# **Abstract**

*The primary goal of this continuous assessment is to demonstrate learning outcomes in big data and advanced data analytics by processing and analyzing a tweet dataset using associated tools and methodologies.*

*The topic I choose was about anti-farm law protest that occurred on the outskirts of India's national capital, New Delhi in 2020. The files related to this event were downloaded from the Kaggle.com and then analysed on Oracle Virtual Machine. The files were cleaned and manipulated using Python libraries, then stored on Hadoop HDFS where then it followed interaction with the PySpark. There are two models were trained regarding to the sentiment scoring and its forecast. The outcomes are discussed in interpretation parts.*

*All the details about the project implementation are documented on the attached Jupyter Notebook file. For Linux Ubuntu Terminal executions there are screenshots uploaded as an image file on this report documentation. Git was used for daily code tracking, and GitHub was used for archiving, monitoring, and sharing. Click on the following link to view the project on GitHub:*

<https://github.com/temulenbd/ca2.git>

# **Introduction**

Social media platforms, particularly Twitter, have become rich sources of data that represent public opinions, emotions, views, and attitudes toward particular entities. Analyzing its data can be beneficial in many different domains such as market research, trend analysis, customer insight, crisis management, academic research and etc. And this assessment focuses on the prediction of the changes of the tweet sentiment related to the farmers protest which occurred in India, 2020.

I found and downloaded 'Farmers Protest Tweets' CSV files related to the protest event from Kaggle.com. The file owner is Kaggle user Pratham Sharma and he collected this dataset from the Twitter API using the SnScrape Python package and the hashtag #FarmersProtest. The files are licensed by the CCO: Public Domain, which enables copying, altering, distributing, and performing the work without seeking permission, and it is free to the public. All the tweets in it are in English and span from November 1st, 2020, to November 21st, 2021.

In order to perform the analysis by following the tasks presented from the school I decided to divide my project into following four interdepended sections, consisting of subsections where the code execution and analysis are thoroughly discussed and interpreted:

1. Tools and Modules List.

2. Data Cleaning and Data Storage.

3. Pre-processing and Sentiment.

4. EDA and Forecast.

5. Databases and Comparison.

# **Tools and Modules list**

(Jupyter Notebook: 1– 3)

Virtual Machine: VM Oracle VirtualBox version 7.0.6

Operating System: Ubuntu 22.04 LTS (4 Gb Memory, 60 Gb Storage)

Coding: Jupyter Notebook version 6.5.2 using Python version 3.10.9

Distributed file system: HDFS version 3.2.4

API for distributed file system: Spark IU version 3.2.4

Version Control: Git version 2.34.1

# **Data Cleaning and Data Storage**

(Jupyter Notebook: 4 – 32)

The Zip file I downloaded included two separate CSV files, one containing the tweets and the other containing the user's information on those tweets. Because both files have helpful information, it is critical to integrate them and sort out only the valuable data. I divided my execution into two subsections, ‘data storing and preparation’ and ‘data cleaning and manipulation’.

## 2.1 Data storing and its preparation

(Jupyter Notebook: 5 – 13)

I examined both files on Pandas by creating <df\_tweets> and <df\_users> data frames. Using the <info> function on both objects, I found that the file <tweets> consisted of 14 columns and 1,084,452 rows of information, whereas the file <users> had 19 columns and 235,660 rows. The most important information for sentimental analysis, the tweets and the tweet dates were in the <df\_tweets> data frame, with the original tweets in the column <renderedContent> and tweeted date in the column <date>. And the data frame <df\_users> didn't have the necessary information that may be used for further analysis. However, instead of using a vast collection of numbers, I chose to extract values from the column <displayname> and assign them to the tweets so that I could know their author.

Furthermore, because I intended to perform EDA on the final dataset, I kept the columns <userId>, <retweetcount>, and <likeCount> from the <tweets> file, then dropped the reset. I kept the columns <displayname> and <userId> from the <users file but dropped the rest. After that, I combined two data frames using the Pandas <merge> function on the left, and the output columns were arranged as <date>, <renderedContent>, <retweetCount>, <likeCount>, and <displayname>.

