Replenishment Engine Project Documentation

1. Introduction

This document details the functionality and operational logic of the Replenishment Engine. This system is designed to optimize inventory levels across pharmacy branches by automating the identification of stock requirements, managing stock allocation from a central warehouse, facilitating external procurement (Local Purchase Orders - LPOs), and identifying excess inventory.

2. Core Business Requirements

The Replenishment Engine addresses the following key business objectives:

- **2.1. Identify Branch Replenishment Needs:** Determine which products at each branch require additional stock based on predefined minimum inventory levels.
- **2.2. Stock Allocation and LPO Generation:** Allocate available stock from the central warehouse to fulfill branch requirements. For any unfulfilled demand, generate LPOs for external procurement.
- **2.3. LPO Management:** Facilitate the generation and tracking of Local Purchase Orders.
- **2.4. Excess Stock Identification:** Identify and flag products at branches that hold inventory exceeding optimal levels, based on sales velocity.

3. Operational Logic: Step-by-Step Flow

The engine processes inventory data for every product across all pharmacy branches through a systematic flow:

3.1. Step 1: Identify Branch Replenishment Needs

Objective: Determine if a product at a specific branch requires additional stock and calculate the quantity needed.

Process: 1. The system compares the `Branch_Stock` (current quantity) of each product at a branch against its `Min_Stock` (minimum required quantity). 2. If `Branch_Stock` is less than `Min_Stock`, a replenishment need is identified. 3. The `ReorderQty` (quantity to reorder) is calculated to bring the `Branch_Stock` up to its `Max_Stock` (maximum desired quantity).

Calculation: `ReorderQty = Max_Stock - Branch_Stock` (if `Branch_Stock < Min_Stock`, otherwise `ReorderQty = 0`)

Dynamic Min/Max Stock: `Min_Stock` and `Max_Stock` are not static values. They are dynamically calculated based on the product's `Sales_30D` (total sales over the last 30 days) and `Lead_Time_Days` (time taken for a product to arrive from the supplier).

- **Average Daily Sales (ADS):** `Sales_30D / 30`
- **Min_Stock:** Calculated to cover demand during lead time plus a safety buffer. (e.g., `ADS * Lead_Time_Days * Safety_Stock_Factor`). A minimum floor is applied to prevent zero or near-zero minimums for slow-moving items.
- **Max_Stock:** Typically set as a multiple of `Min_Stock` (e.g., `Min_Stock * Max_Stock_Factor`).

Example:

SKU	Branch	Branch Stock	Min	Max	Sales_30D
A123	Al Quoz	10	15	40	30

In this example, `Branch_Stock` (10) is less than `Min_Stock` (15). The `ReorderQty` is calculated as `40 - 10 = 30` units.

This check is performed for every product at every branch.

3.2. Step 2: Check Warehouse for Stock Availability

Objective: Determine if the central warehouse can fulfill the identified branch replenishment needs.

Process: 1. For each product requiring replenishment at a branch, the system checks the current `Warehouse_Stock` for that product. 2. **Important Note on Warehouse Stock:** The `Warehouse_Stock.csv` is treated as a static input for each run of the engine. The engine uses a snapshot of the warehouse inventory at the start of the calculation. While the system internally simulates allocation by decrementing an internal copy of the available stock, the original `Warehouse_Stock.csv` file is not modified by the engine. This reflects a scenario where the replenishment run is based on a specific point-in-time inventory record.

Allocation Logic: * **Full Fulfillment:** If `Warehouse_Stock` is greater than or equal to the `ReorderQty` for a branch, the entire `ReorderQty` is allocated from the warehouse. * **Partial Fulfillment (Edge Case Handling):** If `Warehouse_Stock` is less than the `ReorderQty`, the system allocates only the available `Warehouse_Stock` to the branch. The remaining unfulfilled quantity is then flagged for an LPO.

