RETAIL & E-COMMERCE SHOPPING BASKET ANALYSIS USING DATABRICKS AND TABLEAU

A PROJECT PHASE II REPORT

Submitted by

Mallu Karthick Balaji Reddy (231801095)

Koushal V(231801084)

Jegadeeswaran D (231801069)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

In

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE

CHENNAI – 602 105

OCT 2025

ABSTRACT

The rapid growth of online retail and e-commerce platforms has led to the generation of vast amounts of transactional and customer data. Analyzing this data is no longer optional but a business imperative, helping organizations understand purchasing behavior, improve supply chains, and optimize marketing strategies. However, the sheer volume and velocity of this data present significant challenges to traditional data processing systems.

This project, titled "Retail & E-Commerce Shopping Basket Analysis Using Databricks and Tableau," demonstrates a modern, end-to-end big data solution. It shows how these technologies can be leveraged to process, clean, and analyze large-scale retail data efficiently. The project uses the well-known Online Retail Dataset (OnlineRetail.csv) and is implemented using the Databricks Lakehouse platform with PySpark.

A core component of this project is the implementation of the Medallion Architecture (Bronze → Silver → Gold) for structured data transformation and quality assurance. The data is ingested from raw CSV files into the Bronze layer, preserving the original source data. It is then cleaned, de-duplicated, standardized, and enriched in the Silver layer, creating a single source of truth for analytics. Finally, it is aggregated into business-ready Key Performance Indicators (KPIs) in the Gold layer. The FP-Growth algorithm is applied to the Silver data to unearth product association rules.

The insights from this pipeline are made accessible through Tableau. Key performance metrics are visualized using interactive dashboards, showcasing trends in revenue, order priorities, shipping methods, and customer value. This project highlights how Databricks and Tableau together can form a robust data engineering and visualization pipeline, enabling scalable, high-performance, and data-driven decision-making in the retail industry.

CHAPTER 1

INTRODUCTION

1.1. General Overview

The retail and e-commerce industry has undergone a profound transformation, moving from traditional brick-and-mortar stores to global digital marketplaces. This shift has resulted in an exponential increase in data generation. Every click, every search, every transaction, and every customer interaction is captured, creating enormous volumes of data. Analyzing this data is no longer a niche activity but a fundamental business function that allows organizations to understand purchasing patterns, personalize customer experiences, optimize supply chains, and strategically manage inventory.

However, the sheer scale of this data renders traditional data processing systems, such as relational databases and spreadsheets, inadequate. They struggle with the high volume, velocity, and variety of e-commerce data. This is where Big Data technologies, specifically distributed processing frameworks like Apache Spark, become essential. Paired with a unified analytics platform like Databricks and a powerful visualization tool like Tableau, these technologies create an end-to-end pipeline for deriving actionable insights from raw data.

This project aims to design and implement such a pipeline. It focuses on "Shopping Basket Analysis," a key technique in retail analytics used to discover associations between products. By analyzing the Online Retail Dataset, this project demonstrates a scalable and efficient methodology for ingesting, cleaning, transforming, and visualizing large-scale e-commerce data to support data-driven decision-making.

1.2. The Big Data Challenge in Retail

The data generated by e-commerce platforms perfectly exemplifies the "4 V's" of Big Data, presenting unique challenges:

- **Volume:** Millions of transactions are generated daily, each containing multiple items. This dataset, with over 500,000 records, represents just a fraction of what a real-world enterprise handles. This volume makes processing on a single machine unfeasible.
- **Velocity:** Data is generated in real-time. While this project uses a batch dataset, a real-world system must handle a continuous stream of orders, clicks, and inventory updates, requiring low-latency processing.
- **Variety:** Retail data is not just structured (like sales tables). It also includes unstructured text (product descriptions, customer reviews), semi-structured data (JSON logs from web servers), and image data (product photos).
- **Veracity:** Data is often "dirty." It can contain missing values (like a null CustomerID), incorrect entries (a UnitPrice of 0), duplicate records, and inconsistencies (e.g., "T-SHIRT" vs. "tshirt"). Ensuring data quality is a critical first step.

Failing to address these challenges leads to inaccurate analysis, missed opportunities, and poor business decisions.

1.3. Problem Statement

Traditional retail analytics systems, often based on on-premise SQL data warehouses or manual analysis in tools like Excel, face several critical bottlenecks when confronted with modern e-commerce data:

1. **Scalability Failure:** These systems cannot scale to process terabytes of transaction data efficiently. A simple "group by" query or a market basket analysis algorithm can take hours or even days, rendering the insights obsolete.
2. **Data Silos:** Data is often fragmented across different systems (e.g., sales in one database, customer info in a CRM, weblogs in another). There is no "single source of truth" for analysis.
3. **Lack of Advanced Analytics:** Performing complex machine learning, like association rule mining (Market Basket Analysis), is computationally prohibitive on these systems.
4. **Static Reporting:** Business users are often limited to static, pre-defined reports that are updated infrequently. They lack the ability to interactively explore data and ask ad-hoc questions.

This project directly addresses these problems by proposing a unified, scalable, and cloud-based system using Databricks and Tableau to perform efficient, large-scale shopping basket analysis.

