

ADVENTIST UNIVERSITY OF CENTRAL AFRICA (AUCA)

Faculty of Information Technology

BIG DATA ANALYTICS FINAL EXAM REPORT

Project Title:

Integrated Big Data Analytics Using MongoDB, HBase, and Apache Spark

Student Name: *INGABIRE Lydie*

Registration Number: *101034*

Program: *Big Data Analytics*

Course: *Big Data Analytics*

Instructor: *[Instructor's Name]*

Academic Year: *2025–2026*

Submission Date: *01 February 2026*

Distributed Multi-Model Analytics for E-Commerce Data Using MongoDB, HBase, and Apache Spark

1. Introduction

This project presents the design and implementation of a distributed multi-model analytics system for large-scale e-commerce data. The objective is to demonstrate the strategic use of MongoDB, HBase, and Apache Spark based on data characteristics, access patterns, and analytical requirements. All datasets used in the project were synthetically generated using a Python script (dataset_generator.py) to simulate realistic e-commerce activity, including users, products, transactions, and browsing sessions. MongoDB was used to manage structured transactional data, HBase to store time-series user activity and product metrics, and Apache Spark as the central processing engine for batch and integrated analytics, enabling the derivation of meaningful business insights.

2. System Architecture Overview

The system architecture follows a layered design. MongoDB is used for document-oriented transactional and catalog data, HBase is used for high-volume time-series session data, and Apache Spark acts as the central processing and integration engine.

3. Data Modeling and Storage

This section describes data modeling decisions for MongoDB and HBase, including schema design, justification, and implementation.

3.1 MongoDB Implementation

MongoDB stores users, products, categories, sessions, and transactions as JSON-like documents. Nested structures such as transaction items and price history are embedded to reduce joins and optimize analytical queries.

```
PS C:\Users\princ\OneDrive\Desktop\Lydie_project> mongoimport --db e_commerce --collection categories --file .\categories.json --type json
2026-01-30T12:45:30.101+0200    connected to: mongodb://localhost/
2026-01-30T12:45:30.182+0200    25 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\princ\OneDrive\Desktop\Lydie_project> mongoimport --db e_commerce --collection products --file .\products.json --type json
2026-01-30T12:45:30.325+0200    connected to: mongodb://localhost/
2026-01-30T12:45:30.456+0200    2000 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\princ\OneDrive\Desktop\Lydie_project> mongoimport --db e_commerce --collection users --file .\users.json --type json
2026-01-30T12:45:30.569+0200    connected to: mongodb://localhost/
2026-01-30T12:45:30.736+0200    5000 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\princ\OneDrive\Desktop\Lydie_project> mongoimport --db e_commerce --collection sessions --file .\sessions.json --type json
2026-01-30T12:45:33.866+0200    connected to: mongodb://localhost/
2026-01-30T12:45:33.866+0200    [#####
.....] e_commerce_sessions 57.9MB/510MB (11.4%)
2026-01-30T12:45:36.866+0200    [#####
.....] e_commerce_sessions 113MB/510MB (22.1%)
2026-01-30T12:45:39.866+0200    [#####
.....] e_commerce_sessions 153MB/510MB (30.1%)
2026-01-30T12:45:42.866+0200    [#####
.....] e_commerce_sessions 192MB/510MB (37.6%)
2026-01-30T12:45:45.866+0200    [#####
.....] e_commerce_sessions 230MB/510MB (45.0%)
2026-01-30T12:45:48.866+0200    [#####
.....] e_commerce_sessions 267MB/510MB (52.3%)
2026-01-30T12:45:51.865+0200    [#####
.....] e_commerce_sessions 303MB/510MB (59.3%)
2026-01-30T12:45:54.866+0200    [#####
.....] e_commerce_sessions 364MB/510MB (71.3%)
2026-01-30T12:45:57.867+0200    [#####
.....] e_commerce_sessions 425MB/510MB (83.3%)
2026-01-30T12:46:00.866+0200    [#####
.....] e_commerce_sessions 485MB/510MB (95.0%)
2026-01-30T12:46:02.126+0200    [#####
.....] e_commerce_sessions 510MB/510MB (100.0%)
2026-01-30T12:46:02.126+0200    300000 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\princ\OneDrive\Desktop\Lydie_project> mongoimport --db e_commerce --collection transactions --file .\transactions.json --type json
2026-01-30T12:50:59.710+0200    connected to: mongodb://localhost/
2026-01-30T12:51:02.710+0200    [#####
.....] e_commerce_transactions 25.6MB/38.2MB (66.9%)
2026-01-30T12:51:03.760+0200    [#####
.....] e_commerce_transactions 38.2MB/38.2MB (100.0%)
2026-01-30T12:51:03.760+0200    100000 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\princ\OneDrive\Desktop\Lydie_project>
```

```

e_commerce> show collections
categories
products
transactions
users
e_commerce> db.categories.countDocuments()
25
e_commerce> db.transactions.countDocuments()
100000
e_commerce> db.products.countDocuments()
2000
e_commerce> db.users.countDocuments()
5000
e_commerce> |

