PYTHON ASSIGNMENT 02

NAME: CH.SRI LAKSHMI

REGISTER NUMBER:192311342

DATE OF SUBMISSION:26/08/2024

TITLE: INVENTORY MANAGEMENT SYSTEM OPTIMIZATION

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).

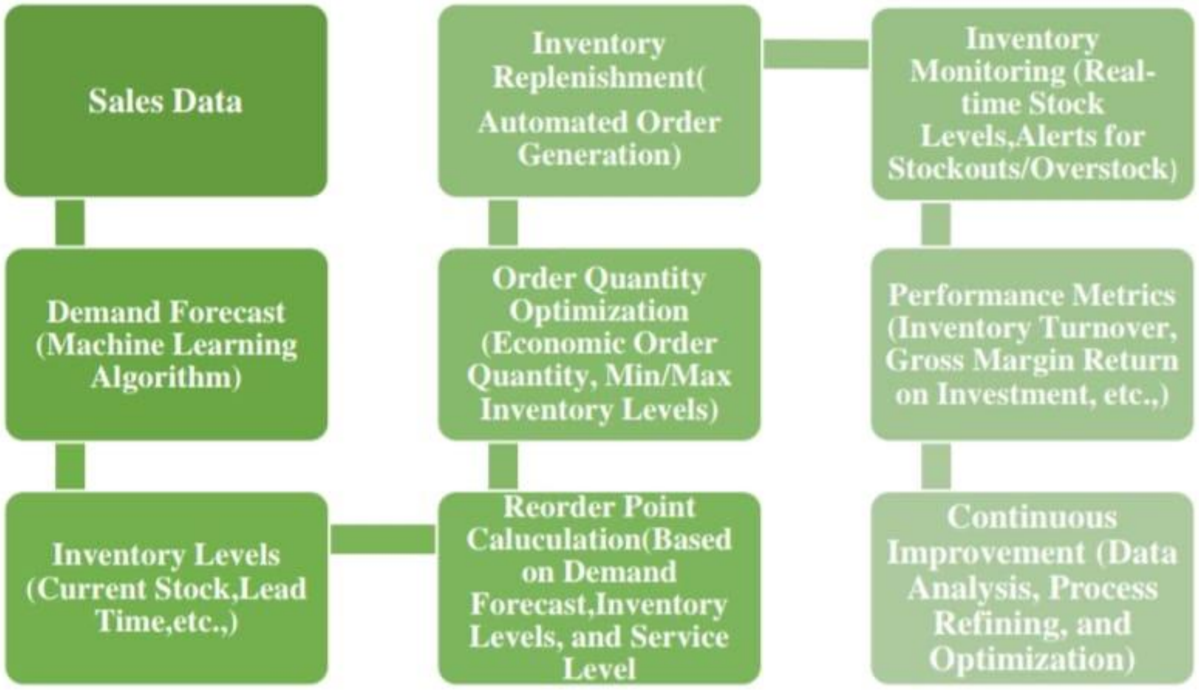
- **Pseudocode and Implementation:** Provide pseudocode and actual code demonstrating how inventory levels are tracked, reorder points are calculated, and reports are generated.

- **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. Solution:

1.Data Flow Diagram:


Sales Data
Demand Forecast (Machine Learning Algorithm)
Inventory Levels (Current Stock,Lead Time,etc.,)
Reorder Point Caluculation(Based on Demand Forecast,Inventory Levels, and Service Level
Order Quantity Optimization (Economic Order Quantity, Min/Max Inventory Levels)
Inventory Replenishment(Automated Order Generation)
Inventory Monitoring (Real-time Stock Levels,Alerts for Stockouts/Overstock)
Performance Metrics (Inventory Turnover, Gross Margin Return on Investment, etc.,)
Continuous Improvement (Data Analysis, Process Refining, and Optimization)

