

A PROJECT REPORT ON

Project-Dynamic_Inventory_System_with_Knapsack-
Based_Profit_Optimization

Submitted by

DAKSH THAKUR (24MCA20014)

in partial fulfillment for the award of the degree of
MASTER OF COMPUTER APPLICATION (MCA)

University Institute of Computing (UIC)

Chandigarh University

BONAFIDE CERTIFICATE

This is to certify that the project report titled “Project Dynamic Inventory System with Knapsack Based Profit Optimization” is the Bonafide work of DAKSH THAKUR, a student of MCA, University Institute of Computing(UIC), who carried out the project work under my guidance and supervision during the academic year 2025-26.

The project report has been submitted in partial fulfillment of the requirements for the award of the MCA and is a record of the original work carried out by the student.

The contents of this report, in full or in part, have not been submitted to any other Institute on or university for the award of any degree or diploma.

BY
DAKSH THAKUR

SUPERVISOR
Mrs. Kawaljit Kaur
MCA
University Institute of Computing(UIC)

05-04-2025

Acknowledgment

I take this opportunity to express my heartfelt gratitude to all those who contributed to the successful completion of my project titled “**Dynamic Inventory System with Knapsack-Based Profit Optimization.**” This project has been an insightful learning experience, and I am deeply thankful for the support, guidance, and encouragement I received throughout the development process.

First and foremost, I would like to express my sincere thanks to **Kawaljit Kaur Ma’am** for her continuous guidance, valuable suggestions, and unwavering support during the course of this project. Her expertise and insightful feedback helped me refine my ideas and translate them into practical implementation.

I would also like to thank my peers and friends who provided their moral support and constructive feedback during various stages of the project. Whether it was brainstorming logic, debugging code, or reviewing the interface, their involvement kept me motivated and goal-oriented.

Lastly, I am grateful to my family for their constant encouragement, patience, and faith in me throughout this journey. Their unwavering support served as a pillar of strength and helped me overcome challenges and stay focused.

.

Table of Contents

1. ABSTRACT
2. INTRODUCTION
3. LITERATURE SURVEY
4. SYSTEM DESIGN AND STRUCTURE
5. IMPLEMENTATION AND CODE OVERVIEW
6. CHALLENGES FACED
7. CONCLUSION
8. FUTURE WORK

ABSTRACT

The project titled “**Dynamic Inventory System with Knapsack-Based Profit Optimization**” aims to solve the real-world problem of maximizing profit within a limited inventory capacity. Utilizing the **fractional knapsack algorithm**, the system selects the most profitable items for storage. This approach provides a strategic advantage for businesses with restricted storage, especially in the fields of **e-commerce and retail**, where optimizing profit is essential. The system allows users to enter, manage, and optimize inventory using a user-friendly GUI built with **Python's Tkinter**.

In today's competitive market, effective inventory management plays a vital role in ensuring profitability. Traditional systems often overlook the potential of algorithmic optimization, relying instead on manual selection or fixed rules. This project bridges that gap by implementing a logic-driven method that prioritizes item selection based on **profit-to-weight ratio**, ensuring that high-value items are always favored.

The core strength of the system lies in its ability to dynamically respond to user-defined constraints, such as total available capacity. It not only facilitates smart decision-making but also encourages experimentation by allowing users to visualize the impact of different inventory configurations on total profit. Through this flexibility, it becomes a practical educational tool for students and a prototype base for developers looking to scale similar systems commercially.

The use of the **fractional knapsack algorithm** ensures that even partial quantities of high-profit items are considered, reflecting real-world practices where goods may be divided or prioritized by urgency or value. This fractional nature differentiates the system from many traditional, binary decision models that either fully include or exclude items.

Furthermore, the modular structure of the application ensures that it is extendable and customizable. It can be integrated with backend databases, enhanced with predictive analytics using machine learning, or even deployed as a web application for cloud-based usage. In essence, this project showcases the fusion of **algorithmic efficiency and user-centric design**, offering a strong platform for future innovation in inventory systems.