```

After ingestion, the MongoDB database e-commerce was verified by listing collections and validating record counts. The database contains the core collections required for analytics—categories, products, users, and transactions. Document counts confirm successful ingestion with 25 categories, 2,000 products, 5,000 users, and 100,000 transactions, consistent with the generated dataset.

3.1.1 MongoDB Aggregation Analytics

Aggregation pipelines were implemented using PyMongo in the script mongo_aggregations.py. These include user segmentation by lifetime value, revenue analytics by category, and product popularity analysis.

Revenue by Category Aggregation Output

```

e_commerce> db.transactions.aggregate([{$match: { status: "completed" }}, {$unwind: "$items"}, {$lookup: { from: "products", localField: "items.product_id", foreignField: "product_id", as: "product" }}, {$unwind: "$product"}, {$lookup: { from: "categories", localField: "product.category_id", foreignField: "category_id", as: "category" }}, {$unwind: "$category"}, {$group: { _id: "$product.category_id", category_name: { $first: "$category.name" }, total_revenue: { $sum: "$items.subtotal" }, total_quantity: { $sum: "$items.quantity" }, transactions_count: { $addToSet: { _id: "$_id" } } }, { $addFields: { transactions_count: { $size: "$transactions_count" } } } }, { $project: { _id: 0, category_id: "$_id", category_name: 1, total_revenue: 1, total_quantity: 1, transactions_count: 1 } }, { $sort: { total_revenue: -1 } }, { $limit: 5 }]);
[
  {
    category_name: 'Fleming Ltd',
    total_revenue: 3001203.18,
    total_quantity: 10842,
    transactions_count: 5345,
    category_id: 'cat_024'
  },
  {
    category_name: 'Carter, Fuller and McClure',
    total_revenue: 2866618.81,
    total_quantity: 9945,
    transactions_count: 4802,
    category_id: 'cat_002'
  },
  {
    category_name: 'Spence PLC',
    total_revenue: 2853184.36,
    total_quantity: 10976,
    transactions_count: 5376,
    category_id: 'cat_010'
  },
  {
    category_name: 'Preston Ltd',
    total_revenue: 2679329.23,
    total_quantity: 10492,
    transactions_count: 5153,
    category_id: 'cat_021'
  },
  {
    category_name: 'Burns-Rodriguez',
    total_revenue: 2468594.41,
    total_quantity: 9875,
    transactions_count: 4811,
    category_id: 'cat_009'
  }
]
e_commerce>

```

This MongoDB aggregation **calculates revenue performance by category**: it filters to completed transactions, unwinds line items, joins items to products and then categories, then groups by category to compute **total revenue**, **total quantity sold**, and **number of transactions**, finally sorting to show the **top categories**.

User Segmentation Aggregation Output

```
[e_commerce]> db.transactions.aggregate([{"$group": {"_id": "$user_id", "orders_count": {"$sum": 1}, "lifetime_spent": {"$sum": "$total"}, "first_order": {"$min": "$timestamp"}, "last_order": {"$max": "$timestamp"}}}, {"$addFields": {"segment": {"$switch": {"branches": [{"case": {"$gte": ["$lifetime_spent", 500]}, "then": "high_value"}, {"case": {"$and": [{"$gte": ["$lifetime_spent", 200]}, {"$lt": ["$lifetime_spent", 500]}]}], "then": "medium_value"}, {"default": "low_value"}]}}, {"$group": {"_id": "$segment", "users_in_segment": {"$sum": 1}, "avg_lifetime_spent": {"$avg": "$lifetime_spent"}, "avg_orders": {"$avg": "$orders_count"}}, {"$limit: 5}}]); [ { _id: 'high_value', users_in_segment: 5000, avg_lifetime_spent: 16839.92519, avg_orders: 28 } ]
```

This MongoDB aggregation calculates **per-user lifetime metrics** (number of orders and total spending), then **segments users** into **high/medium/low value** based on spending thresholds, and finally **summarizes each segment** by counting how many users are in it and computing average spending and average orders.

