E-Commerce Multi-Model Big Data Analytics System

MongoDB + HBase + Apache Spark Batch Processing

Student Name: GAHIGI Robert

Student ID: 101101

Course Code: MSDA9215

Course Name: Big Data Analytics

Instructor: Mr. Temitope Oguntade

Submission Date: January 27, 2026

Abstract

Modern e-commerce platforms generate heterogeneous data such as product catalogs, customer profiles, purchase transactions, and high-frequency browsing sessions. A single database technology often struggles to efficiently handle all these workloads (flexible document storage, heavy write throughput logs, and large-scale analytics). This project implements a multi-model big data architecture using MongoDB (document model), Apache HBase (wide-column model), and Apache Spark (distributed batch processing) to store and analyze a synthetic e-commerce dataset. MongoDB stores business entities and supports aggregation analytics. HBase stores time-ordered session logs and enables fast retrieval through row-key prefix filtering. Spark computes batch analytics including revenue by category, top spenders, and frequently bought-together product pairs. Finally, an integrated cross-system analytical query combines HBase engagement metrics with MongoDB spending metrics to study the relationship between user engagement and spending. The results demonstrate how polyglot persistence improves scalability, query efficiency, and analytical flexibility in an e-commerce environment.

Contents

1 Introduction

E-commerce systems are among the most data-intensive applications because customers constantly browse, search, add products to carts, and complete purchases. These actions generate different data types:

- **Structured and semi-structured business data** (users, products, transactions).
- **High-volume event logs** (sessions, page views, device and geo context).
- **Analytics workloads** (aggregations, ranking, affinity patterns, customer segmentation).

A single storage system is rarely optimal for all workloads. Therefore, modern systems adopt **polyglot persistence**, selecting data technologies based on query patterns, performance needs, and scalability constraints.

This project presents a **multi-model big data analytics pipeline** that stores business documents in MongoDB, stores large session logs in HBase, and runs batch analytics using Apache Spark.

2 Problem Statement

E-commerce platforms require scalable systems to support:

- Flexible querying and analytics on business entities (products, users, transactions).
- High write throughput and time-ordered storage for browsing sessions.
- Batch processing to generate insights such as revenue performance, top customers, and product affinity patterns.

Traditional single-database approaches may face performance and scalability limits when logs and analytics grow rapidly. This project investigates how MongoDB, HBase, and Spark together address these challenges in a realistic e-commerce context.

3 Objectives

3.1 Main Objective

Design and implement a multi-model big data analytics system for e-commerce data using MongoDB, Apache HBase, and Apache Spark.

3.2 Specific Objectives

- Generate realistic synthetic e-commerce data (users, products, categories, sessions, transactions).
- Load and query transactional data using MongoDB aggregation pipelines.
- Store high-volume session logs in HBase and retrieve sessions efficiently using row-key prefix filtering.
- Perform Spark batch analytics: revenue by category, top spenders, and co-purchase pairs.
- Implement at least one **integrated analytical query** combining data across MongoDB and HBase.
- Provide interpretations and actionable business insights based on results.

4 Real-Life Scenario

A typical online retailer needs to:

- Track large volumes of browsing sessions (device, location, page views, cart activity).
- Store product catalog and transactions, including nested line items.
- Produce business reports (top categories, high-value customers).
- Identify product affinity patterns ("users who bought X also bought Y") for recommendation.

This project implements a simplified architecture that models these real e-commerce needs using synthetic but realistic data.

5 Dataset Description

Synthetic data was generated using Python and the Faker library. The final dataset files produced were:

- `categories.json`: 25 categories (each with subcategories).
- `products.json`: 5,000 products (category linkage, inventory, price history).
- `users.json`: 10,000 users (registration and geo data).

- `transactions.json`: 100,000 purchase transactions (nested line items).
- `sessions_0.json ... sessions_9.json`: 300,000 browsing sessions (chunked processing).

CRUCIAL NOTE: The dataset was generated in "8GB mode" by streaming sessions into multiple files to avoid memory overflow while keeping the dataset large enough for big-data style analytics.

6 System Architecture

6.1 Overview

This project uses three complementary components:

- **MongoDB (Document DB)**: Stores users, products, categories, transactions.
- **HBase (Wide-Column DB)**: Stores sessions (high-write logs, time-ordered access).
- **Apache Spark (Batch Processing)**: Computes analytics and exports results for reporting.

