Optimizing Health care Inventory for Enhanced Marketing Strategies

Important Project Files:

1. DataAnalysis.ipynb(Analysis jupyter document)
2. Inventory.sqlite3(Database)
3. SourceData(inv_item.txt,inv_item_price.txt,inv_item_stock.txt,inv_stockroom.txt)
4. Inventory Analysis Report pdf

Business Problem

The marketing team at a health care platform needs actionable insights from inventory data to refine their promotional strategies. By understanding inventory dynamics, they can better target their promotions, ensuring that items in stock are highlighted and those running low are restocked promptly.

Business Objective

To leverage inventory data analysis to enhance the effectiveness of marketing strategies, ensuring optimal stock levels and targeted promotions that drive sales and improve customer satisfaction.

Overview

In this scenario, I was provided with a data set containing detailed information about the inventory on a health care platform. My task was to analyze this data set and extract insights that would help the marketing team improve their promotional strategies. The analysis focused on the following key points:

- Average Price Analysis:
  Calculate the average price of items with physical inventory.
  Calculate the average price of items without physical inventory.
- Storeroom Inventory Analysis:
  Identify which storeroom has the most inventory.
- Price Modification Timing:
  Determine the time in minutes since item prices were last modified.
- Stock Status:
  List items that are running out of stock.
  List items whose quantity is zero and how long (in minutes) they have been in this state.

By addressing these points, the marketing team can gain valuable insights into inventory trends and make informed decisions to optimize their promotional strategies. This will help ensure that high-demand items are adequately stocked and promoted, while also identifying opportunities to clear out items that are not moving.

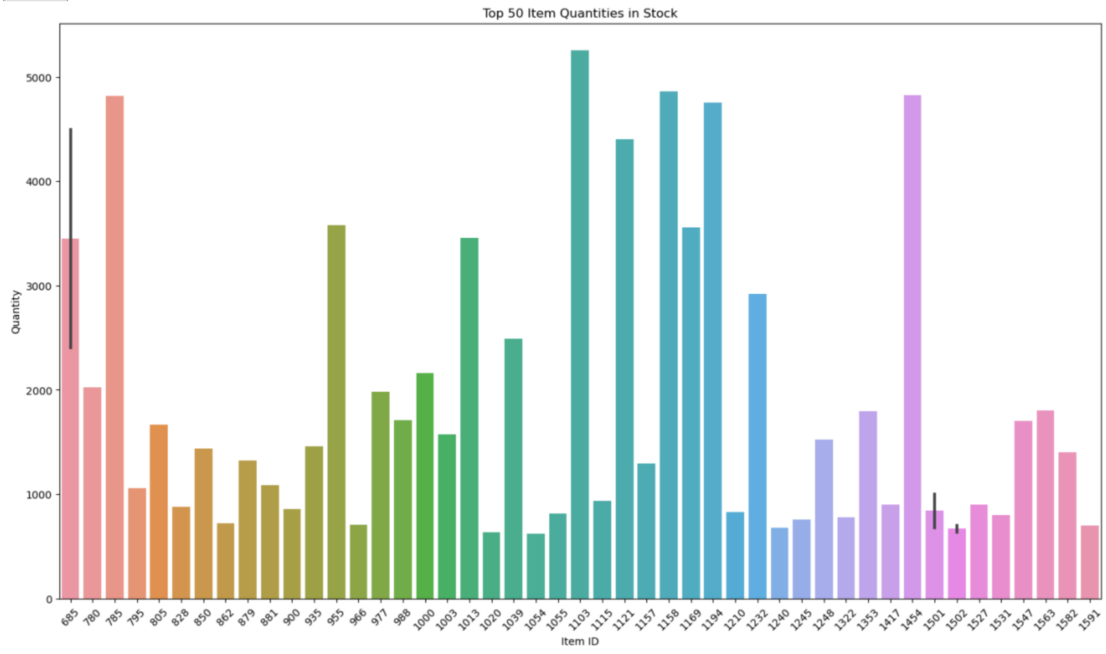
Data Understanding

The data source for our analysis is primarily the Inventory.sqlite3 database, which encompasses detailed inventory information for a health care platform.

