Article

Not peer-reviewed version

Optimizing Retail Decision-Making Through Big Data and Machine Learning Integration

Abrar Hasnain , Waleed Arnaout , Naeema Faisal , Adlina Batrisyia Binti Zainudden , Aaliyah Natasha Hamdan , Noor Amin *

Posted Date: 9 December 2025

doi: 10.20944/preprints202512.0675.v1

Keywords: big data analytics; retail industry; Apache hive; Apache pig; impala; machine learning; real-time processing; data scalability; customer experience; predictive analysis

 doi:10.20944/preprints202512.0675.v1

Article

Optimizing Retail Decision-Making Through Big Data and Machine Learning Integration

Abrar Hasnain, Waleed Arnaout, Naeema Faisal, Adlina Batrisyia Binti Zainudden, Aaliyah Natasha Hamdan and Noor Amin *

Taylor's University, Malaysia

* Correspondence: nooraminnawab@gmail.com

Abstract

The study explores how modern big data technologies can transform operations and customer experience in the retail industry. With the massive growth of structured and unstructured data from online and in-store transactions, businesses face increasing pressure to process information quickly and accurately. This project demonstrates how tools such as Apache Hive, Pig, and Impala can be applied to handle real-time data analytics efficiently. A retail-based dataset consisting of over forty-nine thousand sales transactions was analyzed to uncover sales trends, top-performing products, and revenue patterns. The performance of these tools was evaluated based on execution time, scalability, and memory utilization. Impala exhibited the fastest processing speed, making it suitable for real-time analytics, while Hive and Pig proved effective for batch processing and data transformation tasks. Furthermore, the integration of machine learning algorithms—such as regression, clustering, and decision trees—was discussed for enhancing predictive accuracy and personalized customer insights. The findings highlight that combining these technologies provides a scalable, cost-effective framework for optimizing marketing, forecasting demand, and improving business decision-making in the retail sector.

Keywords: big data analytics; retail industry; Apache hive; Apache pig; impala; machine learning; real-time processing; data scalability; customer experience; predictive analysis

Introduction

The retail industry faces increasing challenges in providing personalized customer experiences amidst the rapidly evolving consumer behavior and large-scale data proliferation. This assignment explores how big data technologies can be leveraged to optimize customer interactions, predict the preferences and improve decision making processes. By utilizing the advanced tools such as Hive, Spark, and machine learning algorithms, the project aims to analyze retail datasets effectively, service actionable insights, and offer practical solutions to real world problems. Through the implementation of SQL- like queries and predictive models, the assignment demonstrates the transformative potential of big data in driving information, enhancing customer satisfaction, and fostering the growth of business.

Background and Overview

1.0. Real-Time Data Processing in Retail

Overview

The term big data is used to describe vast sets of structured and unstructured data that are generated from different sources, including digital libraries, social networks, mobile apps, internet clickstream logs, emails, documents, and customer databases, among others (Guru & Bajnaid, 2023). The analysis of these sets of data is referred to as big data analytics and it involves the systematic

 doi:10.20944/preprints202512.0675.v1

collection, processing, and interpretation of the data to extract valuable insights that can be used to make informed decisions.

In retail, big data analytics has become paramount over the past few decades due to its ability to provide valuable and actionable knowledge to businesses. With large amounts of data being generated daily from social networks, customer transactions, and online interactions, big data analytics has emerged as one of the key drivers of success for businesses in the retail sector (Hammond-Errey, 2023). By helping businesses get key insights into customer behavior and their purchasing patterns, how each product is performing, the current market trends, and the efficiency of supply chains, it helps them make informed decisions that can improve their profitability. For instance, it allows them to know which operations should be optimized, how to tailor their marketing campaigns, and how their pricing should be adjusted to stay competitive and improve their sales.

As highlighted by Giri et al. (2019), analyzing big data requires specialized tools. This is because big data is essentially characterized by three key aspects:

- Volume – it comprises particularly large sets of data that traditional storage and processing systems cannot handle efficiently.
- Velocity - it is generated at very high speeds, and is required to be collected and processed in real-time. For instance, streaming data from social media platforms like YouTube or Instagram.
- Variety – it is collected in different forms, including structured, such as databases; semi-structured such as logs and JSON; and unstructured data such as videos, images, and text (Guru & Bajnaid, 2023).

As such, various technologies and tools are employed to collect, process, and transform this complex and heterogeneous data into something useful that can be used by the business to make informed decisions and optimize their operations accordingly. For instance, emerging technologies like IoT (Internet of Things), AI (artificial intelligence), and machine learning are being implemented for big data analytics (Niu & He, 2019).

Big Data Technologies

These include the various tools and techniques that are utilized to mine, clean, integrate, and visualize big data in real time to make it easy for businesses to obtain valuable insights from large, varying, and complex sets of data (Ravi, 2024). This allows them to personalize their marketing strategies and improve customer interactions. For instance, technologies like Apache Kafka and Google BigQuery facilitate the real-time processing of continuous vast data from social networks, e-commerce sites, and customer interactions. Major companies like LinkedIn, Netflix, Uber, Airbnb, and Spotify use Kafka and BigQuery to process and analyze vast, real-time datasets (Sharma *et al*., 2021). They allow them to create personalized recommendations to improve user experiences, get valuable insights into customer preferences, and make data-driven decisions to improve their productivity. This process follows a specific structure that involves data being collected from multiple sources, immediate computations, such as statistical measures and aggregations, being performed to transform the collected raw data into meaningful formats, and visual representations of the processed data being produced in form of data trends and patterns to help the retailer make informed decisions about their business operations.

 doi:10.20944/preprints202512.0675.v1

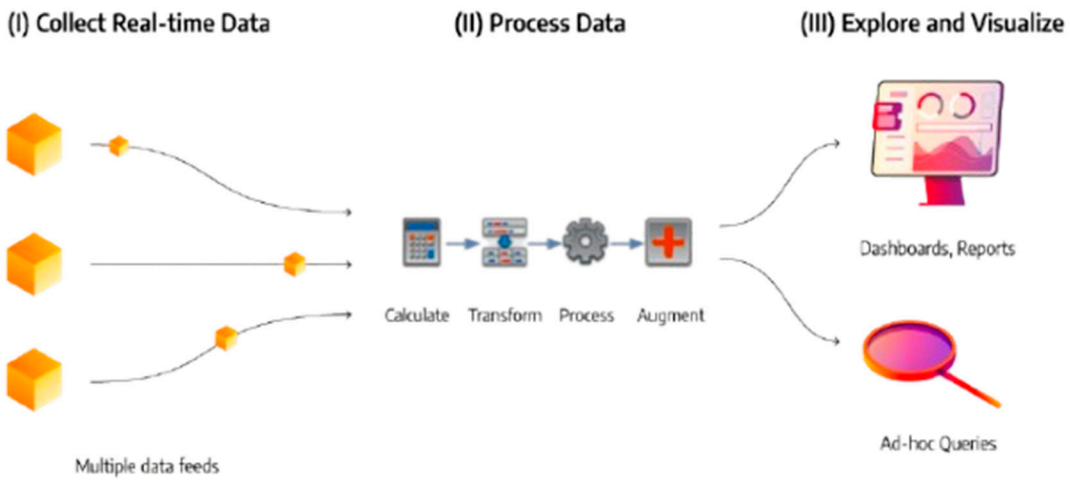


Figure 1. A typical process in real-time big data analytics.

These recommendations can be in the form of:

- Dynamic recommendations, whereby users are given suggestions in real-time according to how they interact with the platform and/or their preferences (Solanki *et al*., 2022). Take, for instance, an individual visiting an e-commerce platform, the program analyzes their previous purchases, search history, and related information to recommend products as they are browsing.
- Targeted promotions, in which the customer is sent tailored discounts and/or offers through the platform, SMS, or email while they are actively using the platform.
- Geo-targeted ads, whereby the customer receives specific alerts or advertisements depending on their current location (Thakur, 2023).

According to Guru & Bajnaid (2023), retailers are able to make such quick and data-driven recommendations due to real-time data processing enabled by data analytics and machine learning algorithms. For instance, retail giant Amazon provides real-time product recommendations to customers using machine learning-powered big data technologies like AWS Machine Learning Services and Amazon Personalize which analyze customers' purchase history, behavior patterns, and browsing activities in real-time to provide suggestions in the "Customers Also Bought" section.

In conclusion, Big data analytics makes it easy for businesses in the retail industry to uncover correlations, patterns, and trends in vast volumes of raw data. Extracting and analyzing this data is a complex process that leverages big data technologies that employ data analytics and machine learning algorithms (Sharma *et al*., 2021). This process enables businesses to take advantage of the rapidly increasing data generated from varied sources, including point-of-sale systems, online purchases, social media interactions, and loyalty programs, among others to obtain actionable intelligence via advanced analytic systems. This intelligence is significant for providing deeper insight into customer behavior, predicting customer preferences, and consequently, future trends, all of which play a major role in boosting the customer experience.

2.0. Data Analytics and Machine Learning Algorithms

Overview

The data collected for big data analytics is normally vast and varied, comprising structured data like databases; semi-structured such as logs and JSON; and unstructured data such as videos, images, and text (Guru & Bajnaid, 2023). This makes it very hard to manage the data or get useful insight

 doi:10.20944/preprints202512.0675.v1

using traditional methods like paper-based analysis, visual inspection, simple calculations, checklists, tally sheets, and comparative analysis. These methods are labor-intensive, prone to errors, and thus, not fitting for analyzing large or complex datasets. For instance, according to a recent report by Invisibly, the giant retailer Amazon collects and processes about 1 exabyte of purchase history data from its customer base for big data analytics. Using the aforementioned traditional methods to collect or analyze this data would be very challenging. As such, AI-powered methods are used to extract, transform, and process this data to make it easy to study patterns related to current trends, customer behavior, and their interactions (Ravi, 2024). This helps the company in a number of ways including, inventory management optimization, personalization of marketing strategies, improvement of pricing strategies, operational efficiency enhancement, fraud detection and security enhancement, and improvement of customer support.

Machine Learning Algorithms for Customer Experience Optimization

These AI-powered methods specifically employ machine learning algorithms to facilitate various processes, including automation of repetitive tasks, sentiment analysis, and prediction of customer needs (Solanki *et al*., 2022). For instance, statistical algorithms, such as linear regression, are used by retail companies like Walmart to analyze historical data and other related information to predict customer behavior and sales and adjust their operations accordingly. Machine learning algorithms, such as collaborative filtering, clustering algorithms, and NLP (natural language processing), are largely used by Amazon to analyze sentiment and textual data such as customer reviews to better understand their preferences and adjust accordingly (Ravi, 2024). This helps the company enhance product recommendations, and improve their services, for a better customer experience.

For instance, Apache Spark, which is one of the common big data analytics programs, being used by world's leading retailers like Amazon, Walmart, Alibaba, Target, and eBay, among others, utilizes a machine learning-based library called MLlib (Luu, 2021). This library leverages key machine learning algorithms to facilitate the collection, processing, analysis, and visualization of big data. These include machine learning algorithms like:

- Linear regression – this helps to facilitate the prediction of numerical outcomes, for example in sales forecasting (Thakur, 2023).
- K-Means clustering – this helps facilitate customer segmentation in which similar customers are grouped according to their purchasing behaviors.
- Decision Trees – this helps in classification tasks like fraud detection.
- Collaborative Filtering – this is used in recommendation systems to suggest products and services to customers in real-time on e-commerce sites (Thakur, 2023).

Types of Data Analytics

Depending on the specific business or analysis objective, there are four key data analytic methods that can be applied to help retailers extract valuable insights from complex unstructured big data (Giri *et al*., 2019). Each of these methods includes a specific purpose to help retailers make informed decisions and improve customer experience. They include:

- Descriptive analytics – this focuses on the analysis of past related data such as customer behavior, preferences, and trends to provide insights into how customers interacted with the business in the past (Tolulope, 2022). For instance, Amazon analyzes customer purchase history and satisfaction survey data to determine the most popular products or the categories with the highest customer visitation.
- Diagnostic analytics – this involves the factor or reasons behind specific customer behavior being investigated to determine their root cause (Guru & Bajnaid, 2023). For instance, it can involve the company reviewing submitted customer complaints and returned products to determine why a specific product is not performing as expected or why someone would be dissatisfied

 doi:10.20944/preprints202512.0675.v1

with it. A good example of this in practice is with Target, which employs diagnostic analysis to investigate why sales are declining in specific stores by analyzing regional customer behavior, competitor activity, and local economic conditions to determine the reason why (Cote, 2021).

- Predictive analytics - in this, past data, such as past customer shopping patterns, is used to forecast customer preferences, behaviors, and market trends (Shi, 2022). This could help the company learn which products are likely to sell more in a specific season. Therefore, it could help the retailer tailor its marketing campaigns and inventory planning to improve customer experience and boost sales. For instance, Walmart employs machine learning algorithms to forecast demand for seasonal products such as holiday decorations or back-to-school supplies, to ensure that the inventory levels are optimum (Yi, 2023).
- Prescriptive analytics – this suggests the best action to take to enhance business operations and improve customer experience (Tolulope, 2022). For instance, the analytics tool can provide recommendations on how to personalize offers, adjust store layout, or improve product availability based on data from customer feedback and predictive models. The goal of this data analytics method is to meet the customer expectations to improve their experience. This has been applied successfully by Macy's via dynamic pricing algorithms capable of adjusting prices in real time based on demand forecasts, competitor pricing, and stock levels, maximizing profitability while maintaining reasonable pricing (Chandramouli, 2023).

In reality, most retail companies leverage a combination of these analytics methods to provide key insights and improve decision-making at various stages of operation (Shi, 2022). For instance, as indicated in Figure 1 below, the company could begin with descriptive analytics to have a clearer picture of past performances of a specific product, followed by diagnostic analytics to ascertain the reasons behind those specific performances, then utilize predictive analytics to predict how the product might perform, and lastly use prescriptive analytics to determine the best actions to take to improve customer experience.

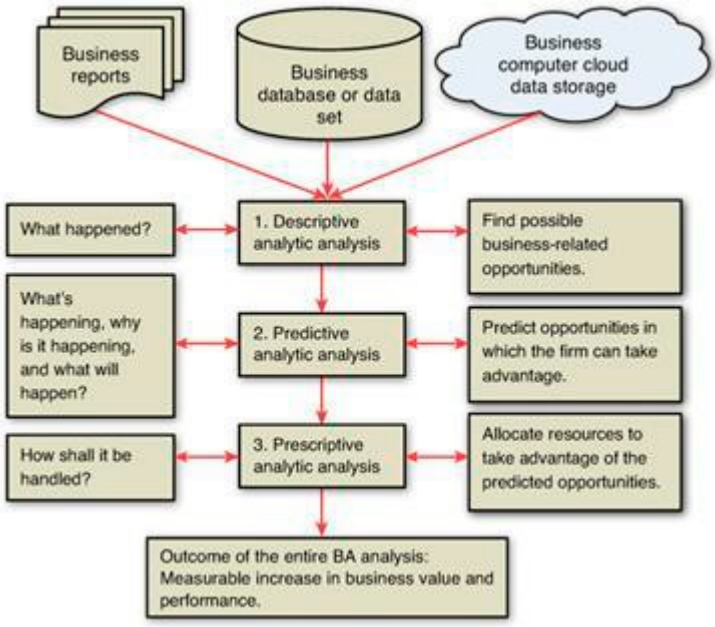


Figure 2. A typical business analytic process depicting a combination of the different analytic methods.

Machine learning algorithms serve a major role in big data analytics for retail. They simplify the processes of processing and analyzing vast amounts of customer, transaction, and inventory data in real-time to uncover trend patterns, optimize inventory, enhance marketing strategies, predict customer behavior, and provide personalized recommendations to improve customer experience. This is why more companies have started to integrate AI-based data analytics [37–40] and the trend is expected to grow. In fact, according to Gartner, the integration of AI technology will play key role

 doi:10.20944/preprints202512.0675.v1

in retail in the next few years, with integration in almost 80% of customer service and support services for improved customer experience and satisfaction (Gartner, 2024).

3.0. Key Considerations for Designing a Scalable Solution

In designing a scalable and efficient big data solution for retail, it is important to consider important aspects such as storage, processing speed and the integration of multiple data sources. Due to the high volume of data being generated by the retail industry, big data solutions and technology play an important role in improving the retail industry in terms of analysing customers' behaviour, improving customer service and increasing sales [41–43].

In terms of storage, the retail industry generates many different types of data which includes inventory data, transactional data and more. Therefore, it is important to have a scalable and efficient storage system in place. To handle the amount of data being generated, a distributed storage system would be the best choice as the data will be distributed across different storage devices which allows for easy and efficient management and security. [1]

An example of a distributed storage system is the Hadoop Distributed File System (HDFS). The HDFS utilises a master-slave architecture where the entire HDFS framework is managed by NameNode (NN) and files are stored in the DataNodes (DNs). Hadoop will divide the data blocks and store them in different DNs. In case of a DN failure, each data block is replicated by default 3 times on several DNs with Hadoop's replication system. [2] With the utilisation of MapReduce methodology, Hadoop is capable of performing parallel computing. Data will be divided into smaller chunks and be processed separately which improves the way data are stored and processed. [3]

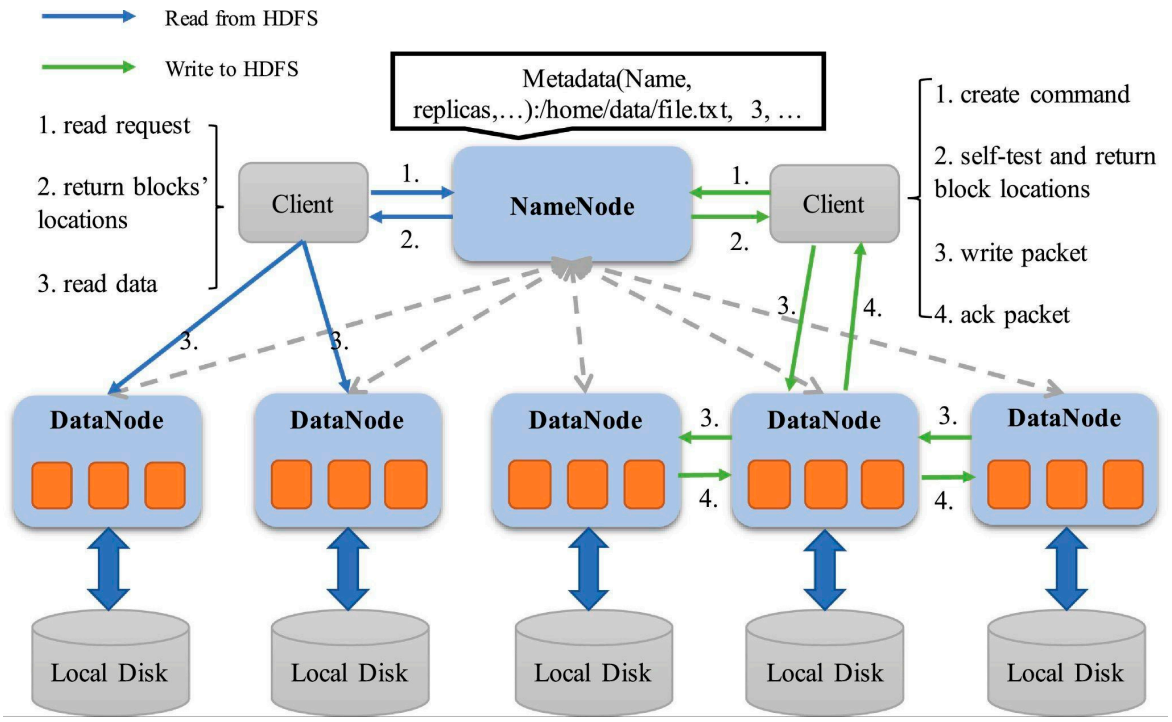


Figure 3. Hadoop Distributed File System Architecture.

In terms of processing speed, due to the high volume of data, it is pivotal to ensure that the data are being handled and processed efficiently to derive valuable analysis that will help improve the retail industry. The MapReduce methodology in Apache Hadoop enables the parallel processing of a large number of datasets which consequently improves the speed of data processing. Other than that, in-memory processing would significantly increase the data processing. An example would be Apache Spark. Spark utilises memory to store data which decreases the read/write cycle which is the slowing factor of processing speed in the Hadoop MapReduce framework. [3]

 doi:10.20944/preprints202512.0675.v1

The retail industry operates across different platforms and systems, where a high volume of data is individually generated. These sources include the POS system, e-commerce platforms and supply chains. Therefore it is important to implement a system where the integration of multiple data sources is done efficiently. Apache Kafka is a tool that will allow effective data integration. Kafka allows the real-time streamlining of data and real-time data processing [44–48]. Other than that, Kafka utilises partitions for scale and multiple replicas which promotes durability and fault

tolerance. In the context of the retail industry, data from different sources can be assigned to a partition where they will be individually processed. The processing is mostly done in parallel as each partition is stored on a different broker. [4]

In conclusion, when designing a scalable and efficient big data solution for retail, it's important to prioritise the storage, processing speed and integration with different sources due to the high amount of data being generated and collected. An effective solution would be a distributed computing system for storage and processing speed where data are distributed across different nodes or machines to be processed or analysed. This helps with the efficiency and scalability of data analytics.

4.0. Dataset Introduction

Table 1. Dataset.

Dataset Title	Online Sales Dataset
Dataset Link	https://www.kaggle.com/datasets/yusufdelikkaya/online-sales-dataset

4.1. Dataset Size and Key Features Description

The dataset introduced a transactional history of customers, so analysis is made to better understand consumer behaviour, sales strategies optimization, and trends prediction.

The dataset has some vital key features for customer behaviour and sales pattern analysis. Describing these key features in the following table:

Table 2. Dataset size and key.

<table>
<tr><td>Number of rows / records</td><td colspan="3">The dataset contains total of 49783 transaction records</td></tr>
<tr><td rowspan="6">Attributes</td><td colspan="3">The dataset contains a total of 17 different variables (attributes).</td></tr>
<tr><td>Key Features</td><td>Description</td><td>Data Type</td></tr>
<tr><td>InvoiceNo</td><td>This is a unique identifier for each transaction</td><td>Integer</td></tr>
<tr><td>StockCode</td><td>Its representing stock-keeping unit (SKU) of the product</td><td>String</td></tr>
<tr><td>Description</td><td>The name/description of the product</td><td>String</td></tr>
<tr><td>Quantity</td><td>How many units of each product in the transaction</td><td>Integer</td></tr>
</table>

 doi:10.20944/preprints202512.0675.v1

Columns	*InvoiceDate*	The date and time of the created transaction	**String**
	UnitPrice	The price of each sold unit in the transaction currency	**Float**
	CustomerID	Each customer has a unique identifier	**Float**
	Country	The country of customer	**String**
	Discount	If any discount applied on the transaction	**Float**
	Payment Method	What is used to pay for the transaction (e.g., PayPal, Bank Transfer)	**String**
	ShippingCost	The cost for shipping the products.	**Float**
	Category	Which category the product falls under (e.g., Apparel, Electronics)	**String**
	SalesChannel	The sales channel (online, in-store, etc.) that was used to make the purchase	**String**
	ReturnStatus	Whether the product was returned or not	**String**
	ShipmentProvider	Who is shipping the order to the customer (e.g., UPS, FedEx)	**String**
	WarehouseLocation	The location where the purchased item was sent from	**String**
	OrderPriority	Determine the priority of the order whether its high, medium or low	**String**

4.2. How is the Dataset Useful for the Given Problem

With such comprehensive structure and extensive records, this dataset is highly suitable for conducting detailed analysis in the domain of retail and e-commerce. There are 49,783 rows each representing a single transaction, hence offering rich sources of data for insight and decision making. Each transaction has 17 varied attributes that describe customer transactions, product details, price, discount, payment methods and many more. Some of the key features in this dataset are InvoiceNo for unique identification for each transaction and StockCode to identify the product SKU, which enables one to keep proper track of various products and orders. Other features like InvoiceDate and Country allow for temporal and geographical analyses that show what was trending where and during which period. This facilitates quantity times UnitPrice for a more precise sales calculation for revenue, while fields such as Category, PaymentMethod, and

SalesChannel make the dataset rich for segmenting categorical data and further detailing customer behavior and dynamic relationships within an operation.

