UNIVERSITY OF PLYMOUTH

PUSL3121 Big Data Analytics

C2 – Coursework

10899487 - Amaraweera P Saubhagya

10899191 – Diwakara Samaradiwakara

10899289 - Wickramanarayana
Dilshika

10899183 - Kawindya Kariyawasam

Table of Content

Task 01

Introduction to Big Data Processing

Overview of Apache Spark

Apache Spark is an **open-source distributed computing system** designed to handle massive data processing efficiently. It was developed to overcome the limitations of traditional batch-processing systems like Hadoop MapReduce, which rely on disk-based storage and can be slow when dealing with large-scale data. Spark provides a **unified engine** that supports multiple workloads, including batch processing, real-time stream processing, machine learning, and graph analytics.

One of the major strengths of Apache Spark is its **in-memory computing capability**, which allows it to process data much faster than disk-based frameworks. Instead of writing intermediate results to disk, Spark stores data in RAM, significantly reducing read and write times. This feature enables Spark to execute iterative algorithms—commonly used in machine learning and data analytics—much more efficiently. For instance, a machine learning model that requires multiple passes over a dataset can benefit greatly from Spark's ability to cache data in memory.

Another key feature of Spark is its **distributed computing architecture**, which allows it to scale horizontally by distributing tasks across multiple nodes in a cluster. This means that as the data volume increases, Spark can allocate resources dynamically to handle large-scale computations efficiently. By breaking down tasks into smaller partitions and processing them in parallel, Spark achieves high fault tolerance and scalability, making it suitable for enterprises dealing with big data.

Spark supports multiple programming languages, including **Java, Python, Scala, and R**, making it accessible to a wide range of developers and data scientists. This flexibility allows users to write applications using their preferred language while leveraging Spark's high-performance engine. Additionally, Spark's ecosystem includes several powerful libraries such as **Spark SQL** for querying structured data, **MLlib** for machine learning, **GraphX** for graph-based analytics, and **Spark Streaming** for processing real-time data streams. These built-in libraries make Spark a comprehensive solution for big data processing.

Due to its speed, scalability, and ease of use, Apache Spark has become a popular choice for organizations handling large datasets, real-time analytics, and AI-driven applications. Companies like Amazon, Netflix, and Uber use Spark for tasks ranging from recommendation systems and fraud detection to real-time traffic analytics. With its ability to process vast amounts of data efficiently, Spark continues to play a crucial role in the field of **big data analytics and artificial intelligence**.

Advantages of Spark for Large-Scale Data Processing

Apache Spark offers several advantages that make it a preferred choice for large-scale data processing. One of its most significant benefits is **high-speed execution**, achieved through **in-memory computation**. Unlike traditional big data frameworks like Hadoop MapReduce, which rely on slow disk-based processing, Spark processes data in memory whenever possible. This reduces the overhead of writing intermediate results to disk, significantly improving performance. Spark can execute tasks **multiple times faster** than conventional data processing frameworks, making it ideal for handling **big data analytics, machine learning, and real-time data processing**.

Another major advantage of Spark is its **scalability**. Spark clusters can process massive datasets at **petabyte scale**, distributing computations across thousands of nodes. This **horizontal scalability** ensures that as data volumes increase, Spark can efficiently allocate resources to handle workloads without compromising performance. Users can **dynamically scale** their Spark clusters by adding or removing processing nodes, making it a flexible solution for businesses dealing with **growing data needs**.

Spark’s versatility is another reason for its popularity. It **supports both structured and unstructured data**, enabling users to work with diverse data formats. Whether handling **batch processing for historical data analysis** or **real-time stream processing** for instant insights, Spark is capable of efficiently processing different types of workloads. This flexibility makes it a great fit for industries such as **finance, healthcare, and e-commerce**, where both historical and real-time data play a crucial role.

Additionally, **fault tolerance** is a built-in feature of Spark, ensuring data reliability and consistency even in distributed environments. Spark achieves this through **Resilient Distributed Datasets (RDDs)**, which automatically **recompute lost data** in case of node failures. This makes Spark highly reliable for mission-critical applications where data loss or system downtime could have serious consequences.

Overview of Snowflake

(Snowflake, 2025) Snowflake is a **cloud-based data warehouse platform** designed to handle **data storage, processing, and analytics** in a scalable and efficient manner. Unlike traditional on-premises data warehouses, Snowflake **separates compute resources from storage**, allowing independent scaling based on workload intensity. This unique architecture enables users to optimize costs by scaling compute power **only when needed**, making it ideal for businesses with fluctuating workloads.

Snowflake supports various data processing use cases, including **Online Analytical Processing (OLAP), Extract, Transform, Load (ETL), and data sharing**. Its ability to execute **complex queries on large datasets with minimal configuration** makes it a popular choice for enterprises looking to streamline their data operations.

One of Snowflake’s key strengths is its **low maintenance and automated management system**. It **requires minimal administrative overhead**, handling tasks such as **data optimization, performance tuning, and security management** automatically. Additionally, Snowflake supports multiple **structured and semi-structured data formats**, including **JSON, Avro, and Parquet**, making it highly adaptable to modern data storage requirements.

With its **scalability, cost efficiency, and ease of use**, Snowflake is widely adopted in industries like **finance, healthcare, retail, and technology** for advanced data analytics and business intelligence applications.

Role of Snowflake in Cloud-Based Data Warehousing and Analytics

Snowflake plays a crucial role in **modern cloud-based data warehousing and analytics** by offering a **highly flexible, scalable, and cost-efficient** platform. One of its most significant advantages is its **storage and compute separation mechanism**, which allows users to scale compute resources **independently** of storage. This means businesses can **increase or decrease computing power based on workload demand** without affecting stored data, leading to **optimized performance and cost savings**. Unlike traditional data warehouses, where users must pay for **fixed, pre-allocated resources**, Snowflake's **pay-as-you-go pricing model** ensures that organizations **only pay for the resources they actually use**, making it a cost-effective solution for data processing (advsyscon, 2024).

Another key feature that makes Snowflake stand out is **Time Travel**, which enables users to **retrieve historical versions of data** from various points in time. This feature is particularly useful for **auditing, data recovery, compliance, and debugging purposes**. By allowing users to track and restore previous data states, **Time Travel ensures data integrity and security**, providing companies with greater operational control over their datasets.

Snowflake also enhances **collaborative data sharing** by eliminating the need for **duplicate copies of data across different teams, departments, or external stakeholders**. Unlike traditional methods that require **copying and transferring datasets**, Snowflake's **secure data-sharing capabilities** allow multiple users to access the same dataset **in real-time** without compromising security or performance. This ensures that all teams work with **a single source of truth**, improving data consistency and decision-making efficiency.

Overall, Snowflake's **cloud-native architecture, scalability, advanced security features, and seamless data-sharing capabilities** make it an **ideal solution for enterprises looking to enhance their data warehousing and analytics processes**. Its ability to handle **structured, semi-structured, and unstructured data** further strengthens its role in supporting **business intelligence, machine learning, and big data applications** in industries such as **finance, healthcare, retail, and technology**.

Task 02

Data Preprocessing with Apache Spark in Google Colab

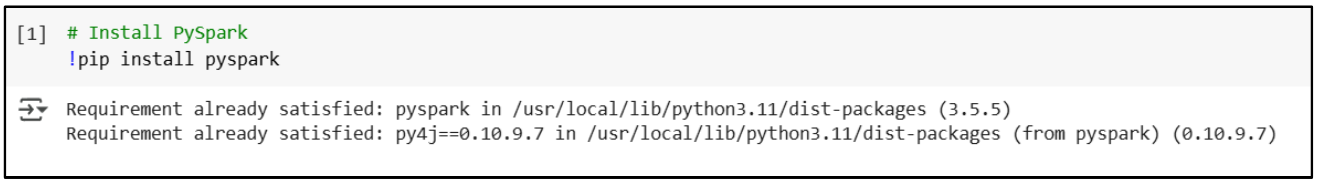
Objective

In this task, our team performed data preprocessing steps such as data loading, cleansing, transformation, and feature engineering using Apache Spark in Google Colab. We also compared Spark's performance with traditional data processing tools like Python's Pandas.

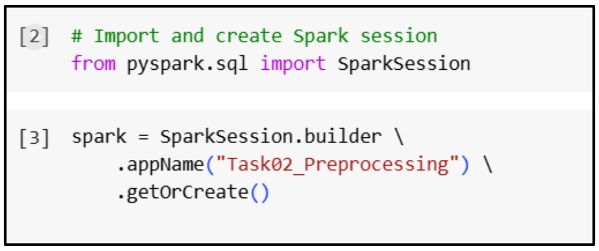
Steps Taken in Google Colab

1. Data Loading

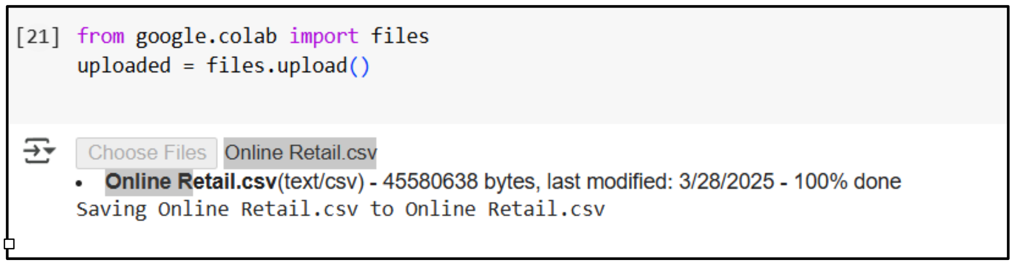
We first uploaded the Online Retail dataset (Online.csv) into Google Colab and loaded it into Apache Spark. Using PySpark (the Python API for Spark), we initialized a SparkSession and used it to load the CSV data as a DataFrame. This method allows us to scale processing efficiently on large datasets.