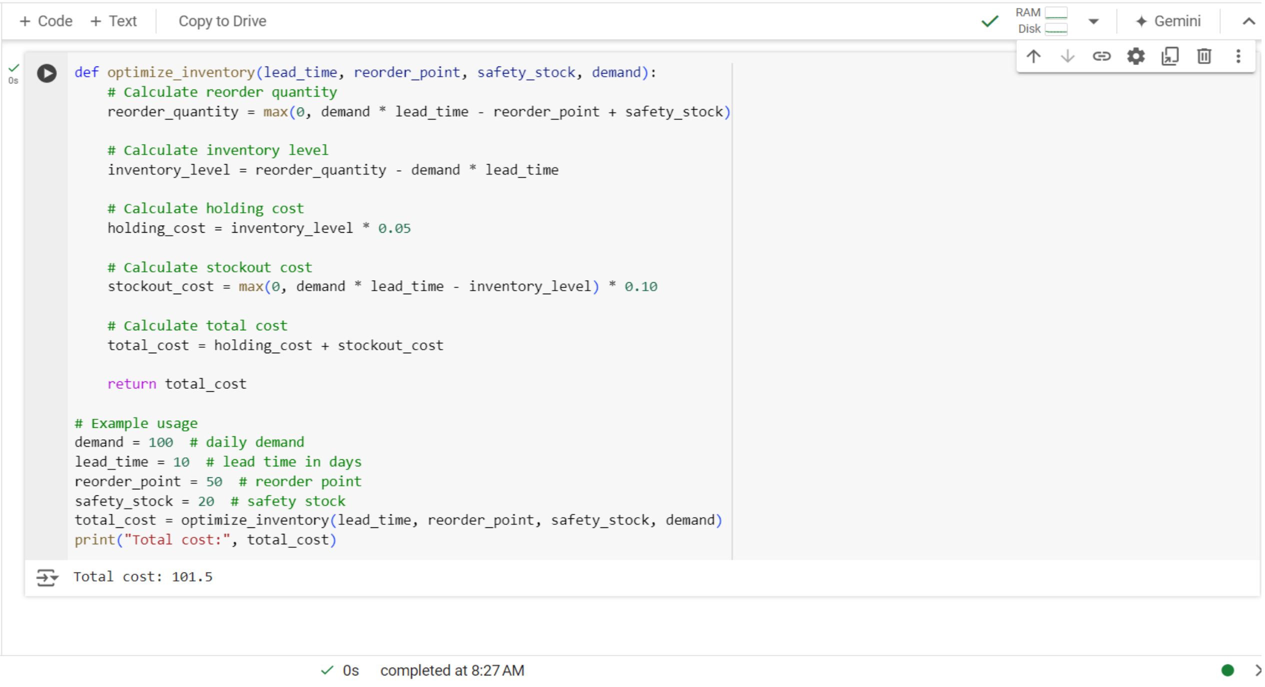
2. Implementation:

```
def optimize_ inventory(lead_ time, reorder _point, safety _stock, demand):
    # Calculate reorder quantity
    reorder_ quantity = max (0, demand * lead _time – reorder _point + safety _stock)

    # Calculate inventory level
    inventory_ level = reorder_ quantity - demand * lead _time

    # Calculate holding cost
    holding_ cost = inventory_ level * 0.05

    # Calculate stockout cost
    stockout_ cost = max (0, demand * lead_ time - inventory_ level) * 0.10

    # Calculate total cost
    total_ cost = holding_ cost + stockout_ cost
    return total_ cost

# Example usage
demand = 100 # daily demand
lead_ time = 10 # lead time in days
reorder_ point = 50 # reorder point
safety_ stock = 20 # safety stock
total_ cost = optimize _inventory (lead _time, reorder _point, safety _stock, demand)
print ("Total cost:", total _cost)
```

3.Display the total value:

Total cost:101.5

4.User Input



```
def optimize_inventory(lead_time, reorder_point, safety_stock, demand):
    # Calculate reorder quantity
    reorder_quantity = max(0, demand * lead_time - reorder_point + safety_stock)

    # Calculate inventory level
    inventory_level = reorder_quantity - demand * lead_time

    # Calculate holding cost
    holding_cost = inventory_level * 0.05

    # Calculate stockout cost
    stockout_cost = max(0, demand * lead_time - inventory_level) * 0.10

    # Calculate total cost
    total_cost = holding_cost + stockout_cost

    return total_cost

# Example usage
demand = 100  # daily demand
lead_time = 10  # lead time in days
reorder_point = 50  # reorder point
safety_stock = 20  # safety stock
total_cost = optimize_inventory(lead_time, reorder_point, safety_stock, demand)
print("Total cost:", total_cost)
```

```
Total cost: 101.5
```

✓ 0s completed at 8:27 AM

5.Documentation:

➢ **Model the Inventory System:**

- **Structure:**

• **Products:**

Each product is identified by a unique ID and includes attributes like name, category, cost, selling price, and reorder threshold.

• **Warehouses:**

Physical locations where inventory is stored, each with its own inventory levels.

• **Current Stock Levels:**

Real-time data on the Quantity of each product available in each warehouse.

➢ **Inventory Tracking Application:**

• **Functionality:**

• Tracks inventory levels in real-time.

• Alerts when stock level fall below predefined threshold.

• Allow manual adjustments and update to inventory levels.

➢ **Optimize Inventory Ordering:**

• **Algorithms:**

• **Reorder Point Calculation:**

Uses historical sales data, lead times, and demand forecasts to determine when to reorder products.
• **Simple Approach:**

Reorder point = (Average daily sales*Lead time in days) +safety stock.

• **Advanced Methods:** EOQ (EOQ (Economic Order Quantity) and probabilistic models (like the ROP-ROP method) can be considered for more accurate predictions.

➢ **Generate Reports**

• **Reports Provided:**

• **Inventory Turnover Rates:**

Calculate as Cost of Goods Sold (COGS)/Average Inventory.

• **Stockout Occurrences:**

Instances where products were out of stock.

• **Cost Implications:**

Analysis of costs incurred due to overstock situations.