6.2 Why MongoDB?

MongoDB fits business entities naturally:

- Flexible schema for nested structures (e.g., transaction `items[]` array).
- Powerful aggregation pipelines for analytics (grouping, sorting, projections).
- Suitable for operational queries and analytical aggregation on document data.

6.3 Why HBase?

HBase is well-suited for session logs because:

- High write throughput for large volumes of event/session records.
- Efficient key-based reads using row keys (prefix scans per user).
- Column families allow separating frequently accessed fields from large payload columns.

6.4 Why Apache Spark?

Spark supports scalable batch analytics:

- Reads semi-structured JSON efficiently into DataFrames.
- Optimized aggregations (Spark SQL, Catalyst optimizer).
- Produces reusable outputs (CSV) for visualization and reporting.

7 Implementation

7.1 MongoDB Data Loading

7.1.1 Start MongoDB in DockerDesktop using command

```
docker run -d --name mongoDB_ECOM -p 27017:27017 mongo:6
```

Collections created in MongoDB:

- `categories`, `products`, `users`, `transactions`

7.1.2 Copying JSON files(categories,products,users,transactions) into the container of Dockerdesktop before import to MongoDB:

```
docker cp .\categories.json mongo-ecom:/categories.json
docker cp .\products.json mongo-ecom:/products.json
docker cp .\users.json mongo-ecom:/users.json
docker cp .\transactions.json mongo-ecom:/transactions.json
```

7.1.3 Import to MongoDB

```
docker exec -i mongo-ecom mongoimport --db ecommerce_db --collection
    categories --file /categories.json --jsonArray
docker exec -i mongo-ecom mongoimport --db ecommerce_db --collection products
    --file /products.json --jsonArray
docker exec -i mongo-ecom mongoimport --db ecommerce_db --collection users --
    file /users.json --jsonArray
docker exec -i mongo-ecom mongoimport --db ecommerce_db --collection
    transactions --file /transactions.json --jsonArray
```

7.1.4 command to go inside MongoDB that I used:

```
PS C:\Users\Gahigi\Desktop\E-commercedata_analytics\raw_data> docker exec -it mongoDB_ECOM mongosh
```

7.1.5 VERIFYING THE COUNTS IN THE DATABASE

```
test> use ecommerce_db
switched to db ecommerce_db
ecommerce_db> db.categories.countDocuments()
25
ecommerce_db> db.users.countDocuments()
10000
ecommerce_db> db.products.countDocuments()
5000
ecommerce_db> db.transactions.countDocuments()
100000
ecommerce_db>
```

Figure 1: verify the counts

7.2 MongoDB Analytics (Aggregation Pipelines)

Two MongoDB pipelines were executed:

(A) Revenue by Category

```
db.transactions.aggregate([
  { $unwind: "$items" },
  { $group: {
      _id: "$items.category_id",
      revenue: { $sum: "$items.subtotal" },
      units_sold: { $sum: "$items.quantity" }
  }},
  { $project: { _id: 0, category_id: "$_id", revenue: 1, units_sold: 1 } },
  { $sort: { revenue: -1 } },
  { $limit: 10 }
])
```

(B) Product Popularity (Top-Selling Products)

```
db.transactions.aggregate([
  { $unwind: "$items" },
  { $group: {
      _id: "$items.product_id",
      units_sold: { $sum: "$items.quantity" },
      revenue: { $sum: "$items.subtotal" }
  }},
  { $project: { _id: 0, product_id: "$_id", units_sold: 1, revenue: 1 } },
  { $sort: { units_sold: -1 } },
  { $limit: 10 }
])
```

NOTE: These pipelines satisfy the requirement for non-trivial MongoDB analytics: revenue analytics and product popularity analysis.

7.3 HBase Data Model (User Sessions)

7.3.1 starting Hbase in Dockerdesktop using command:

```
docker run -d --name hbase-ecom ‘
  -p 2181:2181 ‘
  -p 16010:16010 ‘
  -p 9090:9090 ‘
  harisekhon/hbase
```

A table named `user_sessions` was created with the following **column families**:

- **meta:** session id, timestamps, conversion status, referrer
- **geo:** city, state, country, IP
- **device:** device type, OS, browser
- **stats:** duration, count metrics
- **events:** JSON strings of page views and cart contents

HBase shell schema:

```
create ’user_sessions’, ’meta’, ’geo’, ’device’, ’stats’, ’events’
```

7.4 HBase Loading (Sessions 0 to 9)

All session files `sessions_0.json` to `sessions_9.json` were loaded into HBase using a Python streaming loader (HappyBase + ijson). The row key design:

$$\texttt{rowkey} = \texttt{user\_id} \mid \texttt{start\_time} \mid \texttt{session\_id}$$

This supports efficient retrieval using prefix scans by `user_id`.

