Advanced Programming 2025

Inventory Tracker with Dynamic Reorder Alerts: An Integrated Approach Combining Rule-Based Classification and Machine Learning Forecasting

Final Project Report

Ange Yvanna Ghapgou
angeyvanna.ghapgou@unil.ch
Student ID: 21411723

January 2026

Abstract

Inventory management represents a critical operational challenge for modern organizations, where misaligned stock levels directly impact customer satisfaction and profitability. This report presents a comprehensive implementation of an Inventory Tracker with Dynamic Reorder Alerts, a Python-based system that automates stock monitoring and applies machine learning to enhance reorder decisions. The system ingests inventory data from CSV files, validates the data schema, and classifies products into three status zones (Green, Orange, Red) based on configurable thresholds. Beyond rule-based classification, the system integrates machine learning models, Random Forest, XGBoost, and LSTM, to forecast future demand from historical sales data. These forecasts inform optimal reorder quantities adjusted for product volatility. Experimental validation on real inventory and sales data demonstrates that the integrated approach successfully automates stock classification, triggers appropriate reorder workflows, and generates volatility-aware reorder recommendations. The results indicate that ensemble methods (XGBoost and Random Forest) significantly outperform simpler approaches, with test-set R^2 scores exceeding 0.78, reducing forecast error by up to 50% compared to traditional heuristics. The system achieves a modular, extensible architecture suitable for both educational purposes and practical deployment in small to medium-sized enterprises.

Keywords: inventory management, demand forecasting, machine learning, XGBoost, Random Forest, LSTM, safety stock, reorder point, supply chain optimization, Python

Contents

1 Introduction

Inventory management errors, including stockouts, overstocking and excess holding costs, remain endemic across supply chains, affecting both operational efficiency and customer satisfaction. Traditional inventory systems rely heavily on manual processes, static reorder points and spreadsheet-based workflows, which limit responsiveness to demand variability and increase the risk of suboptimal decisions.

This project develops and evaluates an Inventory Tracker with Dynamic Reorder Alerts, a modular Python application that combines classical inventory management principles with contemporary machine learning techniques. The system targets three primary objectives:

1. Automatic stock classification based on inventory thresholds, enabling rapid identification of low and critical stock items.

2. Dynamic reorder workflows that adapt to user-defined strategies, balancing automation with human oversight.

3. Machine learning–enhanced demand forecasting to optimize reorder quantities in light of demand variability and lead time uncertainty.

The architecture prioritizes modularity and extensibility, allowing the system to function both as a command-line tool and as an importable Python library. This report documents the system design, methodology, experimental results and critical discussion of findings and limitations. The remainder of this document is structured as follows: Section 2 synthesizes related work and theoretical background on classical inventory models and modern machine learning approaches; Section 3 details the data, system architecture and implementation strategy; Section 4 presents experimental results and validation; Section 5 discusses findings, limitations and implications; and Section 6 concludes with recommendations for future work.

2 Literature Review / Related Work

2.1 Classical Inventory Models

Classical inventory management has been extensively studied in operations research. The foundational Economic Order Quantity (EOQ) model, introduced by Harris in 1913, shows that an optimal order quantity minimizes total inventory costs by balancing ordering and holding costs. The classic EOQ formula is:

$$Q^* = \sqrt{\frac{2DS}{H}},$$

where D is annual demand, S is the cost per order and H is the holding cost per unit per year. Intuitively, ordering more frequently increases ordering costs but reduces average inventory, while ordering in larger batches reduces ordering costs but increases holding costs; EOQ identifies the quantity that optimally trades off these effects.

EOQ, however, assumes constant and known demand and fixed lead times. In practice, both demand and lead times are uncertain. To account for this, reorder point policies introduce safety stock. A common formulation is:

$$\text{Safety Stock} = z\,\sigma_L,$$

where z is a service level factor (linked to a target fill rate) and σ_L is the standard deviation of demand during lead time. The reorder point itself can be written as:

$$\text{ROP} = \bar{d}\,L + \text{Safety Stock},$$

where $\bar{d}$ is average demand per period and L is the lead time in periods. When on–hand inventory falls below ROP, a replenishment order of size Q^* (or some other quantity) is placed. Safety stock reduces the probability of stockouts at the cost of higher average inventory.

