Given the business problem and project objectives, here are the steps to perform data analytics using SQL for Supply Chain Intelligence:
The detailed breakdown you provided is methodical and covers all aspects of a thorough data analysis pipeline.

Comprehensive Project Plan:

1. **Data Exploration**:
   - Load and inspect the datasets.
   - Use `SELECT` statements combined with `LIMIT` to preview the first few rows of each table.
   - Check the data types and structure using `DESCRIBE` or equivalent commands.

Steps taken;

- Load tha tables using the table import wizard and inspect the datasets(limited view of the datasets and also the describe function to get an idea of the data type and structure of the datasets
- Inspecting tables;
- SELECT * FROM external_factors LIMIT 5;

Sales Date	GDP	Inflation Rate	Seasonal Factor
13/09/2021	24668.89	2.12	1.06
23/10/2021	23770.38	2.04	1.03
20/03/2018	19363.94	4.94	0.85
06/06/2019	18642.21	1.28	1.07
30/06/2021	20071.23	3.67	0.93

- SELECT * FROM product_data LIMIT 5;

Product ID	Product Category	Promotions
1387	SmartPhones	Yes
7548	Electronics	Yes
5349	SmartPhones	No
4114	Electronics	Yes
6409	Electronics	No

- SELECT * FROM sales_data LIMIT 5;

Product ID	Sales Date	Inventory Quantity	Product Cost
1387	13/09/2021	25	85.71
7548	23/10/2021	69	72.92
5349	20/03/2018	14	123.35
4114	06/06/2019	6	144.67
6409	30/06/2021	41	136.04

- Describe function to get an idea of the data type and structure of the datasets.
- SHOW COLUMNS FROM external_factors;

Field	Type	Null	Key	Default	Extra
Sales Date	text	YES		NULL	
GDP	double	YES		NULL	
Inflation Rate	double	YES		NULL	
Seasonal Factor	double	YES		NULL	

- DESCRIBE product_data;

Field	Type	Null	Key	Default	Extra
Product ID	int	YES		NULL	
Product Category	text	YES		NULL	
Promotions	text	YES		NULL	

- DESC sales_data;

Field	Type	Null	Key	Default	Extra
Product ID	int	YES		NULL	
Sales Date	text	YES		NULL	
Inventory Quantity	int	YES		NULL	
Product Cost	double	YES		NULL	

- Ideally this is supposed to be the structure of the tables in the schema, it seems some columns are not meeting standards.

Sale_Data (
ProductID INT NOT NULL,
SalesDate DATE,
Inventory_ Quantity INT,
ProductCost DECIMAL(10, 2),
PRIMARY KEY (ProductID, SalesDate)

Product_Data (
ProductID INT NOT NULL,
ProductCategory TEXT
Promotions ENUM('yes', 'no'),
PRIMARY KEY (ProductID)

External_Data (
SalesDate DATE,
GDP DECIMAL(15, 2),
InflationRate DECIMAL(5, 2),
SeasonalFactor DECIMAL(5, 2),
PRIMARY KEY (SalesDate)

2. Data Cleaning:

- Change the datatypes to the appropriate datatype
- **External data**

ALTER TABLE external_factors
ADD COLUMN New_Sales_Date DATE;
SET SQL_SAFE_UPDATES = 0; -- turning off safe updates
UPDATE external_factors
SET New_Sales_Date = STR_TO_DATE(Sales_Date, '%d/%m/%Y');
ALTER TABLE external_factors
DROP COLUMN Sales_Date;
ALTER TABLE external_factors
CHANGE COLUMN New_Sales_Date Sales_Date DATE;

ALTER TABLE external_factors
MODIFY COLUMN GDP DECIMAL(15, 2);

ALTER TABLE external_factors
MODIFY COLUMN Inflation_Rate DECIMAL(5, 2);

ALTER TABLE external_factors
MODIFY COLUMN Seasonal_Factor DECIMAL(5, 2);

- Checking to see if it was incorporated

Field	Type	Null	Key	Default	Extra
GDP	decimal(15,2)	YES		NULL	
Inflation_Rate	decimal(5,2)	YES		NULL	
Seasonal_Factor	decimal(5,2)	YES		NULL	
Sales_Date	date	YES		NULL	

