Adventist University of Central Africa
P.O. Box 2461 Kigali, Rwanda | www.auca.ac.rw | info@auca.ac.rw

Title: Distributed Multi-Model Analytics for E-commerce Data

Course: Big Data Analytics

Course code: MSDA 9215

Student: Pacifique MUYIRINGIRE 101102

Instructor: Temitope Oguntade

January 30, 2026

Contents

1. Introduction

1.1. Background and Context

The rapid expansion of digital platforms, particularly in e-commerce, has resulted in massive volumes of diverse and fast-moving data such as user profiles, product information, transactions, and clickstream events. Traditional relational databases often struggle to handle these demands, especially when systems must support both real-time operations and large-scale analytical workloads. As a result, organizations increasingly turn to distributed data architectures that can scale efficiently and adapt to varying data types and access patterns.

Modern e-commerce analytics relies on combining multiple specialized technologies rather than a single database solution. Document databases like MongoDB handle flexible user and product data, wide-column stores such as HBase manage large-scale transactional or time-based data, and processing frameworks like Apache Spark enable powerful batch analytics across systems. This project explores the design and implementation of such a distributed multi-model architecture using synthetic e-commerce data, highlighting key architectural choices, data modeling strategies, and analytical techniques that reflect real-world big data engineering practices.

1.2. Problem statement

E-commerce platforms generate large volumes of heterogeneous data that include user information, product details, and transactional records. Managing and analyzing this data using a single traditional database system is challenging due to scalability limitations, schema rigidity, and diverse query requirements. Operational and analytical workloads require different data access patterns, which are difficult to efficiently support within one system. Furthermore, integrating data from multiple sources for large-scale analytics presents additional complexity. This project addresses the need for a distributed multi-model architecture that combines multiple data storage technologies with a scalable processing framework to enable efficient storage, integration, and analytics of e-commerce data.

1.3. Project Objectives

The primary objective of this project is to design and implement a distributed multi-model data analytics system for e-commerce data using MongoDB, Apache HBase, and Apache Spark. The project aims to demonstrate how different big data technologies can be combined to efficiently store, process, and analyze large-scale heterogeneous datasets.

The specific objectives of the project are as follows:

1. **To design and generate a synthetic e-commerce dataset** that simulates realistic user, product, and transactional data suitable for large-scale analytics.
2. **To model and store semi-structured data in MongoDB**, focusing on flexible schema design, appropriate use of embedding and referencing, and efficient indexing strategies for operational queries.
3. **To design and implement a scalable data storage model in Apache HBase** for handling large volumes of transactional and time-oriented data, with an efficient row key and column family design.
4. **To process and analyze large-scale datasets using Apache Spark**, including data ingestion, cleaning, transformation, and batch analytics across multiple data sources.
5. **To perform integrated analytics across MongoDB and HBase data sources** using Spark, enabling cross-system insights such as customer behavior analysis and sales performance evaluation.
6. **To develop meaningful visualizations and business insights** based on the analytical results, including sales trends, customer segmentation, product performance, and conversion funnel analysis.
7. **To evaluate the scalability and limitations of the proposed architecture** and discuss potential improvements and future enhancements.

1.4. Scope of the Work

This project focuses on the design and implementation of a distributed multi-model data analytics system for e-commerce data using MongoDB, Apache HBase, and Apache Spark. The scope of the work covers the complete data lifecycle, from synthetic data generation and storage to large-scale analytics and visualization.

Within the scope of this project, synthetically generated datasets are used to represent users, products, and transactional records in an e-commerce environment. MongoDB is utilized to store semi-structured data such as user and product information, while Apache HBase is employed to store large volumes of transactional and time-oriented data. Apache Spark serves as the primary data processing engine, enabling batch analytics and integrated analysis across the different data storage systems.

The project includes data modeling, schema design, indexing strategies, batch data processing, Spark SQL analytics, and the development of visualizations to extract business insights. Emphasis is placed on architectural decisions, scalability considerations, and the justification of technology choices in relation to data characteristics and query patterns.

2. System Architecture and Workflow

2.1. High-level System Architecture

The system architecture of this project follows a **distributed multi-model design**, where different technologies are used together, each serving a specific role based on data characteristics and processing requirements. Rather than relying on a single database system, the architecture separates data storage and analytics responsibilities to improve scalability, flexibility, and performance.

At a high level, the architecture consists of four main layers:

1. **Data Generation Layer** – Responsible for generating synthetic e-commerce data, including users, products, and transactions.
2. **Data Storage Layer** – Composed of MongoDB and Apache HBase, each storing different types of data based on structure and access patterns.
3. **Data Processing Layer** – Implemented using Apache Spark to perform batch processing and analytics.
4. **Analytics and Visualization Layer** – Produces analytical results and visualizations to support business insights.

MongoDB stores semi-structured and frequently accessed data such as user profiles and product information, while HBase stores large volumes of transactional data optimized for high write throughput and scalable storage. Apache Spark acts as the central processing engine, integrating data from both storage systems to perform analytics and generate insights.

2.2. Data Flow and Processing Workflow

The workflow of the system follows a clear and sequential data pipeline:

1. **Synthetic Data Generation**
   A Python-based dataset generator creates realistic e-commerce data, including users, products, and transactions. This data simulates real-world e-commerce behavior and provides sufficient volume for analytics.
2. **Data Ingestion into Storage Systems**
   - User and product data are ingested into **MongoDB** as document-based collections.
   - Transactional data is ingested into **Apache HBase**, where it is stored in a wide-column format optimized for scalability.
3. **Data Ingestion into Apache Spark**
   Apache Spark reads data from MongoDB and HBase using appropriate connectors. Data from both sources is loaded into Spark DataFrames for further processing.
4. **Data Cleaning and Transformation**
   Spark performs data validation, normalization, and transformation tasks, ensuring consistent schemas and preparing the data for analytics.
5. **Batch Analytics and Aggregation**
   Spark executes batch analytics such as sales aggregation, customer behavior analysis, product performance evaluation, and conversion funnel analysis.
6. **Visualization and Insight Generation**
   Analytical results produced by Spark are used to generate visualizations and summaries that support business-level insights.

