Retail Store Analysis

By
Sumit Bansod
(220940325081)

Abstract

This paper aims to present a retail store analysis using Spark SQL on Cloudera, a big data platform. The analysis focuses on understanding customer behavior, identifying trends, and improving business operations. The paper will begin with a brief overview of the data source and the retail business. Next, we will discuss the data preparation process, which involves transforming the data. Then, we will present the analysis results, including customer segmentation, sales trends, and product recommendations. Finally, we will conclude with a discussion of the insights gained from the analysis and the potential impact on the business. Overall, this paper demonstrates the power of using Spark SQL on Cloudera to perform retail store analysis and provide valuable insights to support business decisions

Table of Contents

Retail Store Analysis

Introduction

Retail store analysis is a process of evaluating the performance of a retail store by analyzing various aspects of its operations including sales ,customer traffic, inventory ,product sales and their insights.
The information about the customer base,products, and sales is being studied in this project. This analysis helps retailers to identify their strengths and weaknesses, as well as opportunities for growth and improvement. With the increasing competition in the retail industry, it has become crucial for retailers to understand the market trends and consumer behavior to stay ahead of the curve.Therefore, retail store analysis is an essential tool for retailers to make data-driven decisions and optimize their business strategies. In this context, this analysis is an indispensable tool for any retail store owner or manager who aims to thrive in today's dynamic market.

Research Motivation

Retail stores are a crucial part of the economy, and their success or failure can have a significant impact on local communities and the broader business landscape. Understanding the factors that contribute to the success or failure of retail stores is essential for businesses, investors, and policymakers alike. By conducting a comprehensive analysis of retail stores, we can identify patterns and trends in consumer behavior, competition, and market dynamics that can inform strategies for improving store performance and profitability. Moreover, such an analysis can also help identify potential opportunities for innovation and growth within the retail industry, leading to new and more effective ways of serving customers and creating value for stakeholders. Therefore, the research motivation for retail store analysis is to gain a deeper understanding of the factors that drive success in the retail industry and to develop insights that can inform strategic decision-making for businesses, investors, and policymakers.

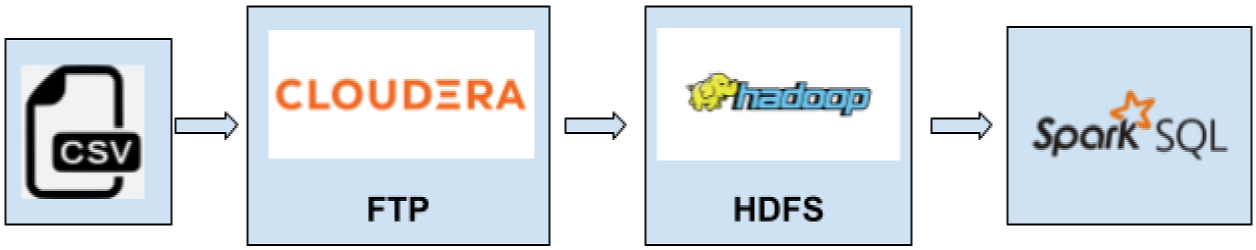
Problem Statement

The retail industry is constantly evolving with the emergence of new technologies and changing consumer behavior. Retail stores are facing the challenge of meeting customer demands while maintaining profitability in a highly competitive market. In order to stay ahead of the competition, retailers must implement efficient systems and strategies to manage inventory, optimize pricing, and enhance customer experience.Our goal is to answer the most frequent question based on customers and sales using spark sql.

Project development

An overview of the data collection to data preparation to data queries , is provided in this section. According to numerous data scientists, data cleaning and formatting can be thought of as the most important element of the entire project.The Project aims to import multiple files from ftp to Hadoop and run SQL queries using pyspark.

Methodology



Technologies used:

- Cloudera FTP
- Cloudera Hadoop HDFS
- Apache spark
- Apache spark SQL

Data Sources

The data source that we have in the analysis is a dataset of retail stores comprising sales records of products and customers. The rows of the dataset represent specific records of customers and their sales while the columns contain extensive information on the retail store such as customer,product ,etc.

For our analysis we'll be using 4 different files of recorded sales and customers for 4 different months comprising roughly around 800k records which will be enough for our analysis.