The dataset also contains variables such as Discount, ShippingCost and OrderPriority, which are crucial in assessing business processes like pricing strategies, logistical efficiency and priorities regarding order fulfillment. Moreover, attributes such as ReturnStatus and ShipmentProvider increase the applicability of the dataset in understanding customer satisfaction and supply chain performance. It gives enough granularity and diversity for all facets of retail operations: everything

 doi:10.20944/preprints202512.0675.v1

from sales and revenue trends down to customer preferences and bottlenecks of operations. The predefined structured nature of the data also ensures adaptability for the scalability of big data tools in both exploratory analytics and advanced applications for retail analytics.

4.3. Dataset Sample Table for Visualizing

The following table is a sample of the 2 transaction records (rows) first and last of the dataset presented showing all the 17 attributes/columns of the dataset.

Table 3. Dataset table.

InvoiceNo	221958	…	772215
StockCode	SKU_1964	…	SKU_1832
Description	White Mug	…	White Mug
Quantity	38	…	30
InvoiceDate	2020-01-01 00:00	…	9/5/2025 5:00:00
UnitPrice	1.71	…	38.27
CustomerID	37039.0	…	53328
Country	Australia	…	France
Discount	0.47	…	0.1
PaymentMethod	Bank Transfer	…	Credit Card
ShippingCost	10.79	…	9.13
Category	Apparel	…	Stationery
SalesChannel	In-store	…	Online
ReturnStatus	Not Returned	…	Not Returned
ShipmentProvider	UPS	…	UPS
WarehouseLocation	London	…	Rome
OrderPriority	Medium	…	Low

5.0. Analysis Using Big Data Technologies

Employment of Hive, Pig and Impala

In this section, an implementation of these three big data technologies (Hive, Kafka and Spark) will be addressed and described, doing analysis using them on the dataset introduced before and visualizing the results. MapReduce tools: HIVE, PIG, IMPALA will be applied and a comparison between them will be done according to: 1. Time Taken for executing the queries, 2. virtual memory utilized and 3. Scalability.

MapReduce is both a programming model and a processing technique for processing huge blocks of data in a distributed way. It was developed by Google and now plays an important role in

 doi:10.20944/preprints202512.0675.v1

many big data frameworks, including Hadoop. This model, in general, breaks down the work into two major phases: the Map phase and the Reduce phase (Khader, & Al-Naymat., 2020).

According to Khader, & Al-Naymat (2020), In the Map phase, the input data is divided into smaller fractions called 'splits'. Each fragment is processed by one instance of a Mapper function that performs parallel processing. This will read data, process the data, and emit out a set of key-value pairs (intermediate). The above key-value pairs will, in fact, denote the output result of a Map phase normally used for representing any type of transformed data. The examples could be word count or aggregate value.

Next came the phase of Shuffle and Sort where the framework does the grouping of generated intermediate key-value pairs based on keys so that all-values with some particular key reach the same Reducer for further reduction. Sorting also plays an essential part in keeping the data proper for efficient reduction (Lin, J., & Dyer., 2022).

During the Reduce phase, it takes every key with a list of values associated and processes them, emitting final output. In most cases, this means aggregation or summary of the data in some sort of way: computation of the sum, average, or count of values associated with each key. Then the final results are written into a distributed file system, for example, Hadoop's HDFS.

Key elements in the MapReduce model: the Mapper performs the Map phase, hence emitting key-value pairs; the Reducer performs the Reduce phase of the model, hence aggregate data; Combiner represents a local reducer that has to do with performance optimization by aggregating data before transmitting them to the Reducer-though this step is optional. Data is read and written to distributed storage to enable the effective processing of big volumes of data (Jankatti, S., Raghavendra, Raghavendra, & Meenakshi, M., 2020).

With reference to Jankatti, S., Raghavendra, Raghavendra, & Meenakshi, M (2020), for instance, if you wanted to count the occurrences of words in a large text file, the Map function would process each block of text and emit a list of key-value pairs, where the key is the word and the value is 1 representing the occurrence. The Shuffle and Sort phase would group by word, and in the Reduce phase, the Reducer does the sum of values for each key into the final count of word occurrences.

One more strong side of MapReduce is its scalability-it can handle big loads because it distributes work between lots of machines. As mentioned by Lin, J., & Dyer (2022), another plus is fault tolerance: if a node has failed during processing, task rescheduling can be forwarded to other nodes. Besides, it is a very simple model to understand, abstracting a great amount of the complexity involved in parallel computing. However, it is far from the best choice to address every workload, especially where low-latency processing is required or iterative computations are involved.

The most common use cases for MapReduce include data aggregation, such as word count and summing values, data transformation, and sorting large datasets. Though it is a powerful tool for distributed computing, more advanced processing, such as iterative algorithms, is often done with newer technologies like Apache Spark due to their higher performance in some scenarios (Lin, J., & Dyer., 2022).

5.1. Shared Folder Creation

Creating a shared folder between Windows (Host) and Cloudera (Virtual Machine) is meant to simplify the process of sharing files between Windows and Cloudera. Then we can start working in Cloudera and use the big data technologies chosen for analyzing the dataset, along with managing and processing it.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 December 2025 doi:10.20944/preprints202512.0675.v1

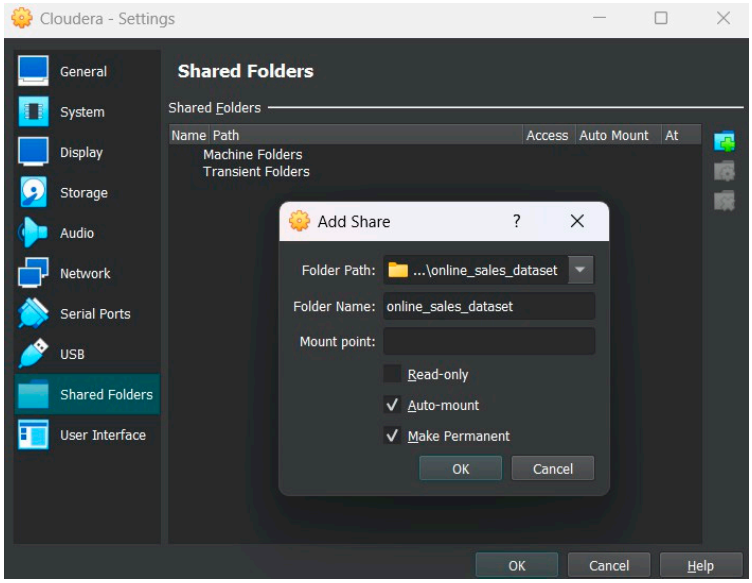


Figure 4. Creating a shared folder.

In this step, we are going to create a shared folder in the **"Machine Folders"** with the name "**online_sales_dataset**" and specify where the location of the folder path is in Windows. Both "**Auto-mount**" and "**Make Permanent**" should be included to avoid mounting the folder every time we reboot the machine and making it permanent to not create a new folder every time.

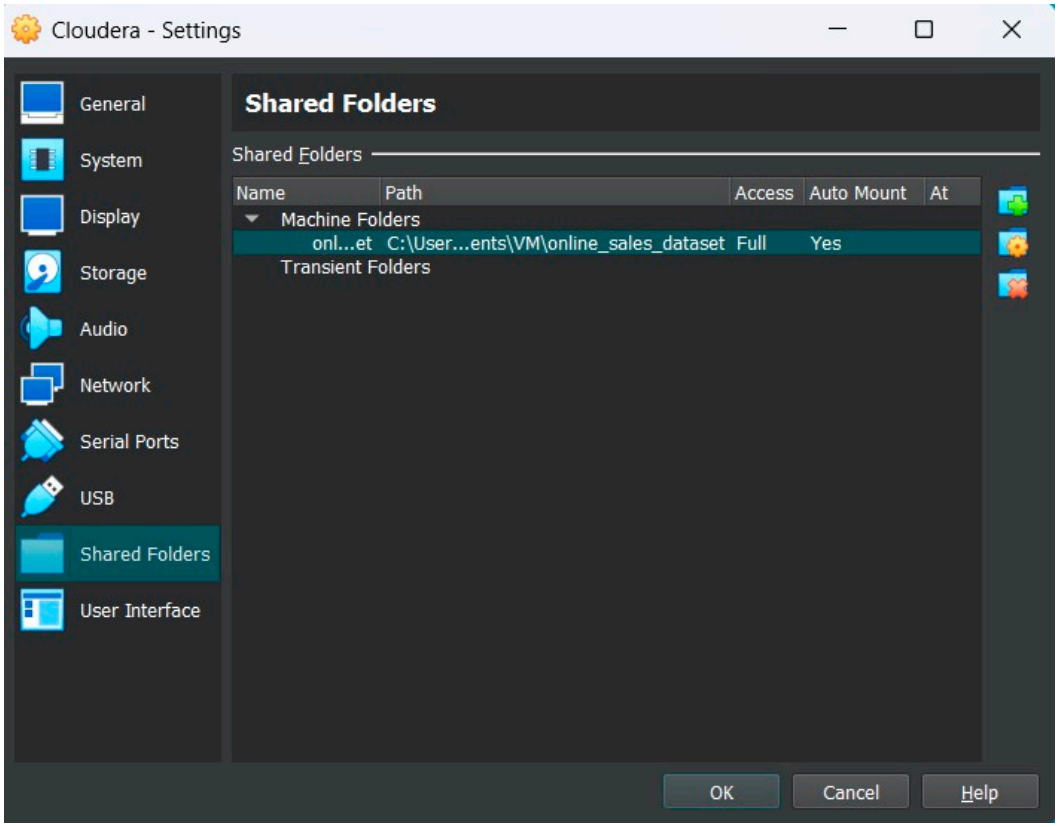


Figure 5. the "online_sales_dataset" shared folder is created but 1 more step left to do.

 doi:10.20944/preprints202512.0675.v1

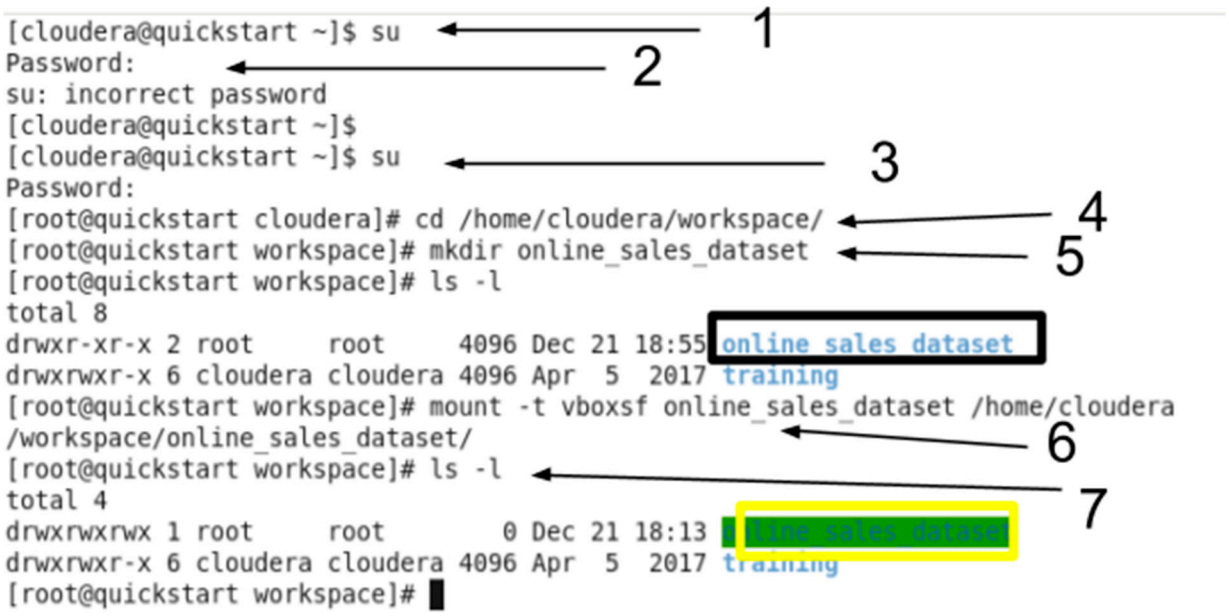


Figure 6. Mounting the shared folder.

In this step, we should mount the folder after starting Cloudera VM and open the terminal.

1. "**su**" switch user. We login in the terminal as root which has different privileges to mount the desired folder.
2. The password is "**cloudera**".
3. After logging in as the root account, we want to change the directory to this directory "**/home/cloudera/workspace/**" to add the shared folder.
4. We made a directory under the name "**online_sales_dataset**".
5. We want to list all the directories in "**workspace**" by using "**ls**"
6. After checking that the directory is created, we want to mount the folder "**online_sales_dataset**" into this path "**/home/cloudera/workspace/online_sales_dataset**" to access the shared folder.
7. We want to check if the directory is mounted, so we list the available directories and we see the directory with green highlight in the cloudera terminal which means it's successfully mounted. Now we can start using it to import the dataset.

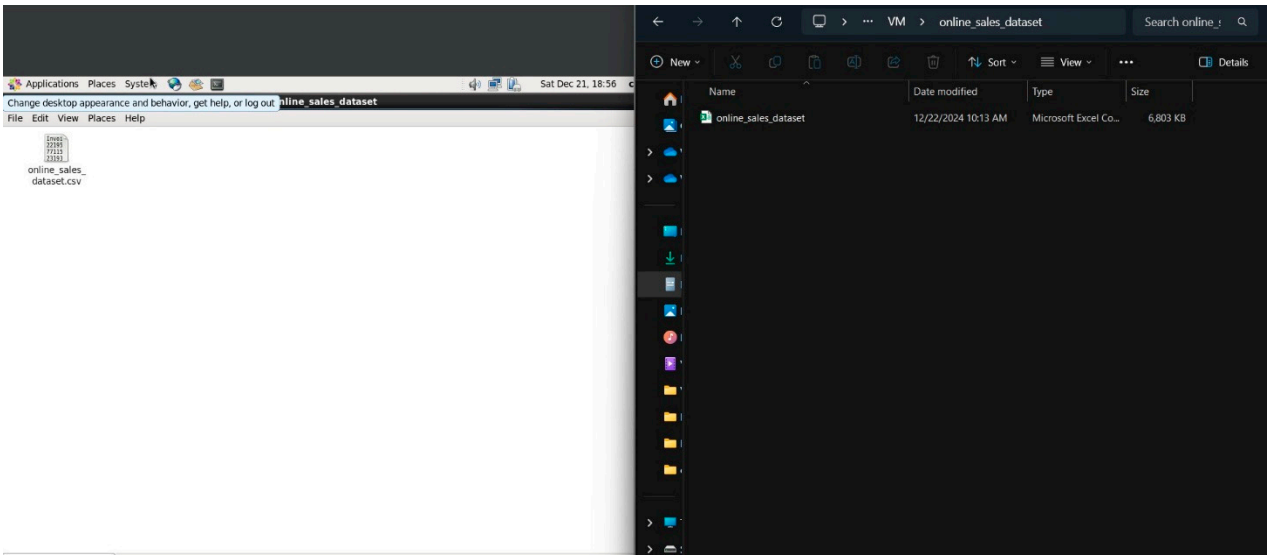


Figure 7. Same folder and file in Windows host and Cloudera Virtual Machine.

 doi:10.20944/preprints202512.0675.v1

It shows the shared folder in Windows host and Cloudera VM under the same name and have the same file which is the dataset. We can simply add any file in this folder and access it in either Windows host and Cloudera VM.

```
[cloudera@quickstart ~]$ head /home/cloudera/workspace/online_sales_dataset/online_sales_dataset.csv
InvoiceNo,StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country,Discount,PaymentMethod,ShippingCost,Categor
y,SalesChannel,ReturnStatus,ShipmentProvider,WarehouseLocation,OrderPriority
221958,SKU_1964,White Mug,38,2020-01-01 00:00,1.71,37039.0,Australia,0.47,Bank Transfer,10.79,Apparel,In-store,Not Returned,U
PS,London,Medium
771155,SKU_1241,White Mug,18,2020-01-01 01:00,41.25,19144.0,Spain,0.19,paypall,9.51,Electronics,Online,Not Returned,UPS,Rome,
Medium
231932,SKU_1501,Headphones,49,2020-01-01 02:00,29.11,50472.0,Germany,0.35,Bank Transfer,23.03,Electronics,Online,Returned,UPS
,Berlin,High
465838,SKU_1760,Desk Lamp,14,2020-01-01 03:00,76.68,96586.0,Netherlands,0.14,paypall,11.08,Accessories,Online,Not Returned,Ro
yal Mail,Rome,Low
359178,SKU_1386,USB Cable,-30,2020-01-01 04:00,-68.11,,United Kingdom,1.501433043360303,Bank Transfer,,Electronics,In-store,N
ot Returned,FedEx,,Medium
744167,SKU_1006,Office Chair,47,2020-01-01 05:00,70.16,53887.0,Sweden,0.48,Credit Card,13.98,Electronics,Online,Not Returned,
DHL,London,Medium
210268,SKU_1087,USB Cable,25,2020-01-01 06:00,85.74,46567.0,Belgium,0.15,Bank Transfer,12.92,Stationery,Online,Not Returned,F
edEx,Amsterdam,High
832180,SKU_1597,Notebook,8,2020-01-01 07:00,95.65,75098.0,Norway,0.04,Bank Transfer,6.48,Electronics,In-store,Not Returned,Ro
yal Mail,Amsterdam,Low
154886,SKU_1907,Wireless Mouse,19,2020-01-01 08:00,98.19,87950.0,Belgium,0.05,paypall,12.56,Apparel,Online,Not Returned,UPS,B
erlin,High
[cloudera@quickstart ~]$
```

Figure 8. We displayed the first 10 lines of the dataset to check the dataset file is working as supposed.

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir -p /user/hive/warehouse/online_sales_dataset/
[cloudera@quickstart ~]$ hdfs dfs -put /home/cloudera/workspace/online_sales_dataset/online_sales_dataset.csv /user/hive/ware
house/online_sales_dataset/
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse/online_sales_dataset/
Found 1 items
-rw-r--r--   1 cloudera supergroup    6965414 2024-12-21 19:02 /user/hive/warehouse/online_sales_dataset/online_sales_dataset
.csv
[cloudera@quickstart ~]$
```

Figure 9. Directory Creation in HDFS.

We are making directory in hadoop HDFS named "**online_sales_dataset**" in "/user"/hive/warehouse/", and the put the dataset file "**/home/cloudera/workspace/online_retail/online_sales_dataset.csv**" in the new directory in HDFS "**/user/hive/warehouse/online_sales_dataset/**". Then we check the contents "ls" and we can verify the new directory created and the dataset file is in there.

5.2. Hive Employment

In this section, we'll be implementing/employing Hive technology to derive insights using the dataset and SQL-like queries and compare the performance between this technology and other 2 technologies.

 doi:10.20944/preprints202512.0675.v1


```
[cloudera@quickstart ~]$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> CREATE TABLE online_sales_dataset (
    >     InvoiceNo INT,
    >     StockCode STRING,
    >     Description STRING,
    >     Quantity INT,
    >     InvoiceDate STRING,
    >     UnitPrice FLOAT,
    >     CustomerID FLOAT,
    >     Country STRING,
    >     Discount FLOAT,
    >     PaymentMethod STRING,
    >     ShippingCost FLOAT,
    >     Category STRING,
    >     SalesChannel STRING,
    >     ReturnStatus STRING,
    >     ShipmentProvider STRING,
    >     WarehouseLocation STRING,
    >     OrderPriority STRING
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.783 seconds
hive>
```

Figure 10. Hive login.

First we log in to hive using cloudera terminal to start importing the dataset to it, after the login we want to CREATE TABLE with the name "**online_sales_dataset**" to hold the dataset values in it and it has the following attributes and their data types: **InvoiceNo INT, StockCode STRING, Description STRING, Quantity INT InvoiceDate STRING, UnitPrice FLOAT, CustomerID FLOAT, Country STRING, Discount FLOAT, PaymentMethod STRING, ShippingCost FLOAT, Category STRING, SalesChannel STRING, ReturnStatus STRING, ShipmentProvider STRING, WarehouseLocation STRING, OrderPriority STRING.**

With Apache Hive's "row format delimited" functionality, fields and lines in a table are terminated using delimiters.

```
hive> LOAD DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sales_data
set.csv'
    > INTO TABLE online_sales_dataset;
Loading data to table default.online_sales_dataset
Table default.online_sales_dataset stats: [numFiles=1, numRows=0, totalSize=6965414
, rawDataSize=0]
OK
Time taken: 0.622 seconds
hive>
```

Figure 11. Load dataset into the table.

In this step, and after creating the table, we want to LOAD the dataset from where the dataset was stored in this path: "**/user/hive/warehouse/online_sales_dataset**". This will load the dataset into the newly created table in figure : .

 doi:10.20944/preprints202512.0675.v1


```
hive> SELECT * FROM online_sales_dataset LIMIT 20;
OK
NULL    StockCode       Description     NULL    InvoiceDate     NULL    NULL    Cou
ntry    NULL    PaymentMethod   NULL    Category        SalesChannel    ReturnStatu
s       ShipmentProvider        WarehouseLocation       OrderPriority
221958  SKU_1964        White Mug       38      2020-01-01 00:00        1.71    370
39.0    Australia       0.47    Bank Transfer   10.79   Apparel In-store        Not
 Returned       UPS     London  Medium
771155  SKU_1241        White Mug       18      2020-01-01 01:00        41.25   191
44.0    Spain   0.19    paypall 9.51    Electronics     Online  Not Returned    UPS
Rome    Medium
231932  SKU_1501        Headphones      49      2020-01-01 02:00        29.11   504
72.0    Germany 0.35    Bank Transfer   23.03   Electronics     Online  Returned  U
PS      Berlin  High
465838  SKU_1760        Desk Lamp       14      2020-01-01 03:00        76.68   965
86.0    Netherlands     0.14    paypall 11.08   Accessories     Online  Not Returne
d       Royal Mail      Rome    Low
359178  SKU_1386        USB Cable       -30     2020-01-01 04:00        -68.11  NUL
L       United Kingdom  1.501433        Bank Transfer   NULL    Electronics     In-
store   Not Returned    FedEx           Medium
744167  SKU_1006        Office Chair    47      2020-01-01 05:00        70.16   538
87.0    Sweden  0.48    Credit Card     13.98   Electronics     Online  Not Returne
d       DHL     London  Medium
210268  SKU_1087        USB Cable       25      2020-01-01 06:00        85.74   465
67.0    Belgium 0.15    Bank Transfer   12.92   Stationery      Online  Not Returne
d       FedEx   Amsterdam       High
832180  SKU_1597        Notebook        8       2020-01-01 07:00        95.65   750
98.0    Norway  0.04    Bank Transfer   6.48    Electronics     In-store        Not
```

Figure 12. First 20 lines query.

This query is to display the first 20 lines from the table "**online_sales_dataset**" to check that the table was created and the dataset was loaded without any issues.

The following three metrics: time taken for the query to be executed and giving a valid output, the memory utilized while executing the query and the last is the scalability which will figure out how the upscaled dataset by **x2** will affect the time taken and the virtual memory utilized.

 doi:10.20944/preprints202512.0675.v1

5.2.1. Time Taken

```
hive> SELECT StockCode, SUM(Quantity) AS TotalQuantity
    > FROM online_sales_dataset
    > GROUP BY StockCode
    > ORDER BY TotalQuantity DESC
    > LIMIT 10;
Query ID = cloudera_20241221191313_70fc040a-92d1-4e10-b170-934d6bd00675
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1734835782760_0002, Tracking URL = http://quickstart.cloudera:80
88/proxy/application_1734835782760_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1734835782760_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-12-21 19:13:39,896 Stage-1 map = 0%,  reduce = 0%
2024-12-21 19:13:45,074 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 sec
2024-12-21 19:13:50,258 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.69 sec
MapReduce Total cumulative CPU time: 2 seconds 690 msec
Ended Job = job_1734835782760_0002
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
```

Figure 13. Top 10 Products by Quantity Sold QUERY.

In this figure, we wanted to display the results of the top 10 products by the quantity sold and the time taken to finish this query.

