INVENTORY OPTIMIZATION MODEL USING PYTHON FOR A MULTI-STORE RETAIL CHAIN IN THE U.S.

1. Background

A mid-sized U.S. retail chain with 60 stores and a central warehouse faced recurring inventory issues—stockouts during peak demand, overstocking for slow-moving items, and excessive manual planning overhead. Their operations team used spreadsheets but lacked a scalable model to standardize reorder quantities and stock safety buffers across all locations.

We were engaged to design a robust, Python-based inventory optimization model tailored to their item-level demand patterns. The solution would help automate reordering decisions using Economic Order Quantity (EOQ), safety stock buffers, and demand variability—all structured for real business implementation.

2. Objective

- To calculate the optimal order quantity per item using EOQ
- To incorporate safety stock logic based on demand variability and lead times
- To provide a Python-based inventory management script that outputs reorder quantity, reorder point, and restocking schedule
- To support store-level and chain-wide inventory decisions with structured reporting

3. Data Used

Source: Internal ERP data exports (CSV format)

Data Details:

- 14,300 SKUs across 60 stores
- Timeframe: Last 12 months of weekly sales data
- Fields:
    - Item_ID, Store_ID, Weekly_Sales_Units, Unit_Cost, Lead_Time_Days, Order_Cost_Per_Order, Holding_Cost_Per_Unit_Per_Year

Assumptions Added:

- Service level = 95%

- Working days = 360/year
- Demand assumed to follow normal distribution for safety stock calculation

4. Methodology

4.1 EOQ Calculation

$$EOQ = \sqrt{\frac{2 \times D \times S}{H}}$$

Where:

- D: Annual demand
- S: Order cost per order
- H: Annual holding cost per unit

4.2 Safety Stock and Reorder Point

$$Safety\ Stock = Z \times \sigma_d \times \sqrt{L}$$

$$Reorder_Point = \left(d_{avg} \times L\right) + Safety\ Stock$$

Where:

- Z = 1.65 for 95% service level
- σ_d: Standard deviation of daily demand
- L: Lead time (in days)
- d_{avg}: Average daily demand

4.3 Python Implementation

- Used pandas for data transformation
- Applied NumPy for EOQ and safety stock formulas
- Created functions to output per SKU:
    - EOQ
    - Reorder Point
    - Order Frequency (in days)
    - Inventory Turnover Ratio

5. Optimization Results

- **Average EOQ Accuracy**: ±9.3% vs actual consumption post-deployment
- **Safety Stock Accuracy**: 94.8% SLA met using modeled reorder points
- **Manual order frequency** reduced from weekly guesswork to once every 15–28 days per item
- Automated restock schedule generated per SKU with adjustable service level input

6. Insights and Recommendations

- **High-velocity items** (15% of SKUs) drove 70% of restock cost—modeled for tighter reorder points
- Stores in high footfall zones required shorter reorder intervals
- Suggested **centralized order batching** for items with similar lead times to reduce order cost
- Model flagged **excess safety stock** for 2,300 low-velocity items—client paused procurement for 90 days

7. Reporting Output

- **Python Script**:
  - Accepts SKU-wise input CSV
  - Calculates EOQ, safety stock, and reorder points
  - Generates reorder schedule and inventory health index
  - Exports Restock_Report.csv with reorder plan per item per store
- **PDF Report (12 pages)**:
  - Visual summaries of EOQ vs historical orders
  - Heatmap of reorder risk zones across stores
  - Inventory turnover ratio tables by category
- **Excel Tracker**:
  - Dashboard for adjusting service level, order cost, and lead time assumptions
  - Color-coded reorder alert flags by item

8. Operational Impact

- **Overstock rate** dropped from 28% to 21% in 60 days
- Stockout complaints reduced by 22% at high-volume stores
- Reorder planning time cut by 65%, freeing planners for higher-value decisions
- EOQ logic now embedded in their quarterly forecasting cycle