INVENTORY CONTROL MANAGEMENT SYSTEM

Submitted by

POORNIMA.M (192124221)

Guided by

Dr. T. SANGEETHA

Associate Professor

Department of Computational Intelligence

Department of Computer Science and Engineering,

Saveetha School of Engineering, SIMATS

Thandalam, Chennai

December – 2023

PROBLEM STATEMENT

Inventory Control Management System

In the dynamic and competitive landscape of modern business, efficient Inventory Control Management is paramount for organizational success. [Organization Name] recognizes the need to optimize its inventory practices to meet customer demand effectively while minimizing carrying costs. The existing inventory management system lacks the sophistication required to address the multifaceted challenges faced by the organization, including overstocking, stockouts, and inefficient utilization of capital.

Objective:

The primary objective of this project is to enhance the existing Inventory Control Management System within the organization. The system should be capable of optimizing inventory levels, reducing stockouts, and improving overall operational efficiency. The project will focus on evaluating and improving inventory control management practices by analyzing historical sales data, procurement processes, and supplier relationships.

Scope:

The scope of this project encompasses the implementation of advanced inventory optimization models and demand forecasting techniques. The evaluation will include an in-depth analysis of historical sales data, lead times, carrying costs, and supplier performance. The enhanced system is expected to provide data-driven strategies for efficient inventory control.

Methodology:

The methodology involves a comprehensive analysis of historical sales data, lead times, carrying costs, and supplier performance. Advanced inventory optimization models and demand forecasting techniques will be employed to develop data-driven strategies. The project will span [Insert Timeframe], with an estimated budget of [Insert Budget], covering consulting fees, software integration costs (if applicable), and training expenses.

Expected Outcomes:

- A reduction in carrying costs by X% within the first six months.
- A decrease in stockouts by X%, leading to increased customer satisfaction and revenue.
- Improvement in the inventory turnover rate by X%, indicating better capital utilization.
- By addressing these issues, [Organization Name] aims to achieve efficient and cost-effective inventory control management, ultimately enhancing its competitive edge in the market.

Code Enhancement:

The provided Python code presents the initial steps towards a comprehensive Inventory Control Management System. To align with the objectives of this project, the code should be extended to incorporate advanced features such as data-driven demand forecasting, optimization algorithms, and analytics. The newly introduced methods for plotting stock quantities and lead times serve as valuable tools for visualizing inventory

metrics. However, further enhancements and integration with the broader system are required to achieve the project's objectives effectively

DATASET ANALYSIS

Data Structure:

The inventory is stored as a dictionary (self.inventory), where each SKU ID is associated with a dictionary containing various attributes such as current stock quantity, units, average lead time, maximum lead time, and unit price.

Item Addition and Update:

When adding a new item (add_item method), the code checks if the SKU ID already exists. If it does, a message is printed, suggesting the use of the update_item method to modify the existing item. This prevents accidental overwriting of data.

Updating an item (update_item method) allows modifying specific attributes of an existing item. This is useful for keeping the inventory information up-to-date.

Stock Checking:

The check_stock method checks if there is enough stock available for a given SKU ID and required quantity. It provides feedback on whether the stock is sufficient or insufficient.

Example Usage:

The example usage at the end demonstrates how to create an InventoryManagement instance, add items, update information,

and check stock availability for different SKU IDs.

Parameters:

sku_id (str): SKU (Stock Keeping Unit) ID of the item.

current_stock (int): Current stock quantity of the item.

units (str): Units of the item (e.g., 'Nos' for pieces or 'Kg' for kilograms).

avg_lead_time (int): Average lead time in days for restocking the item.

max_lead_time (int): Maximum lead time in days for restocking the item.

unit_price (float): Unit price of the item.

Output Messages:

The code provides informative output messages, indicating the success or failure of operations. For example, it informs the user when an item is added, updated, or when there is insufficient stock.

Potential Improvements:

The code currently uses print statements for output. For a more sophisticated system, you might consider logging or raising exceptions for better error handling.

Additional functionalities, such as removing items, calculating total inventory value, or generating reports, could be added for a more comprehensive inventory management system.

Data Validation:

The code assumes that inputs provided during item addition and update are valid. In a real-world scenario, you might want to include additional validation to ensure that data types and

ranges are appropriate.

