Introduction

This report focuses on the comprehensive analysis of a large Twitter dataset, containing 1.6 million rows, to gain insights into online sentiment and time series trends. The analysis leverages various data processing techniques, including data storage on the Hadoop Distributed File System (HDFS), data transformation using Apache Spark, sentiment analysis, and time series analysis.

The primary objective is to systematically process and analyze the vast Twitter dataset, with a focus on sentiment analysis to extract emotional context from the text data. Sentiment analysis aids in discerning the prevailing sentiments and opinions within the dataset.

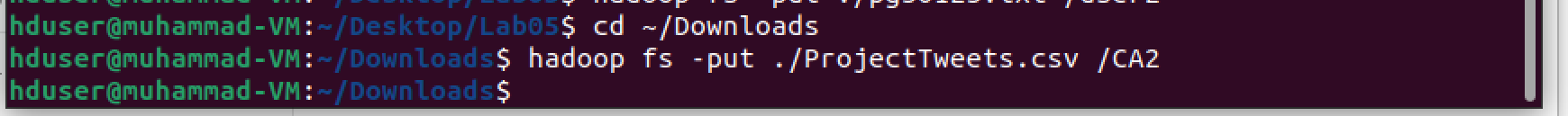
Furthermore, time series analysis is employed to understand the temporal evolution of trends and patterns within the Twitter data. This involves the identification of seasonality, trends, and any anomalies in the data to provide a deeper understanding of the dynamics of Twitter conversations over time.

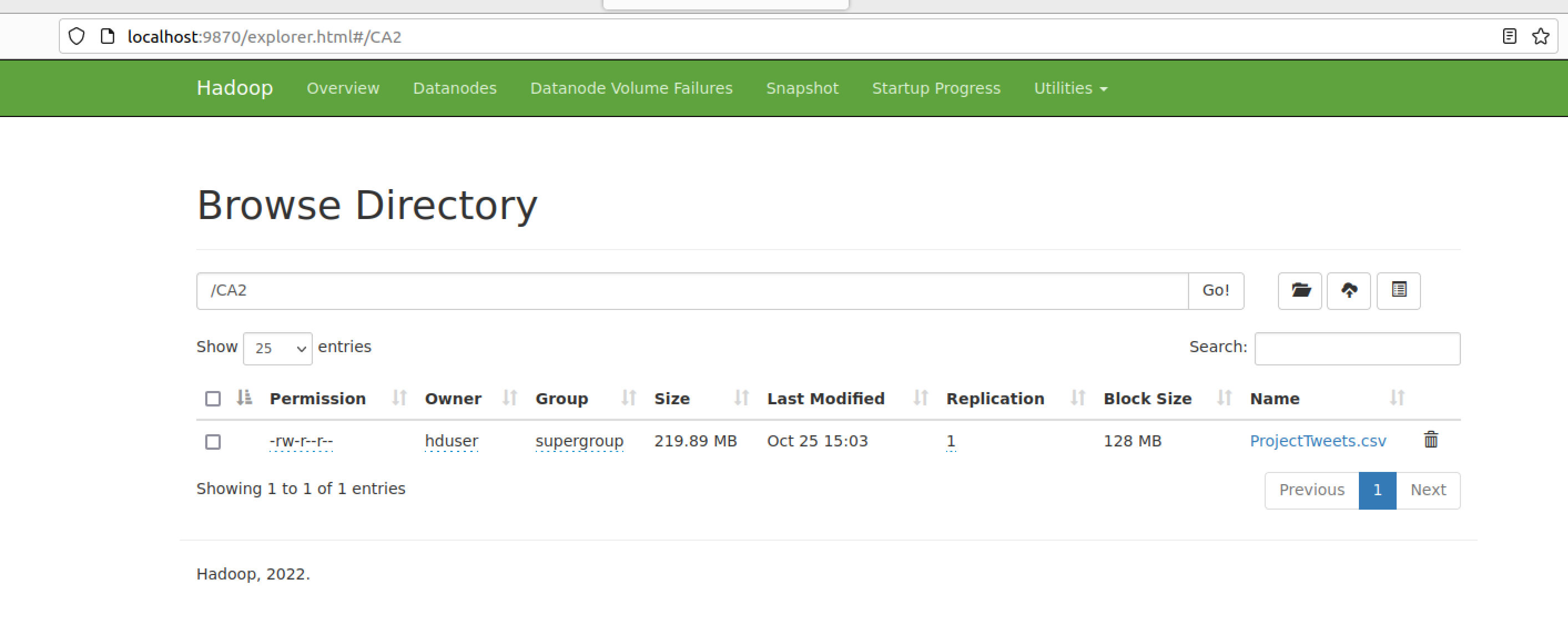
The report outlines the entire data processing pipeline, from data loading and transformation to sentiment and time series analysis. The approach described herein underscores the importance of a multifaceted analysis to derive meaningful insights from the Twitter dataset, with potential applications in understanding online discourse, tracking sentiment shifts, and identifying influential factors driving conversations on social media.

Initial Dataset Storage and Assessment

The first step was to place the Twitter dataset into the Hadoop Distributed File System (HDFS). To organize the data efficiently and maintain a structured approach, we created a dedicated directory within the HDFS, which we named ‘CA2’. This designated folder served as the repository for our dataset, ensuring that the data is stored in a centralised and organised manner, easily accessible for subsequent processing and analysis.

By storing the dataset in HDFS, we capitalised on the distributed and fault-tolerant nature of the file system, which is well-suited for managing substantial data loads. Additionally, this approach allowed us to harness the parallel processing capabilities of Hadoop for subsequent data transformations and analyses.