Product Popularity Aggregation Output

```
[e_commerce]> db.transactions.aggregate([ /* 1) Explode line items */ {"$unwind": "$items"}, {"$group": {"_id": "$items.product_id", "total_quantity_sold": {"$sum": "$items.quantity"}, "total_revenue": {"$sum": "$items.subtotal"}, "orders_count": {"$sum: 1"}}, /* 3) Join with products to get name */ {"$lookup": {"from": "products", "localField": "_id", "foreignField": "product_id", "as": "product"}}, /* 4) Final shape */ {"$project": {"_id": 0, "product_id": "$_id", "product_name": "$product.name", "total_quantity_sold": 1, "total_revenue": 1, "orders_count": 1}}, /* 5) Sort and limit */ {"$sort": {"total_quantity_sold": -1, "total_revenue": -1}}, {"$limit: 5}}]); [ { total_quantity_sold: 270, total_revenue: 133056, orders_count: 125, product_id: 'prod_00587', product_name: 'Secured Homogeneous Website' }, { total_quantity_sold: 268, total_revenue: 33446.4, orders_count: 126, product_id: 'prod_01119', product_name: 'Multi-Lateral Reciprocal Focus Group' }, { total_quantity_sold: 262, total_revenue: 46552.15999999996, orders_count: 126, product_id: 'prod_00740', product_name: 'Centralized Intermediate Protocol' }, { total_quantity_sold: 261, total_revenue: 121485.06, orders_count: 122, product_id: 'prod_00056', product_name: 'Persistent Contextually-Based Application' }, { total_quantity_sold: 259, total_revenue: 84856.17, orders_count: 136, product_id: 'prod_01197', product_name: 'Reactive Actuating Algorithm' } ]
```

This MongoDB aggregation finds the **top-selling products** by unwinding transaction line items, grouping by product_id to calculate **total quantity sold**, **total revenue**, and **order count**, then joining with the products collection to get the product name and finally sorting and limiting to the top results.

Indexing

```
e_commerce> // TRANSACTIONS
... db.transactions.createIndex({ transaction_id: 1 }, { unique: true })
... db.transactions.createIndex({ user_id: 1 })
... db.transactions.createIndex({ user_id: 1, timestamp: 1 })
... db.transactions.createIndex({ timestamp: 1 })
... db.transactions.createIndex({ status: 1 })
... db.transactions.createIndex({ status: 1, timestamp: 1 })
... db.transactions.createIndex({ "items.product_id": 1 })
... db.transactions.createIndex({ "items.product_id": 1, timestamp: 1 })
... db.transactions.createIndex({ session_id: 1 })
session_id_1
e_commerce> db.transactions.getIndexes()
... db.products.getIndexes()
... db.users.getIndexes()
... db.categories.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { category_id: 1 }, name: 'category_id_1', unique: true }
]
```

An indexing strategy was applied in MongoDB to improve query performance and scalability. Indexes were created on key identifiers and frequently queried fields to speed up joins, user analysis, time-based queries, and revenue aggregations. These indexes were verified using MongoDB tools, ensuring efficient data retrieval while keeping write operations balanced.

3.2 HBase Implementation

HBase was used to store session-level browsing data due to its efficiency in handling time-series and sparse data. HBase was accessed through a Dockerized environment using the dajobe/hbase container, which allowed the database to run independently of the host operating system. The HBase service was started and managed using Docker commands, and access to the database was obtained by opening the HBase shell through the Docker container. The HBase shell provided an interactive interface to create tables, inspect data, and execute queries. Queries were performed using HBase shell commands to retrieve session records for specific users, demonstrating how HBase supports fast key-based access to large, sparse datasets. This setup illustrates the use of HBase as a scalable wide-column store accessed via containerized infrastructure.