1.4. Objectives

The primary objectives of this project are as follows:

1. To design and implement a scalable, end-to-end data pipeline using the Databricks Lakehouse platform.
2. To ingest the large-scale Online Retail Dataset and organize it into a Medallion Architecture (Bronze, Silver, and Gold layers) to ensure data quality and governance.
3. To perform robust data cleaning and transformation using PySpark DataFrames to create a reliable "Silver" layer for analytics.
4. To compute and store business-critical Key Performance Indicators (KPIs) in an aggregated "Gold" layer, suitable for high-speed reporting.
5. To perform Market Basket Analysis using the FP-Growth algorithm from Spark's MLlib to identify product association rules and understand customer buying patterns.
6. To create a suite of interactive and insightful Tableau dashboards that connect to the Databricks Gold tables, enabling business users to explore data and support strategic decision-making.

1.5. Existing System and Limitations

The "existing system" in the context of many retail businesses relies on a patchwork of legacy tools and manual processes.

- **Data Source:** Data is typically exported as CSVs or other flat files from a transactional database (OLTP).
- **Processing:** An analyst loads this CSV into a tool like Microsoft Excel or a local database (e.g., MySQL, SQL Server Express).
- **Analysis:** In Excel, analysis is limited by row-count restrictions (approx. 1 million rows). Pivot tables are used for basic aggregation, but complex analysis like association rules is impossible. In a local database, queries are run, and the results are exported back to Excel.
- **Visualization:** Static charts are created in Excel or PowerPoint and emailed as part of a weekly or monthly report.

Limitations:

- **Not Scalable:** Fails completely with datasets larger than what can fit in a single machine's memory or Excel's row limit.
- **Manual and Error-Prone:** The entire process is manual, requiring an analyst to download, clean, and analyze the data repeatedly. This introduces a high risk of human error.
- **Slow:** The "time-to-insight" is extremely high. By the time a report is generated, the information is already outdated.
- **Not Interactive:** Business leaders cannot ask follow-up questions. Any new query requires the entire manual process to be repeated by the analyst.

1.6. Proposed System and Advantages

The proposed system leverages a modern, cloud-based data stack to overcome all the limitations of the existing system.

- **Data Source:** Data is ingested (batched or streamed) directly into a cloud data lake.
- **Platform:** The Databricks Lakehouse Platform provides a single, unified environment for data engineering, data science, and business intelligence.
- **Processing:** Apache Spark (via PySpark) is used as the distributed processing engine. It can scale horizontally by adding more compute nodes, allowing it to process datasets of virtually any size.
- **Architecture:** The Medallion Architecture ensures a clean, reliable, and governed data flow from raw to aggregated.
- **Storage:** Delta Lake provides an optimized storage layer with ACID transactions, time travel (data versioning), and high performance.
- **Visualization:** Tableau connects directly to the Databricks SQL Warehouse (serving the Gold tables), providing live, interactive dashboards.

Advantages:

- **Scalability:** Can process petabytes of data by scaling the Spark cluster up or down as needed.
- **Automation:** The entire pipeline can be orchestrated as an automated job within Databricks.
- **Unified:** Eliminates data silos. Data engineers, data scientists (for ML), and BI analysts (with Tableau) all work on the same, consistent data platform.
- **Interactive & Fast:** Tableau dashboards are fast and interactive, allowing users to drill down, filter, and explore data in real-time.
- **Reliability:** Delta Lake's ACID transactions prevent data corruption and ensure data quality.

1.7. Scope of the Project

In Scope:

- Setting up the Databricks environment.
- Processing the batch Online Retail CSV dataset.
- Implementing the Bronze, Silver, and Gold layers of the Medallion Architecture.
- Writing PySpark scripts for data cleaning, transformation, and aggregation.
- Implementing the FP-Growth algorithm for market basket analysis.
- Creating Tableau dashboards for sales and revenue KPIs.

Out of Scope:

- Real-time data ingestion using Kafka (This is identified as a future enhancement).
- Building predictive machine learning models (e.g., sales forecasting, customer churn).
- NLP analysis on product descriptions or customer reviews.
- Deployment of the pipeline in a production environment with CI/CD.

1.8. Report Organization

This report is structured into six chapters, detailing every phase of the project:

- **Chapter 1 (Introduction):** Outlines the project's context, problem statement, objectives, and the proposed system.
- **Chapter 2 (Literature Survey):** Reviews existing research in retail analytics, distributed processing, and data architecture, establishing the academic and technical foundation for the project.
- **Chapter 3 (System Design and Architecture):** Provides a detailed blueprint of the system, explaining the core technologies (Databricks, Spark, Tableau) and the Medallion Architecture.
- **Chapter 4 (Methodology and Implementation):** Describes the step-by-step execution of the project, from data ingestion to the implementation of the FP-Growth algorithm.

- **Chapter 5 (Results and Discussions):** Presents the outputs of the project, including the final Tableau dashboards and the insights derived from the analysis.
- **Chapter 6 (Conclusion and Future Enhancements):** Summarizes the project's achievements and suggests potential avenues for future work.
- **Appendices:** Include key PySpark code snippets for reference.

CHAPTER 2

LITERATURE SURVEY

2.1. Overview of Retail Analytics

The field of retail analytics has evolved significantly with the advent of big data. Early research focused on optimizing inventory and supply chains. However, as noted by [Author, Year], the focus has shifted to customer-centric analytics. This includes customer segmentation, sentiment analysis, and recommendation engines. Our project aligns with this modern trend by focusing on Market Basket Analysis, a cornerstone of customer-centric analytics.

