Predictive Inventory Management for Stores using Data Analysis

Hiba Qutbuddin Habib
Hibatallah Belhajali
Grupp H&H

1. Introduction

An inefficient inventory management method can lead to unnecessary costs and have a negative environmental impact, while also causing losses due to stock shortages. By creating more accurate forecasts, inventory can be better balanced. This can be achieved through analyzing data collected from the business, such as product inventory value and sales volumes over time.

The purpose of this project is to analyze a dataset to identify trends over a period of time and forecast sales for the coming three months. The goal is to predict which products are expected to sell the most and the least, which can help the business plan inventory management more effectively and make better budget forecasts. To achieve this, various techniques are used, including HDFS, Spark, Cassandra, and Matplotlib.

2. Dataset

To carry out this project, a large dataset of historical sales over a sufficiently long period was required. Finding a dataset with an adequate size and relevant details proved challenging, as most datasets found were limited in time span and contained only around 1,000 rows, which was insufficient for a meaningful analysis to generate forecasts. The goal was to use a dataset related to a specific market, but such datasets were not readily available either. The dataset deemed suitable for this project was sourced from Kaggle[1], containing 13 months of sales history across multiple countries. This dataset includes more than 500,000 rows and 8 columns. The structure of the dataset is as follows:

```
root
|-- Invoice: string (nullable = true)
|-- StockCode: string (nullable = true)
|-- Description: string (nullable = true)
|-- Quantity: integer (nullable = true)
|-- InvoiceDate: string (nullable = true)
|-- Price: double (nullable = true)
|-- Customer ID: integer (nullable = true)
|-- Country: string (nullable = true)
```

Example of the first 5 rows:

Total rows: 541,910

[1] https://www.kaggle.com/datasets/saurabhbadole/supermarket-data?select=supermarket_data.csv

```
+-------+---------+-----------------------+--------+--------------+-----+-----------+--------------+
|Invoice|StockCode|Description            |Quantity|InvoiceDate   |Price|Customer ID|Country       |
+-------+---------+-----------------------+--------+--------------+-----+-----------+--------------+
|536365 |85123A   |WHITE HANGING HEART T-LIGHT|6      |01/12/10 08:26|2.55 |17850      |United Kingdom|
|536365 |71053    |WHITE METAL LANTERN      |6       |01/12/10 08:26|3.39 |17850      |United Kingdom|
|536365 |84406B   |CREAM CUPID HEART        |8       |01/12/10 08:26|2.75 |17850      |United Kingdom|
|536365 |84029G   |KNITTED UNION FLAG HOT WATER BOTTLE|6|01/12/10 08:26|3.39 |17850      |United Kingdom|
|536365 |84029E   |RED WOOLLY HOTTIE HEAT PACK|6    |01/12/10 08:26|3.39 |17850      |United Kingdom|
+-------+---------+-----------------------+--------+--------------+-----+-----------+--------------+
```

3. Method

The implementation of this project was carried out in a Jupyter Notebook, with Python as the programming language. The main components used were Hadoop, HDFS, Spark, Cassandra, and Matplotlib.

To start the implementation, several installations were required on the local machine, including Spark, HDFS, and Cassandra, among others. One of the learning objectives of the project was to read data from a large storage system. This posed challenges during the implementation, as we initially attempted to use AWS databases and similar solutions, but encountered issues with setting up an account and establishing connections. The alternative solution chosen was to use Spark and Hadoop tools in pseudo-distributed mode, which simulates HDFS. In this mode, Spark interacts with the data as if it were in HDFS, even though the data is stored locally. This approach allows for simulating all the functionalities of a true cluster, such as distributed storage and parallel processing, while still running everything locally.

HDFS (Hadoop Distributed File System) was run as a local instance, where the filesystem mimicked storage across multiple nodes, though all data actually resided on the computer’s own hard drive. Replication and block management were simulated, but all "nodes" were, in reality, a single local storage. Spark jobs could also utilize all CPU cores to simulate parallel processing, as if the tasks were running across multiple nodes.

Data was uploaded and then read with Spark as if it were stored in HDFS. The data was processed with Spark to handle empty and missing values, remove duplicates, and filter out negative values. It was then grouped by country, date, and product description. The results were saved in Cassandra, then retrieved for visualization with Matplotlib. A linear regression model was applied to generate forecasts, and these forecast results were also stored in Cassandra.

4. Result:

The implementation resulted in several analytical outputs stored in Cassandra, including tables for positive sales, returns, and negative values. Each table captures important aspects such as country,

product details, and sales metrics. For instance, the `positive_sales` and `positive_values` tables track successful sales transactions, while `returns` and `negativ_values` capture returned or unsold items, providing insights into products that did not sell as expected.

Another key table, `product_sales_per_country`, aggregates total sales by product description and country, helping to identify top-selling items per region. The `sales_forecast` table offers monthly sales forecasts based on historical data, providing a basis for predicting future demand.

These results were utilized to generate visualizations, such as aggregated sales forecasts for different countries, and to identify the products expected to sell the most and the least. The final analysis offers valuable insights for optimizing inventory and budget planning by highlighting which products to prioritize in stock and which ones to avoid over-ordering.

5. Instruction for Running the code:

To replicate this project, follow these steps. These instructions are based on implementation on a MacBook Pro:

Step 1: Install Necessary Dependencies

1. Install Java using → brew install openjdk@11
2. Install Hadoop using → brew install hadoop
3. Install Spark using → brew install apache-spark
4. Install Cassandra → brew install cassandra
5. Load the dataset from → https://www.kaggle.com/datasets/saurabhbadole/supermarket-data?select=supermarket_data.csv

Step 2: Set Up Hadoop and HDFS

1. Configure core-site.xml and hdfs-site.xml as follows:

For core-site.xml:

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

For hdfs-site.xml:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

2. Format and start HDFS using → hdfs namenode -format
start-dfs.sh

Step 3: Load Data to HDFS

Create a directory in HDFS and upload your data using:

- hdfs dfs -mkdir -p /data
- hdfs dfs -put /Users/hibaqutbuddinhabib/Desktop/supermarket_data.csv /data

Step 4: Launch Jupyter Notebook and Spark Session

1. Start Jupyter Notebook using → jupyter notebook
2. In Jupyter Notebook, set up your Spark session, load data from HDFS, perform analyses, save the results to Cassandra, and visualize the findings. Run the provided code in each cell of the notebook to execute these steps sequentially.

`