- **Product Data**

ALTER TABLE product_data
ADD COLUMN NewPromotions ENUM('yes', 'no');
UPDATE product_data
SET NewPromotions = CASE
WHEN Promotions = 'yes' THEN 'yes'

```
 WHEN Promotions = 'no' THEN 'no'
 ELSE NULL
END;
ALTER TABLE product_data
DROP COLUMN Promotions;
ALTER TABLE product_data
CHANGE COLUMN NewPromotions Promotions ENUM('yes', 'no');
```

- Checking;

Field	Type	Null	Key	Default	Extra
Product_ID	int	YES		NULL	
Product _Category	text	YES		NULL	
Promotions	enum('yes','no')	YES		NULL	

- **Sales table**

```
ALTER TABLE Sales_data
ADD COLUMN New_Sales_Date DATE;
UPDATE Sales_data
SET New_Sales_Date = STR_TO_DATE(Sales_Date, '%d/%m/%Y');
ALTER TABLE Sales_data
DROP COLUMN Sales_Date;
ALTER TABLE Sales_data
CHANGE COLUMN New_Sales_Date Sales_Date DATE;

ALTER TABLE Sales_Date
MODIFY COLUMN Product_Cost DECIMAL(15, 2);
```

- Checking;

Field	Type	Null	Key	Default	Extra
Product_ID	int	YES		NULL	
Inventory_Quantity	int	YES		NULL	
Product_Cost	double	YES		NULL	
Sales_Date	date	YES		NULL	

- Identify missing values using `IS NULL` or `COALESCE` functions.

```
-- external_factor
SELECT
   SUM(CASE WHEN Sales_Date IS NULL THEN 1 ELSE 0 END) AS missing_sales_date,
   SUM(CASE WHEN GDP IS NULL THEN 1 ELSE 0 END) AS missing_gdp,
   SUM(CASE WHEN Inflation_Rate IS NULL THEN 1 ELSE 0 END) AS missing_inflation_rate,
   SUM(CASE WHEN Seasonal_Factor IS NULL THEN 1 ELSE 0 END) AS missing_seasonal_factor
FROM external_factors;
-- Product_data
SELECT
   SUM(CASE WHEN Product_ID IS NULL THEN 1 ELSE 0 END) AS missing_product_id,
   SUM(CASE WHEN Product_Category IS NULL THEN 1 ELSE 0 END) AS missing_product_category,
   SUM(CASE WHEN Promotions IS NULL THEN 1 ELSE 0 END) AS missing_promotions
FROM product_data;
-- sales_data
SELECT
   SUM(CASE WHEN Product_ID IS NULL THEN 1 ELSE 0 END) AS missing_product_id,
   SUM(CASE WHEN Sales_Date IS NULL THEN 1 ELSE 0 END) AS missing_sales_date,
   SUM(CASE WHEN Inventory_Quantity IS NULL THEN 1 ELSE 0 END) AS missing_inventory_quantity,
   SUM(CASE WHEN Product_Cost IS NULL THEN 1 ELSE 0 END) AS missing_product_cost
FROM sales_data;
```

- There is no missing data

- Check for duplicates using `GROUP BY` and `HAVING` clauses and remove them if necessary.

```
-- external_factor
SELECT Sales_Date, COUNT(*) as count
FROM external_factors
```

```
GROUP BY Sales_Date
HAVING count > 1;
-- product_data
SELECT Product_ID, Product_Category, COUNT(*) as count
FROM product_data
GROUP BY Product_ID
HAVING count > 1;
-- sales_data
SELECT Product_ID, Sales_Date, COUNT(*) as count
FROM sales_data
GROUP BY Product_ID, Sales_Date
HAVING count > 1;
```

- External_Factors: There are duplicates based on "Sales_Date". Multiple rows have the same sales date, which might indicate redundant or incorrect entries.
- Product_Data: There are duplicates based on "Product_ID". Multiple rows share the same product ID.
- Sales_Data: There are no duplicates based on the combination of "Product_ID" and "Sales_Date".