```
Ended Job = job_1734835782760_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.9 sec   HDFS Read: 6973558 HDFS Write: 29403 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.24 sec   HDFS Read: 34296 HDFS Write: 140 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 140 msec
OK
SKU_1044        1699
SKU_1198        1678
SKU_1218        1668
SKU_1213        1665
SKU_1555        1659
SKU_1587        1654
SKU_1761        1650
SKU_1148        1650
SKU_1726        1621
SKU_1938        1615
Time taken: 39.911 seconds, Fetched: 10 row(s)
```

Figure 14. Top 10 Products by Quantity Sold OUTPUT.

After executing the query, we got the output as expected which has the top 10 products sorted by their code and the sum of the quantity. The time taken in total to fetch 10 rows was **39.911 seconds**.

5.2.2. Virtual Memory Utilization

A reminder of the assigned random access memory to the virtual machine was 6,144 mb.

 doi:10.20944/preprints202512.0675.v1


```
[cloudera@quickstart ~]$ vmstat 2
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0 137092 784184   5080 127088    2   11   258   156  444  974  2  2 95  1  0
 1  0 137092 784132   5080 127088    0    0     0    48 1302 3994  0  1 99  0  0
 2  0 137092 784132   5080 127088    0    0     0     0  848 2353  1  1 99  0  0
 0  0 137092 784180   5088 127088    0    0     0    34  805 2284  0  1 99  0  0
 1  0 137092 784088   5088 127088    0    0     0     0  910 2346  0  0 100  0  0
 3  0 137092 783972   5088 127088    0    0     0     6  870 2333  0  1 99  0  0
 1  0 137092 783988   5096 127088    0    0     0    30  851 2323  1  1 99  0  0
 1  0 137092 796960   5104 127080    0    0     8    26  915 2391  1  1 99  0  0
 1  0 137092 796836   5104 127088    0    0     0   110  836 2231  1  0 99  0  0
 1  0 137092 796836   5104 127088    0    0     0     0  817 2253  0  1 99  0  0
 0  0 137092 796712   5104 127088    0    0     0     6  812 2303  0  0 99  0  0
 0  0 137092 796728   5112 127088    0    0     0    38  842 2275  0  0 99  0  0
 0  0 137092 796604   5144 127080    0    0     0    96  830 2196  0  1 99  0  0
 0  0 137092 796604   5144 127080    0    0     0    10  734 2182  1  1 99  0  0
 1  0 137092 796596   5144 127080    0    0     0     0  882 2318  0  1 99  0  0
 6  0 137092 796472   5152 127076    0    0     0    42  850 2320  0  1 98  0  0
29  0 137092 796308   5152 127080    0    0     0    18  859 2269  0  0 99  0  0
32  0 137092 796308   5152 127080    0    0     0     0  838 2242  0  1 99  0  0
11  0 137092 796308   5160 127080    0    0     0    58  682 2041  0  0 99  0  0
10  0 137092 796324   5160 127080    0    0     0     0  891 2352  1  0 99  0  0
 2  0 137092 796168   5160 127080    0    0     0     6  860 2250  1  1 98  0  0
```

Figure 15. Virtual Memory Utilization BEFORE running any query.

In the previous figure, we can observe the free memory available in the virtual machine before running any query, the avg of the free memory was around **790,382 kb**

```
Ended Job = job_1734835782760_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.9 sec   HDFS Read: 6973558 HDFS Write: 29403 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.24 sec   HDFS Read: 34296 HDFS Write: 140 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 140 msec
OK
SKU_1044        1699
SKU_1198        1678
SKU_1218        1668
SKU_1213        1665
SKU_1555        1659
SKU_1587        1654
SKU_1761        1650
SKU_1148        1650
SKU_1726        1621
SKU_1938        1615
Time taken: 39.911 seconds, Fetched: 10 row(s)
hive>
```

```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
 6  1      0  97588   2324 209756    0    0    22 36128 3862 2946 28 25 45  2  0
 1  0      0 401548    568  90848    0    0  8478   330 3797 3693 14 11 48 27  0
 0  1      0 380084    592  97392    0    0  3150   146 1843 3334  3  3 89  5  0
 5  0      0 279324    596 112680    0    0  7678     0 3452 4062 20 12 56 12  0
 2  0      0 167360    608 118608    0    0  2836    22 2677 3225 21 10 67  3  0
 2  0      0 375092    620 119192    0    0   206   140 1767 2421  5  4 91  1  0
 2  1      0 316944    724 181108    0    0 13508   106 2931 5657  6  7 83  4  0
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
11  0      0 162708    996 216736    0    0   170     8 4037 4017 21 12 67  1  0
 6  1      0 376164    872 177380    0    0  2910   172 4063 3483 23 14 45 18  0
 2  2      0 309528    904 172260    0    0   996    20 2245 3348  8  6 56 30  0
 6  1      0 179784    812 169004    0    0   130   144 3379 2808 26 14 44 16  0
 0  0      0 415328    616  72708    0    0  1860 36318 3874 3527 12 18 41 29  0
 7  0      0 417692    616  73232    0    0   310     0  982 2426  1  1 98  1  0
 2  0      0 310336    796  89884    0    0  8436   170 3670 4670 18 12 53 16  0
11  0      0 176088    796  95412    0    0  2480    32 2969 3909 25 10 63  2  0
 9  0      0 398164    824  96528    0    0   602   152 2038 3767  3  4 92  1  0
```

Figure 16. Top 10 Products by Quantity Sold Virtual Memory Utilization while running the query.

The free memory available when we ran the query achieved the lowest at **97,588 kb.** Which is clearly noticeable by hive since it's considered to be using the most memory.

 doi:10.20944/preprints202512.0675.v1

5.2.3. Scalability

In this metric, what we are going to do is see how scaling up the dataset to a bigger size will affect the time taken to execute queries and memory utilization while execution.

```
[cloudera@quickstart ~]$ hdfs dfs -cp /user/hive/warehouse/online_sales_dataset/online_sales_dataset.csv /user/hive/warehouse
/online_sales_dataset/online_sales_dataset_large.csv
[cloudera@quickstart ~]$
```

Figure 17. Upscaling the dataset by x2.

By copying the dataset to itself and saving the bigger dataset under the name "**online_sales_dataset_large**" in HDFS.

```
hive>
    > LOAD DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sales_dataset_large.csv'
    > INTO TABLE online_sales_dataset;
Loading data to table default.online_sales_dataset
Table default.online_sales_dataset stats: [numFiles=2, numRows=0, totalSize=13930828, rawDataSize=0]
OK
Time taken: 1.239 seconds
```

Figure 18. Load the larger dataset into Hive.

What is done here is loading the upscaled dataset which was saved under the name "online_sales_dataset_large" in the path '**/user/hive/warehouse/online_sales_dataset/online_sales_dataset_large.csv**' INTO the same table used before which holds the previous dataset.

After that, we will rerun the same query and compare execution time and memory usage for the larger dataset.

1. **Time Taken**

```
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.27 sec   HDFS Read: 34296 HDFS Write: 140 SUCCE
SS
Total MapReduce CPU Time Spent: 5 seconds 470 msec
OK
SKU_1044        3398
SKU_1198        3356
SKU_1218        3336
SKU_1213        3330
SKU_1555        3318
SKU_1587        3308
SKU_1761        3300
SKU_1148        3300
SKU_1726        3242
SKU_1938        3230
Time taken: 43.324 seconds, Fetched: 10 row(s)
```

Figure 19. Upscaled dataset time taken and memory utilization.

As we can see in the previous figure, the time taken has increased from **39.911** seconds to **43.324** seconds.

2. **Virtual Memory Utilization:**

 doi:10.20944/preprints202512.0675.v1


```
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.27 sec   HDFS Read: 34296 HDFS Write: 140 SUCCE
SS
Total MapReduce CPU Time Spent: 5 seconds 470 msec
OK
SKU_1044        3398
SKU_1198        3356
SKU_1218        3336
SKU_1213        3330
SKU_1555        3318
SKU_1587        3308
SKU_1761        3300
SKU_1148        3300
SKU_1726        3242
SKU_1938        3230
Time taken: 43.324 seconds, Fetched: 10 row(s)
hive>
```

```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ vmstat 2
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0 528076 643448   1952 112988    3   22   123   104  380  894  1  1 97  1  0
 1  0 528076 568900   1968 183108    0    0 13462   128 3087 7617  6  8 84  2  0
 1  0 528072 387680   2088 212216    0    0   792     0 3624 3870 22 13 64  1  0
 1  0 528024 234448   2116 215784   48    0  1728    32 2845 3488 17  9 72  2  0
 3  0 528016 161620   2128 216352    0    0   256   188 2183 3513 10  4 86  0  0
 2  0 528012  93768   1204 148808    0    0   146 36374 5033 3695 31 26 34  9  0
 0  5 527712 401828    804  31112  284    0 12408   412 4747 3737 13 15 36 35  0
 1  0 527684 396604    860  36296   16    0  2666     4 1237 2660  1  2 94  4  0
 2  1 527664 309844   1028  72344   16    0 18024     0 3357 5076  7  7 68 18  0
 1  0 527660 226552   1196  91480    0    0  9558   128 3856 4469 29 13 46 13  0
 0  0 527660 125396   1204  99852    0    0  4052    72 3466 4771 19 11 63  7  0
 3  0 527656 307872   1376 124504   16    0  4524   156 1954 4341  2  4 91  3  0
 9  0 527652 269280   1376 163796   48    0 10006    42 1996 3658  3  4 90  4  0
 2  1 527632  93380   1136 197912    0    0  1222   146 3709 3885 20 13 59  7  0
 0  2 527632 370688    956 143356    0    0  4716 21980 4744 4216 16 20 48 16  0
```

Figure 20. Virtual memory utilization while running Top 10 products by quantity sold on the larger dataset.

As expected, hive is using more memory in the upscaled dataset than the original dataset before while executing the query to get the top 10 products, free memory reaching the edge be **93,380 kb.**

5.3. IMPALA Employment

```
[cloudera@quickstart ~]$ su
Password:
[root@quickstart cloudera]# impala-shell
Starting Impala Shell without Kerberos authentication
Connected to quickstart.cloudera:21000
Server version: impalad version 2.7.0-cdh5.10.0 RELEASE (build 785a073cd07e2540d
521ecebb8b38161ccbd2aa2)
***************************************************************************
***
Welcome to the Impala shell.
(Impala Shell v2.7.0-cdh5.10.0 (785a073) built on Fri Jan 20 12:03:56 PST 2017)

When pretty-printing is disabled, you can use the '--output_delimiter' flag to s
et
the delimiter for fields in the same row. The default is ','.
***************************************************************************
***
[quickstart.cloudera:21000] > SHOW DATABASES;
Query: show DATABASES
+------------------+----------------------------------------------+
| name             | comment                                      |
+------------------+----------------------------------------------+
| _impala_builtins | System database for Impala builtin functions |
| default          | Default Hive database                        |
+------------------+----------------------------------------------+
Fetched 2 row(s) in 0.25s
[quickstart.cloudera:21000] >
```

Figure 21. login to impala shell.

 doi:10.20944/preprints202512.0675.v1

In the previous figure, we started the cloudera terminal and login as root, after that we used "**impala-shell**" command to access the shell and start employing this technology on the dataset we have. The available databases have been displayed using "**SHOW DATABASES**" and we clearly see only 2 databases available.

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse/online_sales_dataset/
Found 1 items
-rw-r--r--   1 cloudera supergroup    6965414 2024-12-22 11:55 /user/hive/warehouse/online_sales_dat
aset/online_sales_dataset.csv
[cloudera@quickstart ~]$
```

Figure 22. location of the dataset in HDFS.

In this figure, we already have the dataset imported to HDFS and there is no need to put it again from the shared folder.

```
[quickstart.cloudera:21000] > CREATE DATABASE online_sales LOCATION '/user/hive/warehouse/online_sales_dataset';
Query: create DATABASE online_sales LOCATION '/user/hive/warehouse/online_sales_dataset'

Fetched 0 row(s) in 0.39s
[quickstart.cloudera:21000] > SHOW DATABASES;
Query: show DATABASES
+------------------+--------------------------------------------+
| name             | comment                                    |
+------------------+--------------------------------------------+
| _impala_builtins | System database for Impala builtin functions |
| default          | Default Hive database                      |
| online_sales     |                                            |
+------------------+--------------------------------------------+
Fetched 3 row(s) in 0.11s
[quickstart.cloudera:21000] > USE online_sales;
Query: use online_sales
```

Figure 23. Showing available databases.

In this step, we created a new database called "online_sales" and the location of it was in this path '/user/hive/warehouse/online_sales_dataset', displayed the available databases and put the newly created one to be used by the command "**USE online_sales**".

```
Query: create TABLE online_sales.online_sales_dataset (
    InvoiceNo INT,
    StockCode STRING,
    Description STRING,
    Quantity INT,
    InvoiceDate STRING,
    UnitPrice FLOAT,
    CustomerID FLOAT,
    Country STRING,
    Discount FLOAT,
    PaymentMethod STRING,
    ShippingCost FLOAT,
    Category STRING,
    SalesChannel STRING,
    ReturnStatus STRING,
    ShipmentProvider STRING,
    WarehouseLocation STRING,
    OrderPriority STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE

Fetched 0 row(s) in 0.45s
```

Figure 24. Table creation in the database.

In this step, and after creating the database, we created the table "online_sales_dataset" in the database "online_sales".

 doi:10.20944/preprints202512.0675.v1


```
[quickstart.cloudera:21000] > SHOW TABLES;
Query: show TABLES
+----------------------+
| name                 |
+----------------------+
| online_sales_dataset |
+----------------------+
Fetched 1 row(s) in 0.02s
[quickstart.cloudera:21000] > DESCRIBE online_sales_dataset;
Query: describe online_sales_dataset
+-------------------+--------+---------+
| name              | type   | comment |
+-------------------+--------+---------+
| invoiceno         | int    |         |
| stockcode         | string |         |
| description       | string |         |
| quantity          | int    |         |
| invoicedate       | string |         |
| unitprice         | float  |         |
| customerid        | float  |         |
| country           | string |         |
| discount          | float  |         |
| paymentmethod     | string |         |
| shippingcost      | float  |         |
| category          | string |         |
| saleschannel      | string |         |
| returnstatus      | string |         |
| shipmentprovider  | string |         |
| warehouselocation | string |         |
| orderpriority     | string |         |
+-------------------+--------+---------+
Fetched 17 row(s) in 4.70s
```

Figure 25. Displaying the available tables in th database.

In this figure, there are 2 commands, one to show the available tables created, and to describe an available table and in our case it's "online_sales_dataset" which means verifying the attributes and data types of the table.

```
[quickstart.cloudera:21000] > LOAD DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sal
es_dataset.csv' INTO TABLE online_sales.online_sales_dataset;
Query: load DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sales_dataset.csv' INTO TA
BLE online_sales.online_sales_dataset
ERROR: AnalysisException: Unable to LOAD DATA from hdfs://quickstart.cloudera:8020/user/hive/warehou
se/online_sales_dataset/online_sales_dataset.csv because Impala does not have WRITE permissions on i
ts parent directory hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_sales_dataset

[quickstart.cloudera:21000] >
```

```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ hdfs dfs -chmod 777 /user/hive/warehouse/online_sales_dataset
[cloudera@quickstart ~]$ hdfs dfs -ls -d /user/hive/warehouse/online_sales_dataset
drwxrwxrwx   - cloudera supergroup          0 2024-12-23 15:42 /user/hive/warehouse/online_sales_dat
aset
```

Figure 26. Load dataset into the table.

In this step, we tried to load the dataset into the created table but an error showed up requiring impala the permission to be able to write on the parent directory. By using a command in another cloudera terminal 'hdfs dfs -chmod 777 /user/hive/warehouse/online_sales_dataset'. In the next figure we'll be able to load the dataset into the table.

 doi:10.20944/preprints202512.0675.v1


```
[quickstart.cloudera:21000] > LOAD DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sal
es_dataset.csv' INTO TABLE online_sales.online_sales_dataset;
Query: load DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sales_dataset.csv' INTO TA
BLE online_sales.online_sales_dataset
ERROR: AnalysisException: Unable to LOAD DATA from hdfs://quickstart.cloudera:8020/user/hive/warehou
se/online_sales_dataset/online_sales_dataset.csv because Impala does not have WRITE permissions on i
ts parent directory hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_sales_dataset

[quickstart.cloudera:21000] > LOAD DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sal
es_dataset.csv' INTO TABLE online_sales.online_sales_dataset;
Query: load DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sales_dataset.csv' INTO TA
BLE online_sales.online_sales_dataset
+-----------------------------------------------------------+
| summary                                                   |
+-----------------------------------------------------------+
| Loaded 1 file(s). Total files in destination location: 1 |
+-----------------------------------------------------------+
Fetched 1 row(s) in 0.47s
```

Figure 27. Dataset loaded successfully into the table.

5.3.1. Time Taken

```
Query progress can be monitored at: http://quickstart.cloudera:25000/query_plan?query_id=c49f88334a91af0:c97802de00000000
+-----------+---------------+
| stockcode | totalquantity |
+-----------+---------------+
| StockCode | NULL          |
| SKU_1044  | 1699          |
| SKU_1198  | 1678          |
| SKU_1218  | 1668          |
| SKU_1213  | 1665          |
| SKU_1555  | 1659          |
| SKU_1587  | 1654          |
| SKU_1761  | 1650          |
| SKU_1148  | 1650          |
| SKU_1726  | 1621          |
+-----------+---------------+
WARNINGS: Error converting column: 3 to INT
Error parsing row: file: hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_sales.db/online_sales_dataset/online_sale
s_dataset.csv, before offset: 6965414

Fetched 10 row(s) in 0.25s
```

Figure 28. Executing Top 10 Products by Quantity Sold Query.

The time taken in the query using impala technology was **0.25 seconds** to fetch 10 rows.

5.3.2. Virtual Memory Utilization

```
[cloudera@quickstart ~]$ vmstat 2
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 462752  59048 1152916    0    0  1156   480  808 1143  8 10 79  3  0
 0  0      0 462768  59048 1152944    0    0     0     6 1427 4176  2  2 97  0  0
 0  0      0 462496  59056 1152956    0    0     0    58 1578 2651  3  2 94  0  0
 2  0      0 462480  59056 1152956    0    0     0     0  916 2334  1  0 99  0  0
 0  0      0 462448  59056 1152960    0    0     0     0 1066 2457  1  1 99  0  0
 0  0      0 460976  59068 1152964    0    0     0    80 1489 2638  4  2 94  0  0
 2  0      0 460852  59068 1152964    0    0     0     6 1237 3207  2  1 97  0  0
 0  0      0 460836  59076 1152964    0    0     0    84  897 2355  1  1 99  0  0
 0  0      0 460836  59076 1152980    0    0     8     0 1122 2554  1  1 98  0  0
 1  0      0 461984  59084 1152980    0    0     0    38  983 2419  1  1 98  0  0
 0  0      0 462124  59084 1152980    0    0     0    36 1027 2648  1  1 98  0  0
 0  0      0 462108  59084 1152980    0    0     0     6 1185 3125  1  1 98  0  0
```

Figure 29. Virtual Memory Utilization BEFORE running any query.

In the previous figure, we can observe the free memory available in the virtual machine before running any query, the avg of the free memory was around **461,418 kb.**

 doi:10.20944/preprints202512.0675.v1


```
Query progress can be monitored at: http://quickstart.cloudera:25000/query_plan?query_id=c49f88334a91af0:c97802de00000000
+-----------+---------------+
| stockcode | totalquantity |
+-----------+---------------+
| StockCode | NULL          |
| SKU_1044  | 1699          |
| SKU_1198  | 1678          |
| SKU_1218  | 1668          |
| SKU_1213  | 1665          |
| SKU_1555  | 1659          |
| SKU_1587  | 1654          |
| SKU_1761  | 1650          |
| SKU_1148  | 1650          |
| SKU_1726  | 1621          |
+-----------+---------------+
WARNINGS: Error converting column: 3 to INT
Error parsing row: file: hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_sales.db/online_sales_dataset/online_sale
s_dataset.csv, before offset: 6965414

Fetched 10 row(s) in 0.25s
[quickstart.cloudera:21000] >
```

```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ vmstat 2
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 442044  59220 1153296    0    0   799   335  686 1112  6  7 85  2  0
 4  0      0 442060  59220 1153296    0    0     0     0 1158 3183  1  1 98  0  0
 0  0      0 441972  59220 1153308    0    0     0     6 1854 3051  4  3 93  0  0
 2  0      0 442080  59228 1153308    0    0     0    54 1340 3172  1  1 98  0  0
 0  0      0 441320  59228 1153312    0    0     0     0 1149 2777  1  1 97  0  0
 8  0      0 441212  59240 1153312    0    0     0    72 1075 2357  1  1 99  0  0
 1  0      0 441384  59240 1153312    0    0     0     0 1560 4403  2  2 97  0  0
 4  0      0 441524  59240 1153312    0    0     0     6 1126 2445  1  1 99  0  0
 1  0      0 441464  59248 1153312    0    0     0    72 1059 2386  1  1 98  0  0
10  0      0 441496  59248 1153312    0    0     0     0  984 2322  1  1 99  0  0
```

Figure 30. Virtual Memory Utilization While running Top 10 Products by quantity sold.

In the previous figure, we can observe the free memory available in the virtual machine while running Top 10 Products by quantity sold , the avg of the free memory was around 441,708 kb.

5.3.3. Scalability

```
[cloudera@quickstart ~]$ hdfs dfs -cp /user/hive/warehouse/online_sales_dataset/online_sales_dataset/online_sales_dataset.csv
 /user/hive/warehouse/online_sales_dataset/online_sales_dataset/online_sales_dataset_large.csv
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse/online_sales_dataset/online_sales_dataset/
Found 2 items
-rw-r--r--   1 cloudera supergroup    6965414 2024-12-23 15:31 /user/hive/warehouse/online_sales_dataset/online_sales_dataset
/online_sales_dataset.csv
-rw-r--r--   1 cloudera supergroup    6965414 2024-12-24 06:51 /user/hive/warehouse/online_sales_dataset/online_sales_dataset
/online sales dataset large.csv
```

Figure 31. Dataset file duplication.

The dataset file used in previous query, will be copied to the same directory to have larger dataset, so the original dataset has **49,783** rows and the larger dataset will have **x2** the number or the rows in the original one.

```
[quickstart.cloudera:21000] > LOAD DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sales_dataset/online_sales_d
ataset_large.csv' INTO TABLE online_sales.online_sales_dataset;
Query: load DATA INPATH '/user/hive/warehouse/online_sales_dataset/online_sales_dataset/online_sales_dataset_large.csv' INTO
TABLE online_sales.online_sales_dataset
+----------------------------------------------------------+
| summary                                                  |
+----------------------------------------------------------+
| Loaded 1 file(s). Total files in destination location: 2 |
+----------------------------------------------------------+
Fetched 1 row(s) in 0.29s
[quickstart.cloudera:21000] >
```

Figure 32. Loading upscaled dataset into the table.

This step, as previous steps, is to simply load the larger dataset into the table in “online_sales” database.