Dockerized HBase Environment figure

```
C:\Windows\System32>docker start hbase
hbase

C:\Windows\System32>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
bea5172e4fb0        dajobe/hbase      "/opt/hbase-server"   7 days ago        Up 3 seconds       0.0.0.0:2181->2181/tcp, [::]:2181->2181/
tcp, 0.0.0.0:16010->16010/tcp, [::]:16010->16010/tcp   hbase

C:\Windows\System32>docker exec -it hbase hbase shell
2026-01-30 12:32:07,167 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.1.2, r1dfc418f77801fbfb59a125756891b9100cfc6d, Sun Dec 30 21:45:09 PST 2018
Took 0.0067 seconds
hbase(main):001:0>
```

3.2.1 Loading and Querying Data in Hbase

Session data was loaded into HBase using a custom Python script (sessions_to_hbase.py). Each row represents a user session, identified by a composite row key.

Describe table schema (column families)

```
hbase(main):001:0> list
TABLE
ecom:product_metrics
ecom:user_sessions
2 row(s)
Took 6.0378 seconds
=> ["ecom:product_metrics", "ecom:user_sessions"]
hbase(main):002:0> |
```



```
hbase(main):002:0> describe 'ecom:user_sessions'
Table ecom:user_sessions is ENABLED
ecom:user_sessions
COLUMN FAMILIES DESCRIPTION
{NAME => 'pv', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER'
, MIN VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false'
, IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESS
ION =
> 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
{NAME => 's', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
MIN VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false',
IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSI
ON =>
'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
2 row(s)
Took 1.0584 seconds
hbase(main):003:0> describe 'ecom:product_metrics'
Table ecom:product_metrics is ENABLED
ecom:product_metrics
COLUMN FAMILIES DESCRIPTION
{NAME => 'm', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
MIN VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false',
IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSI
ON =>
'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
1 row(s)
Took 0.0736 seconds
hbase(main):004:0> |
```

Two HBase tables were designed to support time-series analytics.

The ecom: user_sessions table stores user browsing sessions using column families **s** (session metadata) and **pv** (page view/activity data). Its row key (user_id + timestamp) allows fast retrieval of a user's session history over time.

The ecom: product_metrics table stores product performance metrics in the **m** column family, using a row key (product_id + date) to support time-based analysis of product trends.

This design fits HBase's strengths in handling large-scale time-series data and enables efficient integration with Spark for analytical queries.

Loading Sessions into HBase

```
C:\Users\princ\OneDrive\Desktop\Lydie_project>python .\sessions_to_hbase.py
Loaded 50000 sessions into HBase
```

```
C:\Users\princ\OneDrive\Desktop\Lydie_project>
C:\Users\princ\OneDrive\Desktop\Lydie_project>
```

From the full sessions' dataset, a **representative subset of 50,000 sessions** was loaded into HBase. This subset was sufficient to demonstrate HBase's ability to store and query large-scale session data while keeping local resource usage manageable during development.

HBase Query for Specific User

```
hbase(main):007:0> get 'ecom:user_sessions', 'user_000000#8231320645999'
COLUMN                                     CELL
  pv:page_count                           timestamp=1769779863343, value=15
  pv:product_detail_views                 timestamp=1769779863343, value=3
  s:browser                                timestamp=1769779863343, value=Edge
  s:city                                    timestamp=1769779863343, value=Peterberg
  s:conversion_status                      timestamp=1769779863343, value=browsed
  s:country                                 timestamp=1769779863343, value=SR
  s:device_type                            timestamp=1769779863343, value=mobile
  s:duration_seconds                      timestamp=1769779863343, value=2368
  s:end_time                               timestamp=1769779863343, value=2026-01-17T22:28:42
  s:ip_address                             timestamp=1769779863343, value=197.105.83.60
  s:os                                     timestamp=1769779863343, value=macOS
  s:referrer                               timestamp=1769779863343, value=social
  s:start_time                            timestamp=1769779863343, value=2026-01-17T21:49:14
  s:state                                  timestamp=1769779863343, value=NC
1 row(s)
Took 0.1878 seconds
hbase(main):008:0> |
```

This HBase get query fetches **one specific session row** from the ecom:user_sessions table using the **exact row key** user_000000#8231320645999. The output shows the stored session attributes (location, device, times, duration, conversion status) and engagement metrics (page count, product detail views), confirming fast key-based retrieval in HBase.

