Cloud Analytics

Assignment 2

Student name: Ahmed Ibraheem Ahmed Badwy	**Student ID:** 300389393
Student name: Anas Ibrahin Ali Elbattra	**Student ID:** 300389368
Student name: Esmael Alahmady Ebrahim Ezz	**Student ID:** 300389885
Student name: Yousef Abdelhaleem Ahmed Shindy	**Student ID:** 300389365

University of Ottawa, Ca

Part 1

1- **GlusterFS:** A distributed file system that provides a global view of a set of files that are spread across multiple servers. scale-out file system that can be scaled up or down by adding or removing servers. fault-tolerant file system which can continue to operate even if one or more servers fail.

- GlusterFS is implemented using a client-server architecture. The client is a FUSE module that allows GlusterFS to be mounted as a regular file system. The server is a daemon that runs on each server in the cluster. The GlusterFS servers communicate with each other to maintain a consistent view of the file system.
- When a client accesses a file on a GlusterFS volume, the client communicates with the servers to determine which server has the most recent copy of the file. The GlusterFS client then reads or writes the file to that server.
- GlusterFS supports a variety of replication and striping strategies. Replication allows GlusterFS to create multiple copies of each file on different servers. This improves the fault tolerance of the file system. Striping allows GlusterFS to divide a file into multiple chunks and store each chunk on a different server. This improves the performance of the file system.

- **Use Cases:**
  - High-performance file sharing
  - Storing and managing large and highly scalable file systems
  - Applications that require high scalability, fault tolerance, and performance

NFS: A client-server protocol distributed file system that allows users to access files stored on remote servers as if they were local.

- NFS is a client-server protocol. The client is a program that runs on the client machine and allows the user to access the NFS file system. The server is a program that runs on the server machine and exports the NFS file system to clients.
- When a client wants to access a file on an NFS server, the client makes a request to the server. The server then sends the file to the client. The NFS client then saves the file to the client machine.
- NFS supports a variety of authentication mechanisms, such as Kerberos and NIS. NFS also supports a variety of file locking mechanisms, which ensure that only one client can write to a file at a time.

- **Use Cases**:
  - File sharing between servers and clients
  - Storing and managing small to medium-sized file systems
  - Applications that require simplicity and performance

2- Apache Hadoop Map-Reduce:

- **Scalability:** MapReduce is highly scalable, that it can be used to process very large datasets across multiple servers. This is because MapReduce breaks down large datasets into smaller tasks that can be processed in parallel.
- **Fault tolerance:** MapReduce is fault-tolerant, meaning that it can continue to operate even if one or more servers fail. This is because MapReduce replicates data across multiple servers and can reschedule tasks to other servers if a server fails.
- **Simplicity:** MapReduce is a relatively simple programming model, making it easy for developers to write parallel applications. This is because MapReduce takes care of the details of distributing data across servers and managing task failures.

The Limitations:

- **Performance:** MapReduce is slower than Spark, especially for iterative workloads. This is because MapReduce has to write intermediate results to disk after each map and reduce phase, while Spark can keep intermediate results in memory.
- **Ease of use:** MapReduce is more difficult to program than Spark due to its two-stage programming model. Spark has a more unified programming model that makes it easier to write and debug parallel applications.
- **Versatility:** MapReduce is primarily designed for batch processing, while Spark can be used for both batch and stream processing. Spark also offers a wider range of libraries for machine learning and other data science tasks.

Spark is a more modern and versatile distributed computing framework than MapReduce. However, MapReduce is still a widely used and powerful tool for batch processing large datasets

3- Apache Spark:

The main difference between the low-level and high-level APIs is the level of control they offer over the underlying execution of Spark jobs. The low-level API gives users complete control over how their data is processed, while the high-level APIs hide many of the implementation details.

- **Low-level API**: Resilient Distributed Datasets (RDDs), provides a direct way to access and manipulate data in Spark.
  - **Advantages:**
    - **Flexibility:** The low-level API provides a more flexible way to represent and manipulate data. For example, RDDs can be used to represent data in a variety of formats, such as lists, tuples, and matrices.
    - **Performance:** The low-level API can be more efficient for certain workloads, such as iterative workloads. This is because the low-level API gives users more control over how their data is processed.
  - **Disadvantages:**
    - **Complexity:** The low-level API can be more difficult to use than the high-level APIs. This is because the low-level API requires users to explicitly manage the partitioning and distribution of their data.
    - **Error-prone:** The low-level API is more error-prone than the high-level APIs. This is because users are responsible for managing many of the implementation details of Spark jobs.

- **Examples:**
    - **Machine learning:** The low-level API can be used to implement custom machine learning algorithms. For example, researchers at the University of California, Berkeley used the low-level API to implement a new algorithm for natural language processing.
    - **Real-time stream processing:** The low-level API can be used to implement real-time stream processing applications. For example, the company Foursquare uses the low-level API to process over 1 billion check-ins per day.
    - **Graph processing:** The low-level API can be used to process large graphs. For example, the company Facebook uses the low-level API to process its social graph, which has over 2 billion nodes and 100 trillion edges.

- **High-level API:** DataFrames and Datasets, provide a more abstract view of data and offer a variety of operations for filtering, sorting, aggregating, and joining data.
  - **Advantages:**
    - **Ease of use:** The high-level APIs are easier to use than the low-level API. This is because the high-level APIs hide many of the implementation details of Spark jobs.
    - **Performance:** The high-level APIs are optimized for performance for most workloads. This is because the high-level APIs use a variety of techniques to improve performance, such as caching and code generation.
    - **Safety:** The high-level APIs are safer to use than the low-level API. This is because the high-level APIs handle many of the potential pitfalls of Spark programming, such as data skew and memory management.
  - **Disadvantages:**
    - **Flexibility:** The high-level APIs are not as flexible as the low-level API. For example, DataFrames and Datasets can only be used to represent data in a structured format.
    - **Performance:** The high-level APIs may not be as efficient for certain workloads, such as iterative workloads. This is because the high-level APIs hide many of the implementation details of Spark jobs.
  - **Examples:**
    - **Data warehousing:** The high-level API can be used to build data warehouses. For example, the company Airbnb uses the high-level API to build its data warehouse, which stores over 100 petabytes of data.
    - **Data analytics:** The high-level API can be used to perform data analytics on large datasets. For example, the company Netflix uses the high-level API to analyze its customer data to improve its recommendation system.
    - **Internet of Things (IoT) processing:** The high-level API can be used to process data from IoT devices. For example, the company General Electric uses the high-level API to process data from its jet engines to predict maintenance needs.

4-

A- Immutability: is a property of an object that means it cannot be changed once it has been created. This is in contrast to mutability, which allows objects to be changed after they have been created.

- **Advantages:**
    - **Predictability:** Immutable objects are more predictable because they cannot be changed after they have been created. This makes it easier to reason about the behavior of code that uses immutable objects.
    - **Thread safety:** Immutable objects are thread safe by default, because they cannot be changed by multiple threads at the same time. This makes them ideal for use in multithreaded applications.
    - **Performance:** Immutable objects can be more performant than mutable objects, because they do not need to be synchronized or copied when they are shared between threads.
- **Examples:**
    - **The hash of a file:** is immutable This means that the hash of a file will always be the same, regardless of when the file was created or modified. This makes hashes useful for verifying the integrity of files.
    - **The blockchain:** is immutable This means that once a transaction is added to the blockchain, it cannot be changed. This makes the blockchain secure and tamper-proof.

B- Lazy Evaluation: is a technique where Spark does not execute transformations on a dataset until an action is called. This means that Spark can build a logical execution plan, called the Directed Acyclic Graph (DAG), incrementally as transformations are applied. Spark can then optimize the DAG before executing it, which can lead to significant performance improvements.

- **Positive Impacts:**
    - **Reduced memory usage:** Spark only needs to keep the data in memory that is needed for the current transformation. This can significantly reduce memory usage, especially for large datasets.
    - **Improved performance:** Spark can optimize the DAG before executing it, which can lead to significant performance improvements. For example, Spark can fuse multiple transformations together or reorder transformations to reduce data shuffling.
    - **Increased flexibility:** Lazy evaluation allows you to experiment with different transformations without having to commit to a specific execution plan. This can be helpful for debugging and performance tuning your Spark applications.
- **Negative Impacts:**
    - **Debugging:** Lazy evaluation can make debugging more difficult because it can be difficult to track down the source of an error, especially if the error occurs in a transformation that has not yet been executed.
    - **Performance:** Lazy evaluation can sometimes lead to performance problems, especially if the DAG is not optimized correctly. For example, if Spark chooses to shuffle the data between two transformations, this can significantly reduce performance.

- **Examples:**
    - **Filtering**
    - **Sorting**
    - **Aggregating**
    - **Joining**
    - **Mapping**
    - **Reducing**

C-
SparkContext: Is a central entry point for Spark functionality. It is responsible for coordinating the execution of tasks across a cluster, managing the distribution of data, and scheduling jobs. The SparkContext is the foundation upon which all other Spark functionality is built, and it is the first thing that needs to be created when starting a Spark application.

- The SparkContext is mainly used for lower-level operations, such as creating RDDs, accumulators, and broadcasting variables.
- The SparkContext is a singleton and can only be created once in a Spark application.
- The SparkContext is created using the SparkConf, which allows you to set various Spark configurations.
- The SparkContext provides methods for creating RDDs, accumulators, and broadcasting variables, as well as methods for starting tasks on the executors.
- The SparkContext is used to access the Spark environment and perform operations on it.

SparkSession: is the entry point to the Spark functionality. It is the main starting point for interacting with data and creating a DataFrame. SparkSession is the new entry point of Spark that replaces the old SQLContext and HiveContext. It is a unified entry point to interact with structured and semi-structured data and it also provides support for reading from and writing to a variety of data sources such as Avro, Parquet, JSON, and JDBC.

- The SparkSession is the preferred way to work with Spark data structures such as DataFrames and Datasets, as it provides a more consistent and simpler interface.
- The SparkSession, can be created multiple times within an application.
- The SparkSession, does not have a corresponding configuration object, but you can set configurations using the **.config** method of the SparkSession.
- The SparkSession provide methods for creating DataFrames and Datasets, as well as methods for reading and writing data.
- The SparkSession, is used to access the data stored in Spark and perform operations on it.

D- Spark MLlib: is a machine learning library for Apache Spark. It provides a variety of algorithms for classification, regression, clustering, and collaborative filtering. Spark MLlib algorithms are designed to be scalable and fault-tolerant, making them ideal for processing large datasets.