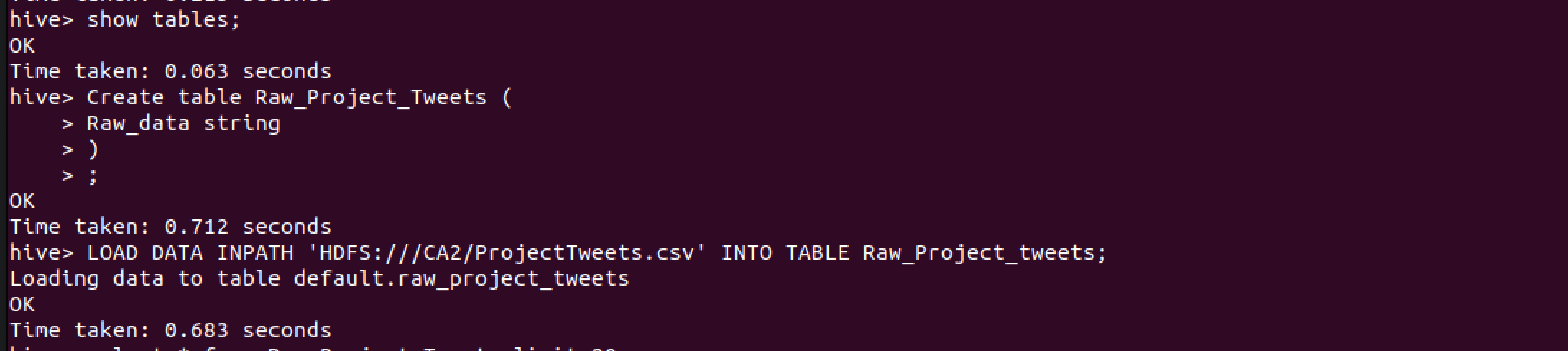


I loaded the data into Hive and also brought the data into Spark for processing using the Pyspark library.

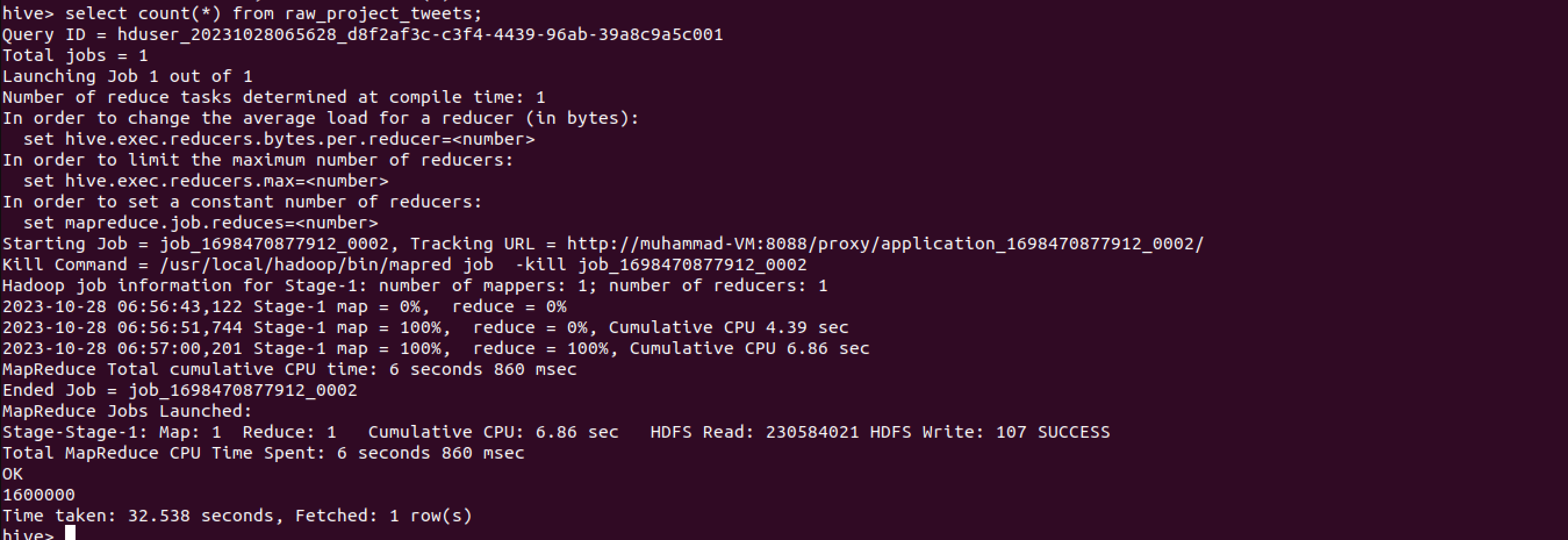
***Hive***

Note: I intentionally brought the data into a single column table in hive because even though the data is csv, I’m aware that there is a free text column that probably contains commas and I don’t want the data to be parsed or corrupted before taking a look at it. It’s important to make no assumptions about the contents of the data before first taking a look at it in its raw state.

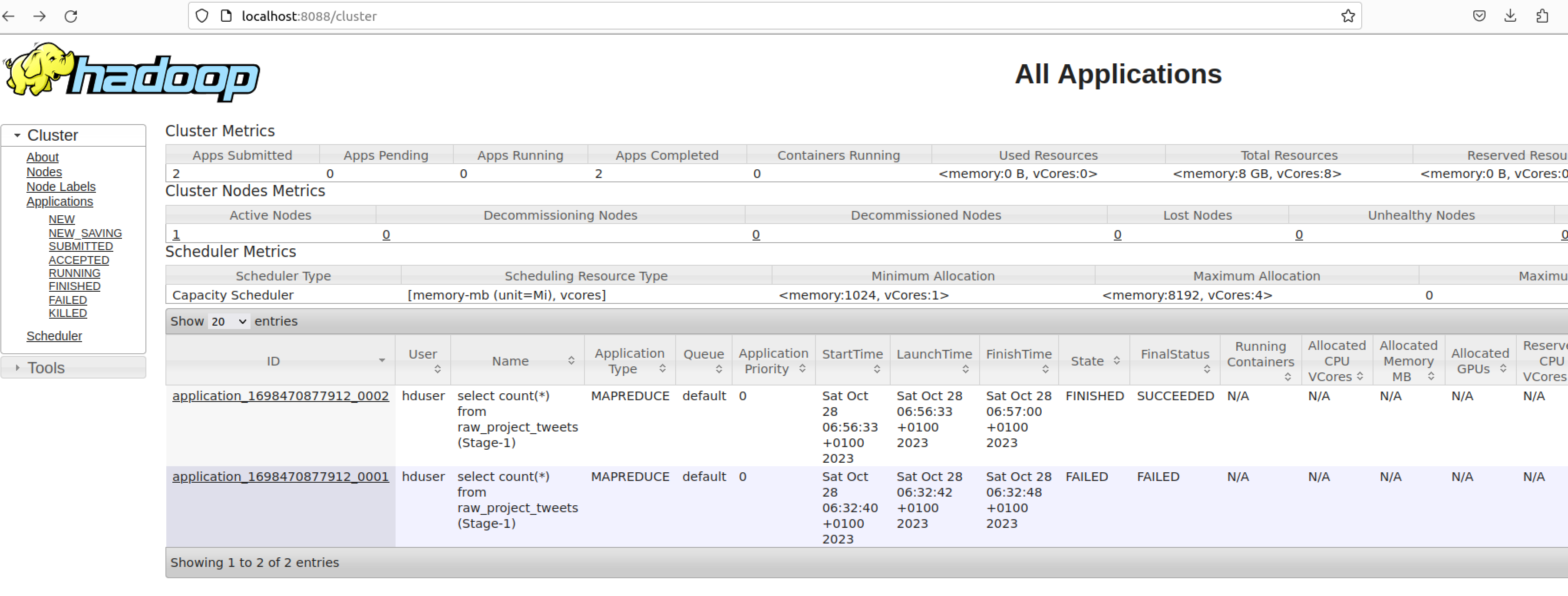
A single column table was created in Hive so that the data could be viewed unparsed in its raw state.