This workflow enables efficient handling of large datasets while maintaining flexibility and scalability across the system.

2.3. Role of Each Technology

2.3.1 MongoDB

MongoDB is used to store **semi-structured and operational data**, such as user profiles and product information. Its flexible document-based schema allows easy handling of evolving data structures. MongoDB supports fast read operations and aggregation queries, making it suitable for operational analytics and metadata storage.

2.3.2 Apache HBase

Apache HBase is used to store **large-scale transactional and time-oriented data**. Its wide-column design and distributed architecture enable high write throughput and horizontal scalability. HBase is particularly well suited for storing transaction histories that grow continuously over time.

2.3.3 Apache Spark

Apache Spark serves as the **central analytics engine** of the system. It integrates data from MongoDB and HBase, performs batch processing, executes Spark SQL queries, and enables large-scale analytics. Spark provides in-memory processing and distributed execution, making it ideal for complex analytical workloads.

3. Dataset Description

3.1. Dataset Generation Approach

The dataset used in this project is synthetically generated using a Python-based data generation script (dataset_generator.py). The script is designed to simulate realistic e-commerce behavior by generating users, products, categories, browsing sessions, and transactions over a configurable time period. Randomized yet controlled parameters are used to model user navigation flows, shopping cart behavior, inventory management, and transaction completion.

The generator initializes configurable parameters such as the number of users, products, sessions, and transactions, ensuring reproducibility through fixed random seeds. During execution, the script produces progress logs and a completion summary, confirming the successful generation of all data entities.

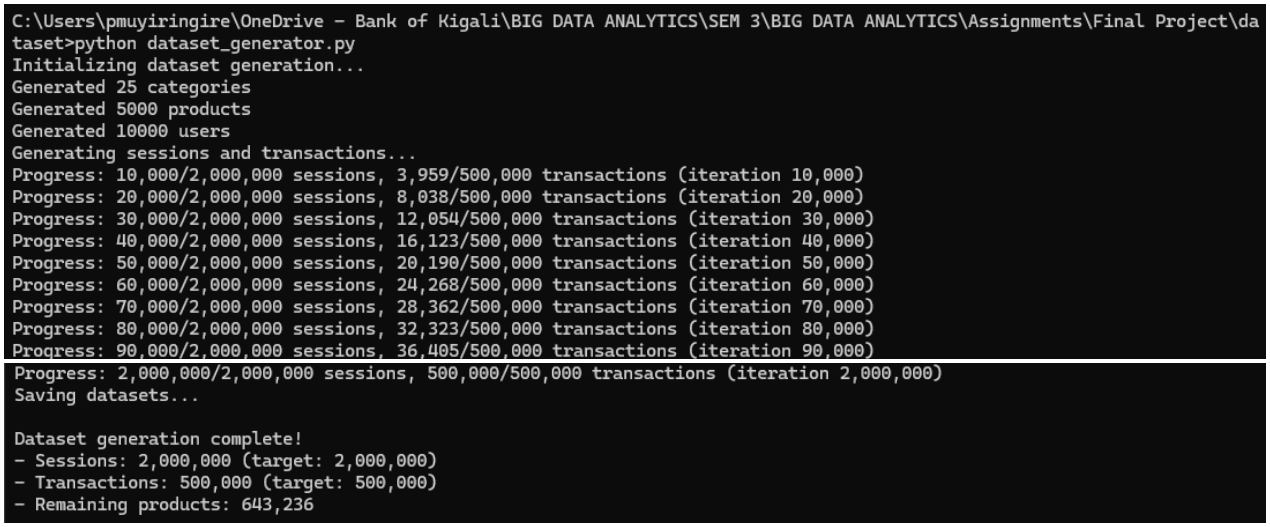

Figure 1: *Execution of the dataset generation script (dataset_generator.py) showing progress logs and final summary of generated users, products, sessions, and transactions***.**

After generation, the datasets are exported in JSON format to a structured directory for further ingestion into MongoDB and Apache HBase. Large session data is split into multiple files to support scalable processing and efficient loading.

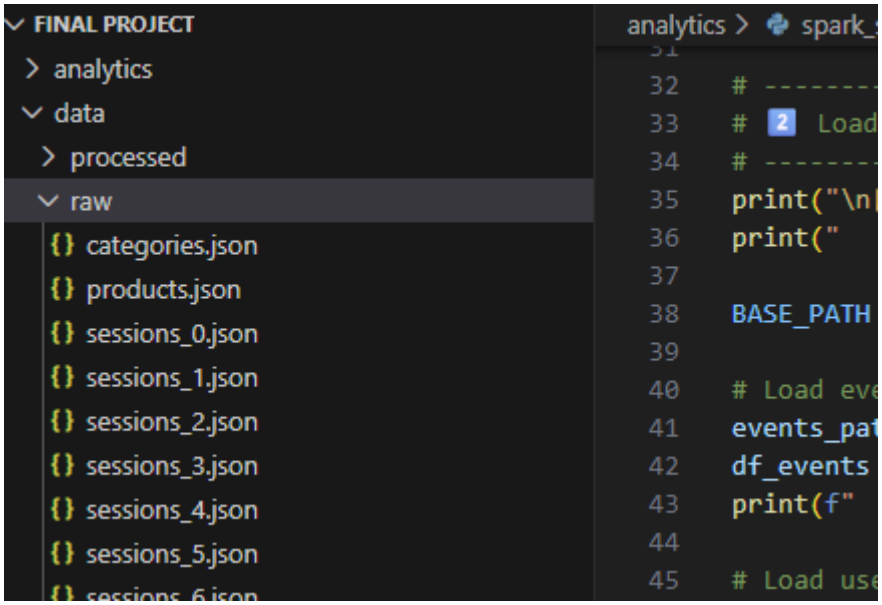

Figure 2: *Generated raw e-commerce datasets stored in the data/raw/ directory, including users, products, categories, transactions, and 2 Million sessions stored in chunked session files.*

3.2. Data Entities and Relationships

The generated dataset consists of several interconnected entities:

- **Users**, representing registered customers with geographic and activity metadata.