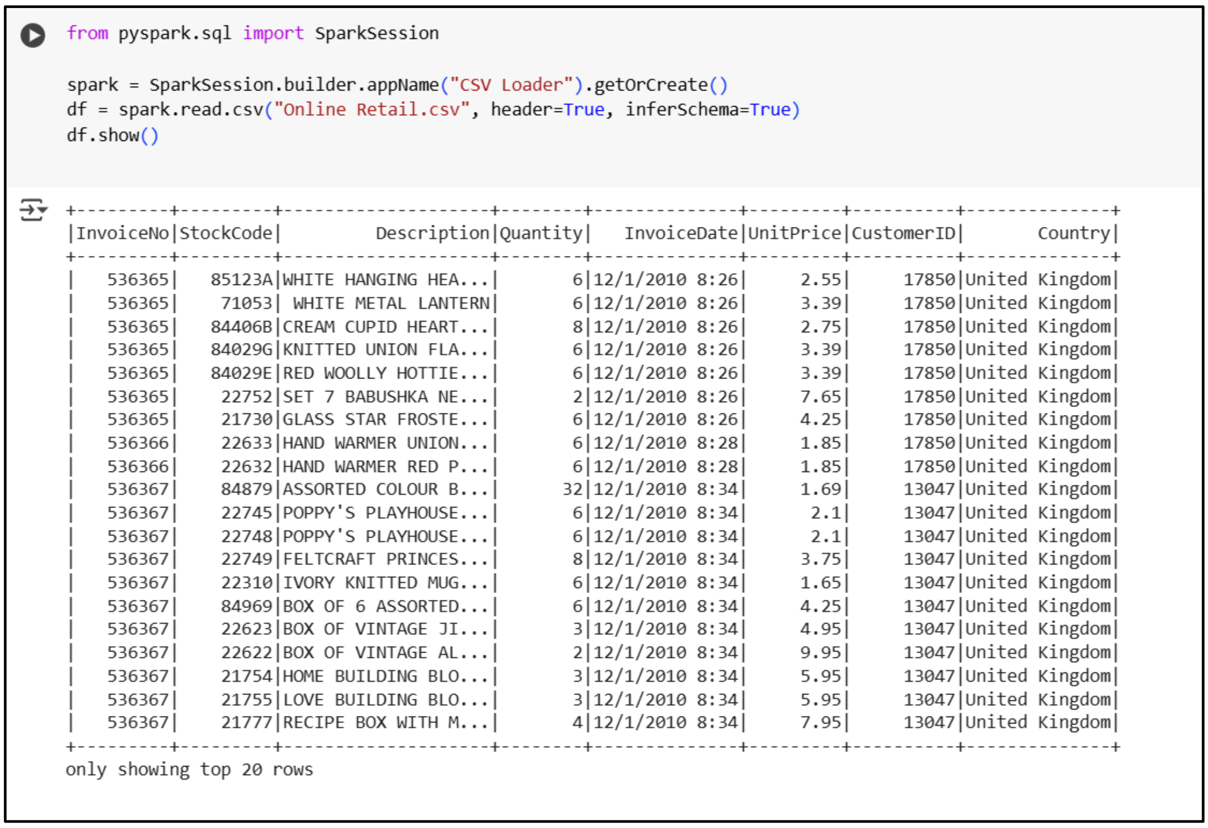
```
[1]  # Install PySpark
     !pip install pyspark

     Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.5)
     Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
```



```
[2]  # Import and create Spark session
     from pyspark.sql import SparkSession

[3]  spark = SparkSession.builder \
         .appName("Task02_Preprocessing") \
         .getOrCreate()
```

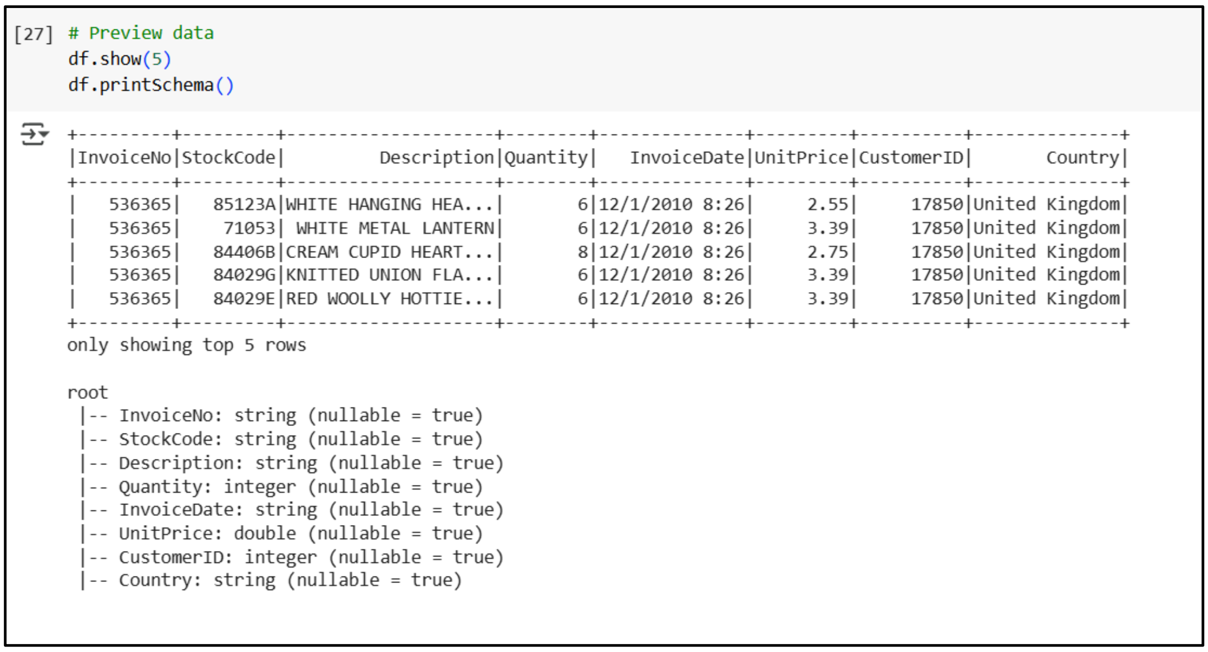


```
[21] from google.colab import files
     uploaded = files.upload()

     Choose Files  Online Retail.csv
     • Online Retail.csv(text/csv) - 45580638 bytes, last modified: 3/28/2025 - 100% done
     Saving Online Retail.csv to Online Retail.csv
```



```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("CSV Loader").getOrCreate()
df = spark.read.csv("Online Retail.csv", header=True, inferSchema=True)
df.show()
```

```
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
|InvoiceNo|StockCode|         Description|Quantity|   InvoiceDate|UnitPrice|CustomerID|       Country|
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/2010 8:26|     2.55|     17850|United Kingdom|
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/2010 8:26|     2.75|     17850|United Kingdom|
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|    22752|SET 7 BABUSHKA NE...|       2|12/1/2010 8:26|     7.65|     17850|United Kingdom|
|   536365|    21730|GLASS STAR FROSTE...|       6|12/1/2010 8:26|     4.25|     17850|United Kingdom|
|   536366|    22633|HAND WARMER UNION...|       6|12/1/2010 8:28|     1.85|     17850|United Kingdom|
|   536366|    22632|HAND WARMER RED P...|       6|12/1/2010 8:28|     1.85|     17850|United Kingdom|
|   536367|    84879|ASSORTED COLOUR B...|      32|12/1/2010 8:34|     1.69|     13047|United Kingdom|
|   536367|    22745|POPPY'S PLAYHOUSE...|       6|12/1/2010 8:34|      2.1|     13047|United Kingdom|
|   536367|    22748|POPPY'S PLAYHOUSE...|       6|12/1/2010 8:34|      2.1|     13047|United Kingdom|
|   536367|    22749|FELTCRAFT PRINCES...|       8|12/1/2010 8:34|     3.75|     13047|United Kingdom|
|   536367|    22310|IVORY KNITTED MUG...|       6|12/1/2010 8:34|     1.65|     13047|United Kingdom|
|   536367|    84969|BOX OF 6 ASSORTED...|       6|12/1/2010 8:34|     4.25|     13047|United Kingdom|
|   536367|    22623|BOX OF VINTAGE JI...|       3|12/1/2010 8:34|     4.95|     13047|United Kingdom|
|   536367|    22622|BOX OF VINTAGE AL...|       2|12/1/2010 8:34|     9.95|     13047|United Kingdom|
|   536367|    21754|HOME BUILDING BLO...|       3|12/1/2010 8:34|     5.95|     13047|United Kingdom|
|   536367|    21755|LOVE BUILDING BLO...|       3|12/1/2010 8:34|     5.95|     13047|United Kingdom|
|   536367|    21777|RECIPE BOX WITH M...|       4|12/1/2010 8:34|     7.95|     13047|United Kingdom|
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
only showing top 20 rows
```



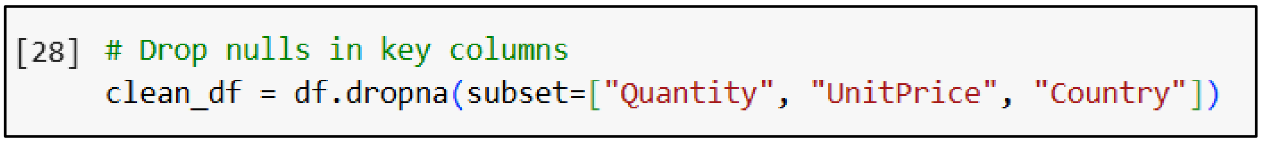
```
[27] # Preview data
     df.show(5)
     df.printSchema()
```

```
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
|InvoiceNo|StockCode|         Description|Quantity|   InvoiceDate|UnitPrice|CustomerID|       Country|
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/2010 8:26|     2.55|     17850|United Kingdom|
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/2010 8:26|     2.75|     17850|United Kingdom|
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
only showing top 5 rows

root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: integer (nullable = true)
 |-- Country: string (nullable = true)
```

2. Data Cleansing

After loading the data, we performed data cleansing by handling missing or corrupted data. In Google Colab, Spark makes it easy to drop rows with missing values or fill them with default values.