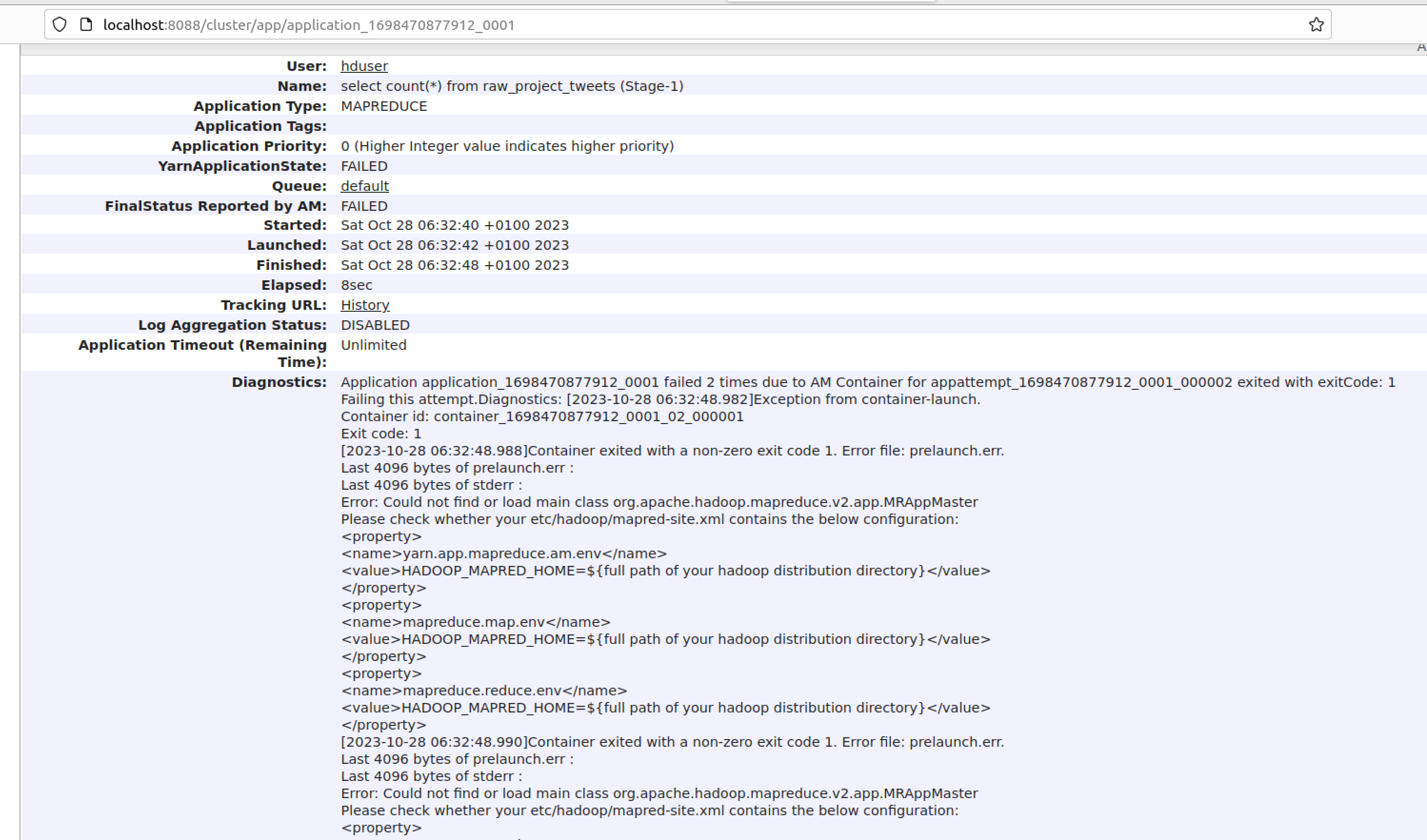
I checked to ensure the data was loaded correctly by first of all selecting a count of the rows:



It can be seen that 1.6 million tweets are present in the table. The mapreduce job failed at first because I didn’t have the relevant properties set up in the Mapred-site.xml file. This was evident in the cluster manager (found on localhost:8088):

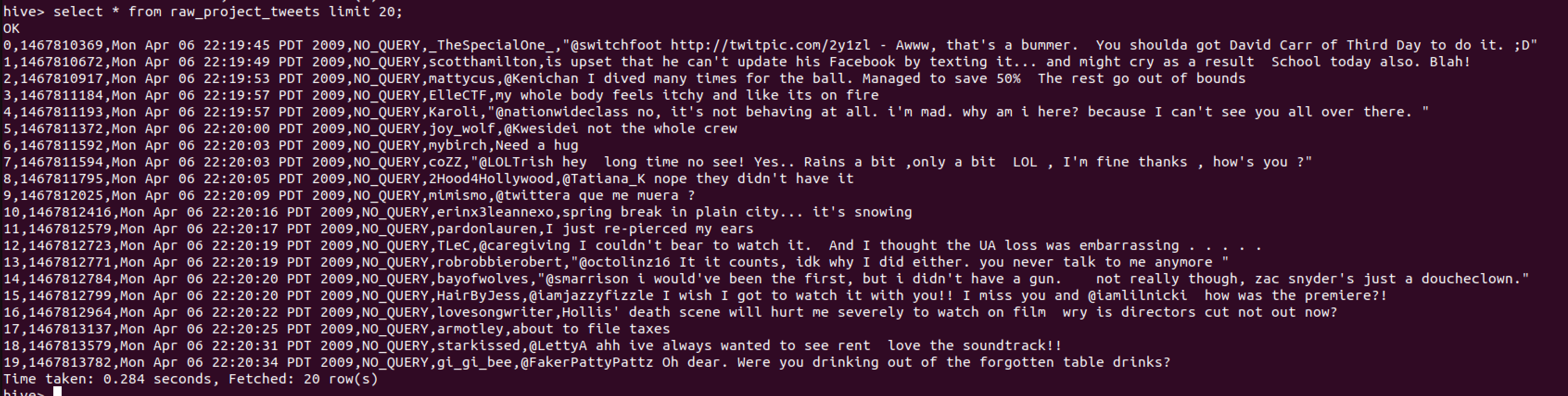


After clicking into the job itself, further details on the fail are visable:



Once I updated the XML file, the job ran successfully.