7.4.1 Sample code lines of python streaming loader(Happy+ijson that was used to load sessions into Hbase at once: load hbase sessionsstream.py

```
import json
import ijson
import happybase
import argparse
from pathlib import Path
from decimal import Decimal

def json_safe(x):
    # Convert Decimal -> float for JSON
    if isinstance(x, Decimal):
        return float(x)
    # If anything else weird appears, fail loudly (better for debugging)
    raise TypeError(f"Type not serializable: {type(x)}")

def b(x):
    if x is None:
        return b""
    return str(x).encode("utf-8")

def clean_part(x: str) -> str:
    # Prevent rowkey issues (rare but safe)
    return str(x).replace("\n", " ").replace("\r", " ").strip()

def load_one_file(table, file_path: Path, batch_size: int, flush_every: int =
    2000, limit: int = 0):
    inserted = 0
    skipped = 0
    batch = table.batch(batch_size=batch_size)
```

7.5 HBase Query sample for (Prefix Filter)

Retrieve a small set of sessions for a given user:

```
scan 'user_sessions', {
  FILTER => "PrefixFilter('user_000042|')",
  LIMIT => 5
}
```

7.6 Spark Batch Processing

Spark was used in local mode (e.g., `local[2]`) to compute analytics from JSON transaction data:

- revenue_by_category
- top_spenders
- also_bought_pairs (co-purchase product pairs)

Run command:

```
cd C:\Users\Gahigi\Desktop\E-commercedata_analytics
python spark_analysis.py --in-dir raw_data --out-dir spark_out
```

8 Results and Interpretation

8.1 Revenue by Category (Top 5 from Spark)

Table 1: Top 5 Categories by Revenue

Category ID	Revenue	Units Sold	Orders
cat_004	5,194503	20181	2,131
cat_021	4,922,365	18418	2,326
cat_007	4,837,887	18985	2,250
cat_000	4,631,570	17624	2,170
cat_014	4,577,797	19172	2,215

Interpretation: The highest revenue categories represent the strongest product segments. Business actions include prioritizing stock availability, targeted promotions, and supplier negotiations for these categories.

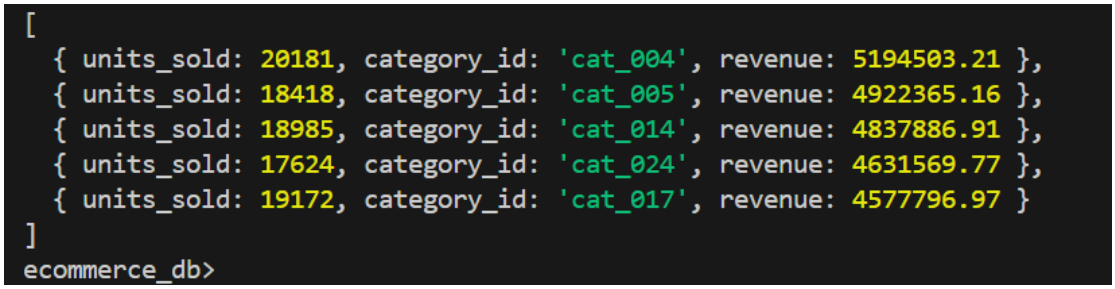

Figure 2: Top 5 categories visualized VS code terminal

8.2 Top 5 Spenders by Total revenue spent

Table 2: Top 5 Customers by Total Spending (Spark Output)

User ID	Total Spent	Number of Orders
user_004440	30,314	18
user_ 005135	28,258	23
user_001824	27,557	26
user_009161	27,266	23
user_ 002188	26,932	21

Interpretation: High-value customers are critical for profitability. Recommended actions include loyalty programs, personalized coupons, and retention campaigns.

