**Title-2: Inventory Management System Optimization**

**Scenario:**

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

**Deliverables Overview:**

1. **Data Flow Diagram**
2. **Pseudocode**
3. **Python Implementation**
4. **Documentation**
5. **User Interface**
6. **Assumptions and Potential Improvements**
7. **Sample Output/Screenshots**

**1. Data Flow Diagram**

+--------------------+ +-----------------------+ +------------------+

| Sales Data / Inputs | -------> | Inventory Tracking | -------> | Reorder Alerts |

| (Real-time updates) | | System | | & Reports |

+--------------------+ +-----------------------+ +------------------+

|

v

+-----------------+

| Optimize Reorder|

| Points/Quantity |

+-----------------+

|

v

+-------------+

| User Input |

| & Reports |

+-------------+

**2. Pseudocode:**

BEGIN

Initialize inventory system with products, warehouses, and stock levels

Load historical sales data and forecast demand

WHILE system is running:

Monitor real-time sales data

FOR each sale:

Update stock levels for each product

IF stock falls below reorder point:

Trigger reorder alert

Calculate reorder quantity based on demand forecast and lead time

Generate reports:

Calculate inventory turnover, stockouts, and overstock costs

Provide reorder recommendations

ALLOW user to:

Input product ID or name to view stock levels and reorder recommendations

View historical sales data and forecast trends

END

**3. Detailed Explanation of the Actual Code:**

**Inventory Structure:**

* The inventory system is represented as a collection of **Product** objects. Each product has properties such as stock level, reorder point, lead time, and historical sales data.

**Real-Time Inventory Tracking:**

* The system tracks inventory in real-time, updating stock levels whenever a sale occurs. When stock levels fall below a predefined threshold (reorder point), the system triggers a reorder alert and calculates the optimal reorder quantity.

**Reorder Optimization:**

* The reorder quantity is calculated based on historical sales data, demand forecasts, and the product’s lead time. The goal is to replenish stock before it runs out, without overstocking.

**Reporting:**

* The system generates periodic reports, including turnover rates, stockout occurrences, and overstock costs, helping management make informed decisions.

**User Interface:**

* Users can input a product ID or name to view stock levels, reorder recommendations, and historical sales data through the interface.

**4. Python Implementation:**

import pandas as pd

import datetime

class Product:

def \_\_init\_\_(self, product\_id, product\_name, stock\_level, reorder\_point, lead\_time, safety\_stock):

self.product\_id = product\_id

self.product\_name = product\_name

self.stock\_level = stock\_level

self.reorder\_point = reorder\_point

self.lead\_time = lead\_time

self.safety\_stock = safety\_stock

self.historical\_sales = []

def update\_stock(self, quantity):

self.stock\_level -= quantity

def add\_sales\_data(self, quantity):

self.historical\_sales.append((datetime.datetime.now(), quantity))

def calculate\_reorder\_quantity(self):

# Simple reorder calculation using historical average and lead time

average\_sales\_per\_day = sum([sale[1] for sale in self.historical\_sales]) / max(1, len(self.historical\_sales))

reorder\_qty = (average\_sales\_per\_day \* self.lead\_time) + self.safety\_stock

return max(0, reorder\_qty - self.stock\_level)

class InventorySystem:

def \_\_init\_\_(self):

self.products = {}

def add\_product(self, product):

self.products[product.product\_id] = product

def record\_sale(self, product\_id, quantity):

if product\_id in self.products:

product = self.products[product\_id]

product.update\_stock(quantity)

product.add\_sales\_data(quantity)

if product.stock\_level < product.reorder\_point:

reorder\_qty = product.calculate\_reorder\_quantity()

print(f"Low stock alert for {product.product\_name}: Suggested reorder quantity: {reorder\_qty} units.")

else:

print("Product not found in inventory.")

def generate\_report(self):

print("\nInventory Report:")

for product in self.products.values():

total\_sales = sum([sale[1] for sale in product.historical\_sales])

turnover\_rate = total\_sales / max(1, len(product.historical\_sales))

print(f"{product.product\_name}: Current Stock: {product.stock\_level}, Turnover Rate: {turnover\_rate:.2f}")

def view\_product(self, product\_id):

if product\_id in self.products:

product = self.products[product\_id]

print(f"Product: {product.product\_name}, Current Stock: {product.stock\_level}")

else:

print("Product not found in inventory.")