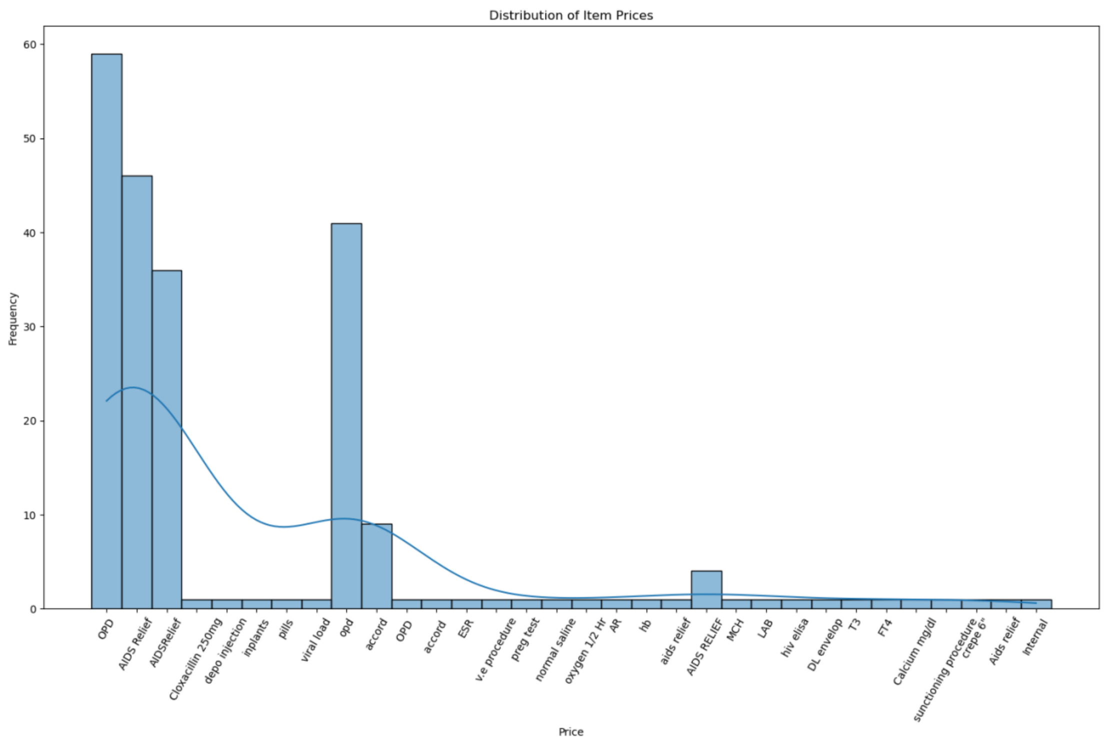
Inventory.sqlite3 (Main data source)
- Source:(inv_item.txt,inv_item_price.txt,inv_item_stock.txt,inv_stockroom.txt)
- Contents: This database includes comprehensive details about inventory items, their prices, stock levels, and storeroom information. It covers various aspects such as item names, department IDs, default prices, creators, creation and modification dates, and unique identifiers (UUIDs).

Our analysis will focus on understanding the trends, patterns, and factors influencing inventory dynamics. This will enable us to provide actionable insights that can enhance marketing strategies, ensuring optimal stock levels and targeted promotions that drive sales and improve customer satisfaction.