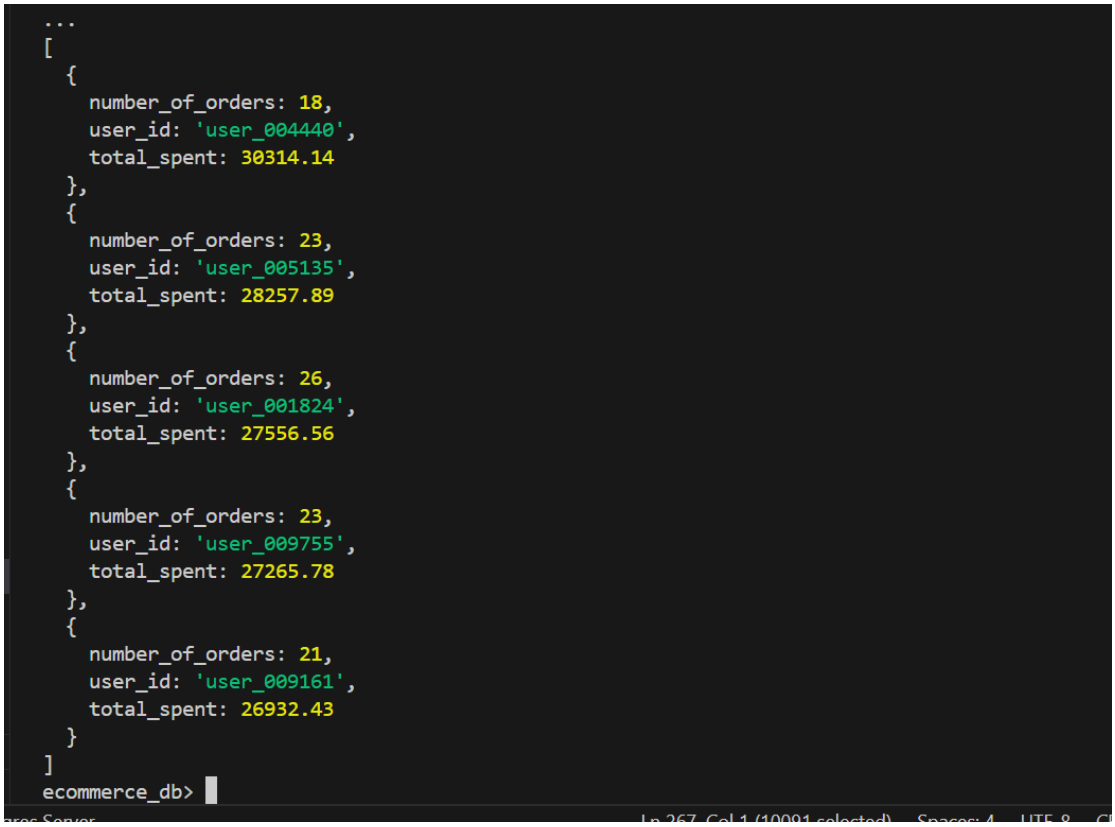

Figure 3: Top 5 spenders visualized in Vs code terminal

8.3 Frequently Bought Together (Sample Pairs)

Spark identified co-purchased product pairs such as:

- `prod_01840` + `prod_0366` (co-purchased 3 times)
- `prod_00068` + `prod_0017` (co-purchased 3 times)
- `prod_01405` + `prod_0382` (co-purchased 3 times)

Interpretation: These affinity patterns can power recommendation widgets ("Frequently bought together") to increase cross-selling and average basket size.