- **Transformers: A**re algorithms that transform one DataFrame into another DataFrame. For example, a Transformer could be used to tokenize a text column, scale a numerical column, or encode a categorical column.
    - **Tokenizer**
    - **HashingTF**
    - **IDF**
    - **StringIndexer**
    - **OneHotEncoder**
    - **StandardScaler**
    - **MinMaxScaler**

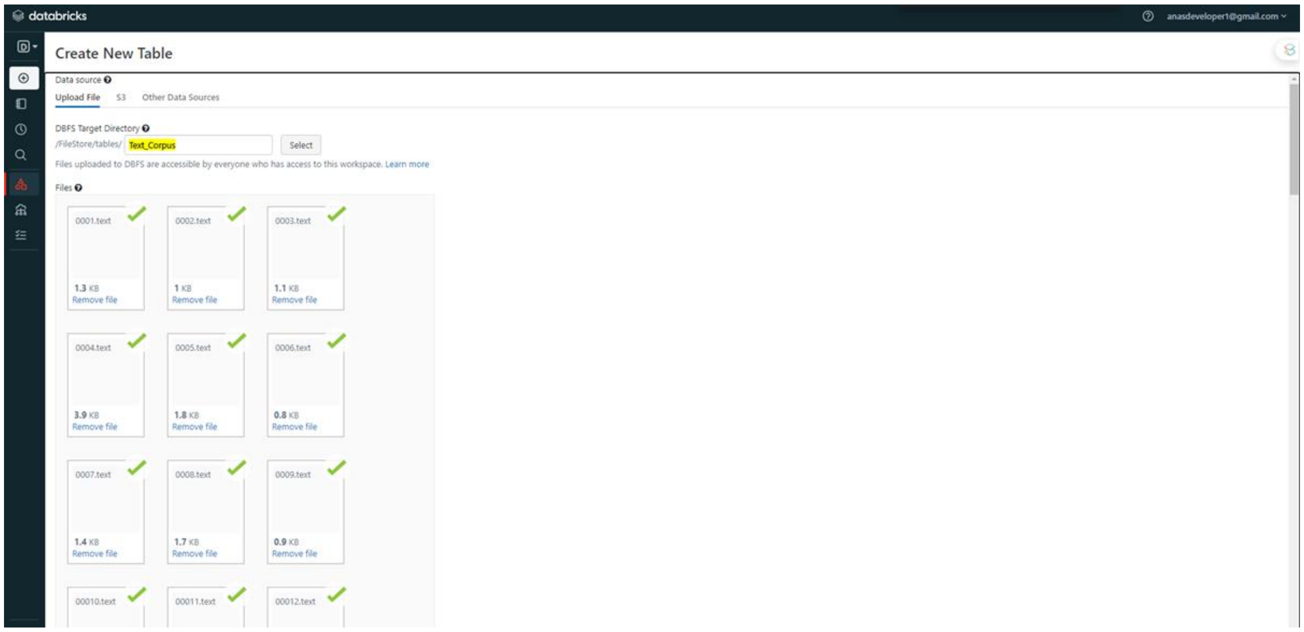
- **Estimators:** are algorithms that fit a model to a DataFrame. The model can then be used to make predictions on new data. For example, an Estimator could be used to train a logistic regression model to predict whether a customer is likely to churn, or a random forest model to predict the price of a house.
    - LogisticRegression
    - RandomForestClassifier
    - RandomForestRegressor
    - KMeans
    - LinearRegression
    - DecisionTreeClassifier
    - DecisionTreeRegressor
- **Evaluators:** are algorithms that evaluate the performance of a machine learning model on a held-out test set. For example, an Evaluator could be used to calculate the accuracy, precision, recall, and F1 score of a classification model.
    - MulticlassClassificationEvaluator
    - BinaryClassificationEvaluator
    - RegressionEvaluator

5- **Spark Streaming:** Is a batch processing framework that is designed to process streaming data as if it were a batch of data. Spark Streaming divides the stream of data into batches and then processes each batch using Spark's batch processing engine.
For example: Spark Streaming can be used to process web traffic data, social media data, and sensor data.
Spark Structured Streaming: is a real-time stream processing framework that is designed to process streaming data as it arrives. Spark Structured Streaming does not divide the stream of data into batches. Instead, Spark Structured Streaming keeps the entire stream of data in memory and processes it as it arrives.
Spark Structured Streaming is a good choice for processing streaming data that is highly stateful.
For example, Spark Structured Streaming can be used to process streaming data from financial markets, IoT devices, and gaming applications.

Part 2

1) Data Transformation Pipelines

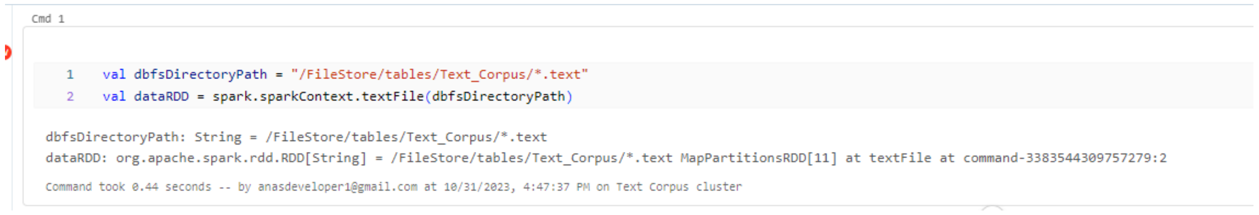
a) Uploaded the files to your DBFS table space.



DBFS Target Directory /FileStore/tables/ Text_Corpus

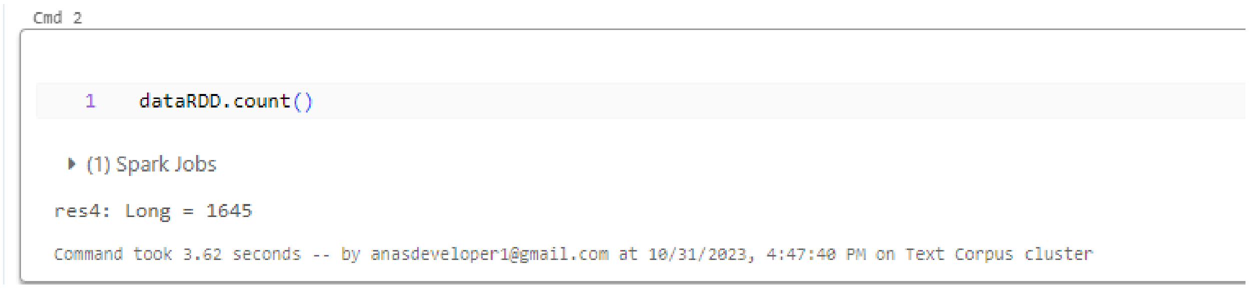
Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more

b) Use Spark Scala to load your data into an RDD.



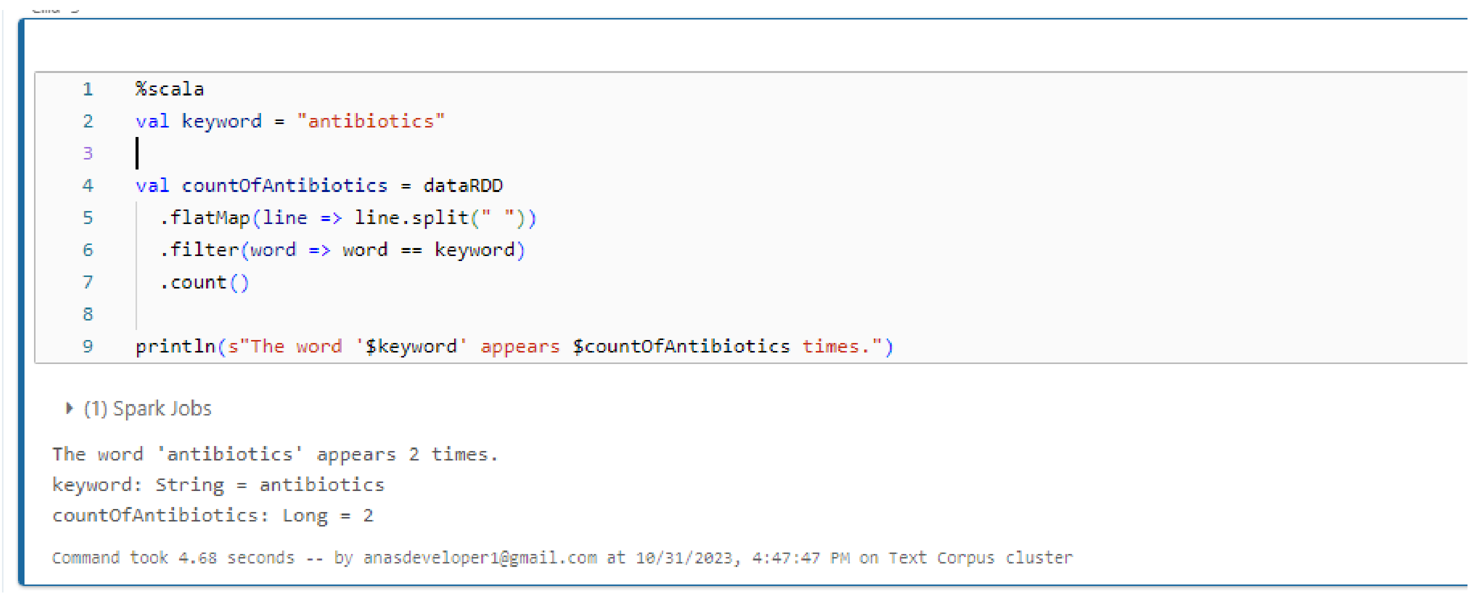
```
Cmd 1

val dbfsDirectoryPath = "/FileStore/tables/Text_Corpus/*.text"
val dataRDD = spark.sparkContext.textFile(dbfsDirectoryPath)

dbfsDirectoryPath: String = /FileStore/tables/Text_Corpus/*.text
dataRDD: org.apache.spark.rdd.RDD[String] = /FileStore/tables/Text_Corpus/*.text MapPartitionsRDD[11] at textFile at command-3383544309757279:2
Command took 0.44 seconds -- by anasdeveloper1@gmail.com at 10/31/2023, 4:47:37 PM on Text Corpus cluster
```

c) Count the number of lines across all the files.



```
Cmd 2

dataRDD.count()

▸ (1) Spark Jobs
res4: Long = 1645
Command took 3.62 seconds -- by anasdeveloper1@gmail.com at 10/31/2023, 4:47:40 PM on Text Corpus cluster
```

d) Find the number of occurrences of the word “antibiotics”.



