**Problem 2: Inventory Management System Optimization**

**Scenario:**

You have been hired by a retail company to optimize their inventory management system. The company wants to minimize stockouts and overstock situations while maximizing inventory turnover and profitability.

**Tasks:**

1. Model the inventory system: Define the structure of the inventory system, including products, warehouses, and current stock levels.
2. Implement an inventory tracking application: Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below a certain threshold.
3. Optimize inventory ordering: Implement algorithms to calculate optimal reorder points and quantities based on historical sales data, lead times, and demand forecasts.
4. Generate reports: Provide reports on inventory turnover rates, stockout occurrences, and cost implications of overstock situations.
5. User interaction: Allow users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.

**Deliverables:**

* **Data Flow Diagram**: Illustrate how data flows within the inventory management system, from input (e.g., sales data, inventory adjustments) to output (e.g., reorder alerts, reports).
* **Pseudocode and Implementation**: Provide pseudocode and actual code demonstrating how inventory levels are tracked, reorder points are calculated, and reports are generated.
* **Documentation**: Explain the algorithms used for reorder optimization, how historical data influences decisions, and any assumptions made (e.g., constant lead times).
* **User Interface**: Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.
* **Assumptions and Improvements**: Discuss assumptions about demand patterns, supplier reliability, and potential improvements for the inventory management system's efficiency and accuracy.

**Approach:**

**+---------------+ +-------------------+**

**| User Input | | Inventory API |**

**+---------------+ +-------------------+**

**| |**

**| User requests |**

**| inventory updates | Inventory data**

**| and reorder options |**

**v v**

**+---------------+ +-------------------+**

**| Inventory App | | Database |**

**+---------------+ +-------------------+**

**| |**

**| Fetch inventory data | Store inventory data**

**| and reorder info |**

**v v**

**+---------------+ +-------------------+**

**| User Output | | Notification |**

**+---------------+ +-------------------+**

1. User Input: Users input product IDs, request inventory updates, and reorder options.
2. Inventory API: Provides real-time inventory data and alerts.
3. Inventory App: Central application that processes user requests and interacts with the database.
4. Database: Stores current stock levels, product details, and transaction history.
5. User Output: Displays current stock levels, reorder recommendations, and historical data to users.
6. Notification: Sends alerts for low stock levels or other important inventory updates.

**Pseudocode:**

Class Product

Initialize with:

product\_id

name

current\_stock

reorder\_point

reorder\_quantity

Class Warehouse

Initialize with:

warehouse\_id

location

products (an empty list)

Function track\_inventory(products)

For each product in products:

If product.current\_stock is less than product.reorder\_point:

Print "Alert: [product.name] is below the reorder point. Current stock: [product.current\_stock]"

Call recommend\_reorder(product)

Function recommend\_reorder(product)

Calculate new\_stock as product.current\_stock + product.reorder\_quantity

Print "Recommended reorder for [product.name]: [product.reorder\_quantity] units. New stock level: [new\_stock]"

Function calculate\_reorder\_point(historical\_sales, lead\_time, desired\_service\_level)

Calculate avg\_sales\_per\_day as the average of historical\_sales

Calculate safety\_stock as avg\_sales\_per\_day \* desired\_service\_level

Calculate reorder\_point as (avg\_sales\_per\_day \* lead\_time) + safety\_stock

Return reorder\_point

Function calculate\_reorder\_quantity(historical\_sales, lead\_time, holding\_cost, ordering\_cost)

Calculate avg\_sales\_per\_day as the average of historical\_sales

Calculate eoq as ((2 \* len(historical\_sales) \* ordering\_cost) / holding\_cost) \*\* 0.5

Calculate reorder\_quantity as avg\_sales\_per\_day \* lead\_time + eoq

Return reorder\_quantity

Function generate\_inventory\_report(products)

For each product in products:

Calculate turnover\_rate as product.current\_stock / (sum of current\_stock of all products / number of products)

Print "Inventory Turnover Rate for [product.name]: [turnover\_rate]"

Print "Stockout Occurrences for [product.name]: [Yes/No based on current\_stock vs reorder\_point]"

Print "Cost Implications of Overstock for [product.name]: [(current\_stock - reorder\_point) \* reorder\_quantity]"

Function test\_inventory\_system()

Create product1 with ID 1, name "Product A", current\_stock 50, reorder\_point 20, reorder\_quantity 30

Create product2 with ID 2, name "Product B", current\_stock 15, reorder\_point 10, reorder\_quantity 25

Create warehouse1 with ID 1, location "Warehouse A"

Add product1 and product2 to warehouse1's products

Print "Tracking Inventory:"

Call track\_inventory(warehouse1.products)

Print a blank line for readability

Print "Generating Inventory Report:"

Call generate\_inventory\_report(warehouse1.products)

Print a blank line for readability

For each product in warehouse1.products:

Print "Product: [product.name]"

Print "Current Stock: [product.current\_stock]"

Call recommend\_reorder(product)

Define historical\_sales as [10, 15, 20, 25]

Calculate reorder\_point using historical\_sales, lead\_time 7, desired\_service\_level 1.5

