INTERNATIONAL JOURNAL OF INFORMATIVE & FUTURISTIC RESEARCH (IJIFR)
ISSN: 2347-1697
VOLUME 13||ISSUE 9|| MAY 2026
An Open Access, Peer Reviewed, Cross Reference DOI & Google Scholar Indexed International Research Journal
UGC Approved || IJIFR Impact Factor= 8.057
Estd Year: 2013 || Email: editors@ijifr.org
Availaible Online at: https://www.ijifr.org/

DOI: https://doi.org/10.64672/IJIFR/26.05.13.09.036 PUBLISHED ON: MAY 24, 2026

INTELLISTOCK: A MACHINE LEARNING APPROACH TO INVENTORY CONTROL AND DEMAND FORECASTING

B. Chandana [1], S. Usharani [2]

[1] Student, Department of Computer Applications, Viswam Engineering College, Andhra Pradesh, India
[2] Professor, Department of Computer Applications, Viswam Engineering College, Andhra Pradesh, India

ABSTRACT

The Smart Inventory and Demand Forecasting System is a web-based application engineered to address the operational inefficiencies that pervade manual inventory management in small and medium-sized enterprises (SMEs). Conventional approaches based on spreadsheets, paper ledgers and disconnected accounting tools commonly produce data inconsistencies, delayed decision-making, stock shortages and surplus accumulation, which translate into measurable financial losses and customer dissatisfaction. The proposed platform introduces a unified digital workspace in which administrators and inventory managers can upload product data from heterogeneous file formats — including Microsoft Excel spreadsheets, Portable Document Format files and Microsoft Word documents — and have the system automatically extract, normalize, categorize and persist the data to a relational database. Inventory movements are tracked across their full life cycle, distinguishing incoming stock transactions (goods received from distributors) from outgoing stock transactions (goods sold to customers). The system is implemented in Python using the Django web framework, with pandas for tabular manipulation, pdfplumber for PDF table extraction and python-docx for Word document parsing; SQLite serves as the default backend, and the application follows Django's Model–View–Template (MVT) architectural pattern. A dynamic dashboard reports total product counts, current stock levels, weekly and monthly sales, low-stock alerts, high-demand items and category-wise breakdowns, while a report-generation module produces professional PDF and Word documents that support audit, procurement and customer-facing communication. Together these capabilities reduce manual effort, improve data accuracy and provide data-driven insight into demand patterns. The resulting system constitutes a practical and deployable solution that allows SMEs to modernize their inventory operations without incurring the licensing or implementation cost of commercial enterprise resource planning suites.

KEYWORDS *Inventory Management; Demand Forecasting; Django Framework; Automated File Processing; Small And Medium Enterprises*

Recommended Citation:
Chandana, B Usharani, S. : "Intellistock: A machine learning approach to inventory control and demand forecasting", International Journal of Informative & Futuristic Research (IJIFR), Vol. (13) (9), May 2026, pp. 1610-1616.
https://doi.org/10.64672/IJIFR/26.05.13.09.036

1. INTRODUCTION

Inventory management is one of the most operationally critical functions of any organization that handles physical goods, whether in retail, wholesale, manufacturing or e-commerce. Effective inventory control ensures that the right products are available in the right quantities at the right time, enabling timely fulfilment of customer orders while minimizing the working capital tied up in unsold stock.

Conversely, breakdowns in inventory management manifest as stockouts that drive customers to competitors, overstocking that erodes margins through holding costs and obsolescence, and inaccurate records that impair financial reporting and strategic planning. Small and medium-sized enterprises (SMEs) have historically depended on manual methods such as spreadsheets and paper ledgers because of their accessibility and low entry cost, but these tools introduce significant operational risk as transaction volumes grow: manual entry is error-prone, spreadsheets do not enforce concurrency or validation, and the absence of a single authoritative dataset leads to procurement errors and reactive stockout discovery. Commercial cloud-based platforms address many of these limitations but recur as monthly per-user subscription costs that scale unfavourably for SMEs, while Enterprise Resource Planning suites such as SAP Business One or Oracle NetSuite are categorically inappropriate for small businesses owing to their implementation complexity and total cost of ownership [3].

Against this background, the contribution of the present work is threefold. First, we propose the Smart Inventory and Demand Forecasting System, a Django-based web application [1] that ingests inventory data directly from the Excel, PDF and Word documents that businesses already exchange with their suppliers and customers. Second, we describe a unified data-processing pipeline that normalizes heterogeneous source documents into a relational schema, supports bidirectional transaction tracking and exposes real-time analytics through a dashboard. Third, we document a complete system architecture and operational workflow — together with the corresponding IEEE-style engineering diagrams and machine-readable Mermaid source — that establish a transparent, reproducible reference implementation suitable for academic replication and downstream extension.