Retail Store Data

The data set contains the data of the following 4 months:

D11: Transaction data collected in November, 2000

D12: Transaction data collected in December, 2000

D01: Transaction data collected in January, 2001

D02: Transaction data collected in February, 2001

<u>Format of Transaction Data:</u>

Data columns separated by “;”

<u>Column definition:</u>

- **Transaction date and time (Time is invalid)**
- **Customer I.D**
- **Age: 10 possible values**

  A <25,B 25-29,C 30-34,D 35-39,E 40-44,F 45-49,G 50-54,H 55-59,I 60-64,J >65

- **Residence Area: 8 possible values, A-F: zip code area: 105,106,110,114,115,221,G:others, H: Unknown Distance to store, from the closest: 115,221,114,105,106,110**
- **Product subclass (category)**
- **Product ID**
- **Qty or Number of units**
- **Total Cost**
- **Total Sales**

Below is an image how the dataset actually looks like:

```
2001-01-01 00:00:00;00141833  ;F ;F ;130207;4710105011011;2;44;52
2001-01-01 00:00:00;01376753  ;E ;E ;110217;4710265849066;1;150;129
2001-01-01 00:00:00;01603071  ;E ;G ;100201;4712019100607;1;35;39
2001-01-01 00:00:00;01738667  ;E ;F ;530105;4710168702901;1;94;119
2001-01-01 00:00:00;02141497  ;A ;B ;320407;4710431339148;1;100;159
2001-01-01 00:00:00;01868685  ;J ;E ;110109;4710043552065;1;144;190
2001-01-02 00:00:00;01101270  ;D ;C ;730303;4714903310314;1;740;969
2001-01-02 00:00:00;01754698  ;H ;A ;560402;4710498601486;1;676;849
2001-01-02 00:00:00;01027365  ;F ;C ;530404;9555008600314;1;170;219
2001-01-03 00:00:00;00956710  ;E ;E ;500303;4710367208648;1;36;59
2001-01-04 00:00:00;00477796  ;E ;H ;100108;50853991     ;2;220;270
2001-01-05 00:00:00;01267471  ;C ;F ;500804;9310022733406;1;185;218
2001-01-06 00:00:00;00904391  ;F ;E ;110109;4716782102028;1;80;89
2001-01-06 00:00:00;00848602  ;A ;F ;530110;4902430000864;1;187;229
2001-01-07 00:00:00;00952521  ;D ;F ;110507;4711220013331;1;70;97
2001-01-07 00:00:00;00582346  ;I ;E ;110105;4710198221113;1;25;29
2001-01-07 00:00:00;00102100  ;E ;E ;100323;4710218360020;1;160;198
2001-01-08 00:00:00;01383744  ;E ;H ;130102;4710105002026;3;42;54
```

❖ Loading The Data using cloudera ftp

❖ Loading data on Hadoop

```
[bigcdac432513@ip-10-1-1-204 ~]$ hadoop fs -mkdir retail
[bigcdac432513@ip-10-1-1-204 ~]$ hadoop fs -put D01 D02 D11 D12 retail
```

Home / user / bigcdac432513 / **retail** Trash

Name	Size	User	Group	Permissions	Date
↥		bigcdac432513	bigcdac432513	drwxr-xr-x	December 07, 2022 07:16 AM
.		bigcdac432513	bigcdac432513	drwxr-xr-x	December 07, 2022 07:18 AM
D01	14.0 MB	bigcdac432513	bigcdac432513	-rw-r--r--	December 07, 2022 07:18 AM
D02	12.8 MB	bigcdac432513	bigcdac432513	-rw-r--r--	December 07, 2022 07:18 AM
D11	14.4 MB	bigcdac432513	bigcdac432513	-rw-r--r--	December 07, 2022 07:18 AM
D12	11.5 MB	bigcdac432513	bigcdac432513	-rw-r--r--	December 07, 2022 07:18 AM

Show 45 of 4 items Page 1 of 1

❖ Initialising Spark session

```
[bigcdac432513@ip-10-1-1-204 ~]$ spark
```

❖ Creating the schema

```
Txn_dt : String
Custno : String
Age : String
Zipcode : String
Category : String
Product : String
Qty : Int
Cost : bigint
Sales : bigint
Row format delimited
Fields terminated by '\;'
```