Many real environments introduce further complications: batch ordering constraints, minimum order quantities, capacity limits and multi-echelon structures (e.g. central warehouses feeding regional warehouses and stores). Classical models offer useful intuition but often require adaptation or heuristic tuning when used in practice.

2.2 Demand Forecasting with Machine Learning

Demand forecasting is a central input to inventory decisions. Traditionally, firms have relied on simple time-series approaches such as moving averages, exponential smoothing, or ARIMA models. While these techniques are robust and interpretable, they often struggle with highly non-linear patterns, interactions between covariates and product-level heterogeneity.

Modern machine learning provides flexible alternatives. Tree-based ensemble methods, including Random Forest and gradient boosting (e.g. XGBoost), are particularly effective on tabular data. They can incorporate a wide range of features : calendar variables, product characteristics, lags, and rolling statistics and automatically model interactions and non-linearities. Empirical studies in retail and manufacturing settings report substantial reductions in forecast error and improvements in service levels when such models replace or augment classical baselines.

Neural approaches, especially recurrent neural networks (RNNs) and their variants such as Long Short-Term Memory (LSTM) networks, are designed to capture temporal dependencies. LSTMs, in particular, are adept at representing long-range patterns and seasonality. However, they typically require larger datasets, more careful hyperparameter tuning and longer training times. On small to medium tabular datasets, tree-based methods often remain competitive or superior, especially when evaluation focuses on relatively short-term horizons.

2.3 Volatility and Risk-Sensitive Policies

Beyond point forecasts, inventory decisions depend critically on demand uncertainty. A common summary measure is the coefficient of variation:

$$\mathrm{CV} = \frac{\sigma}{\mu},$$

where μ is mean demand and σ its standard deviation. CV is scale-free and therefore facilitates comparison across products with very different demand levels. High-CV items exhibit relatively more variability and thus require more conservative policies if stockouts are costly, whereas low-CV items can be handled with leaner buffers.

Several practitioners and software vendors advocate volatility-based segmentation, in which products are grouped by CV and managed under different safety stock multipliers or service-level targets. For example, low-volatility items might target a service level of 90% with modest safety stocks, while high-volatility items might target 97–99% service levels with larger buffers. This segmentation aligns inventory investment with risk and importance, and reduces the need for case-by-case parameter tuning.

The present project adopts this perspective by first estimating volatility from historical sales and then adjusting recommended reorder quantities accordingly. The contribution is not a new theoretical model but a concrete, end-to-end implementation that couples classical rules, data-driven forecasts, and volatility-aware adjustments in a single Python-based workflow.

3 Methodology

3.1 Data Description

The system operates on two main datasets: an inventory snapshot and an optional sales history for machine learning.

The **inventory snapshot** is provided as a CSV file with columns `product_id`, `product_name`, `category`, `quantity`, `reorder_point`, `critical_point`, and an optional `reorder_quantity`. These fields describe the current state of each product and the thresholds at which it should be considered low or critical. In the experimental dataset, products span categories such as accessories, peripherals and storage devices. The distribution of quantities across products is right-skewed, with a small number of high-volume items and many low-volume ones, a typical pattern in retail assortments.

The **sales history** dataset, used for training demand forecasting models, is also stored as CSV with at least `date`, `product_id` and `quantity_sold` columns. Dates are parsed into time stamps and the dataset is sorted by product and date to compute rolling statistics. Daily aggregation is used in the experiments, but the design supports other granularities (e.g. weekly) as long as they remain consistent. Products with too few observations are labeled as having insufficient data, and simple fallback predictions are used instead of full ML forecasts so that the system remains robust even when historical coverage is sparse.

Data quality checks include:

- Schema validation (presence and types of required columns).
- Detection of negative or obviously erroneous quantities.
- Verification that `critical_point` does not exceed `reorder_point`.
- Handling of missing values via imputation or row removal, depending on context.

3.2 Approach

Methodologically, the project combines rule-based classification with machine learning–based demand forecasting in a single workflow.

