DETAIL-PROJECT REPORT

Detail project report

Sentiment Analysis

Table of Contents

1)Objective

Design scalable pipeline using spark to read customer review from s3 bucket and store it into HDFS. Schedule your pipeline to run iteratively after each hour.

Create a folder in the s3 bucket where customer reviews in Json format can be uploaded. The Scheduled big data pipeline will be triggered manually or automatically to read data from The S3 bucket and dump it into HDFS.

Use Spark Machine learning to perform sentiment analysis using customer review stores in HDFS.

2)Prerequisite

- **VsCode Installed**
- **Airflow Container**
- **AWS Account**
- **Docker Installed**

3) Project Introduction

In the age of digital commerce, understanding customer sentiment through their reviews is critical for businesses to make informed decisions and improve customer satisfaction. This project focuses on developing a scalable pipeline using Apache Spark to perform sentiment analysis on customer reviews fetched from an Amazon S3 bucket.

4) Key Components

- Amazon S3 Bucket: A repository for storing customer reviews in JSON format. A dedicated folder is created where new reviews can be uploaded by users or automated systems.
- Hadoop Distributed File System (HDFS): A distributed file system used for storing processed data. It provides fault tolerance and scalability, making it suitable for big data applications.
- Apache Spark: An open-source distributed computing system known for its high performance and scalability. Spark is utilized for processing large volumes of data in parallel and performing sentiment analysis on customer reviews.
- Sentiment Analysis: Utilizing Spark Machine Learning library or Spark NLP library to perform sentiment analysis on customer reviews. This involves analyzing the text content of reviews to determine whether they convey positive, negative, or neutral sentiment.
- Scheduler: Using a scheduling tool like Apache Airflow or Apache Oozie to automate the pipeline execution. The pipeline is scheduled to run iteratively after each hour to process new customer reviews as they become available.

5) Expected Outcomes

1) Real-time sentiment analysis of customer reviews.

   Insights into customer satisfaction and sentiment trends over time.

2) Automated processing and storage of large volumes of customer review data.
3) Scalable and fault-tolerant pipeline architecture capable of handling increasing data volumes.

6) Benefits

1) Enhanced understanding of customer sentiment and preferences.
2) Timely identification of issues and opportunities for improvement.
3) Improved decision-making for product development, marketing, and customer service.
4) Increased efficiency through automation and scalability of the pipeline.

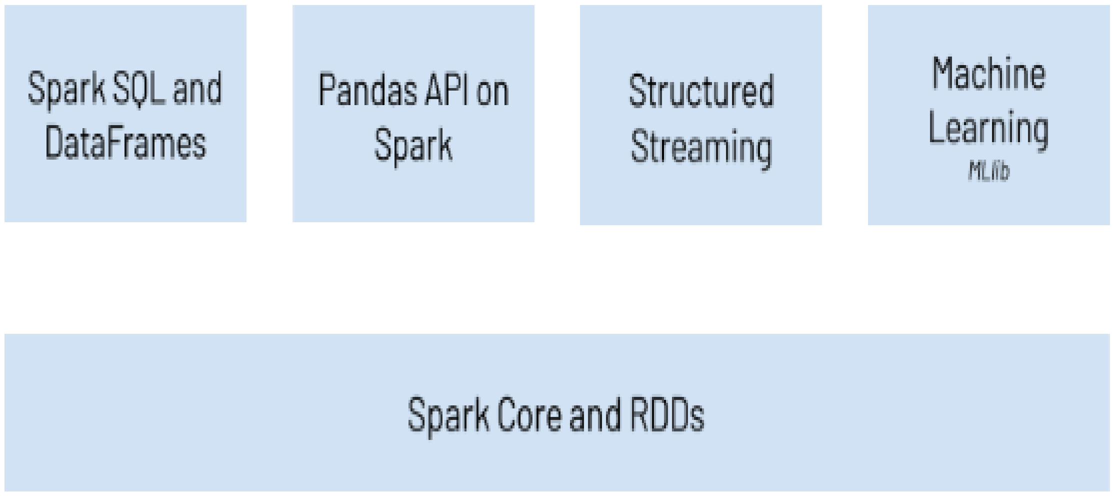
7) Specification

o PySpark

PySpark is the Python API for Apache Spark. It enables you to perform real-time, large-scale data processing in a distributed environment using Python. It also provides a PySpark shell for interactively analysing your data.

PySpark combines Python's learnability and ease of use with the power of Apache Spark to enable processing and analysis of data at any size for everyone familiar with Python.

PySpark supports all of Spark's features such as Spark SQL, Data Frames, Structured Streaming, Machine Learning (MLlib) and Spark Core



Spark SQL and DataFrames

Spark SQL is Apache Spark's module for working with structured data. It allows you to seamlessly mix SQL queries with Spark programs. With PySpark DataFrames you can

Whether you use Python or SQL, the same underlying execution engine is used so you will always leverage the full power of Spark.

Machine Learning (MLlib)

Built on top of Spark, MLlib is a scalable machine learning library that provides a uniform set of high-level APIs that help users create and tune practical machine learning pipelines.

Spark Core and RDDs

Spark Core is the underlying general execution engine for the Spark platform that all other functionality is built on top of. It provides RDDs (Resilient Distributed Datasets) and in-memory computing capabilities.

Note that the RDD API is a low-level API which can be difficult to use and you do not get the benefit of Spark's automatic query optimization capabilities. We recommend using DataFrames (see Spark SQL and DataFrames above) instead of RDDs as it allows you to express what you want more easily and lets Spark automatically construct the most efficient query for you.

○ Airflow

What is Airflow?
Apache Airflow is an open-source platform for developing, scheduling, and monitoring batch-oriented workflows. Airflow's extensible Python framework enables you to build workflows connecting with virtually any technology. A web interface helps manage the state of your workflows. Airflow is deployable in many ways, varying from a single process

Workflows as code

The main characteristic of Airflow workflows is that all workflows are defined in Python code. “Workflows as code” serves several purposes:

- Dynamic: Airflow pipelines are configured as Python code, allowing for dynamic pipeline generation.
- Extensible: The Airflow framework contains operators to connect with numerous technologies. All Airflow components are extensible to easily adjust to your environment.

- ## Amazon S3

What is Amazon S3?

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements.