❖ Without changing the delimiter

```
>>> retail =
spark.read.format("csv").option("header","False").schema(schema10).loa
d("hdfs://nameservice1/user/bigcdac432513/retail")
>>> retail.count()
817741
>>> retail.show()
+--------------------+-------+----+-------+--------+-------+----+----+----+
|              txn_id|cust_id| age|zipcode|category|product| qty|cost|sale|
+--------------------+-------+----+-------+--------+-------+----+----+----+
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
```

```
|2000-11-01 00:00:...|    null|null|    null|     null|    null|null|null|null|
|2000-11-01 00:00:...|    null|null|    null|     null|    null|null|null|null|
|2000-11-01 00:00:...|    null|null|    null|     null|    null|null|null|null|
|2000-11-01 00:00:...|    null|null|    null|     null|    null|null|null|null|
+--------------------+-------+----+-------+--------+-------+----+----+----+
only showing top 20 rows
```

```
>>> retail = spark.read.format("csv").option("header","False").schema(schema10).load("hdfs://
>>> retail.count()
817741
>>> retail.show()
+--------------------+-------+----+-------+--------+-------+----+----+----+
|              txn_id|cust_id| age|zipcode|category|product| qty|cost|sale|
+--------------------+-------+----+-------+--------+-------+----+----+----+
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
|2000-11-01 00:00:...|   null|null|   null|    null|   null|null|null|null|
+--------------------+-------+----+-------+--------+-------+----+----+----+
only showing top 20 rows
```

❖ Changing the default delimiter

```
>>> retail =
spark.read.format("csv").option("sep",";").option("header","False").sc
hema(schema10).load("hdfs://nameservice1/user/bigcdac432513/retail"
)
>>> retail.show()
+-------------------+----------+---+-------+--------+-------------+---+----+----+
|             txn_id|   cust_id|age|zipcode|category|      product|qty|cost|sale|
+-------------------+----------+---+-------+--------+-------------+---+----+----+
|2000-11-01 00:00:00|00046855  | D |     E |  110411|4710085120468|  3|  51|  57|
|2000-11-01 00:00:00|00539166  | E |     E |  130315|4714981010038|  2|  56|  48|
|2000-11-01 00:00:00|00663373  | F |     E |  110217|4710265847666|  1| 180| 135|
|2000-11-01 00:00:00|00340625  | A |     E |  110411|4710085120697|  1|  17|  24|
|2000-11-01 00:00:00|00236645  | D |     H |  712901|8999002568972|  2| 128| 170|
|2000-11-01 00:00:00|01704129  | B |     E |  110407|4710734000011|  1|  38|  46|
```

```
|2000-11-01 00:00:00|00841528  | C |      E |  110102|4710311107102|  1|  20|  28|
|2000-11-01 00:00:00|00768566  | K |      E |  110401|4710088410382|  1|  44|  55|
|2000-11-01 00:00:00|00217361  | F |      E |  130401|4711587809011|  1|  76|  90|
|2000-11-01 00:00:00|02007052  | D |      E |  110504|4710323168054|  1|  17|  20|
|2000-11-01 00:00:00|01607000  | D |      F |  500201|4710291138134|  1|  95| 109|
|2000-11-01 00:00:00|01847987  | D |      E |  110401|4710088410610|  1|  19|  25|
|2000-11-01 00:00:00|00663373  | F |      E |  530105|4901616005822|  1| 113| 129|
|2000-11-01 00:00:00|00539166  | E |      E |  110411|4710085172696|  1|  20|  19|
|2000-11-01 00:00:00|02079127  | A |      E |  100505|4710154012144|  1|  15|  19|
|2000-11-01 00:00:00|02007052  | D |      E |  110506|4710320224265|  1| 157| 168|
|2000-11-01 00:00:00|00663373  | F |      E |  110411|4710085172702|  1|  20|  19|
|2000-11-01 00:00:00|01873139  | C |      F |  120103|4710011409056|  1|  23|  29|
|2000-11-01 00:00:00|00190138  | F |      E |  130315|4714981010038|  2|  56|  48|
|2000-11-01 00:00:00|00647236  | B |      E |  130206|4710339012013|  1|  24|  29|
+-------------------+----------+---+-------+--------+-------------+---+----+----+
only showing top 20 rows
```