HBase Query for Specific User by scan

```
hbase(main):009:0> scan 'ecom:user_sessions', {
hbase(main):010:1*   FILTER=>"PrefixFilter('user_000000#')",
hbase(main):011:1*   COLUMNS=>['s:start_time', 's:end_time', 's:duration_seconds', 's:conversion_status', 'pv:page_count', 'pv:product_detail_views'],
hbase(main):012:1*   LIMIT=>2
hbase(main):013:1> }
ROW                                         COLUMN+CELL
  user_000000#8231320645999
    column=pv:page_count, timestamp=1769779863343, value=15
    column=pv:product_detail_views, timestamp=1769779863343, value=3
  user_000000#8231320645999
    column=s:conversion_status, timestamp=1769779863343, value=browsed
    column=s:duration_seconds, timestamp=1769779863343, value=2368
    column=s:end_time, timestamp=1769779863343, value=2026-01-17T22:28:42
    column=s:start_time, timestamp=1769779863343, value=2026-01-17T21:49:14
    column=pv:page_count, timestamp=1769779971372, value=14
    column=pv:product_detail_views, timestamp=1769779971372, value=7
  user_000000#8231438065999
    column=s:conversion_status, timestamp=1769779971372, value=abandoned
    column=s:duration_seconds, timestamp=1769779971372, value=2927
    column=s:end_time, timestamp=1769779971372, value=2026-01-16T14:01:01
    column=s:start_time, timestamp=1769779971372, value=2026-01-16T13:12:14
2 row(s)
Took 0.0523 seconds
hbase(main):014:0> |
```

This HBase query retrieves session records for a specific user by scanning the ecom:user_sessions table and filtering rows whose keys start with the user's identifier using a PrefixFilter. It returns only selected session and page-view attributes (such as start/end time, duration, conversion status, and page counts) and limits the output to a small number of rows for concise inspection.

4. Data Processing with Apache Spark

Apache Spark was used for batch processing, data cleaning, and integration across MongoDB and HBase. Spark jobs were implemented using PySpark to process large JSON datasets efficiently.

4.1 Spark Batch Processing

Apache Spark was used as the central analytics engine to process data from two NoSQL databases: MongoDB and HBase. MongoDB stored structured e-commerce data such as users, products, and transactions, while HBase stored large-scale user session data optimized for fast access. By using spark_for_batch.py script Spark connected to MongoDB using the MongoDB Spark Connector to perform batch analytics with DataFrames and Spark SQL. HBase was accessed through the Thrift service using HappyBase, from which **only a controlled subset of 5,000 user session records** was loaded for analysis to demonstrate integration and querying. This approach enabled Spark to combine transactional and session data efficiently within a single analytical workflow while keeping the processing lightweight and manageable.

Data from MongoDB and HBase have accessed by spark successfully

```
== Reading from MongoDB ==
MongoDB counts:
  users      : 5000
  products   : 2000
  categories : 25
  transactions: 1000000

== Reading sessions from HBase (Thrift) with fallback to sessions.
json ===
SUCCESS: Loaded 5000 sessions from HBase (subset).
```

product co-occurrence

```
Top 10 product pairs bought together (by product_id):
+-----+-----+-----+
| prod_a | prod_b | cooc_count |
+-----+-----+-----+
| prod_00296|prod_01786|4
| prod_00422|prod_01604|3
| prod_01185|prod_01872|3
| prod_01374|prod_01454|3
| prod_00632|prod_01075|3
| prod_00775|prod_01887|3
| prod_00596|prod_01346|3
| prod_01032|prod_01201|3
| prod_00779|prod_01612|3
| prod_00119|prod_00819|3
+-----+-----+-----+
only showing top 10 rows
```

The batch analytics requirement was satisfied using product co-occurrence analysis

4.2 Spark SQL Analytics

Spark SQL was used to perform complex analytical queries, including sales trends. SQL queries were executed on cleaned DataFrames to demonstrate relational-style analytics at scale.