After seeing that all records made it to the table successfully, I wanted to get a very initial view of the type of data that was present by doing a select \* (limited to 20 rows):



Features such as Tweet id, date and username appear to have a consistent structure with very little noise. However the final column will need a lot of tidying before any worthwhile sentiment analysis can be performed.

Next I started a Pyspark session in Python to Preprocess and clean the data.

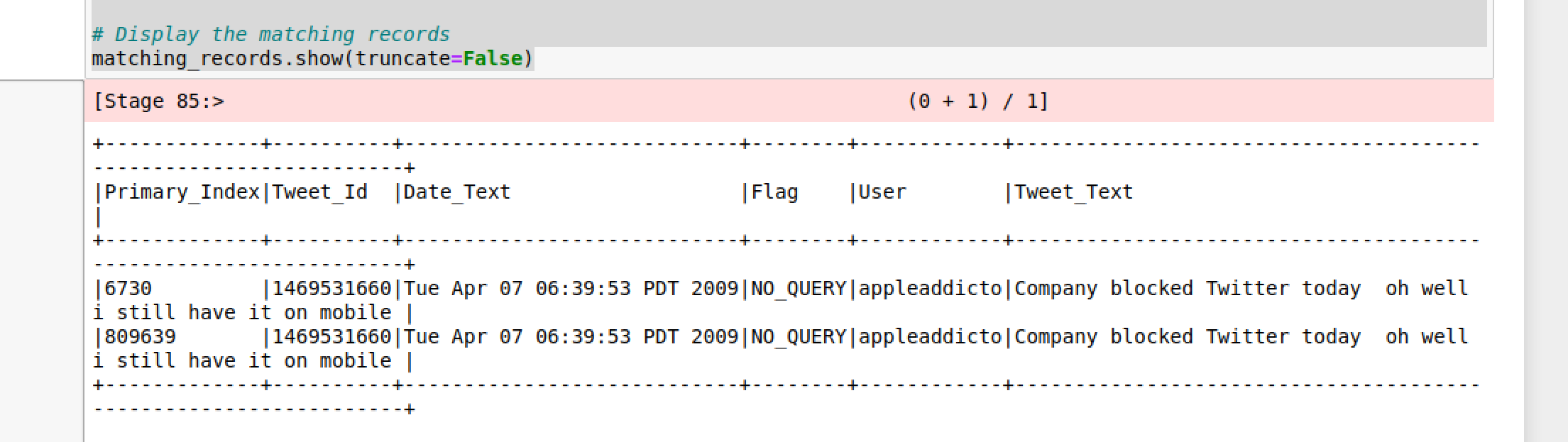
The data was loaded into the Pyspark session and the commas from the final column were removed. Commas don’t have any effect on a sentiment score or forecast, and can cause problems when moving data from one system to another while CSV delimited.

*Early Data Analysis and Preprocessing*

It is important to familiarise oneself with the dataset so further analysis was carried out on the Pyspark dataframe. Max length of each column was found.

A function was run to see if there were any nulls found in any columns. No null values were found.

A function was run to find the number of unique values in each column. This showed up a surprising result. There were only 1,598,315 unique “Tweet Ids” which suggested a level of duplication. A function was run to show an example of this which proved there were duplicate tweets in the dataset:



The reason why this is very relevant is because having duplicate tweets with, for example, a negative sentiment might skew the average sentiment score for a given day.

A new dataframe was created with unique records across the four columns:

Tweet\_Id,

Date\_Text,

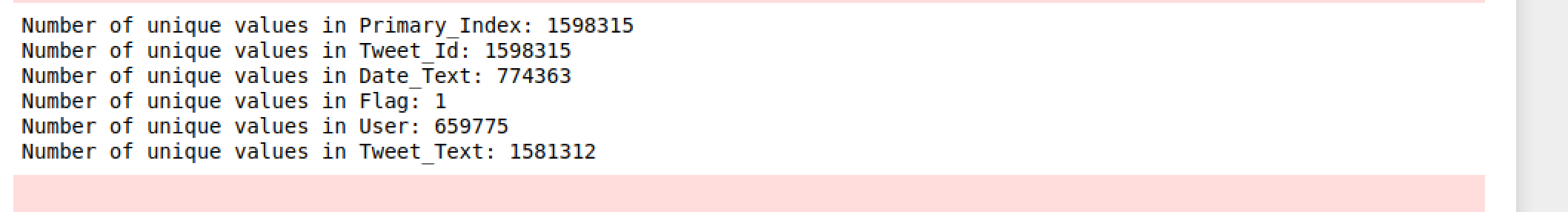
Flag,

User

And Tweet\_Text

To test the outcome, the original example was queried and was found to be unique in the new table.

Again a function was run to count the number of unique values in each column:

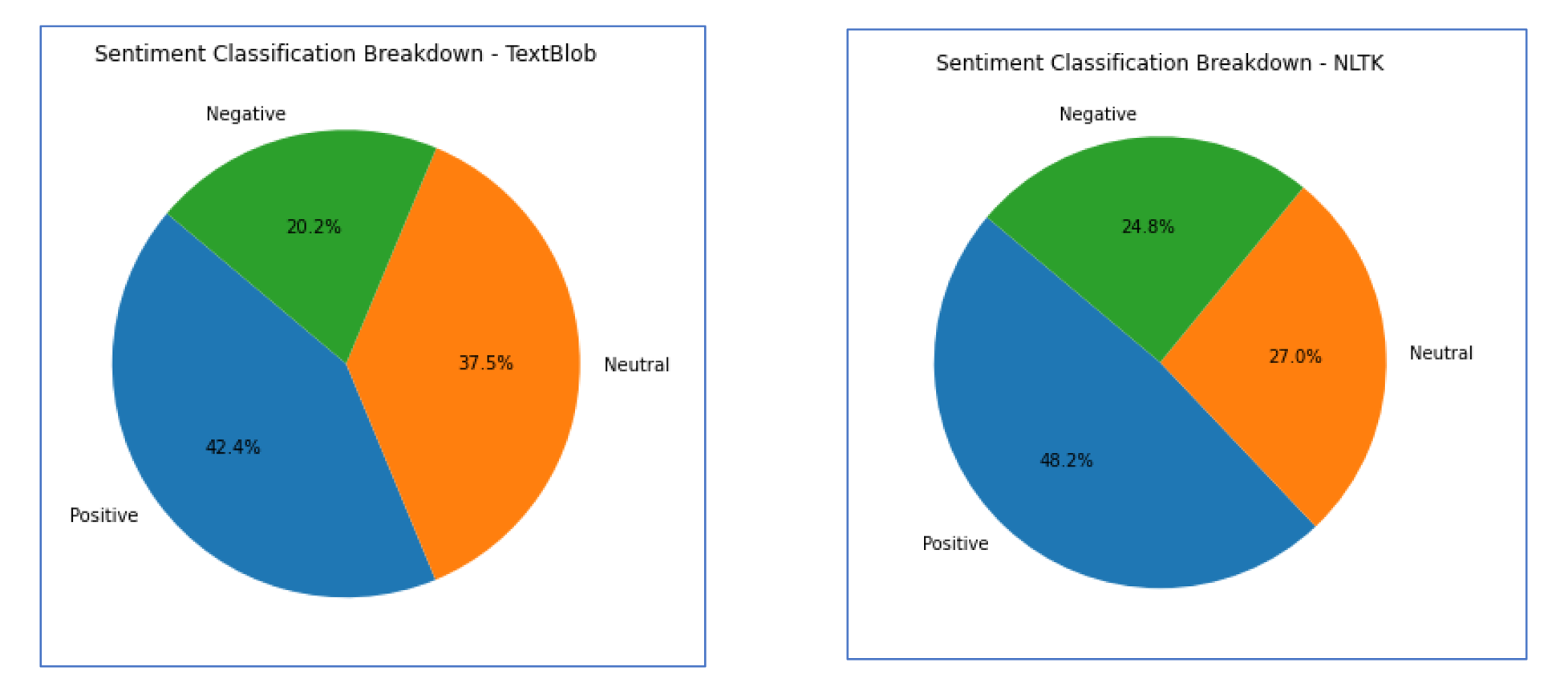


It can be seen that there are still some duplicates in the Tweet\_Text field, but it was deemed necessary to keep these since they have unique Tweet Ids and could be different users posting common tweets such as “Happy Birthday”/ “What a goal!” etc.

Before formatting the date data, I checked to ensure all tweets were made in the same timezone (PDT) and that was the case.

**Sentiment Analysis**

TextBlob and SIA were used to compare sentiment scores and the two provided similar results:



Textblob appeared to classify more tweets as neutral than positive in comparison to NLTK which found more positive tweets, while both found similar amounts of negative tweets.

Missing Values

There are only 81 days between the start of the dataset (‘2009-04-06’) and the end of the dataset (‘2009-06-25’). 33 of those 80 days have no tweets, and therefore no sentiment. With another dataset, it might be prudent to query with the people in the business domain as to whether the null values are “true” null values and there were simply no tweets on that day in which case the sentiment score could arguably set to ‘0’. In this case however, since “Twitter” famously has many tweets per day, it will be assumed that some tweets are missing and we should take action to impute the data.

It is worth noting that to have 33 values missing out of 80 is quite a large amount of missing values. Imputing missing values generally adds noise to the dataset and therefore the less imputing the better.

There are many ways we could impute the missing data. There are options such as forward fill, backward fill, linear interpolation, cubic interpolation as well as seasonal means. The approach taken here is to test each of these approaches against the data we know is correct, to see which approach brought about the most accurate score. That approach will then be applied to the rest of the dataset.

Between the 25/05/2009 and the 07/06/2009 there are no missing dates and a continuous 2 week period of populated data. The proposed approach is to take this small block of data, null 4 values from it, and re-populate those 4 values using the algorithms mentioned above. A comparison can then be drawn between the real data, and the filled data to see which algorithm is the most the most accurate, or has the lowest mean squared error.

This is not the perfect solution, and if time allowed, a lot more experimentation would be performed to see the best fit. For example, it might be more effective to use a mix of these approaches – Forward fill to populate an isolated missing value, and linear interpolation to populate larger blocks of missing data such as 22/04/2009 to 30/04/2009. Before any solution like this is attempted, it would be critical to query the business domain for additional data because real data is always preferential above imputed data.

For the purposes of this project, all options were explored and scientifically assessed using the MSE, and the backfill option appeared to be the most effective and so it was chosen and the data was backfilled. This gave us a complete data set.

Decompose the data

The data was decomposed so that trend and seasonality could be analysed. The trend line appeared very flat in the plot which suggests there is no positive or negative trend over time. The seasonality fluctuations are very low which suggests a very low degree of seasonality.

The Resid plot shows that there is not much left over after the Trend and Seasonality factors have been considered, except towards the end where the does appear to be some noise in the data.

To test the decomposition ran correctly, a single observed value was taken from the table and the sum of the Resid, seasonal, and trend was obtained to ensure they totalled to equal the observed value’s amount.

The example taken showed the trend value was very close to being the observed value itself, with seasonality very low and likewise resid score was very low. This might suggest, along with the plots already analysed, that there is no real long term trend in the dataset and creating an accurate time series forecast might be difficult if not impossible.

Test for Stationarity

The first thing to check before carrying out a Time Series is a check for stationarity. We can see in the above graphs that the mean decreases drastically as time increases, and so visually we know that it is not stationary.

This can also be proven by performing an Augmented Dickey Fuller test.

The Null Hypothesis is that the Time Series is not stationary in nature.

The Alternate Hypothesis is that the Time Series is stationary in nature.

We know that if the ADF statistic is less than the critical value we can reject the null hypothesis, and if it is greater than the critical value, it failed to reject the null hypothesis.

After carrying out the test we can see statistically that we failed to reject the null hypothesis and the data is likely non-stationary.

The ACF and PACF plots help us visualise the correlation of the time series with its lags. It allows us to see the correlation between a given day and previous days. The PACF also helps us visualise whether there is any correlation between a given day and previous days *after* the effects of those days between have been removed. The ACF shows that the correlation with the lags are high and positive with a steady, slow decay. Meanwhile the PACF plot shows a partial autocorrelation having a single spike at lag 1. These are well-known features of a well-known time series called “Random Walk” which is not stationary.

