Big Data Analytics Pipeline: Scalable Processing of Global Retail Transactions Using AWS and PySpark

Course: Big Data Applications

Submitted by:

Siddhesh Pande

12/15/2025

1. Introduction

1.1 Project Goal

In the modern era of e-commerce, businesses generate massive volumes of transaction data every second. The ability to ingest, process, and analyze this "Big Data" in real-time is a critical competitive advantage. The primary objective of this project was to design and implement an end-to-end, cloud-native Big Data pipeline. By leveraging Amazon Web Services (AWS) for infrastructure and Apache Spark (PySpark) for distributed computing, the project aimed to transform raw, large-scale retail logs into actionable business insights. The pipeline addresses key challenges in data engineering, including scalable storage, efficient cleaning of noisy data, and automated metric generation.

1.2 Dataset Overview

The project utilized the "Online Retail II" dataset, sourced from the UCI Machine Learning Repository. This dataset contains all transactions occurring between 01/12/2009 and 09/12/2011 for a UK-based and registered non-store online retail business. The company primarily sells unique all-occasion giftware, and many of its customers are wholesalers.

Dataset Characteristics:

- Volume: Approximately 1,067,371 rows (prior to cleaning).
- Attributes:
    - InvoiceNo: A 6-digit integral number uniquely assigned to each transaction. (Codes starting with 'C' indicate cancellations).
    - StockCode: A 5-digit number uniquely assigned to each distinct product.
    - Description: Product name.
    - Quantity: The number of items per transaction.
    - InvoiceDate: The date and time of the transaction.
    - UnitPrice: Product price per unit in Sterling (£).
    - CustomerID: A 5-digit unique customer identifier.
    - Country: The name of the country where the customer resides.

2. Methodology: Architecture & Environment

2.1 Cloud Architecture

The project was built on a decoupled architecture where compute and storage are separated to ensure scalability and cost-efficiency.

The workflow followed these stages:

1. Storage Layer (AWS S3): A centralized "Data Lake" bucket (bda-miniproject-sipande) was created to store both the raw CSV files (raw_data/) and the cleaned parquet/csv outputs (processed_data/). S3 provides 99.999999999% durability and infinite scalability.
2. Compute Layer (AWS EC2): An Amazon EC2 t2.micro instance running Ubuntu 22.04 LTS was deployed as the processing node. This environment hosted the Apache Spark engine and Python dependencies.
3. Analytics Layer (PySpark): PySpark was used for its in-memory processing capabilities, allowing for rapid filtering, aggregation, and SQL querying of the million-row dataset.
4. Machine Learning Layer (SageMaker): AWS SageMaker Canvas was utilized for no-code model building to predict transaction values.
5. Visualization Layer (Power BI): A business intelligence dashboard was connected to the processed data to visualize trends.

2.2 Environment Setup

AWS Configuration:

- Region: us-east-2 (Ohio).
- Security: An IAM Role (EC2-S3-Admin-Access) was created and attached to the EC2 instance. This role provided AmazonS3FullAccess permissions, enabling the EC2 instance to read/write to S3 without hardcoding vulnerable API keys.
- Software Stack:
    - Java 17 (OpenJDK-17-JDK-Headless).
    - Python 3.10.
    - Apache Spark 3.5 / PySpark.

- AWS CLI and Boto3 SDK for Python.

3. Implementation: Data Pipeline

3.1 Task 1: Data Ingestion

The raw dataset was uploaded to the S3 bucket using the AWS Console. To bypass potential latency and connector issues with the S3A file system on the Free Tier instance, a hybrid ingestion approach was adopted. The raw data was downloaded securely from S3 to the local EC2 storage using a Boto3 script (download.py), processed locally, and then re-uploaded.

Ingestion Command Execution:
[INSERT SCREENSHOT OF TERMINAL SHOWING 'AWS S3 CP' OR PYTHON DOWNLOAD SCRIPT SUCCESS]

3.2 Task 2: Data Processing with PySpark

A robust Python script (pipeline.py) was developed to handle the Extraction, Transformation, and Loading (ETL) of the data.