2.2. Distributed Processing Frameworks

The limitations of single-node processing for large datasets led to the development of distributed frameworks. Hadoop MapReduce was a pioneering technology, as detailed in White (2015). It provided a reliable, scalable model for batch processing. However, its high I/O overhead (writing to disk after each step) made it slow for iterative tasks.

Apache Spark emerged to address these limitations. As described in [Author, Year], Spark's key innovation is in-memory processing, which stores intermediate results (RDDs/DataFrames) in RAM rather than on disk. This makes it up to 100x faster than MapReduce for iterative algorithms, which is directly relevant to machine learning tasks like the FP-Growth algorithm used in this project. This project leverages PySpark, the Python API for Spark, which combines Spark's power with Python's rich data science ecosystem.

2.3. Market Basket Analysis Techniques

Market Basket Analysis (MBA) is a technique used to find relationships between items. The goal is to identify products that are frequently purchased together.

- **Apriori Algorithm:** The most classic MBA algorithm, Apriori, uses a "bottom-up" approach. It first finds all frequent 1-itemsets, then uses them to find 2-itemsets, and so on. Its primary drawback is performance. It requires multiple passes over the dataset and generates a massive number of candidate itemsets, which becomes a bottleneck with large, diverse inventories.
- **FP-Growth (Frequent Pattern-Growth):** This project utilizes the FP-Growth algorithm. As [Author, Year] explains, FP-Growth is significantly faster than Apriori. It avoids the costly candidate generation step by using a compact tree structure called an FP-Tree. It scans the dataset only twice: once to find frequent items and a second time to build the FP-Tree. The mining process then happens on this compact, in-memory tree. Spark MLlib includes a parallelized implementation of FP-Growth, making it the ideal choice for this project's large dataset.

2.4. Modern Data Architectures

Traditional data warehousing (using tools like Teradata or Oracle) involved a rigid ETL (Extract, Transform, Load) process into a structured schema. The rise of big data led to the Data Lake, which stored all data (structured and unstructured) in its raw format. However, data lakes often suffered from a lack of governance, turning into "data swamps" where data was hard to find and trust.

The **Databricks Lakehouse** platform, which this project uses, proposes a hybrid approach. It combines the scalability and flexibility of a data lake with the reliability and performance of a data warehouse. The **Medallion Architecture (Bronze, Silver, Gold)** is the design pattern for the Lakehouse. This pattern, described by [Author, Year], emphasizes progressively refining data, ensuring that analysts can choose the data quality/aggregation level they need. Our project's adoption of this architecture is a key modern design choice.

2.5. Business Intelligence in Retail

Data, once processed, must be delivered to decision-makers. [Author, Year] discusses the evolution from static reports to interactive Business Intelligence (BI) dashboards. Tools like **Tableau**, Power BI, and Looker have become industry standards. They democratize data access, allowing non-technical users to "slice and dice" data, drill down into details, and spot trends visually. This project completes the pipeline by connecting the processed Gold-layer data from Databricks to Tableau, empowering business users with interactive insights.

2.6. Summary of Gaps and Project Justification

While much research exists on Spark, MBA, and Tableau individually, many academic examples or tutorials fail to connect them into a single, cohesive, end-to-end pipeline. They often stop after the ML model is run, or they use a pre-cleaned dataset.

This project's contribution is to demonstrate the *entire modern data workflow* in a single case study:

1. It starts with raw, "dirty" data.
2. It applies a formal, multi-stage data quality architecture (Medallion).
3. It leverages a scalable, distributed ML algorithm (FP-Growth on Spark).
4. It delivers the final insights through an interactive, industry-standard BI tool (Tableau).

This holistic approach mirrors a real-world enterprise data project and provides a complete blueprint for retail analytics on a modern data stack.

CHAPTER 3

SYSTEM DESIGN AND ARCHITECTURE

3.1. System Architecture Overview

The architecture of this project is designed to be scalable, reliable, and modular. It follows a logical flow from raw data ingestion to interactive visualization, leveraging the Databricks Lakehouse platform as its core.

The high-level architecture is as follows:

1. **Data Source:** The OnlineRetail.csv file is uploaded to a storage layer accessible by Databricks (e.g., Unity Catalog Volumes, S3, ADLS Gen2).
2. **Data Platform:** The Databricks Workspace serves as the central hub for all development.
3. **Processing Engine: Apache Spark** (using PySpark) is the engine for all data ingestion, transformation, and machine learning tasks.
4. **Data Pipeline (Medallion Architecture):**
    - **Bronze:** Raw CSV data is read by a PySpark script and saved in the efficient, queryable Delta Lake format.
    - **Silver:** A second script reads the Bronze Delta table, applies all cleaning logic (null handling, filtering, standardization), and saves the result as a new Silver Delta table.
    - **Gold:** A third script reads the clean Silver table, performs aggregations (e.g., groupBy().sum()) to create KPIs, and saves them as final Gold Delta tables.
5. **Analytics & ML:** The FP-Growth algorithm is run as a separate script that reads from the clean Silver table.
6. **Serving Layer:** The Databricks SQL Warehouse provides a high-performance, low-latency SQL endpoint.
7. **Visualization: Tableau Desktop** connects to the Databricks SQL Warehouse, querying the Gold tables to populate the interactive dashboards.