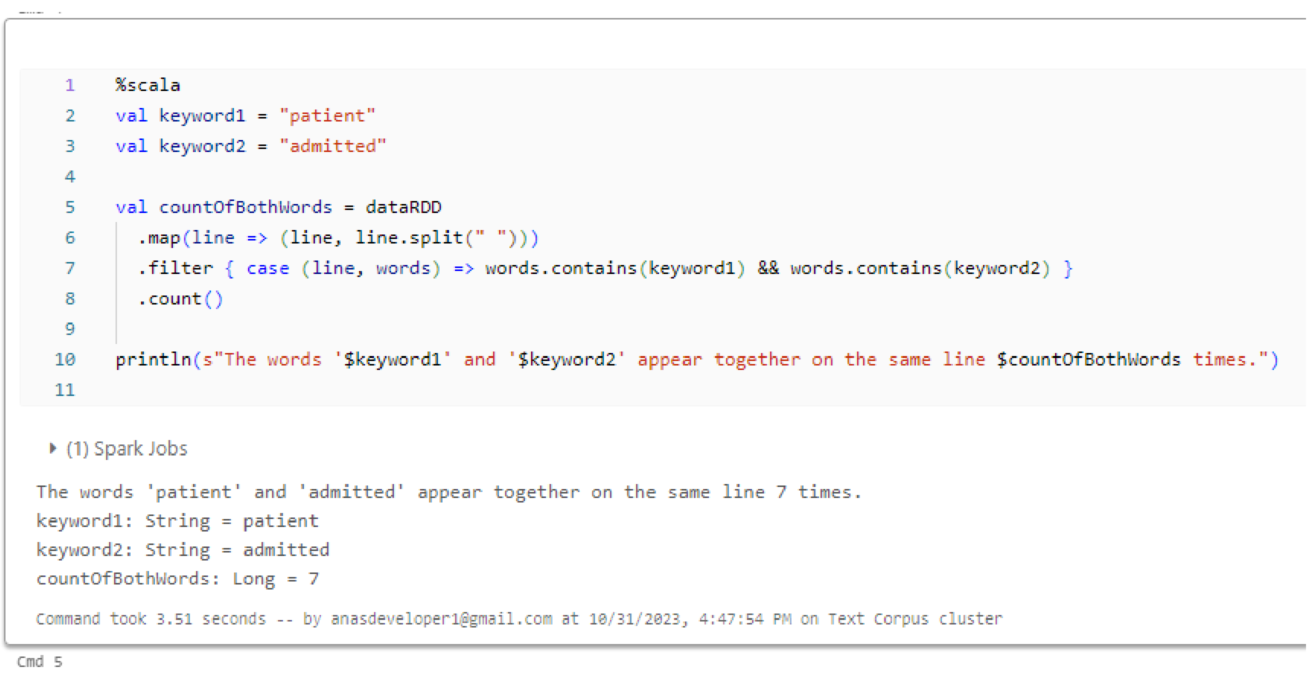
```
%scala
val keyword = "antibiotics"

val countOfAntibiotics = dataRDD
  .flatMap(line => line.split(" "))
  .filter(word => word == keyword)
  .count()

println(s"The word '$keyword' appears $countOfAntibiotics times.")
```

▸ (1) Spark Jobs

```
The word 'antibiotics' appears 2 times.
keyword: String = antibiotics
countOfAntibiotics: Long = 2
```

Command took 4.68 seconds -- by anasdeveloper1@gmail.com at 10/31/2023, 4:47:47 PM on Text Corpus cluster

e) Count the occurrence of the word “patient” and “admitted” on the same line of text. Please ensure that your code contains at least 2 transformation functions in a pipeline.



```
%scala
val keyword1 = "patient"
val keyword2 = "admitted"

val countOfBothWords = dataRDD
  .map(line => (line, line.split(" ")))
  .filter { case (line, words) => words.contains(keyword1) && words.contains(keyword2) }
  .count()

println(s"The words '$keyword1' and '$keyword2' appear together on the same line $countOfBothWords times.")

```

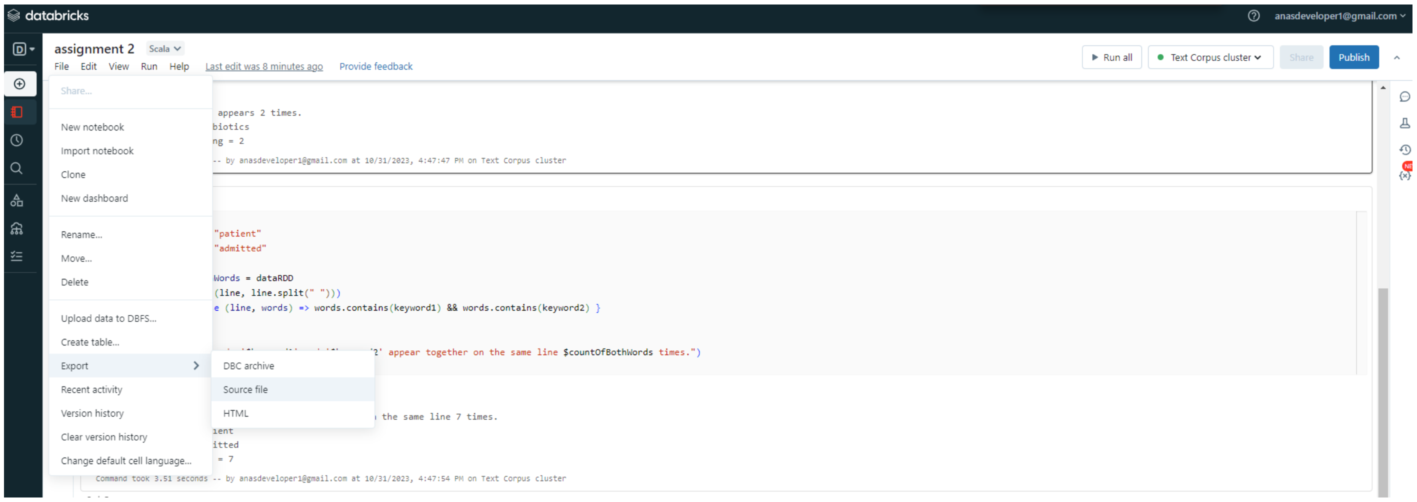
▸ (1) Spark Jobs

```
The words 'patient' and 'admitted' appear together on the same line 7 times.
keyword1: String = patient
keyword2: String = admitted
countOfBothWords: Long = 7
```

Command took 3.51 seconds -- by anasdeveloper1@gmail.com at 10/31/2023, 4:47:54 PM on Text Corpus cluster

Cmd 5

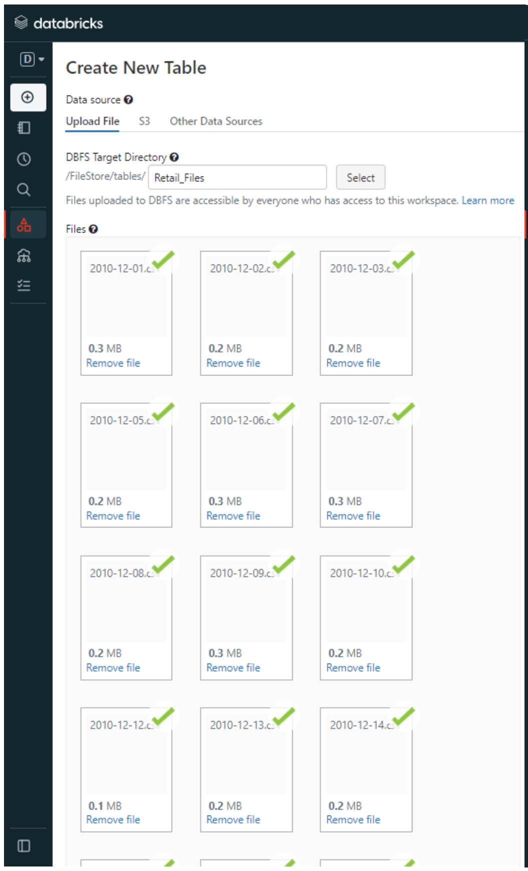
f) Upload your exported (.scala format) notebook as part of your submission. Please include the data in your Brightspace submission.



```
 appears 2 times.
biotics
ng = 2
-- by anasdeveloper1@gmail.com at 10/31/2023, 4:47:47 PM on Text Corpus cluster

"patient"
"admitted"

Words = dataRDD
(line, line.split(" ")))
e (line, words) => words.contains(keyword1) && words.contains(keyword2) }

'2' appear together on the same line $countOfBothWords times.")

 the same line 7 times.
ient
itted
 = 7
Command took 3.51 seconds -- by anasdeveloper1@gmail.com at 10/31/2023, 4:47:54 PM on Text Corpus cluster
```

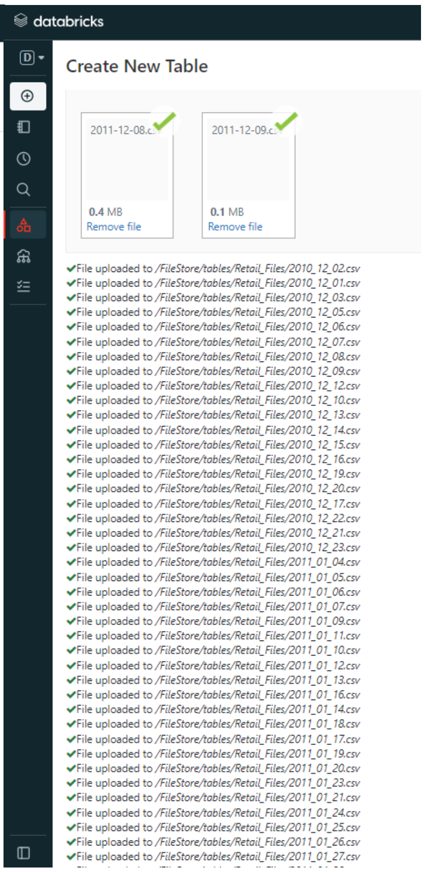
2) Retail Data Analysis.

a) Uploaded the files to your DBFS table space.