How Amazon S3 works

Amazon S3 is an object storage service that stores data as objects within buckets. An object is a file and any metadata that describes the file. A bucket is a container for objects. To store your data in Amazon S3, you first create a bucket and specify a bucket name and AWS Region. Then, you upload

your data to that bucket as objects in Amazon S3. Each object has a key (or key name), which is the unique identifier for the object within the bucket.

S3 provides features that you can configure to support your specific use case. For example, you can use S3 Versioning to keep multiple versions of an object in the same bucket, which allows you to restore objects that are accidentally deleted or overwritten.

Buckets and the objects in them are private and can be accessed only if you explicitly grant access permissions. You can use bucket policies, AWS Identity and Access Management (IAM) policies, access control lists (ACLs), and S3 Access Points to manage access.

- **Amazon EMR**

What is Amazon EMR?

Amazon EMR (previously called Amazon Elastic MapReduce) is a managed cluster platform that simplifies running big data frameworks, such as Apache Hadoop and Apache Spark, on AWS to process and analyze vast amounts of data. Using these frameworks and related open-source projects, you can process data for analytics purposes and business intelligence workloads. Amazon EMR also lets you transform and move large amounts of data into and out of other AWS data stores and databases, such as Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB.

Understanding clusters and nodes

The central component of Amazon EMR is the cluster. A cluster is a collection of Amazon Elastic Compute Cloud (Amazon EC2) instances. Each instance in the cluster is called a node. Each node has a role within the cluster, referred to as the node type. Amazon EMR also installs different software components on each node type, giving each node a role in a distributed application like Apache Hadoop.

The node types in Amazon EMR are as follows:

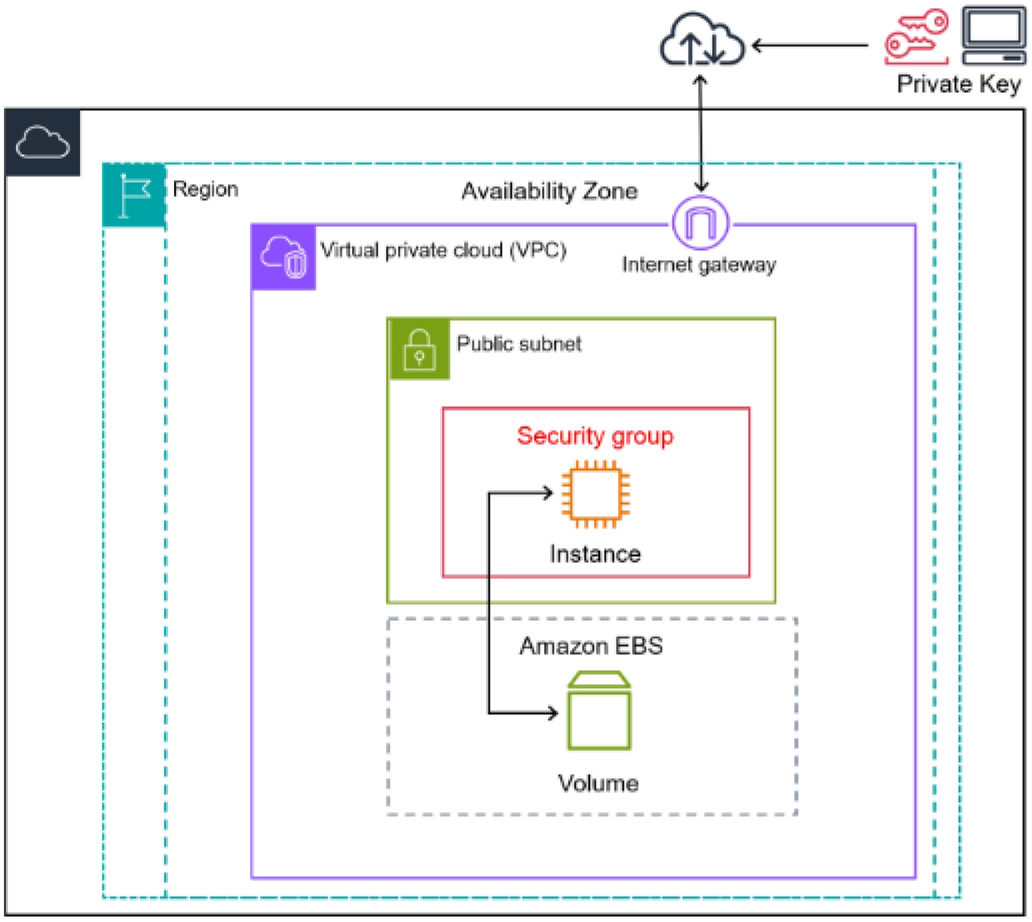
- Primary node: A node that manages the cluster by running software components to coordinate the distribution of data and tasks among other nodes for processing. The primary node tracks the status of tasks and monitors the health of the cluster. Every cluster has a primary node, and it's possible to create a single-node cluster with only the primary node.
- Core node: A node with software components that run tasks and store data in the Hadoop Distributed File System (HDFS) on your cluster. Multi-node clusters have at least one core node.
- Task node: A node with software components that only runs tasks and does not store data in HDFS. Task nodes are optional.

- ## Amazon EC2

What is Amazon EC2?

Amazon Elastic Compute Cloud (Amazon EC2) provides on-demand, scalable computing capacity in the Amazon Web Services (AWS) Cloud. Using Amazon EC2 reduces hardware costs so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. You can add capacity (scale up) to handle compute-heavy tasks, such as monthly or yearly processes, or spikes in website traffic. When usage decreases, you can reduce capacity (scale down) again.

The following diagram shows a basic architecture of an Amazon EC2 instance deployed within an Amazon Virtual Private Cloud (VPC). In this example, the EC2 instance is within an Availability Zone in the Region. The EC2 instance is secured with a security group, which is a virtual firewall that controls incoming and outgoing traffic. A private key is stored on the local computer and a public key is stored on the instance. Both keys are specified as a key pair to prove the identity of the user. In this scenario, the instance is backed by an Amazon EBS volume. The VPC communicates with the internet using an internet gateway. For more information about Amazon VPC



