Final Project

Felipe Rodriguez

Bellevue University

DSC 650 Big Data

Professor Nasheb Ismaily

March 1, 2024

Product Inventory Management can be a challenge for up-and-coming companies. Identifying trends of transactional data can be beneficial to companies to understand the way their business is running and to understand any potential gaps that may exist. The purpose of this project is to analyze transactional e-commerce data from a UK based company. The data in this study ranges from 01/12/2010 and 09/12/2011. Analyzing this data will give insight to the company's transactions and will provide stakeholders insight to their sales. The goal is to discover areas that can be improved and create suggestions to improve profits.

Prior to beginning analytics, the data must be prepared for querying. The tools used in this study are HDFS and Spark. These two are used together in this study because Spark can make use of the data stored in HDFS, spark does not have a system to organize files, while HDFS does (source 1). There are six steps total that are done when preparing the data and querying. The first five steps involve loading the data into HDFS and preparing the data in Spark table for querying. The sixth step is for the queries conducted to reveal insights of the data. Below are the steps laid out and the methods approached.

1. Load data into VM from a local path using scp

```
(base) feliperodriguez@Felipes-MacBook-Pro-2 Downloads % scp sales-data.csv feliperodriguez@34.174.116.170:/tmp
sales-data.csv

data.csv                                              29%   13MB   1.2MB/s   00:24sales-data.csv
es-data.csv               sales-sales-data.csv                    sales-sales-sales-sales-sales-sales-sales-data.csv
                        33%   15MB   1.8MB/s   00:sales-data.csv
                                              34%   15MB   1.6MB/s   0sales-data.csv
               sales-data.csv                                                          34%   15MB   1.9MB/s sales-
B   2.0MB/ssales-data.csv                                                      34%   15MB   1.8MB/sales-data.csv
les-data.csv                                                 3sales-data.csv
                           35%   15MB   1.sales-data.csv                                                        35%
   16MB   sales-data.csv                                          36%   16MB  sales-data.csv
                             37%   16MBsales-data.csv         sales-data.csv                                  sales-dat
    sales-data.csv                                              sales-data.csv
csv                                                sales-data.csv         sales-data.csv
                                                                              41%   sales-data.csv       sales-da
es-data.csv       sales-data.csv
                                    sales-data.csv         sales-data.csv
                                                                 sales-data.csv
.csv                        sales-data.csv                       sales-data.csv     sales-data.csv     sales-data.csv
sales-data.csv
(base) feliperodriguez@Felipes-MacBook-Pro-2 Downloads %
```

2. Copy file into docker container.

```
feliperodriguez@big-data:~/dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase$ docker compose cp /tmp/sales-data.csv master:/data/
[+] Copying 1/1
 ✔ hadoop-hive-spark-hbase_master_1 copy /tmp/sales-data.csv to hadoop-hive-spark-hbase_master_1:/data/ Copied
       0.1s
```

3. Add file and confirm it has been loaded into the container

```
bash-5.0# hdfs dfs -put /data/sales-data.csv /
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/im
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2024-03-01 01:39:52,002 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
bash-5.0# hdfs dfs -ls
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/im
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2024-03-01 01:40:04,369 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
Found 1 items
drwxr-xr-x   - root supergroup          0 2024-02-29 22:47 .hiveJars
bash-5.0# hdfs dfs -ls /
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/im
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2024-03-01 01:40:18,530 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
Found 7 items
drwxr-xr-x   - root supergroup          0 2024-02-29 22:48 /hbase
drwxr-xr-x   - root supergroup          0 2024-02-29 22:46 /log
-rw-r--r--   1 root supergroup   45580638 2024-03-01 01:39 /sales-data.csv
drwxr-xr-x   - root supergroup          0 2024-02-29 22:47 /spark-jars
drwxr-xr-x   - root supergroup          0 2024-02-29 22:47 /tez
drwxr-xr-x   - root supergroup          0 2024-02-29 22:46 /tmp
drwxrwx---   - root supergroup          0 2024-02-29 22:47 /user
```

4. Create SparkScala Session and load data into Spark from the container.

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.0
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_275)
Type in expressions to have them evaluated.
Type :help for more information.

scala>  val df = spark.read.format("csv").option("header", "true").load("/data/sales-data.csv")
df: org.apache.spark.sql.DataFrame = [InvoiceNo: string, StockCode: string ... 6 more fields]

scala>
```

5. Load data into table, create temp view and confirm data has loaded.

```
scala>  val df = spark.read.format("csv").option("header", "true").load("/data/sales-data.csv")
scala> spark.sql("SELECT * FROM df limit 5").show()
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
|InvoiceNo|StockCode|         Description|Quantity|   InvoiceDate|UnitPrice|CustomerID|       Country|
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/2010 8:26|     2.55|     17850|United Kingdom|
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/2010 8:26|     2.75|     17850|United Kingdom|
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
```

6. Begin Data Analytics and query the data.