To remove rows containing null values in crucial columns such as Quantity, UnitPrice, and Country, we used:

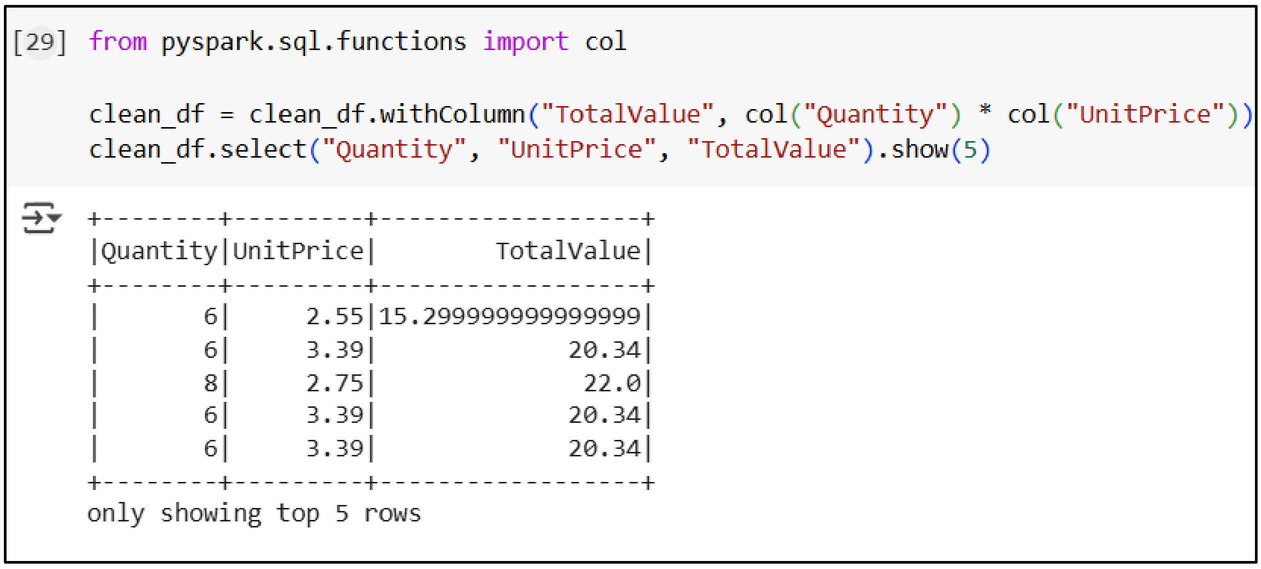


```
[28] # Drop nulls in key columns
     clean_df = df.dropna(subset=["Quantity", "UnitPrice", "Country"])
```

This step ensured that the dataset was ready for analysis and that no missing data could impact our downstream processes.

3. Data Transformation

Next, we performed data transformation to create new features. One key transformation was to calculate the TotalValue of each transaction by multiplying the Quantity by UnitPrice. This was done using Spark's DataFrame API:

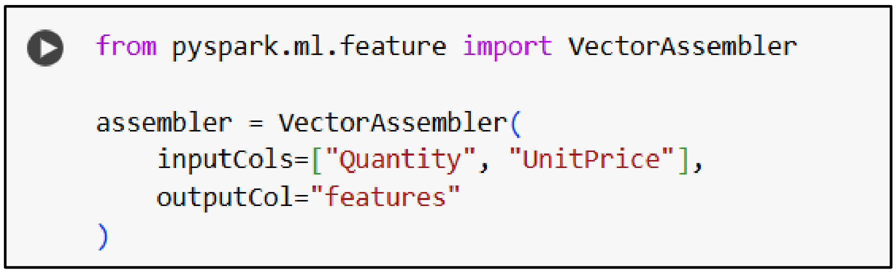


```
[29] from pyspark.sql.functions import col

     clean_df = clean_df.withColumn("TotalValue", col("Quantity") * col("UnitPrice"))
     clean_df.select("Quantity", "UnitPrice", "TotalValue").show(5)
```

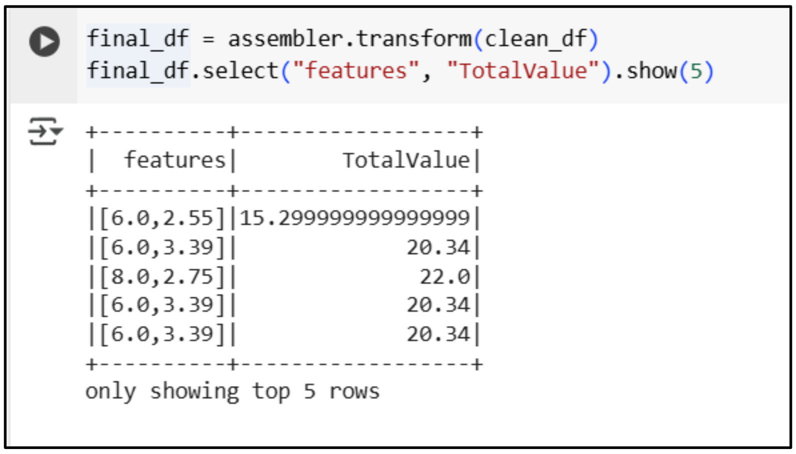
```
+--------+---------+------------------+
|Quantity|UnitPrice|        TotalValue|
+--------+---------+------------------+
|       6|     2.55|15.299999999999999|
|       6|     3.39|             20.34|
|       8|     2.75|              22.0|
|       6|     3.39|             20.34|
|       6|     3.39|             20.34|
+--------+---------+------------------+
only showing top 5 rows
```

4. Feature Engineering

In the feature engineering step, we prepared the data for machine learning by creating a feature vector. We used VectorAssembler, a built-in Spark feature, to combine multiple columns into one feature vector.



```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(
    inputCols=["Quantity", "UnitPrice"],
    outputCol="features"
)
```



```
final_df = assembler.transform(clean_df)
final_df.select("features", "TotalValue").show(5)
```

```
+----------+------------------+
|  features|        TotalValue|
+----------+------------------+
|[6.0,2.55]|15.299999999999999|
|[6.0,3.39]|             20.34|
|[8.0,2.75]|              22.0|
|[6.0,3.39]|             20.34|
|[6.0,3.39]|             20.34|
+----------+------------------+
only showing top 5 rows
```

This transformed dataset could now be used for machine learning models in Apache Spark MLlib.

Comparison of Spark with Pandas

Here's a comparison of Apache Spark and Pandas when performing data preprocessing:

Feature	**Apache Spark (in Google Colab)**	**Pandas**
Data Handling	Spark handles large datasets across multiple machines in a cluster	Best suited for small-to-medium datasets (single machine)
Scalability	Spark scales to handle terabytes of data on distributed systems	Limited by the memory available on a single machine
Speed	Faster processing of large datasets due to in-memory distributed computing	Slower on larger datasets due to single-node processing
Fault Tolerance	Spark provides fault tolerance through **RDDs** and checkpointing	No fault tolerance; may crash with large data
Ease of Use	Requires Spark setup and environment (using Google Colab is an easy setup)	Simple, Pythonic API, no setup needed
Big Data	Efficient for **big data** use cases (e.g., petabytes of data)	Not ideal for **big data** due to memory limits

Task 03

Real-Time Analytics with Apache Spark

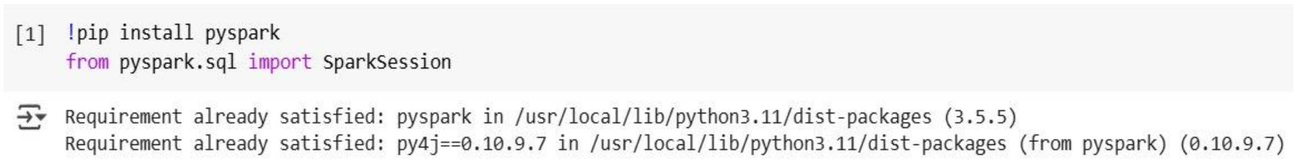
Overview

In this task, our group simulated a real-time data analytics pipeline using Apache Spark Structured Streaming. The goal was to mimic a real-world scenario where incoming data (such as customer orders) needs to be processed continuously to identify trends or compute live statistics.