1. **Time Taken**

 doi:10.20944/preprints202512.0675.v1


```
+-----------+---------------+
| StockCode | NULL          |
| SKU_1044  | 3398          |
| SKU_1198  | 3356          |
| SKU_1218  | 3336          |
| SKU_1213  | 3330          |
| SKU_1555  | 3318          |
| SKU_1587  | 3308          |
| SKU_1761  | 3300          |
| SKU_1148  | 3300          |
| SKU_1726  | 3242          |
+-----------+---------------+
WARNINGS: Error converting column: 3 to INT
Error parsing row: file: hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_sales.db/online_sales_dataset/online_sale
s_dataset.csv, before offset: 6965414
Error converting column: 3 to INT
Error parsing row: file: hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_sales.db/online_sales_dataset/online_sale
s_dataset_large.csv, before offset: 6965414

Fetched 10 row(s) in 0.34s
```

Figure 33. Time taken for upscaled dataset.

This query has been executed using the large dataset to fetch 10 rows. The 10 rows were fetched in **0.34 seconds**. It took around **more 0.10 seconds** to fetch the rows of the top 10 products than the original dataset.

2. Virtual Memory Utilization

```
+-----------+---------------+
| StockCode | NULL          |
| SKU_1044  | 3398          |
| SKU_1198  | 3356          |
| SKU_1218  | 3336          |
| SKU_1213  | 3330          |
| SKU_1555  | 3318          |
| SKU_1587  | 3308          |
| SKU_1761  | 3300          |
| SKU_1148  | 3300          |
| SKU_1726  | 3242          |
+-----------+---------------+
WARNINGS: Error converting column: 3 to INT
Error parsing row: file: hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_sales.db/online_sales_dataset/online_sal
s_dataset.csv, before offset: 6965414
Error converting column: 3 to INT
Error parsing row: file: hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_sales.db/online_sales_dataset/online_sal
s_dataset_large.csv, before offset: 6965414

Fetched 10 row(s) in 0.34s
[quickstart.cloudera:21000] >
```

```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ vmstat 2
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 438920  59328 1153436    0    0   673   283  650 1117  5  6 87  2  0
 1  1      0 425528  59328 1155804    0    0  1180     6 1887 4238  5  3 92  0  0
 0  0      0 422708  59336 1160304    0    0  2250    62 1547 3722  2  2 96  0  0
 0  0      0 423012  59336 1160304    0    0     0     0  990 2417  0  1 99  0  0
 3  0      0 422996  59344 1160296    0    0     0    44 1387 3841  2  2 97  0  0
 0  0      0 423044  59344 1160304    0    0     0     0  933 2276  1  1 99  0  0
 0  0      0 423168  59344 1160304    0    0     0     6  976 2401  1  1 99  0  0
 0  0      0 423248  59344 1160304    0    0     0    22  929 2346  0  1 99  0  0
 1  0      0 423124  59352 1160308    0    0     0    18 1107 2764  1  1 98  0  0
```

Figure 34. Virtual Memory Utilization for the upscaled dataset while executing the top 10 products by quantity sold.

In the previous figure, we can observe the free memory available in the virtual machine while running Top 10 Products by quantity sold on the upscaled dataset, the avg of the free memory was around **423,561 kb** which is using more memory than executing the same query on original dataset.

5.4. PIG Employment

Apache Pig is a high-level platform for creating data processing programs that run on Hadoop. It uses a scripting language called Pig Latin, which simplifies the task of analyzing large datasets. Pig translates these scripts into MapReduce jobs, making it an efficient tool for big data analytics. In this project, Pig was employed to perform various analytical operations on the dataset, such as calculating

 doi:10.20944/preprints202512.0675.v1

total revenue, identifying top products by sales, analyzing monthly trends, determining the average discount per product category, and calculating total orders by payment method. We chose to demonstrate the execution time using the query for calculating total revenue and memory utilization using the query for identifying top products by sales. All queries were executed in the Pig shell.

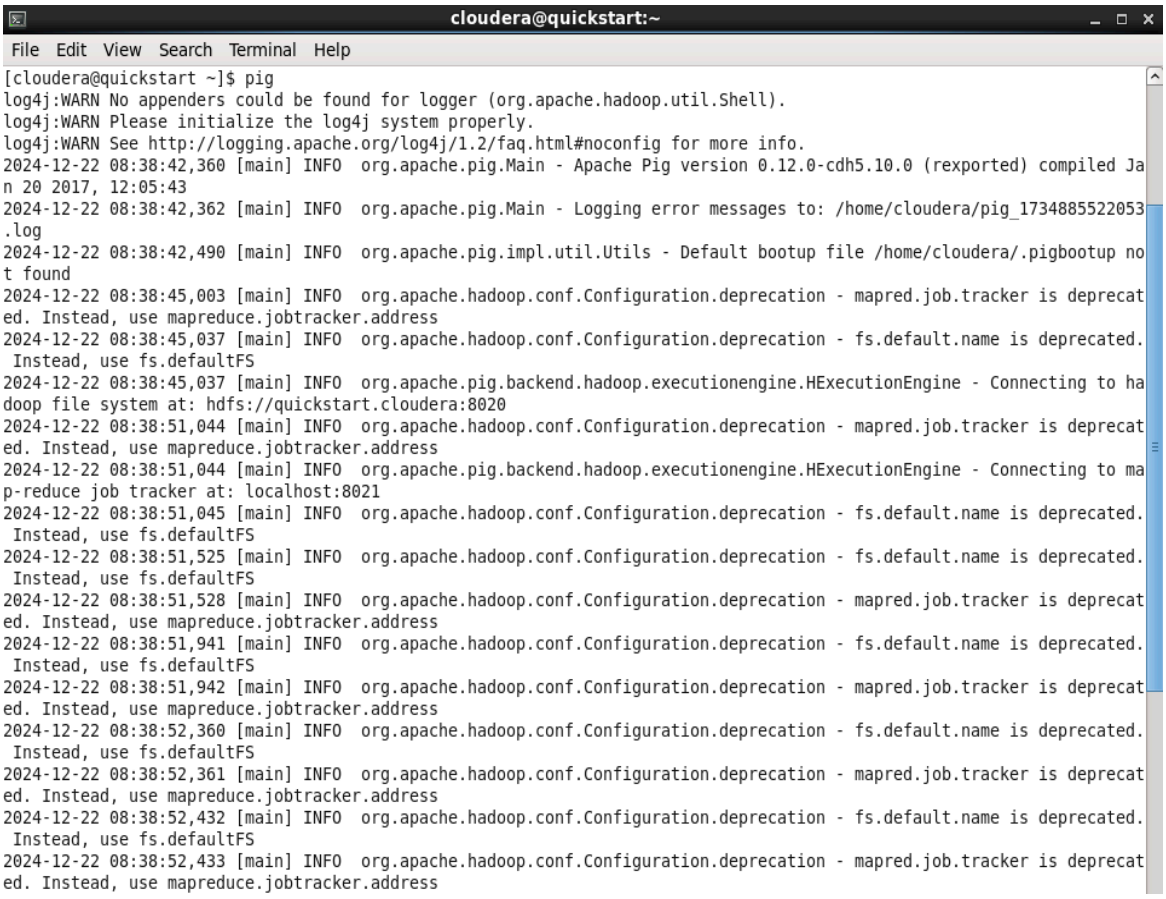


Figure 35. Entering the PIG shell.

After entering into the pig shell it is compulsory to load the dataset within the pig before running any queries.

```
grunt> -- load the dataset from HDFS
grunt>
grunt> raw_data = LOAD '/user/cloudera/online_sales_dataset/online_sales_dataset.csv' USING PigStorage(',')
>> AS (InvoiceNo:chararray,
    StockCode:chararray,
    Description:chararray,
    Quantity:int,
    InvoiceDate:chararray,
    UnitPrice:float,
    CustomerID:float,
    Country:chararray,
    Discount:float,
    PaymentMethod:chararray,
    ShippingCost:float,
    Category:chararray,
    SalesChannel:chararray,
    ReturnStatus:chararray,
    ShipmentProvider:chararray,
    WarehouseLocation:chararray,
    OrderPriority:chararray);
grunt>
```

Figure 36. Loading the dataset from HDFS.

 doi:10.20944/preprints202512.0675.v1

5.4.1. Time Taken:

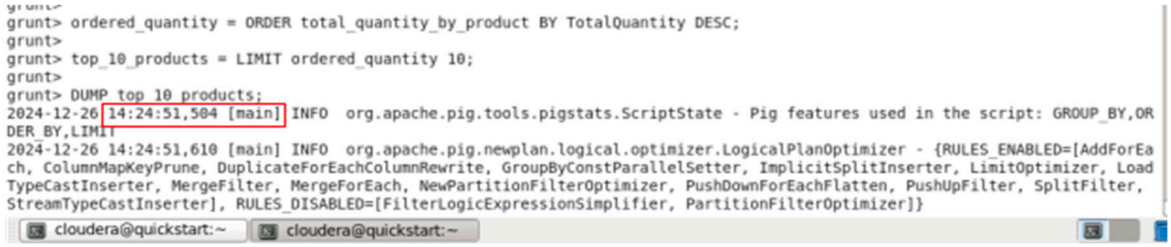


Figure 37. Starting time.

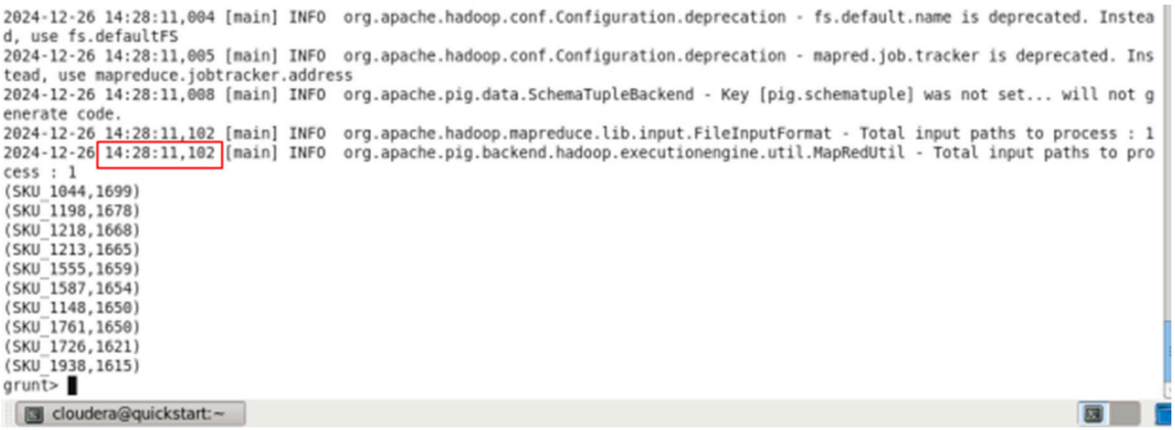


Figure 38. Finishing time.

The execution time for the query “Top 10 Products by Quantity Sold” was 3 minutes and 26 seconds. This time reflects the duration required to process the dataset and compute the total quantity sold for each product, followed by ordering the results and limiting the output to the top 10 products. The process involved grouping the data by StockCode, aggregating the Quantity

field, sorting the results in descending order of total quantity sold, and then limiting the result to the top 10 products.

This execution time is a crucial metric, as it demonstrates the efficiency of the query in processing a large dataset containing over 49,000 transaction records, making it suitable for real-time or near-real-time analysis in a retail or e-commerce environment.

5.4.2. Virtual Memory Utilization

The virtual memory utilization during the execution of the “Top 10 Products by Quantity Sold” query on the dataset offers valuable insights into the system’s resource consumption when processing large volumes of data in Pig.

 doi:10.20944/preprints202512.0675.v1


```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0 1572508 619036   1572  97268    0    0    76   452 2033 3277  0  1 99  0  0
 0  0 1572492 619100   1580  97264    0    0     0   100 2003 3346  1  0 99  0  0
 0  0 1572452 618984   1580  97272   48    0    48     0 1808 3127  0  1 98  0  0
 0  0 1572432 618984   1580  97248   32    0    32     0 1630 3060  0  1 99  0  0
 1  0 1572400 619024   1580  97272    0    0     0     0 1707 3112  0  1 99  0  0
 0  0 1572360 619024   1580  97276   32    0    32     0 1851 3201  1  1 99  0  0
 1  0 1570584 614504   1592  97368 4612    0  4760    80 2985 4085  2  3 93  3  0
 0  0 1570564 614592   1592  97396   64    0    64     0 1615 3042  0  1 99  0  0
 0  0 1570524 614732   1592  97396   40    0    40     0 1754 3070  0  1 99  0  0
 0  0 1570480 614608   1592  97436   32    0    32     0 1911 3244  1  1 98  0  0
 0  0 1570240 608036   1592 102240 2080    0  6920   328 2659 4318  1  1 94  4  0
 0  0 1569892 606408   1600 102212 1568    0  1720   120 2204 3617  1  1 98  1  0
 0  0 1569844 606160   1600 102344  168    0   168     0 1900 3287  0  1 98  0  0
 4  0 1569792 605912   1600 102432   64    0   244     0 1711 3113  1  1 98  0  0
 5  0 1569764 605756   1600 102600    0    0     0     0 1948 3546  1  1 98  0  0
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
```

Figure 39. Before running Top 10 Products by Quantity Sold.

```
[cloudera@quickstart ~]$ vmstat 1
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0 1105952 161584  16192 216176   15   43   123    69  259  345  2  3 93  2  0
 1  0 1105812 161040  16192 216048  280    0   280    12 2351 4038  1  1 97  0  0
 0  0 1105764 160876  16192 216176   16    0    16     0 1847 3594  1  1 99  0  0
 1  1 1105716 148548  16200 218704   96    0  2864   116 3954 4107  8  1 89  1  0
 6  1 1105640 105516  14216 202768   64    0 21120    52 6546 6820  6  9 79  6  0
 2  1 1105592 100184  14184 193116   48    0   272    44 4092 3798  7  3 85  5  0
 1  1 1105532 100488  13280 186088   64    0   108     0 3837 2820  8  2 81  9  0
 6  1 1105476  94164  11616 180356   32    0   164     0 3471 2642  7  2 81 10  0
 2  0 1105440  92428  10516 170688    0    0    28   176 3230 2560  7  2 81  9  0
 1  1 1105396  92884   9896 161488   40    0   360     0 4313 3467  6  5 81  8  0
 4  1 1105352  88180   8556 160972   32    0  3936    76 7246 5490  7  7 78  9  0
 1  1 1105296  90936   6684 157636   32    0  1068    20 4530 4352  6  4 81  9  0
 4  6 1105264  92860   6428 151400  108    0   300  6296 4586 2960  5 29 54 12  0
 9  2 1105164  95176   4688 135472  192    0  1132 42668 5543 3457 11 26 41 22  0
 3  3 1104844 108168   4724 123472  232    0  3308     4 5474 4293  5 19 55 21  0
```

Figure 40. Top 10 Products by Quantity Sold VMU WHILE RUN.

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0 1953404 604660   1572 153120  128    0   128     0 2027 3489  1  2 96  0  0
 1  0 1953372 603940   1576 153836   64    0   764     0 1516 2948  1  2 97  0  0
 2  4 1953312 603384   1584 154896  224    0  1228    80 1994 3126  1  2 96  1  0
 0  0 1953232 601260   1584 156484   96    0  1724    12 1811 3278  1  1 96  2  0
 0  0 1953216 601276   1584 156468    0    0     0     0 1937 3473  1  1 98  0  0
 0  0 1953112 600400   1584 157288  264    0  1216     0 2322 3662  1  2 97  1  0
 0  0 1953024 587936   1656 169476  352    0 12336     0 2313 4915  1  1 92  5  0
 2  1 1953004 587664   1656 169520  128    0   128     0 1861 3348  1  1 98  0  0
 0  0 1952956 594304   4556 161880  288    0  3792    56 3416 4870  2  6 89  3  0
 0  0 1952904 978056   4560 162368   64    0   336     0 1947 3095  1  1 98  0  0
 0  0 1952864 977172   4560 162820   96    0   508     0 1506 2775  1  1 97  1  0
 0  0 1952848 977304   4560 162820   32    0    32     0 1790 3199  1  1 98  0  0
```

Figure 41. After finishing and query and closing terminal.

Initially, before running the query, the system had 619,063 KB of free memory. However, as the query was executed, the memory utilization increased, causing the free memory to decrease significantly, reaching its lowest point at 88,180 KB. This decline in available memory can be attributed to the processing load involved in aggregating and sorting a large number of records,

 doi:10.20944/preprints202512.0675.v1

particularly given the substantial size of the dataset with over 49,000 transactions. After the query execution was completed and the terminal was closed, the system's free memory returned to its maximum value of 978,056 KB, suggesting that resources were efficiently freed up. This fluctuation in virtual memory utilization highlights the system's memory demands during intensive data operations and the need for effective resource management when running large-scale queries in Pig.

5.4.3. Scalability

```
[cloudera@quickstart ~]$ hdfs dfs -cp /user/pig/online_sales_dataset.csv /user/p
ig/online_sales_dataset_large.csv
[cloudera@quickstart ~]$ hdfs dfs -ls /user/pig/
Found 2 items
-rw-r--r--   1 cloudera supergroup    6965414 2024-12-26 11:52 /user/pig/online_
sales_dataset.csv
-rw-r--r--   1 cloudera supergroup    6965414 2024-12-26 15:33 /user/pig/online_
sales_dataset_large.csv
[cloudera@quickstart ~]$
```

Figure 42. Enlarging Dataset.

Here we have upscaled the dataset by just copying the older dataset and using large.csv.

1. **Time Taken:** The execution time for the query "Top 10 Products by Quantity Sold" was 3 minutes 31.32 seconds.

```
grunt> raw_data = LOAD '/user/pig/online_sales_dataset_large.csv' USING PigStora
ge(',')
>>     AS (InvoiceNo:chararray,
>>         StockCode:chararray,
>>         Description:chararray,
>>         Quantity:int,
>>         InvoiceDate:chararray,
>>         UnitPrice:float,
>>         CustomerID:float,
>>         Country:chararray,
>>         Discount:float,
>>         PaymentMethod:chararray,
>>         ShippingCost:float,
>>         Category:chararray,
>>         SalesChannel:chararray,
>>         ReturnStatus:chararray,
>>         ShipmentProvider:chararray,
>>         WarehouseLocation:chararray,
>>         OrderPriority:chararray);
grunt> quantity_by_product = FOREACH raw_data GENERATE StockCode, Quantity;
grunt>
grunt> -- Group data by StockCode for efficient aggregation
grunt> grouped_quantity = GROUP quantity_by_product BY StockCode;
grunt>
grunt> total_quantity_by_product = FOREACH grouped_quantity GENERATE
>>     group AS StockCode,
>>     SUM(quantity_by_product.Quantity) AS TotalQuantity;
grunt>
grunt> ordered_quantity = ORDER total_quantity_by_product BY TotalQuantity DESC;
grunt>
grunt> top_10_products = LIMIT ordered_quantity 10;
grunt>
grunt> DUMP top_10_products;
2024-12-26 17:04:31,804 [main] INFO  org.apache.pig.tools.pigstats.ScriptState -
 Pig features used in the script: GROUP_BY,ORDER_BY,LIMIT
2024-12-26 17:04:31,865 [main] INFO  org.apache.pig.newplan.logical.optimizer.Lo
gicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateFor
EachColumnRewrite, GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptim
```

 doi:10.20944/preprints202512.0675.v1