*Differencing the data*

To make the data more stationary, ‘differencing’ was used. This is a common technique to make non-stationary time series more stationary. The first differenced time series doesn’t show a trend:

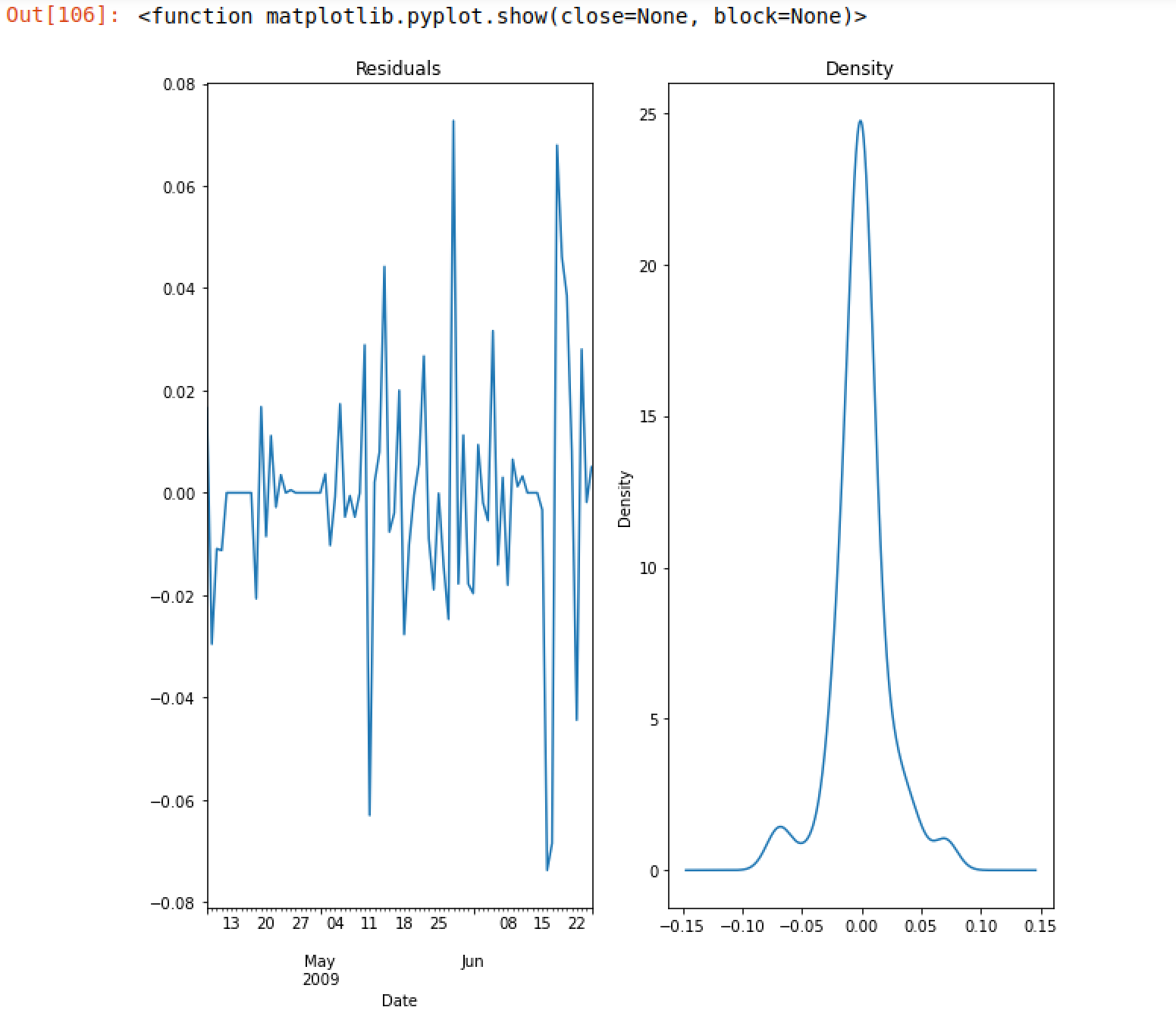
The ACF and PACF plots were again run. In comparison to the original series, the acf plot drops in value more quickly which means the Time Series is less correlated to its lag. Meanwhile the PACF plot also shows a less strong spike at lag 1. These are both signs that the series is more stationary.

Statistically, we re-run the adf test which shows a p-value which is a lot less than the confidence level of 0.05 which means the null hypothesis can be rejected and the first difference time series is likely to be stationary.