○ Hadoop HDFS

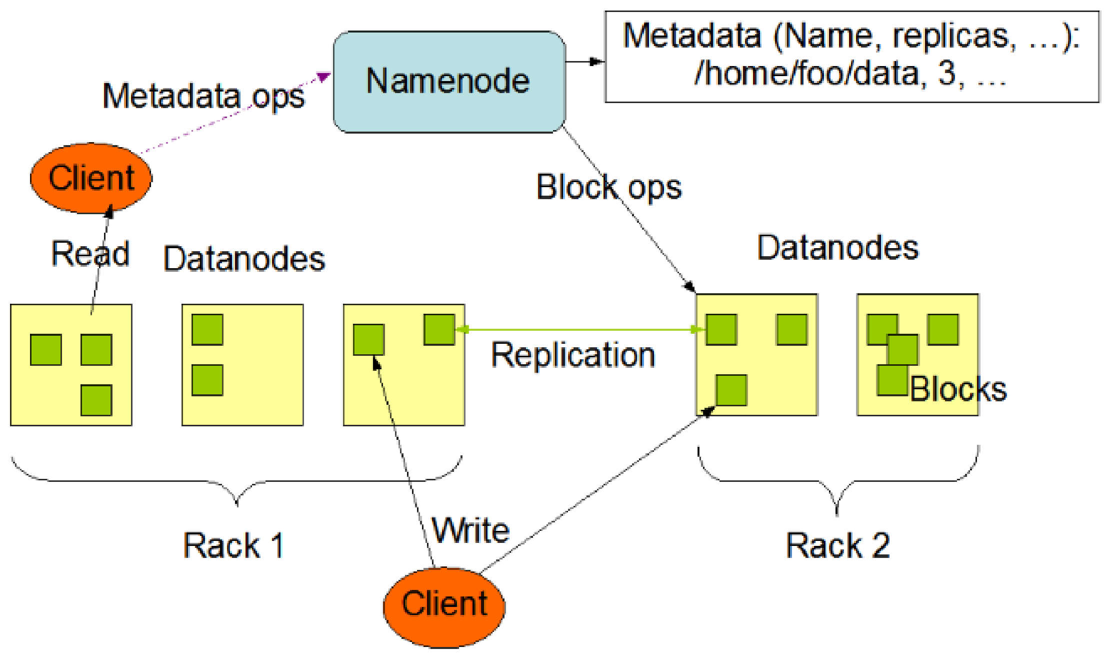
The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject

NameNode and DataNodes

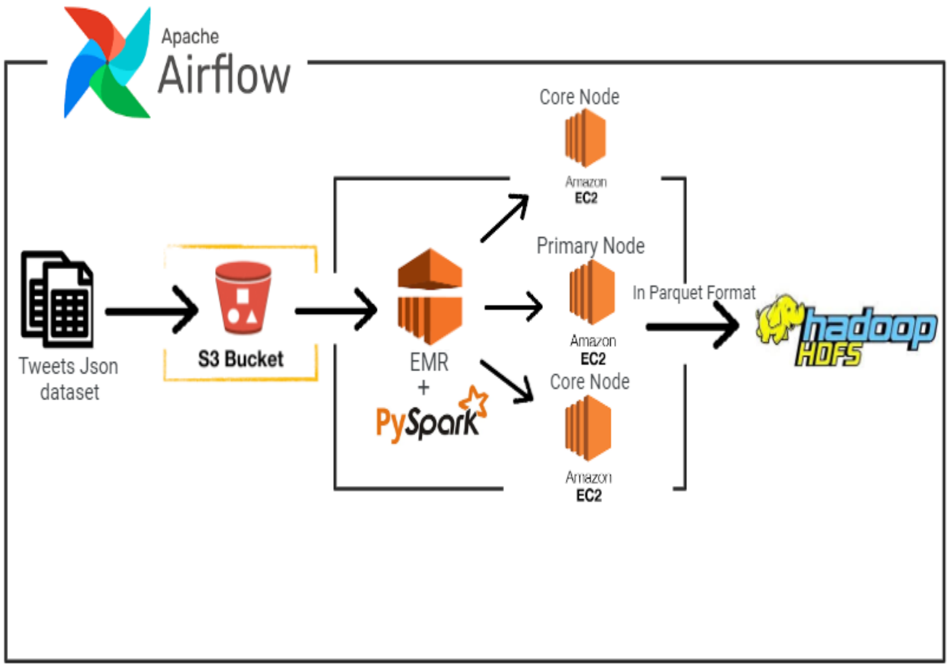
HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

HDFS Architecture

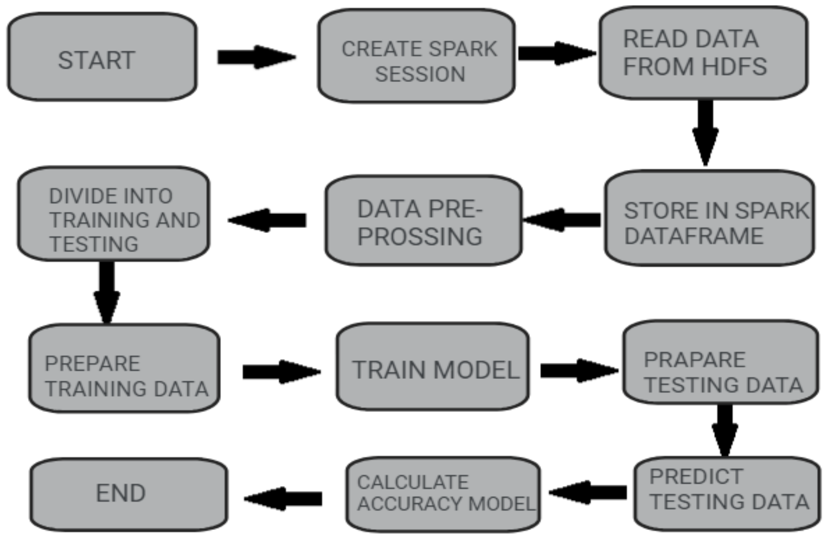


8. Architecture

8.1 Scalable pipeline using spark to read customer review from s3 bucket and store it into HDFS



8.2 Sentiment analysis Model



9. Architecture Description

9.1 Pipeline components:

9.1.1 Data Source:

S3 Bucket: Set up an Amazon S3 bucket to store customer reviews in JSON format. Create a dedicated folder within the bucket to organize the data. New reviews can be uploaded to this folder periodically.

9.1.2 Spark Cluster:

Master Node: The master node manages the distribution of tasks across worker nodes. It coordinates the execution of the Spark application.

Worker Nodes: Worker nodes are responsible for executing tasks assigned by the master node. They process the data in parallel, enabling efficient distributed computing.

9.1.3 Scheduler:

Apache Airflow : Use a workflow management platform like Apache Airflow to schedule and or-chestrate the pipeline. Define tasks and dependencies to ensure the pipeline runs smoothly and efficiently.