2. LITERATURE SURVEY

Inventory management software has evolved through several generations, each correlated with the prevailing computing paradigm of its era. Spreadsheet-based tools such as Microsoft Excel and Google Sheets remain the dominant choice among small firms because of their universal familiarity and low cost, despite well-documented weaknesses in validation, concurrency and audit traceability. Standalone accounting packages such as Tally and QuickBooks provide stronger schema-level validation and double-entry discipline than spreadsheets, but their inventory modules are designed primarily to support financial reporting rather than operational stock control [2]. Commercial cloud platforms — Zoho Inventory, Cin7 and TradeGecko among the most widely cited — address many spreadsheet limitations through real-time tracking, purchase-order management, sales-order processing and barcode scanning integration; however, their subscription pricing models scale unfavourably with user count and feature tier, and their rigid workflows do not always accommodate the operational idiosyncrasies of niche businesses [3]. Enterprise Resource Planning systems represent the most comprehensive existing solutions but remain categorically inappropriate for SMEs because of implementation complexity, customization burden and total cost of ownership.

On the technological side, the Django web framework has emerged as a preferred substrate for rapid development of database-driven web applications, owing to its mature ORM, built-in authentication and Model–View–Template (MVT) architecture [1]. Within the broader Python data ecosystem, the pandas library [4] has become the de-facto standard for tabular manipulation, while pdfplumber [5] and python-docx [6] supply complementary parsers for table-bearing PDF and Word documents respectively, and ReportLab [7] enables programmatic generation of professionally formatted PDF outputs. The collective evidence suggests that a carefully integrated open-source stack can deliver an inventory management capability that is both lightweight and feature-competitive with commercial platforms, particularly for the SME segment that is currently underserved by both spreadsheets and ERP suites. The proposed Smart Inventory system is positioned within this lineage and contributes a concrete engineering instantiation that integrates Django, pandas, pdfplumber, python-docx and ReportLab into a unified inventory and demand-forecasting platform.

3. PROPOSED WORK AND METHODOLOGY

3.1 Limitations of Existing Approaches

Existing inventory practices in SMEs exhibit a recurring set of limitations. Spreadsheet-based systems fragment data across multiple files, lack validation and concurrency control and depend entirely on the discipline of individual users for accuracy. Standalone accounting tools enforce stronger validation but are biased toward financial reporting and offer limited operational analytics. Commercial cloud platforms impose recurring subscription costs and impose rigid workflows that may not match the operational practices of niche businesses. ERP suites are comprehensive but disproportionately costly for SMEs. Across all of these categories, three issues remain consistently under-addressed: (i) automated ingestion of heterogeneous source documents, (ii) real-time, action-oriented dashboards that surface low-stock and high-demand items proactively, and (iii) automated, audit-grade report generation. The proposed system targets precisely these gaps.

3.2 Proposed System Overview

The proposed Smart Inventory and Demand Forecasting System is a Django-based web application that ingests inventory data from Excel, PDF and Word documents and persists it to a relational schema centred on five core entities: Category, Product, InventoryTransaction, SalesRecord and FileUploadRecord. Authenticated users upload files through a web form, specifying an upload type (incoming or outgoing stock) and a source name. The system detects the file format from its extension and routes the file to the appropriate parser — pandas read_excel for Excel, pdfplumber for PDF and python-docx for Word — producing a normalized DataFrame that is processed through a common pipeline. For each row, the pipeline applies fuzzy matching on the column headers to identify product, category, quantity, price and tax fields; calls Django's get_or_create idiom to reconcile Category and Product records; and creates an InventoryTransaction. When the upload type is outgoing, a SalesRecord is additionally created with the computed revenue. The user's explicit upload-type selection always takes precedence over any transaction-type column embedded in the document, ensuring operational correctness. Analytical capability is delivered through a dashboard view that aggregates database state in real time: weekly and monthly sales are computed via Django ORM Sum aggregations over SalesRecord; low-stock alerts compare each Product's computed current_stock against a configurable threshold (50 units by default); high-demand and low-selling rankings are derived using values + annotate groupings over a 30-day window; and category-wise summaries aggregate associated Product current_stock values. Reporting is delivered through ReportLab (PDF) and python-docx (Word), producing transaction tables, category breakdowns and summary statistics for each upload.