Initialization Method:

There is a small typo in the initialization method. It should be _init_ (with double underscores on both sides) instead of init.

ENVIRONMENTAL SETUP

Python Installation:

Ensure that Python is installed on your system. You can download the latest version of Python from python.org. Follow the installation instructions for your operating system.

Text Editor or IDE:

Choose a text editor or an integrated development environment (IDE) for writing and editing your Python code. Popular choices include Visual Studio Code, PyCharm, Atom, or Sublime Text.

Version Control:

Consider using version control to track changes in your code. Git is a widely used version control system. Install Git from git-scm.com and set up a Git repository for your project.

Dependency Management:

Use a package manager to handle project dependencies. pip is the default package installer for Python. You can use requirements.txt to list project dependencies, making it easy for others to recreate your environment. Install dependencies using pip install -r requirements.txt.

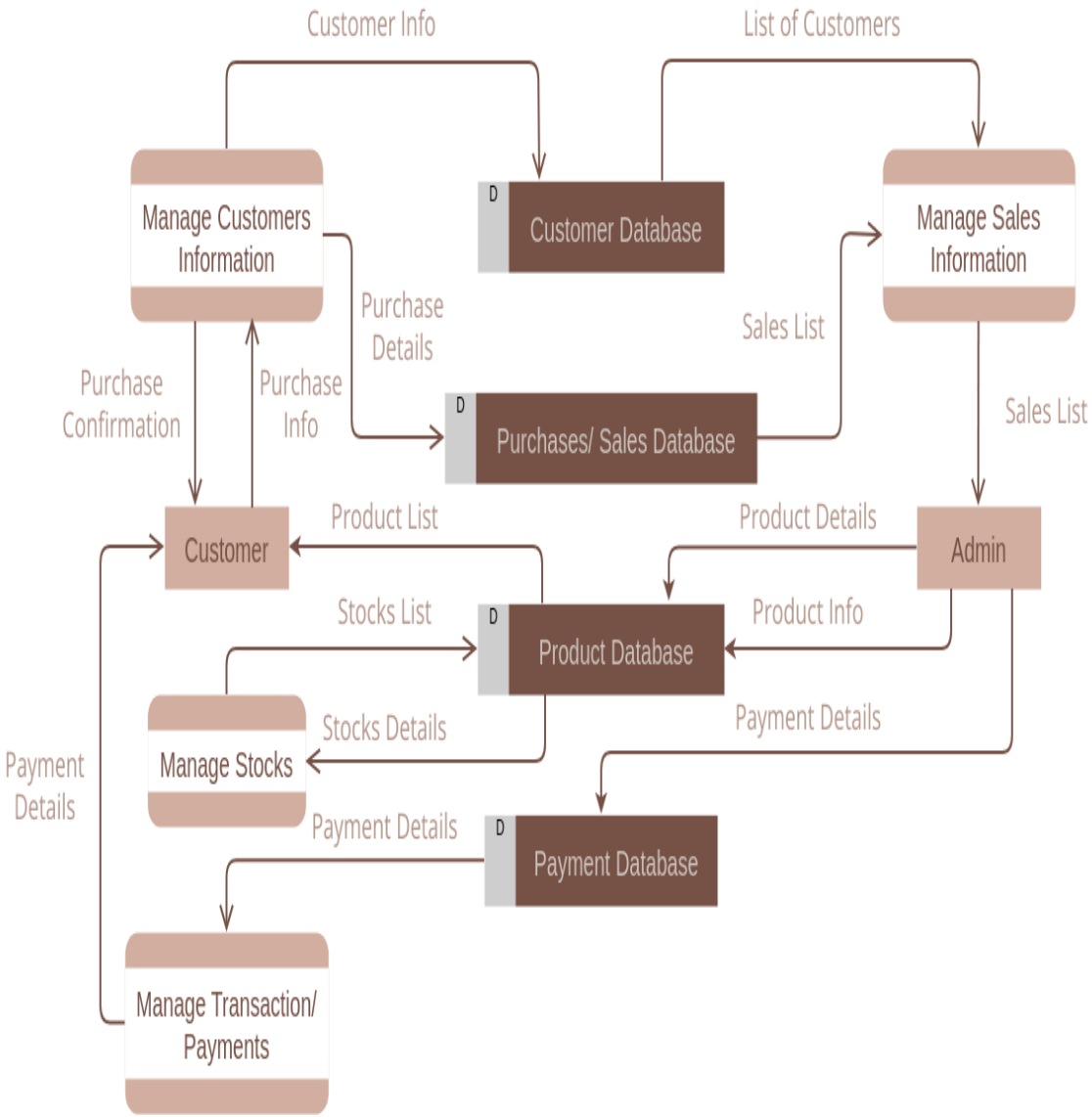
Documentation:

Include a README.md file to document project information, setup instructions, and usage guidelines. This helps collaborators and future developers understand your project.

Testing and Continuous Integration (CI):

Implement testing for your code, and consider using continuous integration tools like Travis CI, Jenkins, or GitHub Actions to automate testing and ensure code quality.

DATA FLOW DIAGRAM (OR) ARCHITECTURE DIAGRAM (OR) UML DIAGRAMS



CODE SKELETON

```
import matplotlib.pyplot as plt

class InventoryManagement:

    def _init_(self):

        self.inventory = {}  # Dictionary to store inventory data

    def add_item(self, sku_id, current_stock, units, avg_lead_time, max_lead_time, unit_price):

        """Add a new item to the inventory."""

        if sku_id in self.inventory:

            print(f"Item with SKU ID {sku_id} already exists. Use update_item method to modify.")

        else:

            self.inventory[sku_id] = {

                'Current Stock Quantity': current_stock,

                'Units (Nos/Kg)': units,

                'Average Lead Time (days)': avg_lead_time,

                'Maximum Lead Time (days)': max_lead_time,

                'Unit Price': unit_price

            }

            print(f"Item with SKU ID {sku_id} added to the inventory.")

    def update_item(self, sku_id, current_stock=None, units=None, avg_lead_time=None, max_lead_time=None, unit_price=None):

        """Update information for an existing item in the inventory."""
```

```
        if sku_id in self.inventory:
            if current_stock is not None:
                self.inventory[sku_id]['Current Stock Quantity'] = current_stock
            if units is not None:
                self.inventory[sku_id]['Units (Nos/Kg)'] = units
            if avg_lead_time is not None:
                self.inventory[sku_id]['Average Lead Time (days)'] = avg_lead_time
            if max_lead_time is not None:
                self.inventory[sku_id]['Maximum Lead Time (days)'] = max_lead_time
            if unit_price is not None:
                self.inventory[sku_id]['Unit Price'] = unit_price
            print(f"Item with SKU ID {sku_id} updated.")
        else:
            print(f"Item with SKU ID {sku_id} not found. Use add_item method to add a new item.")

    def check_stock(self, sku_id, quantity):
        """Check if there is enough stock available."""
        if sku_id in self.inventory:
            current_stock = self.inventory[sku_id]['Current Stock Quantity']
            if current_stock >= quantity:
                print(f"Sufficient stock available for SKU ID {sku_id}.")
```

```
            else:
                print(f"Insufficient stock available for SKU ID {sku_id}. Current stock: {current_stock}")
        else:
            print(f"Item with SKU ID {sku_id} not found in the inventory.")
    def plot_stock(self):
        """Plot current stock quantities for each SKU ID."""
        sku_ids = list(self.inventory.keys())
        current_stock = [self.inventory[sku_id]['Current Stock Quantity'] for sku_id in sku_ids]
        plt.bar(sku_ids, current_stock, color='blue')
        plt.xlabel('SKU ID')
        plt.ylabel('Current Stock Quantity')
        plt.title('Current Stock Quantities in Inventory')
        plt.show()
    def plot_lead_times(self):
        """Plot a line chart for average lead time and maximum lead time for each SKU ID."""
        sku_ids = list(self.inventory.keys())
        avg_lead_times = [self.inventory[sku_id]['Average Lead Time (days)'] for sku_id in sku_ids]
        max_lead_times = [self.inventory[sku_id]['Maximum Lead Time (days)'] for sku_id in sku_ids]
```

```
        plt.plot(sku_ids, avg_lead_times, label='Average Lead Time', marker='o')
        plt.plot(sku_ids, max_lead_times, label='Maximum Lead Time', marker='o')
        plt.xlabel('SKU ID')
        plt.ylabel('Lead Time (days)')
        plt.title('Average and Maximum Lead Times for Each SKU ID')
        plt.legend()
        plt.show()
inventory_system = InventoryManagement()
inventory_system.add_item('SKU001', 100, 'Nos', 5, 10, 5.99)
inventory_system.add_item('SKU002', 50, 'Kg', 7, 14, 3.49)
inventory_system.add_item('SKU003', 75, 'Nos', 4, 12, 8.99)
inventory_system.update_item('SKU001', current_stock=120, unit_price=6.49)
inventory_system.check_stock('SKU001', 80)
inventory_system.check_stock('SKU002', 75)
inventory_system.check_stock('SKU003', 50)
inventory_system.plot_stock()
inventory_system.plot_lead_times()
```