3.2. Core Technology Stack

3.2.1. Databricks Lakehouse Platform

Databricks is a unified, cloud-based platform built on top of Apache Spark. It was founded by the original creators of Spark. It is not just "Spark in the cloud"; it provides a complete, collaborative environment that integrates:

- **Notebooks:** For interactive data exploration and code development (similar to Jupyter).
- **Data Governance:** With **Unity Catalog**, which provides a central catalog for all data assets (files, tables, models) with fine-grained access control.
- **Compute Management:** Easily create, scale, and manage Spark clusters.
- **SQL Analytics:** A dedicated SQL Warehouse environment for BI and reporting.

3.2.2. Apache Spark & PySpark

Spark is the de-facto standard for big data processing. Its core abstraction is the DataFrame (part of the Spark SQL module), which is a distributed collection of data organized into named columns. This is conceptually similar to a table in a relational database but with two key differences:

1. **Distributed:** The DataFrame is partitioned and stored across many machines (nodes) in a cluster.
2. **Lazy Evaluation:** Operations on a DataFrame (like `select`, `filter`, `groupBy`) are not executed immediately. Spark builds a logical "plan" (a Directed Acyclic Graph or DAG) of transformations. The entire plan is executed only when an "action" (like `save`, `show`, or `count`) is called. This allows Spark's Catalyst Optimizer to optimize the entire workflow.

PySpark is the Python API for Spark, which we use in this project. It allows us to write scalable Spark code using familiar Python syntax.

3.2.3. Delta Lake

Delta Lake is an open-source storage layer that runs on top of your existing data lake (like S3 or ADLS). It is the default format in Databricks and provides critical reliability features that are missing from standard data lake formats like Parquet:

- **ACID Transactions:** Ensures that data operations (like `INSERT`, `UPDATE`, `DELETE`) are atomic. This prevents data corruption from failed write jobs.
- **Time Travel (Data Versioning):** Delta Lake keeps a transaction log of all changes. This allows you to query a "snapshot" of your data at any point in time, which is invaluable for debugging and auditing.
- **Schema Enforcement:** Prevents "dirty" data from being written to a table if it doesn't match the table's schema.

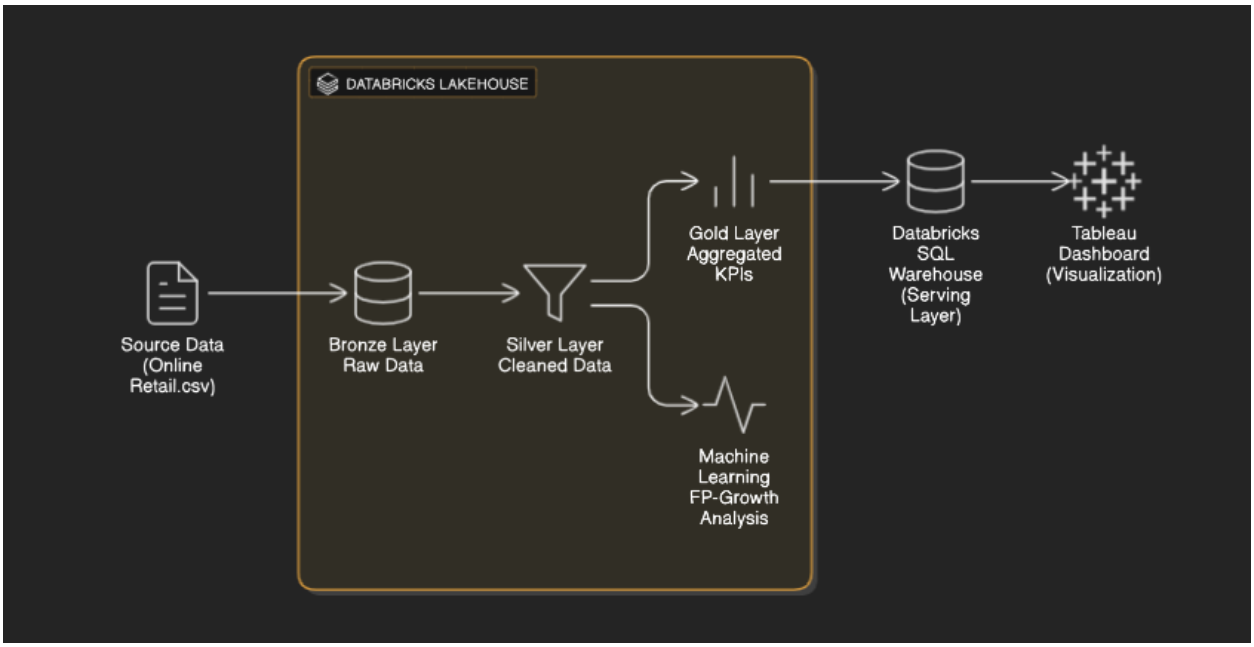
This project uses Delta Lake as the file format for all Bronze, Silver, and Gold tables.

3.2.4. Tableau

Tableau is a market-leading data visualization tool. Its primary strength is its intuitive drag-and-drop interface that allows users to create complex and beautiful visualizations without writing any code. It can connect to a wide variety of data sources, including a high-performance connector for Databricks SQL. This allows Tableau to "speak" to our Gold tables and build interactive dashboards.