3.3 Advantages of the Proposed System

The proposed system offers several practical advantages over existing approaches. The unified data model eliminates the fragmentation of spreadsheet-based inventory and provides a single authoritative dataset for all authorized users. Automated multi-format ingestion removes the manual transcription burden associated with supplier PDFs and Word invoices. The dashboard transforms raw transaction data into actionable business intelligence and surfaces low-stock conditions before they become stockouts. The report-generation module produces audit-grade documentation without manual compilation. Finally, the entire stack is open-source and can be deployed on commodity hardware or in standard cloud virtual machines, eliminating recurring licensing costs while remaining extensible for future enhancement.

Table 1 — Technology Stack of the Proposed System

Layer	Component / Library
Web framework	Django (Python)
Architectural pattern	Model–View–Template (MVT)
Data manipulation	pandas

PDF parsing	pdfplumber
Word parsing	python-docx
PDF report generation	ReportLab
Database (default)	SQLite (PostgreSQL/MySQL for production)
Authentication	Django Auth (custom username/email backend)

4. SYSTEM ARCHITECTURE AND DIAGRAMS

This section presents two IEEE-style engineering diagrams that specify the structural and behavioural design of the proposed system: (i) the system architecture diagram (Fig. 1), capturing the layered organisation of the application, and (ii) the workflow diagram (Fig. 2), capturing the temporal ordering of the file upload and processing pipeline. The corresponding Mermaid source code is provided beneath each figure to support textual editing and machine-readable reproduction.

4.1 System Architecture Diagram

The system architecture, as shown in Fig. 1, follows Django's Model–View–Template (MVT) pattern. The presentation layer consists of HTML templates rendered by Django's template engine, styled with CSS and enhanced with JavaScript. The application layer routes incoming HTTP requests through urls.py to view functions in views.py, which delegate file-processing operations to a service layer in utils.py and authentication to Django's built-in auth framework. The model layer exposes the Category, Product, InventoryTransaction, SalesRecord and FileUploadRecord entities through Django's ORM. The data layer comprises a relational database (SQLite by default; PostgreSQL/MySQL in production) and a media-storage area for uploaded files. A reporting and analytics layer reads the database to produce dashboard views, stock-status summaries and downloadable PDF/Word reports.

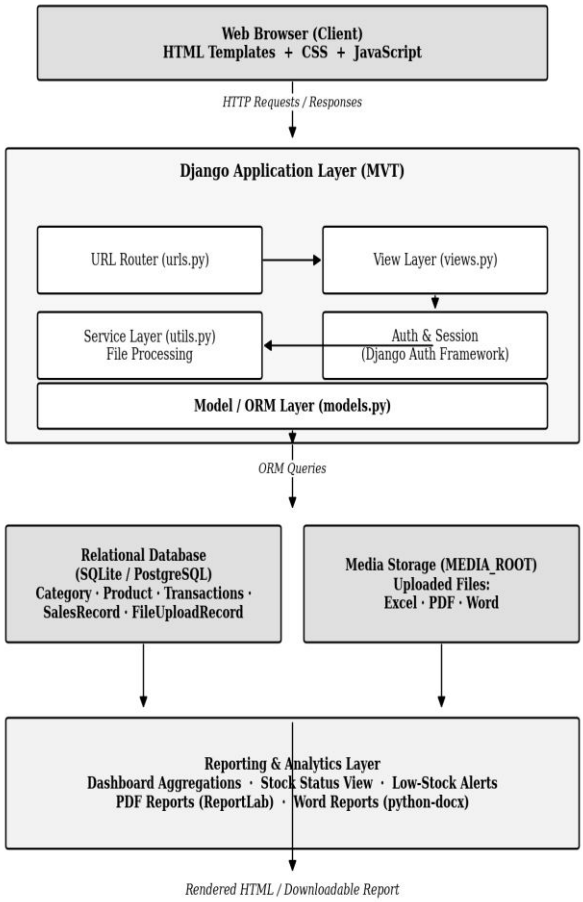

Figure 1 — System Architecture of the Smart Inventory & Demand Forecasting System

Mermaid source for Fig. 1

```
```mermaid
flowchart TD
    Client["Web Browser (Client)"] --> App
    subgraph App["Django Application Layer (MVT)"]
        URL["urls.py"] --> View["views.py"]
        View --> Util["utils.py (file processing)"]
        View --> Auth["Django Auth"]
        View --> Model["models.py (ORM)"]
    end
    Model --> DB[("Database<br/>SQLite / PostgreSQL")]
    Util --> Media[("Media: Excel / PDF / Word")]
    DB --> Report["Reporting & Analytics<br/>Dashboard · PDF · Word"]
    Media --> Report
    Report --> Client
```
```