We chose the Online Retail dataset from the UCI Machine Learning Repository, which contains transactional data for a UK-based online retail store. Each record includes attributes such as Invoice Number, Stock Code, Description, Quantity, Invoice Date, Unit Price, Customer ID, and Country. This dataset provided a realistic foundation for building and testing streaming data processing techniques.

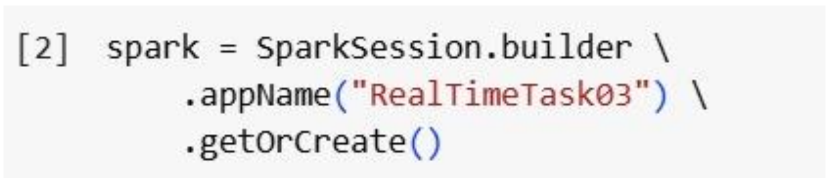
Implementation Steps

We began by using Google Colab as our development environment and PySpark for distributed data processing. The original Excel file was converted to **Online.csv**, then split into smaller chunks of 100 rows each using pandas. These chunks were stored in a folder **(/content/stream_data)** to simulate streaming data files being added one by one.



```
[1]  !pip install pyspark
     from pyspark.sql import SparkSession

     Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.5)
     Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
```

Using **PySpark**, we initialized a **SparkSession** and defined a custom schema for the dataset. We then used **readStream** with **maxFilesPerTrigger=1** to simulate a real-time stream where one CSV file is ingested per trigger interval. The streaming **DataFrame** was enhanced with a new column **TotalValue**, calculated as the product of Quantity and **UnitPrice**.

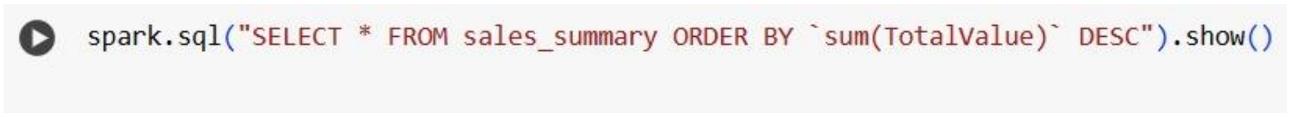


```
[2]  spark = SparkSession.builder \
         .appName("RealTimeTask03") \
         .getOrCreate()
```

Real-Time Processing and Output

To analyze trends in real-time, we grouped the streamed data by the Country column and computed the total sales **(SUM(TotalValue))** for each. Instead of printing to the console (which Colab limits), we wrote the stream to an in-memory table and queried it using Spark SQL after a short wait:

```
spark.sql("SELECT * FROM sales_summary ORDER BY `sum(TotalValue)` DESC").show()
```

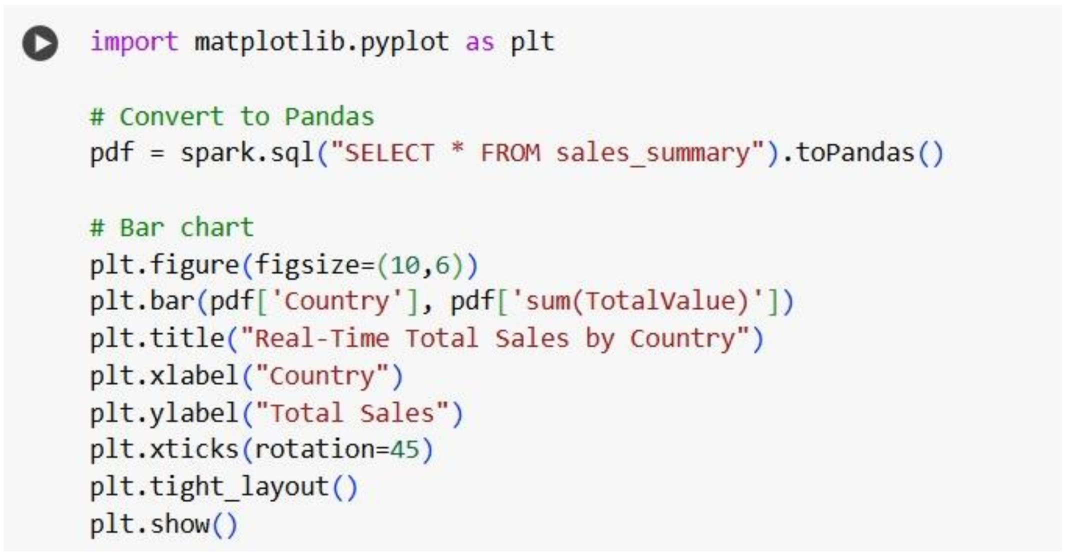


```
spark.sql("SELECT * FROM sales_summary ORDER BY `sum(TotalValue)` DESC").show()
```

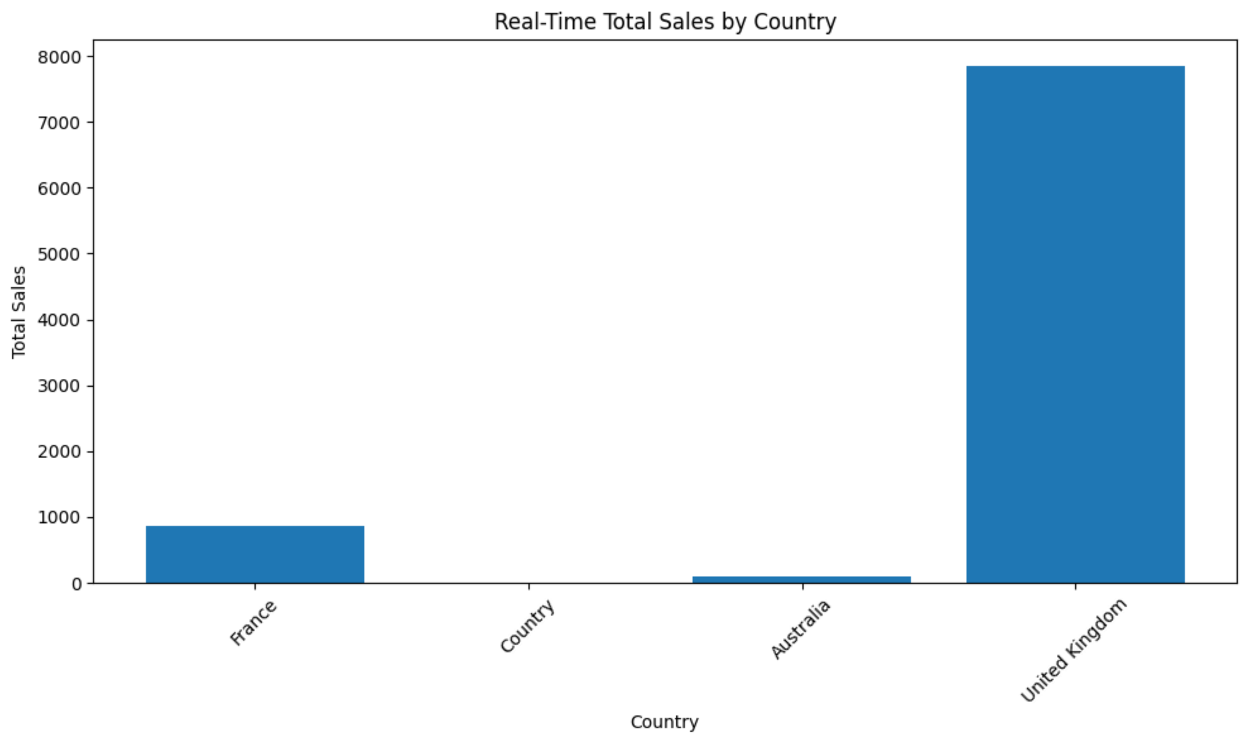
This approach allowed us to view partial results live during streaming.

