Retail Inventory Analysis

Anirudha Thakur

Database Setup

The database schema inventory_project_sql contains tables: DimProduct (containing all the product details, ID and category), DimStore (containing the store information, ID and region), DimDate (dates from 01-Jan-2022 and 31-Dec-2023) and FactInventorySales (with information derived from the inventory_forecasting.csv dataset). The database schema is defined in 0. schema_data.sql. It normalizes the raw inventory dataset into a relational schema to ensure data integrity and query efficiency. Key tables include:

- DimProduct: Stores product information like ProductID and Category.
- DimStore: Contains store details such as StoreID, Region, and a surrogate key StoreSK.
- DimDate: Stores date-related attributes like PDate, PYear, PMonth, PDay, PDayOfWeek, and Seasonality.
- FactInventorySales: The central fact table linking the dimension tables. It includes metrics like InventoryLevel, UnitsSold, UnitsOrdered, DemandForecast, Price, and Discount. The data from the csv file are loaded into a temporary table (temp_inventory_data) before being inserted into these dimension and fact tables.

Summary of the Key SQL Views

The SQL views in 1.analysis_compiled.sql provide direct data outputs for inventory analysis:

- ViewABCAnalysis: Outputs ABCCategory ('A', 'B', 'C') for products based on revenue contribution, alongside TotalRevenue, CumulativePercentage, TotalUnitsSold, and AverageInventoryLevel.
- ViewInventoryTurnover: Calculates and outputs InventoryTurnoverRatio and DaysInInventory for each product per store, along with TotalUnitsSold and AvgInventoryLevel.
- ViewCurrentStockLevels: Reports the CurrentStockLevel and LastRecordedDate for products.
- ViewReorderPoints: Calculates and outputs ReorderPoint, AvgDailyDemand, and SafetyStock for each product and store.
- ViewProductMovementClassification: Outputs MovementCategory ('Fast-moving', 'Mediummoving', 'Slow-moving') based on TotalSold from ViewProductStats (statistics of the products).
- ViewInventoryActions: Provides a RecommendedAction (e.g., 'URGENT REORDER around: X units', 'REDUCE by Y units', 'MAINTAIN current level'), InventoryStatus ('LOW', 'NOT LOW', 'ADEQUATE'), MovementCategory, and DaysOfStockRemaining.
- ViewStockMovementTrends: Outputs AvgStock30Days, MinStock30Days, MaxStock30Days, and StockVolatility.
- ViewProductDailySales: Outputs DailySales, DailyInventory, and DailyForecast for each product.
- ViewProductAverageDailySales: Outputs AvgDailySales, AvgDailyForecast, and AvgForecastAccuracy.
- ViewCompetitorPricingImpact: Outputs Avg_OurPrice, Avg_CompetitorPrice, DifferencePercent, sales figures when priced higher vs. lower, and a PricingRecommendation (e.g., 'Reduce Price', 'Maintain Competitive Pricing', 'Monitor Competitor Response').

- ViewSeasonalDemandAnalysis: Outputs AvgMonthlySales, OverallAvgSales, WeatherConditionIndex, and SeasonalPattern ('High Season', 'Low Season', 'Regular Season') based on monthly and overall statistics.

- weather_stats: Outputs AvgSeasonalSales, AvgSeasonalDemand, and Turnover grouped by WeatherCondition.

ABC Analysis

- Purpose: Classify products into A/B/C categories by cumulative revenue per store. Top 85% is in A, next 15% is in B and remaining in C.

- Usage: Used to prioritize high-revenue products for stock control and promotion.

StoreID	Identifier for store location
Region	Geographic region of the store
ProductID	Unique product identifier
ABCCategory	Category A, B or C
TotalRevenue	Total revenue per product
InventoryTurnoverRate	Ratio of sales to average inventory

We can use SELECT * FROM ViewABCAnalysis WHERE ProductID LIKE 'P0016' (example) to observe the store-wise categorization of P0016.
We have kept a window of 30 days for analysis. The placeholder end-date of the window is kept as '2023-12-31'. One can change it as desired.

Suggested Actions

In each store,

- Prioritize 'A' Items: Implement stricter inventory control, more frequent monitoring, and potentially higher service level targets for 'A' items, as they are critical revenue drivers.

- Optimize 'B' Items: Maintain moderate control and standard service levels.

- Manage 'C' Items: Consider more lenient controls, potentially lower stock levels or even delist the continuously low performing items. Regularly review these items.

Inventory Turnover Analysis

- Analyze how efficiently products are sold compared to the average inventory held using the view ViewInventoryTurnover

- Key Metrics: Inventory Turnover Ratio and Days in Inventory

- Usage: Check the turnover statistics to make informed decisions about the products.

We used the query SELECT * FROM ViewInventoryTurnover WHERE StoreID LIKE 'SOO1' AND Region LIKE 'North' LIMIT 5; to obtain Figure 1.