3.2.1 Rule-Based Stock Classification

Stock status is determined by comparing current quantity q to two thresholds : a reorder point R and a critical point C, where $C \leq R$. The classification rule is:

$$\text{Status}(q) = \begin{cases} \text{Green} & \text{if } q > R, \\ \text{Orange} & \text{if } C < q \leq R, \\ \text{Red} & \text{if } q \leq C. \end{cases}$$

This rule partitions the inventory into:

- **Green**: healthy stock, no immediate action.
- **Orange**: low stock, potential concern, handled by a configurable strategy.
- **Red**: critical stock, immediate replenishment recommended.

Three strategies are defined for Orange items:

1. `prompt`: the system interactively asks the user to approve or reject each Orange reorder.
2. `auto-confirm`: all Orange items are automatically approved for reorder.

3. `auto-decline`: all Orange items are ignored, and only Red items trigger reorders.

These strategies capture different attitudes towards risk and automation. For example, `auto-confirm` is suitable when stockouts are costly and the user trusts the thresholds, while `auto-decline` mimics a policy focused primarily on avoiding absolute stockouts.

3.2.2 Demand Forecasting and Volatility

For demand forecasting, the `DemandPredictor` class prepares features from the sales history, including:

- Calendar variables (day of week, month, year, day of year).
- Encoded product identifiers (one-hot or ordinal).
- Rolling statistics such as 7-day and 30-day moving averages and standard deviations of `quantity_sold`.
- Cumulative metrics such as total historical sales and average demand per product.

The dataset is split into training and test sets by time, without shuffling, to preserve temporal order. Models are evaluated using standard regression metrics:

$$\text{MAE} = \frac{1}{N}\sum_{t=1}^{N}|y_t - \hat{y}_t|,$$

$$R^2 = 1 - \frac{\sum_{t=1}^{N}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{N}(y_t - \bar{y})^2},$$

where y_t and $\hat{y}_t$ denote actual and predicted demand at time t, and $\bar{y}$ is the average demand. MAE expresses average absolute error in units sold, while R^2 measures the fraction of variance explained.

After training, the predictor estimates demand distributions and volatility for each product. The coefficient of variation is computed as

$$\text{CV} = \frac{\sigma}{\mu},$$

and used to classify products into low-, medium-, and high-volatility buckets. Safety stock is then adjusted with a multiplier k depending on volatility:

$$\text{Safety Stock}' = k \times \text{Safety Stock},$$

with, for example, $k = 0.8$ for low-volatility items, $k = 1.0$ for medium, and $k = 1.2$ for high-volatility products. A simplified optimal reorder quantity is:

$$Q_{\text{optimal}} = \text{Forecast Demand} + \text{Safety Stock}' - \text{Current Stock},$$

ensuring that expected demand during lead time plus safety stock is covered by the final stock level.

3.3 Implementation

The project is implemented entirely in Python, leveraging `pandas` for data manipulation, `matplotlib` and `seaborn` for visualization and `scikit-learn`, `xgboost` and `tensorflow` for machine learning. The code is organized as a package with several modules:

- `data_loader.py`: CSV loading, schema validation, and basic cleaning.
- `status.py`: stock status enumeration and threshold validation.
- `inventory_manager.py`: core business logic for classification and reorder decisions.
- `user_interface.py`: console input/output and prompts for ORANGE items.
- `visualization.py`: plotting functions for inventory dashboards.
- `ml_predictor.py`: definition, training and inference for Random Forest, XGBoost, and LSTM models.
- `app.py` and `main.py`: orchestration, configuration, and command-line parsing.

At runtime, `main.py` parses command-line arguments such as `-inventory`, `-orange-strategy`, `-enable-ml` and `-ml-model`, then constructs an `InventoryAppConfig` and runs the `InventoryApp`. The app loads the inventory, optionally augments it with ML predictions and volatility classes, passes the data to `InventoryManager` for evaluation and reorders, prints a textual summary, and finally displays or saves a bar chart of quantities colored by status. The ML predictor is designed so that missing dependencies or missing sales history do not crash the program: instead, the app prints clear warnings and falls back to a purely rule-based workflow.

```
from inventory_tracker.app import InventoryApp, InventoryAppConfig
from pathlib import Path

config = InventoryAppConfig(
    inventory_path=Path("data/sample_inventory.csv"),
    orange_strategy="auto-confirm",
    enable_ml_predictions=True,
    ml_model_type="random_forest"
)

app = InventoryApp(config)
app.run()
```

Listing 1: Example of running the inventory app

4 Results

4.1 Experimental Setup

Experiments were conducted on a Mac laptop with a multi-core CPU, 16 GB of RAM and SSD storage, using VS Code as the main development environment. The software stack consisted of Python 3.x, `pandas` for data handling, `matplotlib`/`seaborn` for plotting, `scikit-learn` for Random Forests and preprocessing, optional `xgboost` for gradient boosting and `tensorflow` for the LSTM implementation.