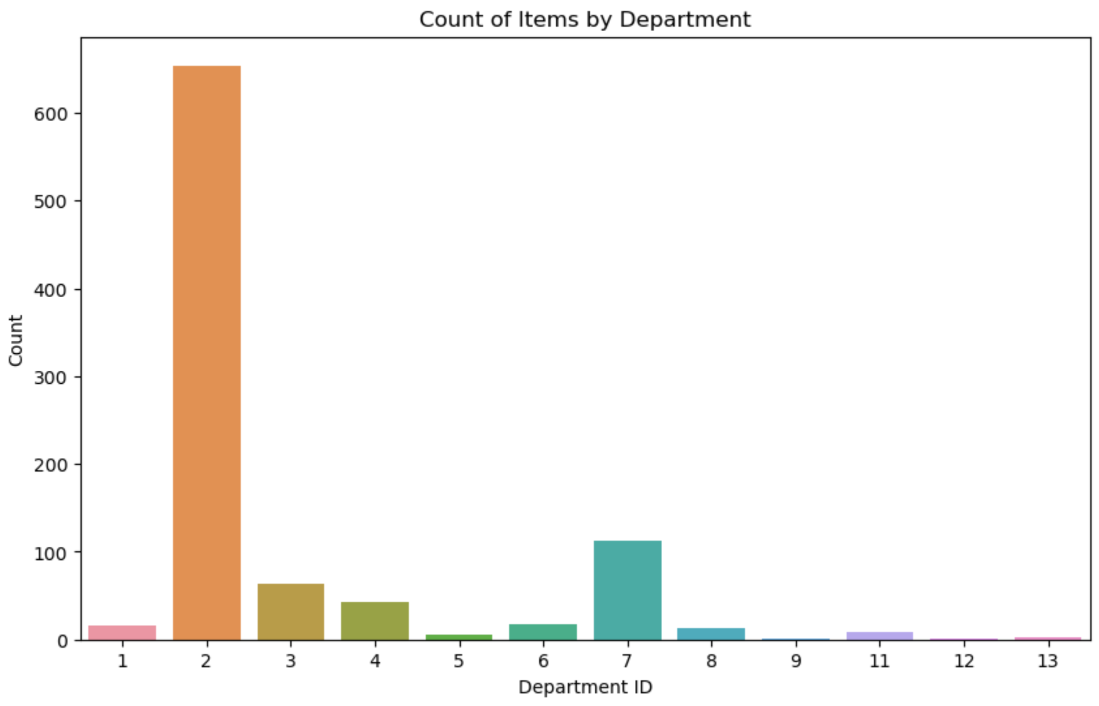
Data Visualization



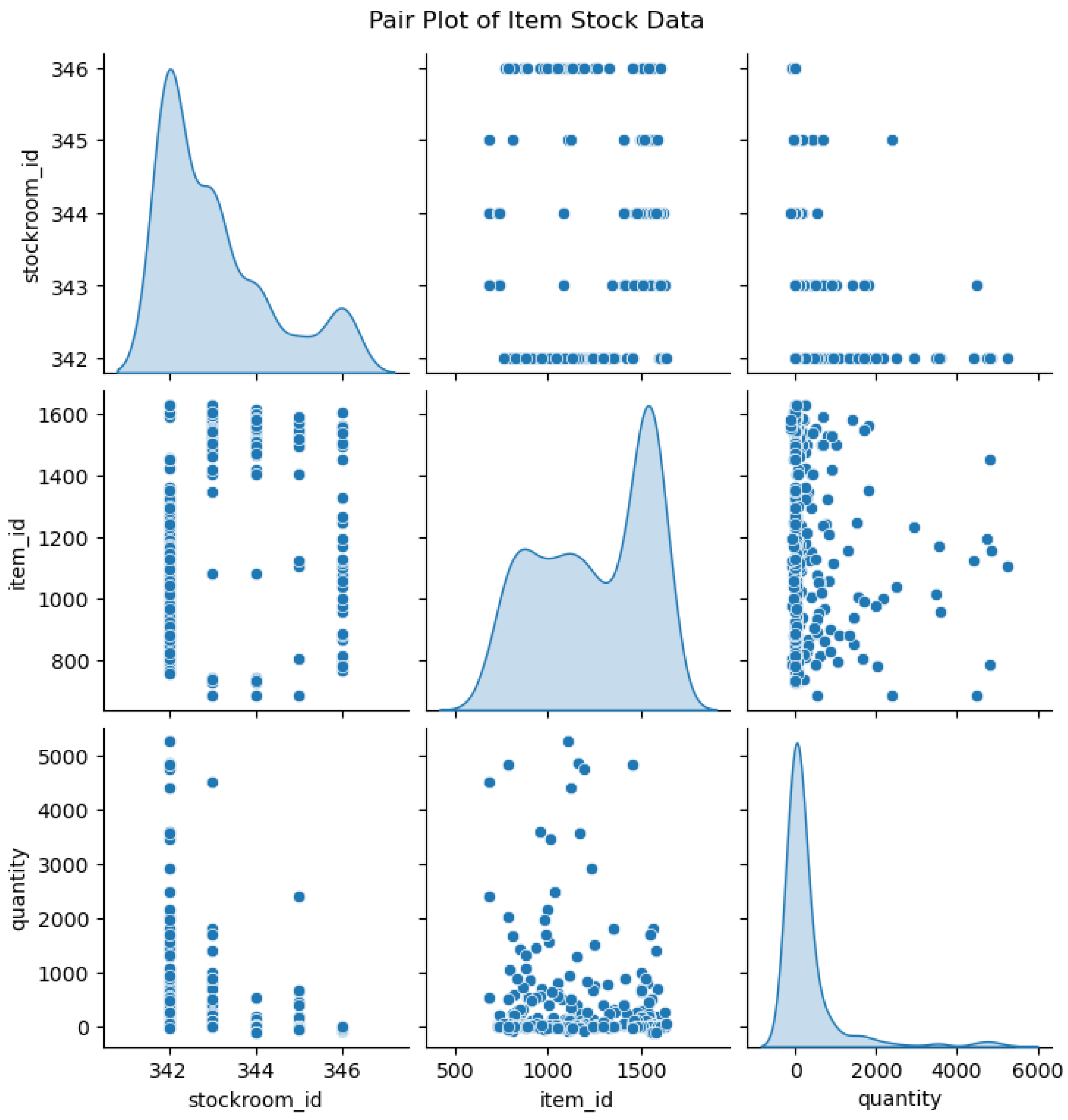
The plot visually highlights the top 50 items with the highest quantities in stock, making it easy to identify which items are most abundant.