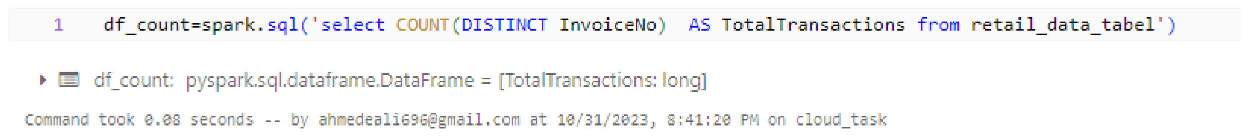
DBFS Target Directory: /FileStore/tables/ Retail_Files

Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more



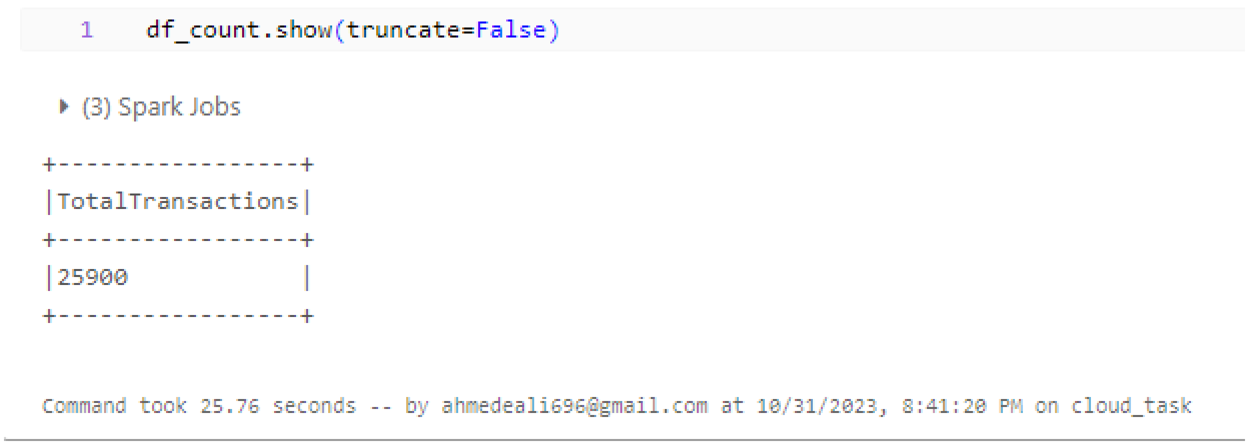
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_02.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_01.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_03.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_05.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_06.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_07.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_08.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_09.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_12.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_10.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_13.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_14.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_15.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_16.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_19.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_20.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_17.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_22.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_21.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2010_12_23.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_04.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_05.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_06.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_07.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_09.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_11.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_10.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_12.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_13.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_16.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_14.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_18.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_17.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_19.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_20.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_23.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_21.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_24.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_25.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_26.csv*
✓File uploaded to */FileStore/tables/Retail_Files/2011_01_27.csv*

b) Output the total number of transactions across all the files and the total value of the transactions.



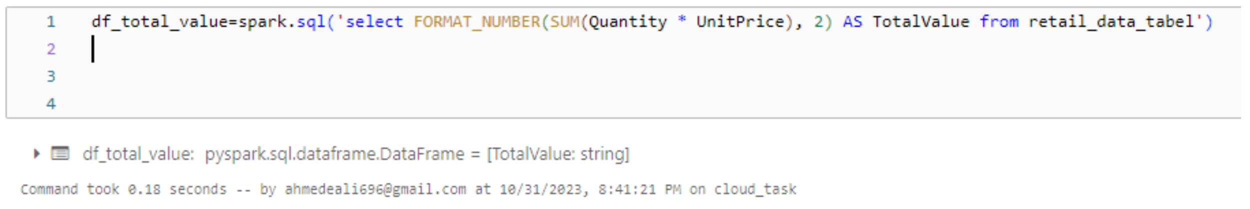
```
df_count=spark.sql('select COUNT(DISTINCT InvoiceNo)  AS TotalTransactions from retail_data_tabel')
```

```
▸ df_count:  pyspark.sql.dataframe.DataFrame = [TotalTransactions: long]
Command took 0.08 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 8:41:20 PM on cloud_task
```



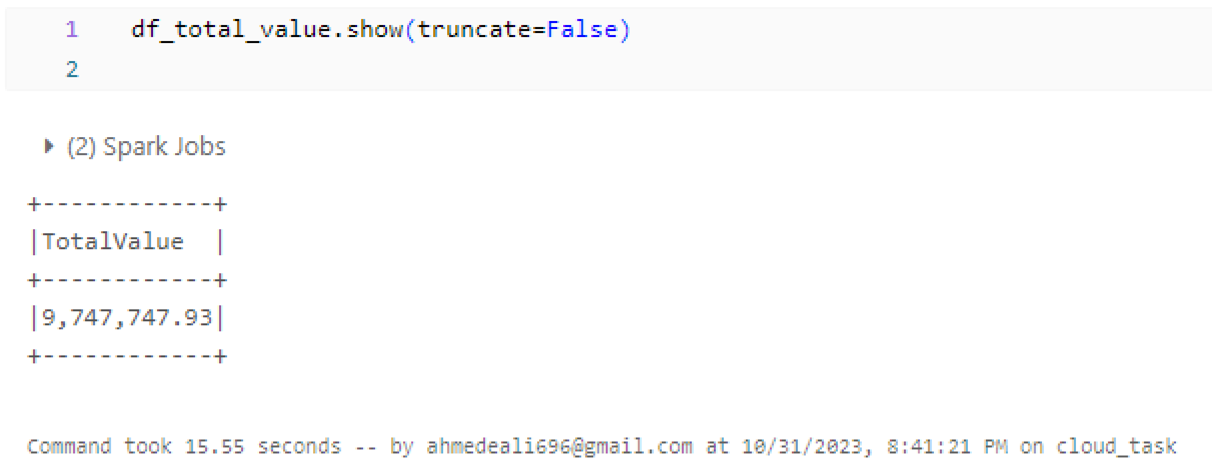
```
df_count.show(truncate=False)
```

```
▸ (3) Spark Jobs
+-----------------+
|TotalTransactions|
+-----------------+
|25900            |
+-----------------+

Command took 25.76 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 8:41:20 PM on cloud_task
```



```
df_total_value=spark.sql('select FORMAT_NUMBER(SUM(Quantity * UnitPrice), 2) AS TotalValue from retail_data_tabel')
```

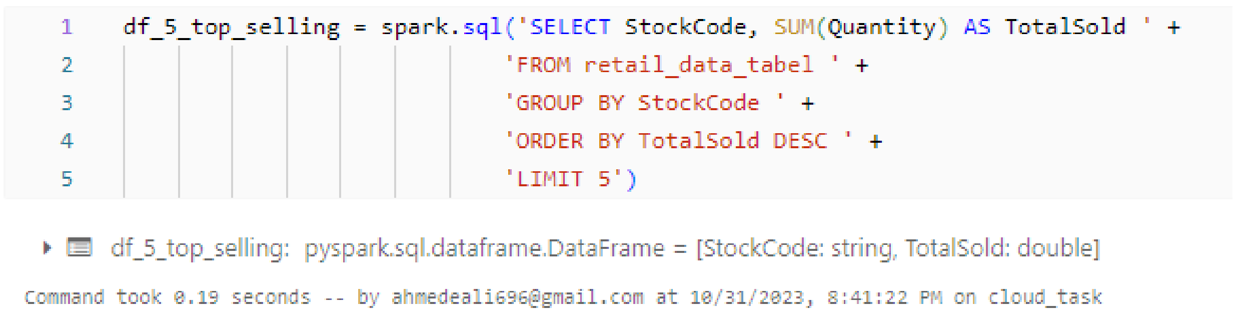
```
▸ df_total_value:  pyspark.sql.dataframe.DataFrame = [TotalValue: string]
Command took 0.18 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 8:41:21 PM on cloud_task
```



```
df_total_value.show(truncate=False)
```

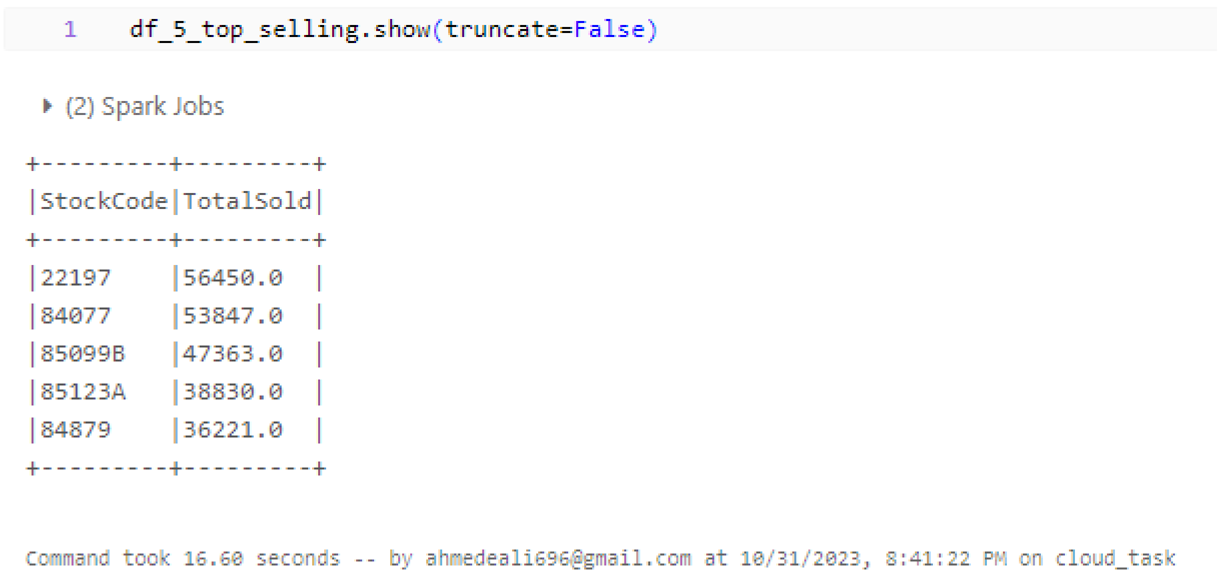
```
▸ (2) Spark Jobs
+------------+
|TotalValue  |
+------------+
|9,747,747.93|
+------------+

Command took 15.55 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 8:41:21 PM on cloud_task
```

c) Output the 5 top-selling products.



```
df_5_top_selling = spark.sql('SELECT StockCode, SUM(Quantity) AS TotalSold ' +
                             'FROM retail_data_tabel ' +
                             'GROUP BY StockCode ' +
                             'ORDER BY TotalSold DESC ' +
                             'LIMIT 5')
```