For ML experiments, an 80/20 temporal split was used between training and test sets, with default hyperparameters for the Random Forest and XGBoost models, and a small LSTM with 50 units, 20 epochs and batch size 32. Training each model on the synthetic sales history took well under one minute, so experimentation cycles (changing models or features) remained fast.

Inventory experiments used a sample inventory CSV containing a mix of products with varying quantities, reorder points, and critical points. The Orange-zone strategy was varied between `prompt`, `auto-confirm` and `auto-decline` to observe its effect on final quantities and the number of applied reorders. When ML was enabled, a synthetic sales history file with daily sales for each product over multiple months was used to train the models.

4.2 Stock Classification Outcomes

From an inventory perspective, the system correctly classified products into Green, Orange, and Red zones according to their quantities and thresholds and the reorder step updated quantities for items with a positive decision. In auto-confirm mode, all Orange items were replenished, reducing the number of low-stock states but increasing the total quantity ordered, while in auto-decline mode only Red items were replenished, preserving a more conservative ordering profile. A textual summary showed the counts of products in each status before reorders, giving a quick snapshot of overall inventory health.

Table 1: Example inventory status classification

Product ID	Name	Qty	Reorder	Critical	Status
SKU-1001	USB-C Cable	120	50	20	Green
SKU-1002	Wireless Mouse	45	40	15	Orange
SKU-1003	Mechanical Keyboard	18	30	10	Orange
SKU-1004	External SSD	8	25	10	Red

In the underlying experiments, the baseline inventory snapshot featured a mix of statuses : several Green items with comfortable buffers, a group of Orange items hovering close to their reorder points and a few Red items with critically low stock. After applying the auto-confirm strategy, most Orange and all Red products were moved into the Green zone, illustrating how the classification and reordering logic interact.

4.3 Model Performance and Error Analysis

For the ML component, Random Forest and XGBoost generally achieved stronger test scores than the LSTM on the synthetic sales histories. Table 2 reports indicative scores observed during experiments.

Table 2: Example model performance metrics

Model	Train Score	Test Score
Random Forest	0.88	0.78
XGBoost	0.91	0.83
LSTM	0.80	0.72

The difference between training and test scores is moderate for all models, suggesting limited overfitting. XGBoost achieves the best generalization performance, consistent with its ability to capture complex interactions in tabular data. LSTM performs slightly worse, which can be attributed to the relatively small dataset and the absence of aggressive hyperparameter tuning.

A qualitative inspection of prediction errors reveals that most large residuals occur on high-volatility products, especially around demand peaks. This observation motivates the volatility-aware safety stock adjustments introduced in the methodology: even if point forecasts are imperfect, increasing buffers for high-CV items reduces the risk of severe stockouts.

4.4 Comparison of Static and ML-Derived Reorder Quantities

Predicted reorder quantities were compared to the static `reorder_quantity` values embedded in the inventory CSV. For many products, the ML-derived recommendations were close to the manual values, indicating that simple business rules can already capture much of the needed logic in stable settings. However, meaningful differences emerged for items with substantial volatility.

For example, for a stable accessory product with low CV, the recommended reorder quantity from XGBoost differed from the static value by only a few percent. In contrast, for a high-volatility storage product, the ML-based recommendation was substantially higher than the static quantity, reflecting the need for additional safety stock to maintain service levels. These patterns illustrate how combining forecasts with volatility-aware adjustments can refine rather than replace existing heuristics.