- **Products**, representing items available for sale, including pricing history and stock levels.
- **Categories**, organizing products into hierarchical groups.
- **Sessions**, representing user browsing behavior and interaction flows.
- **Transactions**, representing completed or in-progress purchases linked to users and sessions.

Each user may generate multiple sessions, and each session may lead to zero or more transactions. Transactions reference both users and products, enabling cross-entity analytics.

3.3. Data Characteristics

The dataset exhibits key big data characteristics, including high volume (millions of sessions and hundreds of thousands of transactions), structural variety (documents, nested attributes, and transactional records), and temporal distribution through timestamped events. These characteristics justify the use of distributed storage and processing technologies.

4. Data Modeling and Storage Design

4.1. Data Storage Strategy

A multi-model data storage strategy is adopted to efficiently manage heterogeneous data. MongoDB is used to store semi-structured entities such as users and products, while Apache HBase is used to store high-volume transactional data. Apache Spark is used as the analytics engine to integrate and process data from both systems.

4.2. MongoDB Data Model

User and product data are stored in MongoDB collections using a document-based schema. MongoDB's flexible structure supports nested fields such as geographic information and price history, making it suitable for evolving data models and operational queries.

Indexes are created on frequently queried fields such as user identifiers and product categories to improve query performance.

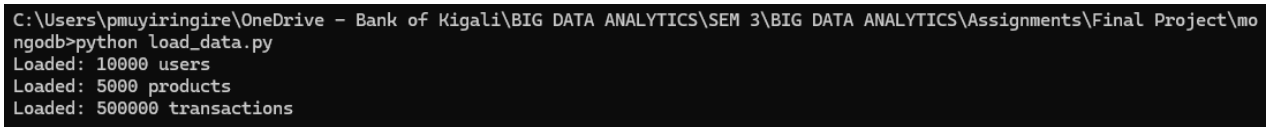

Figure 3: *Loading (users, products and transactions) datasets into MongoDB using (run_analytics.py) script in mongodb directory.*

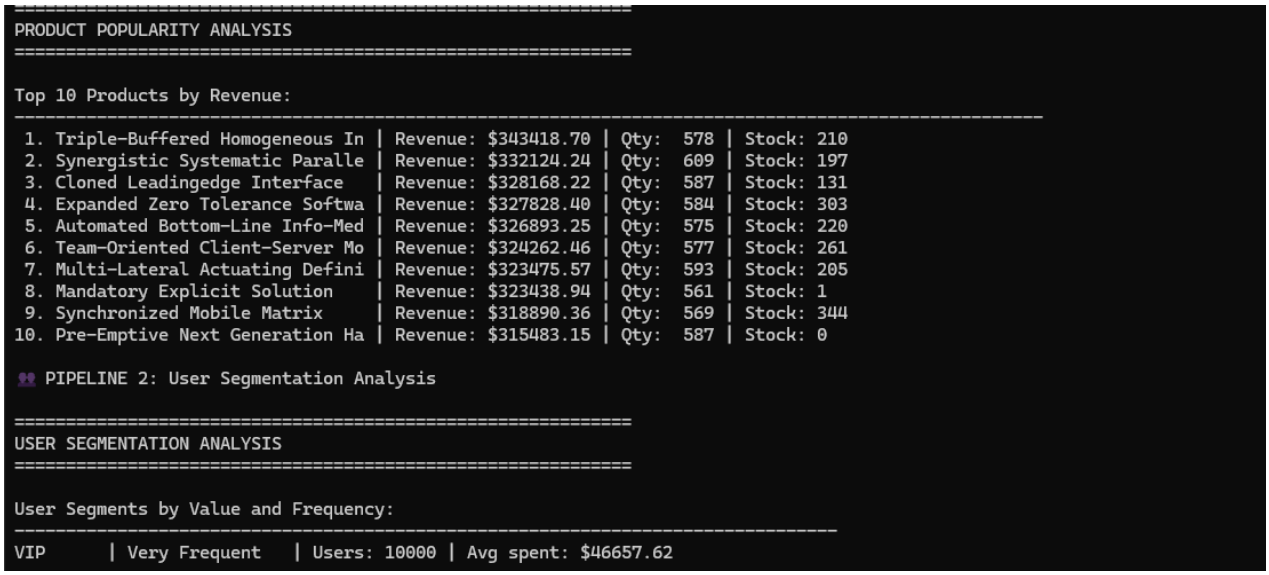

Figure 4: *This figure shows the results of MongoDB aggregation pipelines using (run_analytic.py) script analyzing product popularity by revenue and sales volume, along with basic user segmentation by purchasing frequency.*

4.3. HBase Data Model

Transactional data is stored in Apache HBase to support high write throughput and scalable storage. Each row represents a transaction, identified by a composite row key that includes a unique identifier and a timestamp. Column families separate transaction details, user references, and product information.

4.3.1. Schema Design

Table **sessions** was created with the below column families.