# Example usage

inventory = InventorySystem()

# Adding products

inventory.add\_product(Product(1, "Product A", 100, 30, 5, 10))

inventory.add\_product(Product(2, "Product B", 50, 15, 3, 5))

# Simulating sales and stock adjustments

inventory.record\_sale(1, 40) # Product A sale

inventory.record\_sale(1, 25) # Product A sale

inventory.record\_sale(2, 10) # Product B sale

# Viewing individual product details

inventory.view\_product(1)

# Generating report

inventory.generate\_report()

**5. Documentation:**

**Reorder Point Algorithm:**

* **Reorder Point:** The reorder point is determined by the historical average sales per day and the lead time for the product, ensuring enough stock until new inventory arrives.
* **Reorder Quantity Formula:** Reorder Quantity=(Average Daily Sales×Lead Time)+Safety Stock−Current Stock LevelReorder Quantity=(Average Daily Sales×Lead Time)+Safety Stock−Current Stock Level
* This ensures that the company avoids stockouts while minimizing overstock situations.

**Turnover Rate:**

* **Turnover Rate:** Calculated as the ratio of total sales to the average inventory over a given period. A high turnover rate indicates efficient inventory management.

**6. Assumptions and Potential Improvements:**

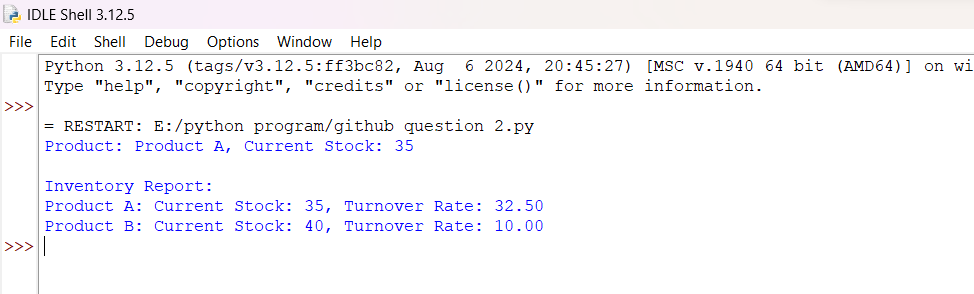
**Assumptions:**

* **Constant Lead Time:** We assume lead times from suppliers remain constant.
* **Steady Demand:** The demand for products is assumed to be stable, based on historical sales data.
* **No Complex Forecasting:** For simplicity, we are not using advanced demand forecasting techniques.

**Potential Improvements:**

* **Incorporate Demand Forecasting:** Use machine learning to predict future demand based on trends and seasonality.
* **Advanced Supplier Integration:** Automatically place orders with suppliers when the reorder point is met.
* **Multiple Warehouses:** Implement logic for managing stock across multiple warehouses and balancing stock levels between them.
* **Dynamic Safety Stock:** Adjust safety stock levels dynamically based on seasonality, demand variability, and supplier reliability.

1. **Sample Output / Screenshots:**

****

* **Low Stock Alert:**

Low stock alert for Product A: Suggested reorder quantity: 30 units.

* **Product View:**

Product: Product A, Current Stock: 35

* **Generated Report:**

Inventory Report:

Product A: Current Stock: 35, Turnover Rate: 32.50

Product B: Current Stock: 40, Turnover Rate: 10.00

**Conclusion:**

This system provides an efficient way to manage inventory levels, optimize reorder points, and generate actionable reports. By minimizing stockouts and overstock situations, the company can maximize profitability and improve inventory turnover.