```
ILLISECONDS)
2024-12-26 17:07:30,475 [main] INFO  org.apache.hadoop.ipc.Client - Retrying con
nect to server: quickstart.cloudera/10.0.2.15:37681. Already tried 1 time(s); re
try policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=3, sleepTime=1000 M
ILLISECONDS)
2024-12-26 17:07:31,479 [main] INFO  org.apache.hadoop.ipc.Client - Retrying con
nect to server: quickstart.cloudera/10.0.2.15:37681. Already tried 2 time(s); re
try policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=3, sleepTime=1000 M
ILLISECONDS)
2024-12-26 17:07:31,594 [main] INFO  org.apache.hadoop.mapred.ClientServiceDeleg
ate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirect
ing to job history server
2024-12-26 17:07:32,337 [main] WARN  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Encountered Warning FIELD_DISCARDED_TYPE_C
ONVERSION_FAILED 1 time(s).
2024-12-26 17:07:32,338 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2024-12-26 17:07:32,356 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2024-12-26 17:07:32,357 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2024-12-26 17:07:32,358 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2024-12-26 17:07:32,409 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2024-12-26 17:07:32,409 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(SKU_1044,1699)
(SKU_1198,1678)
(SKU_1218,1668)
(SKU_1213,1665)
(SKU_1555,1659)
(SKU_1587,1654)
(SKU_1148,1650)
(SKU_1761,1650)
(SKU_1726,1621)
(SKU_1938,1615)
grunt>
```

Figure 43. Upscaled Datasets Output for Top 10 Products by Quantity Sold.

This will give the time taken for processing the enlarged dataset and calculating the total quantity sold for every product, ordering the result, and showing only the top 10 products. This involved grouping the data by StockCode, summing up the Quantity field, sorting the results in descending order of total quantity sold, and then limiting the result to the top 10 products.

This execution time is one of the most important metrics, since it shows how well the query scales for an upscaled larger dataset with more than 49,000 transaction records, thus making it suitable for real-time or near-real-time analysis in a retail or e-commerce environment.

2. **VMU :** Virtual memory used by the "Top 10 Products by Quantity Sold" query, when executed on the larger dataset, gives an idea about how the system consumes resources to process a large volume of data in Pig.

 doi:10.20944/preprints202512.0675.v1

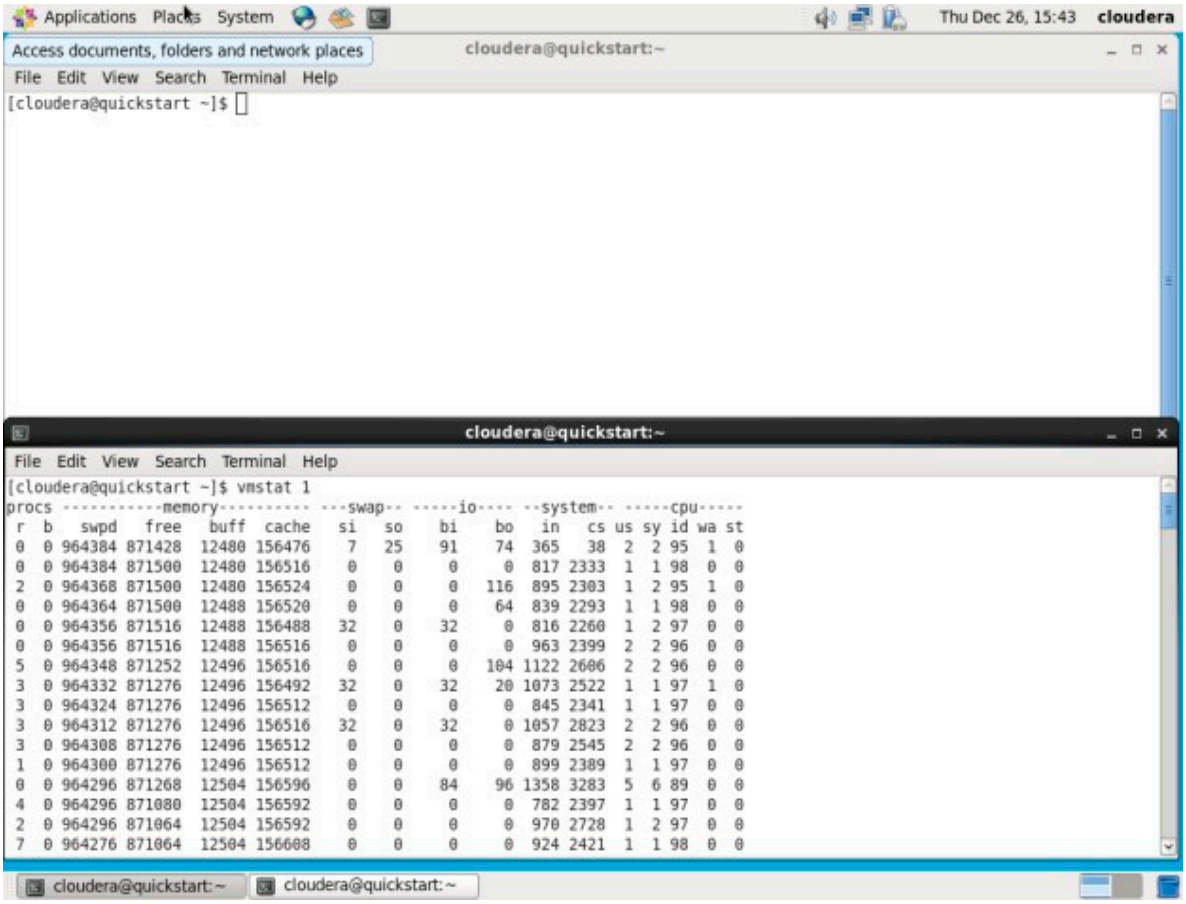
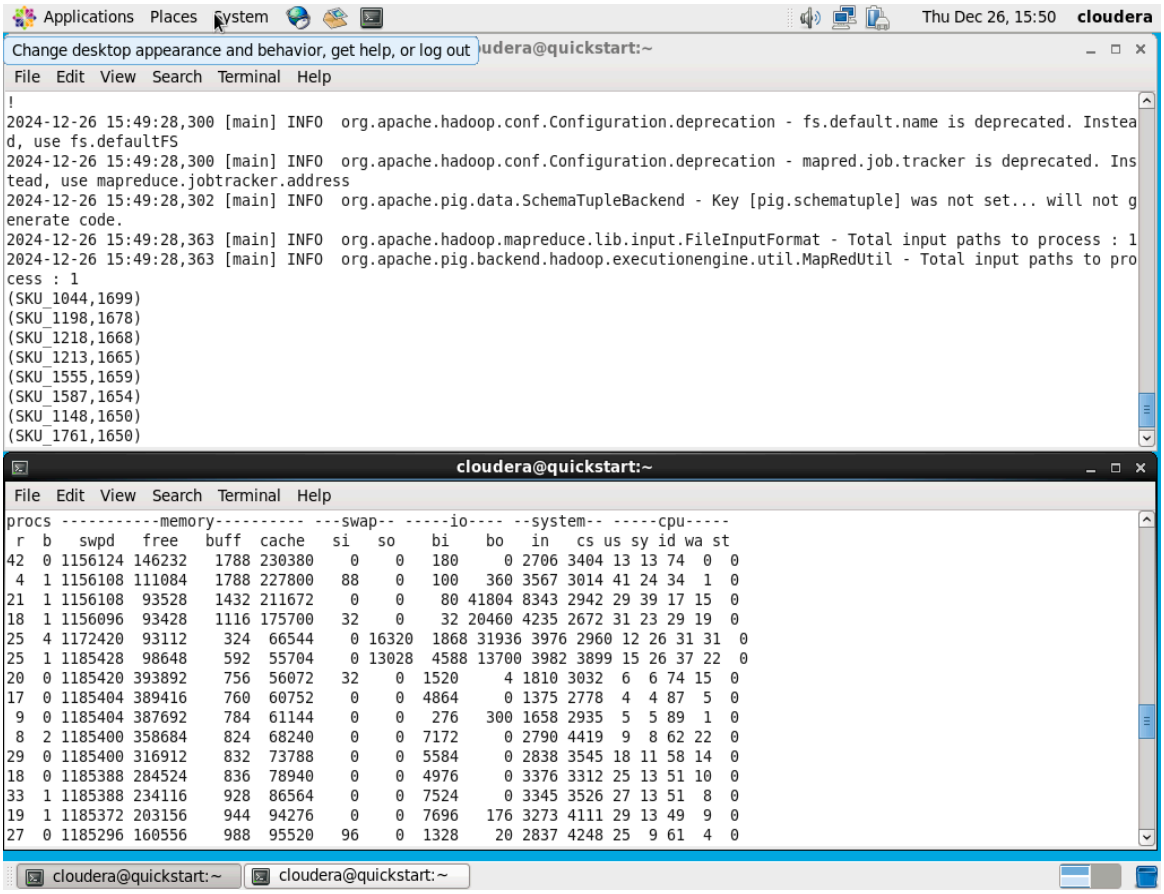


Figure 44. VMU before running the query.



 doi:10.20944/preprints202512.0675.v1

Figure 45. While running the query.

This query increased memory utilization drastically during its execution, where minimum free memory was 93,112.When the execution of queries was over and the terminal was closed, free memory reached again its maximum value, equal to 39,3832 KB, hence resources were well freed. This fluctuation in virtual memory utilization indicates the memory demands of the system during intensive data operations and the need for effective resource management when running large-scale queries in Pig. After it finished execution, it went back to 871,516kb.

6.0. Performance Comparison

A comparison between the three tools Hive, Impala and Pig is done based on three metrics: time taken to finish executing a query, virtual memory utilized while executing a query and the scalability.

Time Taken

	Hive	Impala	Pig
Time Taken in seconds (s)	39.911s	0.25s	199.60s

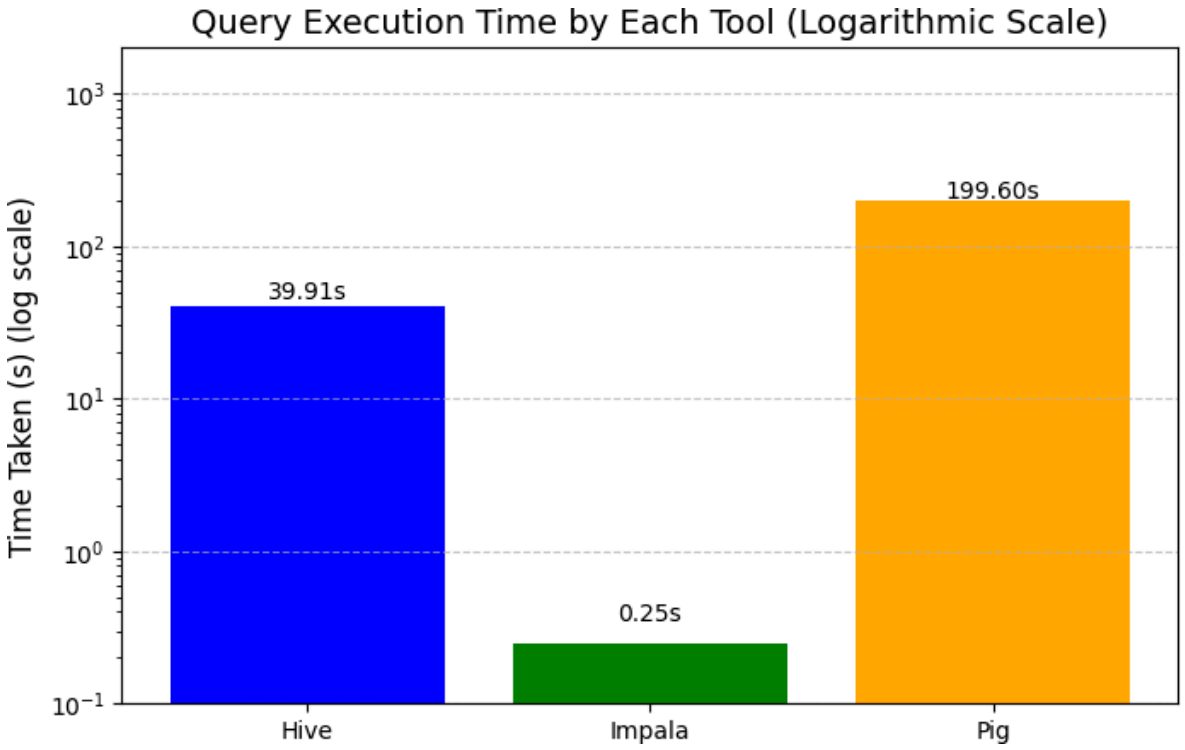


Figure 46. Execution time illustration.

The execution time of the SQL-like query varies widely amongst the three big data tools: Hive, Impala, and Pig. Indeed, Pig has the highest execution time, with about 199.6 seconds, to emerge as the slowest. Hive performed moderately at roughly 39.91 seconds, while Impala recorded the fastest time with an execution time of just about 0.25 seconds.

This huge difference in execution times points to the performance characteristics of each tool. Impala is faster because of its in-memory processing engine, which avoids disk I/O overheads and gives very low latency. In contrast, Pig depends on batch processing through MapReduce, involving high disk operations and storage of intermediate data, hence the slow performance. Hive, though

 doi:10.20944/preprints202512.0675.v1

faster than Pig, is also based on a MapReduce-like execution model but applies additional query execution optimizations, hence its intermediary performance.

Results prove that for real-time or low latency analytics, Impala is suitable, whereas Hive and Pig are good candidates for batch-oriented processing, where the speed is not so important

Virtual Memory Utilization

	Hive	Impala	Pig
Free Memory Available (kb)	97,588 kb	461,418 kb	88,180 kb

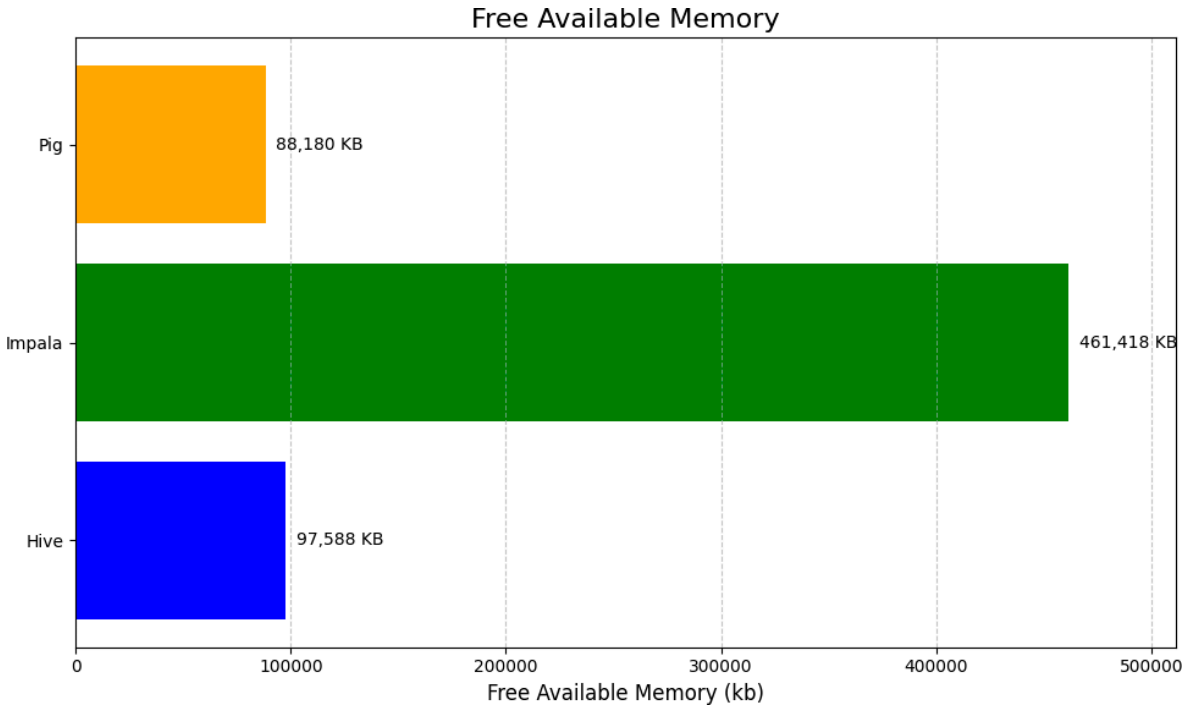


Figure 47. Free Memory Available while execution illustration.

The horizontal bar chart visualizes the free available memory utilized by three big data tools—Hive, Impala, and Pig—measured in kilobytes (KB). Impala shows the highest memory utilization at 461,418 KB, significantly exceeding that of Hive (97,588 KB) and Pig (88,180 KB). The differences in memory usage highlight Impala's more resource-intensive operations compared to the relatively lower memory demands of Hive and Pig. The chart effectively uses annotations to display exact memory usage for each tool, providing a clear and concise comparison.

Scalability

- ***Time Taken***

	Hive	Impala	Pig
Time Taken in seconds (s)	43.324s	0.34s	181.32s

 doi:10.20944/preprints202512.0675.v1

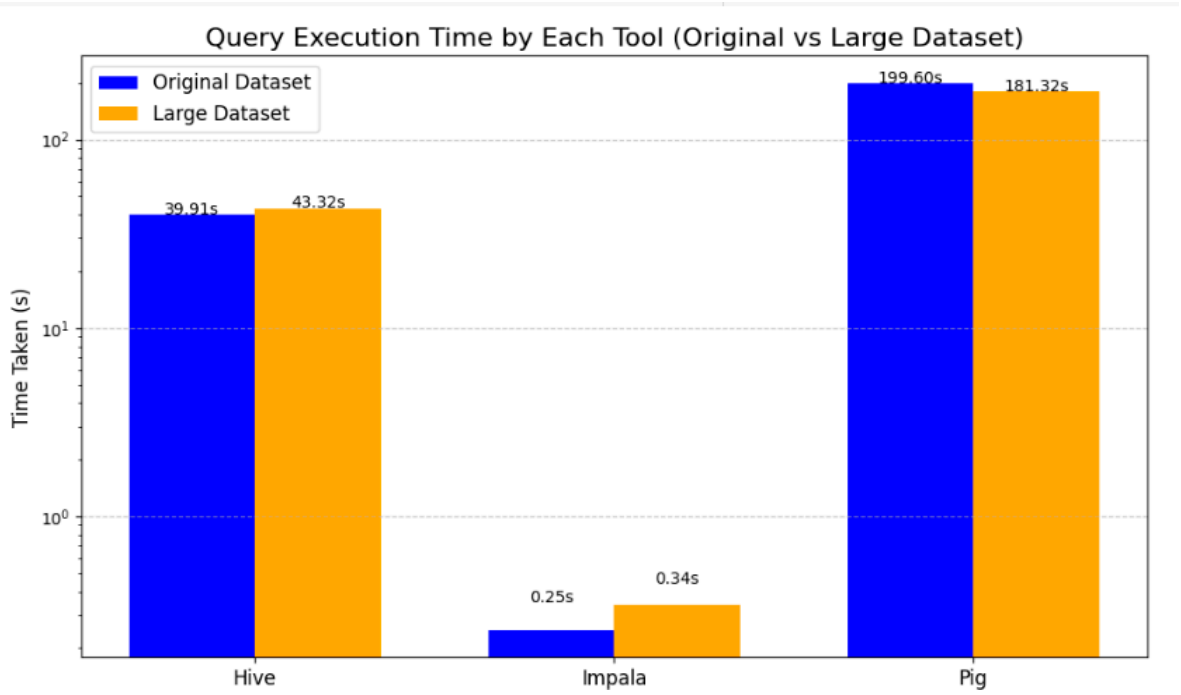


Figure 48. Query execution time comparison.

The bar chart illustrates the query execution times for three big data tools—Hive, Impala, and Pig—when applied to two datasets: the original dataset and a larger dataset. The y-axis is scaled logarithmically to effectively capture the significant variations in execution times across the tools. Hive exhibits a slight increase in execution time, rising from 39.91 seconds for the original dataset to 43.32 seconds for the larger dataset. Impala, known for its low-latency query performance, maintains exceptionally short execution times, increasing marginally from 0.25 seconds to 0.34 seconds as the dataset size grows. Interestingly, Pig demonstrates an improvement in execution efficiency for the larger dataset, with its execution time decreasing from 199.60 seconds to 181.32 seconds.

This comparison underscores the consistent efficiency of Impala, particularly for rapid query execution, while Hive and Pig reveal more variable performance patterns depending on dataset size and complexity.

- ***Virtual Memory Utilization***

	Hive	Impala	Pig
Virtual Memory Utilized (kb)	93,380 kb	423,561 kb	93,112 kb

 doi:10.20944/preprints202512.0675.v1

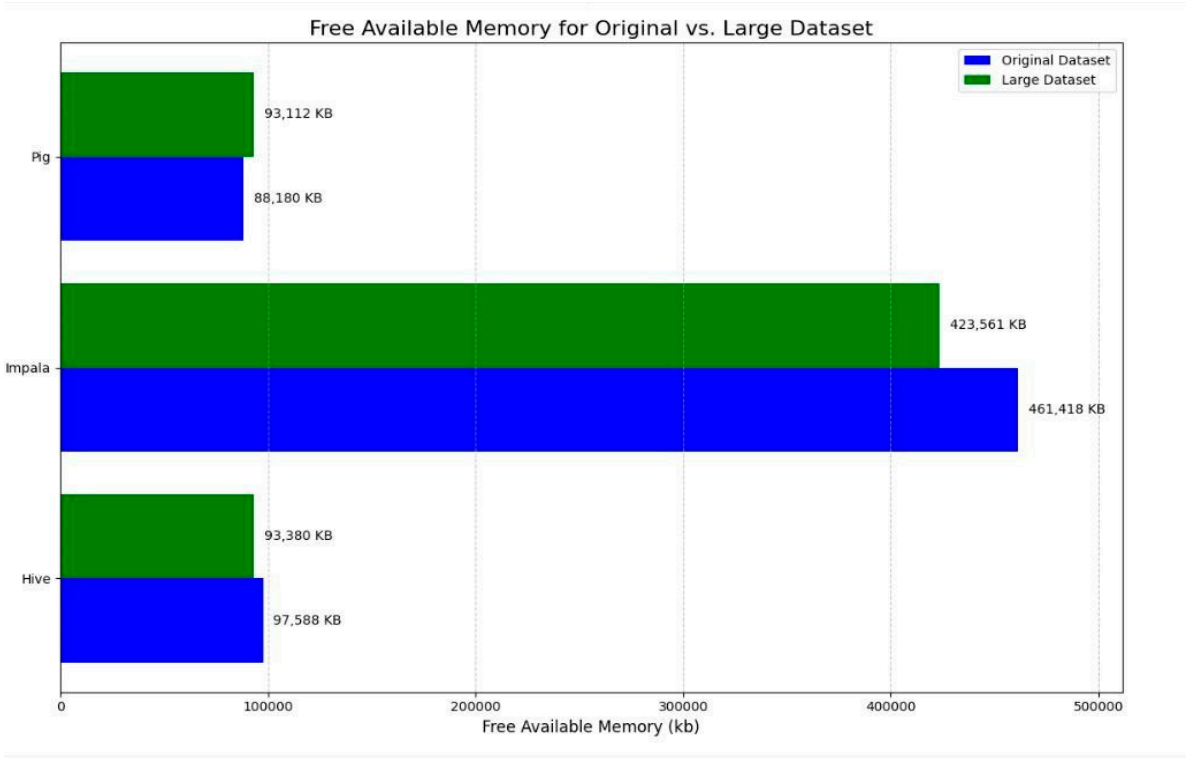


Figure 49. Free Available Memory Comparison.

7.0. Performance Visualization

Query results, discussion, and visualizations

1. Executing 5 Queries on Hive

1st Query

```
    > SELECT Country, SUM(Quantity * UnitPrice) AS TotalRevenue
    > FROM online_sales_dataset
    > WHERE Country IS NOT NULL AND Country != 'Country'
    > GROUP BY Country
    > ORDER BY TotalRevenue DESC;
Query ID = cloudera_20241226132727_f984d4cf-3fcf-4675-b1ce-3bc643afebe6
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1735230749899_0014, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1735230749899_0014/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1735230749899_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-12-26 13:27:48,123 Stage-1 map = 0%,  reduce = 0%
2024-12-26 13:27:56,763 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.06 sec
2024-12-26 13:28:02,968 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.06 sec
MapReduce Total cumulative CPU time: 3 seconds 140 msec
Ended Job = job_1735230749899_0014
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1735230749899_0015, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1735230749899_0015/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1735230749899_0015
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2024-12-26 13:28:10,900 Stage-2 map = 0%,  reduce = 0%
2024-12-26 13:28:16,319 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.98 sec
2024-12-26 13:28:23,615 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.62 sec
```

Figure 50. Total Revenue by Country QUERY.

 doi:10.20944/preprints202512.0675.v1


```
OK
United Kingdom  5128232.814706683
Belgium 5124058.2221621275
Germany 5111343.359713912
United States   5076174.167352557
France  5063516.554258347
Sweden  5028849.943408966
Spain   5014548.617865086
Portugal        4974618.95413053
Netherlands     4934815.540013552
Norway  4930232.349189281
Italy   4909348.5976758
Australia       4846793.981884122
Time taken: 43.583 seconds, Fetched: 12 row(s)
```

Figure 51. Total Revenue by Country OUTPUT.

In the query command, it sums up transaction revenues as "TotalRevenues" by do a multiplication between "(Quantity * UnitPrice)" and grouping them by country to determine the total revenue earned per country. A condition was added to prevent a NULL value in front of

Country "**WHERE Country IS NOT NULL AND Country != 'Country'**". The top-performing countries are identified by sorting the output in order from highest country's total revenue to lowest country. The insights driven from this query is to guide price plans, specific advertising efforts, and distribution of resources.

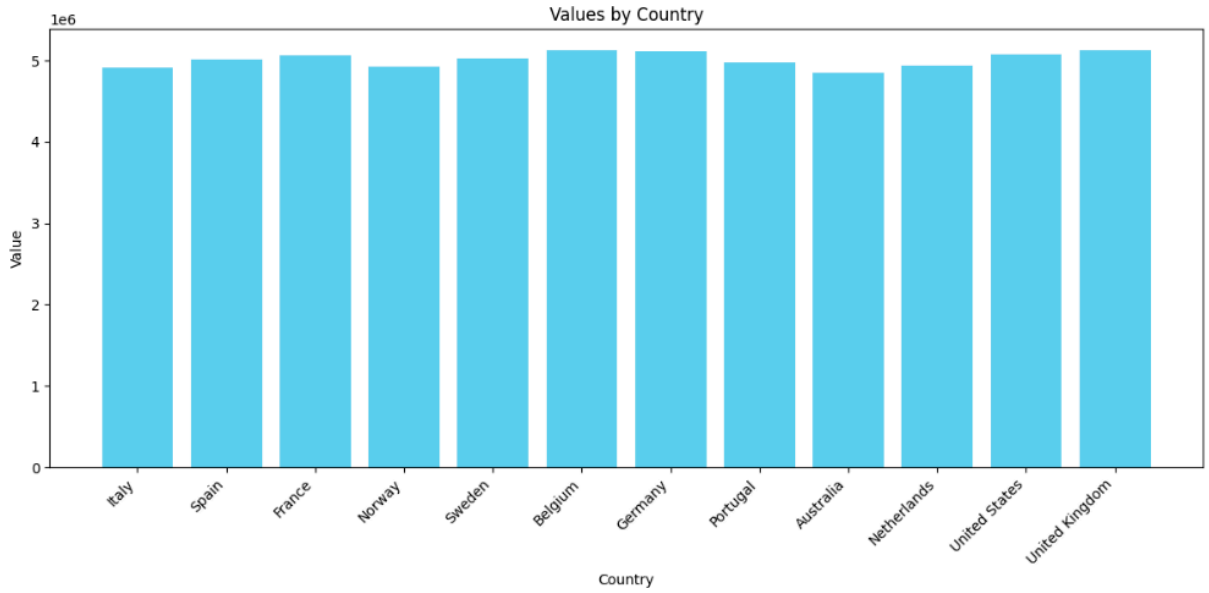


Figure 52. Total revenue by countries bar chart.

2nd Query

 doi:10.20944/preprints202512.0675.v1


```
hive> SELECT StockCode, SUM(Quantity) AS TotalQuantity
    > FROM online_sales_dataset
    > GROUP BY StockCode
    > ORDER BY TotalQuantity DESC
    > LIMIT 10;
Query ID = cloudera_20241221191313_70fc040a-92d1-4e10-b170-934d6bd00675
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1734835782760_0002, Tracking URL = http://quickstart.cloudera:80
8/proxy/application_1734835782760_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1734835782760_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-12-21 19:13:39,896 Stage-1 map = 0%,  reduce = 0%
2024-12-21 19:13:45,074 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 sec
2024-12-21 19:13:50,258 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.69 sec
MapReduce Total cumulative CPU time: 2 seconds 690 msec
Ended Job = job_1734835782760_0002
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
```

Figure 53. Top 10 Products by Quantity Sold QUERY.