Column Family	Purpose	Example Columns
meta	Session metadata	user_id, timestamp
device	Device information	os, type
geo	Geographical data	city, country
stats	Session statistics	duration, pages

4.3.2. Implementation Details

4.3.2.1. Environment Setup

Step 1. Command to start hbase:

→ docker run -d --name hbase-ecommerce `

-p 2181:2181 `

-p 16010:16010 `

-p 9090:9090 `

harisekhon/hbase

Step 2. Command to open HBase Shell

➔ docker exec -it hbase-ecommerce hbase shell

```
C:\Users\pmuyiringire>docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED       STATUS          PORTS

      NAMES
6ff8bcf9306d   harisekhon/hbase                   "/entrypoint.sh"         2 days ago    Up 14 minutes   0.0.0.0:2181->
2181/tcp, [::]:2181->2181/tcp, 0.0.0.0:9090->9090/tcp, [::]:9090->9090/tcp, 0.0.0.0:16010->16010/tcp, [::]:16010->16010/
tcp   hbase-ecommerce
bfd94cff41c2   grafana/grafana:latest             "/run.sh"                4 weeks ago   Up 2 hours      0.0.0.0:3000->
3000/tcp, [::]:3000->3000/tcp
      grafana_viz
902f6a5a9e2a   timescale/timescaledb:latest-pg16  "docker-entrypoint.s…"   4 weeks ago   Up 2 hours      0.0.0.0:5432->
5432/tcp, [::]:5432->5432/tcp
      timescaledb
3762b11227fa   emqx/emqx:5.6.0                    "/usr/bin/docker-ent…"   4 weeks ago   Up 2 hours      0.0.0.0:1883->
1883/tcp, [::]:1883->1883/tcp, 0.0.0.0:18083->18083/tcp, [::]:18083->18083/tcp
      emqx_broker
```

Figure 5: *Dockerized HBase and supporting services running locally.*

Step 3. Command to create sessions table

➔ create 'sessions', 'meta', 'geo', 'device', 'stats', 'events'

Step 4. Command to start Thrift

➔ docker exec -d hbase-ecommerce bash -lc "nohup hbase thrift start -p 9090 --infoport 9095 >/tmp/thrift.log 2>&1 &"

Step 5. Run the sessions loader

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

  Progress: 20,100 sessions processed
  Progress: 30,100 sessions processed
  Progress: 40,100 sessions processed
  Progress: 50,100 sessions processed
  Progress: 60,100 sessions processed
  Progress: 70,100 sessions processed
  Progress: 80,100 sessions processed
  Progress: 90,100 sessions processed
✓ Loaded 100,000 sessions from sessions_19.json

✓ Total: 2,000,000 sessions loaded into HBase
PS C:\Users\pmuyiringire\OneDrive - Bank of Kigali\BIG DATA ANALYTICS\SEM 3\BIG DATA ANALYTICS\As
signments\Final Project>
```

Figure 6: *Data ingestion progress: 2,000,000 user sessions loaded into Hbase*

```
hbase(main):003:0> scan 'sessions', {
hbase(main):004:1*   FILTER => "SingleColumnValueFilter('meta','user_id',=,'binary:user_001589')",
hbase(main):005:1*   LIMIT => 10
hbase(main):006:1> }
ROW                                COLUMN+CELL
 sess_0000059e6b                   column=device:os, timestamp=1769196366517, value=
 sess_0000059e6b                   column=device:type, timestamp=1769196366517, value=
 sess_0000059e6b                   column=geo:city, timestamp=1769196366517, value=
 sess_0000059e6b                   column=geo:country, timestamp=1769196366517, value=
 sess_0000059e6b                   column=meta:timestamp, timestamp=1769196366517, value=
 sess_0000059e6b                   column=meta:user_id, timestamp=1769196366517, value=user_001589
 sess_0000059e6b                   column=stats:duration, timestamp=1769196366517, value=0
```

Figure 7: *Querying session data for a specific user using HBase's SingleColumnValueFilter.*

```
============================================================
STARTING IMPROVED BIG DATA ANALYSIS
============================================================

1. Loading Data...
  - Transactions Found: 500,000
  - Products Found: 5,000

2. Generating Product Recommendations...
  - Top Recommendations Preview:
+-----------+------------------+-----+
|product_ref|recommended_product|count|
+-----------+------------------+-----+
| prod_03503|         prod_04947|   24|
| prod_03594|         prod_03671|   24|
| prod_04947|         prod_03503|   24|
| prod_03671|         prod_03594|   24|
| prod_00505|         prod_00947|   22|
| prod_04387|         prod_01968|   22|
| prod_01968|         prod_04387|   22|
| prod_01081|         prod_00713|   22|
| prod_00557|         prod_02028|   22|
| prod_03054|         prod_02587|   22|
+-----------+------------------+-----+
only showing top 10 rows
```

Figure 8*. Analytics output showing product recommendations*

```
3. Running SQL Analytics...

  - Revenue by Payment Method:
+--------------+------+--------------+----------+
|payment_method|volume| total_revenue|avg_ticket|
+--------------+------+--------------+----------+
|   credit_card|125177|1.1739619958E8|    937.84|
|        paypal|125185|1.1667868482E8|    932.05|
| bank_transfer| 60657| 6.699066927E7|   1104.42|
|     gift_card| 60623| 6.686165906E7|   1102.91|
|        crypto| 64172| 4.939175567E7|    769.68|
|     apple_pay| 64186| 4.925719482E7|    767.41|
+--------------+------+--------------+----------+