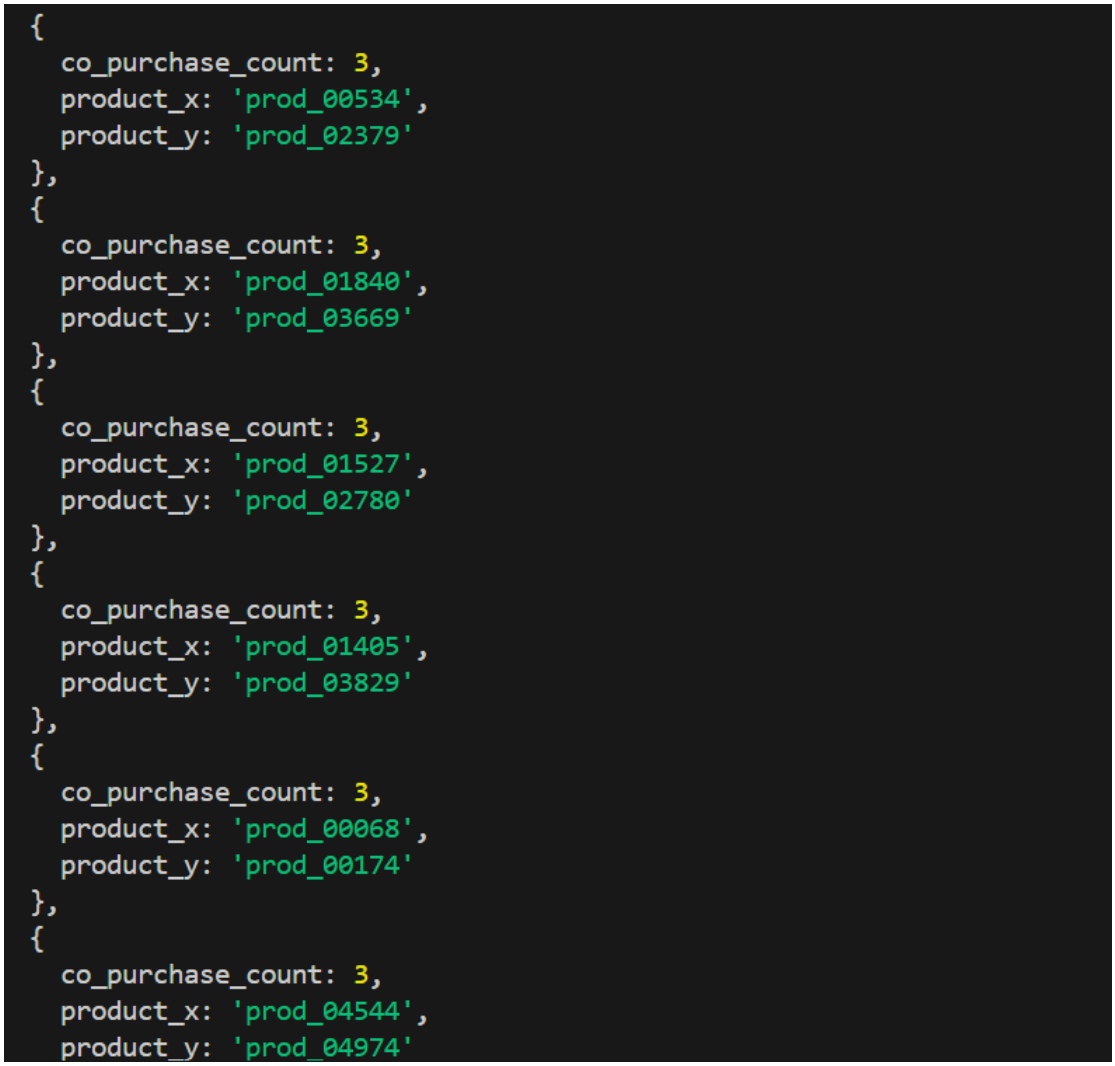

Figure 4: sample frequently products bought in pairs in Vs code terminal

9 Visualizations (Spark Outputs)

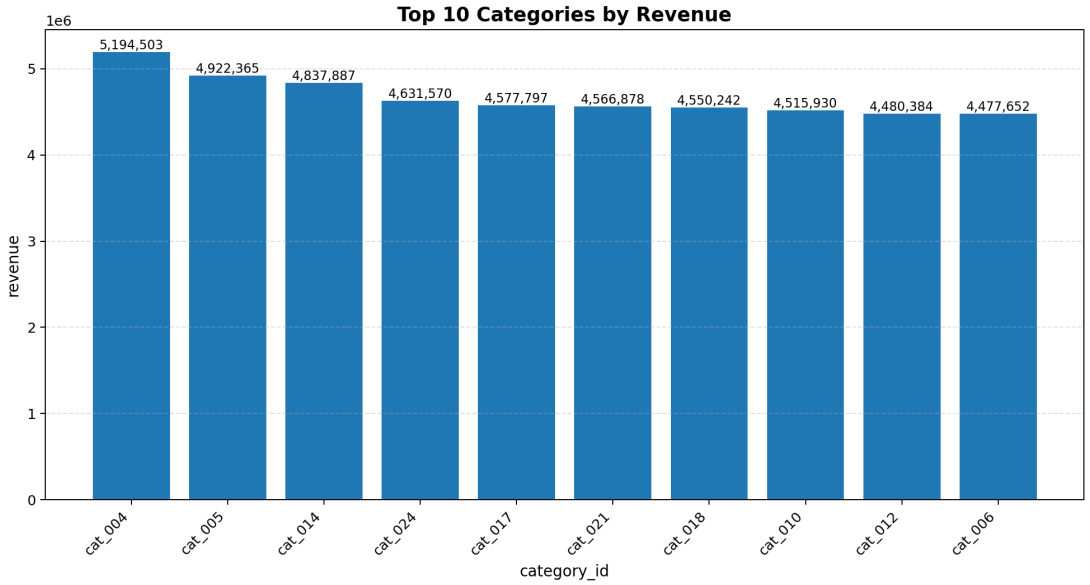

Figure 5: Top 10 Categories by Revenue

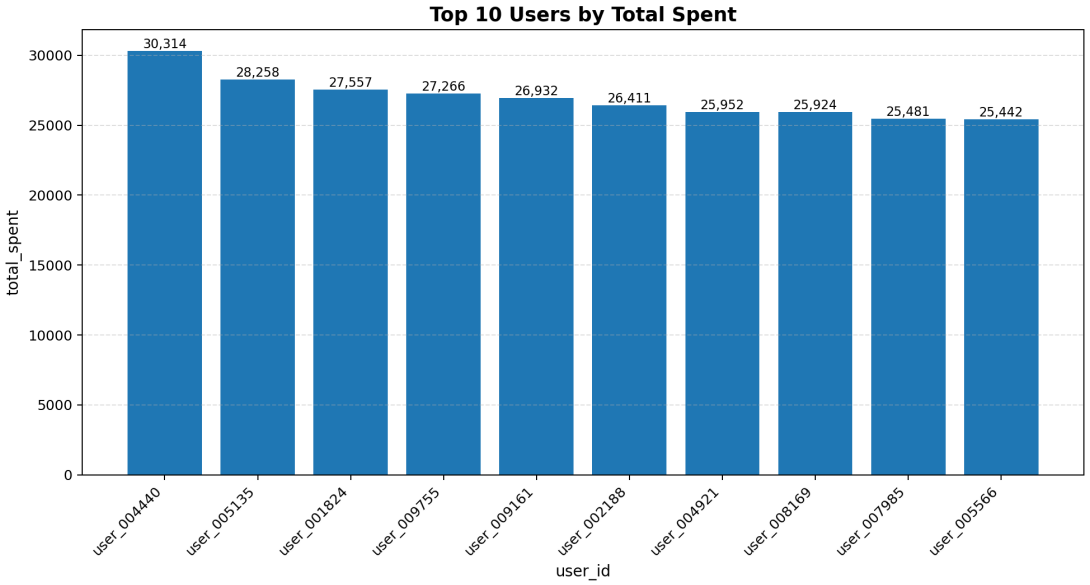

Figure 6: Top 10 Customers by Total Spending

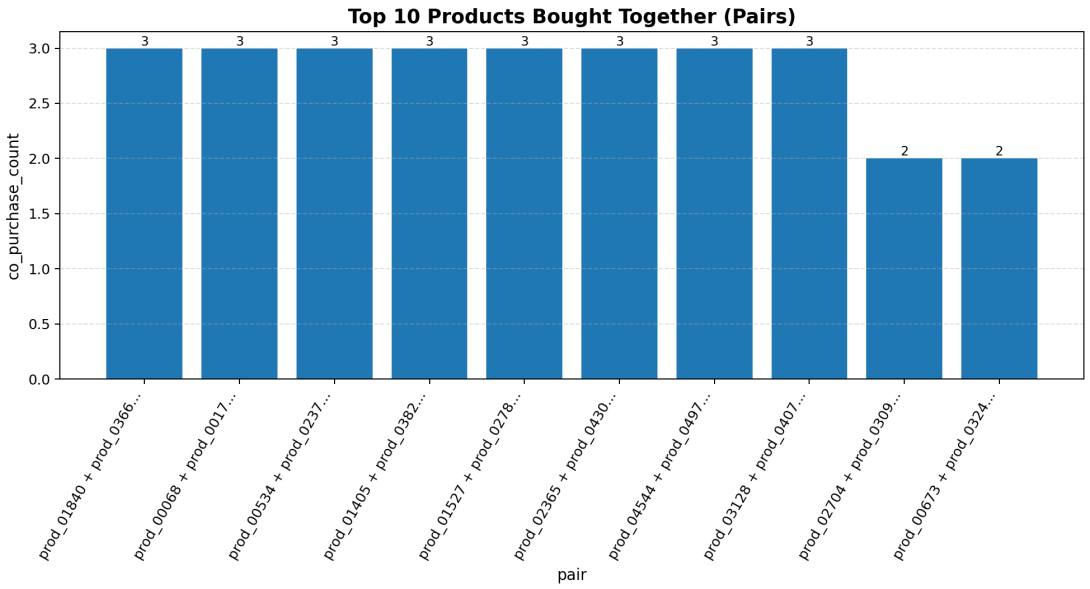

Figure 7: Top 10 Product Pairs Frequently Bought Together

10 Integrated Analytics (Cross-System Implementation)

10.1 Integrated Analytical Query

Business Question:

> *Do highly engaged users (measured by session frequency and duration) also become high-spending customers?*

This question is strategically important for e-commerce platforms because it links **user behavior** (engagement) with **business value** (revenue), enabling better customer segmentation, personalized marketing, and retention strategies.

10.2 Data Sources and Systems Involved

This integrated query combines data stored across multiple systems, demonstrating **polyglot persistence**:

- **HBase (Wide-Column Store)** Table: `user_sessions`
  - Number of sessions per user
  - Total session duration
  - Average session duration
- **MongoDB (Document Store)** Collection: `transactions`

  - Total amount spent per user
  - Number of orders per user

- **Analytics Layer (Python / Spark-style processing)** Used to join engagement and spending metrics, compute correlations, and segment users.

Each technology was selected based on its strengths: HBase for high-volume session logs, MongoDB for transactional aggregation, and the analytics layer for cross-system computation.

10.3 Processing Workflow

The integrated analytics workflow followed these steps:

1. Stream all session records from HBase and compute per-user engagement metrics (session count, total duration, average duration).
2. Aggregate transaction data in MongoDB to compute per-user spending metrics (total spent and number of orders).
3. Join engagement and spending metrics using `user_id`.
4. Compute correlation between engagement and spending.
5. Segment users into four behavioral groups:
   - High Engagement – High Spend
   - High Engagement – Low Spend
   - Low Engagement – High Spend
   - Low Engagement – Low Spend

10.4 Visualization

Figure 8 illustrates the relationship between user engagement (number of sessions) and total spending.