```
Starting Job = job_1734835782760_0003, Tracking URL = http://quickstart.cloudera:80
88/proxy/application_1734835782760_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1734835782760_0003
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2024-12-21 19:13:57,428 Stage-2 map = 0%,  reduce = 0%
2024-12-21 19:14:02,971 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.92 sec
2024-12-21 19:14:09,226 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.53 sec
MapReduce Total cumulative CPU time: 2 seconds 530 msec
Ended Job = job_1734835782760_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.69 sec   HDFS Read: 6973558 HD
FS Write: 29403 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.53 sec   HDFS Read: 34296 HDFS
 Write: 140 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 220 msec
OK
SKU_1044        1699
SKU_1198        1678
SKU_1218        1668
SKU_1213        1665
SKU_1555        1659
SKU_1587        1654
SKU_1761        1650
SKU_1148        1650
SKU_1726        1621
SKU_1938        1615
Time taken: 36.533 seconds, Fetched: 10 row(s)
```

Figure 54. Top 10 Products by Quantity Sold OUTPUT.

In the query command, it identifies the top 10 sold products by their quantity. Both “StockCode” and “Description” are grouped to calculate the “TotalQuantity” of the sold products using “SUM(Quantity)”, and sort the results from highest to highest sold ones to the lowest sold ones. This query reveals the most desirable products, allowing companies to set priorities for inventory management, enhance marketing strategies, and concentrate on those that bring the highest revenue.

 doi:10.20944/preprints202512.0675.v1

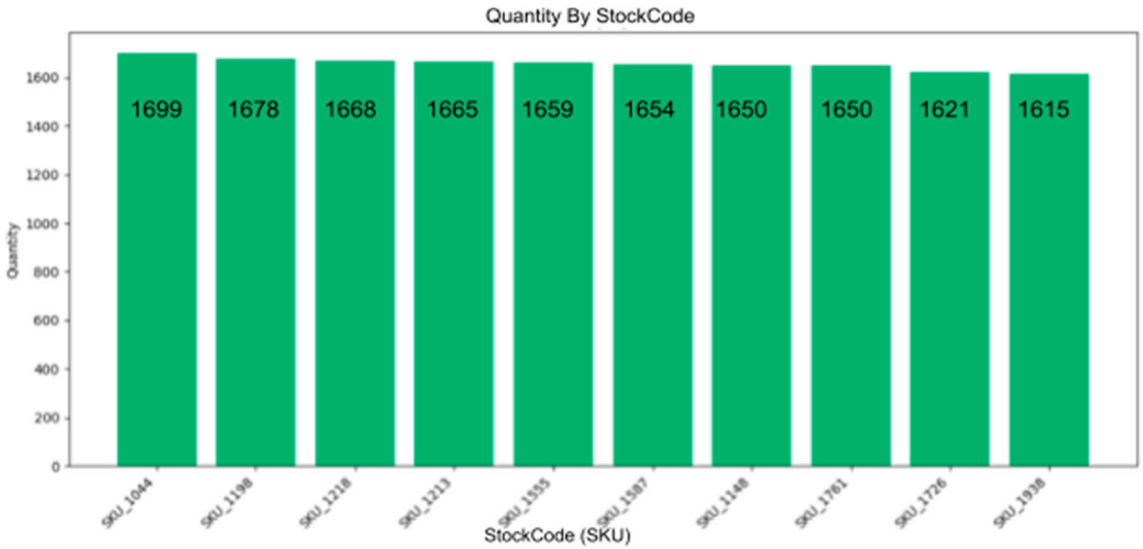


Figure 55. Top 10 products by quantity sold bar chart.

3rd Query

```
hive> SELECT
    >     YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(InvoiceDate, 'yyyy-MM-dd HH:mm'))) AS Year,
    >     SUM(Quantity * UnitPrice) AS AnnualRevenue
    > FROM
    >     online_sales_dataset
    > GROUP BY
    >     YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(InvoiceDate, 'yyyy-MM-dd HH:mm')))
    > ORDER BY
    >     Year;
Query ID = cloudera_20241227070202_f33c5b76-6953-46fe-9ce6-3b131f3e5248
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1735306856049_0001, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1735306856049_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1735306856049_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-12-27 07:02:37,787 Stage-1 map = 0%,  reduce = 0%
2024-12-27 07:02:45,328 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.74 sec
2024-12-27 07:02:53,651 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.23 sec
MapReduce Total cumulative CPU time: 5 seconds 230 msec
Ended Job = job_1735306856049_0001
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1735306856049_0002, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1735306856049_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1735306856049_0002
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2024-12-27 07:03:02,073 Stage-2 map = 0%,  reduce = 0%
```

Figure 56. Yearly Sales Query.

```
OK
NULL    NULL
2020    1.0522915882369518E7
2021    1.0664760209717631E7
2022    1.056025614107585E7
2023    1.0492226550386548E7
2024    1.0716573185217857E7
2025    7185801.133593559
Time taken: 53.151 seconds, Fetched: 7 row(s)
```

Figure 57. Yearly Sales Query Output 1&2.

 doi:10.20944/preprints202512.0675.v1

The previous query command was to calculate the total revenue each year by taking out the year from "InvoiceDate" and then sums up the revenue by a multiplication between (Quantity * UnitPrice" for every year. The results will be both sorted and grouped by the years. This is deriving insights about the overall sales trends over time, making it possible to assess the growth and performance of the company. Through the identification of revenue changes, companies are able to evaluate how plans and competition affect their annual results.

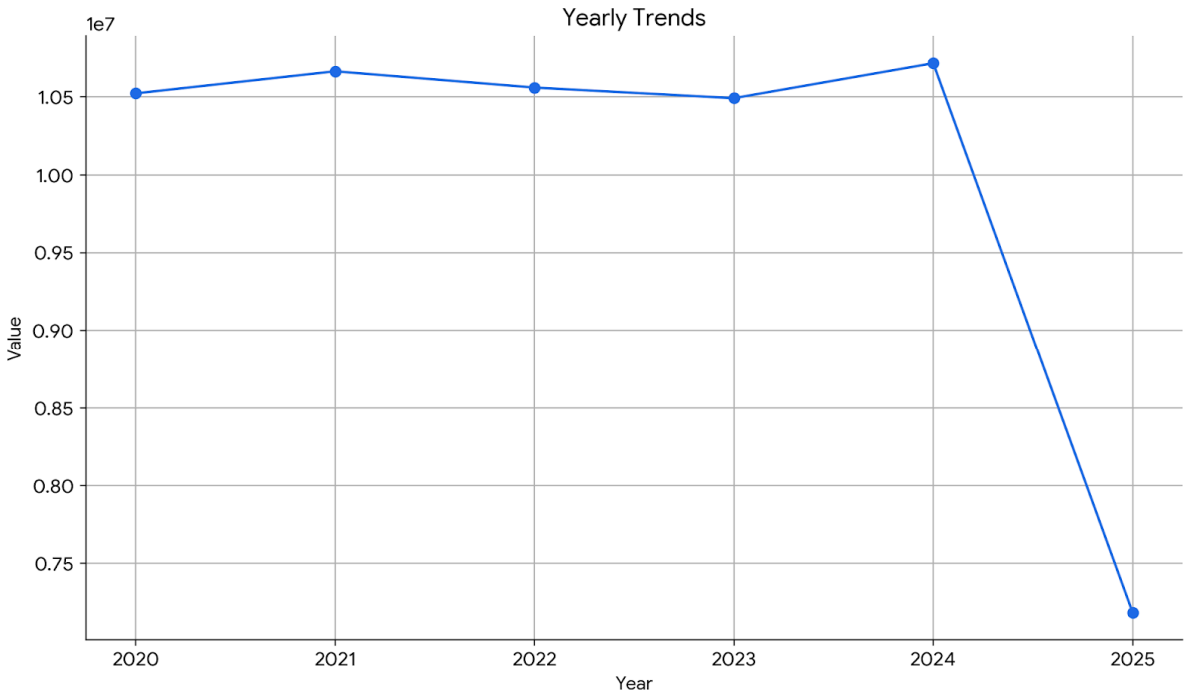


Figure 58. Yearly sales trends illustrations.

4th Query

```
hive> SELECT Category, AVG(Discount) AS AvgDiscount
    > FROM online_sales_dataset
    > WHERE Category IS NOT NULL AND Category != 'Category'
    > GROUP BY Category
    > ORDER BY AvgDiscount DESC;
Query ID = cloudera_20241226133333_b122a913-8f14-4fa2-adc8-8db99f4419f7
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1735230749899_0016, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1735230749899_0016/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1735230749899_0016
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-12-26 13:33:43,440 Stage-1 map = 0%,  reduce = 0%
2024-12-26 13:33:50,769 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.1 sec
2024-12-26 13:33:59,068 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.13 sec
MapReduce Total cumulative CPU time: 3 seconds 130 msec
Ended Job = job_1735230749899_0016
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1735230749899_0017, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1735230749899_0017/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1735230749899_0017
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2024-12-26 13:34:06,838 Stage-2 map = 0%,  reduce = 0%
2024-12-26 13:34:11,988 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.96 sec
2024-12-26 13:34:18,151 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 1.96 sec
```

 doi:10.20944/preprints202512.0675.v1

Figure 59. Average Discount per Product Category Query.

```
MapReduce Total cumulative CPU time: 1 seconds 960 msec
Ended Job = job_1735230749899_0017
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.13 sec   HDFS Read: 6975025 HDFS Write: 274 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 1.96 sec   HDFS Read: 5031 HDFS Write: 149 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 90 msec
OK
Furniture       0.27842945259308993
Electronics     0.2765903997156882
Stationery      0.2758971431469327
Accessories     0.2746105756717069
Apparel 0.2731570573491539
Time taken: 40.971 seconds, Fetched: 5 row(s)
```

Figure 60. Average Discount per Product Category Query Output.

The command of the query is determining the average discount "AvgDiscount" for each category of the product by using "AVG(Discount)". A condition was added to prevent a NULL value in front of Category "**WHERE Category IS NOT NULL AND Category != 'Category'**". It then classifies the data by the category, demonstrating the categories with the largest average discounts. Companies can use such data to enhance pricing, analyze the success of advertisements, and figure out how discounts affect sales and customer behavior.

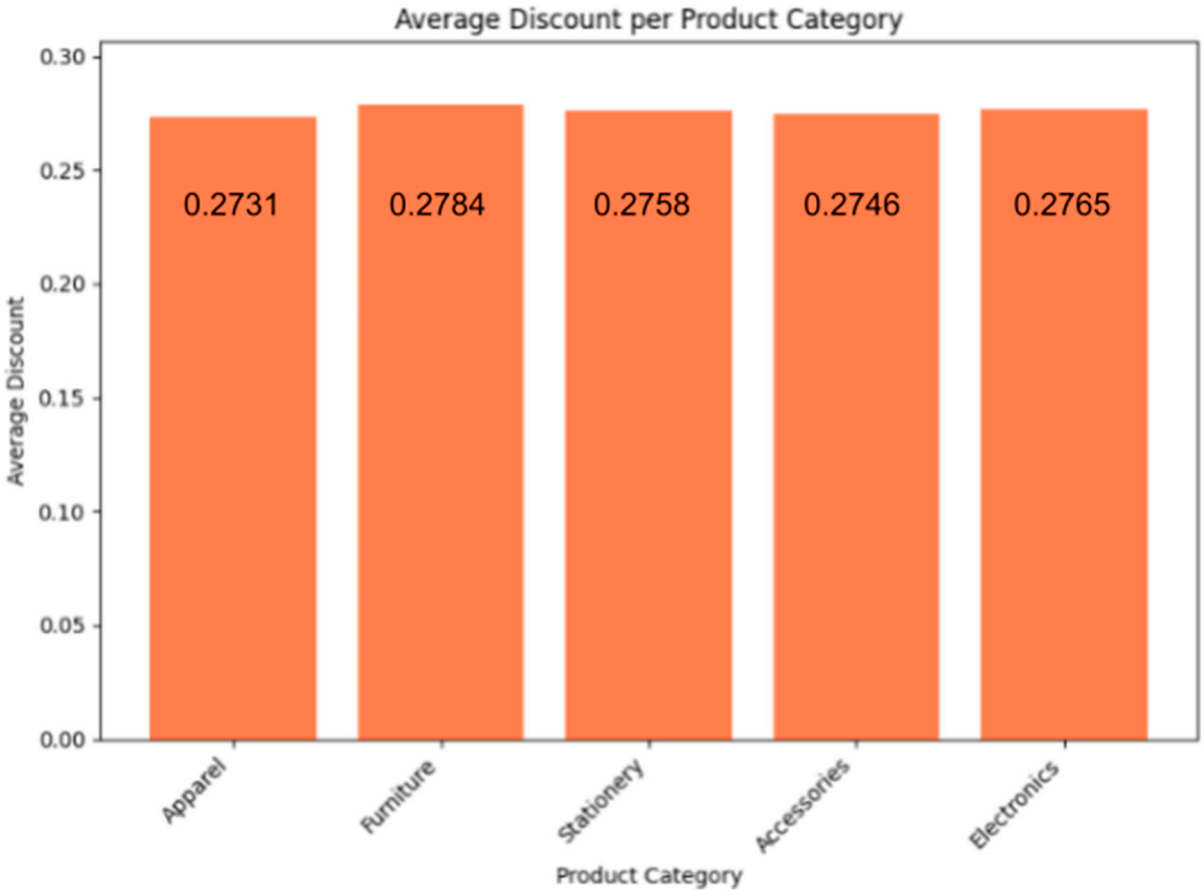


Figure 61. Average Discount Per Product Category illustrations.

5th Query

 doi:10.20944/preprints202512.0675.v1


```
hive> SELECT PaymentMethod, COUNT(*) AS TotalOrders
    > FROM online_sales_dataset
    > WHERE PaymentMethod IS NOT NULL AND PaymentMethod != 'PaymentMethod'
    > GROUP BY PaymentMethod
    > ORDER BY TotalOrders DESC;
Query ID = cloudera_20241226131515_d74673db-2c72-4214-a71b-6e7fda70e9da
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1735230749899_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1735230749899_0007/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1735230749899_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-12-26 13:15:59,403 Stage-1 map = 0%,  reduce = 0%
2024-12-26 13:16:13,617 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.54 sec
2024-12-26 13:16:21,961 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.87 sec
MapReduce Total cumulative CPU time: 3 seconds 870 msec
Ended Job = job_1735230749899_0007
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1735230749899_0008, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1735230749899_0008/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1735230749899_0008
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2024-12-26 13:16:31,126 Stage-2 map = 0%,  reduce = 0%
2024-12-26 13:16:37,375 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.09 sec
2024-12-26 13:16:43,591 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.18 sec
```

Figure 62. Total Orders by PaymentMethod Query.

```
MapReduce Total cumulative CPU time: 2 seconds 180 msec
Ended Job = job_1735230749899_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.87 sec   HDFS Read: 6973572 HDFS Write: 190 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.18 sec   HDFS Read: 4930 HDFS Write: 52 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 50 msec
OK
Bank Transfer   16747
Credit Card     16530
paypall 16505
Time taken: 55.33 seconds, Fetched: 3 row(s)
```

Figure 63. Total Orders by PaymentMethod QueryOutput.

This query is grouping out the orders by "PaymentMethod" and sorting them from highest payment method number to lowest number, and that's to determine the total number of orders in each payment method used. A condition was added to prevent a NULL value in front of PamentMethod "**WHERE PamentMethod IS NOT NULL AND PamentMethod != 'PamentMethod**'". This query is giving insights to reveal the most popular used payment options and then to give recommendations to add more options and to maximize client comfort, which will improve the overall payment experience.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 December 2025 doi:10.20944/preprints202512.0675.v1

Figure 64. Total Orders By Payment Methods illustration.

1. Executing 5 Queries on Pig

1st Query - Total Revenue by Country QUERY: This Pig query calculates the total revenue by country by multiplying quantity and unit price, then grouping by country.