- Let's deal with them;
- We'll remove the duplicates from the External_Factors and Product_data datasets. We'll keep only one record for each duplicated entry.
- Since there are no duplicates in the Sales Data dataset, no action is required for it.

```
-- external factor
DELETE e1 FROM external_factors e1
INNER JOIN (
   SELECT Sales_Date,
        ROW_NUMBER() OVER (PARTITION BY Sales_Date ORDER BY Sales_Date) as rn
   FROM external_factors
) e2 ON e1.Sales_Date = e2.Sales_Date
WHERE e2.rn > 1;
SELECT COUNT(*) FROM external_factors;

-- product data
DELETE p1 FROM product_data p1
INNER JOIN (
   SELECT Product_ID,
        ROW_NUMBER() OVER (PARTITION BY Product_ID ORDER BY Product_ID) as rn
   FROM product_data
) p2 ON p1.Product_ID = p2.Product_ID
WHERE p2.rn > 1;
SELECT COUNT(*) FROM product_data;
```

- After removing duplicates:
    - External_Factors: There are now 667 unique records based on the "Sales Date".
    - Product_data: There are now 1,258 unique records based on "Product ID"

3. Data Integration:

- Combine relevant datasets using SQL joins (`INNER JOIN`, `LEFT JOIN`, etc.).
- Integrate `External_Factors`, `Product_data`, and `Sales data` to create a unified dataset for analysis by creating a view of the combined dataset.

```
--  sales_data and product_data first
CREATE VIEW sales_product_data AS
SELECT
   s.Product_ID,
   s.Sales_Date,
   s.Inventory_Quantity,
   s.Product_Cost,
   p.Product_Category,
   p.Promotions
FROM sales_data s
JOIN product_data p ON s.Product_ID = p.Product_ID;

-- sale_product_data and external_Factors
CREATE VIEW Inventory_data AS
SELECT
```

```
    sp.Product_ID,
    sp.Sales_Date,
    sp.Inventory_Quantity,
    sp.Product_Cost,
    sp.Product_Category,
    sp.Promotions,
    e.GDP,
    e.Inflation_Rate,
    e.Seasonal_Factor
FROM sales_product_data sp
LEFT JOIN external_factors e ON sp.Sales_Date = e.Sales_Date;
```

- This dataset contains:
    - Sales_data (Product ID, Sales Date, Inventory Quantity, Product Cost)
    - Product_data(Product Category, Promotions)
    - External_factors (GDP, Inflation Rate, Seasonal Factor) for each sales date.
- With this integrated dataset, we can now proceed with further analysis as outlined in our project plan.

4. Descriptive Analytics:

- Calculate basic statistics like average sales, median stock levels, and product performance metrics.
- Identify top-selling and least-selling products.
- Calculate the frequency of stockouts for high-demand products.
- Are there any seasonality patterns in the sales data that could inform stock levels?

- Basic Statistics:
    - Average sales (calculated as the product of "Inventory Quantity" and "Product Cost").
    - Median stock levels (i.e., "Inventory Quantity").
    - Product performance metrics (total sales per product).
- Identify Top-Selling and Least-Selling Products:
    - Based on the total sales for each product.
    - Frequency of Stockouts for High-Demand Products:
- A stockout will be defined as an "Inventory Quantity" of zero.
    - We'll identify high-demand products as those with the highest average sales.
    - Calculate the frequency of stockouts for these high-demand products

- Steps;

```
-- Average sales (calculated as the product of "Inventory Quantity" and "Product Cost").
SELECT Product_ID,
     ROUND(AVG(Inventory_Quantity * Product_Cost))as avg_sales
FROM Inventory_data
GROUP BY Product_ID;

-- Median stock levels (i.e., "Inventory Quantity").
SELECT Product_ID, AVG(Inventory_Quantity) as median_stock
FROM (
   SELECT Product_ID,
        Inventory_Quantity,
        ROW_NUMBER() OVER(PARTITION BY Product_ID ORDER BY Inventory_Quantity) as row_num_asc,
        ROW_NUMBER() OVER(PARTITION BY Product_ID ORDER BY Inventory_Quantity DESC) as
row_num_desc
   FROM Inventory_data
) AS subquery
WHERE row_num_asc IN (row_num_desc, row_num_desc - 1, row_num_desc + 1)
GROUP BY Product_ID;

-- Product performance metrics (total sales per product).
SELECT Product_ID,
     SUM(Inventory_Quantity * Product_Cost) as total_sales
FROM inventory_data
GROUP BY Product_ID
ORDER BY total_sales DESC
```

- Observation:
    - For instance, the product with Product ID 2010 has an average sale value of approximately $19,669.