INTRODUCTION

Inventory management is a fundamental aspect of any retail, manufacturing, or e-commerce business. Efficient inventory control not only reduces operational costs but also maximizes profitability by ensuring that high-value goods are prioritized within limited storage space. The increasing complexity of product management and customer demand has made it imperative to adopt intelligent solutions that go beyond traditional manual or rule-based approaches.

This project introduces a **Dynamic Inventory System** integrated with a **Knapsack-Based Profit Optimization algorithm**, specifically the **fractional knapsack algorithm**, which enables decision-making based on real-time input from the user. The main goal is to assist businesses in selecting items for storage that yield the highest possible return while staying within predefined capacity constraints.

What sets this system apart is its ability to account for fractional quantities of inventory items. This is a significant advantage in industries where products can be divided, like raw materials, bulk items, or perishable goods. The algorithm calculates the optimal selection strategy by evaluating the **profit-to-weight ratio** of each item, ensuring the best outcome in terms of total profit.

Developed using **Python** and its GUI library **Tkinter**, the application is designed to be intuitive, fast, and suitable for non-technical users. The user interface allows easy input of item details (name, weight, and profit), along with a straightforward mechanism to define the total storage capacity. Once data is entered, the algorithm processes the list and displays the selected items and maximum profit achievable.

The project not only provides a robust solution to inventory optimization problems but also demonstrates the practical implementation of theoretical algorithms. It bridges the gap between academic knowledge and real-world applications, offering a scalable model that can be further enhanced with database integration, cloud deployment, or machine learning techniques for demand forecasting.

Ultimately, this project underscores the growing importance of **data-driven decision-making** in inventory management and highlights how classical algorithmic techniques, when paired with accessible technology, can deliver innovative and efficient solutions.

Significance of the Project

Inventory management is a crucial aspect of operations in industries such as retail, logistics, and e-commerce. A poor inventory system can lead to overstocking, stockouts, financial losses, and missed opportunities. This project addresses the critical need for a system that not only stores inventory data but also makes intelligent decisions to **maximize profit while staying within storage limitations**.

As businesses face limitations in warehouse space, especially startups and small to medium-sized enterprises (SMEs), there is a growing need for **smart systems** that can optimize resources and drive better decisions. The **Dynamic Inventory System with Knapsack-Based Profit Optimization** directly targets this challenge by using a mathematical model that makes selections based on potential profitability rather than random or manual choices.

Furthermore, this system serves as a learning model for understanding how optimization algorithms can solve real-world problems, combining both theoretical and practical aspects. It

provides a strategic edge to businesses and academic learners by merging computational logic with business insights.

Approach Used

The project uses the **fractional knapsack algorithm**, a type of greedy algorithm known for its effectiveness in solving optimization problems. Unlike the 0/1 knapsack, which accepts items as a whole, the fractional approach allows the selection of parts of an item, ensuring that even partially valuable items can contribute to the overall profit.

Key steps of the approach include:

- **Accepting input** from users: item names, their weights, profits, and total capacity.
- **Sorting** the items based on their **profit-to-weight ratio**.
- **Selecting** items greedily to fill the capacity with the highest possible profit.
- **Visualizing** the selected inventory and the calculated profit using a **Graphical User Interface (GUI)** developed with **Python's Tkinter library**.

This modular, algorithm-driven approach ensures that the system is both **scalable and adaptable**, allowing future integration with features like inventory tracking, database storage, and cloud deployment.

Overall, the project successfully demonstrates how a **classic algorithmic strategy can be applied to modern inventory systems**, making it both innovative and practically useful.

3. LITERATURE SURVEY

Inventory optimization has long been a focus area in operations research, logistics, and supply chain management. Various studies and models have been proposed over the decades to improve inventory decisions by minimizing costs and maximizing profits. The knapsack problem, a classic in the field of algorithm design and combinatorial optimization, has been a key model used in these efforts.