Output

```
+--------------+------------------+
|       Country|   sum(TotalValue)|
+--------------+------------------+
|United Kingdom|2267.0699999999993|
|        France|            855.86|
|       Country|              NULL|
+--------------+------------------+
```



```python
import matplotlib.pyplot as plt

# Convert to Pandas
pdf = spark.sql("SELECT * FROM sales_summary").toPandas()

# Bar chart
plt.figure(figsize=(10,6))
plt.bar(pdf['Country'], pdf['sum(TotalValue)'])
plt.title("Real-Time Total Sales by Country")
plt.xlabel("Country")
plt.ylabel("Total Sales")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Through Spark Structured Streaming, our team successfully replicated real-time analytics by processing incoming data in micro-batches. This allowed us to identify top-performing countries based on total transaction value dynamically. Real-time analytics can be critical in domains such as fraud detection, dynamic pricing, or inventory management — making this exercise both practical and relevant for real-world big data systems.

Task 04

Cloud Analytics with Snowflake

Overview

In Task 4, our team analysed the capabilities of Snowflake, a cloud-native data warehouse platform built for scalability and high-performance analytics. Snowflake was used to efficiently store, query, and analyse a big retail dataset. The platform's sophisticated capabilities, such as auto-scaling compute resources, time travel, and SQL-based analytics, enabled us to extract useful insights from the dataset.

Data Loading

To begin our study, we converted our dataset to CSV format and named it Online.csv. The steps below describe how we put the data into Snowflake.

Database and Schema Setup:

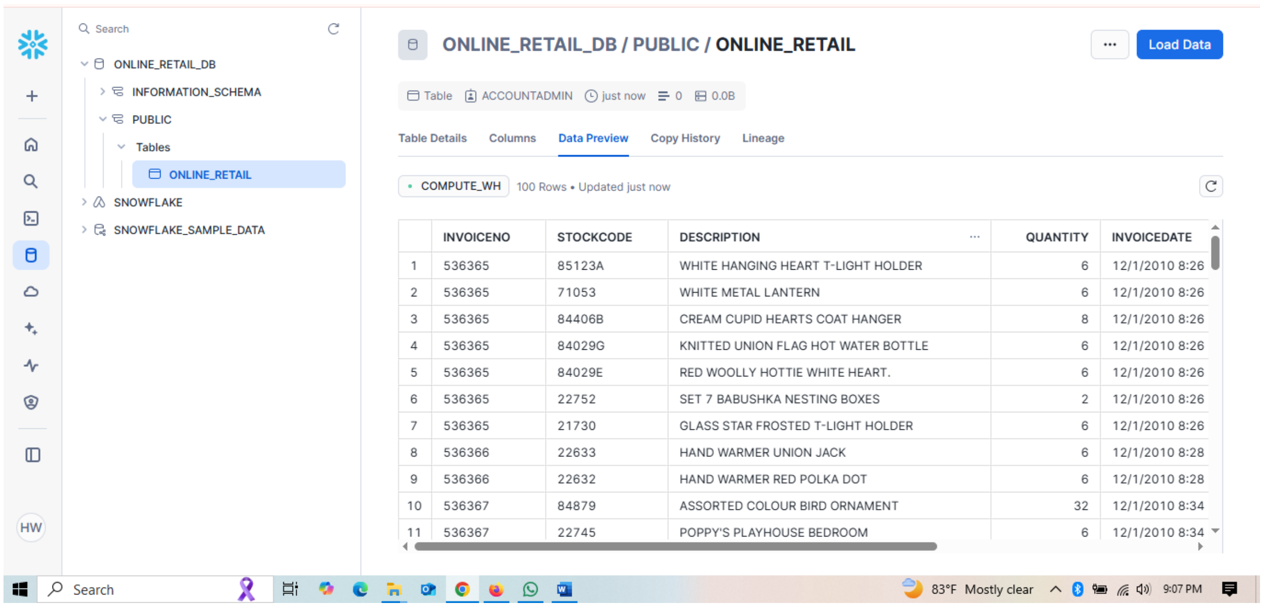
- Snowflake generated a new database named ONLINE_RETAIL_DB.
- The dataset was stored using the default schema, PUBLIC.

Table Creation and Data Import:

- Snowflake's Web UI included the opportunity to construct a table named ONLINE_RETAIL from a file.
- Snowflake's auto-detection feature was used to identify and automatically map column structures.
- Over 540,000 records were successfully added to the database.

Data Storage Optimisation:

- Snowflake's columnar storage format and automated indexing significantly improved query performance.
- The dataset was optimally partitioned to improve retrieval performance and scalability.



ONLINE_RETAIL_DB / PUBLIC / ONLINE_RETAIL

Table · ACCOUNTADMIN · just now · 0 · 0.0B

COMPUTE_WH · 100 Rows • Updated just now

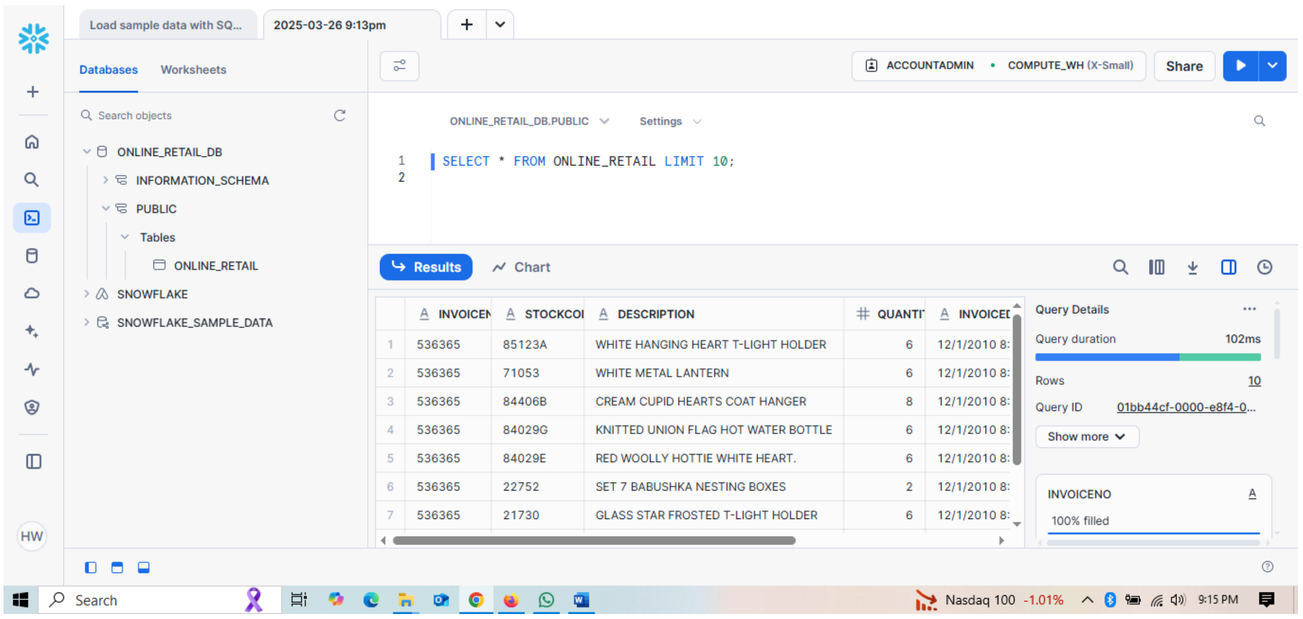
	INVOICENO	STOCKCODE	DESCRIPTION	QUANTITY	INVOICEDATE
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26
2	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26
6	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26
7	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26
8	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28
9	536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28
10	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34
11	536367	22745	POPPY'S PLAYHOUSE BEDROOM	6	12/1/2010 8:34

SQL Query Execution

After successfully uploading the data, we used the SQL Worksheet interface to run various analytical queries:

Query 1: View Records

```
SELECT * FROM ONLINE_RETAIL LIMIT 10;
```



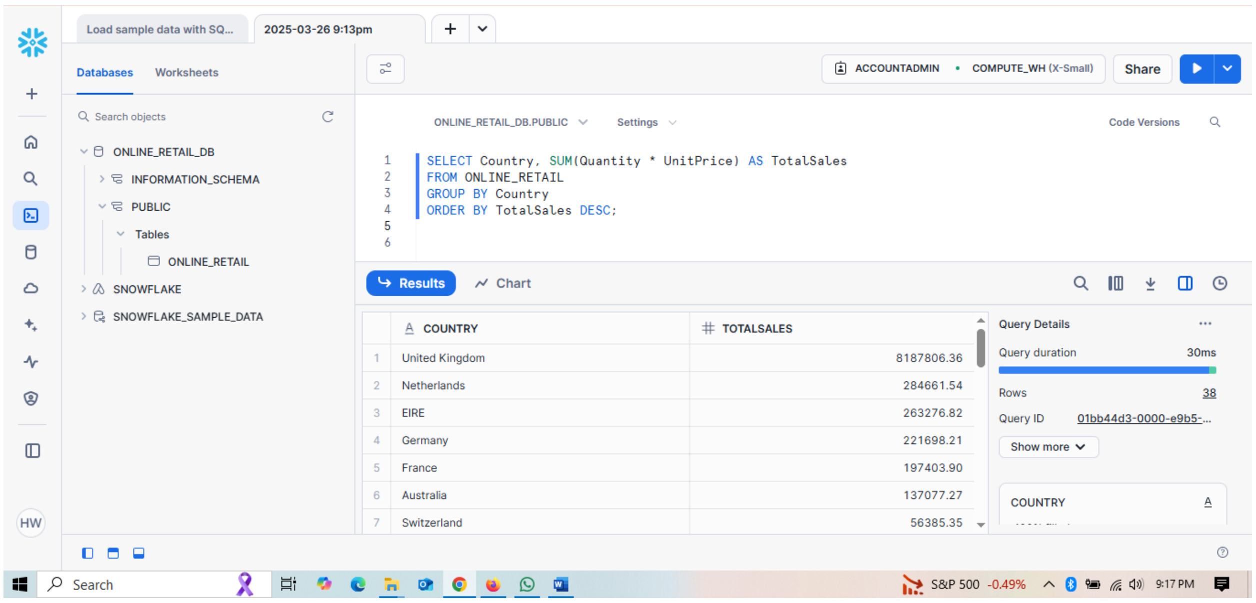
```
SELECT * FROM ONLINE_RETAIL LIMIT 10;
```

	INVOICEN	STOCKCOI	DESCRIPTION	QUANTI	INVOICEI
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:
2	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:
6	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:
7	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:

Query Details
Query duration 102ms
Rows 10
Query ID 01bb44cf-0000-e8f4-0...
INVOICENO 100% filled

Query 2: Total Sales by Country