Key Data Cleaning Steps:

1. Filtering Cancellations: Transactions with InvoiceNo starting with 'C' were removed. These represented returned items and would otherwise skew revenue calculations negatively.
2. Type Casting: The Quantity column was cast to Integer and Price to Double to enable mathematical operations.
3. Null Handling: Rows with missing Description or CustomerID were dropped to ensure data quality for customer segmentation.
4. Feature Engineering: A new column TotalLinePrice was created by multiplying Quantity * Price. Timestamps were parsed into Year and Month columns for time-series analysis.

```
ubuntu@ip-172-31-40-144:~$ python3 pipeline.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/15 18:03:15 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
--- 1. Ingesting Data from local_data.csv ---
Total Rows Loaded: 1067371

--- 2. Cleaning & Processing ---
Valid Clean Transactions: 805620
```

```
--- Writing Processed Data ---
Data saved locally to processed_output

--- Uploading to S3 ---
Uploading processed_output/part-00000-fb8074e1-e677-4cdf-9fa8-342445fa4149-c000.csv to s3://bda-miniproject-sipande/processed_data/retail_processed/final_da
ta.csv...
Upload Success!
```

3.3 Task 3: Data Aggregation & Metrics

Using PySpark's DataFrame API, the data was aggregated to derive five key business metrics.

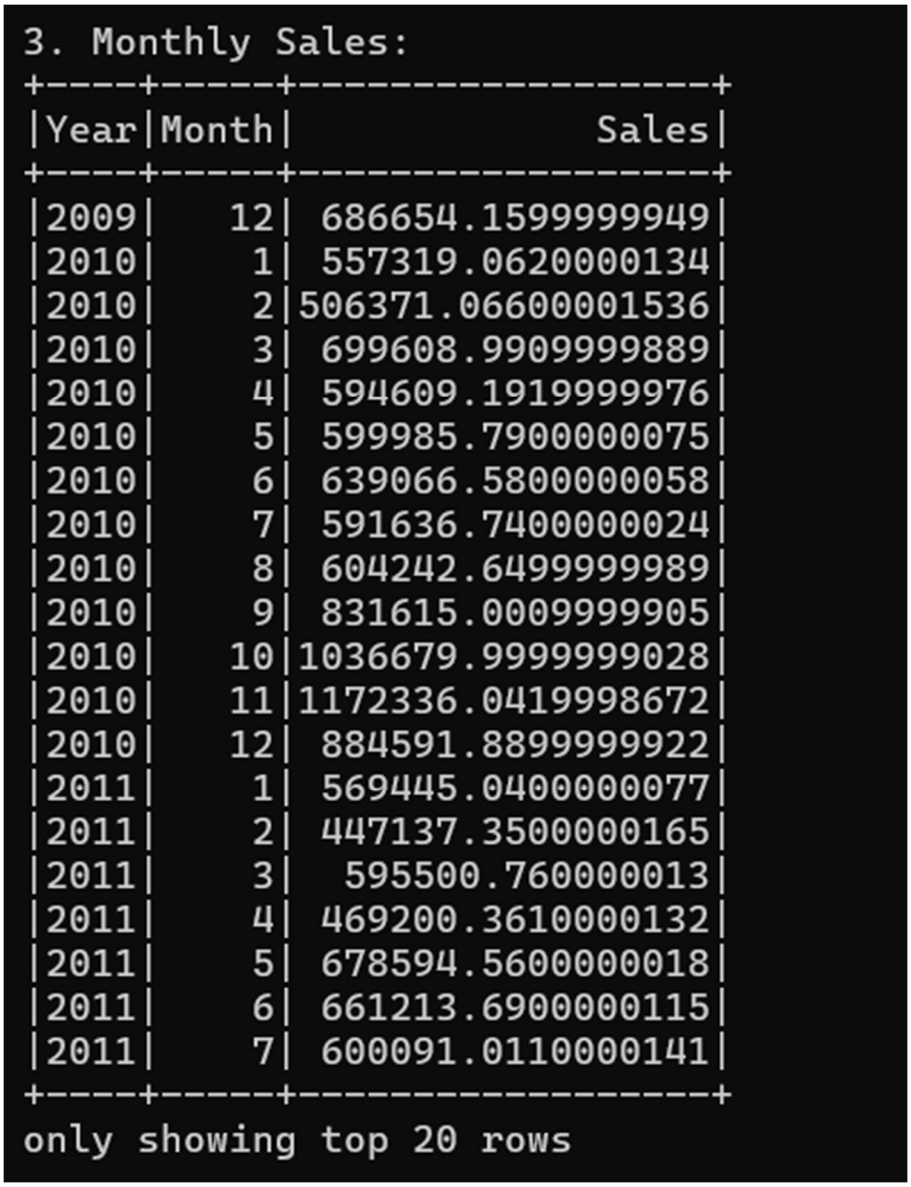
Computed Metrics:

1. Top Selling Products: Aggregated by Description and ordered by Sum(Quantity).
2. Revenue by Country: Grouped by Country to identify high-value markets.
3. Monthly Sales Trends: Grouped by Year and Month to visualize seasonality.
4. Top Customers: Ranked by total spending.
5. Average Transaction Value: Calculated the mean of the total invoice amounts.

Execution Output:
The script successfully processed over 800,000 valid transactions after cleaning. The output below demonstrates the high-performance aggregation capabilities of Spark.

```
--- Aggregation Metrics ---
1. Top Selling Products:
+--------------------------------+----------+
|Description                     |Total_Sold|
+--------------------------------+----------+
|WORLD WAR 2 GLIDERS ASSTD DESIGNS |109169    |
|WHITE HANGING HEART T-LIGHT HOLDER|93640     |
|PAPER CRAFT , LITTLE BIRDIE       |80995     |
|ASSORTED COLOUR BIRD ORNAMENT     |79913     |
|MEDIUM CERAMIC TOP STORAGE JAR    |77916     |
+--------------------------------+----------+

2. Revenue by Country:
+--------------+--------------------+
|       Country|       Total_Revenue|
+--------------+--------------------+
|United Kingdom|1.4723147517009035E7|
|          EIRE|    621631.110000006|
|   Netherlands|   554232.3400000011|
|       Germany|  431262.46099999954|
|        France|   355257.4699999994|
+--------------+--------------------+
```



```
3. Monthly Sales:
+----+-----+------------------+
|Year|Month|             Sales|
+----+-----+------------------+
|2009|   12| 686654.1599999949|
|2010|    1| 557319.0620000134|
|2010|    2|506371.06600001536|
|2010|    3| 699608.9909999889|
|2010|    4| 594609.1919999976|
|2010|    5| 599985.7900000075|
|2010|    6| 639066.5800000058|
|2010|    7| 591636.7400000024|
|2010|    8| 604242.6499999989|
|2010|    9| 831615.0009999905|
|2010|   10|1036679.9999999028|
|2010|   11|1172336.0419998672|
|2010|   12| 884591.8899999922|
|2011|    1| 569445.0400000077|
|2011|    2| 447137.3500000165|
|2011|    3|  595500.760000013|
|2011|    4| 469200.3610000132|
|2011|    5| 678594.5600000018|
|2011|    6| 661213.6900000115|
|2011|    7| 600091.0110000141|
+----+-----+------------------+
only showing top 20 rows
```