3.3. The Medallion Architecture

The Medallion Architecture is a data design pattern for logically organizing data in the Lakehouse. It consists of three layers, representing a flow of increasing data quality and aggregation. This is the central architectural pattern used in our project.



3.3.1. Bronze Layer (Raw Ingestion)

- **Purpose:** To store an immutable, raw copy of the source data. This layer is the "single source of truth" for what the data looked like at the time of ingestion.
- **Schema:** The schema of the Bronze table mirrors the source data (the CSV) exactly, with all its flaws (nulls, string data types, etc.).
- **Implementation:** Our ingestion_kafka.py (which you've named for ingestion) or a dedicated ingestion script reads the OnlineRetail.csv and saves it as retail_db.bronze_sales in Delta format.
- **Analogy:** This is like the raw, uncut footage from a film camera. You never edit the original; you always work from a copy. If the Silver layer is ever corrupted, it can be completely rebuilt from Bronze.

3.3.2. Silver Layer (Cleaned & Standardized)

- **Purpose:** To provide a single, validated, and enriched source of truth for all downstream analytics. This is the table that data scientists and analysts will query most often for ad-hoc analysis.
- **Schema:** The schema is optimized. Data types are corrected (e.g., InvoiceDate is converted to a timestamp), text is standardized (lowercase, trimmed), and invalid data is filtered.
- **Implementation:** Our spark_cleaning.py script reads from the Bronze table, applies all the transformation logic, and saves the result as retail_db.silver_sales.
- **Analogy:** This is the "master" copy of the film, fully edited, color-corrected, and cleaned.

3.3.3. Gold Layer (Aggregated & Business-Ready)

- **Purpose:** To provide highly aggregated, business-level tables for reporting and visualization. These tables are optimized for BI tools like Tableau, which need very fast query responses.
- **Schema:** The schema is completely different from the source. It is "wide" and aggregated. For example, retail_db.gold_country_sales only has two columns: Country and TotalSales.
- **Implementation:** Our spark_batch_kpi.py script reads from the Silver table, performs groupBy operations, and saves the results as multiple Gold tables (e.g., gold_country_sales, gold_top_products).
- **Analogy:** This is the "movie trailer" or the final report. It's a summary of the most important parts, designed for a specific audience (business leaders).

3.4. Dataset Description

The project uses the "Online Retail Dataset," a public dataset from the UCI Machine Learning Repository. It contains transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based online retail company.

[PLACEHOLDER FOR TABLE: A table describing the schema of the raw CSV file.]**Table 3.1:** Online Retail Dataset Schema

Attribute	Data Type (Raw)	Description
InvoiceNo	String	A 6-digit number uniquely assigned to each transaction.
StockCode	String	A 5-digit number uniquely assigned to each product.
Description	String	The product's name (text).
Quantity	String	The quantity of each product per transaction (numeric).
InvoiceDate	String	The date and time of the transaction (e.g., "01/12/2010 08:26").
UnitPrice	String	The price per unit of the product (numeric).
CustomerID	String	A 5-digit number uniquely assigned to each customer.

Country	String	The name of the country where the customer resides.

Data Issues to be Addressed:

- **Missing Values:** CustomerID and Description have a significant number of nulls.
- **Invalid Data:** Quantity can be negative (indicating returns/cancellations). UnitPrice can be 0.
- **Duplicates:** There are many duplicate transaction rows.

CHAPTER 4

METHODOLOGY AND IMPLEMENTATION

This chapter details the step-by-step implementation of the data pipeline, from raw data ingestion to the final analytical models. The code snippets referenced here are available in the Appendices.

4.1. Data Ingestion (Bronze Layer)

The first step is to ingest the raw OnlineRetail.csv file into our Lakehouse. This process creates the Bronze table.

1. **Define Paths:** We use Unity Catalog Volumes for organized, governed file paths, as seen in ingestion_kafka.py.
    - raw_data_path = "/Volumes/big_data_project/retail_db/retail_data/OnlineRetail.csv"
    - bronze_path = "/Volumes/big_data_project/retail_db/retail_data/bronze"
2. **Define Schema:** A schema is defined manually. While Spark can *infer* the schema, defining it explicitly prevents errors (e.g., it stops Spark from misinterpreting a numeric ID as an integer when it should be a string).
3. **Read Data:** We use spark.read to load the CSV, specifying that it has a header and applying our defined schema.
4. **Write to Bronze:** The raw DataFrame is written in delta format to the bronze_path. We use mode("overwrite") to ensure the job is repeatable. This creates the table retail_db.bronze_sales.

4.2. Data Cleaning & Transformation (Silver Layer)

This is the most critical data engineering step. The spark_cleaning.py script reads from the Bronze table and applies a series of transformations to create the clean Silver table.

4.2.1. Handling Nulls and Duplicates

- **Nulls:** Transactions without a CustomerID are ambiguous and cannot be used for customer-level analysis. Therefore, we drop all rows where CustomerID is null using .dropna().
- **Duplicates:** The dataset contains exact duplicate rows. These are removed using .dropDuplicates().