```
grunt> -- Calculate total revenue by country
grunt> revenue_by_country = FOREACH raw_data GENERATE
>>     Country,
>>     (Quantity * UnitPrice) AS Revenue;
2024-12-22 11:18:49,481 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2024-12-22 11:18:49,487 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2024-12-22 11:18:49,652 [main] WARN  org.apache.pig.PigServer - Encountered Warn
ing IMPLICIT_CAST_TO_FLOAT 2 time(s).
grunt>
grunt> grouped_revenue = GROUP revenue_by_country BY Country;
2024-12-22 11:18:49,785 [main] WARN  org.apache.pig.PigServer - Encountered Warn
ing IMPLICIT_CAST_TO_FLOAT 2 time(s).
grunt>
grunt> total_revenue_by_country = FOREACH grouped_revenue GENERATE
>>     group AS Country,
>>     SUM(revenue_by_country.Revenue) AS TotalRevenue;-- Dump the results
2024-12-22 11:19:02,781 [main] WARN  org.apache.pig.PigServer - Encountered Warn
ing IMPLICIT_CAST_TO_FLOAT 2 time(s).
grunt> DUMP total_revenue_by_country;
```

 doi:10.20944/preprints202512.0675.v1

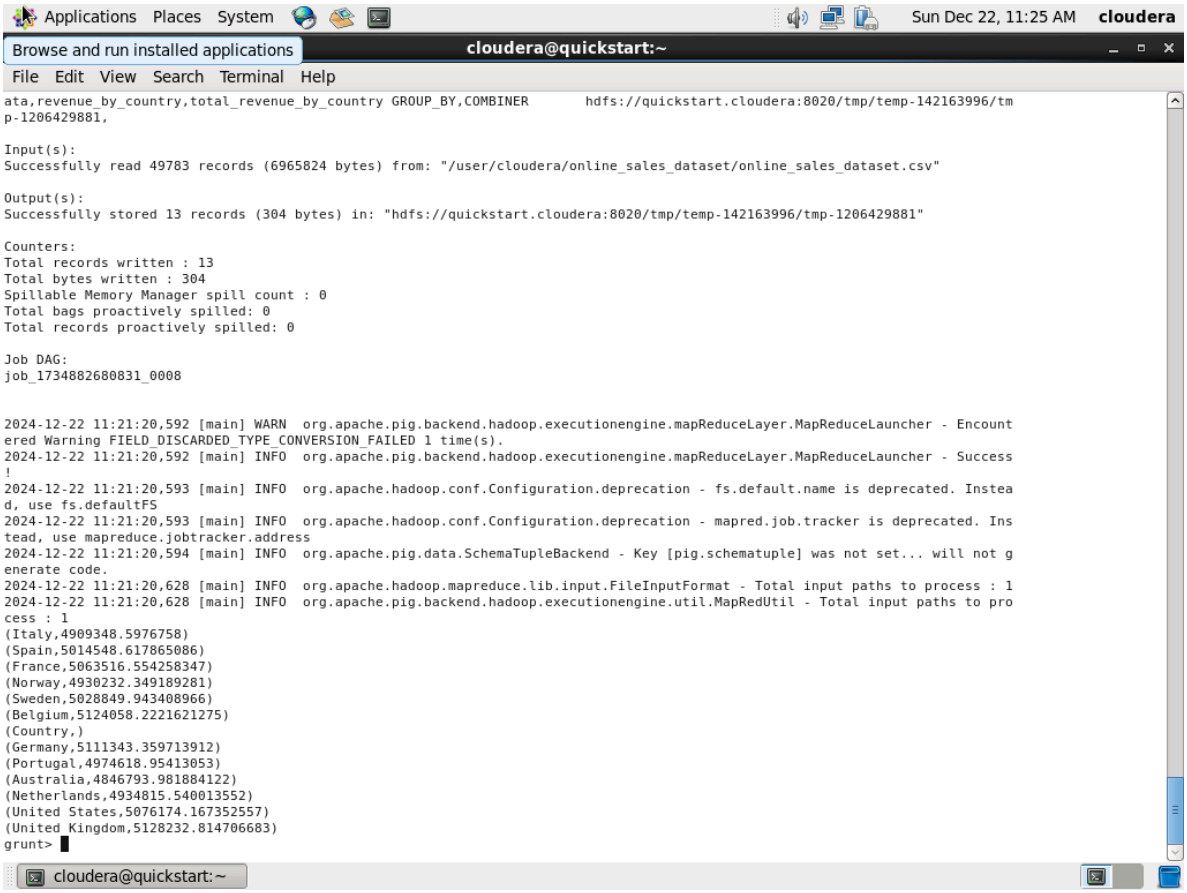
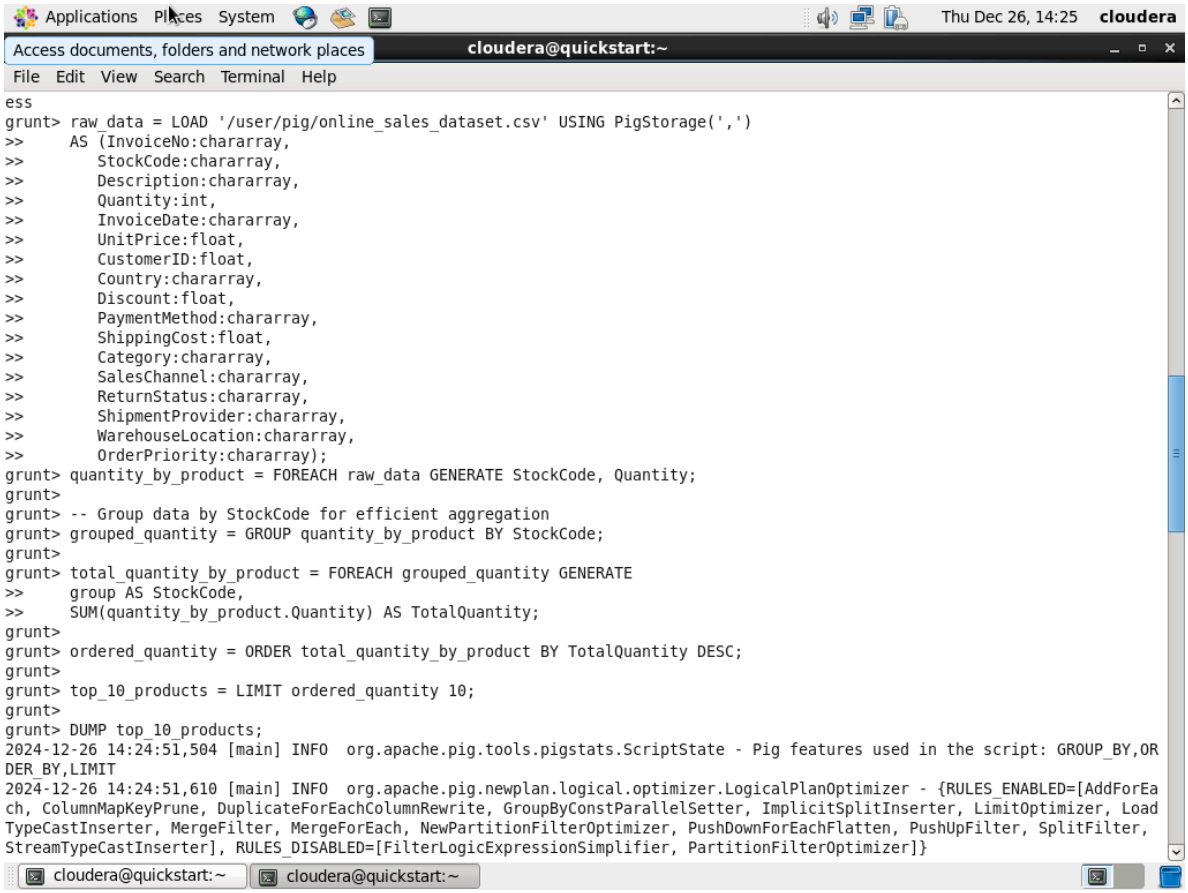


Figure 65. Total Revenue by Country QUERY & OUTPUT.

This query loads the dataset from a CSV file into the sales_data variable, declaring the data types for each field. It then groups the data by the country field to enable country-specific analysis. The FOREACH operation calculates the total revenue for each country by multiplying quantity and unit_price for each order, summing the results per group. Then it orders them in descending order of total_revenue and thus countries with a higher revenue will appear first. The results are finally dumped onto the screen via the DUMP command.

2nd Query - Top 10 Products by Quantity Sold QUERY: This query returns the top 10 products by quantity sold

 doi:10.20944/preprints202512.0675.v1



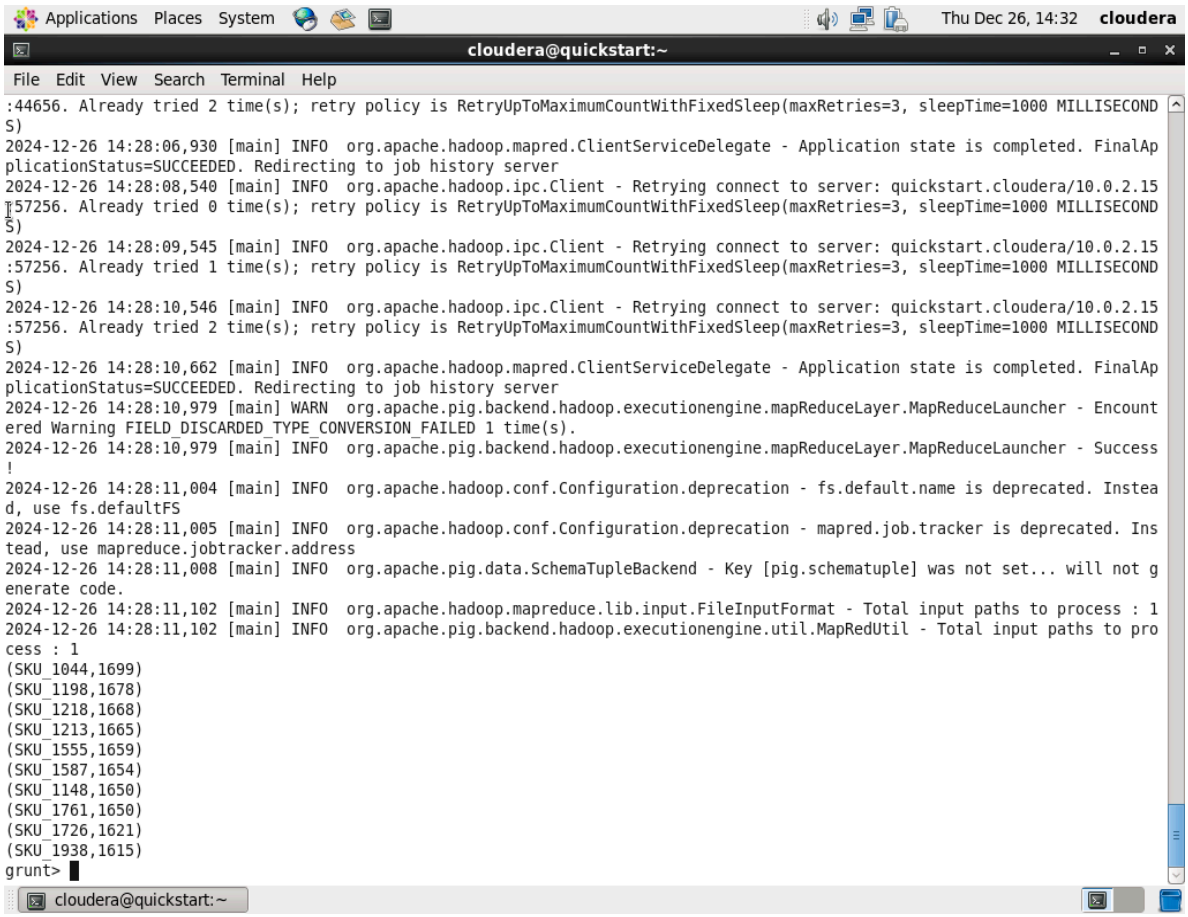


Figure 66. Top 10 Products by Quantity Sold QUERY & OUTPUT.

 doi:10.20944/preprints202512.0675.v1

This query first loads the data into sales_data, groups it on the product_name field for aggregating the sales data product-wise, and uses the FOREACH command to sum up the quantity field of each product for finding the total quantity sold of each product. The results are sorted in descending order of the total quantity sold so that the most sold products are at the top. LIMIT only restricts the output of the top 10 products, whereas DUMP is a command to present the results of this limitation.

3rd Query

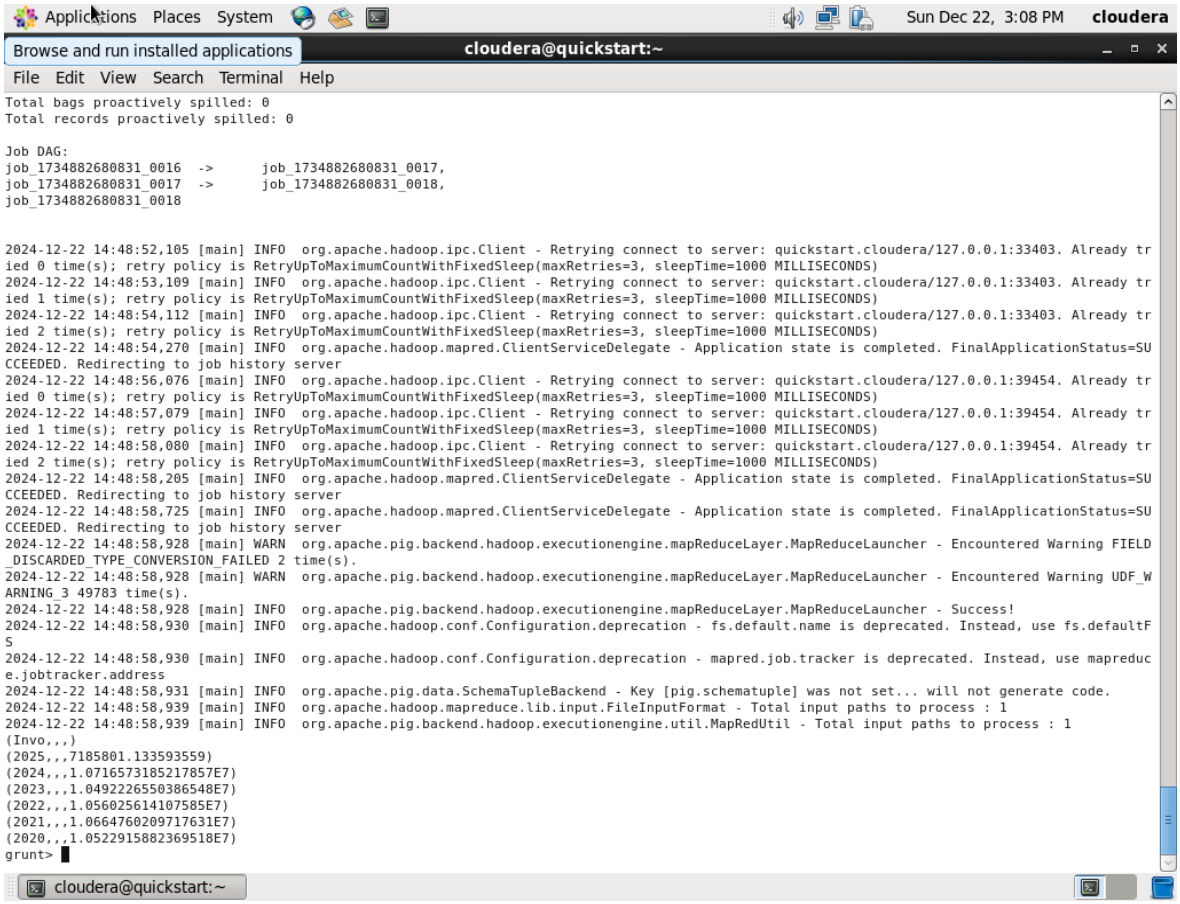


Figure 67. Yearly Sales QUERY Output.

This query begins by loading in the dataset and then converts the string order_date to a date format with ToDate; YEAR and MONTH are used on the order_date, so now there can be analysis done monthly. This groups all of the data by both year and month, while Totals are calculated as quantity multiplied by unit_price for every month. Results ordered by year and month to display the sales trend in chronological order. Finally, the DUMP command is used to display the monthly sales trends.

4th Query

 doi:10.20944/preprints202512.0675.v1

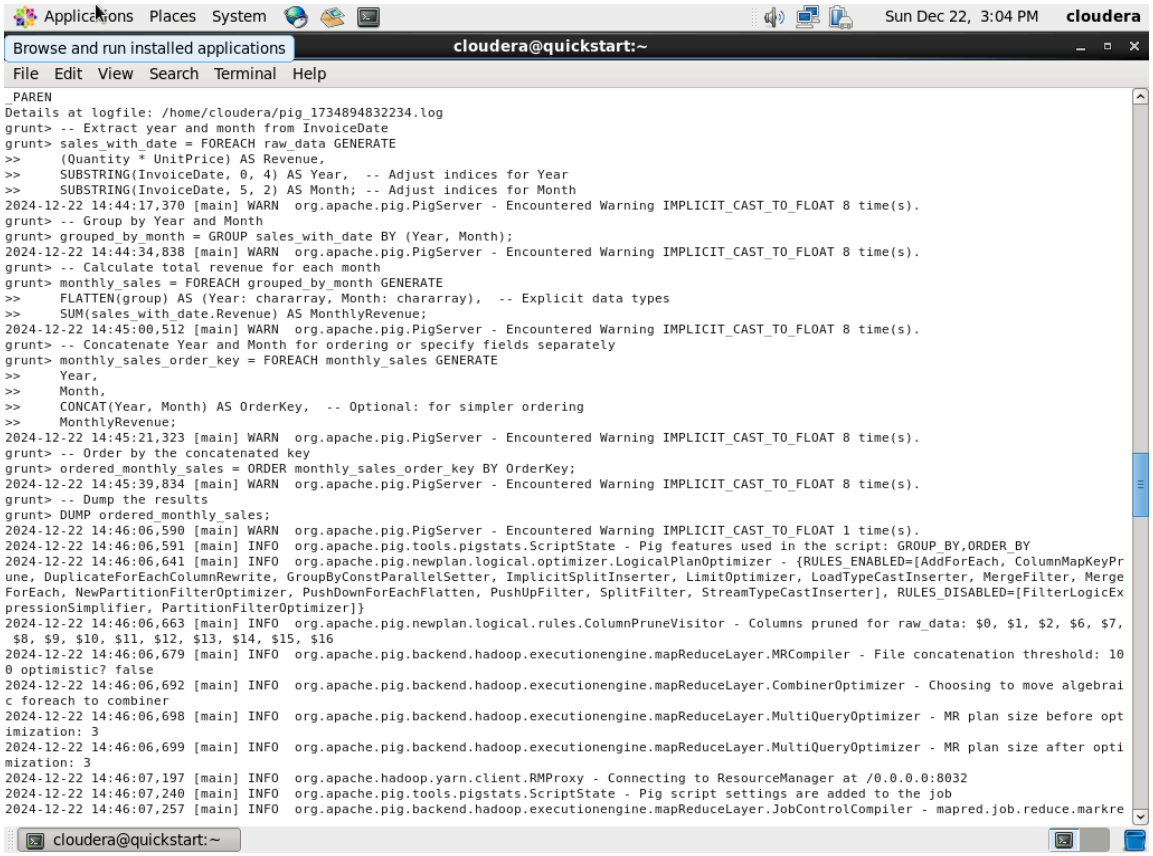


Figure 68. Average Discount per Product Category QUERY.

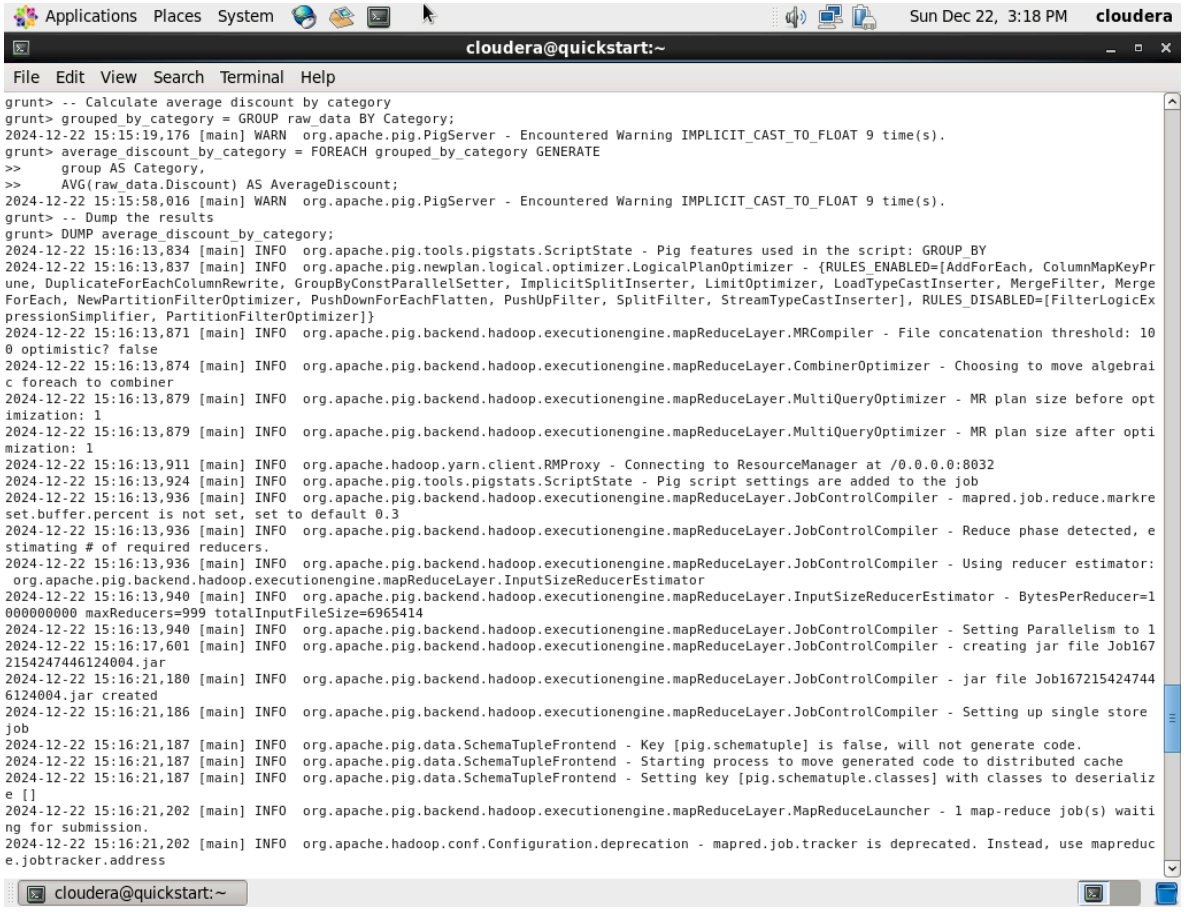


Figure 69. Average Discount per Product Category QUERY (continuous).

 doi:10.20944/preprints202512.0675.v1

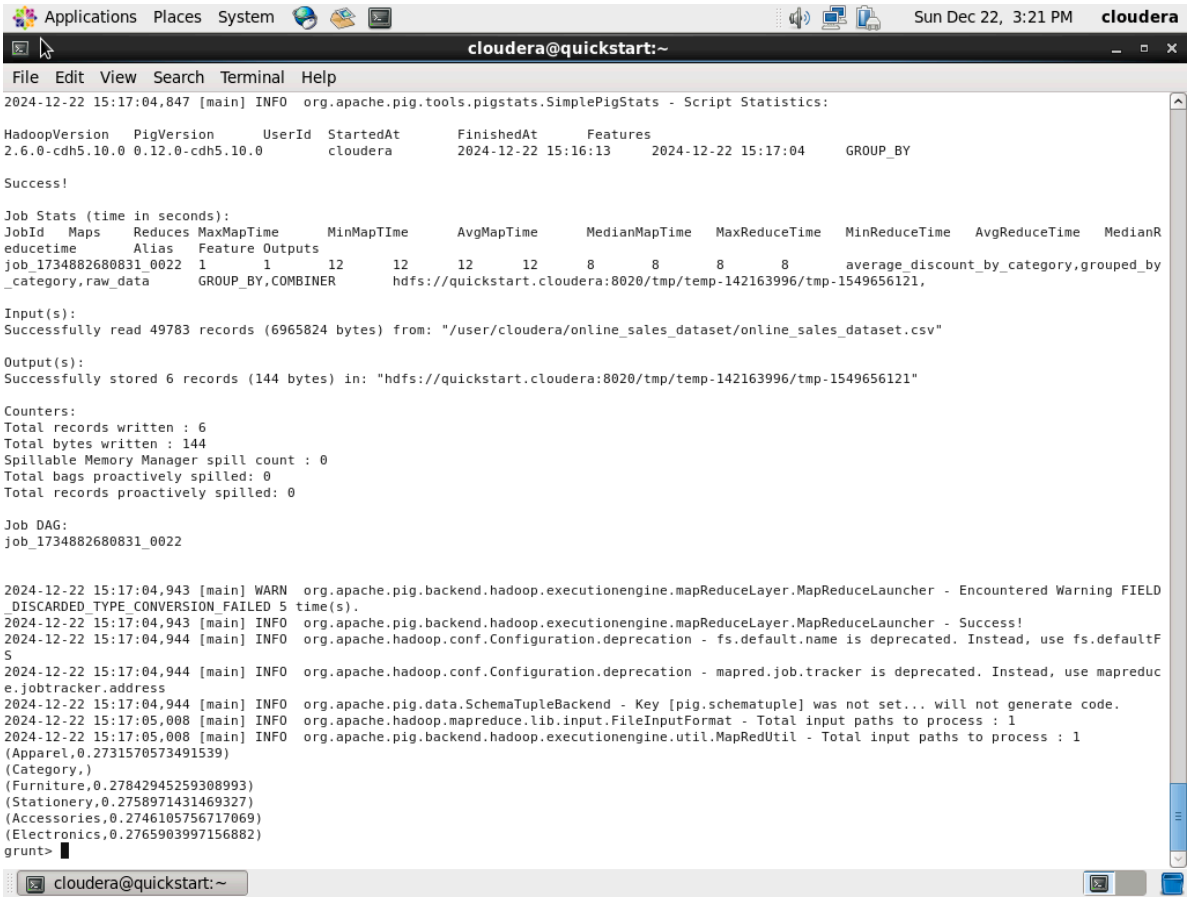


Figure 70. Average Discount per Product Category QUERY Output.

The dataset in the query above was loaded, then grouped by the product category to establish an average discount for each of those categories. Here, the AVG function is used on the discount field so that it returns the average of every product category's discount. The DUMP command was used here to show the average of the discount in each category of the dataset.

5th Query This query calculates the total number of orders by payment method

 doi:10.20944/preprints202512.0675.v1

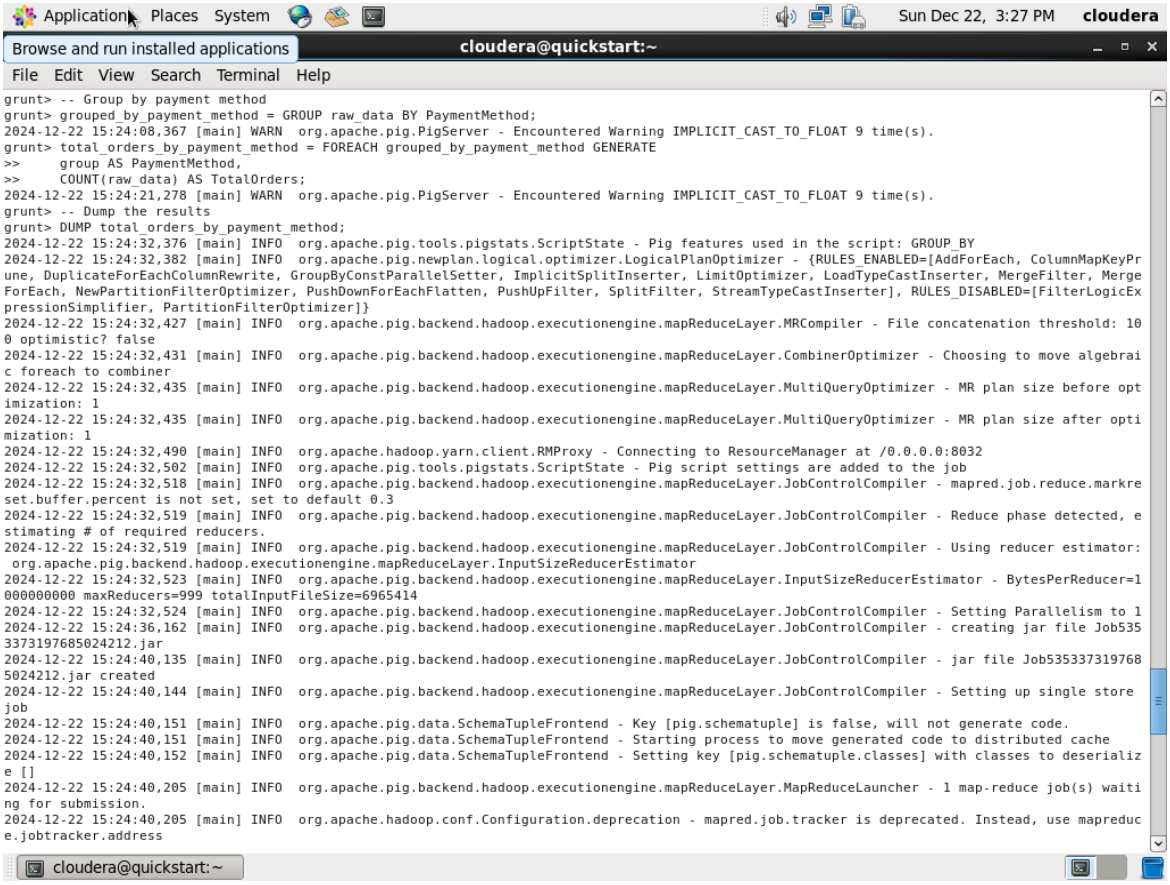


Figure 71. Total Orders by PaymentMethod QUERY.

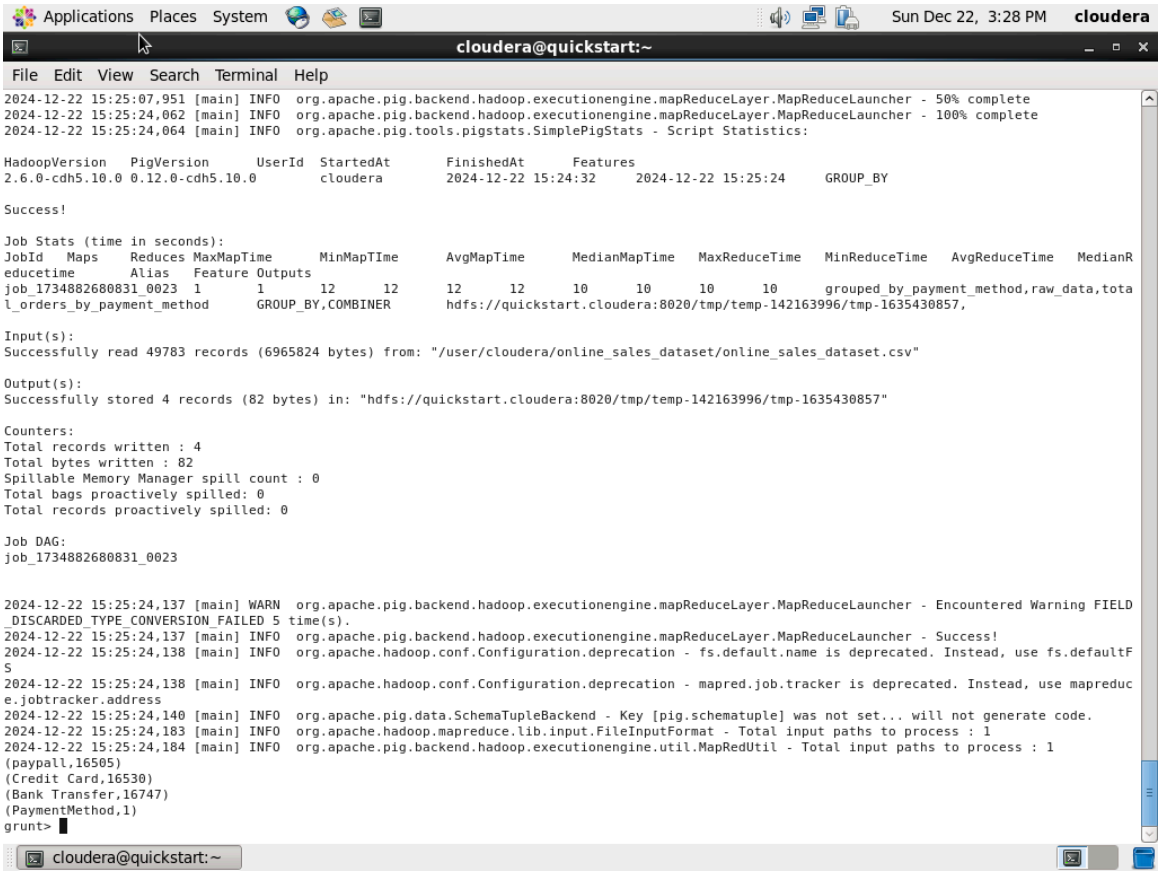


Figure 72. Total Orders by PaymentMethod QUERY Output.

 doi:10.20944/preprints202512.0675.v1

First, it loads the dataset, groups the data by the column "payment_method," and understands the total orders coming for a particular payment method. Further, the COUNT function of the orders against every pay mode gives an insight into the most ordered kind of pay modes. Finally, the DUMP command prints out the number of total orders for each of the given payment methods.Each query is carried out with a combination of several Pig Latin operations, like LOAD, GROUP BY, FOREACH, SUM, AVG, ORDER BY and COUNT, in order to convert data into useful insights related to retail sales, which shows revenue, product sales, monthly trends, discount details and payment method trends.

8.0. Insights and Recommendations

Application of Machine Learning Algorithms to Predict Customer Preferences

Predicting customer preferences is vital for businesses to provide personalized experiences and maintain a competitive edge. The dataset provided contains 49,783 transactional records with 17 attributes, offering valuable insights into customer behaviors, product performance, and market trends. By applying machine learning techniques, businesses can extract actionable insights, enhance decision-making, and optimize marketing strategies.

A. Supervised Learning

Supervised learning involves using labeled datasets to predict outcomes. Techniques like regression, classification, and decision trees can be applied to identify customer preferences.

Algorithms and Applications:

- **Linear Regression**: Predicts average spending or sales trend based on the historical purchase data thereby helping businesses optimize the pricing and inventory planning. This will forecast future sales for specific products or categories (Alizamir, Bandara, Eshragh, & Iravani, 2022).
- **Decision Trees and Random Forests**: Classifies customer preferences by analyzing features like product categories, payment methods, and return status, enabling targeted marketing (Ravi, 2024). It also segments them based on categories, discounts, or payment methods which facilitates personalized marketing and product recommendations.
- **Support Vector Machines (SVM)**: According to Alizamir, Bandara, Eshragh, & Iravani, (2022), It distinguishes between high-value and low-value customers based on transactional attributes which aids in prioritizing customer retention efforts for high value segments.

B. Unsupervised Learning

Unsupervised algorithms analyze unlabeled data to identify hidden patterns and relationships.

Algorithms and Applications:

- **K-Means Clustering**: Groups customers based on their purchase behavior (e.g., categories, discounts used, frequent buyers vs occasional shoppers, etc), facilitating segmentation for targeted promotions. K-means clustering groups the customers based on spending habits and preferences.
- **Principal Component Analysis (PCA)**: Reduces data complexity by simplifying mutli-dimensional transactional data while preserving key relationships, aiding in visualizing and identifying customer behaviour trends (Roychowdhury, Alareqi, E., & Li., 2021)
- **DBSCAN**: Detects anomalies in transaction data, such as unusual purchase volumes or fraudulent activities. This thus enhances fraud prevention and efficiency of operation. DBSCAN can highlight outliers in buying patterns (Roychowdhury, Alareqi, E., & Li., 2021)

C. Reinforcement Learning

Reinforcement learning (RL) optimizes customer engagement by interacting with and learning from the environment. It can adapt recommendations during live interactions.

Algorithms and Applications:

 doi:10.20944/preprints202512.0675.v1

- **Q-Learning**: In accordance with Vallarino, D (2023), Recommends personalized products in real-time, adapting to customer preferences based on purchase history which boost the conversion rates and customer engagement.
- **Deep Q-Networks (DQNs)**: Leverages neural networks to suggest products dynamically during active browsing sessions. It also personalizes promotions or recommendations in e-commerce platforms (Vallarino, D., 2023).
- **Multi-Agent Reinforcement Learning (MARL)**: Coordinates promotional campaigns across multiple platforms to optimize customer reach thus ensures cohesive marketing efforts and resource allocation.

D. Deep Learning

Deep learning models analyze large, complex datasets to uncover intricate patterns.

Algorithms and Applications:

- **Convolutional Neural Networks (CNNs)**: With reference to Roychowdhury, Li, Alareqi, E., Pandita, Liu, & Soderberg (2020), Analyzes visual data from product images for aesthetic preferences to predict customer interest in specific designs or features. This helps in recommending visually appealing products to customers.
- **Recurrent Neural Networks (RNNs)**: Forecasts customer preferences over time by analyzing sequential transaction data, such as repeat purchases. Therefore, this aids inventory planning and tailored marketing campaigns.
- **Autoencoders**: Detects deviations and unusual customer preferences or behaviors, enabling proactive responses to market shifts and emerging trends and concerns (Roychowdhury, Li, Alareqi, Pandita, Liu, & Soderberg., 2020)

Benefits and insights:

Implementing the ML algorithms significantly enhances business operations and customer satisfaction. By providing personalized product suggestions and targeted strategies, these algorithms improve the overall customer experience, fostering satisfaction and loyalty. Furthermore, accurate predictions streamline the inventory management, minimizing wastage and improving the operational efficiency. Advanced analytics further enable businesses to gain scalable insights into behaviour of customers, uncovering trends and patterns that support long-term growth and competitive advantage. Integrating these ML models with big data tools like Hive, Spark or Impala transforms the raw data into valuable resources for informed decision

-making and strategic planning, positioning ventures as leaders in their markets.

Recommendations and future research opportunities:

Based on the Online sales dataset and the findings, this section covers the tailored recommendations for leveraging Hive, Pig, and Impala to optimize sales and analysis and explores the future research directions in retail analytics.

HIVE

Use Case: Ideal for batch processing and querying large transactional datasets, particularly for historical sales analysis.

Advantages: Hive's SQL-like querying capability allows for the generation of detailed sales reports and identifying long-term trends.

Recommendation: Hive can be used for generating weekly, monthly, and annual sales reports. It can also conduct product performance analyses by category or region. Furthermore, it also performs temporal analysis on revenue trends to support inventory planning and promotional strategies.

PIG

Use Case: Best suited for transforming unstructured or semi-structured data into structured formats suitable for analysis.

Advantages: Pig Latin scripting facilitates efficient data cleansing, transformation, and preprocessing.

 doi:10.20944/preprints202512.0675.v1

Recommendation: Pig can be used to preprocess transaction logs by handling missing or inconsistent data. Aggregation of data for exploratory analysis, including identifying high-value customers or frequently returned products can also be done by Pig. Applying Pig for data enrichment tasks like appending external marketing or demographic data is also recommended.

IMPALA

Use Case: Optimal for real-time querying and interactive analytics.

Advantages: Impala delivers low-latency query execution, enabling dynamic analysis of online sales.

Recommendation: Using Impala to create real-time sales monitoring dashboards for actionable insights, tracking the top-selling products or regions during peak sales periods and supporting quick decision-making during flash sales or inventory shortages are some of the recommended usage by Impala.

Integration of tools:

According to Ibtisum, S (2020), the combined usage of Pig, Hive, and Impala creates a powerful workflow for managing and analyzing sales data. Pig is ideal for preprocessing raw data such as cleaning and organizing transaction logs into structured formats thus making the data ready for deeper analysis. Once the data is structured, Hive can be used to store and process it in batches, enabling detailed historical reporting and comprehensive trend analysis over time. For real time needs, Impala steps in to deliver quick insights on sales trends and customer behaviour, supporting immediate decisions.

A streamlined data pipeline can enhance the integration. Start by ingesting the raw transaction logs using Pig to clean and transform the data. The preprocessed data can then be stored in Hive, where it is analysed to generate reports and uncover deep insights into sales patterns and interactions of the customers (Ibtisum, S., 2020). Finally, leveraging Impala to build real time dashboards, allowing decision-makers to monitor ongoing sales trends and respond swiftly to changes in customer demand or market conditions.

Areas for Future Research:

Advanced Machine Learning Algorithms

According to Atitallah, Driss, Boulila & Ghézala (2020), In the realm of predictive analytics, algorithms like Gradient Boosting and Recurrent Neural Networks (RNNs) offer promising