Schedule the pipeline to run iteratively after each hour to process new customer reviews.

Cron Jobs: Alternatively, you can use cron jobs on the server hosting the Spark cluster to trigger the pipeline at specified intervals.

9.1.4 Data Storage:

HDFS: Hadoop Distributed File System (HDFS) is used for storing the processed data. Write the original customer reviews, to HDFS.

Parquet Format: Store the data in Parquet format, which is efficient for columnar storage and supports compression. Partition the data based on relevant criteria such as date or product category for optimized querying.

9.1.5 Data Processing:

Spark Application: Develop a PySpark application using Python to read data from the S3 bucket and to store in HDFS.

Preprocessing: Preprocess the raw text data to clean and tokenize it. This may involve removing punctuation, converting text to lowercase, and tokenizing sentences into words.

9.1.6 Sentiment Analysis:

Spark MLlib : Utilize Spark's machine learning libraries such as MLlib for sentiment analysis. Train a sentiment analysis model on labelled data or use pre-trained models available in Spark NLP.

9.2 Pipeline Workflow:

9.2.1 Data ingestion to S3:

Create a folder in S3 bucket where Json data can be stored .The Json data contained the Customer Review datasets. These datasets need to be transformed in parquet file for fast Processing .

9.2.2 Create a cluster:

Create a Cluster using Amazon EMR (Elastic MapReduce) . Using EMR we can create a Hadoop cluster . In Hadoop cluster we need primary node and secondary node for that we use Amazon EC2 instance. We use a EC2 instance for creating the primary node and secondary node. Attach the EMR cluster with EC2 instance . We have to create VPC for EC2 There we need to set the security groups , subnets and route tables. Use pyspark to read the Customer Review Json data and store in HDFS in parquet file format. In addsteps, we need to add the pyspark file to run the program or we can create the Scheduler using airflow which will schedule your pipeline to run iteratively after each hour.

9.2.3 Create a Scheduler:

Create a scheduler using Apache airflow to run iteratively after each hour . For that we can use Amazon Managed airflow or we can use container image from Docker .We need to install the Docker to run the docker image . For not getting charged from Amazon Managed airflow we can install Vscode and by using compose.yml file we can create an airflow image inside the Vscode. For Scheduling the pipeline, we have to create a python file .In python file, we have to create a DAG(Directed Acyclic Graph). And in Dag we need to Specify the EMR configuration and the scheduling time we have to set the aws_default config. Using airflow UI, we can trigger the Dag.

9.2.4 Data storage in HDFS:

The Json Data which is stored in S3 bucket after triggering the airflow Dag ,the data transformed in

parquet file and store in HDFS for fast processing .
Using these parquet data we need to perform Sentiment Analysis.

9.2.5 Sentiment Analysis using spark ML

Spark MLlib : Utilize Spark's machine learning libraries such as MLlib for sentiment analysis. Train a sentiment analysis model on labelled data . Use Logistic Regression for modelling the tweets for Sentiment Analysis

10. Code:

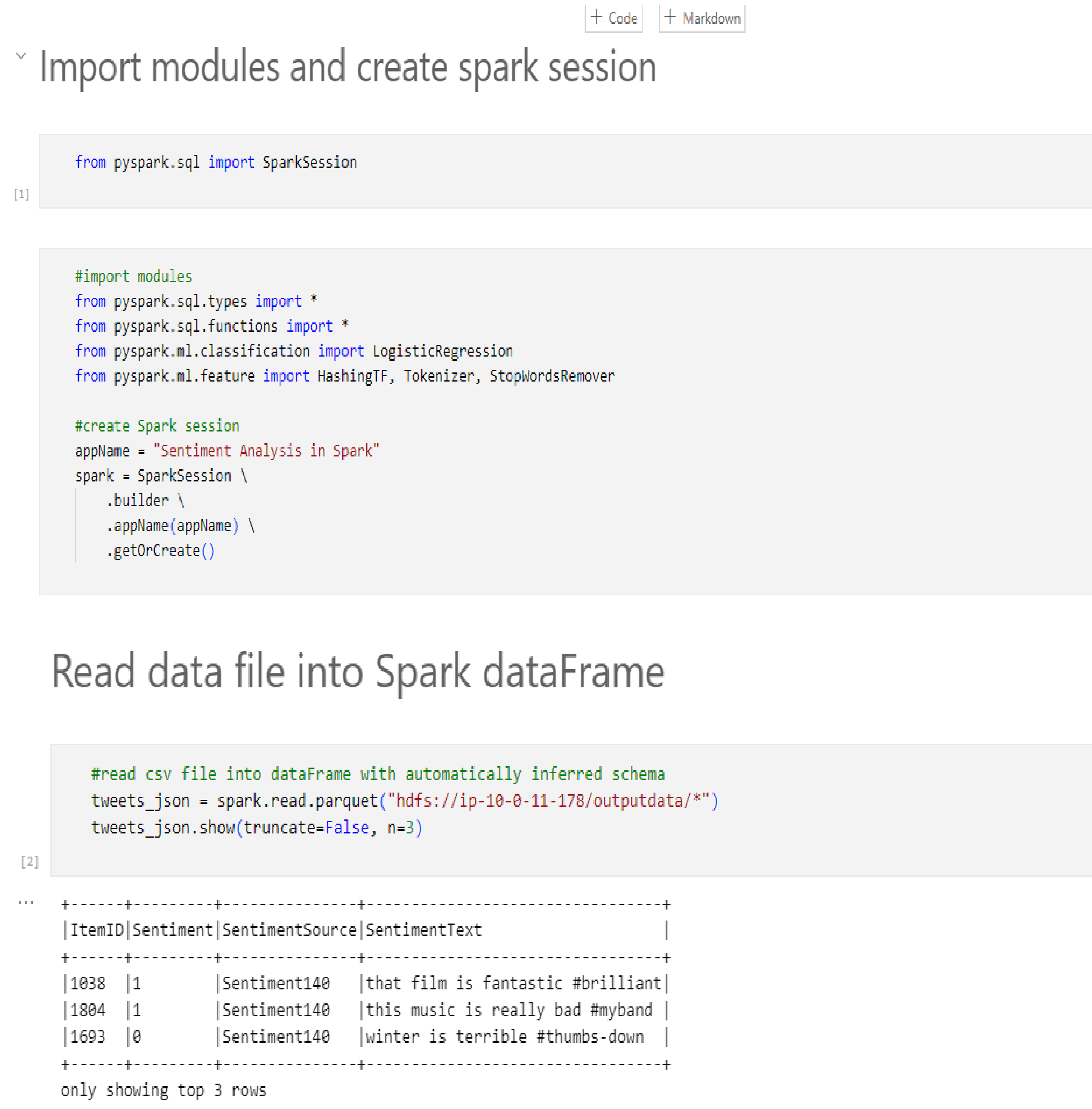
10.1 Pyspark code to read Customer Review from S3 bucket and store in HDFS.

```
# First Initialize The SparkSession
from pyspark.sql import SparkSession

# Then import the required module
from pyspark.sql.types import StructType, StructField, StringType,IntegerType

# Start The SparkSession And create The Application
spark=SparkSession.builder.appName("CustomerReviewSentimentAnalysis").getOrCreate()

# Set The Schema For The Json Data
schema = StructType([
    StructField("ItemID", IntegerType()),
    StructField("Sentiment", IntegerType()),
    StructField("SentimentSource", StringType()),
    StructField("SentimentText", StringType())
])

# Create The CustomerRevivedf And Upload The Json Data From The S3 Bucket
CustomerRevivedf = spark.read.schema(schema).option("mode", "PERMISSIVE").json("s3://myprojectemr/myinputfolder/tweets.json")

# Print The Schema Of CustomerRevivedf
CustomerRevivedf.printSchema()

# Show the CustomerRevivedf DataFrame
CustomerRevivedf.show()

# Write The CustomerRevivedf Into Parquet Format In HDFS File
CustomerRevivedf.write.mode("overwrite").parquet("hdfs://ip-10-0-11-178/outputdata/")

# At last Stop The Spark Job
spark.stop()
```