- The median stock level for the product with Product ID 1002 is 12 units.
- Product Performance Metrics:The product with Product ID 2010 has the highest total sales of approximately $19,669, making it the top-selling product and Product ID '8821' has the lowest total sales approximately $ 17.

- A stockout will be defined as an "Inventory Quantity" of zero.

```
-- Identify high-demand products based on average sales
-- We'll consider the top 5% of products in terms of average sales as high-demand products
WITH HighDemandProducts AS (
   SELECT Product_ID, AVG(Inventory_Quantity) as avg_sales
   FROM Inventory_data
   GROUP BY Product_ID
   HAVING avg_sales > (
      SELECT AVG(Inventory_Quantity) * 0.95 FROM Sales_data  -- This approximates the top 5% threshold
   )
)

-- Calculate stockout frequency for high-demand products
SELECT s.Product_ID,
      COUNT(*) as stockout_frequency
FROM Inventory_data s
WHERE s.Product_ID IN (SELECT Product_ID FROM HighDemandProducts)
AND s.Inventory_Quantity = 0
GROUP BY s.Product_ID;
```

- Observation:
  - None of the identified high-demand products have experienced stockouts (where "Inventory Quantity" is 0).
  - This is a positive sign for inventory management since it indicates that high-demand products have been adequately stocked and have not faced stockout situations.

5. Influence of External Factors:

- How do external factors influence sales trends?
- Which specific external factors consistently lead to increased or decreased sales for certain products?
- GDP (Gross Domestic Product):
  - Represents the overall economic health and growth of a country.
  - A higher GDP usually indicates more consumer spending, leading to increased sales for businesses.
  - Conversely, a lower GDP can signify an economic downturn, potentially leading to decreased sales.

- Inflation Rate:
  - Represents the rate at which the general level of prices for goods and services is rising, and purchasing power is falling.
  - A high inflation rate might deter consumers from purchasing non-essential items, leading to reduced sales.
  - On the other hand, if consumers anticipate future price rises due to inflation, they might make purchases earlier, boosting sales temporarily.

- **Steps:**

```
-- GDP-  the overall economic health and growth of a country. it's
SELECT Product_ID,
   AVG(CASE WHEN `GDP` > 0 THEN Inventory_Quantity ELSE NULL END) AS avg_sales_positive_gdp,
   AVG(CASE WHEN `GDP` <= 0 THEN Inventory_Quantity ELSE NULL END) AS
avg_sales_non_positive_gdp
FROM Inventory_table
GROUP BY Product_ID
HAVING avg_sales_positive_gdp IS NOT NULL;


-- Inflation influence on the Sales
SELECT Product_ID,
     AVG(CASE WHEN Inflation_Rate > 0 THEN Inventory_Quantity ELSE NULL END) AS
avg_sales_positive_inflation,
     AVG(CASE WHEN Inflation_Rate <= 0 THEN Inventory_Quantity ELSE NULL END) AS
avg_sales_non_positive_inflation
FROM Inventory_table
GROUP BY Product_ID
```

HAVING avg_sales_positive_inflation IS NOT NULL;

Observation:

- For GDP Influence:
  - For Instance, Product ID 1387, we observe an average sales quantity of 25 during positive GDP periods. There isn't any data available for non-positive GDP periods (which includes neutral or negative GDP) for this product.
  - Similar observations can be made for other products, indicating that the data might only contain sales quantities for positive GDP periods.

- For Inflation Rate Influence:
  - For instance, Product ID 1387, the average sales quantity is 25 during positive inflation rate periods. Again, there's no data for non-positive inflation rate periods (which includes neutral or negative inflation) for this product.

6. Inventory Optimization: the main part of the analysis

- Determine the optimal reorder point for each product based on historical sales data and external factors.
- Use SQL window functions to analyze sales trends.
- Calculate safety stock levels based on sales variability and lead time.