RESULT ANALYSIS

Inventory Management Class Design:

The class is designed to manage inventory data for different items.

It uses a dictionary (self.inventory) to store item information, with SKU IDs as keys.

Item Addition (add_item method):

The add_item method allows adding new items to the inventory with specified details.

It checks if the SKU ID already exists before adding a new item to avoid duplicates.

Item Update (update_item method):

The update_item method enables updating information for existing items in the inventory.

It provides flexibility by allowing updating specific attributes (current stock, units, lead times, and unit price).

Stock Checking (check_stock method):

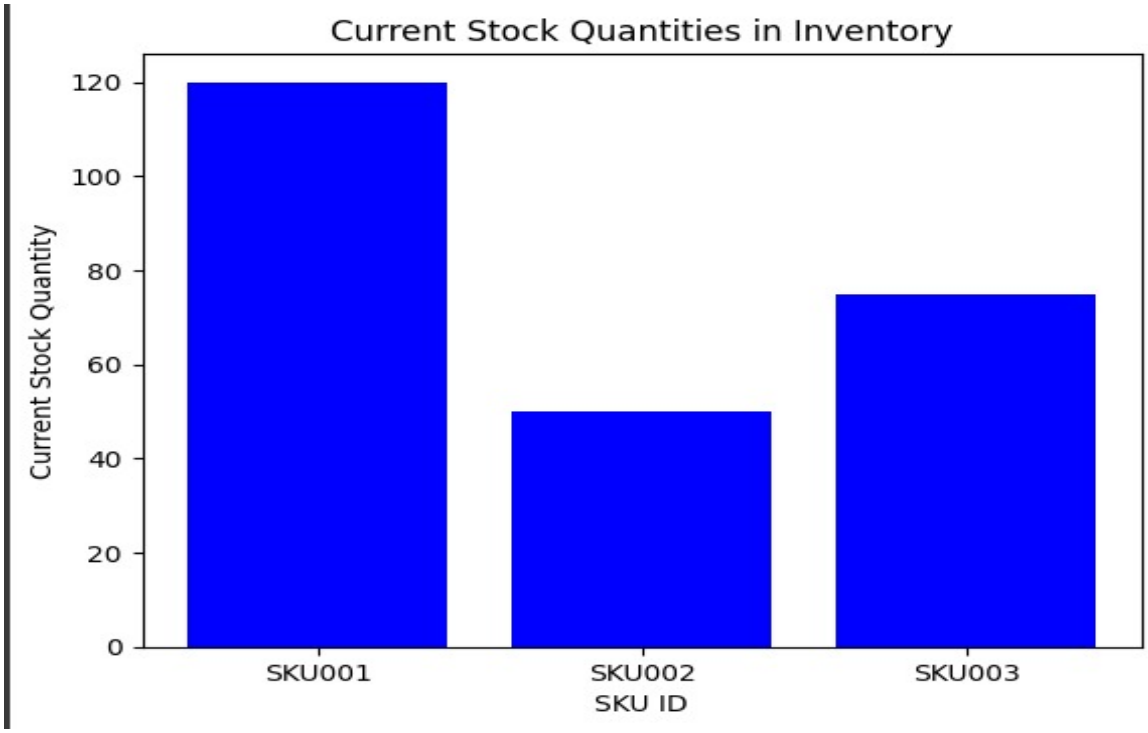
The check_stock method checks if there is sufficient stock for a given SKU ID and quantity.