Price Distribution: The histogram visually represents the distribution of item prices, showing how frequently different price ranges occur.The height of the bars indicates how many items fall within each price range, with that the KDE line provides a smoothed view of the distribution, helping to identify trends and patterns in the data, such as peaks and valleys.



The count plot visually represents the distribution of items across different departments, showing how many items belong to each department



Each cell in the matrix represents a scatter plot of two variables from the inv_item_stock Data Frame.The scatter plots show the relationship between pairs of variables, allowing for visual inspection of correlation.From this there are strong linear relationships between certain pairs of variables, indicating high correlation and Some variables have skewed distributions, as shown by the KDE plots on the diagonal.Finally a few outliers are present in the scatter plots, which may need further investigation.

Query Notebook

Following up on the data understanding sqlite3 was used in querying the data. Here is a description of functions used:

1. pd.read_csv():

   Description: Reads a comma-separated values (CSV) file into a Data Frame.

Usage:
inv_item = pd.read_csv('inv_item_exported.csv')
inv_item_price = pd.read_csv('inv_item_price_exported.csv')
inv_item_stock = pd.read_csv('inv_item_stock_exported.csv')
inv_stockroom = pd.read_csv('inv_stockroom_exported.csv')

Purpose: To load data from CSV files into pandas DataFrames for further processing and analysis.

2. sqlite3.connect():

   Description: Establishes a connection to an SQLite database. If the database does not exist, it will be created.
   Usage:
   conn = sqlite3.connect('inventory.db')

   Purpose: To create or connect to an SQLite database named inventory.db where the DataFrames will be stored as tables.
3. Data Frame.to_sql():

   Description: Writes records stored in a Data Frame to a SQL database.
   Usage:

   inv_item.to_sql('inv_item', conn, if_exists='replace', index=False)
   inv_item_price.to_sql('inv_item_price', conn, if_exists='replace', index=False)
   inv_item_stock.to_sql('inv_item_stock', conn, if_exists='replace', index=False)
   inv_stockroom.to_sql('inv_stockroom', conn, if_exists='replace', index=False)

   Purpose: To save the DataFrames as tables in the SQLite database. The if_exists='replace' parameter ensures that if the table already exists, it will be replaced. The index=False parameter prevents the Data Frame index from being written as a column in the SQL table.