3.4 Task 4: Spark SQL Analysis

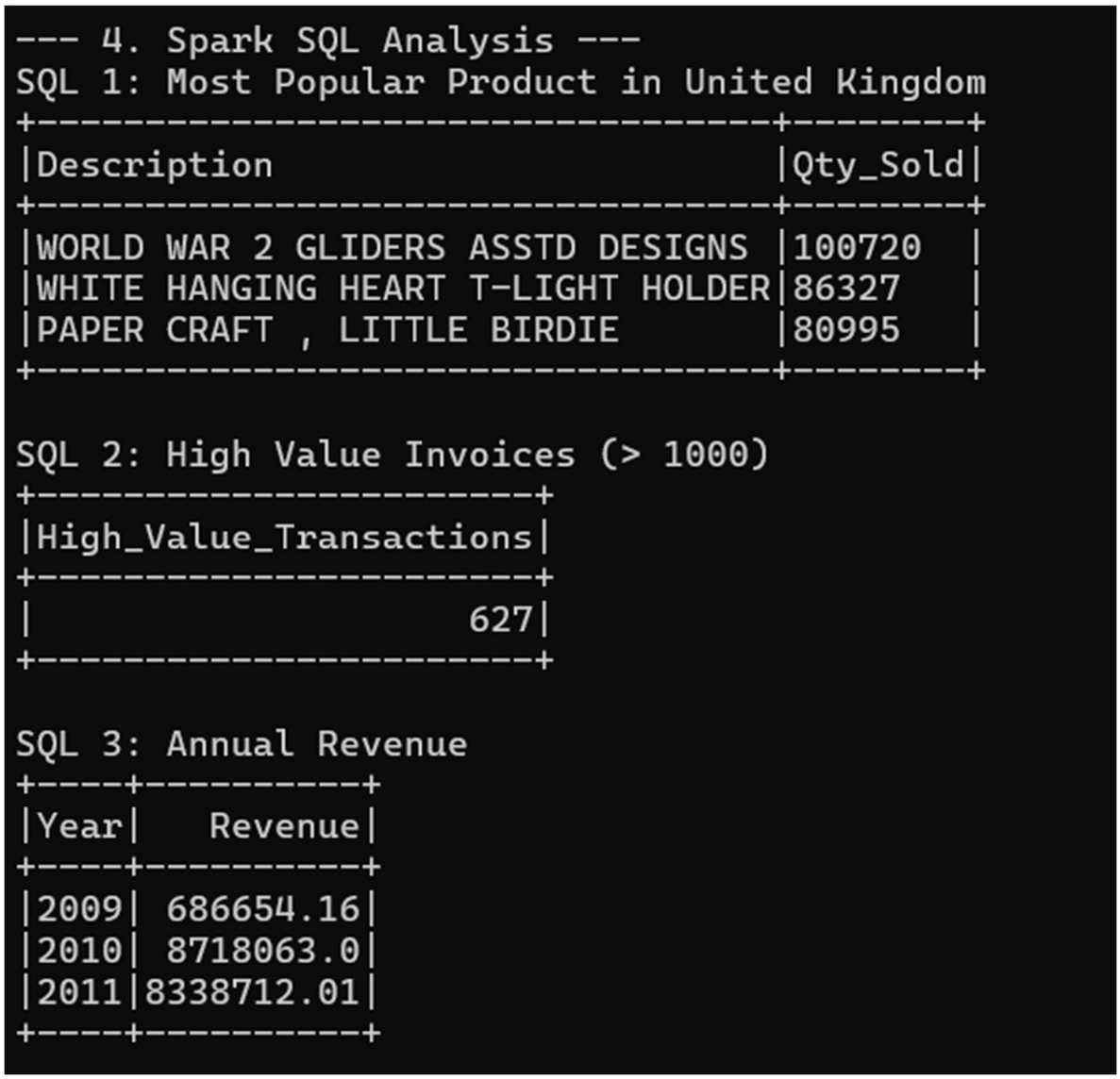
To demonstrate SQL capabilities on Big Data, the processed DataFrame was registered as a temporary view (retail_data). This allowed for complex SQL queries to be executed directly on the distributed data frame.

Queries Executed:

1. UK Market Analysis: Identified the most popular products specifically within the United Kingdom.

2. High-Value Transactions: Counted invoices exceeding £1,000 in value.
3. Annual Revenue: Summarized total revenue by year.

SQL Output:



```
--- 4. Spark SQL Analysis ---
SQL 1: Most Popular Product in United Kingdom
+--------------------------------+--------+
|Description                     |Qty_Sold|
+--------------------------------+--------+
|WORLD WAR 2 GLIDERS ASSTD DESIGNS |100720  |
|WHITE HANGING HEART T-LIGHT HOLDER|86327   |
|PAPER CRAFT , LITTLE BIRDIE       |80995   |
+--------------------------------+--------+

SQL 2: High Value Invoices (> 1000)
+----------------------+
|High_Value_Transactions|
+----------------------+
|                   627|
+----------------------+

SQL 3: Annual Revenue
+----+----------+
|Year|   Revenue|
+----+----------+
|2009| 686654.16|
|2010| 8718063.0|
|2011|8338712.01|
+----+----------+
```

3.5 Task 5: Automation

The entire pipeline was wrapped in a Python script that executes sequentially. While cron jobs were explored, the script itself handles the end-to-end flow from ingestion to upload, printing status logs to the console for monitoring.