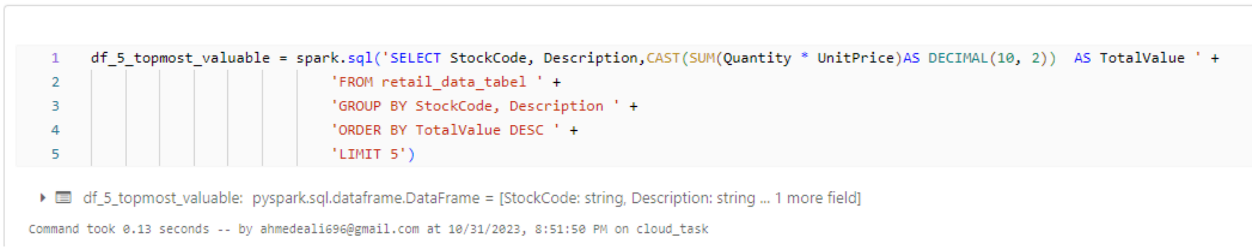
```
df_5_top_selling:  pyspark.sql.dataframe.DataFrame = [StockCode: string, TotalSold: double]
Command took 0.19 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 8:41:22 PM on cloud_task
```



```
df_5_top_selling.show(truncate=False)
```

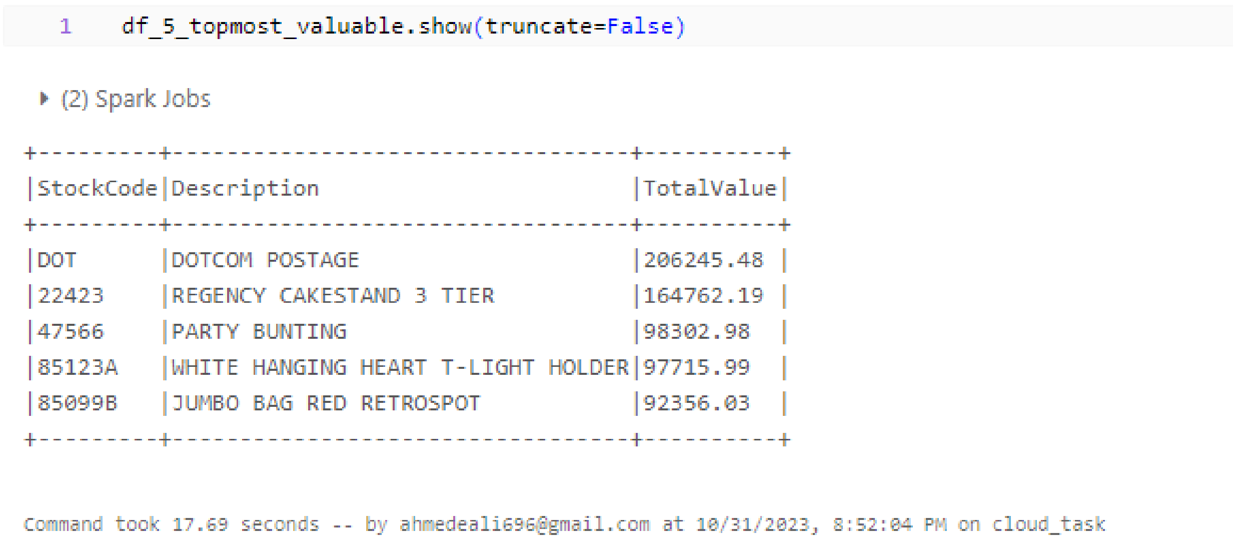
```
(2) Spark Jobs
+---------+---------+
|StockCode|TotalSold|
+---------+---------+
|22197    |56450.0  |
|84077    |53847.0  |
|85099B   |47363.0  |
|85123A   |38830.0  |
|84879    |36221.0  |
+---------+---------+

Command took 16.60 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 8:41:22 PM on cloud_task
```

d) Output the 5 topmost valuable products.



```
df_5_topmost_valuable = spark.sql('SELECT StockCode, Description,CAST(SUM(Quantity * UnitPrice)AS DECIMAL(10, 2))  AS TotalValue ' +
                                  'FROM retail_data_tabel ' +
                                  'GROUP BY StockCode, Description ' +
                                  'ORDER BY TotalValue DESC ' +
                                  'LIMIT 5')
```

```
df_5_topmost_valuable:  pyspark.sql.dataframe.DataFrame = [StockCode: string, Description: string ... 1 more field]
Command took 0.13 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 8:51:50 PM on cloud_task
```



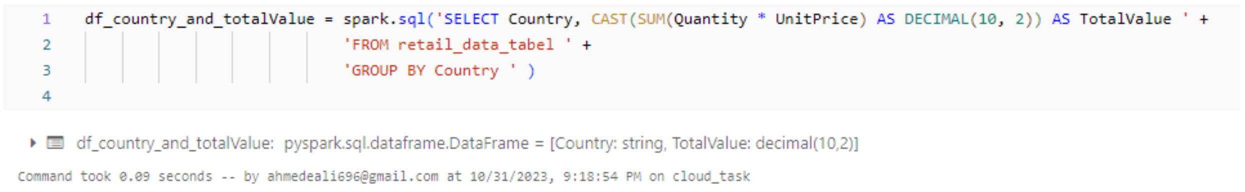
```
df_5_topmost_valuable.show(truncate=False)
```

▸ (2) Spark Jobs

```
+---------+----------------------------------+----------+
|StockCode|Description                       |TotalValue|
+---------+----------------------------------+----------+
|DOT      |DOTCOM POSTAGE                    |206245.48 |
|22423    |REGENCY CAKESTAND 3 TIER          |164762.19 |
|47566    |PARTY BUNTING                     |98302.98  |
|85123A   |WHITE HANGING HEART T-LIGHT HOLDER|97715.99  |
|85099B   |JUMBO BAG RED RETROSPOT           |92356.03  |
+---------+----------------------------------+----------+
```

Command took 17.69 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 8:52:04 PM on cloud_task

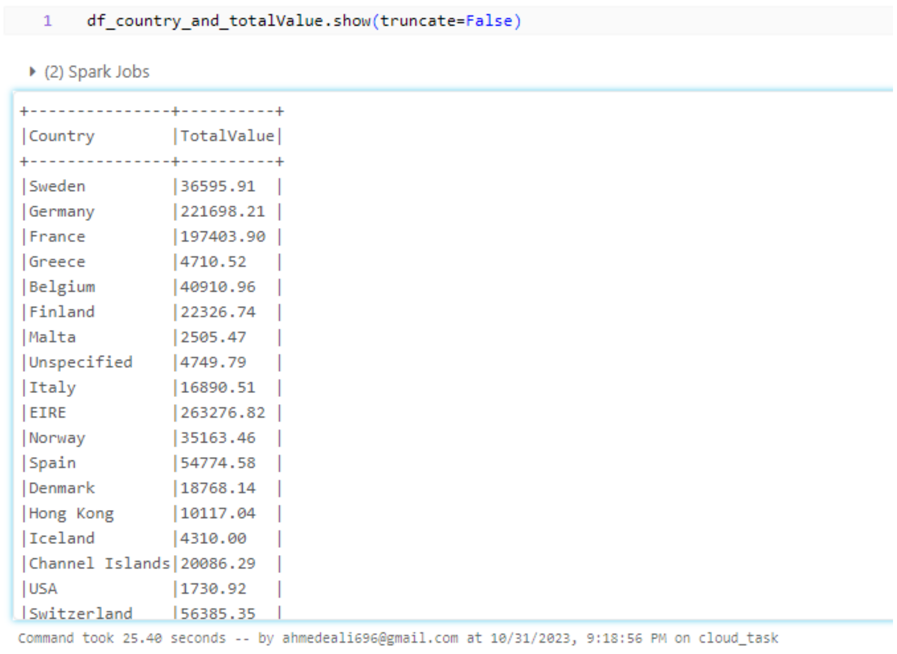
e) Output each country and the total value of their purchases.



```
df_country_and_totalValue = spark.sql('SELECT Country, CAST(SUM(Quantity * UnitPrice) AS DECIMAL(10, 2)) AS TotalValue ' +
                                      'FROM retail_data_tabel ' +
                                      'GROUP BY Country ' )
```

▸ df_country_and_totalValue: pyspark.sql.dataframe.DataFrame = [Country: string, TotalValue: decimal(10,2)]

Command took 0.09 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 9:18:54 PM on cloud_task



```
df_country_and_totalValue.show(truncate=False)
```

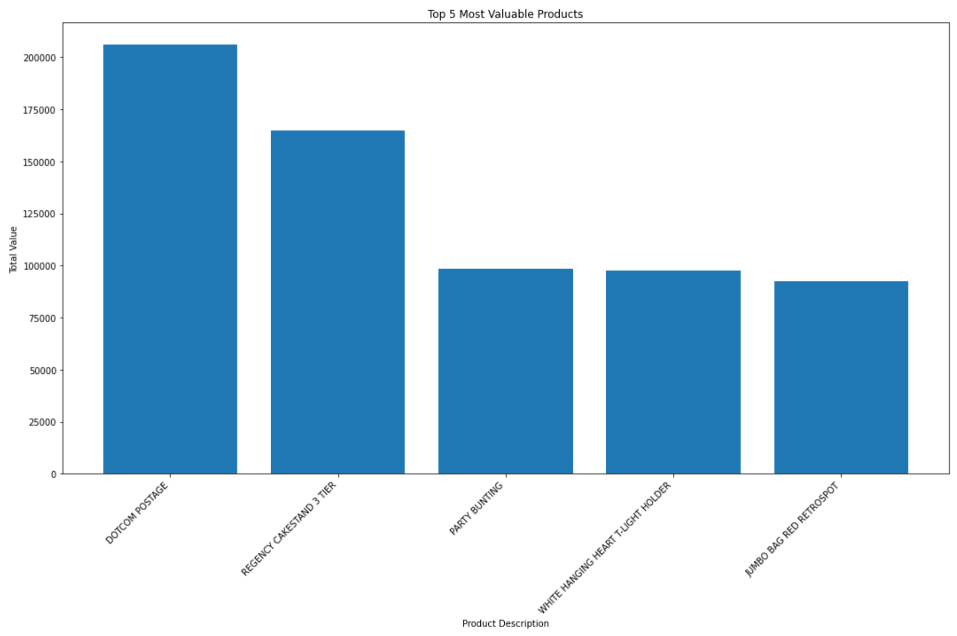
▸ (2) Spark Jobs

```
+---------------+----------+
|Country        |TotalValue|
+---------------+----------+
|Sweden         |36595.91  |
|Germany        |221698.21 |
|France         |197403.90 |
|Greece         |4710.52   |
|Belgium        |40910.96  |
|Finland        |22326.74  |
|Malta          |2505.47   |
|Unspecified    |4749.79   |
|Italy          |16890.51  |
|EIRE           |263276.82 |
|Norway         |35163.46  |
|Spain          |54774.58  |
|Denmark        |18768.14  |
|Hong Kong      |10117.04  |
|Iceland        |4310.00   |
|Channel Islands|20086.29  |
|USA            |1730.92   |
|Switzerland    |56385.35  |
```

Command took 25.40 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 9:18:56 PM on cloud_task

f) Use a graphical representation to describe the result from step (d).