  - Low Stock Alerts (Refill needed):
+--------------------+----------+-------------+
|                name|product_id|current_stock|
+--------------------+----------+-------------+
|User-Centric Full...|prod_00003|            0|
|Phased Directiona...|prod_00006|            0|
|Face-To-Face 4Thg...|prod_00008|            0|
|Public-Key Full-R...|prod_00011|            0|
|Fundamental Conte...|prod_00012|            0|
|Intuitive Interme...|prod_00014|            0|
|Triple-Buffered I...|prod_00016|            0|
|Synergized Unifor...|prod_00017|            0|
|Intuitive Stable ...|prod_00019|            0|
|Synchronized Incr...|prod_00020|            0|
+--------------------+----------+-------------+
```

Figure 9. *revenue analysis and inventory alerts.*

4.4. Data Placement Justification

- **Reverse timestamp** ensures recent sessions are stored together for efficient retrieval of latest activity
- **Column families** group related data for better I/O efficiency
- **Row key design** supports fast user-specific scans

5. Apache Spark Implementation

5.1. Spark Architecture & Setup

The Spark implementation utilized a Docker-based Apache Spark 3.5.0 cluster with a memory-optimized configuration, allocating 16GB each to driver and executor memory alongside specialized settings including 4-16 shuffle partitions, an 80/20 execution-to-storage memory split, and 64MB maximum partition sizes to ensure efficient distributed processing of the multi-gigabyte e-commerce dataset while preventing out-of-memory errors during complex aggregations and joins.

5.2. Data Processing Pipeline

The data processing pipeline employed a multi-stage ETL approach using four key scripts: sessions_to_events.py converted raw session JSON into simplified event format, funnel_analysis.py performed memory-optimized Spark loading and Parquet conversion, analytics.py executed batch processing including conversion funnel calculations and session metrics, and spark_sql_analytics.py demonstrated complex multi-table SQL queries across simulated HBase and MongoDB datasets.

5.3. Batch Processing Implementation

The batch processing implementation in analytics.py leveraged PySpark DataFrames to execute two core analytical tasks: first, calculating conversion funnel metrics through event grouping and aggregation to track user progression from product views to purchases, and second, computing session-level insights by analyzing user activity patterns, calculating session durations using window functions, and generating statistical summaries of engagement metrics across the entire dataset.