- Steps;
- Inventory optimization aims to ensure that the right amount of stock is maintained to meet customer demand while minimizing holding costs and potential stockouts. For this, we need to calculate:

  - Reorder Point: The inventory level at which a new order should be placed.
  - Reorder Point=Lead Time Demand+Safety Stock

  - Where:
    - Lead Time Demand is the expected sales during the lead time.
    - Lead Time Demand= Rolling Average Sales × Lead Time
    - Safety Stock: An extra buffer stock to account for variability in demand and supply.
    - Safety Stock=Z× $\sqrt{Lead\ Time}$ ×Standard Deviation of Demand
    - Where:
    - Z is the number of standard deviations for the desired service level (we can assume a common service level, like 95%, which corresponds to Z=1.645 for a normal distribution).
    - Lead Time is the time taken between placing an order and receiving it.
    - Standard Deviation of Demand is the variability in demand over a specified period.

  - For simplicity, let's make the following assumptions:
    - A constant lead time of 7 days for all products.
    - We aim for a 95% service level.
  - First, we'll use SQL some window functions to analyze sales trends for each product. We'll compute a rolling average of sales (average sales over the past 7 days) and the standard deviation of sales over the same period.

```
WITH InventoryCalculations AS (
    SELECT Product_ID,
        AVG(rolling_avg_sales) as avg_rolling_sales,
        AVG(rolling_variance) as avg_rolling_variance
    FROM (
        SELECT Product_ID,
            AVG(daily_sales) OVER (PARTITION BY Product_ID ORDER BY Sales_Date ROWS BETWEEN 6
PRECEDING AND CURRENT ROW) as rolling_avg_sales,
            AVG(squared_diff) OVER (PARTITION BY Product_ID ORDER BY Sales_Date ROWS BETWEEN 6
PRECEDING AND CURRENT ROW) as rolling_variance
        FROM (
            SELECT Product_ID,
                Sales_Date,
                Inventory_Quantity * Product_Cost as daily_sales,
                (Inventory_Quantity * Product_Cost - AVG(Inventory_Quantity * Product_Cost) OVER (PARTITION BY
Product_ID ORDER BY Sales_Date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)) *
(Inventory_Quantity * Product_Cost - AVG(Inventory_Quantity * Product_Cost) OVER (PARTITION BY Product_ID
ORDER BY Sales_Date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)) as squared_diff
```

```
            FROM Inventory_data
        ) subquery -- Add an alias for the subquery
    ) subquery2 -- Add an alias for the subquery
    GROUP BY Product_ID
)
SELECT Product_ID,
    avg_rolling_sales * 7 as lead_time_demand,
    1.645 * (avg_rolling_variance * 7) as safety_stock,
    (avg_rolling_sales * 7) + (1.645 * (avg_rolling_variance * 7)) as reorder_point
FROM InventoryCalculations;
```

Observation;

- Lead Time Demand: Represents the expected sales during the lead time (which we assumed to be 7 days for all products). For instance, the product with ID 1002 has a lead time demand of approximately 13,476.96 units.

- Safety Stock: This is an extra buffer stock to account for variability in demand and supply. Given our assumptions, the safety stock for many products is zero, likely due to zero variance in their sales.. In a more nuanced real-world scenario, the safety stock would vary based on sales variability.

- Reorder Point: This is the inventory level at which a new order should be placed. It's the sum of the Lead Time Demand and the Safety Stock. For the product with ID 1002, the reorder point is approximately 13,476.96 units.

