accurate short-term and long-term demand projections.

- **Reorder Recommendation Engine:**

  Based on forecasted demand, current inventory levels, and lead times, the system computes optimal reorder points and order quantities using inventory optimization techniques. This ensures that each item is restocked just in time—reducing both excess stock and the risk of stockouts.

- **Supplier Sync Module:**

  Automatically generates purchase orders and sends them to suppliers based on reorder recommendations. It also tracks order confirmations, delivery timelines, and exceptions, helping shop owners maintain a smooth and reliable restocking process without manual intervention.

- **Notification & Dashboard:**

  A user-friendly dashboard provides real-time insights, including low-stock alerts, upcoming order schedules, expiry risk flags, and a visual comparison of sales vs. waste trends. Notifications can be sent via mobile, email, or in-app alerts, keeping the retailer informed and in control at all times.

---

13. Product Details

• **How it Works:**

The shopkeeper begins by syncing sales and inventory data—either automatically via a Point of Sale (POS) system integration or through manual uploads. Once received, this data flows into the backend machine learning (ML) pipelines, where algorithms forecast future demand, identify low-stock or overstock situations, and generate restock alerts. The frontend user interface (UI) then presents these insights in a clean, user-friendly dashboard. For added efficiency, the system can also trigger automated supplier order requests based on predefined thresholds, reducing manual effort and response time.

• Data Sources:

The core data source is the retailer's historical sales and stock records. This is enriched with external data such as regional market trends, local economic indicators, and public holiday calendars, which influence consumer behavior and purchasing patterns. Together, these inputs enable more accurate and context-aware forecasting.

• Algorithms & Frameworks:

The forecasting engine relies on proven time-series models like Facebook Prophet (for seasonality and trend detection) and Autoregressive Integrated Moving Average (ARIMA) (for baseline statistical forecasting). For real-time mobile inference and lighter deployments, TensorFlow Lite and PyTorch Mobile will be utilized. Feature engineering and model training leverage frameworks like scikit-learn for data preprocessing and baseline modeling.

• Tech Stack:

- Frontend: Built with React Native, ensuring a smooth, cross-platform experience on both Android and iOS.
- Backend: Powered by FastAPI, enabling fast, scalable RESTful API development.
- Database: Uses PostgreSQL, a reliable and powerful open-source relational database.
- Cloud Infrastructure: Deployed on Amazon Web Services (AWS) or Google Cloud Platform (GCP), providing flexibility, scalability, and access to cloud-native artificial intelligence (AI) tools and managed services.

• Team Composition:

- 1 Product Manager (PM): Leads product planning, execution, and stakeholder coordination.
- 1 Machine Learning (ML) Engineer: Develops, tests, and maintains forecasting and analytics models.
- 1 Frontend Developer: Implements mobile and web interfaces with seamless user experience.
- 1 Backend Developer: Manages APIs, system logic, and data integration.
- 1 User Experience (UX) Designer: Designs intuitive, inclusive interfaces tailored to non-technical retailers.

• Estimated Development Cost:

- Minimum Viable Product (MVP): ₹10 lakhs – includes basic POS integration, ML-based restocking alerts, and a mobile/web app interface.
- Full-Scale Solution: ₹25 lakhs – includes advanced ML forecasting, automated supplier orders, support for multiple locations, multi-language support, and detailed analytics dashboards.

14. Code Implementation/Validation on Small Scale

The Smart Inventory Optimizer project enables businesses to forecast sales using ARIMA based on past trends. It uses a clean script-based design and provides day-wise sales predictions with minimal setup. This helps in better stock planning, purchase decisions, and inventory control.

In this prototype we will:

1. Outline the minimal file structure required.
2. List and install necessary Python libraries.
3. Walk through each code component—loading data, training the ARIMA model, forecasting, and displaying results.
4. Demonstrate how to run the prototype and view its output.

14.1 Project Structure

This subsection shows the files that comprise the prototype.

```
smart_inventory_optimizer/
│
├── forecast_model.py    ← ARIMA forecasting logic
├── run_forecast.py     ← Script to execute forecast and display results
└── sample_data.csv     ← Historical sales data (input)
```

14.2 Required Python Libraries

Here we list the libraries you must install before running the prototype.

```
terminal
pip install pandas statsmodels
```

- **pandas** – for data loading and manipulation
- **statsmodels** – for ARIMA modeling and forecasting

14.3 Code Breakdown

In this subsection we examine each code file in turn.

- This subsection provides a detailed look at each prototype component, explaining how it contributes to the forecasting pipeline.
- We begin with forecast_model.py to understand data loading, filtering, and ARIMA model training.
- Next, we examine run_forecast.py for executing the forecast and printing results, and review sample_data.csv as the source of historical input data.

14.3.1 forecast_model.py

Loads historical data, fits an ARIMA model, and returns future sales predictions.

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA

def forecast_sales(product_name,
csv_file=r'C:\Users\yazda\PycharmProjects\PythonProject1\Basics\Programming_Basvanagudi\
SMART INVENTORY OPTIMIZER2\sample_data.csv', days=21):

    # Load data
    df = pd.read_csv(csv_file, parse_dates=['date'])

    # Filter by product
    product_df = df[df['product'] == product_name].copy()
    product_df.sort_values('date', inplace=True)

    # Set date as index
    product_df.set_index('date', inplace=True)
    ts = product_df['units_sold']

    # Train ARIMA model
    model = ARIMA(ts, order=(2,1,2))  # You can tune this
    model_fit = model.fit()

    # Forecast
    forecast = model_fit.forecast(steps=days)
    return forecast.tolist()
```

14.3.2 run_forecast.py

Calls the forecasting function, prints a 21-day forecast, and can be extended for longer horizons.