Run the Arima Time Series

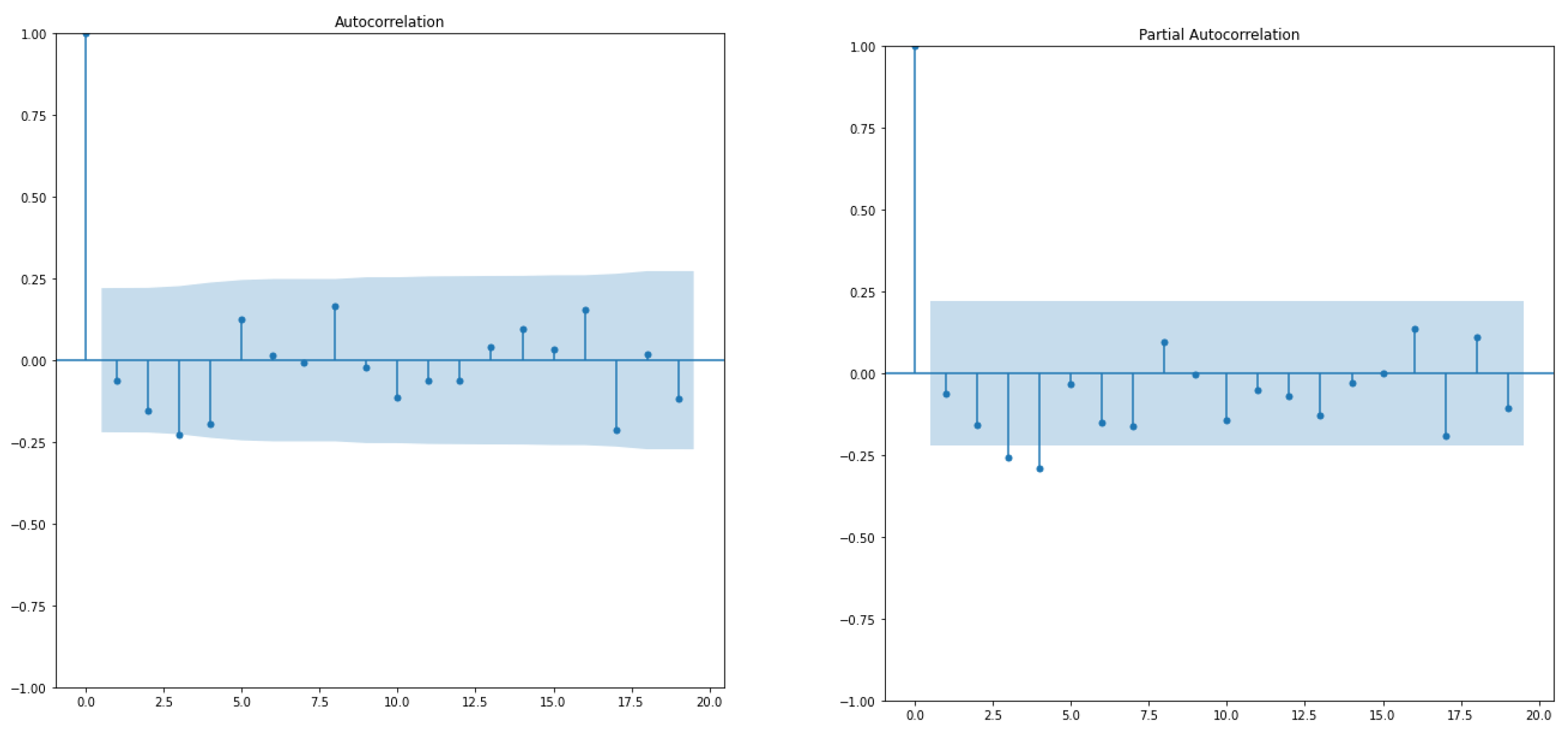
It can be hard to fine tune the parameters for Arima models.

Before using the model to make predictions, it’s important to check to see that the model took adequate information from the data. For this we plot the residuals and the density was also checked.



The residuals graph is showing mostly noise, and the Density plot is showing a normal distribution which is centered around zero.

The ACF and PACF graphs can also be generated for the residuals to check for this.



The first number of lags barely show any significant spikes which means the residuals are close to white noise.

<https://builtin.com/data-science/time-series-forecasting-python>

^ good for seasonality vs trend and gives a link to description. Plus it talks through a few Time Series approaches

Time Series

Now that the missing values are filled in, we can continue with the time series

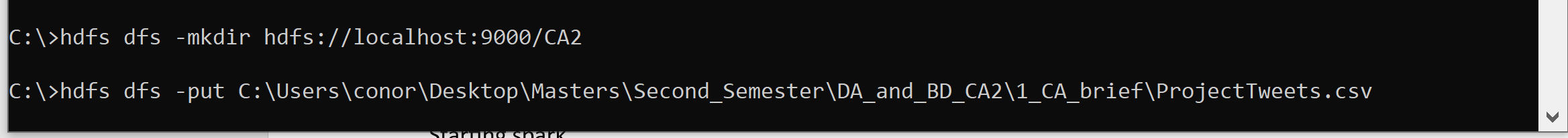
The data dips quite suddenly towards the end of the time series. Since it doesn’t appear to be a true reflection of the general trend which preceded it, a larger sample (30%) was taken because it contains a better balance.

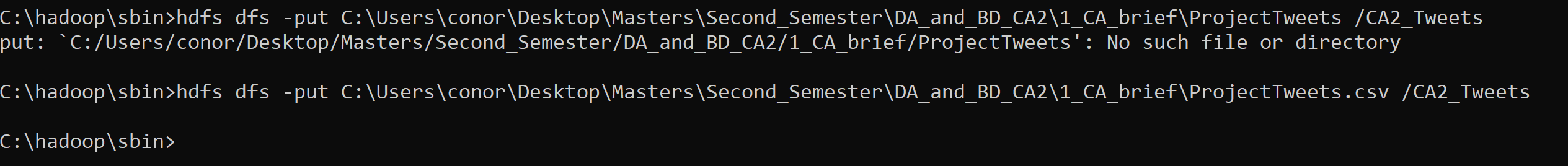
First a quick Arma Time Series was performed and it failed to produce anything worth while. This is probably because the