Example (Full Fulfillment):

SKU	Reorder Needed	Warehouse Stock
A123	30	100

In this case, the warehouse has enough stock (100 >= 30). A transfer order for 30 units will be created.

Example (Partial Fulfillment):

SKU	Reorder Needed	Warehouse Stock
B456	40	15

Here, the warehouse has insufficient stock (15 < 40). The system will: * Allocate 15 units from the warehouse (creating a transfer order). * Flag the remaining 25 units (40 - 15 = 25) for a Local Purchase Order (LPO). This demonstrates precise LPO quantity identification.

3.3. Step 3: Create Transfer Orders

Objective: Generate a detailed list of products to be moved from the central warehouse to specific branches.

Process: For every product and branch where stock was allocated from the warehouse (either fully or partially), a record is created in the Transfer Orders list.

Output Format:

SKU	From_Warehouse	To_Branch	Transfer_Qty
A123	WH01	Al Quoz	30
B456	WH01	Deira	15

3.4. Step 4: Prepare and Flag LPO Needs

Objective: Consolidate and list all products that require external procurement from suppliers due to insufficient warehouse stock.

Process: For every product that could not be fully fulfilled by the warehouse, an LPO need is generated. These needs are grouped by SKU and Vendor to create consolidated purchase requests.

Output Format:

SKU	Required_Qty	Vendor
B456	25	Vendor B
D789	100	Vendor C

3.5. Step 5: LPO Management and Tracking

Objective: Provide a mechanism for managing and tracking generated LPOs.

Process: For the current prototype phase, the "sending" of LPOs is simulated by generating a `LPO_Needs.csv` file. In a production environment, this step would involve integration with procurement systems (e.g., sending emails to vendors, updating an ERP system, or displaying alerts on a dashboard).

The system also provides a count of the total LPOs generated in each run, fulfilling the requirement for tracking.

3.6. Step 6: Identify Excess Stock

Objective: Identify products at branches that are overstocked, enabling proactive inventory balancing and reducing holding costs.

Process: Excess stock is determined using a "Days of Stock" (DOS) metric, which relates current inventory levels to recent sales velocity.

Calculation: 1. **Average Daily Sales (ADS):** Calculated as `Sales_30D / 30`. 2. **Target Stock for Excess:** This is the stock level considered "optimal" for a given number of days. It is calculated as `ADS * Configurable_Excess_DOS_Threshold`. * **Edge Case:** If `ADS` is 0 (no sales in the last 30 days), the `Target Stock for Excess` is considered 0 to ensure any stock is flagged as excess. 3. **Excess Quantity:** Calculated as `Branch_Stock - Target_Stock_for_Excess`. Only positive values indicate excess.

Configurable Parameter: The `Configurable_Excess_DOS_Threshold` is a parameter that can be adjusted (defaulting to 60 days in the current implementation). This allows stakeholders to define what constitutes "excess" based on business policy.

Example: If a branch has `Sales_30D` of 30 units (meaning 1 unit/day ADS) and `Branch_Stock` of 80 units, with an `EXCESS_DOS_THRESHOLD` of 70 days: * `Target Stock for Excess` = 1 unit/day * 70 days = 70 units. * `Excess Quantity` = 80 (Branch_Stock) - 70 (Target Stock) = 10 units. This means the branch has 10 units of excess stock, equivalent to 10 days of supply.

Output Format: The identified excess stock is outputted with details such as Branch, SKU, Product Name, current Branch Stock, Sales_30D, Average Daily Sales, the calculated Target Stock for Excess, and the Excess Quantity.

4. Outputs

Upon completion of each replenishment engine run, the following CSV files are generated in the `outputs/` directory:

- **`Transfer_Orders.csv`:** Details all products and quantities to be transferred from the central warehouse to specific branches.

- **`LPO_Needs.csv:`** Lists all products, required quantities, and associated vendors for external procurement.
- **`Excess_Stock.csv:`** Identifies products at branches that are overstocked, including the calculated excess quantity based on Days of Stock.