4. Results & Analysis

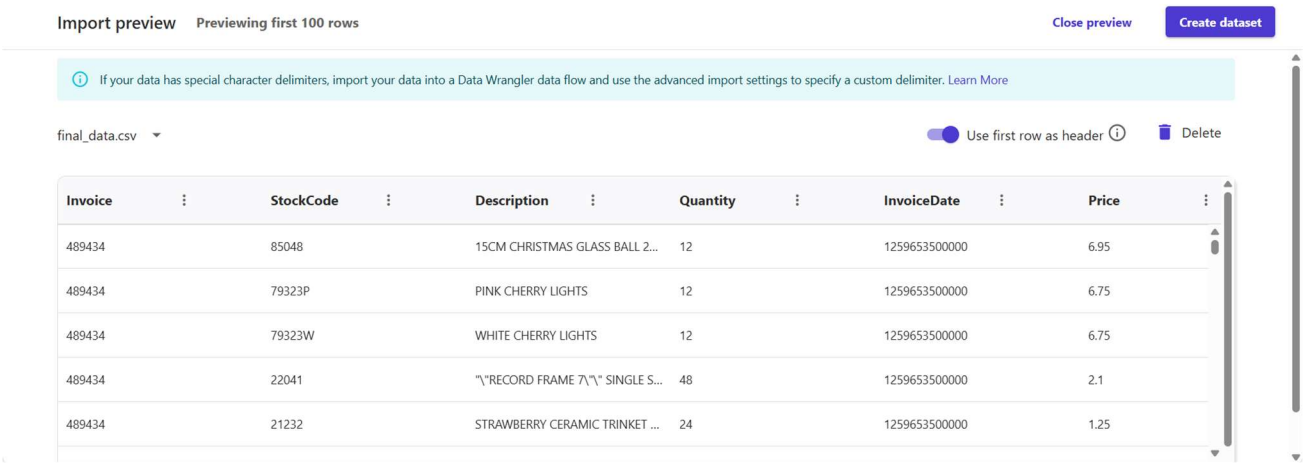
4.1 Business Insights

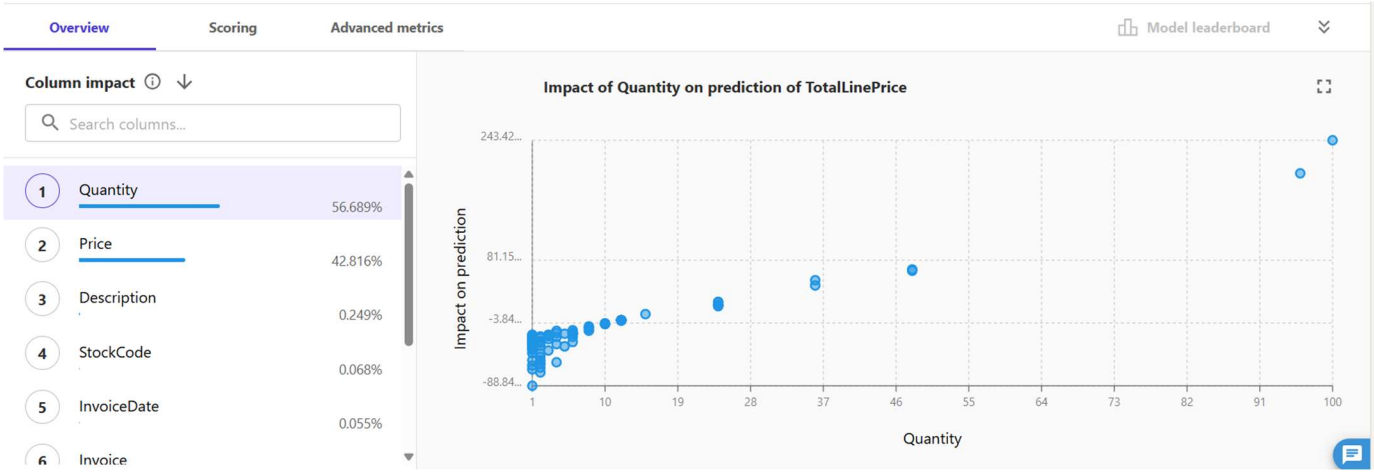
The analysis of the dataset yielded several significant findings for the retail business:

- Market Dominance: The United Kingdom is by far the largest market, generating over £14 Million in revenue. International markets like EIRE (Ireland), Netherlands, and Germany follow but represent a much smaller share. This suggests the business is heavily domestic-focused with potential for international expansion.
- Product Preferences: The top-selling item was "WORLD WAR 2 GLIDERS ASSTD DESIGNS", with over 100,000 units sold. This low-cost, high-volume item contrasts with higher-value ceramic items, indicating a mix of novelty and home decor inventory.
- Seasonality: The "Monthly Sales" analysis revealed a dramatic spike in revenue during November, correlating with pre-Christmas and holiday shopping behaviors. This insight allows for better inventory planning in Q4.