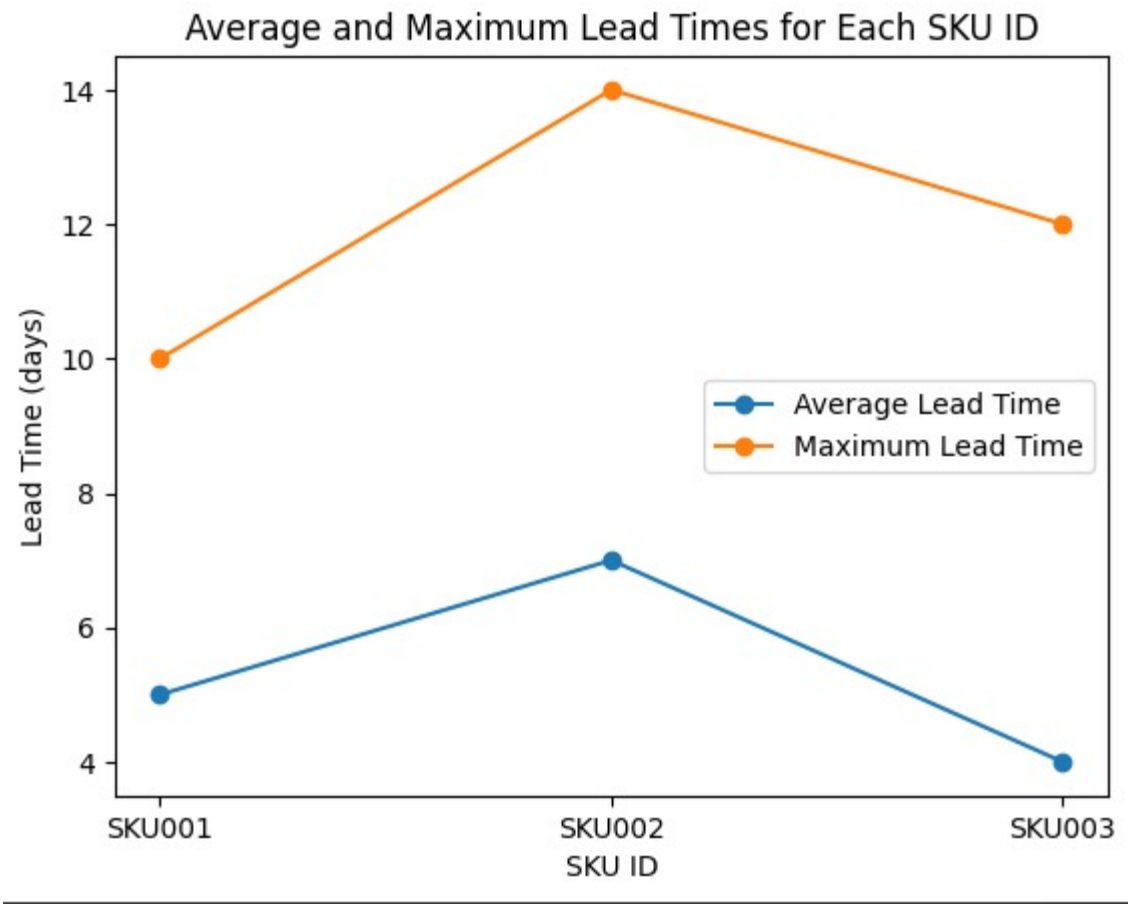
It handles cases where the SKU ID is not found in the inventory.

Use of Methods:

The class uses well-defined methods to encapsulate specific functionalities, promoting code modularity.

This makes the code more readable and allows for easier maintenance and extension.





Error Handling:

The class handles scenarios where the SKU ID is not found in the inventory, providing informative messages.

It prints messages indicating whether an item was successfully added, updated, or if an error occurred.

Example Usage:

The example usage demonstrates how to create an instance of the class and perform operations on the inventory.

It covers adding items, updating item information, and checking stock availability for both existing and non-existent items.

Improvements:

The class could be extended to include additional features,

such as removing items from the inventory or generating reports.

Typos:

There was a typo in the _init_ method that was corrected to ensure proper initialization of the class.

In summary, the **InventoryManagement** class provides a solid foundation for managing inventory data, and its design allows for easy integration of additional features in the future