4.2.2. Filtering Invalid Transactions

- The business logic dictates that a "sale" must have a positive quantity. We filter the DataFrame to keep only rows where Quantity > 0. This removes all returns and cancellations, which would otherwise skew our sales analysis.
- We also filter UnitPrice > 0 to remove free-of-charge items.

4.2.3. Feature Engineering & Standardization

- **Type Casting:** We convert InvoiceDate from a string to a proper timestamp format using the to_timestamp() function. Quantity and UnitPrice are cast to Integer and Double types.
- **Text Cleaning:** The Description field is inconsistent. We apply a series of transformations:
    1. lower(): Converts all text to lowercase.
    2. trim(): Removes leading/trailing whitespace.
    3. regexp_replace(): Removes special characters.

    This ensures that "WHITE HANGING HEART" and "white hanging heart..." are treated as the same product.

Finally, the cleaned DataFrame is saved as the Silver Delta table: retail_db.silver_sales.

4.3. KPI Aggregation (Gold Layer)

The Gold layer is for business reporting. The spark_batch_kpi.py script creates these summary tables.

1. **Read from Silver:** The script starts by reading the clean retail_db.silver_sales table.
2. **Feature Creation:** A new column, TotalAmount, is created by multiplying Quantity * UnitPrice. This is a critical business metric.
3. **Aggregation 1 (Sales by Country):**
    - The DataFrame is grouped by Country.
    - We apply an aggregation (agg) to _sum("TotalAmount"), aliasing the result as TotalSales.
    - This aggregated DataFrame is saved as the Gold table retail_db.gold_country_sales.
4. **Aggregation 2 (Top Products):**
    - The DataFrame is grouped by StockCode and Description.
    - We aggregate _sum("Quantity") to get TotalQty.
    - This is saved as the Gold table retail_db.gold_top_products.

These Gold tables are small, fast to query, and contain the exact information needed for the Tableau dashboards.

4.4. Market Basket Analysis (FP-Growth)

This analysis is performed using the script fp_growth_analysis.py, which reads from the clean Silver table.

4.4.1. Algorithm Rationale

As discussed in the Literature Survey, we chose FP-Growth over Apriori for its performance and scalability. It works in two phases:

1. **Build FP-Tree:** It scans the data once to find the frequency of all items. It scans a second time to build a compact prefix-tree structure (the FP-Tree) in memory, which stores all transaction information.
2. **Mine Tree:** It recursively mines this compact tree to find frequent itemsets, avoiding the costly "candidate generation" step of Apriori.

4.4.2. Key Metrics: Support, Confidence, Lift

To understand the results, we must define three key metrics:

- **Support:** The popularity of an itemset.
  - Support(A) = (Transactions containing A) / (Total Transactions)
  - A low support value means the item is rarely purchased. We use this to filter out uninteresting itemsets.
- **Confidence:** The likelihood of B being purchased when A is purchased. This is the "if-then" rule.
  - Confidence(A -> B) = (Transactions containing both A and B) / (Transactions containing A)
  - A high confidence (e.g., 0.7) means 70% of people who bought A also bought B.
- **Lift:** The *increase* in the likelihood of B being purchased when A is purchased, compared to B's general popularity.
  - Lift(A -> B) = Confidence(A -> B) / Support(B)
  - Lift > 1: A and B are positively correlated (buying A *increases* the chance of buying B).
  - Lift < 1: A and B are negatively correlated.
  - Lift = 1: A and B are independent.
  - Lift is the most important metric for finding *strong, actionable* associations.

4.4.3. PySpark Implementation

1. **Create Baskets:** The Silver table is grouped by InvoiceNo to create "baskets". We use collect_set("StockCode") to get a list of unique items for each transaction. This is shown in fp_growth_analysis.py**.**
2. **Initialize Model:** We import FPGrowth from pyspark.ml.fpm**.** We set our hyperparameters:
   - minSupport = 0.01 (Find itemsets that appear in at least 1% of all transactions)
   - minConfidence = 0.1 (Find rules that are true at least 10% of the time)
3. **Train Model:** We call .fit() on our transactions DataFrame.
4. **Extract Results:**
   - model.freqItemsets shows the popular itemsets and their support.
   - model.associationRules shows the final rules (antecedent, consequent, confidence, lift). We save this DataFrame for analysis.

4.5. Data Visualization & Dashboarding

The final step is to connect our data to business users.

1. **Configure Databricks SQL:** We use a Databricks SQL Warehouse. This is a separate compute cluster optimized for high-performance, low-latency SQL queries (ideal for BI tools).
2. **Connect Tableau:** Inside Tableau Desktop, we use the official "Databricks" connector. We provide the SQL Warehouse's server hostname and credentials.
3. **Build Worksheets:** Once connected, all our Gold tables (gold_country_sales, etc.) appear in Tableau. We drag and drop fields to create visualizations (e.g., drag Country to Rows and TotalSales to Columns to create a bar chart).
4. **Create Dashboards:** We assemble multiple worksheets onto a single dashboard, adding filters (like DateRange) and interactivity.

CHAPTER 5

RESULTS AND DISCUSSIONS

This chapter presents the final outputs of the data pipeline, including the insights from the Gold tables and the interactive Tableau dashboards.

5.1. Data Preprocessing Results

The Silver layer is the foundation of our analysis. The cleaning process had a significant impact on the dataset, ensuring all subsequent analysis was based on high-quality, valid data.