ProductID	**StoreID**	**Region**	**TotalUnitsSold**	**AvgInventoryLevel**	**InventoryTurnoverRatio**	**DaysinInventory**	**TotalRevenue**	**AvgSellingPrice**
P0068	S001	North	18832	131.3394	143.3843	2.5456	987066.77	50.995792
P0187	S001	North	21387	156.915	136.2967	2.678	1050735.52	49.01815
P0046	S001	North	20154	150.4456	133.962	2.7247	1002707.85	49.689534
P0125	S001	North	21122	159.5	132.4263	2.7562	1040242.19	49.589394
P0066	S001	North	20257	155.7097	130.0947	2.8056	1048909.04	51.759946

Table 1: Top 5 of S001-North sorted according to Turnover Ratio

Figure 1. Top 5 of S001-North sorted according to Turnover Ratio

Stock Recommendations

Logic and Description

This section focuses on identifying current stock levels, calculating reorder points, and classifying product movement to recommend inventory actions.

- ViewCurrentStockLevels: Provides the most recent inventory level for each product at each store.
- ViewReorderPoints: Estimates reorder points based on average daily demand, lead time (assumed as 1 day), and safety stock. The formulae used are:
- Assuming normally distributed demand, the safety stock SS at a 95% service level (using a z-score of 1.65) is:

$$SS = z \cdot \sigma$$

Where:

- $z = 1.65$ (z-score for 95% service level)
- σ is the standard deviation of daily demand

SQL equivalent code is: CEIL (COALESCE (1.65 * STDDEV (UnitsSold) , 0)) AS SafetyStock (CEIL for an integer value).

- The reorder point ROP is the sum of expected demand during lead time and the safety stock:

$$ROP = D \cdot L + SS$$

Where L is the lead time in days. For $L = 1$, this simplifies to: $ROP = D + SS$. SQL equivalent code is: FLOOR ((AVG(UnitsSold) * 1) + COALESCE (1.65 * STDDEV (UnitsSold), 0)) AS ReorderPoint (FLOOR for an integer value).

- ViewProductMovementClassification: Classifies products as 'Fast-moving', 'Medium-moving', or 'Slow-moving' based on sales in the last 30 days.
- ViewInventoryActions: Combines the above views to provide specific recommendations like 'URGENT REORDER', 'INCREASE', 'REDUCE', or 'MAINTAIN current level'.

These interconnected views provide current stock levels, calculated reorder points, product movement speeds (fast, medium, slow), and direct recommendations for action (e.g., "URGENT REORDER", "REDUCE by X units").

Suggested Actions

- Execute Urgent Reorders: Immediately act upon "URGENT REORDER" recommendations to prevent stockouts of critical and fast-moving items.
- Implement Stock Reductions: For products flagged with "REDUCE" recommendations, especially slow-movers, reduce the stock levels to free up working capital.
- Refine Reorder Parameters: Periodically review and adjust the LeadTimeDays and safety stock logic (e.g., the multiplier for DemandStdDev) in ViewReorderPoints to ensure reorder calculations remain accurate as business conditions change. (This is IMPORTANT!)
- Prioritize Actions: Use the DaysOfStockRemaining and MovementCategory fields in ViewInventoryActions to prioritize which actions need the most immediate attention.

Competitor Pricing Impact

- Purpose: Evaluate how our product pricing compares to competitors and its effect on sales and revenue.
- View Used: ViewCompetitorPricingImpact
- Usage: Make dynamic pricing decisions to stay competitive.
- Example: SELECT * FROM ViewCompetitorPricingImpact WHERE PricingRecommendation LIKE 'Reduce Price'

Logic Behind Calculation

```
-- Snippet from ViewCompetitorPricingImpact
SELECT
    ProductID,
    AVG(Price) AS Avg_OurPrice,
    AVG(CompetitorPricing) AS Avg_CompetitorPrice,
    CASE
        WHEN AVG(CASE WHEN Price > CompetitorPricing THEN UnitsSold ELSE NULL
    END) < AVG(CASE WHEN Price <= CompetitorPricing THEN UnitsSold ELSE NULL
    END) * 0.8
            AND AVG(Price - CompetitorPricing) > 0
        THEN 'Reduce Price'
        -- ... other conditions ...
        ELSE 'Monitor Competitor Response'
    END AS PricingRecommendation
FROM FactInventorySales
```

We recommend reducing the price when the number of units sold with a price higher than the competitor falls to 80% of the number of units sold with a price lower than the competitor. The ' 80% ' here is purely speculative and is a safe bound. More detailed and dynamic analysis can (and should) be done depending on the product.

Suggested Actions

- Evaluate the "Reduce Price" Recommendations: For products where a price reduction is suggested and data indicate a potential positive impact on sales volume (as per the columns AvgSalesWhenPricedLower vs. AvgSalesWhenPricedHigher), carefully consider implementing price adjustments, balancing volume gain with margin impact.
- Monitor Other Recommendations: For "Maintain Competitive Pricing" or "Monitor Competitor Response", establish a process for regular review of competitor pricing for these items to ensure continued optimal positioning.

Concluding Remarks

We did not add any specific dashboard summary as the data vary from store to store. We did try a executive summary but the data seemed vague and would've not helped in accurate estimation of the troubles.

One of the key challenges are improving the data visualization and making the setup more userfriendly. But we figured it would require specialization way more than SQL and hence chose to conveniently ignore them. Other important challenges include:

- Limited integration with real-time data.
- Static analysis windows: most queries depend on fixed date cutoffs. Dynamic management of dates need to be implemented.
- Manual updating of date parameters and thresholds, which affects automation.

- The analysis is on historic data, and the bounds/suggestions and their performance are not tested on future data. Most of the parameters in the logic are decided by us and are not tested further. This part needs careful consideration.