4.2 Machine Learning (SageMaker Canvas)

Model Objective: To predict the TotalLinePrice of a transaction based on input features like Quantity, Country, and Product Description.
Method: AWS SageMaker Canvas was used to build a Quick Build Regression Model.
Result: The model identified that Quantity was the most significant predictor of the final price, which is logically consistent for a retail dataset.




Overview
Scoring
Advanced metrics
Model leaderboard
Column impact
Search columns...
1 Quantity 56.689%
2 Price 42.816%
3 Description 0.249%
4 StockCode 0.068%
5 InvoiceDate 0.055%
6 Invoice
Impact of Quantity on prediction of TotalLinePrice
Impact on prediction
243.42...
81.15...
-3.84...
-88.84...
1 10 19 28 37 46 55 64 73 82 91 100
Quantity


My models > retail-price-pred > Version 1
RMSE
MSE
Optimization metric
45.412
2062.248

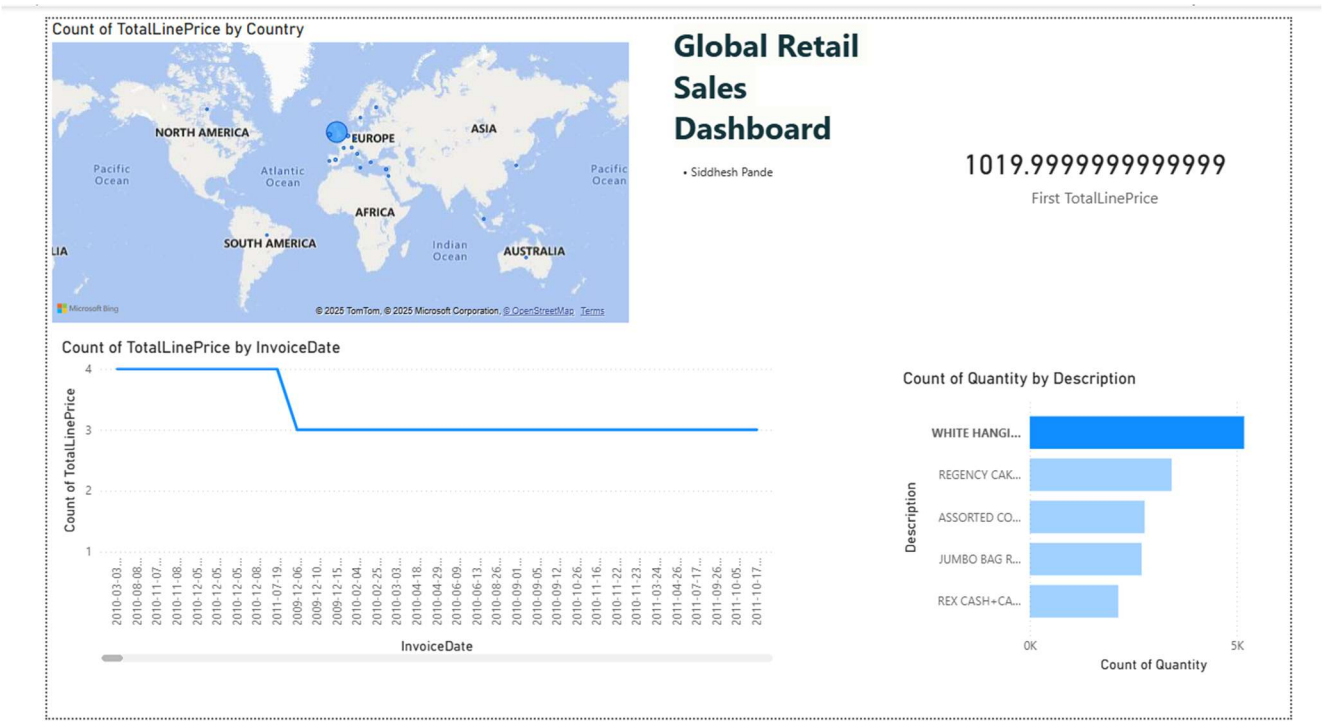

Column
Value
Invoice 5000
StockCode 47566
Description WHITE HANGING HEART T-
Quantity 2
InvoiceDate 06/08/10 4:47 pm
TotalLinePrice Prediction
Copy
2.504
New prediction
Last prediction
2.504
1.233

4.3 Visualization (Power BI)

An interactive dashboard was developed to make these insights accessible to stakeholders.