```
from forecast_model import forecast_sales
import pandas as pd
from datetime import timedelta

# Starting date for forecasting (adjust if needed)
start_date = pd.to_datetime('2023-02-01')


# Call the function to get 30-day predictions
predictions = forecast_sales("ProductA", days=21)

# Prepare forecast data
results = []

for i, value in enumerate(predictions, 1):
    forecast_date = start_date + timedelta(days=i - 1)
    rounded_value = round(value)
    formatted_value = f"{value:.8f}"  # Format to 8 decimal places
    results.append({
        'Day': f'Day {i:02}',  # Pad with leading zero for Day 01, 02, ..., 30
        'Date': forecast_date.strftime('%Y-%m-%d'),
        'Forecasted Units': formatted_value,  # Use formatted_value here with 8 decimal
places
        'Rounded Forecasted Units': rounded_value
    })

# Convert to DataFrame and save to CSV
df_forecast = pd.DataFrame(results)
df_forecast.to_csv('forecast_output.csv', index=False)

# Print to terminal
print(f"{'Day':<10}{'Date':<15}{'Forecasted Units':<25}{'Rounded Forecasted Units'}")
print("-" * 70)  # Adjust line length for better alignment

for result in results:
    print(f"{result['Day']:<10}{result['Date']:<15}{result['Forecasted
Units']:<25}{result['Rounded Forecasted Units']}")
```

14.3.2.1 OUTPUT (run_forecast.py)

```
Day       Date           Forecasted Units         Rounded Forecasted Units
----------------------------------------------------------------------
Day 01    2023-02-01     14.87635153              15
Day 02    2023-02-02     18.64039425              19
Day 03    2023-02-03     15.69417487              16
Day 04    2023-02-04     16.68125209              17
Day 05    2023-02-05     18.03560821              18
Day 06    2023-02-06     14.86461509              15
Day 07    2023-02-07     18.62371347              19
Day 08    2023-02-08     15.73280929              16
Day 09    2023-02-09     16.63563208              17
Day 10    2023-02-10     18.07054033              18
Day 11    2023-02-11     14.85390296              15
Day 12    2023-02-12     18.60605699              19
Day 13    2023-02-13     15.77199282              16
Day 14    2023-02-14     16.59010225              17
Day 15    2023-02-15     18.10477794              18
Day 16    2023-02-16     14.84422059              15
Day 17    2023-02-17     18.58743447              19
Day 18    2023-02-18     15.81170445              16
Day 19    2023-02-19     16.54468683              17
Day 20    2023-02-20     18.13830299              18
Day 21    2023-02-21     14.83557285              15
```

14.3.3 sample_data.csv

Contains the sample sales history used to train and validate the model.

```
date,product,units_sold
2023-01-01,ProductA,10
2023-01-02,ProductA,12
2023-01-03,ProductA,15
2023-01-04,ProductA,13
2023-01-05,ProductA,17
2023-01-06,ProductA,14
2023-01-07,ProductA,16
2023-01-08,ProductA,18
2023-01-09,ProductA,20
2023-01-10,ProductA,19
2023-01-11,ProductA,21
2023-01-12,ProductA,20
2023-01-13,ProductA,23
2023-01-14,ProductA,22
2023-01-15,ProductA,24
2023-01-16,ProductA,23
2023-01-17,ProductA,25
2023-01-18,ProductA,26
2023-01-19,ProductA,28
2023-01-20,ProductA,27
2023-01-21,ProductA,29
2023-01-22,ProductA,30
2023-01-23,ProductA,32
2023-01-24,ProductA,31
2023-01-25,ProductA,33
2023-01-26,ProductA,35
2023-01-27,ProductA,34
2023-01-28,ProductA,36
2023-01-29,ProductA,37
2023-01-30,ProductA,38
```

14.4 Running the Prototype

Step-by-step instructions to execute and view the forecast.

1. **Run the forecast script**:

```
terminal

python run_forecast.py
```

2. **Output :**

SEE SECTION - 14.3.2.1 OUTPUT (run_forecast.py)

14.5 NOTE

Prototype's capabilities and next-step possibilities.

This prototype validates the Smart Inventory Optimizer's core functionality: loading historical sales, fitting an ARIMA model, and producing day-wise forecasts with minimal code. For production, one could extend it to longer horizons, automate parameter tuning, or integrate with a UI or database.

15. Conclusion

The Smart Inventory Optimizer brings the power of AI to small and medium-sized retailers, making advanced technology accessible without the need for large IT teams or complex systems. It transforms raw sales data into clear, actionable insights by automatically analyzing trends, predicting future demand, and identifying slow-moving items. The system streamlines inventory forecasting and automates communication with suppliers, ensuring that the right products are stocked at the right time. This reduces overstock and stockouts, minimizes waste, and improves cash flow. Ultimately, it empowers smaller retailers to operate with the efficiency and precision of large enterprises, helping them stay competitive in a fast-changing retail environment.