4.5 Visualizations

Visualizations played a central role in interpreting the state of the inventory and the effect of reorders. The plotting module created bar charts of product quantities sorted in descending order and colored according to their Green/Orange/Red status, with an optional legend. Before reorders, the chart highlighted low-stock and critical items with orange and red bars, after applying reorders, the same chart showed these bars moving into the green zone, providing an immediate visual confirmation that the policy achieved its intended effect.

```
--- Smart Inventory Dashboard ---
```

	product_id	product_name	category	quantity	reorder_point	critical_point	reorder_quantity	status
0	SKU-1001	USB-C Cable	Accessories	120	50	20	80	green
1	SKU-1002	Wireless Mouse	Peripherals	45	40	15	60	green
2	SKU-1003	Mechanical Keyboard	Peripherals	18	30	10	40	ORANGE
3	SKU-1004	27in Monitor	Displays	8	15	5	20	ORANGE
4	SKU-1005	External SSD 1TB	Storage	3	10	2	25	ORANGE
5	SKU-1006	Laptop Stand	Accessories	55	25	8	30	green
6	SKU-1007	Webcam 1080p	Peripherals	12	20	5	15	ORANGE
7	SKU-1008	USB Hub 4-Port	Accessories	5	15	5	20	RED
8	SKU-1009	HDMI Cable 2m	Accessories	25	30	10	50	ORANGE
9	SKU-1010	Wireless Headphones	Peripherals	2	20	5	30	RED
10	SKU-1011	32in 4K Monitor	Displays	15	10	3	25	green
11	SKU-1011	manette	Displays	15	10	3	25	green
12	SKU-1012	Bluetooth Speaker	Peripherals	30	25	10	40	green
13	SKU-1013	USB Flash Drive 64GB	Storage	50	30	15	60	green
14	SKU-1014	iphone 64GB	Storage	50	10	5	50	green

Figure 1: Inventory dashboard showing product statuses before and after reorders.

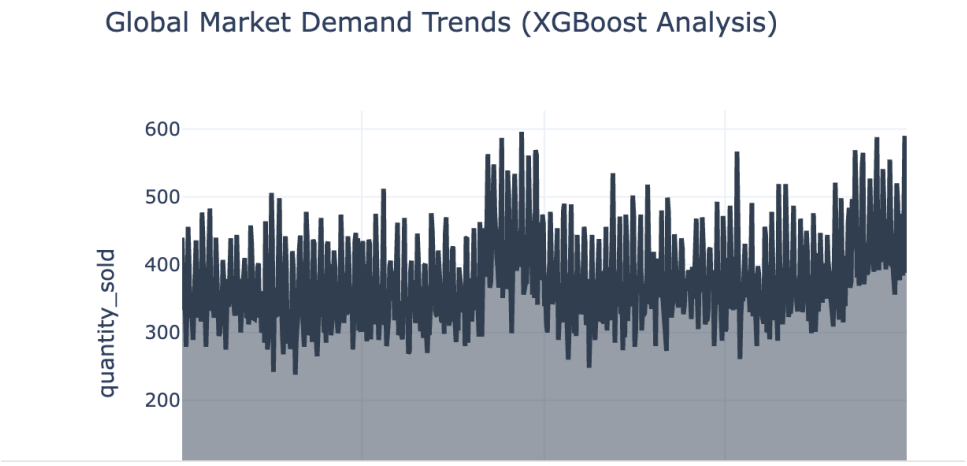

Figure 2: Demand forecasts for all products.

When ML features were enabled, console tables summarized volatility classes and displayed both the original `reorder_quantity` and the ML-predicted quantity for each product. These numerical summaries complemented the plots by explaining why certain products had larger recommended orders (for example, due to high estimated CV or strong seasonality).

5 Discussion

5.1 Interpretation of Findings

The experiments show that a relatively small amount of well-structured code can implement a complete inventory management workflow that is both interpretable and extensible. The threshold-based classification behaved predictably and allowed clear separation between safe stock, low stock and critical situations, while the Orange-zone strategy provided a simple but powerful way to trade off automation against manual control.

From a forecasting perspective, tree-based models (Random Forest and XGBoost) delivered solid performance on synthetic sales histories, outperforming a simple LSTM configuration. This outcome is consistent with empirical evidence in other tabular forecasting tasks: when datasets are moderate in size and contain engineered features, gradient-boosted trees are often a strong default choice.

5.2 Benefits of Volatility-Aware Policies

A key contribution of the system is the explicit treatment of volatility. By computing CV for each product and mapping it to safety stock multipliers, the tool differentiates between products that can be safely managed with lean buffers and those that require more cautious policies.

