Assignment: Python Programming for GUI Development

Name: A.Meghanadh

Register Number: 192372281

Department: CSE(AI)

Date of Submission: 29.8.2024

Problem 1: 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:

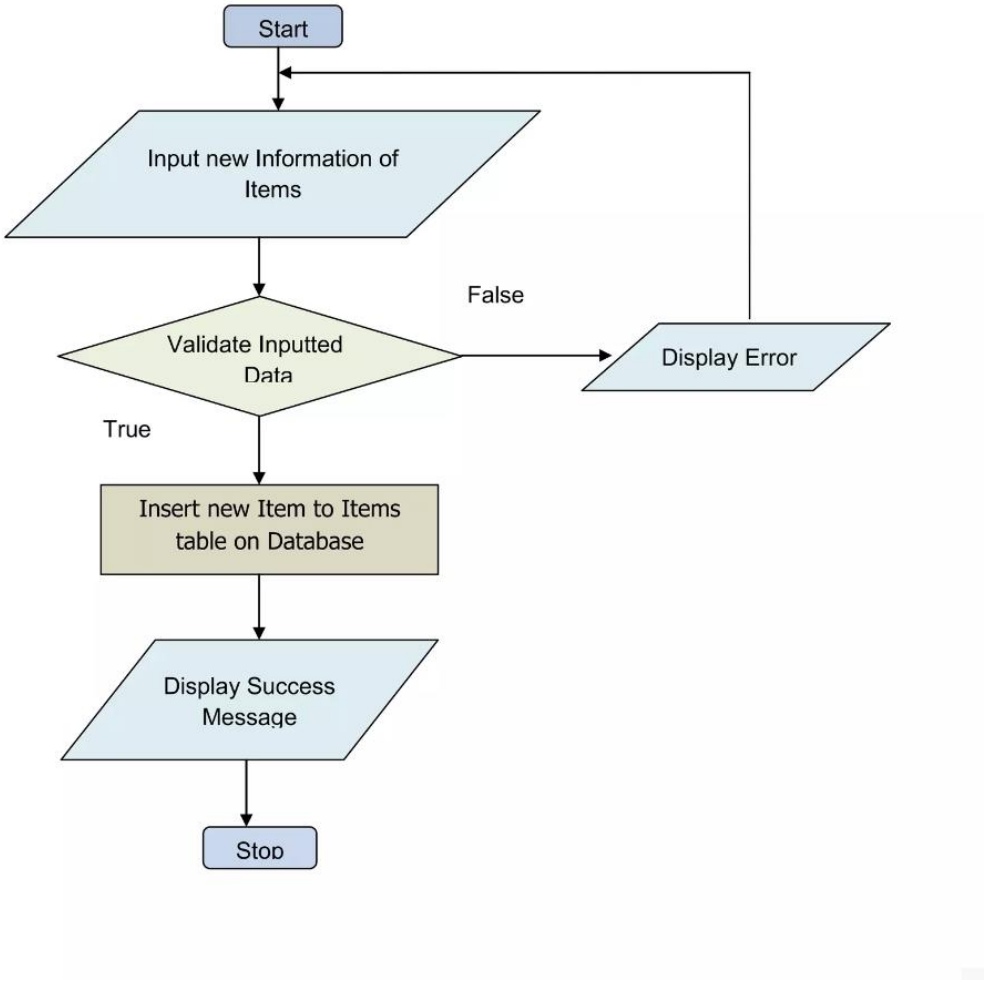
1. **Model the inventory system**: Define the structure of the inventory system, including products, warehouses, and current stock levels.
2. **Implement an inventory tracking application**: Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below 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:

Real-Time Weather Monitoring System



1.Data Flow Diagram

2. Implementation

```
inventory = {}

def add_item(name, reorder_level):
    inventory[name] = {'stock': 0, 'reorder_level':
reorder_level}

def update_inventory(name, quantity, transaction_type):
    if transaction_type == "purchase":
        inventory[name]['stock'] += quantity
    elif transaction_type == "sale":
        inventory[name]['stock'] -= quantity

def check_and_reorder():
    for name, details in inventory.items():
        if details['stock'] < details['reorder_level']:
            print(f"Reorder needed for {name}: Current stock
= {details['stock']}")

# Example usage
add_item("Item A", 17)
update_inventory("Item A", 20, "purchase")
update_inventory("Item A", 5, "sale")
add_item("Item B",20)
update_inventory("Item B", 20, "purchase")
update_inventory("Item B", 7, "sale")
check_and_reorder()
```

3.Display the Current weather information

Number of items A:17
Inventory purchase: 20
Sale:5
Number of items B: 20
Inventory purchase: 20
Sale: 7

4.User Input

```
inventory = ()

def add item(name, reorder level): inventory Iname('stock": 0, "reorder level: reorder_level)

def update inventory(name, quantity, transaction_type):

If transaction type "purchase":

inventory [name] [stock] + quantity

niif transaction type="sale":

inventory [name]['stock quantity

def check and_reorder():

for name, details in inventory.items():

if details! stock'] details reorder_level'li

printf Reorder needed for (name): Current stock (details'stock"]}")

Exangle usage

add item("Item A", 171 update inventory("Item A, 20, "purchase")

update inventory("Item A, 5, "sale") add_item("Item B",201

update Inventory!"Item ", 20, "purchase") update inventory("item ", 7, "sale")

check_and_reorder()
```

5.Documentation

Algorithms

Inventory management systems use reorder optimization algorithms to ensure that stock levels are maintained efficiently, preventing both stockouts and overstock situations.They help balance trade-offs between holding costs, ordering costs, stockouts, and overstocks. They can also consider other factors like lead times, service levels, discounts, and promotions.

How Historical Data Influences Decisions

Algorithms analyze past sales trends to predict future demand. Techniques like moving averages, exponential smoothing, or more advanced machine learning models
Historical lead time data helps refine the assumptions used in the reorder point calculation.
By tracking when stockouts or overstock events occurred, the system learns to adjust reorder parameters to avoid repeating mistakes.

Conclusion

Reorder optimization algorithms, influenced by historical data, aim to ensure efficient inventory management by balancing costs, demand, and supply constraints. Assumptions such as constant lead times and demand rates simplify the system but may need periodic adjustment to account for real-world variability. These algorithms are key to maintaining optimal stock levels and avoiding costly stockouts or excess inventory.