10.2 Code for creating a DAG

```
from airflow import DAG
from airflow.contrib.operators.emr_add_steps_operator import EmrAddStepsOperator
from airflow.contrib.sensors.emr_step_sensor import EmrStepSensor
from datetime import datetime, timedelta

# Define your DAG
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 3, 5),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    's3_to_hdfs_on_emr',
    default_args=default_args,
    description='Read JSON data from S3 and write it to HDFS on existing EMR cluster'
,
    schedule_interval='@hourly',
)

# Define your EMR steps
emr_steps = [
    {
        'Name': 'Read JSON from S3 and write to HDFS',
        'ActionOnFailure': 'CONTINUE',
        'HadoopJarStep': {
            'Jar': 'command-runner.jar',
            'Args': [
                'spark-submit',
                '--deploy-mode', 'cluster',
                's3://myprojectemr/sparkscript/s3_to_hdfs.py',  # Adjust to your Spar
k application
                's3://myprojectemr/myinputfolder/tweets.json',  # Adjust to your S3 i
nput path
                'hdfs://ip-10-0-11-178/outputdata/'  # Adjust to your HDFS output pat
h
            ]
        }
    }
]

# Add steps to the existing EMR cluster
add_step_task = EmrAddStepsOperator(
    task_id='add_step_to_emr_cluster',
    job_flow_id='j-E65C9OZZ8BJF',  # Adjust to your EMR cluster ID
    aws_conn_id='aws_default',  # Your AWS connection ID
    steps=emr_steps,
    dag=dag,
)

# Define sensor to wait for the EMR step to complete
step_sensor_task = EmrStepSensor(
    task_id='watch_step',
    job_flow_id='j-E65C9OZZ8BJF',  # Adjust to your EMR cluster ID
    step_id="{{ task_instance.xcom_pull(task_ids='add_step_to_emr_cluster', key='retu
rn_value')[0] }}",
    aws_conn_id='aws_default',  # Your AWS connection ID
    dag=dag,
)
add_step_task >> step_sensor
```

10.3 Code for sentiment analysis using spark ML



Import modules and create spark session

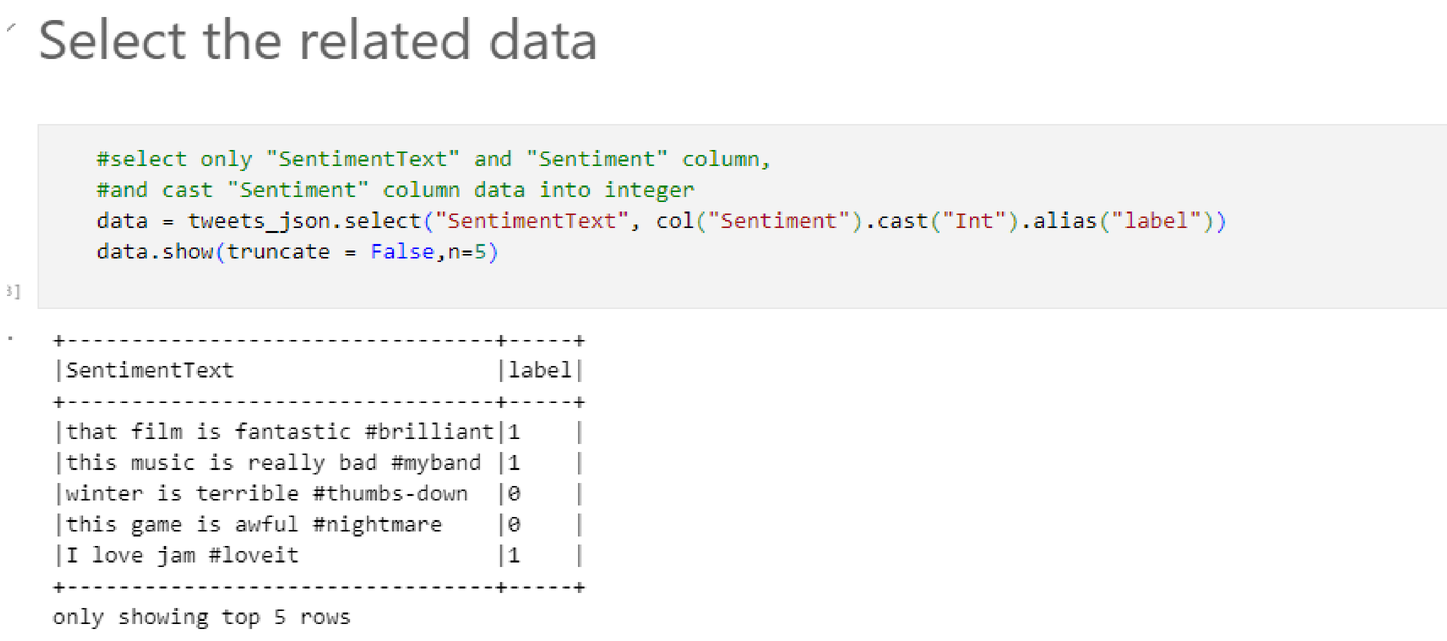
```
from pyspark.sql import SparkSession
```

```
#import modules
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer, StopWordsRemover

#create Spark session
appName = "Sentiment Analysis in Spark"
spark = SparkSession \
    .builder \
    .appName(appName) \
    .getOrCreate()
```