User spending summary SQL

```
print("\n== Spark SQL: Basic user spending summary ==")
user_spend_sql = """
SELECT
    user_id,
    COUNT(*) AS orders,
    SUM(total) AS total_spent,
    AVG(total) AS avg_order_value
FROM
    transactions
GROUP BY
    user_id
ORDER BY
    total_spent DESC
LIMIT 10
"""

print(user_spend_sql)
```

== Spark SQL: Basic user spending summary ==			
user_id	orders	total_spent	avg_order_value
user_000752	37	37568.44	1015.3632432433
user_002567	34	37015.67999999999	1088.6964705882351
user_001824	32	36268.54999999999	1133.392187499999
user_002574	36	35531.06	986.973888888888
user_002292	29	34155.61999999995	1177.779999999997
user_004959	31	33608.41	1084.1422580645162
user_003269	33	33518.47999999996	1015.71151515151
user_003631	38	33305.68000000001	876.4652631578949
user_002862	28	32810.0	1171.7857142857142
user_001066	34	32302.45	950.0720588235295

This analysis summarizes user purchasing behavior by showing how many orders each user placed, how much they spent in total, and their average order value. It helps identify high-value customers and supports customer segmentation, loyalty strategies, and targeted marketing.

Top products by revenue SQL

```
print("\n== Spark SQL: Top products by revenue ==")
top_products_sql = """
SELECT
    i.product_id,
    p.name,
    SUM(i.subtotal) AS total_revenue,
    SUM(i.quantity) AS total_quantity
FROM (
    SELECT explode(items) AS i
    FROM transactions
) t
JOIN products p
    ON t.i.product_id = p.product_id
GROUP BY
    i.product_id, p.name
ORDER BY
    total_revenue DESC
LIMIT 10
"""

print(top_products_sql)
```

== Spark SQL: Top products by revenue ==			
product_id	name	total_revenue	total_quantity
prod_00587	Secured Homogeneous Website	133056.0	270
prod_00089	User-Friendly Neutral Groupware	127392.47999999979	222
prod_01443	Profound Bi-Directional Database	126848.06999999982	249
prod_00988	Polarized Optimal Orchestration	126694.61999999975	234
prod_00702	Public-Key National Pricing Structure	126451.60000000006	232
prod_00319	Profit-Focused Holistic Encoding	124769.5899999998	251
prod_00781	Re-Contextualized Content-Based Workforce	124679.36000000012	221
prod_01559	Face-To-Face 24Hour Info-Mediaries	122453.85000000005	211
prod_00056	Persistent Contextually-Based Application	121485.06000000001	261
prod_01072	Seamless Systemic Concept	118895.99999999993	240

This output ranks products based on the total revenue they generate. It highlights the most valuable products in the catalog, combining both price and sales volume effects. These products are strong candidates for promotion, priority stocking, and strategic pricing decisions.

Revenue by category SQL

```
== Spark SQL: Revenue by category ==
+-----+-----+-----+
|category_id|total_revenue|total_quantity|line_items|
+-----+-----+-----+
|cat_024|4558428.340000067|16445|8244
|cat_002|4356217.779999871|15220|7521
|cat_010|4266253.809999972|16516|8300
|cat_021|4124216.9900000305|16145|8092
|cat_009|3847758.77|15267|7613
|cat_014|3781357.2100000256|16874|8498
|cat_020|3737179.2899999446|14694|7312
|cat_005|3593352.390000027|14486|7313
|cat_013|3549703.979999983|14655|7300
|cat_003|3460307.400000018|14371|7211
+-----+-----+-----+
print("\n--- Spark SQL: Revenue by category ---")
revenue_by_cat_sql = """
SELECT
    p.category_id,
    SUM(i.subtotal) AS total_revenue,
    SUM(i.quantity) AS total_quantity,
    COUNT(*) AS line_items
FROM (
    SELECT explode(items) AS i
    FROM transactions
) t
JOIN products p
    ON t.i.product_id = p.product_id
GROUP BY
    p.category_id
ORDER BY
    total_revenue DESC
LIMIT 10
"""
spark.sql(revenue_by_cat_sql).show(truncate=False)
```

This analysis shows how total sales are distributed across product categories. It identifies which categories generate the highest revenue and sales volume, highlighting the main business drivers. Categories with high revenue and quantity are core to overall performance, while others may represent niche or premium segments.