- **Naming the table as retail.**

```
>>> retail.registerTempTable("retail")
```

```
>>> retail.show()
+-------------------+----------+---+-------+--------+-------------+---+----+----+
|             txn_id|   cust_id|age|zipcode|category|      product|qty|cost|sale|
+-------------------+----------+---+-------+--------+-------------+---+----+----+
|2000-11-01 00:00:00|00046855  | D |      E |  110411|4710085120468|  3|  51|  57|
|2000-11-01 00:00:00|00539166  | E |      E |  130315|4714981010038|  2|  56|  48|
|2000-11-01 00:00:00|00663373  | F |      E |  110217|4710265847666|  1| 180| 135|
|2000-11-01 00:00:00|00340625  | A |      E |  110411|4710085120697|  1|  17|  24|
|2000-11-01 00:00:00|00236645  | D |      H |  712901|8999002568972|  2| 128| 170|
|2000-11-01 00:00:00|01704129  | B |      E |  110407|4710734000011|  1|  38|  46|
|2000-11-01 00:00:00|00841528  | C |      E |  110102|4710311107102|  1|  20|  28|
|2000-11-01 00:00:00|00768566  | K |      E |  110401|4710088410382|  1|  44|  55|
|2000-11-01 00:00:00|00217361  | F |      E |  130401|4711587809011|  1|  76|  90|
|2000-11-01 00:00:00|02007052  | D |      E |  110504|4710323168054|  1|  17|  20|
|2000-11-01 00:00:00|01607000  | D |      F |  500201|4710291138134|  1|  95| 109|
|2000-11-01 00:00:00|01847987  | D |      E |  110401|4710088410610|  1|  19|  25|
|2000-11-01 00:00:00|00663373  | F |      E |  530105|4901616005822|  1| 113| 129|
|2000-11-01 00:00:00|00539166  | E |      E |  110411|4710085172696|  1|  20|  19|
|2000-11-01 00:00:00|02079127  | A |      E |  100505|4710154012144|  1|  15|  19|
|2000-11-01 00:00:00|02007052  | D |      E |  110506|4710320224265|  1| 157| 168|
|2000-11-01 00:00:00|00663373  | F |      E |  110411|4710085172702|  1|  20|  19|
|2000-11-01 00:00:00|01873139  | C |      F |  120103|4710011409056|  1|  23|  29|
|2000-11-01 00:00:00|00190138  | F |      E |  130315|4714981010038|  2|  56|  48|
|2000-11-01 00:00:00|00647236  | B |      E |  130206|4710339012013|  1|  24|  29|
+-------------------+----------+---+-------+--------+-------------+---+----+----+
only showing top 20 rows
```

- Now we can work on the analysis of the retail store and answer the queries.
- Following are the insights we have got using the SQL queries

1) Count of unique customers and total sales for each age group and for a given month = Jan

A	5000	600000
B	4500	540000

```
>>> C_uc_ts =spark.sql("select count(distinct(cust_id)),age ,
sum(sale) as total from retail where month(txn_id)=1 group by age
order by age")
>>> C_uc_ts.show()
+-----------------------+---+-------+
|count(DISTINCT cust_id)|age|  total|
+-----------------------+---+-------+
|                    686| A | 908494|
|                   1489| B |2565452|
|                   2922| C |5358113|
|                   3346| D |6694313|
|                   2691| E |5521770|
|                   2031| F |3797484|
|                   1229| G |2135538|
|                    594| H |1040837|
|                    506| I | 938571|
|                    769| J | 955095|
|                    315| K | 772812|
+-----------------------+---+-------+
```

2) Count of unique customers and total sales for <u>one age group(A) </u>for all products - [sorting data on totalsales desc- to find top 10]

ProdA count of unique cust total sales

ProdB count of unique cust total sales

```
>>> Ques2 = spark.sql("select
count(distinct(cust_id)),product,sum(sale) as totalsales from retail
where trim(age)='A' group by product order by total
sales desc")
>>> Ques2.show(10)
+-----------------------+-------------+----------+
|count(DISTINCT cust_id)|      product|totalsales|
+-----------------------+-------------+----------+
|                      3|4711588210441|    446185|
|                     15|8712045008539|     38521|
|                     18|4710036008562|     16137|
|                     58|4710036003581|     15843|
|                      3|4718387034025|     15178|
|                      4|4710628119010|     14997|
|                     65|4710265849066|     14440|
|                      4|20553418     |     14360|
|                     53|4710114128038|     14114|
|                      8|4973167032060|     13702|
+-----------------------+-------------+----------+
only showing top 10 rows
```