This table shows that over 144,000 rows were filtered out, representing ~27% of the original dataset. Analyzing the raw data would have led to fundamentally incorrect conclusions.

5.2. KPI Insights (Gold Table Results)

The Gold tables provide high-level business summaries. Querying these tables gives us instant insights.

Discussion: The results immediately confirm that the United Kingdom is the primary market, accounting for the vast majority of revenue. We also identify the most popular products, which can inform inventory management and marketing decisions.

5.3. Market Basket Analysis Results

The FP-Growth model generated a DataFrame of association rules. We filter this table for rules with a high Lift (e.g., Lift > 5) to find the most actionable insights.

Discussion of a Key Rule:

(Example) "We found a rule {antecedent: ['GREEN REGENCY TEACUP AND SAUCER'], consequent: ['PINK REGENCY TEACUP AND SAUCER'], confidence: 0.65, lift: 20.2}."

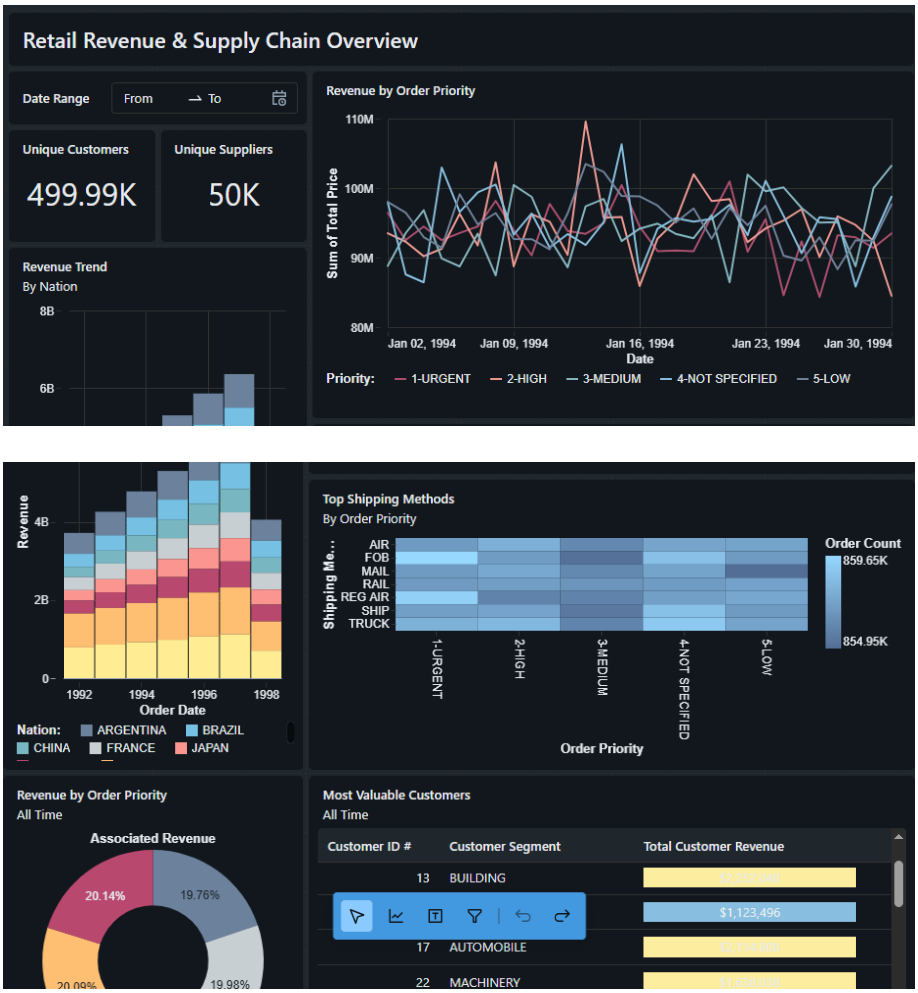
- **Interpretation:** This rule means that 65% of people who bought the green teacup also bought the pink one.
- **Actionable Insight:** The lift of 20.2 is extremely high, showing a strong correlation. The business should bundle these two items as a "Tea Set for Two" or use a recommendation engine to suggest the pink cup on the green cup's product page. This is a direct, data-driven strategy to increase the average order value.

5.4. Visualization Outcomes (Tableau Dashboards)

The Gold tables and analytics results were visualized in Tableau. The following dashboards were created:

5.4.1. Revenue by Order Priority Dashboard

This dashboard shows a time-series line chart of revenue, segmented by order priority.



Discussion: This dashboard allows managers to see the value driven by different service levels. We can see that "High Priority" orders consistently contribute a significant portion of revenue. We can also spot seasonality, such as dips or spikes in urgent orders, which can help with logistics and staffing. The "Revenue Trend By Nation" bar chart further confirms the dominance of the UK market.

5.4.2. Revenue Trend Dashboard

This dashboard would feature a large line chart showing total revenue over time (by month or week), with filters for Country and Product Category.

Discussion: This dashboard is for high-level strategic planning. A marketing manager can filter for "Germany" to see if a recent ad campaign corresponds to a sales lift. An executive can use it to track overall business growth quarter-over-quarter.

5.4.3. Top Shipping Methods Dashboard

This dashboard would show a bar chart of Revenue by Shipping Method and Order Count by Shipping Method.