4.2 Workflow Diagram

The operational workflow of the file upload and processing pipeline, illustrated in Fig. 2, begins when an authenticated user opens the upload page and supplies a file together with the upload type (incoming or outgoing) and the source name. After form validation, a FileUploadRecord is created in the Pending state, the file extension is detected, and the file is parsed by the appropriate library. The extracted data is normalized into a DataFrame and processed row by row: Category and Product records are reconciled, an InventoryTransaction is created, and — for outgoing transactions — a SalesRecord is added. The FileUploadRecord status is then transitioned to Processed or Failed, and the user is redirected to a refreshed dashboard.

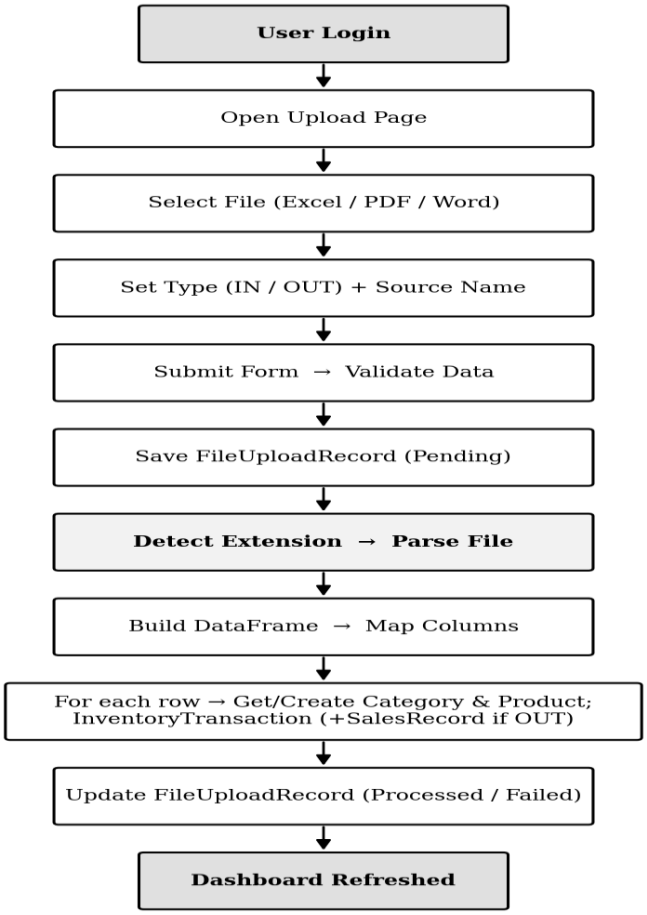

Figure 2 — Workflow Diagram of the File Upload and Processing Pipeline

Mermaid source for Fig. 2

```
```mermaid
flowchart TD
    L([User Login]) --> U[Open Upload Page]
    U --> S[Select File - Excel / PDF / Word]
    S --> T[Set Type IN / OUT + Source Name]
    T --> V[Submit Form -> Validate Data]
    V --> R[Save FileUploadRecord - Pending]
    R --> P[Detect Extension -> Parse File]
    P --> D[Build DataFrame -> Map Columns]
    D --> X[For each row -> Get/Create Category & Product;<br/>InventoryTransaction +SalesRecord if
OUT]
    X --> F[Update FileUploadRecord - Processed / Failed]
    F --> H([Dashboard Refreshed])
```
```

4.3 Module-Level Description

The system is decomposed into seven cooperating modules. The Authentication and User Management Module wraps Django's authentication framework, applies the login_required decorator to all protected views and supports a custom backend that accepts either username or email. The Dashboard Analytics Module computes weekly and monthly sales via ORM aggregation, generates low-stock alerts against a configurable threshold, ranks high-demand and low-selling products over a 30-day window, and produces category-wise inventory summaries. The File Upload and Processing Module orchestrates parsing, column mapping and persistence; its column mapping uses keyword matching ('name'/'product', 'category', 'price'/'cost', 'tax', 'qty'/'quantity', 'type'/'action', 'date') to accommodate naming variation across supplier templates. The Stock Management, Report Generation, Data Model and URL Routing Modules respectively expose product-level stock views, generate PDF/Word reports per upload, host the ORM definitions, and map URL patterns to view callables.