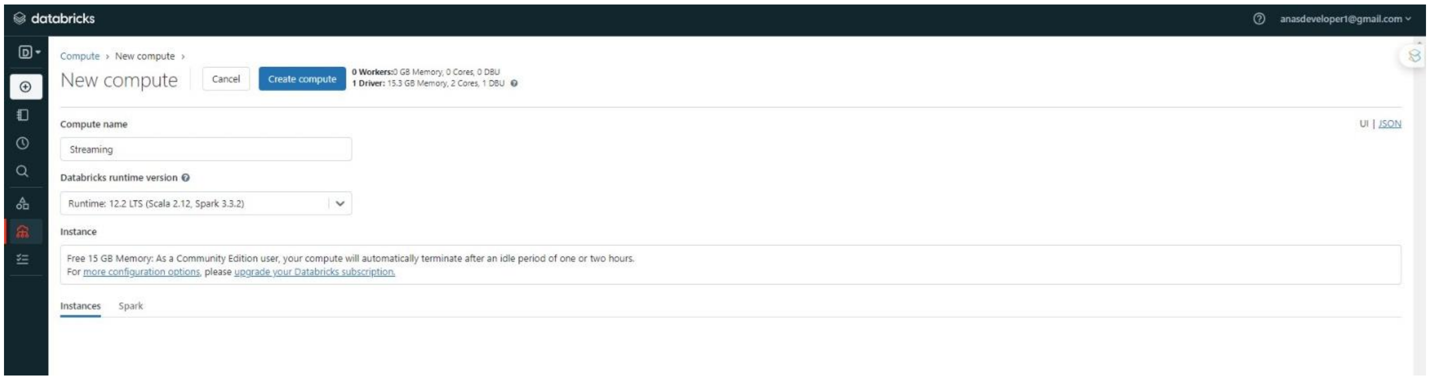
```python

import matplotlib.pyplot as plt

df_pandas = df_5_topmost_valuable.toPandas()


plt.figure(figsize=(15, 10))
plt.bar(df_pandas['Description'], df_pandas['TotalValue'])
plt.xlabel('Product Description')
plt.ylabel('Total Value')
plt.title('Top 5 Most Valuable Products')
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

```



```
Command took 19.77 seconds -- by ahmedeali696@gmail.com at 10/31/2023, 9:29:51 PM on cloud_task
```

3) Structured Streaming.

a) Create a new notebook.



Compute › New compute ›

New compute

0 Workers: 0 GB Memory, 0 Cores, 0 DBU
1 Driver: 15.3 GB Memory, 2 Cores, 1 DBU

Compute name: Streaming

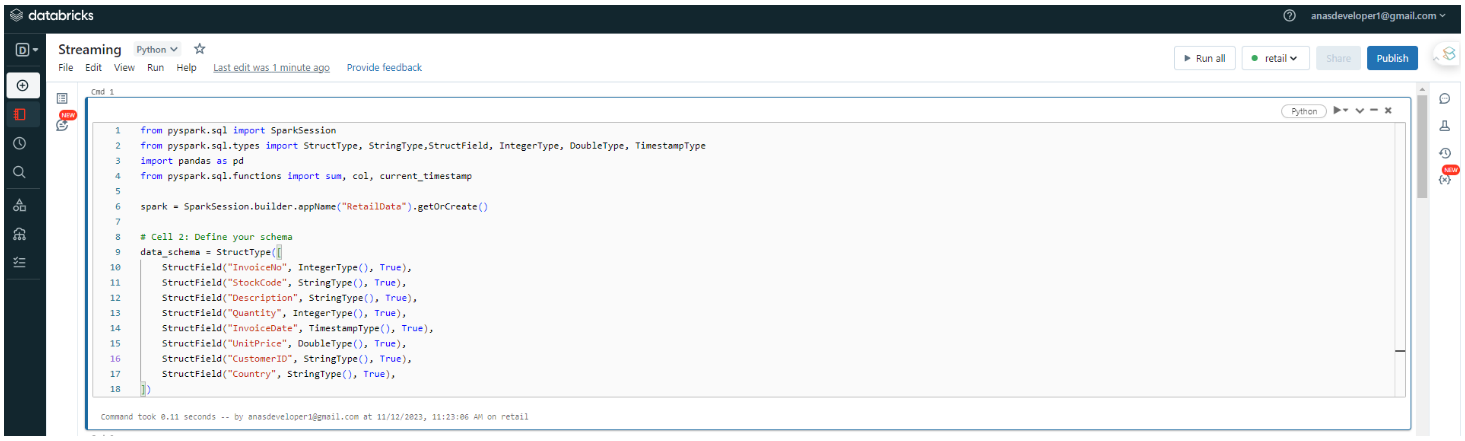
Databricks runtime version: Runtime: 12.2 LTS (Scala 2.12, Spark 3.3.2)

Instance

Free 15 GB Memory: As a Community Edition user, your compute will automatically terminate after an idle period of one or two hours.
For more configuration options, please upgrade your Databricks subscription.

b) Load the retail data as a stream, at 20 files per trigger. For each batch pulled, capture the customer stock aggregates – total stocks, total value.

- **Define your schema.**



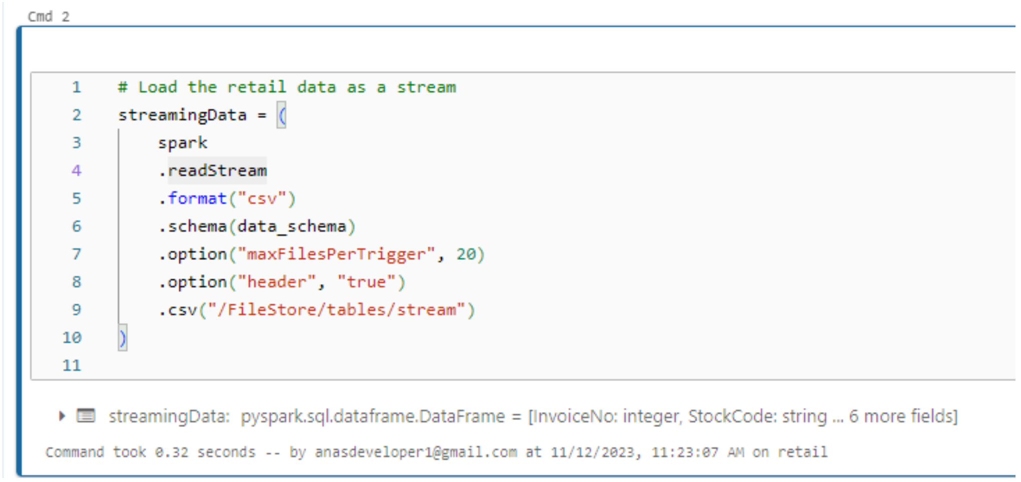
Streaming

Cmd 1

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StringType,StructField, IntegerType, DoubleType, TimestampType
import pandas as pd
from pyspark.sql.functions import sum, col, current_timestamp

spark = SparkSession.builder.appName("RetailData").getOrCreate()

# Cell 2: Define your schema
data_schema = StructType([
    StructField("InvoiceNo", IntegerType(), True),
    StructField("StockCode", StringType(), True),
    StructField("Description", StringType(), True),
    StructField("Quantity", IntegerType(), True),
    StructField("InvoiceDate", TimestampType(), True),
    StructField("UnitPrice", DoubleType(), True),
    StructField("CustomerID", StringType(), True),
    StructField("Country", StringType(), True),
])
```

```
Command took 0.11 seconds -- by anasdeveloper1@gmail.com at 11/12/2023, 11:23:06 AM on retail
```

- **Load the retail data as a stream.**

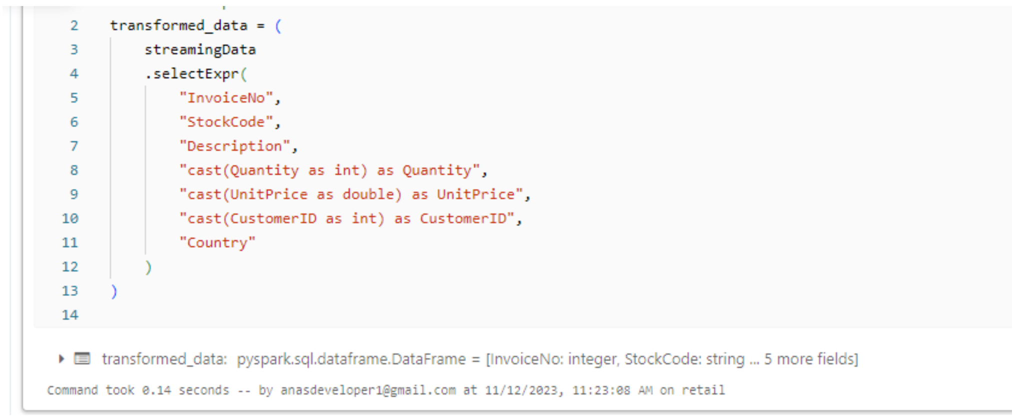


Cmd 2

```python
# Load the retail data as a stream
streamingData = (
    spark
    .readStream
    .format("csv")
    .schema(data_schema)
    .option("maxFilesPerTrigger", 20)
    .option("header", "true")
    .csv("/FileStore/tables/stream")
)

```

```
streamingData:  pyspark.sql.dataframe.DataFrame = [InvoiceNo: integer, StockCode: string ... 6 more fields]
Command took 0.32 seconds -- by anasdeveloper1@gmail.com at 11/12/2023, 11:23:07 AM on retail
```

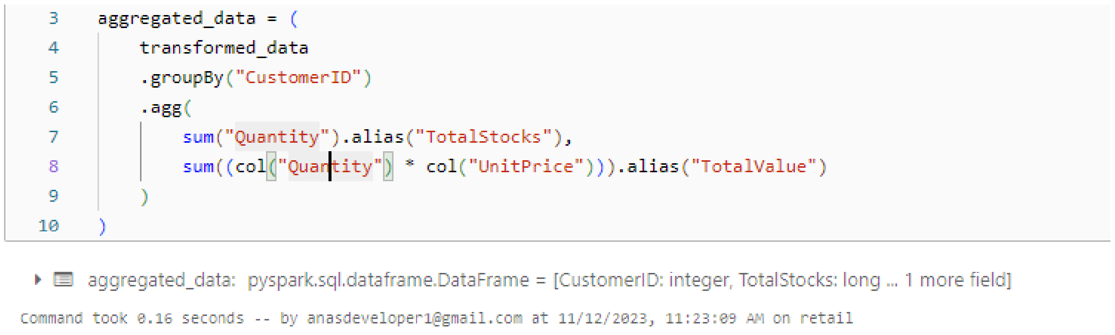
- **Prepare the data.**