Previous Research and Studies

Several research papers and academic studies have explored the effectiveness of optimization algorithms in inventory and resource allocation. The **fractional knapsack algorithm**, introduced in the context of greedy strategies, has been proven to be optimal for problems where items can be broken into smaller parts. It is particularly useful in scenarios involving divisible goods, such as raw materials or bulk inventory.

A study titled *"Greedy Algorithms in Resource Management"* highlighted how greedy strategies outperform other brute-force methods in speed and efficiency when applied to real-time decision-making problems. Similarly, research on *"Inventory Optimization in Retail Using Heuristic Techniques"* emphasized the potential of algorithmic systems in enhancing profitability by choosing the right product mix under constrained storage conditions.

Another relevant work includes the application of **AI and machine learning** to predict future demand and reorder points. While this project does not integrate predictive analytics yet, it lays

the groundwork for such advancements by providing an algorithm-based core system for static optimization.

Technological References

From a technical perspective, the use of **Python** as a language for implementing optimization algorithms is widely supported in the academic and developer community due to its readability, large library ecosystem, and integration capabilities. **Tkinter**, as a lightweight GUI toolkit, offers a simple yet effective way to build desktop-based interactive applications.

The approach taken in this project aligns with methodologies discussed in algorithm textbooks such as *“Introduction to Algorithms”* by Cormen et al., where the knapsack problem is used to demonstrate efficient greedy solutions. It also draws from inventory theory, particularly in the area of constrained optimization and profit analysis.

Gaps Identified

While many enterprise-level systems use advanced ERP tools for inventory control, there is a gap in affordable, algorithm-backed solutions for smaller businesses or learners. Most available systems are either too complex or do not incorporate intelligent selection strategies. This project fills that gap by offering a **simple, cost-effective, and efficient model** that applies a well-known algorithm in a practical, user-oriented context.

4. SYSTEM DESIGN AND STRUCTURE

The system design of the **Dynamic Inventory System with Knapsack-Based Profit Optimization** is modular and user-centric, ensuring both functionality and ease of use. It is structured to follow a logical workflow—from data input to algorithmic computation, and finally, to result display and optimization feedback.

System Architecture Overview

The system consists of three major layers:

1. User Interface (UI) Layer

- Developed using **Python’s Tkinter** library, the GUI provides a clean and intuitive interface.
- Users can enter item names, weights, profits, and the total inventory capacity.
- Buttons are included for functions like “Add Item”, “Optimize”, and “Clear Data”.
- Real-time feedback such as total profit and selected items is shown clearly after computation.

2. Application Logic Layer

- The core functionality resides in this layer.
- When the user initiates the optimization process, the system executes the **fractional knapsack algorithm**.
- It sorts the items based on their **profit-to-weight ratio** and adds them to the knapsack until the capacity is filled.
- If the remaining capacity can’t hold a full item, it includes the fractional part to maximize profit.

3. Data Storage (Runtime)

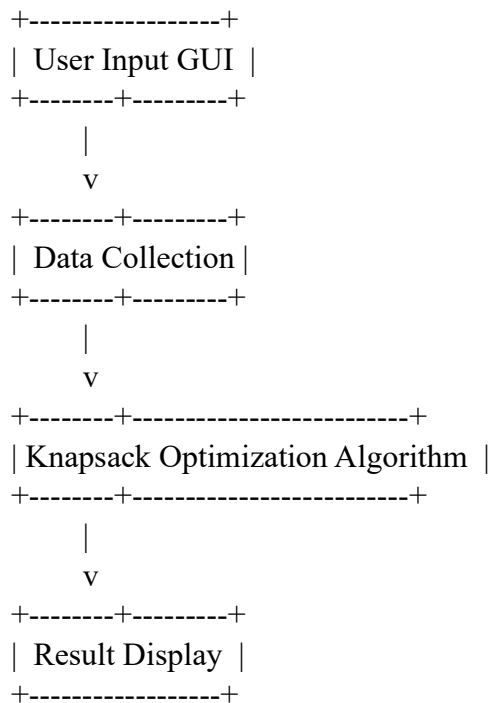
- In the current implementation, data is stored **temporarily in runtime using Python lists and dictionaries**.