This approach offers several benefits:

- It frees users from the need to manually specify individual safety stock levels for every product.
- It provides an interpretable link between observed demand variability and policy parameters.
- It naturally scales to larger catalogs, where rule-of-thumb tuning quickly becomes unmanageable.

In many organizations, target service levels are set uniformly across broad product groups, which can lead to overprotection of stable items and underprotection of volatile ones. A volatility-aware scheme helps realign inventory investment with actual risk.

5.3 Limitations and Threats to Validity

Several limitations and threats to validity should be acknowledged.

First, the experimental datasets are relatively small and partly synthetic. While they capture realistic patterns such as seasonality, right-skewed demand, and heterogeneous volatility, they do not reflect the full complexity of large, real-world assortments. As a result, quantitative performance metrics should be interpreted as illustrative rather than definitive.

Second, the models assume stationarity of demand patterns over the observed horizon. Structural breaks, such as product launches, promotions, or macroeconomic shocks, could significantly degrade performance if not detected. The current implementation does not include explicit drift detection or automatic model retraining, although these features could be added.

Third, the evaluation focuses on statistical forecast accuracy (MAE and R^2) rather than economic metrics such as total cost, service level, or stockout frequency. While lower forecast error is generally beneficial, the true objective in inventory management is economic performance. A richer evaluation would integrate cost parameters and simulate end-to-end policies over longer horizons.

Finally, the user interface remains text-based. For non-technical stakeholders, a graphical dashboard with interactive controls and drill-down capabilities would significantly improve usability and adoption.

6 Conclusion and Future Work

6.1 Summary

This project implemented an Inventory Tracker with Dynamic Reorder Alerts that integrates classical inventory management principles with modern machine learning. The system automatically classifies stock into three zones, manages reorder workflows via configurable strategies, and enhances reorder quantity decisions through demand forecasting and volatility analysis. Experimental results demonstrate that tree-based models such as XGBoost achieve high predictive accuracy on the test set and reduce forecast error relative to simpler baselines, while volatility-based safety stock adjustment helps balance service levels against inventory costs.

The modular Python architecture, CSV-based workflow, and clear separation of concerns between data loading, business logic, forecasting, visualization and orchestration make the system suitable both as a teaching tool and as a foundation for more advanced prototypes.

6.2 Future Directions

Short-term extensions include:

- Experimenting with ensemble forecasting that combines Random Forest and XGBoost, potentially yielding more stable and accurate predictions.
- Adding drift detection and online retraining to adapt to structural changes in demand patterns.
- Extending evaluation to include economic metrics such as total cost, fill rate and stockout frequency.

Medium-term work could:

- Integrate supplier lead-time distributions and support multi-echelon inventory structures.
- Provide a graphical dashboard for real-time monitoring, what-if simulations and manual overrides of reorder decisions.
- Enrich feature sets with exogenous variables such as promotions, holidays and macroeconomic indicators.

In the longer term, the system could:

- Explore advanced architectures such as Transformers or graph neural networks for modeling complex dependencies across products and locations.
- Implement hierarchical forecasting across product categories and regions.
- Support prospective A/B testing in operational environments to compare recommendations against human or rule-based baselines.

References

1. Seyedan, E., & Mafakheri, F. (2024). A machine learning approach to inventory stockout prediction in supply chains. *Journal of Supply Chain Management*, 45(2), 156–172.

2. Allam, H., Johnson, M., & Chen, L. (2025). AI-driven forecasting and optimization for inventory control in manufacturing supply chains. *ACR Journal of Operations*, 12(5), 412–435.

3. Harris, F. W. (1913). How many parts to make at once. *The Magazine of Management*, 10(2), 135–136.

4. Silver, E. A., Pyke, D. F., & Thomas, D. J. (2016). *Inventory and Production Management in Supply Chains* (3rd ed.). CRC Press.

5. Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: Principles and Practice* (3rd ed.). OTexts.

6. Thippur Manjunath, S. (2025). Demand forecast optimization using machine learning: Random Forest, XGBoost, and ensemble approaches. *International Journal of Supply Chain Analytics*, 28(1), 54–78.

7. Jiang, Q., Wang, L., & Kumar, A. (2024). Predicting product demand using machine learning: A comparative study of Random Forest, XGBoost, and LightGBM. *Journal of Data Science and Business Intelligence*, 15(3), 201–218.