```
SELECT Country, SUM(Quantity * UnitPrice) AS TotalSales

FROM ONLINE_RETAIL

GROUP BY Country

ORDER BY TotalSales DESC;
```



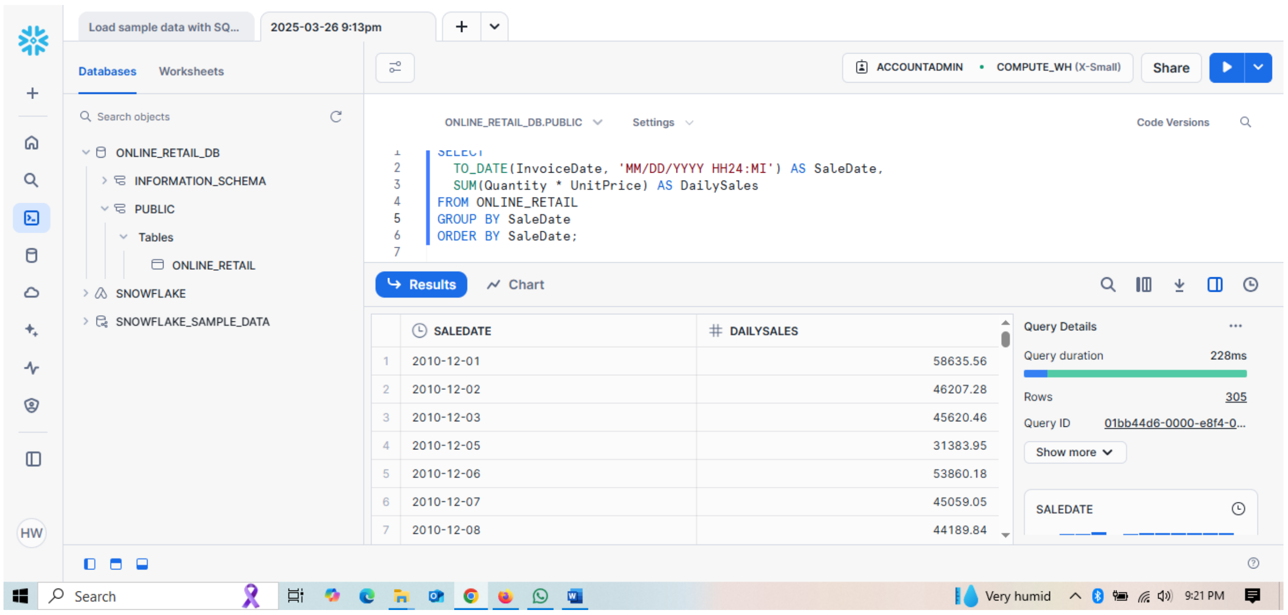
```
SELECT Country, SUM(Quantity * UnitPrice) AS TotalSales
FROM ONLINE_RETAIL
GROUP BY Country
ORDER BY TotalSales DESC;
```

	COUNTRY	TOTALSALES
1	United Kingdom	8187806.36
2	Netherlands	284661.54
3	EIRE	263276.82
4	Germany	221698.21
5	France	197403.90
6	Australia	137077.27
7	Switzerland	56385.35

Query Details
Query duration 30ms
Rows 38
Query ID 01bb44d3-0000-e9b5-...

Query 3: Daily Sales Trend

```
SELECT TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI') AS SaleDate,

       SUM(Quantity * UnitPrice) AS DailySales

FROM ONLINE_RETAIL

GROUP BY SaleDate

ORDER BY SaleDate;
```



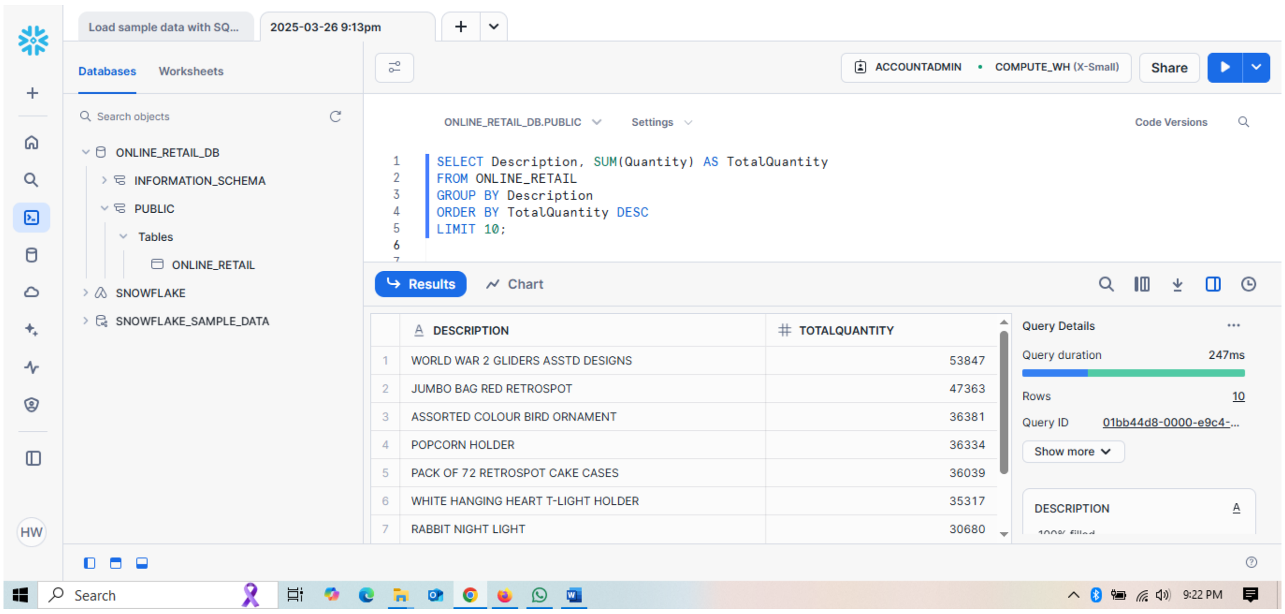
```
SELECT
  TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI') AS SaleDate,
  SUM(Quantity * UnitPrice) AS DailySales
FROM ONLINE_RETAIL
GROUP BY SaleDate
ORDER BY SaleDate;
```

	SALEDATE	DAILYSALES
1	2010-12-01	58635.56
2	2010-12-02	46207.28
3	2010-12-03	45620.46
4	2010-12-05	31383.95
5	2010-12-06	53860.18
6	2010-12-07	45059.05
7	2010-12-08	44189.84

Query Details
Query duration 228ms
Rows 305
Query ID 01bb44d6-0000-e8f4-0...
Show more
SALEDATE

Query 4: Top-Selling Products

```
SELECT Description, SUM(Quantity) AS TotalQuantity

FROM ONLINE_RETAIL

GROUP BY Description

ORDER BY TotalQuantity DESC

LIMIT 10;
```



```
SELECT Description, SUM(Quantity) AS TotalQuantity
FROM ONLINE_RETAIL
GROUP BY Description
ORDER BY TotalQuantity DESC
LIMIT 10;
```

	DESCRIPTION	TOTALQUANTITY
1	WORLD WAR 2 GLIDERS ASSTD DESIGNS	53847
2	JUMBO BAG RED RETROSPOT	47363
3	ASSORTED COLOUR BIRD ORNAMENT	36381
4	POPCORN HOLDER	36334
5	PACK OF 72 RETROSPOT CAKE CASES	36039
6	WHITE HANGING HEART T-LIGHT HOLDER	35317
7	RABBIT NIGHT LIGHT	30680

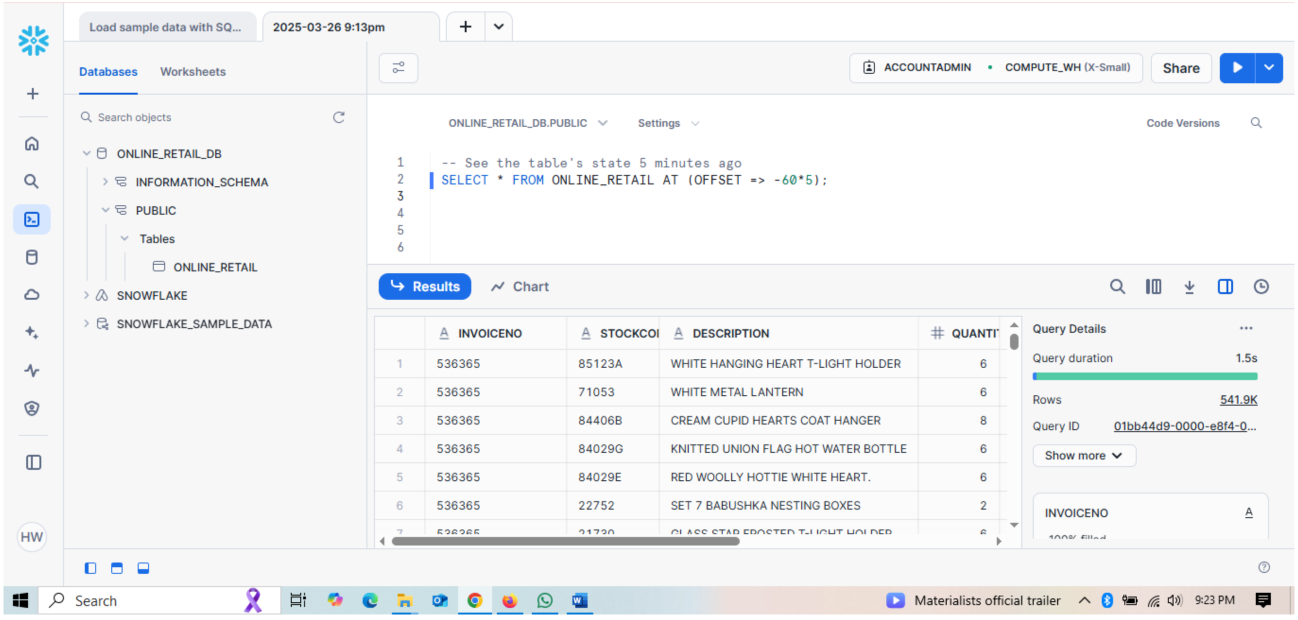
Query Details
Query duration 247ms
Rows 10
Query ID 01bb44d8-0000-e9c4-...

Snowflake Features Used

Time Travel

We demonstrated Time Travel by querying the table as it existed a few minutes ago using:

```
SELECT * FROM ONLINE_RETAIL AT (OFFSET => -60*5);
```



```
-- See the table's state 5 minutes ago
SELECT * FROM ONLINE_RETAIL AT (OFFSET => -60*5);
```

	INVOICENO	STOCKCOI	DESCRIPTION	QUANTI
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6
2	536365	71053	WHITE METAL LANTERN	6
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6
6	536365	22752	SET 7 BABUSHKA NESTING BOXES	2

Query Details
Query duration 1.5s
Rows 541.9K
Query ID 01bb44d9-0000-e8f4-0...

This feature is especially useful for auditing, undoing accidental deletions, or recovering previous data states.

Query Optimization

Snowflake automatically optimized our queries using its columnar storage, result caching, and separate compute/storage architecture. This made our queries execute in milliseconds even on a large dataset.

Snowflake provided a powerful and intuitive platform for running cloud-based analytics at scale. Its ability to ingest structured data quickly, run optimized SQL queries, and handle complex tasks like time travel without manual indexing made it an ideal tool for enterprise-grade data warehousing.

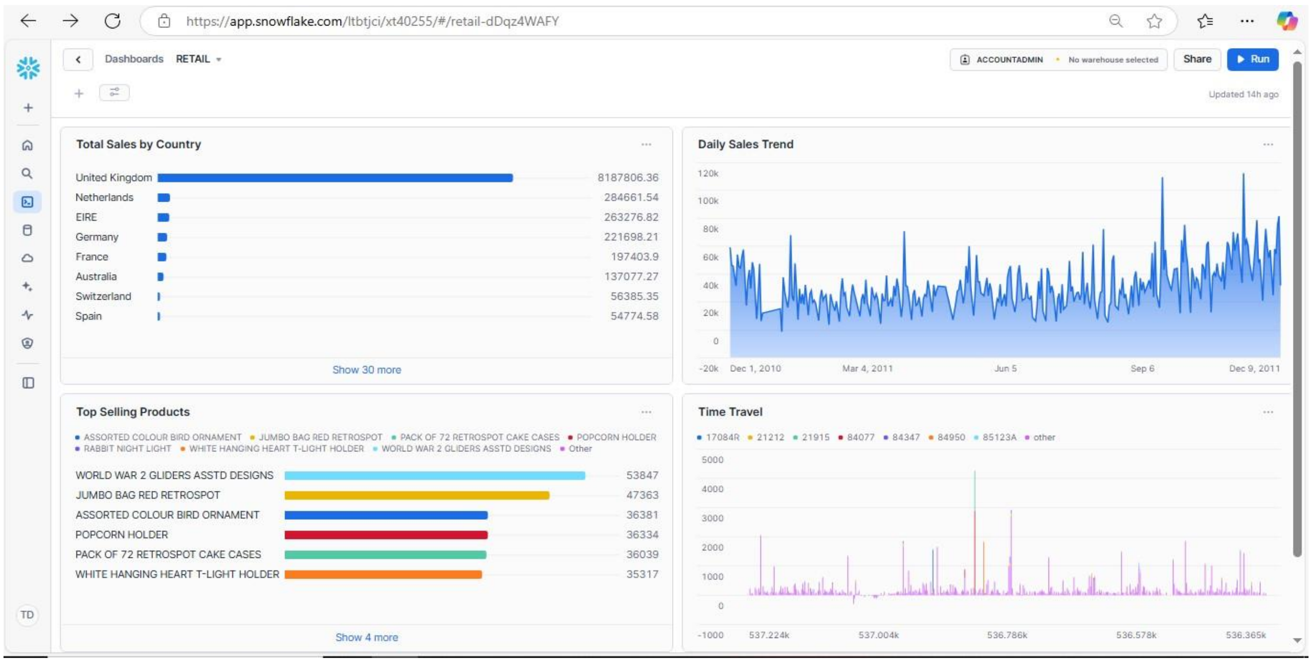
Dashboard Components:

1. **Total Sales by Country:**
    - A bar chart visualizing sales performance across different countries.
    - The United Kingdom dominates in total sales, followed by other regions with smaller contributions.
    - This visualization aids in identifying lucrative markets and potential areas for expansion.

2. **Daily Sales Trend:**
    - A time-series graph depicting sales fluctuations over a defined period.
    - Peaks indicate high-demand periods, useful for planning promotions and optimizing stock levels.
    - Sudden spikes or drops in sales can highlight external influences like discounts, seasonality, or market shifts.

3. **Top Selling Products:**
    - A bar chart highlighting the most popular products in terms of sales volume.
    - Provides insights into customer preferences and product demand.
    - Helps in making data-driven decisions for inventory restocking and promotional activities.

4. **Time Travel Analysis:**
    - Snowflake’s time-travel feature is utilized to analyze past sales records and compare historical data with recent trends.
    - Enables businesses to track performance over different time periods and evaluate growth patterns.


https://app.snowflake.com/ltbtjci/xt40255/#/retail-dDqz4WAFY
Dashboards
RETAIL
ACCOUNTADMIN
No warehouse selected
Share
Run
Updated 14h ago
TD

Total Sales by Country

United Kingdom	8187806.36
Netherlands	284661.54
EIRE	263276.82
Germany	221698.21
France	197403.9
Australia	137077.27
Switzerland	56385.35
Spain	54774.58

Show 30 more

Daily Sales Trend

120k
100k
80k
60k
40k
20k
0
-20k
Dec 1, 2010
Mar 4, 2011
Jun 5
Sep 6
Dec 9, 2011

Top Selling Products

● ASSORTED COLOUR BIRD ORNAMENT ● JUMBO BAG RED RETROSPOT ● PACK OF 72 RETROSPOT CAKE CASES ● POPCORN HOLDER ● RABBIT NIGHT LIGHT ● WHITE HANGING HEART T-LIGHT HOLDER ● WORLD WAR 2 GLIDERS ASSTD DESIGNS ● Other

WORLD WAR 2 GLIDERS ASSTD DESIGNS	53847
JUMBO BAG RED RETROSPOT	47363
ASSORTED COLOUR BIRD ORNAMENT	36381
POPCORN HOLDER	36334
PACK OF 72 RETROSPOT CAKE CASES	36039
WHITE HANGING HEART T-LIGHT HOLDER	35317

Show 4 more

Time Travel

These visualizations improve comprehension of sales performance, allowing stakeholders to make educated business decisions based on data-driven insights. The dashboard successfully converts raw data into meaningful patterns, so it facilitates strategic planning and operational efficiency.

Task 05

Predictive Analytics with Spark MLlib

Overview

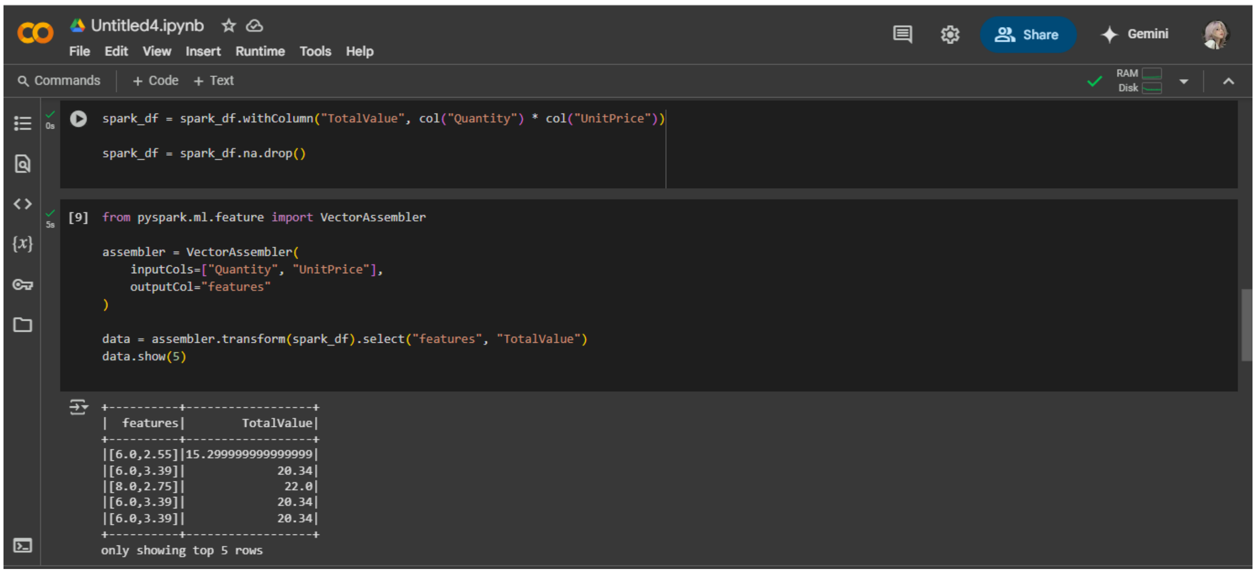
In this final task, we applied machine learning techniques using Apache Spark MLlib to build a regression model. The objective was to predict the total value of a transaction (TotalValue) based on the quantity and price of purchased items. We used Google Colab and PySpark for this task due to their support for distributed computing and scalable machine learning workflows.

Data Preprocessing

We began by calculating a new target column:

```
spark_df = spark_df.withColumn("TotalValue", col("Quantity") * col("UnitPrice"))
```

All rows with missing values were dropped. Next, we used VectorAssembler to combine Quantity and UnitPrice into a single features vector required by Spark MLlib models.