3) Area wise sales

```
>>> Area3 = spark.sql("select sum(sale),zipcode from retail group by zipcode")

>>> Area3.show()
+---------+-------+
|sum(sale)|zipcode|
+---------+-------+
| 37994770|     E |
| 31548341|     F |
|  5245595|     H |
|  5447881|     D |
|  9866326|     C |
|  2344440|     A |
|  2799090|     B |
| 12593633|     G |
+---------+-------+
```

4)Top 10 viable products (prod which give highest profit)

```
>>> viablep= spark.sql("select product,sum((sale-cost)) as profit from retail group by product order by profit desc limit 10")
>>> viablep.show()
+-------------+------+
|      product|profit|
+-------------+------+
|4909978112950| 71312|
|8712045008539| 46586|
|20564100     | 38699|
|4710628131012| 34429|
|0729238191921| 33645|
|4902430493437| 32970|
|20556433     | 31862|
|4901422038939| 31616|
|4710114128038| 29168|
|7610053910787| 26839|
+-------------+------+
```

```
>>> q4.count()
23812
```

5) ***Identifying all loss making products - Display all the loss making products from highest loss to the least***

```
>>> viableloss= spark.sql("select product,sum(cost-sale) as loss from retail group by product order by loss desc limit 10")
>>> viableloss.show()
+-------------+------+
|      product|  loss|
+-------------+------+
|4714981010038|131002|
|4711271000014| 46213|
|4719090900065| 44331|
|4710265849066| 38969|
|4712425010712| 17646|
|2110119000377| 17457|
|4710908110232| 15072|
|4719090900058| 14034|
|4710265847666| 11657|
|4710683100015| 11456|
+-------------+------+
```

```
>>> q5.count()
101
```

```
>>> q5.count()
101
>>> q5.rdd.getNumPartitions()
95
>>> q51 = q5.repartition(1)
>>> q51.show()
+-------------+------+
|      product|profit|
+-------------+------+
|4710265849066|-38969|
|4710063031106| -1556|
|4710395340020| -1166|
|4710036008548|  -835|
|4934567661519|   -35|
|20571344     |   -31|
|4712425010712|-17646|
|2110119000377|-17457|
|4719090900058|-14034|
|4710265847666|-11657|
|4714008033507| -8065|
|4710060000099| -6291|
|4712162000038| -6111|
|4902430489133| -4173|
|4710012114331| -4053|
|4710168705056| -3893|
|4710422600035| -3851|
|4710036003581| -2948|
|4710036008562| -2596|
|4713071811159| -2395|
+-------------+------+
only showing top 20 rows
```

❖ Coalesce

- ❖ In PySpark, coalesce() is a transformation function that is used to reduce the number of partitions in a DataFrame or RDD (Resilient Distributed Datasets) without shuffling the data. It takes a single argument, which is the desired number of partitions.

- ❖ The coalesce() function works by merging adjacent partitions together to form larger partitions. It does not perform a full shuffle operation, which means that it can be more efficient than repartition() in cases where the data does not need to be evenly distributed across the new partitions.

```
>>> q51 = q5.coalesce(1)
>>> q51.show()
+-------------+-------+
|      product| profit|
+-------------+-------+
|4714981010038|-131002|
|4711271000014| -46213|
|4719090900065| -44331|
|4710265849066| -38969|
|4712425010712| -17646|
|2110119000377| -17457|
|4710908110232| -15072|
|4719090900058| -14034|
|4710265847666| -11657|
|4710683100015| -11456|
|4711001121101| -10887|
|4714008033507|  -8065|
|4710363352000|  -6871|
|4710060000099|  -6291|
|4712162000038|  -6111|
|4719852310019|  -5003|
|20562670     |  -4819|
|4902430489133|  -4173|
|4710012114331|  -4053|
|4710168705056|  -3893|
+-------------+-------+
only showing top 20 rows
```

Conclusion

- Highest sales is for the age group D i.e.,6694313
- Product with the product id 4711588210441 has most sales i.e., 446185
- Area with zipcode "E" has the most sale i.e., 37994770 followed by area "F" with sale 31548341.
- Product with id 4909978112950 was most profitable with profit of 71312 followed by product 8712045008539 with profit 46586.
- Product with id 4714981010038 has the most loss of all the products i.e., 131002.