Calculate reorder\_quantity using historical\_sales, lead\_time 7, holding\_cost 2, ordering\_cost 1

Print "Reorder Point for [product.name]: [reorder\_point]"

Print "Reorder Quantity for [product.name]: [reorder\_quantity]"

Print a blank line for readability

Call test\_inventory\_system()

**Detailed explanation of the actual code:**

**Defining the Inventory System Structure**

* **Product**: Represents the product details such as its unique identifier (ID), name, reorder point (the stock level at which reordering is triggered), and reorder quantity (the amount to reorder when stock falls below the reorder point).
* **Warehouse**: Represents a physical location where products are stored. Each warehouse has a unique identifier, location details, and a list of products with their current stock levels.
* **InventoryLevel**: Tracks the stock level of each product within a specific warehouse. This includes the product details and the current quantity of that product in stock.

**Assumptions made (:if any)**

1. The company has a well-defined product catalog and warehouse locations.
2. Historical sales data and lead times are available for the inventory optimization algorithms.
3. The company has defined thresholds for reorder points and minimum order quantities.
4. The cost of holding inventory and placing orders are known.

**Limitations:**

1. The current implementation assumes constant lead times, which may not always be the case in real-world scenarios.
2. The demand forecasting algorithms are not included in the pseudocode, as they can be complex and require more detailed analysis of the company's sales patterns.
3. The user interface is only briefly mentioned, and a more comprehensive design would be required for a production-ready system.

**Code:**

class Product:

    def \_\_init\_\_(self, product\_id, name, current\_stock, reorder\_point, reorder\_quantity):

        self.product\_id = product\_id

        self.name = name

        self.current\_stock = current\_stock

        self.reorder\_point = reorder\_point

        self.reorder\_quantity = reorder\_quantity

class Warehouse:

    def \_\_init\_\_(self, warehouse\_id, location):

        self.warehouse\_id = warehouse\_id

        self.location = location

        self.products = []

def track\_inventory(products):

    for product in products:

        if product.current\_stock < product.reorder\_point:

            print(f"Alert: {product.name} is below the reorder point. Current stock: {product.current\_stock}")

            recommend\_reorder(product)

def recommend\_reorder(product):

    new\_stock = product.current\_stock + product.reorder\_quantity

    print(f"Recommended reorder for {product.name}: {product.reorder\_quantity} units. New stock level: {new\_stock}")

def calculate\_reorder\_point(historical\_sales, lead\_time, desired\_service\_level):

    avg\_sales\_per\_day = sum(historical\_sales) / len(historical\_sales)

    safety\_stock = avg\_sales\_per\_day \* desired\_service\_level

    reorder\_point = (avg\_sales\_per\_day \* lead\_time) + safety\_stock

    return reorder\_point

def calculate\_reorder\_quantity(historical\_sales, lead\_time, holding\_cost, ordering\_cost):

    avg\_sales\_per\_day = sum(historical\_sales) / len(historical\_sales)

    eoq = ((2 \* len(historical\_sales) \* ordering\_cost) / holding\_cost) \*\* 0.5

    reorder\_quantity = avg\_sales\_per\_day \* lead\_time + eoq

    return reorder\_quantity

def generate\_inventory\_report(products):

    for product in products:

        turnover\_rate = product.current\_stock / (sum([p.current\_stock for p in products]) / len(products))

        print(f"Inventory Turnover Rate for {product.name}: {turnover\_rate:.2f}")

        print(f"Stockout Occurrences for {product.name}: {'Yes' if product.current\_stock < product.reorder\_point else 'No'}")

        print(f"Cost Implications of Overstock for {product.name}: {(product.current\_stock - product.reorder\_point) \* product.reorder\_quantity:.2f}")

def test\_inventory\_system():

    # Define sample products and warehouses

    product1 = Product(1, "Product A", 50, 20, 30)

    product2 = Product(2, "Product B", 15, 10, 25)

    warehouse1 = Warehouse(1, "Warehouse A")

    warehouse1.products = [product1, product2]

    # Test tracking inventory

    print("Tracking Inventory:")

    track\_inventory(warehouse1.products)

    print()  # Blank line for readability

    # Test generating inventory report

    print("Generating Inventory Report:")

    generate\_inventory\_report(warehouse1.products)

    print()  # Blank line for readability

    # Display product details and recommendations

    for product in warehouse1.products:

        print(f"Product: {product.name}")

        print(f"Current Stock: {product.current\_stock}")

        recommend\_reorder(product)

        # Display historical data and reorder calculations

        historical\_sales = [10, 15, 20, 25]  # Example data

        reorder\_point = calculate\_reorder\_point(historical\_sales, 7, 1.5)

        reorder\_quantity = calculate\_reorder\_quantity(historical\_sales, 7, 2, 1)

        print(f"Reorder Point for {product.name}: {reorder\_point:.2f}")

        print(f"Reorder Quantity for {product.name}: {reorder\_quantity:.2f}")

        print()  # Blank line for readability

# Run the test function

test\_inventory\_system()

**Sample Output / Screen Shots**