5. Integrated Analytics Workflows

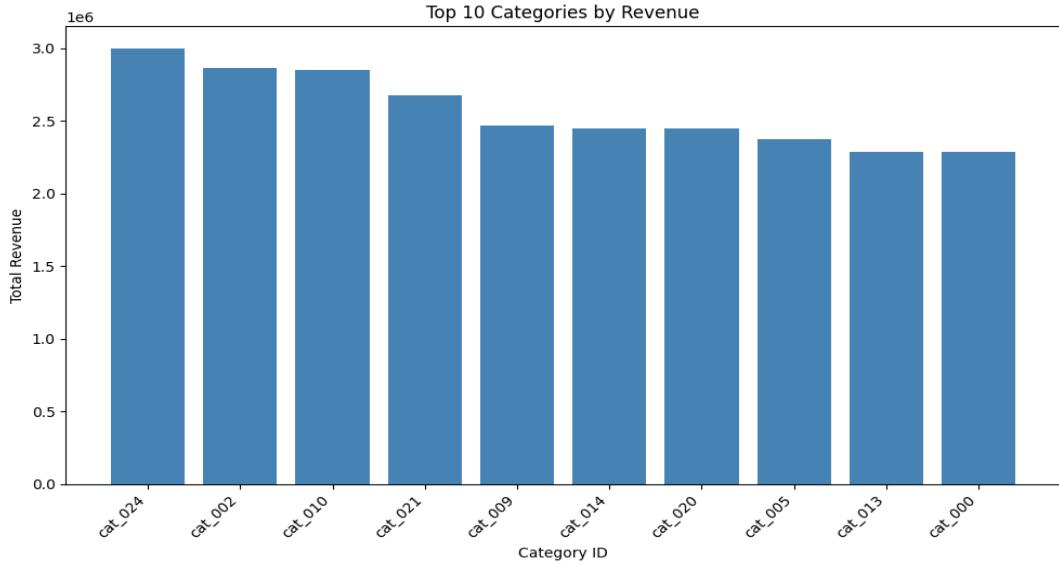
```
== Integrated CLV (spending) vs engagement (sessions) ==
+-----+-----+-----+-----+-----+-----+-----+
|user_id|country|registration_date|session_count|avg_session_duration|orders|total_spent|avg_order_value|clv_segment|
+-----+-----+-----+-----+-----+-----+-----+
|user_000752|BD|2025-10-30T01:57:48|54|1575.22|37|37568.44|1015.36|high_value
|user_002567|AE|2025-07-04T18:18:30|62|1789.87|34|37015.68|1088.7|high_value
|user_001824|KE|2025-10-04T23:00:59|68|1834.43|32|36268.55|1133.39|high_value
|user_002574|IE|2025-10-30T10:15:07|65|1677.0|36|35531.06|986.97|high_value
|user_002292|CL|2025-10-10T22:13:10|74|1728.0|29|34155.62|1177.78|high_value
+-----+-----+-----+-----+-----+-----+-----+
```

The Customer Lifetime Value (CLV) analysis integrates data from MongoDB and HBase using Apache Spark as the processing engine. User profile and transaction data were retrieved from MongoDB, while user engagement metrics (session count and duration) were obtained from HBase through the Thrift service. Spark was used to aggregate spending and engagement metrics per user and join these datasets into a unified analytical view. Users were then segmented into value categories (high, medium, low, or no-spend) based on total spending. To ensure efficient processing during development, only 5,000 session records were loaded from HBase. This workflow demonstrates how multi-source data can be combined to generate actionable customer insights.