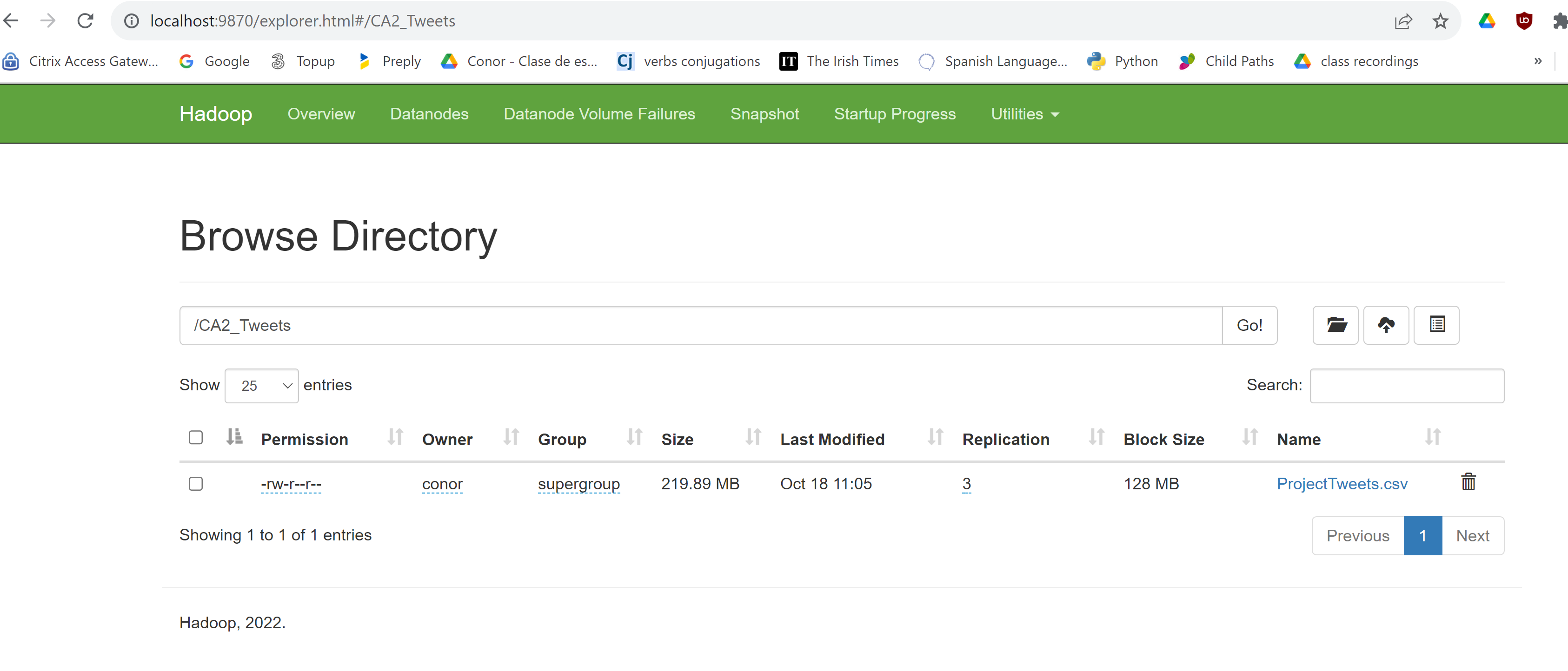
Loading the file into HDFS

I opened HDFS using the CMD prompt (ran as an administrator) and I uploaded the ProjectTweets dataset using the put function in the CMD line:

Initially there were issues because the file path had spaces within it but after those spaces were removed, the file successfully uploaded into HDFS:



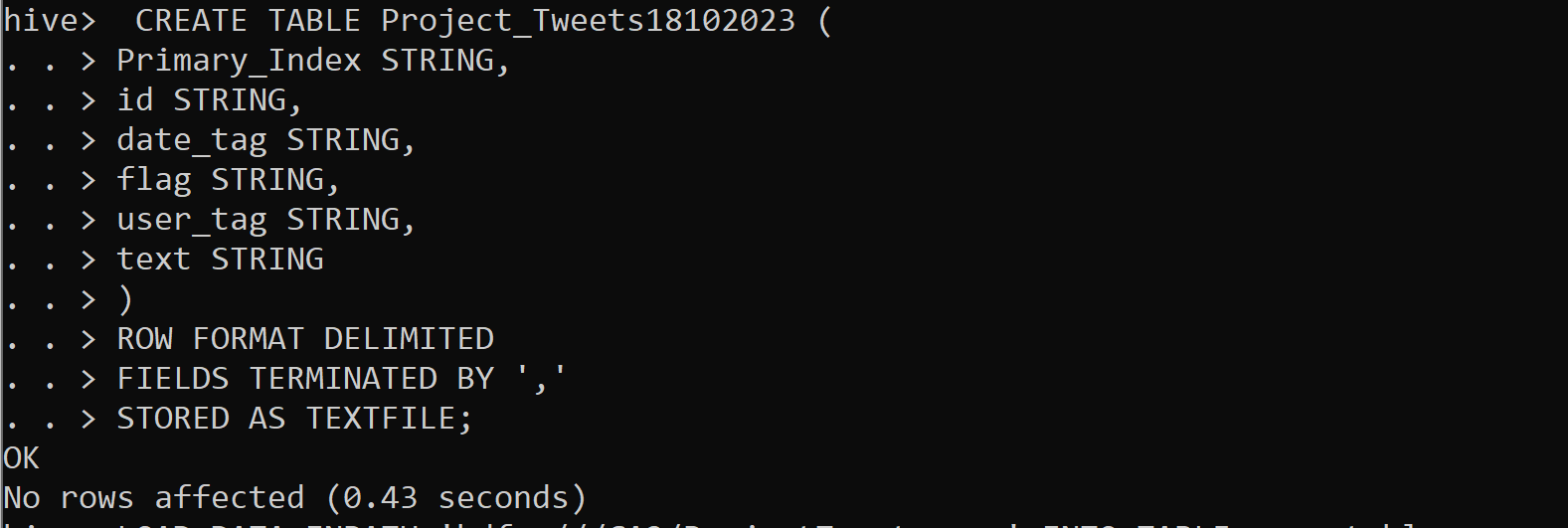


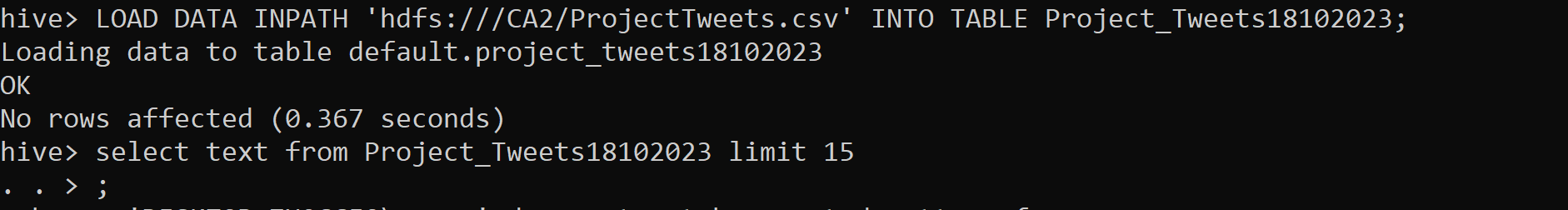


We are using a standalone Hadoop distribution. But it’s possible to check the cluster detail, so it’s possible to check how many clusters on the standalone machine Hadoop is developing.

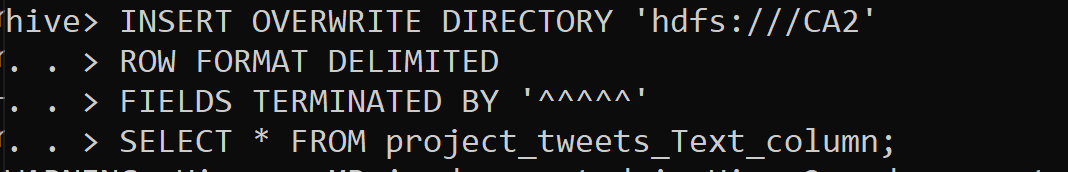
If we had different machines we could decide how to distribute the data, Hadoop automatically takes responsibility that but there are. Hadoop does it all internally.

HIVE





Send back the one column for the map reduce job



Mapreduce:

C:\>hadoop jar C:\hadoop\share\hadoop\mapreduce\hadoop-mapreduce-examples-2.10.2.jar wordcount /CA2/000000\_0 /output/part\_1