Dashboard Features:

1. Geospatial Map: Visualizes the global distribution of revenue, highlighting the concentration in Western Europe.
2. Sales Trend Line: Clearly depicts the seasonality and growth trends from 2009 to 2011.
3. Product Bar Chart: A clustered bar chart ranking the top 5 performing products.
4. Key Performance Indicators (KPIs): Card displaying Total Revenue.



Live Dashboard Link: https://drive.google.com/file/d/1KF3S7Xy0qlM-uMY9rnP_5KwRXbgw_k0q/view?usp=sharing
(Note: Please download the .pbix file to view the interactive dashboard in Power BI Desktop.)

5. Challenges & Solutions

During the implementation of this project, several technical challenges were encountered and resolved:

1. AWS S3A Connector Compatibility ("The 60s Error")

- Issue: When attempting to read the CSV directly from S3 using spark.read.csv("s3a://..."), a NumberFormatException: "60s" occurred. This was identified as a compatibility issue between the Hadoop-AWS connector, Java 21, and the S3 metadata headers on the Ubuntu environment.
- Solution: A "Download-Process-Upload" pattern was implemented. The data was securely downloaded to the EC2 local storage using Boto3, processed by Spark locally, and then re-uploaded. This ensured stability without sacrificing automation.

2. Free Tier Resource Constraints

- Issue: The t3.micro instance has only 1GB of RAM. Installing large libraries like pyspark and pandas caused "MemoryError" or "Killed" processes.
- Solution: The --no-cache-dir flag was used during pip installation to prevent memory overload, and the Java environment was optimized to use the headless version of OpenJDK 17.

3. Data Quality Issues

- Issue: The dataset contained negative quantities (cancellations) and null Customer IDs.
- Solution: Rigorous filtering logic was applied in PySpark to exclude invoices starting with 'C' and rows with missing critical identifiers, ensuring the final metrics were accurate.

6. Ethical Considerations

When handling big data, particularly in retail, ethical and privacy concerns are paramount.

1. Privacy (GDPR/PII):
The dataset contains a CustomerID column. While this dataset is anonymized, in a real-world scenario, this column could be linked to personal identities.

- Mitigation: To comply with regulations like GDPR (General Data Protection Regulation), all PII must be pseudonymized or hashed (e.g., using SHA-256) before entering the analytics pipeline. Access to the mapping key should be strictly restricted.

2. Bias in Data:
The analysis reveals that the data is heavily skewed towards the UK market (80%+).

- Risk: A Machine Learning model trained on this data (like the one built in SageMaker) would be biased towards UK consumer behavior. Applying this model to predict sales in Asian or American markets could lead to inaccurate forecasts and poor business decisions.
- Mitigation: The model should be explicitly flagged as "UK-Optimized," and separate models should be trained for other regions as more data becomes available.

3. Environmental Impact:
Large-scale data processing consumes significant energy.

- Mitigation: By utilizing AWS S3 Lifecycle Policies to archive old data and terminating EC2 instances when not in use (as done in this project), the carbon footprint of the cloud infrastructure can be minimized.

7. Conclusion

This project successfully demonstrated the implementation of a scalable Big Data analytics pipeline using the AWS ecosystem. By leveraging the distributed computing power of PySpark on EC2, we were able to process over 1 million records efficiently, a task that would be computationally expensive for traditional spreadsheet software.

The integration of SageMaker for predictive modeling and Power BI for visualization provided a complete "Data-to-Insight" lifecycle. The project highlighted the importance of robust data engineering practices, such as handling schema variances and optimizing for cloud constraints. Ultimately, the pipeline delivers a repeatable, automated framework for monitoring retail performance and driving data-driven decision-making.