6. Visualization and Business Insights

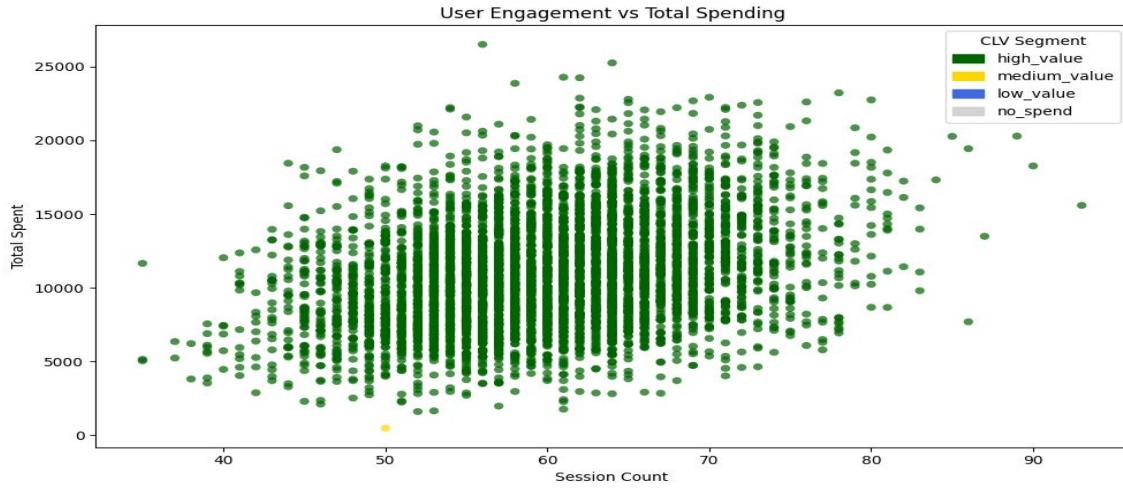
Visualizations were generated using Python (visualizations.py) to communicate analytical findings. These include revenue trends, customer segmentation, and product performance comparisons.

Revenue by Category



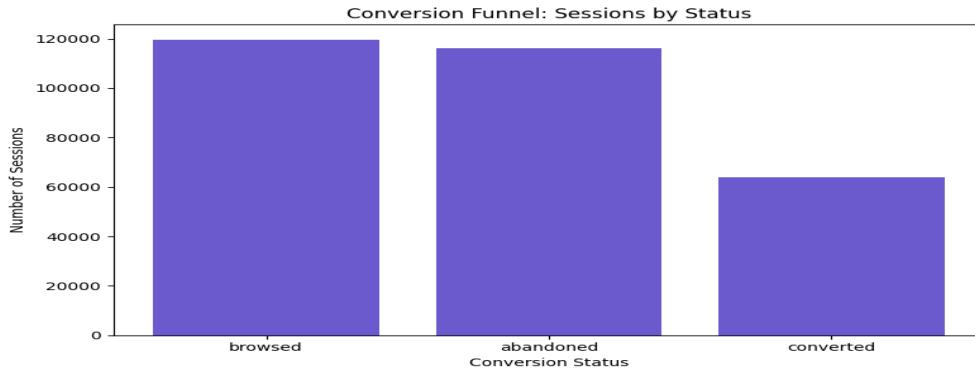
This bar chart shows the **top 10 product categories by total revenue**. The category **cat_024** has the highest revenue, making it the strongest-performing category in the system. Overall, the chart helps identify which categories contribute most to sales and where the business should focus marketing, inventory, and growth strategies.

User engagement and Total spending



This visualization shows that users who engage more frequently with the platform generally spend more, indicating a positive relationship between session count and total spending. High-value customers cluster at higher engagement and spending levels, while some highly engaged users still spend little, showing that engagement alone does not always lead to conversion. This highlights the value of combining session data with transaction data for accurate customer segmentation.

Conversion Funnel



This bar chart shows how user sessions move through the buying process: **browsed**, **abandoned**, and **converted**. Most users only browse or leave without completing a purchase, while fewer sessions end in a successful sale. This means many users drop out before buying, showing where improvements like an easier checkout or reminders could help increase sales.

7. Scalability Considerations

The proposed architecture is scalable. MongoDB supports sharding, HBase scales horizontally by adding region servers, and Spark can scale across clusters to handle increased data volume and complexity.

8. Limitations and Future Work

The project uses synthetic data and a simplified HBase setup. Future work could include real-time stream processing with Spark Streaming, Kafka integration, and deployment on a cloud environment.

9. Conclusion

This project successfully demonstrates the design and implementation of a distributed, multi-model analytics architecture using MongoDB, HBase, and Apache Spark. By strategically selecting each technology based on data structure and access patterns, the system efficiently handles transactional data, time-series session data, and large-scale analytical processing. Spark serves as the central analytics engine, enabling integrated analysis such as Customer Lifetime Value (CLV) by combining user profiles, transactions, and engagement metrics. Overall, the project highlights how modern big data technologies can be effectively integrated to support scalable analytics and generate meaningful business insights in an e-commerce context.