```
transformed_data = (
    streamingData
    .selectExpr(
        "InvoiceNo",
        "StockCode",
        "Description",
        "cast(Quantity as int) as Quantity",
        "cast(UnitPrice as double) as UnitPrice",
        "cast(CustomerID as int) as CustomerID",
        "Country"
    )
)

```

```
transformed_data: pyspark.sql.dataframe.DataFrame = [InvoiceNo: integer, StockCode: string ... 5 more fields]
Command took 0.14 seconds -- by anasdeveloper1@gmail.com at 11/12/2023, 11:23:08 AM on retail
```

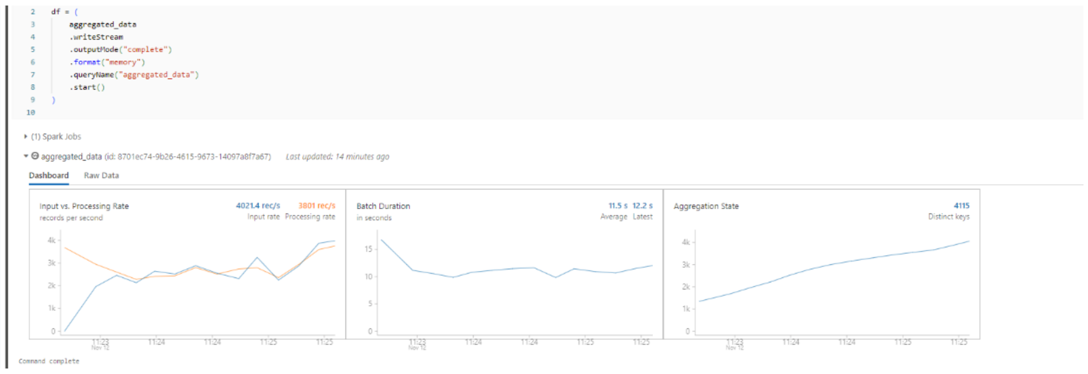
- **Capture customer stock aggregates.**



```
aggregated_data = (
    transformed_data
    .groupBy("CustomerID")
    .agg(
        sum("Quantity").alias("TotalStocks"),
        sum((col("Quantity") * col("UnitPrice"))).alias("TotalValue")
    )
)
```

```
aggregated_data: pyspark.sql.dataframe.DataFrame = [CustomerID: integer, TotalStocks: long ... 1 more field]
Command took 0.16 seconds -- by anasdeveloper1@gmail.com at 11/12/2023, 11:23:09 AM on retail
```

- **Start the streaming query and get the results.**

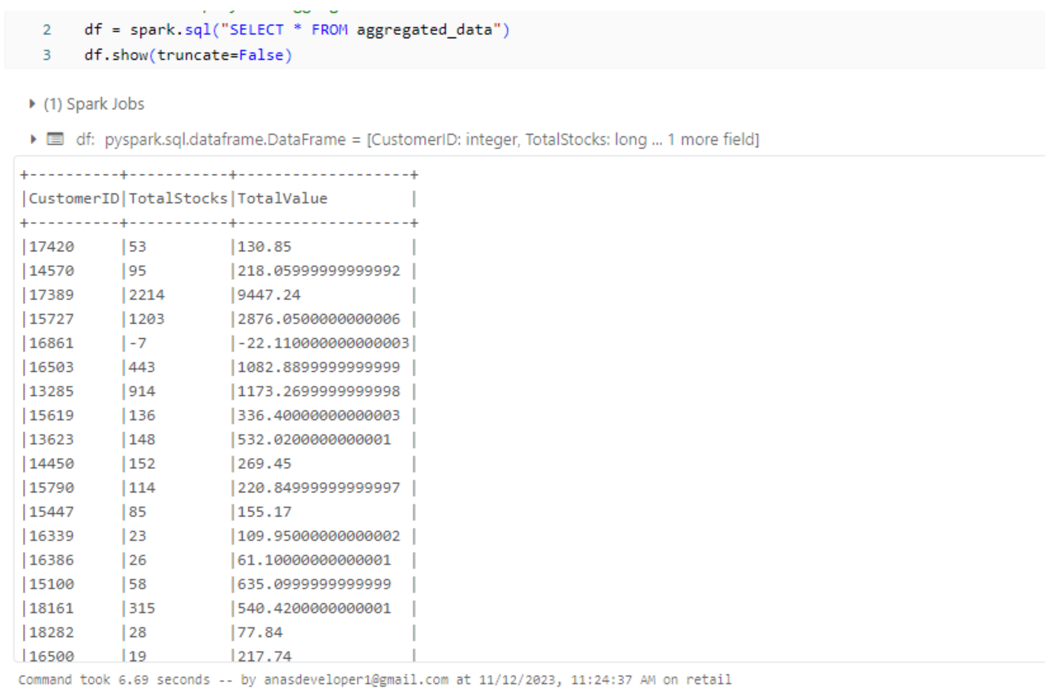


```
df = (
    aggregated_data
    .writeStream
    .outputMode("complete")
    .format("memory")
    .queryName("aggregated_data")
    .start()
)

```

```
(1) Spark Jobs
aggregated_data (id: 8701ec74-9b26-4615-9673-14097a8f7a67)    Last updated: 14 minutes ago
```

```
Command complete
```

- **Display the aggregated data.**



```
df = spark.sql("SELECT * FROM aggregated_data")
df.show(truncate=False)
```

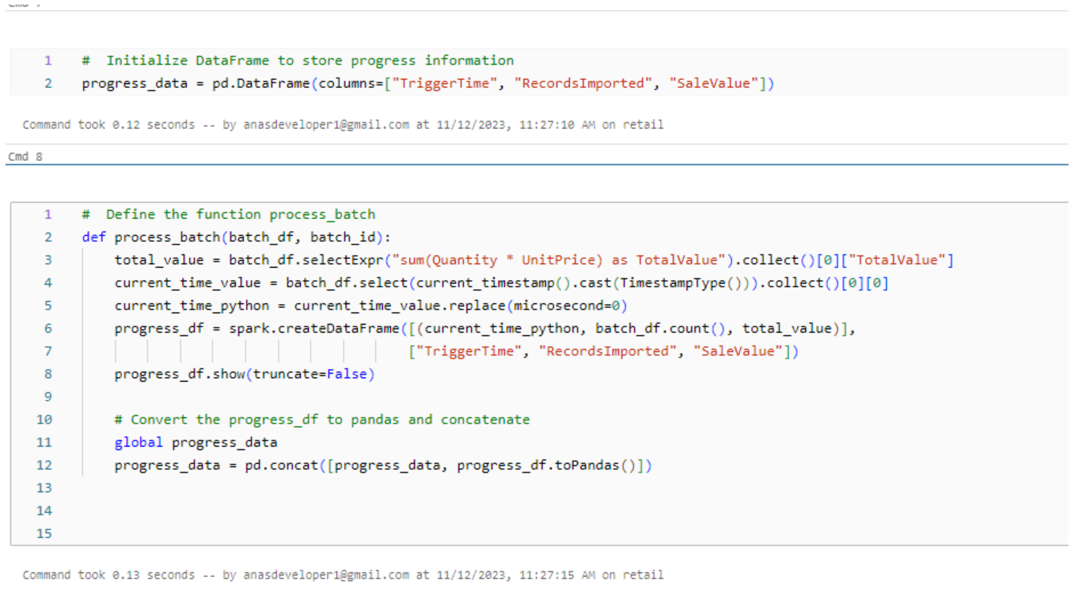
```
▸ (1) Spark Jobs
▸ df: pyspark.sql.dataframe.DataFrame = [CustomerID: integer, TotalStocks: long ... 1 more field]
```

```
+----------+-----------+------------------+
|CustomerID|TotalStocks|TotalValue        |
+----------+-----------+------------------+
|17420     |53         |130.85            |
|14570     |95         |218.05999999999992|
|17389     |2214       |9447.24           |
|15727     |1203       |2876.0500000000006|
|16861     |-7         |-22.110000000000003|
|16503     |443        |1082.8899999999999|
|13285     |914        |1173.2699999999998|
|15619     |136        |336.40000000000003|
|13623     |148        |532.0200000000001 |
|14450     |152        |269.45            |
|15790     |114        |220.84999999999997|
|15447     |85         |155.17            |
|16339     |23         |109.95000000000002|
|16386     |26         |61.10000000000001 |
|15100     |58         |635.0999999999999 |
|18161     |315        |540.4200000000001 |
|18282     |28         |77.84             |
|16500     |19         |217.74            |
```