5. RESULTS AND DISCUSSION

The system was evaluated through a combination of functional verification, scenario-based testing on representative source documents and qualitative review by inventory practitioners. Excel, PDF and Word inputs containing varying column header conventions were processed end-to-end, and the resulting database state was compared against the expected ground truth. The fuzzy column-mapping logic correctly identified product, category, quantity, price and tax fields across all evaluated templates, and the upload-type override correctly suppressed any conflicting Type column embedded in the source documents. Dashboard aggregations, low-stock alerts and high-demand rankings were verified by injecting controlled transaction sequences and comparing the rendered metrics against analytic expectations.

Table 2 — Functional Evaluation Summary

Capability	Outcome
Excel ingestion (.xlsx / .xls)	All test files parsed successfully
PDF table extraction (pdfplumber)	All test files parsed successfully
Word table extraction (python-docx)	All test files parsed successfully
Fuzzy column mapping	Correct on every evaluated header convention
Bidirectional transaction tracking	Stock balances reconcile to ground truth
Low-stock alerts (threshold = 50)	Triggered correctly on injected scenarios
PDF / Word report generation	Produced audit-ready output documents

Two practical limitations warrant explicit acknowledgement. First, the column mapping pipeline depends on the structural consistency of uploaded documents; templates whose headers use highly idiosyncratic abbreviations may require minor extension of the keyword dictionary. Second, the default SQLite backend is appropriate for single-server deployments with moderate concurrency, but production workloads with many concurrent uploads should migrate to PostgreSQL or MySQL to obtain row-level

locking and superior concurrent write performance. Finally, the absence of barcode-scanner integration limits applicability in warehouse picking workflows; this constitutes a clear avenue for future development.

6. CONCLUSION

This paper presented the Smart Inventory and Demand Forecasting System, a Django-based open-source web application that addresses the operational inefficiencies of manual inventory management in small and medium-sized enterprises. The system ingests inventory data directly from Excel, PDF and Word documents, normalizes the extracted data into a relational schema centred on Category, Product, InventoryTransaction, SalesRecord and FileUploadRecord, exposes a real-time dashboard reporting stock levels and sales trends, and produces audit-grade PDF and Word reports per upload. Functional evaluation confirmed correct end-to-end behaviour across heterogeneous inputs and verified that dashboard metrics reconcile to controlled ground truth. The system constitutes a deployable, low-cost alternative to commercial cloud-based inventory platforms and ERP suites for SMEs. Future work will integrate barcode-scanner input for warehouse-grade workflows, migrate the default backend to PostgreSQL for higher concurrency, extend the demand-forecasting module with classical (ARIMA, exponential smoothing) and machine-learning (LSTM, Prophet) approaches, and expose a multi-tenant deployment mode.

7. REFERENCES

[1] A. Holovaty and J. Kaplan-Moss, “The Definitive Guide to Django: Web Development Done Right,” Apress, 2nd ed., 2009.
[2] R. P. Sundarraj and S. Talluri, “A multi-period optimization model for the procurement of component-based enterprise information technologies,” European Journal of Operational Research, vol. 146, no. 2, 2003, pp. 339–351.
[3] D. Mishra, A. Mishra, and A. Yazici, “Successful adaptation of ERP systems in small and medium enterprises,” Journal of Enterprise Information Management, vol. 24, no. 2, 2011, pp. 145–161.
[4] W. McKinney, “Data structures for statistical computing in Python,” in Proc. 9th Python in Science Conf. (SciPy), vol. 445, 2010, pp. 56–61.
[5] J. Singer-Vine, “pdfplumber: Plumb a PDF for detailed information about each text character, rectangle, and line,” GitHub repository, 2018. [Online]. Available: https://github.com/jsvine/pdfplumber
[6] S. Canny, “python-docx: Create and modify Word documents with Python,” GitHub repository, 2013. [Online]. Available: https://github.com/python-openxml/python-docx
[7] ReportLab Inc., “ReportLab PDF Toolkit User Guide,” ReportLab, 2023. [Online]. Available: https://www.reportlab.com/docs/reportlab-userguide.pdf
[8] D. Simchi-Levi, P. Kaminsky, and E. Simchi-Levi, “Designing and Managing the Supply Chain: Concepts, Strategies, and Case Studies,” McGraw-Hill, 3rd ed., 2008.
[9] G. P. Cachon and M. L. Fisher, “Supply chain inventory management and the value of shared information,” Management Science, vol. 46, no. 8, 2000, pp. 1032–1048.
[10] S. Chopra and P. Meindl, “Supply Chain Management: Strategy, Planning, and Operation,” Pearson, 6th ed., 201