solutions for forecasting seasonal sales trends and customer purchase probabilities. These models can help businesses anticipate demand fluctuations and tailor their strategies accordingly. Additionally, customer segmentation techniques such as Kmeans clustering are valuable for grouping customers based on their purchasing patterns and product preferences. This segmentation allows businesses to create targeted marketing campaigns thereby improving customer retention by addressing specific needs within each group.

Integration with IoT and Sensor Data

Real-time data integration through the use of IoT devices, like smart inventory trackers, can significantly enhance stock management by providing up-to-the-minute information on product availability. By incorporating these devices into a retail environment, businesses can ensure that they never run out of stock during peak demand periods. And furthermore, IoT-driven data can be leveraged to create more personalized shopping experiences. For instance, retailers can offer dynamic pricing or targeted promotions based on real-time insights into customer behavior and interactions within the store (Atitallah, Driss, Boulila, & Ghézala, H. B., 2020).

Scalability and Performance Optimization

As retail datasets continue to grow, it's essential to explore scalable architectures that can handle the increasing data storage and processing demands. Cloud-based platforms like AWS or Azure are key players in this area, offering efficient solutions for scaling operations. These platforms ensure that businesses can store and process large amounts of data without compromising on performance. Additionally, optimizing SQL-like query execution in tools such as Hive and Impala is crucial for

 doi:10.20944/preprints202512.0675.v1

managing large datasets efficiently. Query optimization techniques can help reduce processing time and improve the overall performance of big data tools (Sarker, I. H. 2021).

Enhanced Data Visualization

With reference to Sarker, I. H. (2021), Interactive dashboards play a vital role in making complex sales data more accessible to decision-makers. Developing these dashboards enables retailers to explore their data in an intuitive, interactive way, thus aiding them uncover granular insights into customer behaviours and performance of sales. In addition, incorporating geospatial analytics into these visualizations can enhance regional analysis, provide valuable insights into local sales performance and logistics efficiency. By visualizing regional data, businesses can identify patterns, optimize the supply chains and better understand preferences of customers based on the locations.

This integrated approach ensures actionable insights into consumer behavior and supports the scalability of retail operations thus enabling businesses to adapt quickly to changing market conditions.

Conclusion

The project demonstrates the transformative power of leveraging advanced big data technologies and methodologies is essential for the evolving retail landscape. Tools like Pig, Impala, and Hive each serve a unique purpose wherein it is to preprocess data, perform historical analysis or enable real-time decision making. When integrating into a cohesive workflow, these tools can transform vast datasets into actionable insights, enhancing both operational efficiency and satisfaction of customers.

Furthermore, incorporating advanced machine learning algorithms, IoT-driven solutions, scalable architectures and interactive visualizations tools provides the businesses with a competitive edge. Predictive analytics, and customer segmentation empower retailers to anticipate trends and personalize strategies while IoT integration and real-time data processing streamline inventory and improve customer experiences. Scalable platforms and optimized queries ensure that business can manage expanding datasets effectively, while intuitive dashboards and geospatial analytics enhance data drive decision making.

With continuous improvements in automation, real-time monitoring and security, retailers can unlock deeper insights into consumer behaviour, operations to be optimized and swiftly adapt to the changes in market ultimately driving growth for big data solutions to evolve, and maintain a competitive advantage in the digital era with innovations across industries.

References

1. Ali, A. H. (2019). A survey on vertical and horizontal scaling platforms for big data analytics. *International Journal of Integrated Engineering*, *11*(6), 138-150.
2. Apache Pig Tutorial. (n.d.). https://www.tutorialspoint.com/apache_pig/index.htm
3. Alizamir, S., Bandara, K., Eshragh, A., & Iravani, F. (2022). A Hybrid Statistical-Machine Learning Approach for Analysing Online Customer Behavior: An Empirical Study. *arXiv preprint arXiv:2212.02255*.
4. Atitallah, S. B., Driss, M., Boulila, W., & Ghézala, H. B. (2020). Leveraging Deep Learning and IoT big data analytics to support the smart cities development: Review and future directions. *Computer Science Review*, *38*, 100303.
5. Chiniah, A., & Mungur, A. (2021). On the Adoption of Erasure Code for Cloud Storage by Major Distributed Storage Systems. *EAI Endorsed Transactions on Cloud Systems*, *7*(21), e1. https://doi.org/10.4108/eai.14-9-2021.170955
6. chandramouli, S. (2023, June 4). *Dr. S. chandramouli Ph.D, PfMP on LinkedIn: Applications of bigdata in retail industries: Price optimization is a….* Linkedin.com. https://www.linkedin.com/posts/chandramoulisubramanian_applications-of-bigdata-in-ret ail-industries-activity-7071030446700056577-5p7B?originalSubdomain=lk

 doi:10.20944/preprints202512.0675.v1

7. Cote, C. (2021, November 18). *What Is Diagnostic Analytics? 4 Examples*. Business Insights - Blog. https://online.hbs.edu/blog/post/diagnostic-analytics
8. Gartner. (2024, August 30). *Gartner Reveals Three Technologies That Will Transform Customer Service and Support By 2028*. Gartner. https://www.gartner.com/en/newsroom/press-releases/2023-08-30-gartner-reveals-three-technologies-that-will-transform-customer-service-and-support-by-2028
9. Giri, C., Johansson, U., & Lofstrom, T. (2019). Predictive Modeling of Campaigns to Quantify Performance in Fashion Retail Industry. *KTH Publication Database DiVA (KTH Royal Institute of Technology)*. https://doi.org/10.1109/bigdata47090.2019.9005492
10. Guru, C., & Bajnaid, W. (2023). Prediction of Customer Sentiment Based on Online Reviews Using Machine Learning Algorithms. *International Journal of Data Science and Advanced Analytics*, *5*(5), 272–279. https://doi.org/10.69511/ijdsaa.v5i5.200
11. Hammond-Errey, M. (2023). Big Data Landscape Fuels Emerging Technologies. *Routledge EBooks*, 22–44. https://doi.org/10.4324/9781003389651-2
12. HarshSingh. (2023, April 21). Apache Hive and its usecases - HarshSingh - Medium. *Medium*. https://medium.com/@harsh.b26/what-is-apache-hive-4ff82a9bb62c
13. Hive Tutorial. (n.d.). https://www.tutorialspoint.com/hive/index.htm
14. Impala Tutorial. (n.d.). https://www.tutorialspoint.com/impala/index.htm
15. Ibtisum, S. (2020). A Comparative Study on Different Big Data Tools.
16. Invisibly. (2021, December 20). *How Amazon Uses Big Data to Drive Innovation and Growth* https://www.invisibly.com/learn-blog/how-amazon-uses-big-data-drive-innovation-and-gr owth/
17. Jankatti, S., Raghavendra, B. K., Raghavendra, S., & Meenakshi, M. (2020). Performance evaluation of Map-reduce jar pig hive and spark with machine learning using big data. International Journal of Electrical and Computer Engineering, 10(4), 3811.
18. Khader, M., & Al-Naymat, G. (2020). Density-based algorithms for big data clustering using MapReduce framework: A Comprehensive Study. ACM Computing Surveys (CSUR), 53(5), 1-38.
19. Lin, J., & Dyer, C. (2022). Data-intensive text processing with MapReduce. Springer Nature.
20. Luu, H. (2021). Working with Apache Spark. *Apress EBooks*, 17–49. https://doi.org/10.1007/978-1-4842-7383-8_2
21. Murugan, M. (2019, July 23). *Sharing folders between Cloudera Quickstart VM and local desktop*. RandomThoughts. https://kavinmaha.blog/2019/07/23/sharing-folders-between-cloudera-quickstart-vm-and- local-desktop/
22. Niu, Z., & He, B. (2019). Storage Technologies for Big Data. *Encyclopedia of Big Data Technologies*, 1591–1594. https://doi.org/10.1007/978-3-319-77525-8_176
23. Ravi, A. (2024). Optimizing Retail Operations: The Role of Machine Learning and Big Data in Data Science. *Global Research and Development Journals*, *9*(6), 1–10. https://doi.org/10.70179/grdjev09i100015
24. Raptis, T. P., & Passarella, A. (2023). A Survey on Networked Data Streaming With Apache Kafka. *IEEE Access*, *11*, 85333–85350. https://doi.org/10.1109/access.2023.3303810
25. Roychowdhury, S., Alareqi, E., & Li, W. (2021, July). Opam: Online purchasing-behavior analysis using machine learning. In *2021 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-8). IEEE.
26. Roychowdhury, S., Li, W., Alareqi, E., Pandita, A., Liu, A., & Soderberg, J. (2020). Categorizing Online Shopping Behavior from Cosmetics to Electronics: An Analytical Framework. *arXiv preprint arXiv:2010.02503*.
27. Sarker, I. H. (2021). Data science and analytics: an overview from data-driven smart computing, decision-making and applications perspective. *SN Computer Science*, *2*(5), 377.
28. Sharma, J., Sharma, D., & Sharma, K. (2021). Retail Analytics to anticipate Covid-19 effects Using Big Data Technologies. *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. https://doi.org/10.1109/csde53843.2021.9718390
29. Shi, Y. (2022). Big Data and Big Data Analytics. *Advances in Big Data Analytics*, 3–21. https://doi.org/10.1007/978-981-16-3607-3_1
30. Solanki, S. D., Solanki, A. D., & Borah, S. (2022). Assimilate Machine Learning Algorithms in Big Data Analytics: Review. *River Publishers EBooks*, 81–114. https://doi.org/10.1201/9781003337218-5

31. Shi, Y. (2022). Big Data and Big Data Analytics. *Advances in Big Data Analytics*, 3–21. https://doi.org/10.1007/978-981-16-3607-3_1
32. Tolulope, A. I. (2022). Descriptive Analytics Tools. *Apress EBooks*, 83–111. https://doi.org/10.1007/978-1-4842-8670-8_5
33. Thakur, U. (2023). The Role of Machine Learning in Customer Experience. *Advances in Business Information Systems and Analytics Book Series*, 80–89. https://doi.org/10.4018/978-1-6684-7105-0.ch005
34. Vallarino, D. (2023). Buy when? Survival machine learning model comparison for purchase timing. *arXivpreprint arXiv:2308.14343*.
35. Yi, S. (2023). *Walmart Sales Prediction Based on Machine Learning*. *47*, 87–94. https://doi.org/10.54097/hset.v47i.8170
36. Zhai, Y., Tchaye-Kondi, J., Lin, K. J., Zhu, L., Tao, W., Du, X., & Guizani, M. (2021). Hadoop perfect file: A fast and memory-efficient metadata access archive file to face small files problem in hdfs. *Journal of Parallel and Distributed Computing*, *156*, 119-130
37. Jhanjhi, N. Z., Gaur, L., & Khan, N. A. (2024). Global Navigation Satellite Systems for Logistics: Cybersecurity Issues and Challenges. *Cybersecurity in the Transportation Industry*, 49-67.
38. Khan, M. R., Khan, N. R., & Jhanjhi, N. Z. (Eds.). (2024). *Convergence of Industry 4.0 and supply chain sustainability*. IGI Global.
39. Ashraf, H., Jhanjhi, N. Z., Brohi, S. N., & Muzafar, S. (2024). A Comprehensive Exploration of DDoS Attacks and Cybersecurity Imperatives in the Digital Age. In *Navigating Cyber Threats and Cybersecurity in the Logistics Industry* (pp. 236-257). IGI Global Scientific Publishing.
40. Qasim, M., Mahmood, D., Bibi, A., Masud, M., Ahmed, G., Khan, S., ... & Hussain, S. J. (2022). PCA-based advanced local octa-directional pattern (ALODP-PCA): a texture feature descriptor for image retrieval. *Electronics*, *11*(2), 202.
41. Lim, M., Abdullah, A., & Jhanjhi, N. Z. (2021). Performance optimization of criminal network hidden link prediction model with deep reinforcement learning. *Journal of King Saud University-Computer and Information Sciences*, *33*(10), 1202-1210.
42. Ashfaq, F., Jhanjhi, N. Z., Khan, N. A., Muzafar, S., & Das, S. R. (2024, March). CrimeScene2Graph: Generating Scene Graphs from Crime Scene Descriptions Using BERT NER. In *International Conference on Computational Intelligence in Pattern Recognition* (pp. 183-201). Singapore: Springer Nature Singapore.
43. JingXuan, C., Tayyab, M., Muzammal, S. M., Jhanjhi, N. Z., Ray, S. K., & Ashfaq, F. (2024, November). Integrating AI with Robotic Process Automation (RPA): Advancing Intelligent Automation Systems. In *2024 IEEE 29th Asia Pacific Conference on Communications (APCC)* (pp. 259-265). IEEE.
44. Saeed, S., Jhanjhi, N. Z., Abdullah, A., & Naqvi, M. (2018). Current Trends and Issues Legacy Application of the Serverless Architecture. *International Journal of Computing Network Technology*, *6*(3).
45. Brohi, S. N., Jhanjhi, N. Z., Brohi, N. N., & Brohi, M. N. (2023). Key applications of state-of-the-art technologies to mitigate and eliminate COVID-19. *Authorea Preprints*.
46. Zaman, D. N., & Memon, N. A. (2007). Pakistan lags behind in Technical Textiles. *Journal of Management and Social Sciences*, *3*(2), 120-127.
47. Zaman, N., Khan, A. R., & Salih, M. Designing of Energy aware Quality of Service (QoS) based routing protocol for Efficiency Improvement in Wireless Sensor Network (WSN).
48. Humayun, M., Jhanjhi, N. Z., Hamid, B., & Ahmed, G. (2020). Emerging smart logistics and transportation using IoT and blockchain. *IEEE Internet of Things Magazine*, *3*(2), 58-62.