- Each item is treated as a data object containing its name, weight, and profit, which is stored in a collection.
- While there is no database integration in this version, the design allows future inclusion of databases such as **SQLite or MySQL**.

Key Components

- **Input Module:** Accepts user inputs (item details and capacity). Performs validation to ensure no field is left empty or invalid.
- **Knapsack Algorithm Module:** Contains the logic for selecting the most profitable combination of items. Executes in linearithmic time ($O(n \log n)$ due to sorting).
- **Display/Output Module:** Shows the results of the optimization including:
 - Items selected (fully or partially)
 - Total weight used
 - Total profit achieved
 - Graphical/textual representation of result

Flow Diagram (Conceptual)



Design Strengths

- **Modularity:** Each function is separated for better readability and debugging.
- **Scalability:** Easily upgradable to web platforms or integrated with databases.
- **Reusability:** Algorithm can be reused in other domains like logistics, scheduling, or financial portfolio management.
- **Accessibility:** The GUI simplifies user interaction, eliminating the need for command-line inputs or coding knowledge.

This structured design ensures that the application is not only effective in solving the problem but also maintainable and expandable for future enhancements such as **machine learning integration, REST APIs, or cloud deployment**.

5. IMPLEMENTATION AND CODE OVERVIEW

The **Dynamic Inventory System with Knapsack-Based Profit Optimization** was developed using **Python**, a widely-used programming language known for its simplicity and robustness. The graphical user interface (GUI) was created using the **Tkinter** library, making the system accessible and user-friendly. The core logic for profit optimization relies on the **Fractional Knapsack Algorithm**, which helps in selecting the most profitable combination of items that can be stored within a limited capacity.

Implementation Architecture

The code is modular, and its structure consists of three primary classes:

1. Item Class

The Item class represents each inventory item. It stores attributes such as:

- name: Name of the item
- weight: Weight of the item
- profit: Profit that can be earned
- profit_per_weight: A key value used for prioritizing items during knapsack optimization

python

CopyEdit

class Item:

```
def __init__(self, name, weight, profit):
```

```
...
```

2. Inventory Class

The Inventory class handles the main business logic. It:

- Manages a list of Item objects
- Provides CRUD operations: add, update, remove, view
- Implements the **Fractional Knapsack Algorithm**:
 - Sorts items based on their profit-to-weight ratio
 - Adds full or fractional items to maximize profit
 - Calculates and returns the total profit

python

CopyEdit

class Inventory:

```
def fractional_knapsack(self, capacity):
```

```
...
```

The sorting and selection logic ensures that the most profitable items (by unit weight) are prioritized.

3. InventoryApp Class (Tkinter GUI)

This class is responsible for the interface. It allows users to:

- Enter item details (name, weight, profit)

- Define the knapsack's total capacity
- Perform operations like Add, Update, Delete, View, and Calculate Optimization
- View outputs in a scrollable text area

Functional Elements of the GUI:

- Entry widgets: To input item details
- Buttons: For user actions (Add, Update, Remove, View, Optimize)
- Text widget: To display results and selected items for knapsack

Example GUI layout code:

```
python
CopyEdit
tk.Label(root, text="Item Name:").grid(...)
self.name_entry = tk.Entry(root)
...
tk.Button(root, text="Add Item", command=self.add_item).grid(...)
```

Knapsack Optimization Example

If a user inputs:

- Item A: 10kg, ₹60
- Item B: 20kg, ₹100
- Item C: 30kg, ₹120

And sets capacity = 50kg, the output will be:

Added full item: A (10kg, ₹60)

Added full item: B (20kg, ₹100)

Added partial item: C (20kg of 30kg, ₹80)

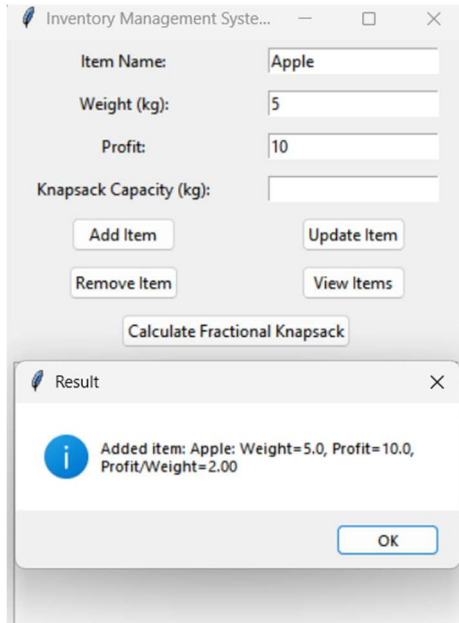
Total profit: ₹240.00

Key Features in Code

- **Efficient Item Management:** Add, update, delete, and view inventory in real time.
- **Optimal Item Selection:** Uses a greedy algorithm (Fractional Knapsack) to calculate best profit.
- **Clean UI:** Built with Tkinter, designed for ease of use.
- **Validation & Messaging:** User actions are confirmed with message popups using messagebox.

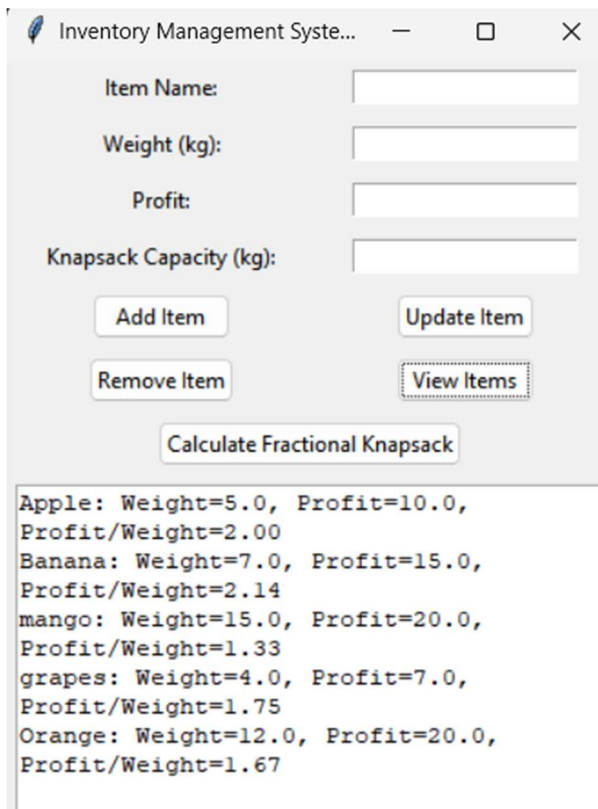
Outcomes-The application outputs include:

- **Adding items:** Displays a confirmation message.



The screenshot shows the 'Inventory Management System' window. It has input fields for 'Item Name' (Apple), 'Weight (kg)' (5), 'Profit' (10), and 'Knapsack Capacity (kg)'. Below these are buttons for 'Add Item', 'Update Item', 'Remove Item', 'View Items', and 'Calculate Fractional Knapsack'. A 'Result' dialog box is open, displaying an information icon and the text: 'Added item: Apple: Weight=5.0, Profit=10.0, Profit/Weight=2.00'. An 'OK' button is at the bottom of the dialog.