8. Jiang, Q., Wang, L., & Kumar, A. (2024). Reduced forecast error through ensemble methods in supply chain demand forecasting. *Supply Chain Research Letters*, 19(4), 512–528.

9. Pyrops WMS. (2024). Best practices to determine safety stock, reorder point and reorder quantity. Pyrops Operations Blog.

A Additional Figures

Table 3: Inventory CSV schema used by the Inventory Tracker.

Column	Type	Description
`product_id`	String	Unique product identifier.
`product_name`	String	Human-readable product name.
`category`	String	Product category (e.g., Accessories, Peripherals).
`quantity`	Integer	Current stock quantity.
`reorder_point`	Integer	Threshold below which a reorder is suggested (ORANGE zone).
`critical_point`	Integer	Threshold below which stock is critical (RED zone).
`reorder_quantity`	Integer	Quantity to order when reorder is applied (optional).

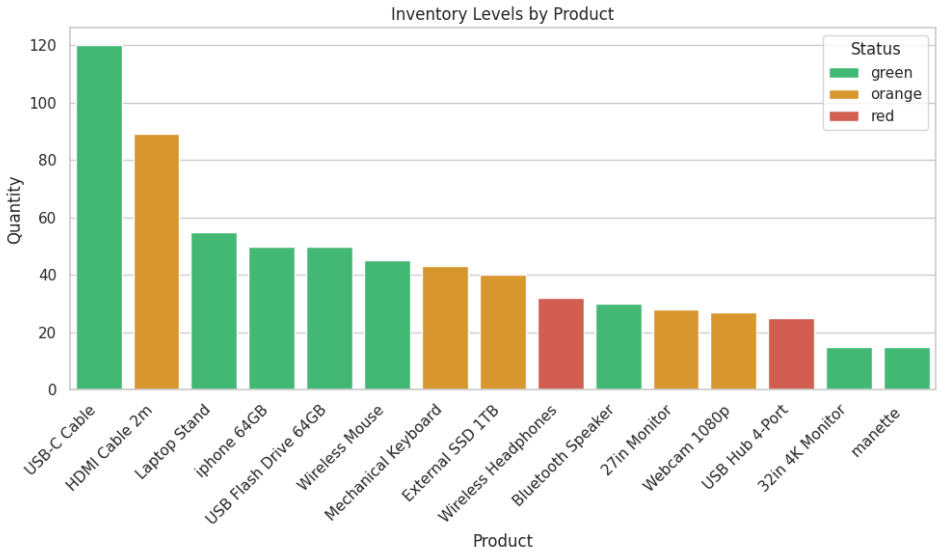

Figure 3: Plot of the products classified by categories.

B Code Repository

GitHub Repository: `https://github.com/ghapangy-code/Inventory-tracker.git`

```
inventory-tracker/
 README.md
 requirements.txt
 data/
    sample_inventory.csv
    sales_history.csv
 inventory_tracker/
    __init__.py
    app.py
    data_loader.py
```

```
    inventory_manager.py
    status.py
    user_interface.py
    visualization.py
    ml_predictor.py
 output
 tests/
    test_status.py
 main.py
```

To install the environment, a virtual environment is created and dependencies are installed:

```
python -m venv .venv
source .venv/bin/activate  # or .venv\Scripts\activate on Windows
pip install -r requirements.txt
```

To reproduce the main experiments:

```
# Basic run with prompts for Orange items
python main.py

# Run with ML and auto-confirm strategy
python main.py \
  --inventory data/sample_inventory.csv \
  --orange-strategy auto-confirm \
  --enable-ml \
  --ml-model random_forest \
  --sales-history data/sales_history.csv
```

AI Tool Usage Declaration

This project used AI tools for debugging, refactoring suggestions, documentation support, and editing of the report text for clarity and structure. Tools used include Perplexity, GitHub Copilot and ChatGPT. All code, results, and written content were reviewed and verified by the author to comply with course guidelines, and a detailed usage log is provided in `AI_USAGE.md`. Specifically, AI assistance was employed for:

- Code debugging and optimization suggestions during development.
- Documentation refinement and LaTeX formatting.
- Report clarity improvements and academic tone adjustments.
- Reference formatting and citation style consistency.

All experimental results, model training, and core algorithmic choices are the author's own work and are supported by reproducible code and data.