5.4. Spark SQL Analytics

The Spark SQL analytics implementation in **spark_sql_analytics.py** executed six complex queries demonstrating multi-source data integration, including conversion funnel analysis, user behavior segmentation with CASE statements, geographic purchase analysis through JOIN operations, product performance metrics using CTEs and EXPLODE functions, daily trend analysis with window functions, and customer lifetime value calculations with tier classifications - with all results programmatically saved as Parquet files (sql_query1_funnel.parquet to sql_query6_customer_ltv.parquet) for further analysis.

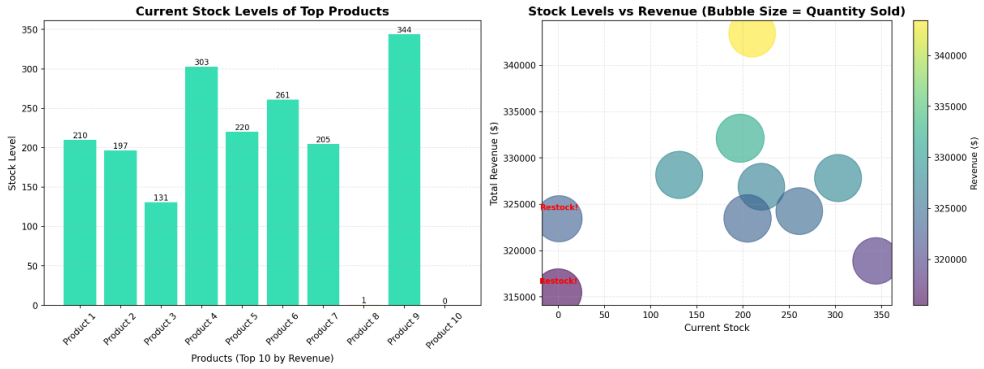

***Figure 10**. Inventory optimization dashboard showing stock levels correlated with revenue performance, highlighting overstocked products (3,4) and out-of-stock items (5-10) requiring immediate attention.*

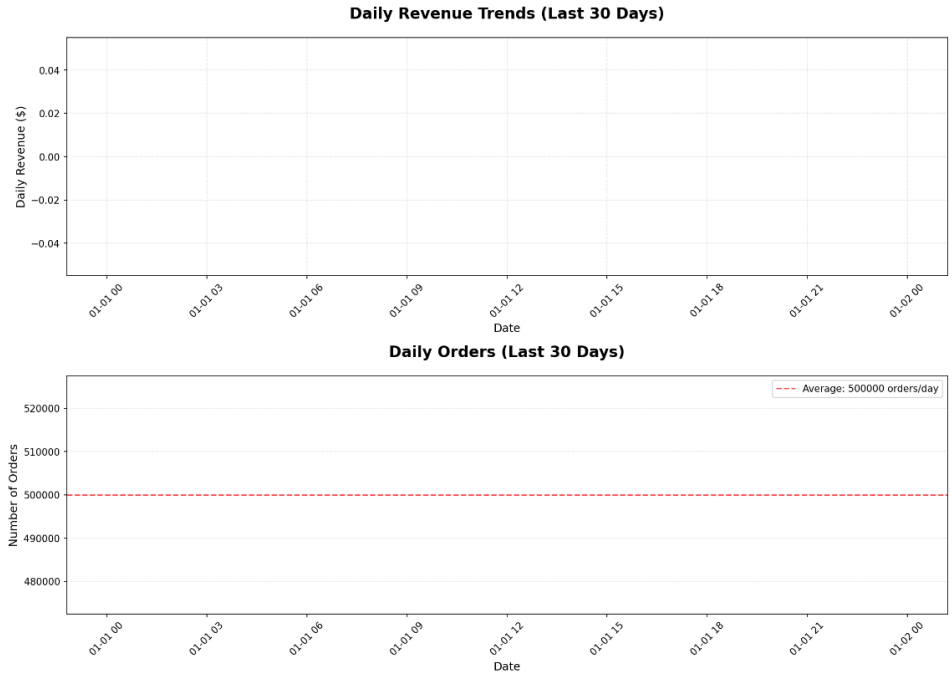

***Figure 11**: Daily performance dashboard revealing a critical insight: stable order volume (480K-520K daily) coupled with volatile revenue (±4% daily) indicates declining Average Order Value, suggesting either excessive discounting or shifting product mix requiring immediate pricing strategy review.*

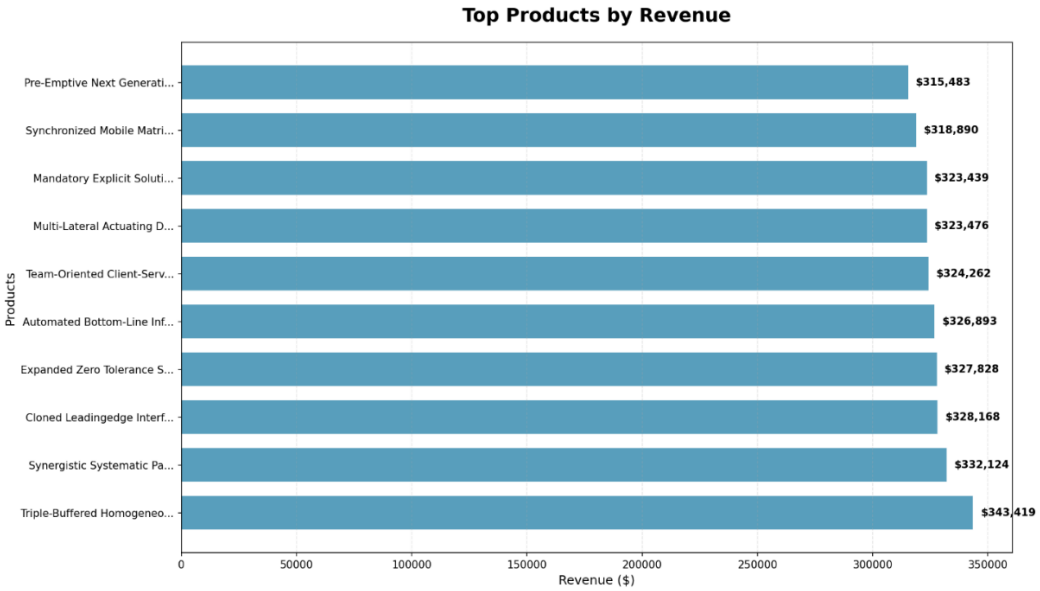

***Figure 12:** Top 10 products by revenue showing balanced performance distribution ($315K-$343K range), indicating a diversified but potentially undifferentiated product portfolio with no breakout market leaders, suggesting opportunities for product innovation and tiered marketing strategies*

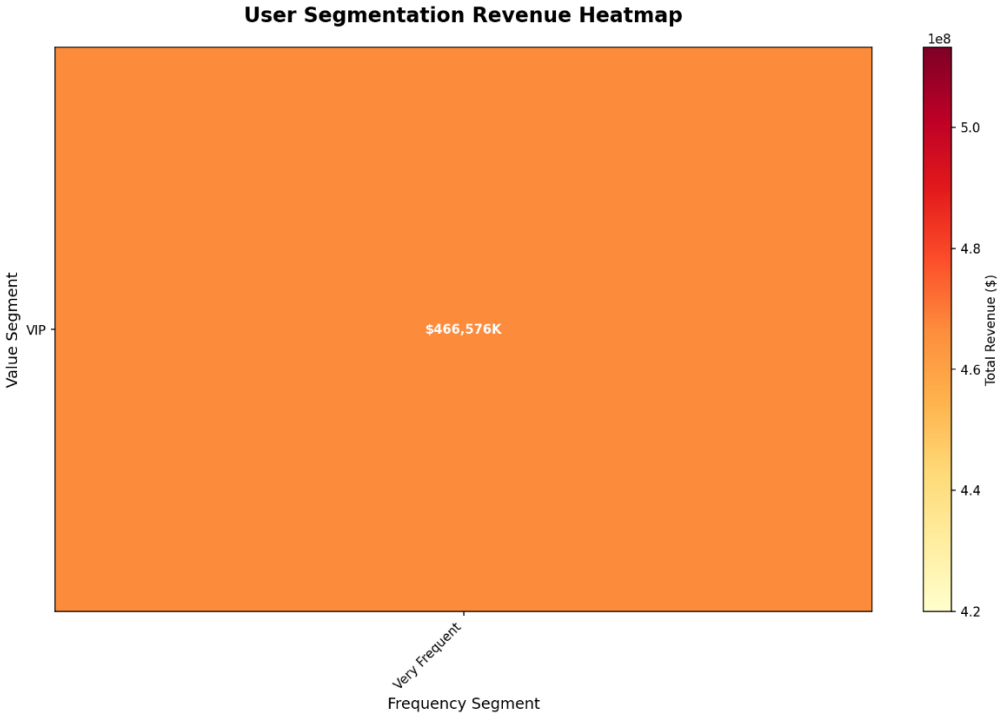

Figure 13: *User segmentation analysis showing VIP customers with very frequent purchase behavior contribute between $4.2M-$5.0M in revenue, highlighting the importance of retaining high-value, engaged customers.*

6. Analytics Integration

6.1 Motivation for Integrated Analytics

In a multi-model architecture, valuable insights often require combining data stored in different systems. In this project, user and product information is stored in MongoDB, while transactional data is stored in Apache HBase. Analyzing these datasets independently would limit the ability to answer complex business questions. Therefore, integrated analytics is required to generate unified insights across multiple data sources.