Read data file into Spark dataFrame

```
#read csv file into dataFrame with automatically inferred schema
tweets_json = spark.read.parquet("hdfs://ip-10-0-11-178/outputdata/*")
tweets_json.show(truncate=False, n=3)
```

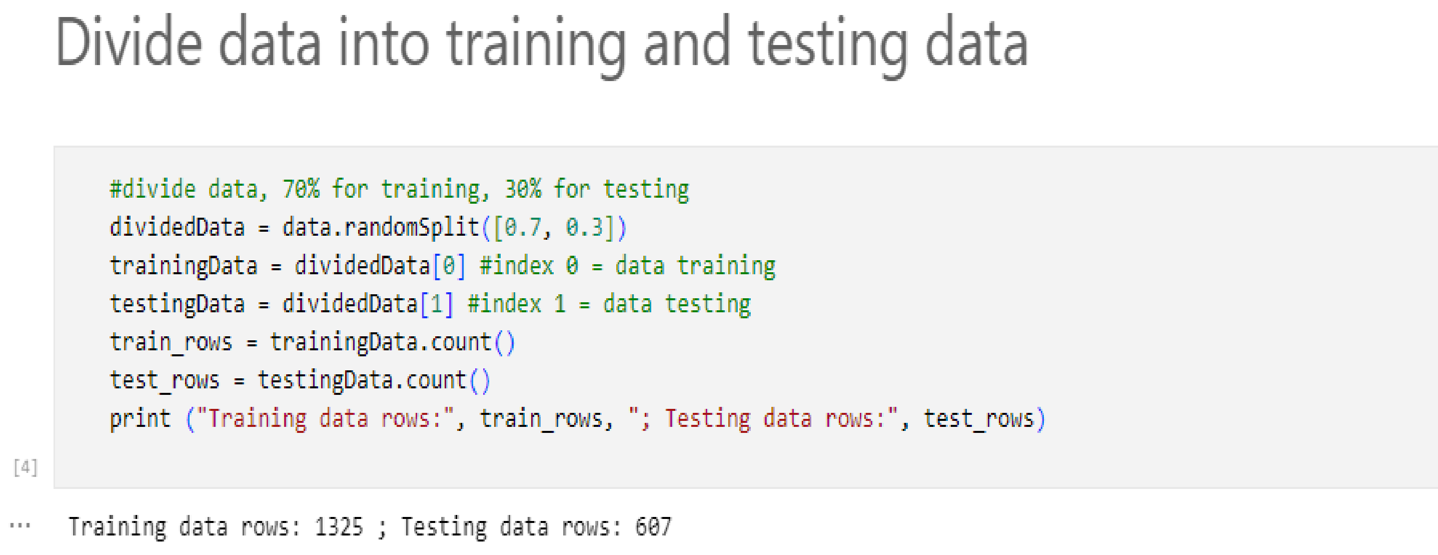
```
+------+---------+---------------+--------------------------------+
|ItemID|Sentiment|SentimentSource|SentimentText                   |
+------+---------+---------------+--------------------------------+
|1038  |1        |Sentiment140   |that film is fantastic #brilliant|
|1804  |1        |Sentiment140   |this music is really bad #myband |
|1693  |0        |Sentiment140   |winter is terrible #thumbs-down  |
+------+---------+---------------+--------------------------------+
only showing top 3 rows
```



Select the related data

```
#select only "SentimentText" and "Sentiment" column,
#and cast "Sentiment" column data into integer
data = tweets_json.select("SentimentText", col("Sentiment").cast("Int").alias("label"))
data.show(truncate = False,n=5)
```

```
+---------------------------------+-----+
|SentimentText                    |label|
+---------------------------------+-----+
|that film is fantastic #brilliant|1    |
|this music is really bad #myband |1    |
|winter is terrible #thumbs-down  |0    |
|this game is awful #nightmare    |0    |
|I love jam #loveit               |1    |
+---------------------------------+-----+
only showing top 5 rows
```



Divide data into training and testing data

```
#divide data, 70% for training, 30% for testing
dividedData = data.randomSplit([0.7, 0.3])
trainingData = dividedData[0] #index 0 = data training
testingData = dividedData[1] #index 1 = data testing
train_rows = trainingData.count()
test_rows = testingData.count()
print ("Training data rows:", train_rows, "; Testing data rows:", test_rows)
```

```
Training data rows: 1325 ; Testing data rows: 607
```

Prepare training data

Separate "SentimentText" into individual words using tokenizer

```
tokenizer = Tokenizer(inputCol="SentimentText", outputCol="SentimentWords")
tokenizedTrain = tokenizer.transform(trainingData)
tokenizedTrain.show(truncate=False, n=5)
```

[5]