The notebook uses the pandas library to read data from CSV files into DataFrames and the sqlite3 library to store these DataFrames as tables in an SQLite database. This process facilitates data persistence and enables efficient querying and manipulation of the data using SQL.

```
# 1. Average price of items with physical inventory and those without
query1 = """
SELECT
    CASE WHEN has_physical_inventory = 1 THEN 'With Inventory' ELSE
'Without Inventory' END AS inventory_status,
    AVG(buying_price) AS average_price
```

```
FROM inv_item
WHERE buying_price IS NOT NULL
GROUP BY inventory_status;

"""
```

4. This SQL query calculates the average buying price of items based on their physical inventory status. Here's a breakdown of the query:

    SELECT Clause:

    CASE WHEN has_physical_inventory = 1 THEN 'With Inventory' ELSE 'Without Inventory' END AS inventory_status: This creates a new column inventory_status that categorizes items into 'With Inventory' or 'Without Inventory' based on the value of has_physical_inventory.
    AVG(buying_price) AS average_price: This calculates the average buying price of items within each inventory status category.

    FROM Clause:

    FROM inv_item: This specifies the table inv_item from which to retrieve the data.

    WHERE Clause:

    WHERE buying_price IS NOT NULL: This filters out any rows where the buying_price is NULL, ensuring that only valid prices are considered in the average calculation.

    GROUP BY Clause:

    GROUP BY inventory_status: This groups the results by the inventory_status column, so the average price is calculated separately for items with and without physical inventory.
    The function used here is AVG(), which is an aggregate function in SQL. It is used to calculate the average value of a numeric column. In this context, it helps to determine the average buying price of items based on their inventory status. This can be useful for understanding pricing trends and inventory management.

```
# 2. Show which storeroom has the most inventory
query2 = """
SELECT stockroom_id, SUM(quantity) AS total_quantity
FROM inv_item_stock
GROUP BY stockroom_id
ORDER BY total_quantity DESC
LIMIT 1
"""
```

5. This SQL query identifies the storeroom with the highest total inventory. Breaking it down:

   SELECT Clause:

   stockroom_id: This selects the identifier for the storeroom.
   SUM(quantity) AS total_quantity: This calculates the total quantity of items in each storeroom by summing up the quantity column.

   FROM Clause:

   FROM inv_item_stock: This specifies the table inv_item_stock from which to retrieve the data.

   GROUP BY Clause:

   GROUP BY stockroom_id: This groups the results by stockroom_id, so the total quantity is calculated for each storeroom separately.

   ORDER BY Clause:

   ORDER BY total_quantity DESC: This orders the results by total_quantity in descending order, so the storeroom with the highest total quantity appears first.

   LIMIT Clause:

   LIMIT 1: This limits the result to only the top entry, effectively selecting the storeroom with the most inventory.
   The function used here is SUM(), which is an aggregate function in SQL. It is used to calculate the total sum of a numeric column. In this context, it helps to determine the total quantity of items in each storeroom. This can be useful for inventory management and identifying which storeroom holds the most stock.

```
# 3. Time in minutes since item prices were last modified
query3 = """
SELECT item_id, price,
       (strftime('%s', 'now') - strftime('%s', date_changed)) / 60 AS
minutes_since_modified
FROM inv_item_price;
"""
```

6. This SQL query calculates the time in minutes since the prices of items were last modified. Here's a breakdown of the query:

   SELECT Clause:

   item_id: This selects the identifier for the item.

price: This selects the price of the item.

(strftime('%s', 'now') - strftime('%s', date_changed)) / 60 AS minutes_since_modified: This calculates the time in minutes since the price was last modified. The strftime('%s', 'now') function returns the current time in seconds since the Unix epoch, and strftime('%s', date_changed) returns the time in seconds since the Unix epoch for the date_changed value. Subtracting these two values gives the difference in seconds, which is then divided by 60 to convert it to minutes.

FROM Clause:

FROM inv_item_price: This specifies the table inv_item_price from which to retrieve the data.