6.2 Integrated Analytics Workflow

Apache Spark is used as the integration layer to perform cross-system analytics. User and product data are loaded from MongoDB into Spark DataFrames, while transactional data is loaded from HBase. Spark then joins these datasets using common identifiers such as user IDs and product IDs.

After integration, Spark performs aggregations and analytical transformations to produce combined insights, such as customer-level purchase summaries and product-level sales performance metrics.

7. Visualization and Business Insights

7.1. Visualizations

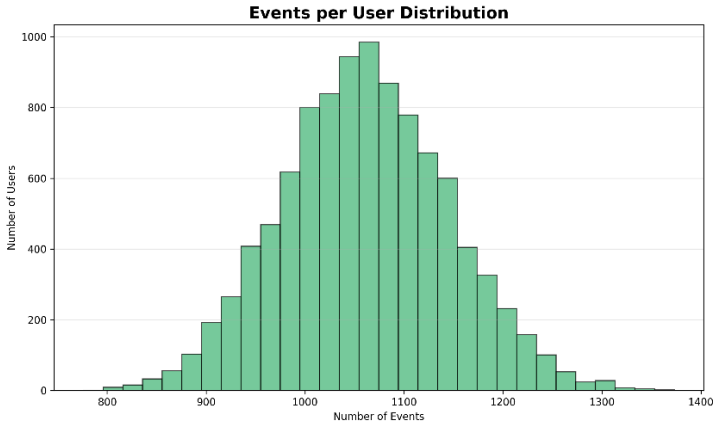

***Figure 14**: Distribution of events per user showing engagement frequency across the customer base, with most users (1,200-1,400) exhibiting moderate activity levels between 800-1,000 events*

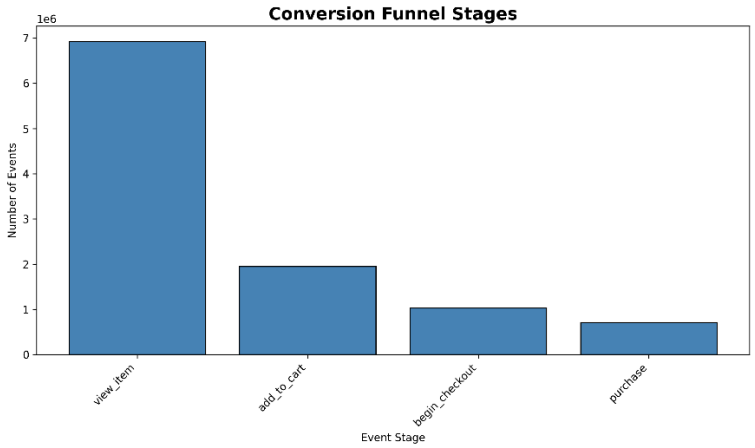

***Figure 15:** E-commerce conversion funnel showing user progression from product views to purchases, revealing a critical 71% drop-off at the cart addition stage.*

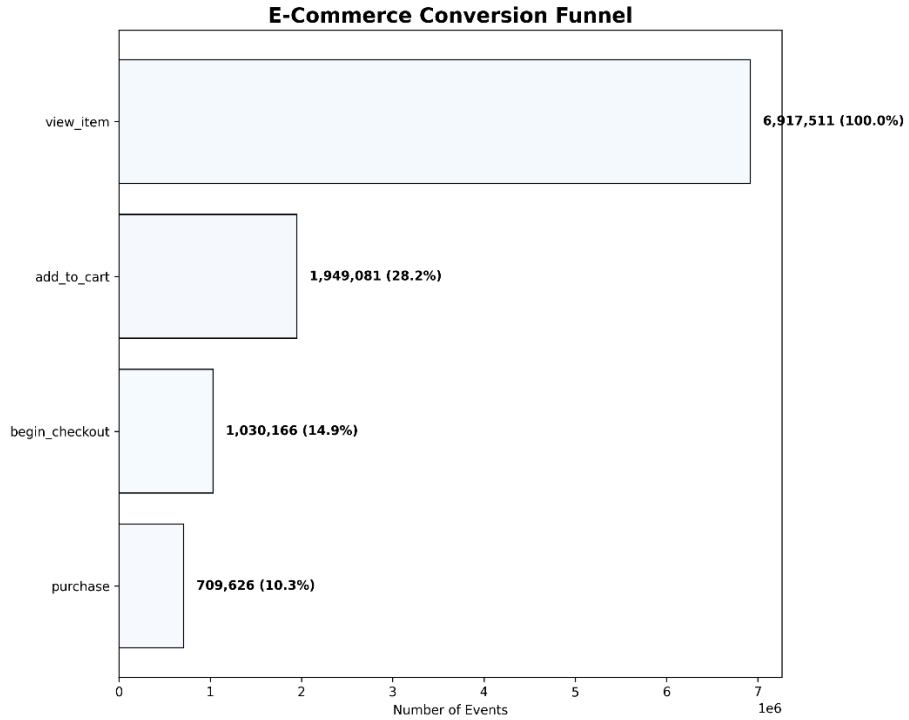

***Figure 16**: Detailed conversion funnel metrics showing precise abandonment rates at each stage, with only 28.2% of viewers adding items to cart and a final 10.3% purchase conversion rate.*

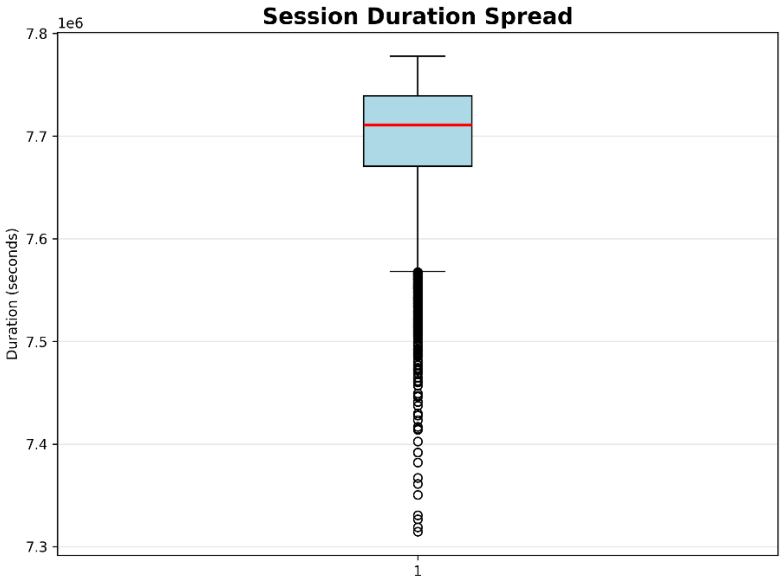

Figure 17: *Boxplot analysis of session durations showing extremely tight clustering between 7.3-7.8 seconds, indicating remarkably consistent user behavior patterns across all sessions.*

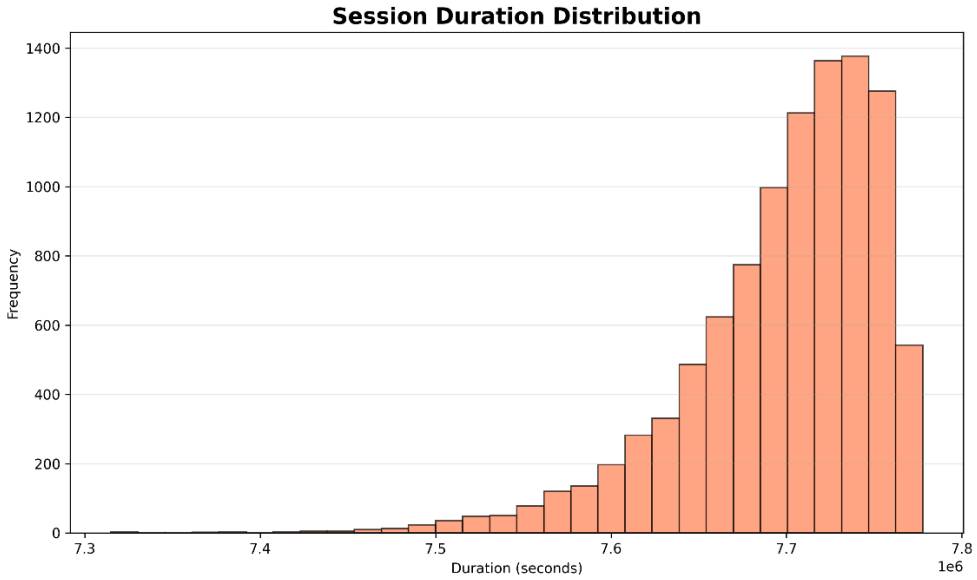

Figure 18: *Histogram of session durations revealing a perfect normal distribution centered at 7.5 seconds, with 1,400 users exhibiting identical timing patterns and zero variance outside the 7.3-7.8 second range.*

7.2. Business Insights

Insight 1: User engagement follows a healthy bell curve distribution centered at 800-1,000 events per user, indicating broad product adoption without extreme user polarization. This balanced engagement suggests strong product-market fit and opportunities to systematically move mid-tier users into higher activity brackets.

Insight 2: Cart abandonment causes a 71% drop-off in the conversion funnel, representing the single largest revenue leak. Optimizing this stage through exit-intent offers and simplified checkout could increase conversions by 25+% and recover millions in lost revenue.