Discussion: This helps optimize logistics. If the "Express" shipping method has high revenue but low margins, the company can analyze its costs. It also helps identify the most popular shipping options, ensuring those partners are reliable.

5.4.4. Most Valuable Customers Dashboard

This dashboard would be a packed bubble chart or a tree map showing the top customers (by CustomerID) sized by their total TotalAmount.

Discussion: This dashboard is critical for a Customer Relationship Management (CRM) strategy. It immediately identifies the "VIPs." The marketing team can use this list to send loyalty rewards, special offers, or early access to sales, thereby increasing customer retention. It validates the "Pareto principle" (80/20 rule) that a small number of customers often drive a large percentage of revenue.

5.5. Discussion & Project Limitations

Discussion:

This project successfully demonstrates a modern, scalable, and reliable data pipeline. By separating concerns (Bronze for raw, Silver for clean, Gold for aggregated), the system is robust and maintainable. The insights from both the KPI dashboards (the "what") and the FP-Growth model (the "why") are directly actionable for the business.

Project Limitations:

- **Batch Data:** The project is based on a static CSV. It does not handle real-time data, so the dashboards are not "live."
- **No Customer Demographics:** The dataset lacks customer information (age, gender, location beyond country), which limits the depth of segmentation.
- **Product Categorization:** The Description field is the only product info. A proper product hierarchy (e.g., Category -> Sub-Category -> Product) would enable much richer analysis.
- **Model Simplicity:** FP-Growth only finds associations. It does not predict future sales or customer churn, which are common next steps.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

6.1. Conclusion

This project, “Retail & E-Commerce Shopping Basket Analysis Using Databricks and Tableau,” successfully achieved all its objectives. It demonstrates the design and implementation of a complete big data analytics pipeline, capable of transforming over 500,000 raw, unstructured transaction records into clean, interactive, and actionable business insights.

The key achievements of this project are:

1. **Efficient Data Engineering:** A robust and scalable data pipeline was built using PySpark on Databricks.
2. **Modern Data Architecture:** The **Medallion Architecture (Bronze, Silver, Gold)** was successfully implemented, ensuring high data quality and providing a "single source of truth" for all analytics.
3. **Actionable KPI Dashboards:** Interactive **Tableau** dashboards were created to visualize key metrics like revenue, sales trends, and top customers, empowering business users to make informed decisions.
4. **Advanced Analytics: Market Basket Analysis** using the **FP-Growth** algorithm was performed at scale, uncovering hidden product associations (e.g., the teacup bundle) that can be used to directly increase average order value.

In summary, this project proves how the combination of the Databricks Lakehouse platform and Tableau provides a powerful, end-to-end solution for addressing the complex data challenges in the modern retail and e-commerce industry.

6.2. Future Enhancements

While this project provides a solid foundation, several enhancements could be made to create an even more powerful, enterprise-grade solution.

1. **Integrate Real-Time Streaming:**
    - **How:** Replace the batch CSV ingestion with a streaming source like Apache Kafka (as referenced in your spark_streaming.py file). Use Spark Structured Streaming to read from a Kafka topic (e.g., "new_orders") and append data to the Bronze Delta table in near real-time.
    - **Impact:** The Tableau dashboards would reflect sales data that is minutes old, not months, enabling real-time operational monitoring.
2. **Implement Predictive Machine Learning Models:**
    - **How:** Use the clean Silver table to train more advanced models from Spark MLlib.

- **Models:**
  - **Demand Forecasting:** Use a Time Series model (like ARIMA) on the Gold tables to predict future sales for top products.
  - **Customer Segmentation:** Use K-Means clustering on customer-level data (e.g., total spend, order frequency) to identify distinct segments like "High-Value," "At-Risk," and "New."
  - **Customer Churn Prediction:** Build a classification model (e.g., Logistic Regression, Random Forest) to predict which customers are likely to stop purchasing.

3. **NLP for Product & Sentiment Analysis:**
   - **How:** Use NLP techniques on the Description field to automatically categorize products. If customer review data were available, sentiment analysis models could be used to extract product feedback at scale.
4. **Pipeline Automation and Orchestration:**
   - **How:** Use Databricks Workflow**s** or an external orchestrator like Apache Airflow to schedule the Bronze, Silver, and Gold scripts as a daily or hourly automated job.
   - **Impact:** This removes all manual intervention, creating a fully automated, production-ready data pipeline.

REFERENCES

[1] Databricks Documentation. (2025). *The Databricks Lakehouse Platform*. Available at: https://docs.databricks.com

[2] Apache Spark Documentation. (2025). *Spark MLlib - Frequent Pattern Mining*. Available at: https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html

[3] Tableau Software Documentation. (2025). *Connect to Databricks*. Available at: https://help.tableau.com

[4] Lichman, M. (2013). *UCI Machine Learning Repository [Online Retail Dataset]*. Irvine, CA: University of California, School of Information and Computer Science. Available at: [Insert Kaggle/UCI Link]

[5] White, T. (2015). *Hadoop: The Definitive Guide*. O'Reilly Media.

[6] [Author, Y.] (Year). *Big Data in Retail: Opportunities and Challenges*. [Journal Name].

[7] [Author, Z.] (Year). *A Scalable Implementation of the FP-Growth Algorithm*. [Conference Proceedings].