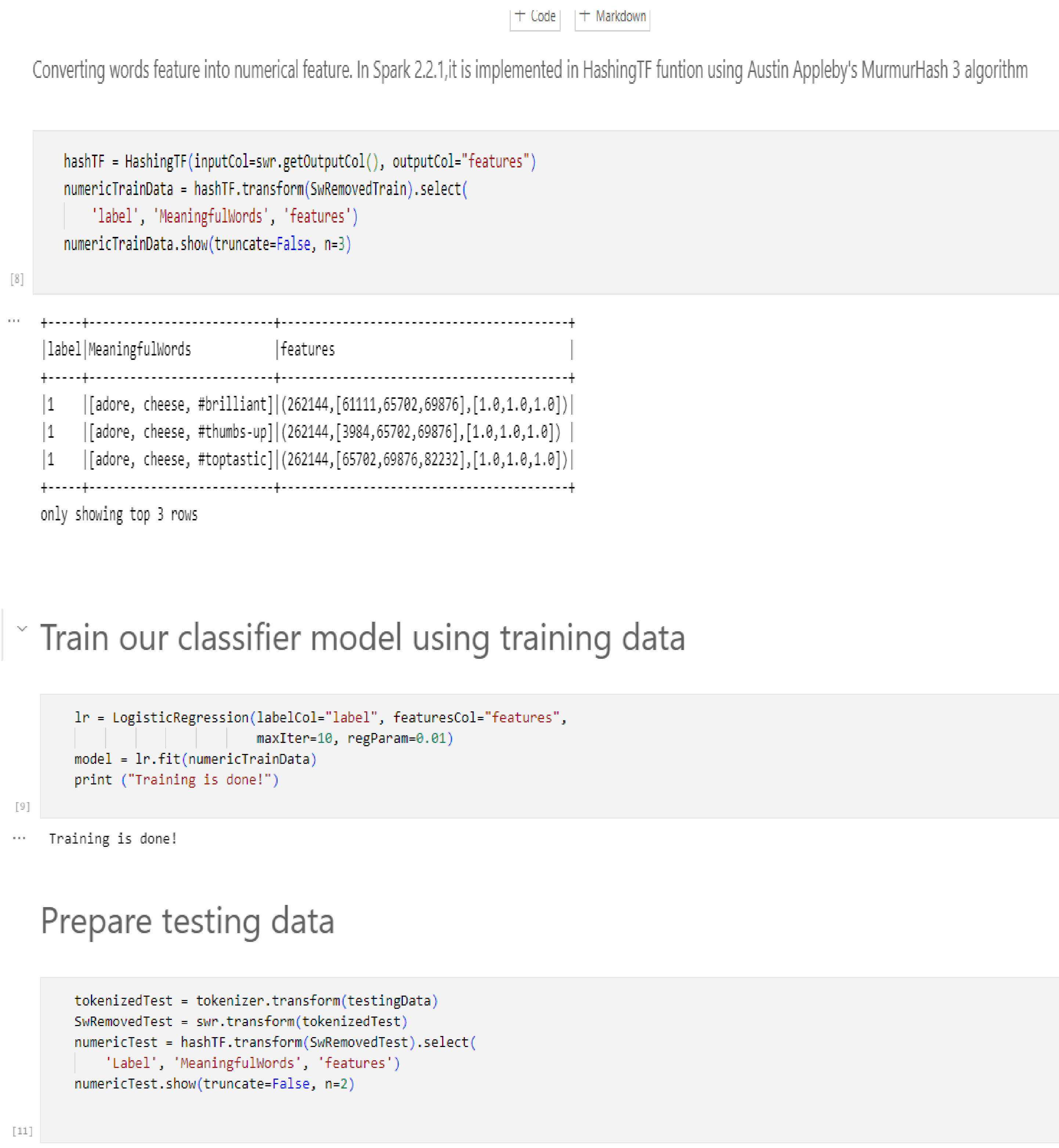
```
+--------------------------------+-----+---------------------------------------+
|SentimentText                   |label|SentimentWords                         |
+--------------------------------+-----+---------------------------------------+
|I adore cheese #brilliant       |1    |[i, adore, cheese, #brilliant]         |
|I adore cheese #thumbs-up       |1    |[i, adore, cheese, #thumbs-up]         |
|I adore cheese #toptastic       |1    |[i, adore, cheese, #toptastic]         |
|I adore classical music #brilliant|1  |[i, adore, classical, music, #brilliant]|
|I adore classical music #favorite |1  |[i, adore, classical, music, #favorite] |
+--------------------------------+-----+---------------------------------------+
only showing top 5 rows
```

Removing stop words (unimportant words to be features)

```
swr = StopWordsRemover(inputCol=tokenizer.getOutputCol(),
                       outputCol="MeaningfulWords")
SwRemovedTrain = swr.transform(tokenizedTrain)
SwRemovedTrain.show(truncate=False, n=5)
```

[7]

```
+--------------------------------+-----+----------------------------------------+-----------------------------------+
|SentimentText                   |label|SentimentWords                          |MeaningfulWords                    |
+--------------------------------+-----+----------------------------------------+-----------------------------------+
|I adore cheese #brilliant       |1    |[i, adore, cheese, #brilliant]          |[adore, cheese, #brilliant]        |
|I adore cheese #thumbs-up       |1    |[i, adore, cheese, #thumbs-up]          |[adore, cheese, #thumbs-up]        |
|I adore cheese #toptastic       |1    |[i, adore, cheese, #toptastic]          |[adore, cheese, #toptastic]        |
|I adore classical music #brilliant|1  |[i, adore, classical, music, #brilliant]|[adore, classical, music, #brilliant]|
|I adore classical music #favorite |1  |[i, adore, classical, music, #favorite] |[adore, classical, music, #favorite] |
+--------------------------------+-----+----------------------------------------+-----------------------------------+
only showing top 5 rows
```



Converting words feature into numerical feature. In Spark 2.2.1,it is implemented in HashingTF funtion using Austin Appleby's MurmurHash 3 algorithm

```
hashTF = HashingTF(inputCol=swr.getOutputCol(), outputCol="features")
numericTrainData = hashTF.transform(SwRemovedTrain).select(
    'label', 'MeaningfulWords', 'features')
numericTrainData.show(truncate=False, n=3)
```

[8]

```
+-----+--------------------------+---------------------------------------+
|label|MeaningfulWords           |features                               |
+-----+--------------------------+---------------------------------------+
|1    |[adore, cheese, #brilliant]|(262144,[61111,65702,69876],[1.0,1.0,1.0])|
|1    |[adore, cheese, #thumbs-up]|(262144,[3984,65702,69876],[1.0,1.0,1.0]) |
|1    |[adore, cheese, #toptastic]|(262144,[65702,69876,82232],[1.0,1.0,1.0])|
+-----+--------------------------+---------------------------------------+
only showing top 3 rows
```

Train our classifier model using training data

```
lr = LogisticRegression(labelCol="label", featuresCol="features",
                        maxIter=10, regParam=0.01)
model = lr.fit(numericTrainData)
print ("Training is done!")
```