```
Command took 6.69 seconds -- by anasdeveloper1@gmail.com at 11/12/2023, 11:24:37 AM on retail
```

c) **For each batch of the input stream, create a new stream that populates another dataframe or dataset with progress for each loaded set of data. This data set should have the columns – TriggerTime (Date/Time), Records Imported, Sale value (Total value of transactions).**

- **Start the Stream**



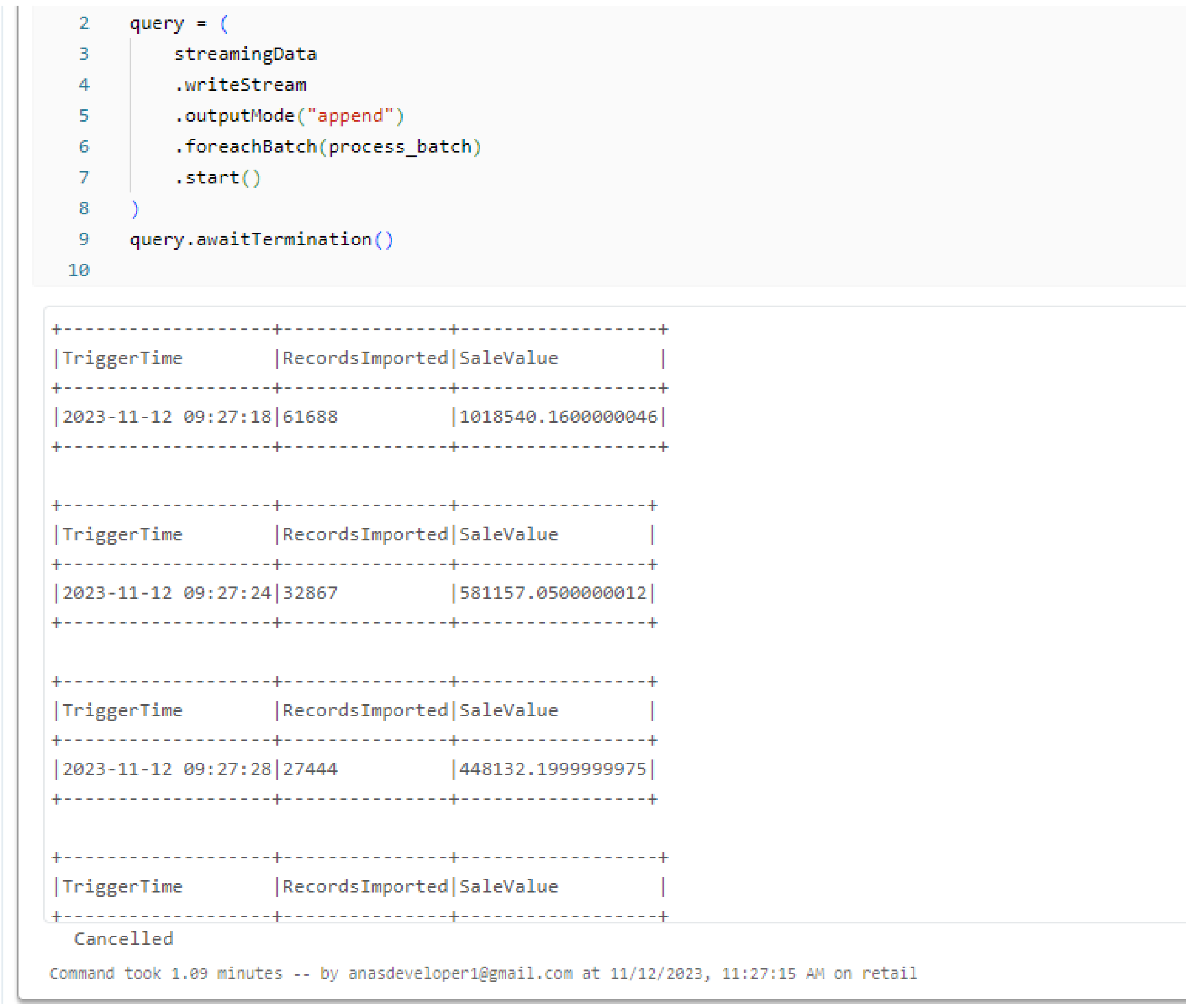
```
#  Initialize DataFrame to store progress information
progress_data = pd.DataFrame(columns=["TriggerTime", "RecordsImported", "SaleValue"])
```

```
Command took 0.12 seconds -- by anasdeveloper1@gmail.com at 11/12/2023, 11:27:10 AM on retail
```

```
Cmd 8
```

```
#  Define the function process_batch
def process_batch(batch_df, batch_id):
    total_value = batch_df.selectExpr("sum(Quantity * UnitPrice) as TotalValue").collect()[0]["TotalValue"]
    current_time_value = batch_df.select(current_timestamp().cast(TimestampType())).collect()[0][0]
    current_time_python = current_time_value.replace(microsecond=0)
    progress_df = spark.createDataFrame([(current_time_python, batch_df.count(), total_value)],
                                        ["TriggerTime", "RecordsImported", "SaleValue"])
    progress_df.show(truncate=False)

    # Convert the progress_df to pandas and concatenate
    global progress_data
    progress_data = pd.concat([progress_data, progress_df.toPandas()])



```

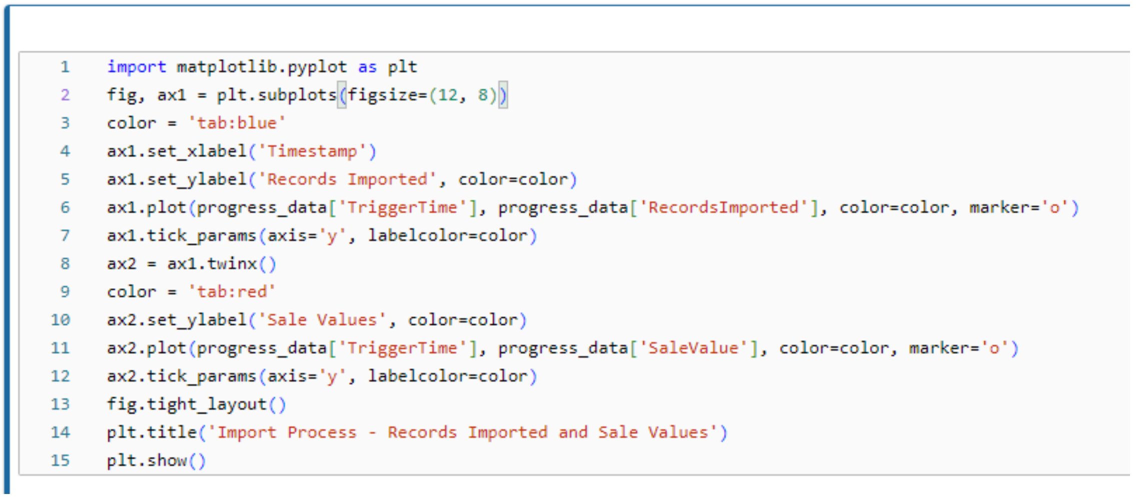
```
Command took 0.13 seconds -- by anasdeveloper1@gmail.com at 11/12/2023, 11:27:15 AM on retail
```



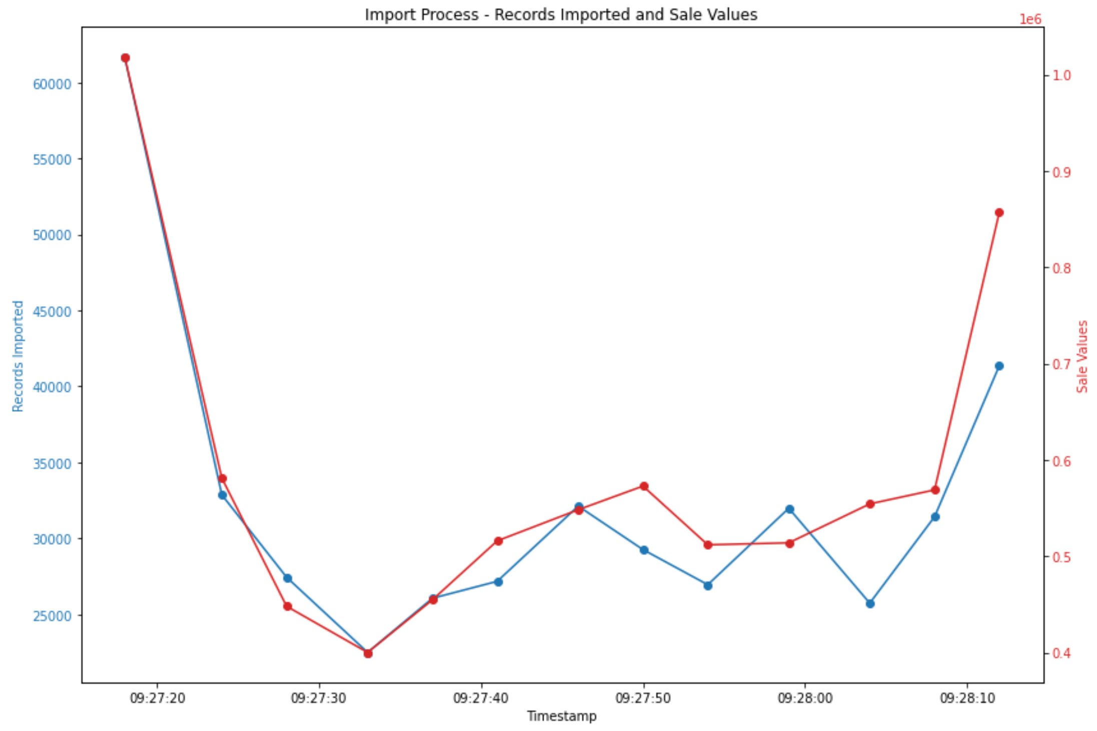
```
query = (
    streamingData
    .writeStream
    .outputMode("append")
    .foreachBatch(process_batch)
    .start()
)
query.awaitTermination()

```

```
+-------------------+--------------+-----------------+
|TriggerTime        |RecordsImported|SaleValue        |
+-------------------+--------------+-----------------+
|2023-11-12 09:27:18|61688         |1018540.1600000046|
+-------------------+--------------+-----------------+

+-------------------+--------------+----------------+
|TriggerTime        |RecordsImported|SaleValue       |
+-------------------+--------------+----------------+
|2023-11-12 09:27:24|32867         |581157.0500000012|
+-------------------+--------------+----------------+

+-------------------+--------------+----------------+
|TriggerTime        |RecordsImported|SaleValue       |
+-------------------+--------------+----------------+
|2023-11-12 09:27:28|27444         |448132.1999999975|
+-------------------+--------------+----------------+

+-------------------+--------------+-----------------+
|TriggerTime        |RecordsImported|SaleValue        |
+-------------------+--------------+-----------------+
  Cancelled
Command took 1.09 minutes -- by anasdeveloper1@gmail.com at 11/12/2023, 11:27:15 AM on retail
```

d) Use the dataset from step (c) to plot a line graph of the import process – showing two timelines – records imported and sale values.



```python
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots(figsize=(12, 8))
color = 'tab:blue'
ax1.set_xlabel('Timestamp')
ax1.set_ylabel('Records Imported', color=color)
ax1.plot(progress_data['TriggerTime'], progress_data['RecordsImported'], color=color, marker='o')
ax1.tick_params(axis='y', labelcolor=color)
ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Sale Values', color=color)
ax2.plot(progress_data['TriggerTime'], progress_data['SaleValue'], color=color, marker='o')
ax2.tick_params(axis='y', labelcolor=color)
fig.tight_layout()
plt.title('Import Process - Records Imported and Sale Values')
plt.show()
```



```
Command took 0.53 seconds -- by anasdeveloper1@gmail.com at 11/12/2023, 11:43:17 AM on retail
```

References:

[1] https://hadoop.apache.org
[2] https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
[3] https://medium.com/analytics-vidhya/spark-rdd-low-level-api-basics-using-pyspark-a9a322b58f6
[4] https://medium.com/analytics-vidhya/sparksql-and-dataframe-high-level-api-basics-using-pyspark-eaba6acf944b
[5] https://spark.apache.org/docs/latest/api/python/index.html