```
scala> val uniqueItemCount = df.select("country").distinct().count()
uniqueItemCount: Long = 38
```

```
scala> val totalItems = df.agg(sum("Quantity")).collect()(0)(0)
totalItems: Any = 5176450.0
```

```
scala> val dfWithTotalPrice = df.withColumn("TotalPrice", col("Quantity") * col("UnitPrice"))
dfWithTotalPrice: org.apache.spark.sql.DataFrame = [InvoiceNo: string, StockCode: string ... 7 more fields]

scala> val totalSales = dfWithTotalPrice.agg(sum("TotalPrice")).collect()(0)(0)
totalSales: Any = 9747747.93399951
```

```
scala> val dfWithTotalPrice = df.withColumn("TotalPrice", col("Quantity") * col("UnitPrice"))
dfWithTotalPrice: org.apache.spark.sql.DataFrame = [InvoiceNo: string, StockCode: string ... 7 more fields]

scala> val totalPriceByCountry = dfWithTotalPrice.groupBy("Country").agg(sum("TotalPrice").as("TotalPriceSum"))
totalPriceByCountry: org.apache.spark.sql.DataFrame = [Country: string, TotalPriceSum: double]

scala> val uniqueCostumerCount = df.select("CustomerID").distinct().count()
uniqueCostumerCount: Long = 4373

scala> val uniqueInvoiceCount = df.select("InvoiceNo").distinct().count()
uniqueInvoiceCount: Long = 25900
```

```
scala> totalPriceByCountry.orderBy(desc("TotalPriceSum")).show(10)
+--------------+------------------+
|       Country|     TotalPriceSum|
+--------------+------------------+
|United Kingdom| 8187806.363998687|
|   Netherlands| 284661.5399999992|
|          EIRE| 263276.8199999992|
|       Germany|221698.21000000037|
|        France|197403.90000000037|
|     Australia|137077.26999999987|
|   Switzerland| 56385.35000000011|
|         Spain| 54774.58000000016|
|       Belgium|40910.960000000014|
|        Sweden| 36595.90999999998|
+--------------+------------------+
only showing top 10 rows
```

```
scala> val reordersCount = df.groupBy("CustomerID", "InvoiceNo").count().filter("count > 1").groupBy("CustomerID").count()
reordersCount: org.apache.spark.sql.DataFrame = [CustomerID: string, count: bigint]
```

```
scala> val averageReorders = reordersCount.agg(avg("count")).collect()(0)(0)
averageReorders: Any = 4.709791030758394
```

```
scala> val averageUnitPrice = df.agg(avg("UnitPrice")).collect()(0)(0)
averageUnitPrice: Any = 4.611113626082972

scala> val totalPriceByInvoice = dfWithTotalPrice.groupBy("InvoiceNo").agg(sum("TotalPrice").as("TotalPriceSum"))
totalPriceByInvoice: org.apache.spark.sql.DataFrame = [InvoiceNo: string, TotalPriceSum: double]

scala> val averageInvoiceTotal = totalPriceByInvoice.agg(avg("TotalPriceSum")).collect()(0)(0)
averageInvoiceTotal: Any = 376.3609240926644
```

```
scala> val totalQuantityByInvoice = df.groupBy("InvoiceNo").agg(sum("Quantity").as("TotalQuantity"))
totalQuantityByInvoice: org.apache.spark.sql.DataFrame = [InvoiceNo: string, TotalQuantity: double]

scala> val averageQuantityByInvoice = totalQuantityByInvoice.agg(avg("TotalQuantity")).collect()(0)(0)
averageQuantityByInvoice: Any = 199.86293436293437
```

Findings

The analysis of this data provided some insights in various areas. It is uncovered that there are a total of 38 countries that products are ordered from. Additionally, there are 4,373 costumer that ordered from 01/12/2010 to 09/12/2011. These customers had a total of 25,900 invoices. The total items sold were 5,176,450 which gave net sales of $9,747,747.94. The average price of the products is $4.61 while the average invoice is $376.36. Each consumer reorders 4.7 times on average. Each invoice had around 200 items. The top ten countries were in the following order: United Kingdom, Netherlands, EIRE, Germany, France, Australia, Switzerland, Spain, Belgium, and Sweden. The Analytical Summary below outlines the data.

Analytical Summary:

Countries sold to: 38

Unique Customers: 4,373

Total invoices: 25,900

Average Reorders: 4.7

Average Quantity per invoice: 199.86

Average Price per Item: $4.61

Total Items sold: 5,176,450

Total Sales: 9,747,747.94

Top Ten Countries

Country	TotalPriceSum
United Kingdom	8,187,806.36
Netherlands	284,661.54
EIRE	263,276.82
Germany	221,698.21
France	197,403.90
Australia	137,077.27
Switzerland	56,385.35
Spain	54,774.58
Belgium	40,910.96
Sweden	36,595.91

The findings that stick out the most are the number of items sold and price of the items. When looking at the stats, the price of the items seems low for the quantity being sold. If this was increased by 12 – 25%, the overall margin would improve. Additionally, there seems to be great retention of consumers which is displayed by the number of times they reorder. If the total customer base can be increased, this will create long term business.

In conclusion, the data provides great insight to the company and reveals that there are certain areas for improvement. There are no major areas of concern as the recommendations provide enhancements. This analysis should be refreshed when enhancements have been implemented to analyze the changes and find more opportunities.

Reference:

Carrie. (2017, August 17). *E-commerce data*. Kaggle. https://www.kaggle.com/datasets/carrie1/ecommerce-data

Jeevan, M. (2022, December 14). *Difference between Hadoop and spark: All you need to know*. Simplilearn.com. https://www.simplilearn.com/spark-vs-hadoop-article#:~:text=is%20a%20demand.-,How%20Spark%20and%20Hadoop%20Process%20Data,Distributed%20File%20System(HDFS).