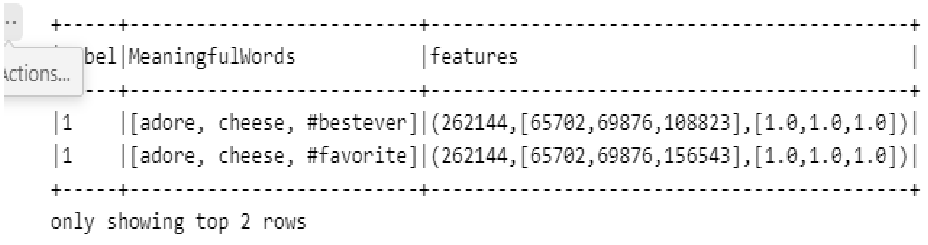
[9]

```
Training is done!
```

Prepare testing data

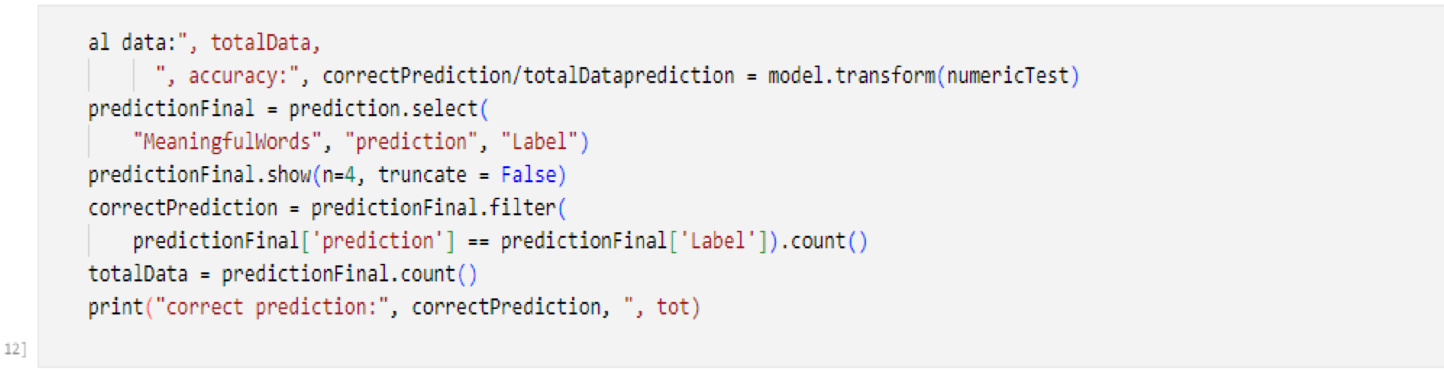
```
tokenizedTest = tokenizer.transform(testingData)
SwRemovedTest = swr.transform(tokenizedTest)
numericTest = hashTF.transform(SwRemovedTest).select(
    'Label', 'MeaningfulWords', 'features')
numericTest.show(truncate=False, n=2)
```

[11]

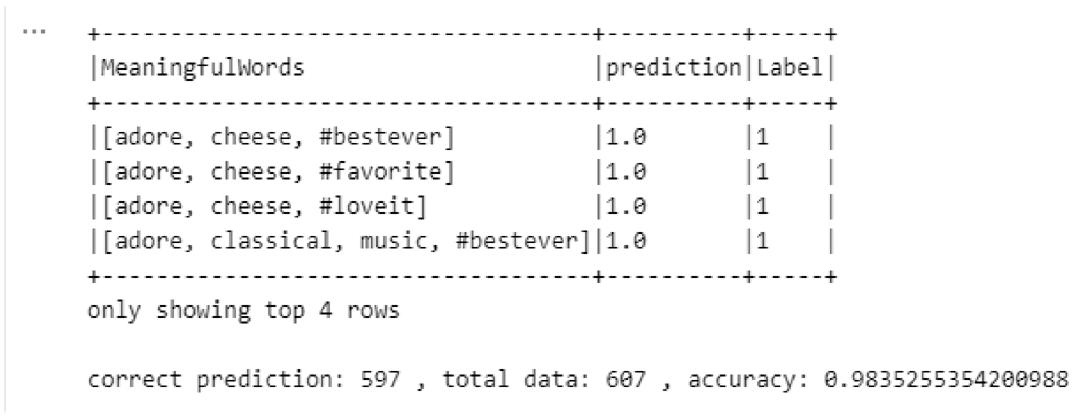


```
+-----+------------------------+----------------------------------------+
   bel|MeaningfulWords         |features                                |
---+-----------------------+----------------------------------------+
|1    |[adore, cheese, #bestever]|(262144,[65702,69876,108823],[1.0,1.0,1.0])|
|1    |[adore, cheese, #favorite]|(262144,[65702,69876,156543],[1.0,1.0,1.0])|
+-----+------------------------+----------------------------------------+
only showing top 2 rows
```

Predict testing data and calculate the accuracy model



```
al data:", totalData,
      ", accuracy:", correctPrediction/totalDataprediction = model.transform(numericTest)
predictionFinal = prediction.select(
    "MeaningfulWords", "prediction", "Label")
predictionFinal.show(n=4, truncate = False)
correctPrediction = predictionFinal.filter(
    predictionFinal['prediction'] == predictionFinal['Label']).count()
totalData = predictionFinal.count()
print("correct prediction:", correctPrediction, ", tot)
```



```
+-----------------------------------+----------+-----+
|MeaningfulWords                    |prediction|Label|
+-----------------------------------+----------+-----+
|[adore, cheese, #bestever]         |1.0       |1    |
|[adore, cheese, #favorite]         |1.0       |1    |
|[adore, cheese, #loveit]           |1.0       |1    |
|[adore, classical, music, #bestever]|1.0      |1    |
+-----------------------------------+----------+-----+
only showing top 4 rows

correct prediction: 597 , total data: 607 , accuracy: 0.9835255354200988
```

11. Conclusion:

In conclusion, the proposed project of designing a scalable pipeline using Spark for sentiment analysis on customer reviews stored in an S3 bucket and storing the analyzed data into HDFS offers significant benefits for businesses seeking to gain insights from customer feedback. By implementing this project, organizations can:

- **Efficiently Process and Analyze Customer Feedback**: The scalable pipeline leverages Apache Spark's distributed computing capabilities to efficiently process and analyze large volumes of customer review data.
- **Derive Actionable Insights**: Through sentiment analysis, businesses can gain valuable insights into customer sentiments, allowing them to identify trends, patterns, and areas for improvement.
- **Enhance Decision-Making**: Armed with insights from sentiment analysis, organizations can make data-driven decisions to improve products, services, and overall customer experience.
- **Enable Real-Time Analysis**: The iterative scheduling of the pipeline ensures that customer feedback is analyzed in near real-time, enabling timely responses to emerging trends or issues.
- **Ensure Scalability and Performance**: The use of Spark and HDFS ensures scalability and high performance, allowing the pipeline to handle growing datasets efficiently.