Automate the Calculate of the Reorder Point to ensure optimal inventory levels

- Create the inventory_optimization table.
- Create the RecalculateReorderPoint stored procedure.
- Make inventory_data a permanent table.
- Create the trigger that will call the stored procedure after inserts on the inventory_data table.

```
-- Step 1: Create the Inventory Optimization Table
CREATE TABLE inventory_optimization (
    Product_ID INT PRIMARY KEY,
    Reorder_Point DOUBLE
);

-- Step 2: Create the Stored Procedure to Recalculate Reorder Point
DELIMITER //
CREATE PROCEDURE RecalculateReorderPoint(productID INT)
BEGIN
    DECLARE avgRollingSales DOUBLE;
    DECLARE avgRollingVariance DOUBLE;
    DECLARE leadTimeDemand DOUBLE;
    DECLARE safetyStock DOUBLE;
    DECLARE reorderPoint DOUBLE;

    -- Calculate average rolling sales and variance for the product
    SELECT AVG(rolling_avg_sales), AVG(rolling_variance)
    INTO avgRollingSales, avgRollingVariance
    FROM (
        SELECT Product_ID,
            AVG(daily_sales) OVER (PARTITION BY Product_ID ORDER BY Sales_Date ROWS BETWEEN 6
PRECEDING AND CURRENT ROW) as rolling_avg_sales,
            AVG(squared_diff) OVER (PARTITION BY Product_ID ORDER BY Sales_Date ROWS BETWEEN 6
PRECEDING AND CURRENT ROW) as rolling_variance
        FROM (
            SELECT Product_ID,
                Sales_Date,
                Inventory_Quantity * Product_Cost as daily_sales,
                (Inventory_Quantity * Product_Cost -
                 AVG(Inventory_Quantity * Product_Cost) OVER (PARTITION BY Product_ID ORDER BY Sales_Date
ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)) *
                (Inventory_Quantity * Product_Cost -
                 AVG(Inventory_Quantity * Product_Cost) OVER (PARTITION BY Product_ID ORDER BY Sales_Date
ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)) as squared_diff
            FROM inventory_data
            WHERE Product_ID = productID
        ) AS InnerDerived
    ) AS OuterDerived;
```

```
    SET leadTimeDemand = avgRollingSales * 7;  -- Assuming a lead time of 7 days
    SET safetyStock = 1.645 * SQRT(avgRollingVariance * 7);  -- Using Z value for 95% service level
    SET reorderPoint = leadTimeDemand + safetyStock;

    -- Insert or update the reorder point in the inventory_optimization table
    INSERT INTO inventory_optimization (Product_ID, Reorder_Point)
    VALUES (productID, reorderPoint)
    ON DUPLICATE KEY UPDATE Reorder_Point = reorderPoint;
END //
DELIMITER ;

-- Step 3: make inventory_data a permanent table
CREATE TABLE Inventory_table AS SELECT * FROM Inventory_data;

-- Step 4: Create the Trigger
DELIMITER //
CREATE TRIGGER AfterInsertUnifiedTable
AFTER INSERT ON Inventory_table
FOR EACH ROW
BEGIN
    CALL RecalculateReorderPoint(NEW.Product_ID);
END //
DELIMITER ;
```

7. Analyse Overstock and Understock Situations:

- Identify Overstocked Products:
    - Products with inventory levels consistently higher than their sales trends (rolling average sales) can be considered overstocked.

- Identify Understocked Products:
    - Products that frequently have an inventory quantity of zero, despite having sales, can be considered understocked.

- Calculate Potential Lost Sales:
    - For understocked products, we'll quantify the potential lost sales due to stockouts. The potential lost sales can be calculated as the product of the sales trend (rolling average sales) and the number of days the product was out of stock.
- Steps:

```
WITH RollingSales AS (
    SELECT Product_ID,
        Sales_Date,
        AVG(Inventory_Quantity * Product_Cost) OVER (PARTITION BY Product_ID ORDER BY Sales_Date ROWS
BETWEEN 6 PRECEDING AND CURRENT ROW) as rolling_avg_sales
    FROM inventory_table
),

-- Calculate the number of days a product was out of stock
StockoutDays AS (
    SELECT Product_ID,
        COUNT(*) as stockout_days
    FROM inventory_table
    WHERE Inventory_Quantity  = 0
    GROUP BY Product_ID
)

-- Join the above CTEs with the main table to get the results
SELECT f.Product_ID,
     AVG(f.Inventory_Quantity  * f.Product_Cost) as avg_inventory_value,
     AVG(rs.rolling_avg_sales) as avg_rolling_sales,
     COALESCE(sd.stockout_days, 0) as stockout_days
FROM inventory_table f
JOIN RollingSales rs ON f.Product_ID = rs.Product_ID AND f.Sales_Date = rs.Sales_Date
LEFT JOIN StockoutDays sd ON f.Product_ID = sd.Product_ID
GROUP BY f.Product_ID, sd.stockout_days;
```

- Observation:
- Overstocked Products:
    - We've identified several products that are frequently overstocked. For instance:
    - Product with ID 1387 has an average inventory value of approximately $2,145.75, while its rolling average sales is also $2,145.75.
    - It seems all products have its average inventory value equal to rolling average. It could also indicate that the inventory for that product hasn't changed over the period. If a product's inventory value remains constant and matches its sales value, but the actual sales quantity is low or zero, it could suggest the product is not being sold but is constantly stocked. This situation could result in overstocking.