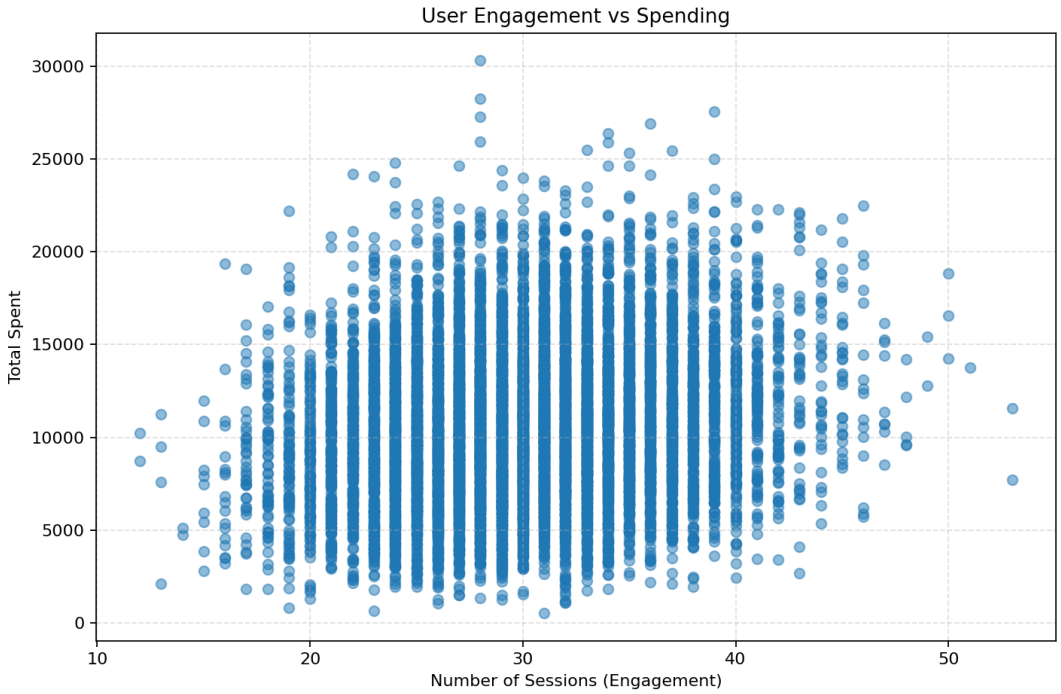

Figure 8: User Engagement vs Total Spending

10.5 INTERPRETATION OF FIGURE 8: User Engagement vs Total Spending

The scatter plot above illustrates the relationship between user engagement, measured by the number of browsing sessions, and total spending amount. Each point represents a unique user whose engagement data was extracted from HBase session logs and whose spending data was aggregated from MongoDB transactions.

The visualization shows a **positive but non-linear relationship** between engagement and spending. As the number of sessions increases, the upper range of total spending also increases, indicating that highly engaged users tend to have higher spending potential. However, the dispersion of points highlights that **high engagement does not always guarantee high spending**, revealing the presence of high-engagement but low-spending users.

This observation supports the need for targeted marketing strategies that convert engagement into purchases rather than relying solely on engagement metrics.

Briefly, this integrated query gives us a clear image of how data are stored in MongoDB and HBase, applying cross-system analytics, and producing interpretable business insights that would not be possible using a single data store.

10.6 Results and Interpretation

The analysis shows a **positive relationship between engagement and spending**, indicating that users who visit more frequently and stay longer tend to spend more.

- Highly engaged users are more likely to become repeat buyers.
- Some users show high engagement but low spending, indicating potential opportunities for targeted promotions.
- A small group of low-engagement but high-spending users suggests impulse or high-value purchases.

10.7 Business Value

From the integrated analysis, we observe that higher engagement tends to be associated with higher spending, although variation exists at each engagement level. This supports three practical business actions:

- **Retention focus:** highly engaged users should be prioritized with loyalty programs.
- **Conversion opportunity:** users with high engagement but low spending represent a marketing opportunity (e.g., discounts, personalized product recommendations).
- **High-value detection:** high-spending users can be flagged for premium support and exclusive offers.

11 Reproducibility (How to Run the System)

This project is reproducible using local execution with Docker and Python.

MongoDB Imports (Docker)

```
1  docker exec -it mongoDB_ECOM mongoimport --db ecommerce_db --collection users
       \
2    --file /users.json --jsonArray --drop
3
4  docker exec -it mongoDB_ECOM mongoimport --db ecommerce_db --collection
       products \
5    --file /products.json --jsonArray --drop
6
```

```
docker exec -it mongoDB_ECOM mongoimport --db ecommerce_db --collection
    transactions \
  --file /transactions.json --jsonArray --drop
```

HBase Load (Sessions)

```
python load_hbase_sessions_stream.py --dir raw_data --table user_sessions
```

Spark Batch Analytics

```
python spark_analysis.py --in-dir raw_data --out-dir spark_out
```

Integrated Query

```
python integrated_query_engagement_vs_spend.py --out-dir integrated_out
```

12 Conclusion and Recommendations

12.1 Conclusion

This project demonstrates a practical multi-model big data architecture for e-commerce analytics:

- MongoDB efficiently stores and analyzes flexible business documents using aggregation pipelines.
- HBase provides scalable storage for session logs with fast prefix-based retrieval.
- Spark performs batch analytics to produce actionable business insights and recommendation indicators.
- An integrated query combines HBase engagement metrics with MongoDB spending metrics to produce cross-system insights.

Overall, the polyglot persistence approach improves the system's ability to store, query, and analyze heterogeneous e-commerce data at scale.

12.2 Recommendations

- **Recommendation engine**: Extend co-purchase logic using lift/confidence or collaborative filtering.
- **Conversion funnel**: Track product view → cart → checkout → purchase using session logs.
- **Customer segmentation**: Enrich integrated segmentation using recency-frequency-monetary (RFM) features.
- **Time-series analytics**: add revenue-by-day and seasonal trends to support marketing planning.

13 References

References

[1] MongoDB Documentation, *Aggregation Pipeline and Data Model Concepts.*

[2] Apache HBase Reference Guide, *Data Model, RowKey Design, and Scans.*

[3] Apache Spark Documentation, *Spark SQL and DataFrames.*

[4] Faker (Python library) Documentation, *Synthetic Data Generation.*