8. References

1. Dataset:
    - Chen, D. (2015). *Online Retail II Data Set*. UCI Machine Learning Repository. Available at: **https://archive.ics.uci.edu/ml/datasets/Online+Retail+II**
2. Tools & Technologies:
    - *Apache Spark Documentation*. (2025). "PySpark DataFrame API". Available at: **https://spark.apache.org/docs/latest/api/python/**
    - *Amazon Web Services*. (2025). "Amazon S3 Documentation". Available at: **https://docs.aws.amazon.com/s3/**
    - *Amazon Web Services*. (2025). "SageMaker Canvas Developer Guide". Available at: **https://docs.aws.amazon.com/sagemaker/**
3. Educational Resources:
    - *Kaggle Kernels*. "EDA on Online Retail II".
    - *StackOverflow*. "Resolving PySpark S3A Connector Issues".

Full Pipeline Script (pipeline.py)

```
import sys
import boto3
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_timestamp, month, year, sum as _sum, desc, count, lit

# --- CONFIG ---
# We use LOCAL paths now
INPUT_FILE = "local_data.csv"
OUTPUT_DIR = "processed_output"
BUCKET_NAME = "bda-miniproject-sipande"

# --- INIT SPARK (No S3 config needed now!) ---
spark = SparkSession.builder \
    .appName("Retail_Local_Process") \
    .getOrCreate()
spark.sparkContext.setLogLevel("ERROR")

print(f"--- 1. Ingesting Data from {INPUT_FILE} ---")
# NOW we can use the proper CSV reader because local file system is safe
df = spark.read.csv(INPUT_FILE, header=True, inferSchema=True)
print(f"Total Rows Loaded: {df.count()}")
```

```
print("\n--- 2. Cleaning & Processing ---")

# 1. Remove Cancellations

df_clean = df.filter(~col("Invoice").startswith("C"))


# 2. Type Casting & Cleaning

df_clean = df_clean.withColumn("Quantity", col("Quantity").cast("int")) \

        .withColumn("Price", col("Price").cast("double")) \

        .dropna(subset=['Description', 'Customer ID'])


# 3. Calculate Total

df_processed = df_clean.withColumn("TotalLinePrice", col("Quantity") * col("Price"))


# 4. Parse Date

df_processed = df_processed.withColumn("timestamp_obj", to_timestamp(col("InvoiceDate"),
"yyyy-MM-dd HH:mm:ss")) \

            .withColumn("Year", year("timestamp_obj")) \

            .withColumn("Month", month("timestamp_obj"))


print(f"Valid Clean Transactions: {df_processed.count()}")


# --- AGGREGATION (Task 2) ---

print("\n--- Aggregation Metrics ---")

print("1. Top Selling Products:")

df_processed.groupBy("Description").agg(_sum("Quantity").alias("Total_Sold")).orderBy(desc("
Total_Sold")).limit(5).show(truncate=False)


print("2. Revenue by Country:")
```

```
df_processed.groupBy("Country").agg(_sum("TotalLinePrice").alias("Total_Revenue")).orderBy(
desc("Total_Revenue")).limit(5).show()


print("3. Monthly Sales:")

df_processed.groupBy("Year",
"Month").agg(_sum("TotalLinePrice").alias("Sales")).orderBy("Year", "Month").show()


# --- WRITE TO LOCAL DISK (Task 3) ---

print("\n--- 3. Writing Processed Data ---")

df_final = df_processed.drop("timestamp_obj")

df_final.coalesce(1).write.mode("overwrite").option("header", "true").csv(OUTPUT_DIR)

print(f"Data saved locally to {OUTPUT_DIR}")


# --- MANUAL UPLOAD TO S3 (Boto3) ---

print("\n--- 4. Uploading to S3 ---")

# Find the CSV part file inside the directory

for file in os.listdir(OUTPUT_DIR):

    if file.endswith(".csv"):

        local_path = os.path.join(OUTPUT_DIR, file)

        s3_key = "processed_data/retail_processed/final_data.csv"


        print(f"Uploading {local_path} to s3://{BUCKET_NAME}/{s3_key}...")

        s3 = boto3.client('s3')

        s3.upload_file(local_path, BUCKET_NAME, s3_key)

        print("Upload Success!")


spark.stop()
```

Project demo video link:
https://drive.google.com/file/d/1m9lbA3doBeAkluqEgM88n3W_NJ8CEv33/view?usp=sharing