- Understocked Products:
    - It seems there are no products in the dataset that frequently have an inventory quantity of zero despite having sales. This means there are no identified stockouts based on the given data.

.

8. Monitor and Adjust:(stored procedures)

Establish SQL queries to routinely monitor inventory levels, sales trends, and stockout frequencies.

```
-- Monitor inventory levels
DELIMITER //
CREATE PROCEDURE MonitorInventoryLevels()
BEGIN
   SELECT Product_ID, AVG(Inventory_Quantity) as AvgInventory
   FROM Inventory_table
   GROUP BY Product_ID
   ORDER BY AvgInventory DESC;
END//
DELIMITER ;

-- Monitor Sales Trends
DELIMITER //
CREATE PROCEDURE MonitorSalesTrends()
BEGIN
   SELECT Product_ID, Sales_Date,
        AVG(Inventory_Quantity * Product_Cost) OVER (PARTITION BY Product_ID ORDER BY Sales_Date ROWS
BETWEEN 6 PRECEDING AND CURRENT ROW) as RollingAvgSales
   FROM inventory_table
   ORDER BY Product_ID, Sales_Date;
END//
DELIMITER ;

-- Monitor Stockout frequencies
DELIMITER //
CREATE PROCEDURE MonitorStockouts()
BEGIN
   SELECT Product_ID, COUNT(*) as StockoutDays
   FROM full_integrated_data
   WHERE Inventory_Quantity = 0
   GROUP BY Product_ID
   ORDER BY StockoutDays DESC;
END//
DELIMITER ;
```

9. Feedback Loop:

- Establish a mechanism for feedback from stakeholders.
- Refine and improve the inventory optimization system based on feedback.

- Feedback Loop Establishment:

    - Feedback Portal: Develop an online platform for stakeholders to easily submit feedback on inventory performance and challenges.
    - Review Meetings: Organize periodic sessions to discuss inventory system performance and gather direct insights.
    - System Monitoring: Use established SQL procedures to track system metrics, with deviations from expectations flagged for review.

- Refinement Based on Feedback:
    - Feedback Analysis: Regularly compile and scrutinize feedback to identify recurring themes or pressing issues.
    - Action Implementation: Prioritize and act on the feedback to adjust reorder points, safety stock levels, or overall processes.
    - Change Communication: Inform stakeholders about changes, underscoring the value of their feedback and ensuring transparency.

General Insights:

1. Inventory Discrepancies: The initial stages of the analysis revealed significant discrepancies in inventory levels, with instances of both overstocking and understocking. These inconsistencies were contributing to capital inefficiencies and customer dissatisfaction.

2. Sales Trends and External Influences: The analysis indicated that sales trends were notably influenced by various external factors. Recognizing these patterns provides an opportunity to forecast demand more accurately.

3. Suboptimal Inventory Levels: Through the inventory optimization analysis, it was evident that the existing inventory levels were not optimized for current sales trends. Products was identified that had either close excess inventory.

Recommendations:

1. Implement Dynamic Inventory Management: The company should transition from a static to a dynamic inventory management system, adjusting inventory levels based on real-time sales trends, seasonality, and external factors.

2. Optimize Reorder Points and Safety Stocks: Utilize the reorder points and safety stocks calculated during the analysis to minimize stockouts and reduce excess inventory. Regularly review these metrics to ensure they align with current market conditions.

3. Enhance Pricing Strategies: Conduct a thorough review of product pricing strategies, especially for products identified as unprofitable. Consider factors such as competitor pricing, market demand, and product acquisition costs.

4. Reduce Overstock: Identify products that are consistently overstocked and take steps to reduce their inventory levels. This could include promotional sales, discounts, or even discontinuing products with low sales performance.

5. Establish a Feedback Loop: Develop a systematic approach to collect and analyze feedback from various stakeholders. Use this feedback for continuous improvement and alignment with business objectives.

6. Regular Monitoring and Adjustments: Adopt a proactive approach to inventory management by regularly monitoring key metrics and making necessary adjustments to inventory levels, order quantities, and safety stocks.

By addressing these areas, TechElectro Inc. can significantly improve its inventory management system, leading to increased operational efficiency, reduced costs, and enhanced customer satisfaction.