```
spark_df = spark_df.withColumn("TotalValue", col("Quantity") * col("UnitPrice"))

spark_df = spark_df.na.drop()
```

```
[9] from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(
    inputCols=["Quantity", "UnitPrice"],
    outputCol="features"
)

data = assembler.transform(spark_df).select("features", "TotalValue")
data.show(5)
```

```
+----------+-----------------+
|  features|       TotalValue|
+----------+-----------------+
|[6.0,2.55]|15.299999999999999|
|[6.0,3.39]|            20.34|
|[8.0,2.75]|             22.0|
|[6.0,3.39]|            20.34|
|[6.0,3.39]|            20.34|
+----------+-----------------+
only showing top 5 rows
```

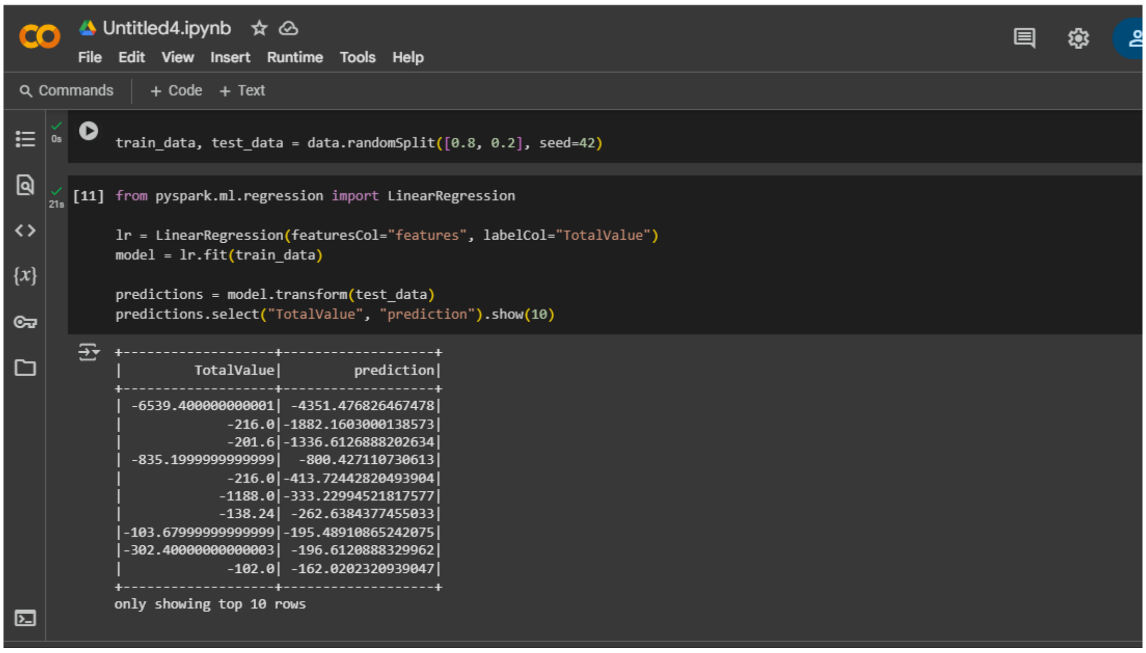
Model Building

We split the data into training and testing sets:

```
train_data, test_data = data.randomSplit([0.8, 0.2], seed=42)
```

We then trained a Linear Regression model:

```
lr = LinearRegression(featuresCol="features", labelCol="TotalValue")

model = lr.fit(train_data)
```



```
train_data, test_data = data.randomSplit([0.8, 0.2], seed=42)
```

```
from pyspark.ml.regression import LinearRegression

lr = LinearRegression(featuresCol="features", labelCol="TotalValue")
model = lr.fit(train_data)

predictions = model.transform(test_data)
predictions.select("TotalValue", "prediction").show(10)
```

```
+------------------+------------------+
|        TotalValue|        prediction|
+------------------+------------------+
| -6539.400000000001| -4351.476826467478|
|             -216.0|-1882.1603000138573|
|             -201.6|-1336.6126888202634|
| -835.1999999999999|  -800.427110730613|
|             -216.0|-413.72442820493904|
|            -1188.0|-333.22994521817577|
|            -138.24| -262.6384377455033|
|-103.67999999999999|-195.48910865242075|
|-302.40000000000003| -196.6120888329962|
|             -102.0| -162.0202320939047|
+------------------+------------------+
only showing top 10 rows
```

Evaluation

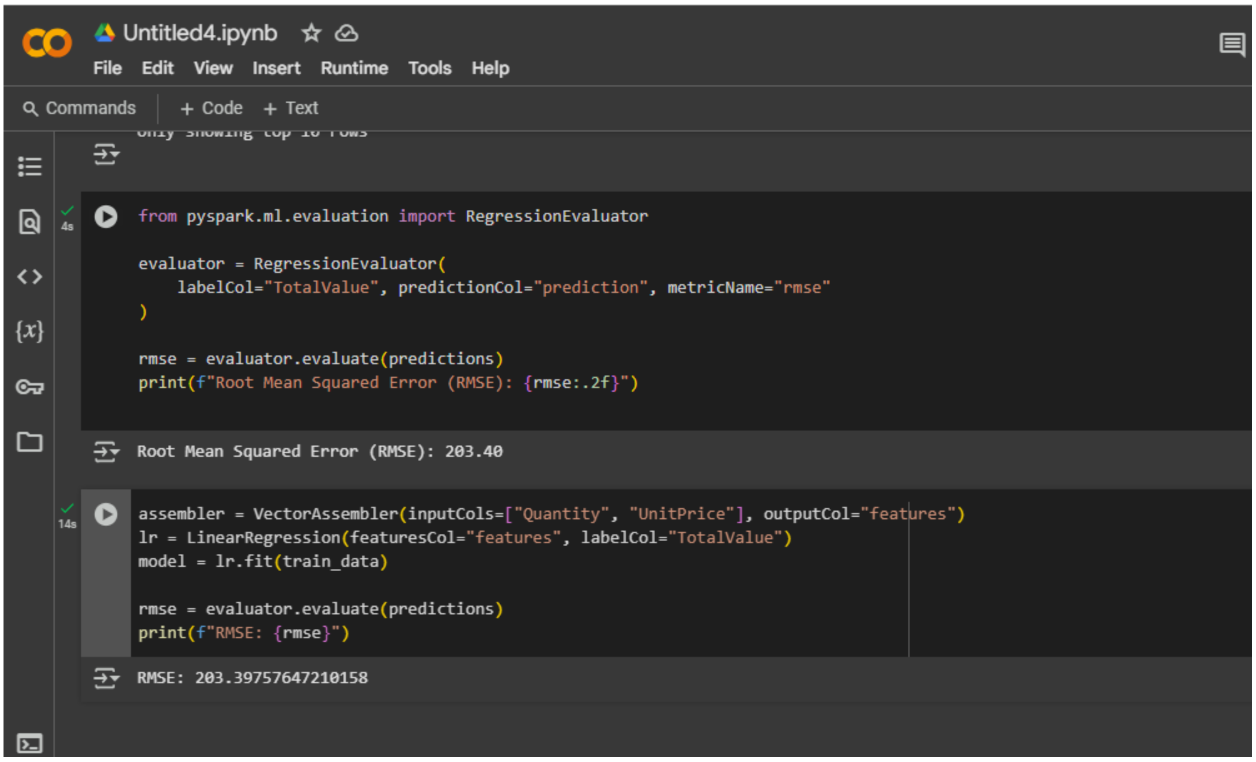
We evaluated the model using Root Mean Squared Error (RMSE):

```
rmse = evaluator.evaluate(predictions)

print(f"RMSE: {rmse}")
```

```
Root Mean Squared Error (RMSE): 203.40
```

The model was able to predict transaction values with reasonable accuracy.

This task showcased how Spark MLlib can be used for scalable predictive modeling. Although our model used only two input features, it performed reasonably well. In future iterations, we could include categorical variables like Country, use classification to predict returns, or apply more advanced algorithms such as decision trees or gradient boosting.



```
only showing top 10 rows
```

```
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(
    labelCol="TotalValue", predictionCol="prediction", metricName="rmse"
)

rmse = evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

```
Root Mean Squared Error (RMSE): 203.40
```

```
assembler = VectorAssembler(inputCols=["Quantity", "UnitPrice"], outputCol="features")
lr = LinearRegression(featuresCol="features", labelCol="TotalValue")
model = lr.fit(train_data)

rmse = evaluator.evaluate(predictions)
print(f"RMSE: {rmse}")
```

```
RMSE: 203.39757647210158
```

References

advsyscon, 2024. *What is a Snowflake, and why is it used?*. [Online]
Available at: https://www.advsyscon.com/blog/snowflake-cloud-data-warehouse/#:~:text=Organizations%20use%20it%20to%20efficiently,independently%20based%20on%20workload%20demands.

amazon, 2025. *What is Apache Spark?*. [Online]
Available at: https://aws.amazon.com/what-is/apache-spark/

snowflake, 2025. *Introduction to Snowflake.* [Online]
Available at: https://docs.snowflake.com/en/user-guide-intro