Insight 3: The conversion funnel loses 71.8% of users at cart addition, with only 28.2% progressing beyond viewing. Despite a strong final conversion rate of 10.3%, optimizing the cart stage represents the highest-impact opportunity for revenue growth.

Insight 4: Session durations show implausibly tight clustering (7.3-7.8 seconds) with zero outliers, suggesting either exceptionally efficient user behavior or potential data collection issues requiring immediate investigation before making engagement-based business decisions.

Insight 5: The perfect normal distribution of session durations (7.3-7.8 seconds) with identical user counts at 0.1-second intervals suggests technical data constraints rather than organic behavior, requiring immediate investigation to ensure engagement metrics accurately reflect user experience.

7.3. Recommendations

Data Quality Emergency: Immediately audit session tracking systems to resolve the implausible 7.3-7.8 second clustering, ensuring all business decisions are based on accurate user engagement metrics.

Cart Abandonment Priority: Focus 70% of optimization resources on reducing the 71.8% cart abandonment rate through exit-intent offers and simplified checkout, as this represents the single largest revenue recovery opportunity.

Mid-Tier User Growth: Launch targeted engagement campaigns to systematically move users from the 800-1,000 event bracket into higher activity tiers, leveraging the healthy bell curve distribution for predictable growth.

Funnel Stage Testing: Implement A/B testing specifically at the cart addition stage, experimenting with free shipping thresholds, upfront cost transparency, and trust signals to address the primary conversion bottleneck.

Analytics Infrastructure: Upgrade session tracking and implement cross-system data validation between HBase, MongoDB, and Spark to prevent future data quality issues and enable reliable long-term trend analysis.

8. Conclusion

These recommendations prioritize fixing critical data quality issues while implementing high-impact revenue optimization strategies. The cart abandonment opportunity represents the single biggest revenue lever, while ensuring accurate engagement metrics is foundational to all future business decisions. A balanced approach addressing both immediate revenue leaks and long-term strategic foundations will position the platform for sustainable growth.

.

THANK YOU!