The function used here is strftime(), which is a date and time function in SQL. It is used to format date and time values. In this context, it helps to calculate the difference in time between the current moment and the last modification date of the item prices. This can be useful for monitoring how frequently item prices are updated.

```
# 4. List items that are running out of stock
query4 = """
SELECT * FROM inv_item_stock
WHERE quantity < 5
"""
```

7. This SQL query lists items that are running out of stock by selecting all columns from the inv_item_stock table where the quantity is less than 5. Here's a breakdown of the query:

SELECT Clause:

SELECT *: This selects all columns from the table.

FROM Clause:

FROM inv_item_stock: This specifies the table inv_item_stock from which to retrieve the data.

WHERE Clause:

WHERE quantity < 5: This filters the results to include only rows where the quantity is less than 5, indicating that these items are running low on stock.

This query is useful for inventory management, allowing you to quickly identify items that need to be restocked.

```
# 5. List items whose quantity is Zero (0) and how long (In minutes)
they have been in this state
query5 = """
SELECT
    i.name AS item_name,
    s.quantity AS current_quantity,
    ROUND((julianday('now') - julianday(i.date_changed)) * 24 * 60) AS
minutes_since_changed
FROM inv_item AS i
INNER JOIN inv_item_stock AS s ON i.item_id = s.item_id
WHERE s.quantity = 0;

"""
```

8. This SQL query lists items whose quantity is zero and calculates how long (in minutes) they have been in this state.

SELECT Clause:

i.name AS item_name: This selects the name of the item from the inv_item table and renames it to item_name.

s.quantity AS current_quantity: This selects the current quantity of the item from the inv_item_stock table and renames it to current_quantity.

ROUND((julianday('now') - julianday(i.date_changed)) * 24 * 60) AS minutes_since_changed: This calculates the time in minutes since the item's quantity was last changed. The julianday('now') function returns the current date and time in Julian day format, and julianday(i.date_changed) returns the date and time when the item's quantity was last changed. The difference between these two values gives the time in days, which is then multiplied by 24 (hours per day) and 60 (minutes per hour) to convert it to minutes. The ROUND() function rounds the result to the nearest whole number.

FROM Clause:

FROM inv_item AS i: This specifies the inv_item table with an alias i.

INNER JOIN Clause:

INNER JOIN inv_item_stock AS s ON i.item_id = s.item_id: This performs an inner join between the inv_item table (aliased as i) and the inv_item_stock table (aliased as s) on the item_id column, ensuring that only rows with matching item_id values in both tables are included in the result.

WHERE Clause:

WHERE s.quantity = 0:

This filters the results to include only rows where the quantity in the inv_item_stock table is zero, indicating that these items are out of stock.This query is useful for inventory management, allowing you to identify items that are out of stock and how long they have been in this state.

In Conclusion,to optimize sales and enhance inventory management, the marketing team should implement a multifaceted strategy. Firstly, promotional efforts should be concentrated on the top 10 high-demand items. By highlighting these products in marketing campaigns, the team can maximize sales and ensure that popular items are readily available to meet customer demand.

In parallel, it is essential to address low-demand items. Offering discounts or clearance sales for the top 10 low-demand items can help clear out inventory that is not moving as quickly. This approach not only frees up valuable storage space but also provides an opportunity to attract price-sensitive customers and increase overall sales volume.

Effective stock management is another critical component. The stockroom with the most inventory should be meticulously managed to ensure it is well-stocked with high-demand items. This proactive approach helps prevent stock outs, ensuring that customers can always find the products they are looking for, thereby enhancing customer satisfaction and loyalty.

Lastly, a thorough analysis of the impact of pricing on sales is crucial. By understanding how different price points affect sales volume, the marketing team can make informed decisions to adjust prices strategically. This data-driven approach allows for the optimization of pricing strategies to maximize revenue, ensuring that the health care platform remains competitive and profitable.

By integrating these strategies, the marketing team can create a balanced approach that not only drives sales but also maintains efficient inventory levels, ultimately contributing to the overall success of the health care platform.