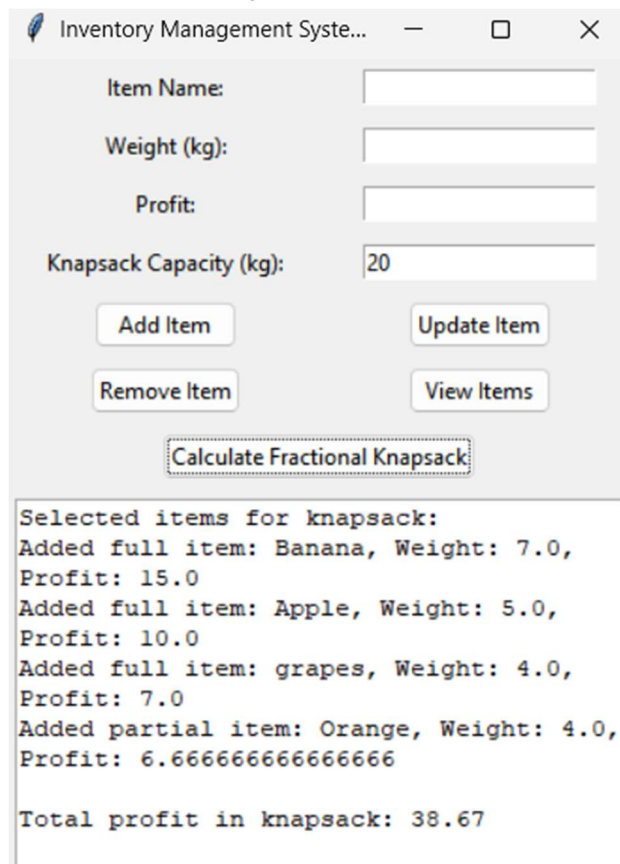
- **Viewing items:** Lists all items in the inventory with details.



The screenshot shows the 'Inventory Management System' window with the 'View Items' button highlighted. Below the form, a text area displays the following list of items:

```
Apple: Weight=5.0, Profit=10.0, Profit/Weight=2.00
Banana: Weight=7.0, Profit=15.0, Profit/Weight=2.14
mango: Weight=15.0, Profit=20.0, Profit/Weight=1.33
grapes: Weight=4.0, Profit=7.0, Profit/Weight=1.75
Orange: Weight=12.0, Profit=20.0, Profit/Weight=1.67
```

- **Knapsack Calculation:** Shows selected items for maximum profit and the total profit within capacity constraints.



The screenshot shows a window titled "Inventory Management System...". It contains several input fields and buttons. The input fields are labeled "Item Name:", "Weight (kg):", "Profit:", and "Knapsack Capacity (kg):". The "Knapsack Capacity (kg)" field has the value "20" entered. Below these fields are four buttons: "Add Item", "Update Item", "Remove Item", and "View Items". At the bottom of the form is a button labeled "Calculate Fractional Knapsack". Below the form is a text area displaying the following output:

```

Selected items for knapsack:
Added full item: Banana, Weight: 7.0,
Profit: 15.0
Added full item: Apple, Weight: 5.0,
Profit: 10.0
Added full item: grapes, Weight: 4.0,
Profit: 7.0
Added partial item: Orange, Weight: 4.0,
Profit: 6.666666666666666
Total profit in knapsack: 38.67

```

Conclusion of Implementation

This implementation integrates algorithmic logic with a functional GUI. It provides both academic insight into algorithm application and practical utility for small-scale inventory management systems. The modularity, clarity, and user interface design make the system extensible and educational.

6. CHALLENGES FACED

During the development of the **Dynamic Inventory System with Knapsack-Based Profit Optimization**, several technical and conceptual challenges were encountered. These challenges provided valuable learning opportunities and were resolved through research, debugging, and optimization.

1. Algorithm Selection and Application

Implementing the **Fractional Knapsack Algorithm** posed initial difficulties in terms of:

- Properly calculating and applying partial item selection when the remaining capacity could not accommodate a full item.
- Ensuring the sorting of items based on their profit-to-weight ratio was correctly implemented to maximize profit.
- Maintaining floating-point precision while calculating partial profits.

2. GUI Design and User Interaction

Creating a clean and user-friendly interface with **Tkinter** presented the following challenges:

- Managing the layout effectively using grid placement.
- Ensuring responsiveness and clarity for different screen sizes.
- Providing real-time feedback to users through messagebox pop-ups and text display areas.

3. Data Validation and Error Handling

Ensuring that the user inputs were valid required implementing checks such as:

- Preventing negative or zero values for weight, profit, and capacity.
- Handling incorrect data types (e.g., entering text in numeric fields).
- Managing operations when no item was found for update or removal, and providing proper feedback.

4. Inventory Update Logic

Managing item updates by name required careful handling to:

- Avoid duplicate entries.
- Locate and modify the correct object in the item list.
- Reflect changes immediately in the internal list and UI.

5. Optimization vs. Usability

Striking the balance between algorithmic efficiency and user experience was key. The app had to:

- Run the knapsack calculation quickly even as more items were added.
- Display selected items and total profit clearly to enhance decision-making.

6. Debugging GUI Logic

As with most GUI applications, one of the challenges was:

- Debugging widget behavior and ensuring button actions were tied to the correct logic.
- Clearing and updating the interface dynamically without crashing the program.

7. Time Constraints and Feature Limitations

Due to time limitations, certain advanced features such as:

- Database integration for storing items permanently
 - Exporting reports
 - Sorting and filtering views dynamically in the GUI
- ...were not implemented but noted as potential enhancements in the "Future Work" section.

Despite these challenges, the project was successfully completed by methodically identifying issues and applying appropriate programming and algorithmic solutions.

8. FUTURE WORK

While the **Dynamic Inventory System with Knapsack-Based Profit Optimization** has successfully achieved its primary objectives, there remains ample scope for enhancement and expansion. Future improvements can increase the system's robustness, usability, and real-world applicability.

1. Integration with Databases

- Implementing a database like **MySQL** or **SQLite** would allow inventory data to be stored persistently.
- This would enable retrieval of past records, support multi-user access, and make the system suitable for large-scale deployments.

2. User Authentication and Roles

- Introducing **user login functionality** with role-based access control (e.g., Admin, Staff, Viewer) can enhance security and manageability.
- Different users could have permissions to add, update, or view inventory based on their roles.

3. Enhanced Analytics and Reporting

- Add features to generate **profit reports**, **item usage statistics**, and **capacity usage graphs** using libraries like **Matplotlib** or **Seaborn**.
- Export reports in formats like **PDF** or **Excel** for business insights and decision-making.

4. Cloud-Based and Web Deployment

- Transitioning the desktop application to a **web-based interface** using frameworks such as **Flask** or **Django** would improve accessibility.
- Cloud integration can allow multiple branches or users to access the system remotely and simultaneously.

5. Advanced Optimization Algorithms

- Explore other optimization techniques such as **0/1 Knapsack**, **Genetic Algorithms**, or **Linear Programming** for more complex decision-making scenarios.
- Introduce constraints such as item expiration, restocking frequency, or supplier preferences.

6. Real-Time Inventory Tracking

- Integrate the system with **barcode scanners**, **RFID**, or **IoT sensors** to track inventory levels in real-time.
- Provide automatic notifications when stock levels are low or when capacity is about to be exceeded.

7. Mobile App Development

- Developing a **mobile version** of the system using tools like **Kivy** or **Flutter** can improve flexibility and allow on-the-go inventory management.

8. Machine Learning Integration

- In future versions, **machine learning** models can be used to predict demand, recommend stock levels, or automatically suggest the best items to store based on past data trends.

By implementing these enhancements, the inventory system can evolve from a simple optimization tool into a **full-fledged intelligent inventory management solution** adaptable to diverse industrial and commercial needs.