I copied the file in CSV format to my Oracle VM and measured its size. After merging two files and removing unnecessary information, the total size was decreased from 1.7 GB to 659 MB. Despite its small size, the file offered all the useful information for further analysis. It is now ready to be imported into HDFS and then manipulated with PySpark. As shown in Figure 1, I created a directory on the HDFS called <ca2> and moved the file <new\_tweets.csv> to it.

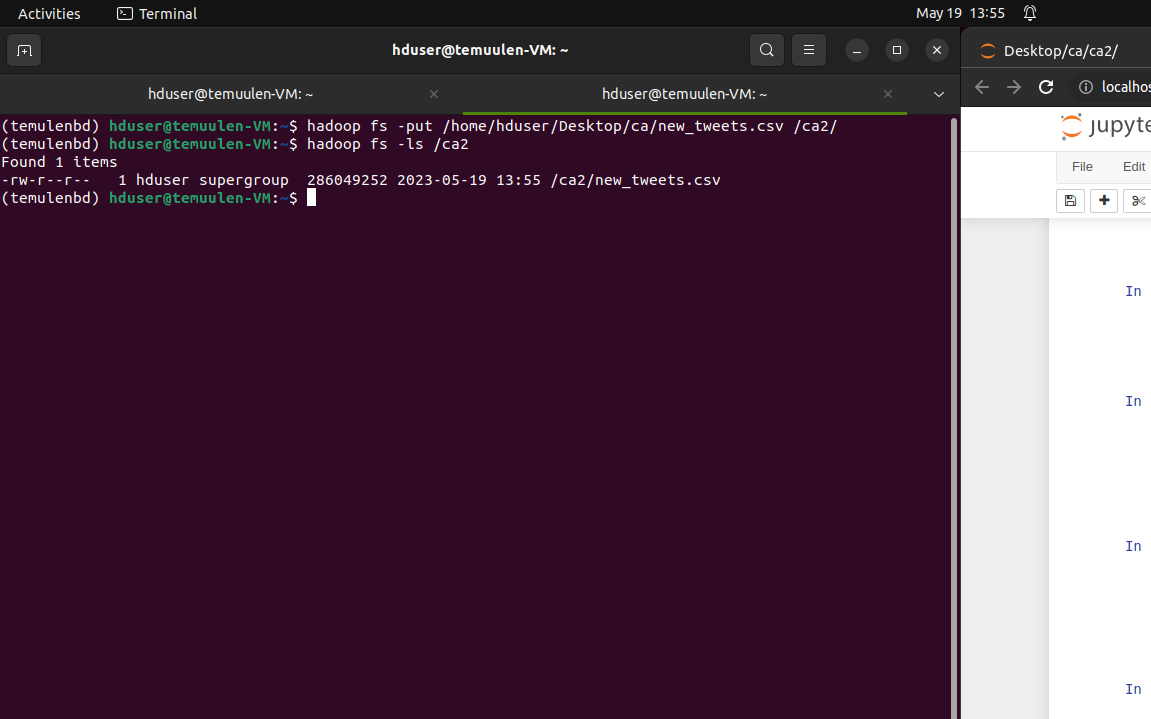


Figure 1. File import into HDFS, 19-05-2023.

## 2.2 Data cleaning and manipulation

(Jupyter Notebook: 14 – 32)

This section is solely dedicated to PySpark. The functions and techniques of PySpark were only used to clean and manipulate the new tweets file and integrate with HDFS. I started by setting up a new Spark session called 'Sentiment Forecast' and imported the <new\_tweet> file from HDFS to create a new PySpark data frame called <df\_spark>. After examining the columns and their datatypes, I discovered that all the values imported from HDFS were assigned to the string data type. Later, after reading more about the PySpark data frame, I learned that if I don't indicate the schema before reading the CSV file, PySpark will treat all columns and their values as strings by default. But this time I'll manually convert them to the proper format.

Column names were shortened but made clearer, data types were converted, and each column now has an appropriate data type. Now the following step is to remove any null values and duplicates. There are 2,953,820 rows with several duplicates, as shown by the execution of the cell number sixteen and twenty-three. These unnecessary rows are the result from Panda's data frame's <merge> method and must be removed.

Furthermore, I can see from the above-mentioned executed cells that many values on the column with the username information have a null value. It could mean one of these options: the file owner who extracted the data from Twitter was unable to pull it entirely; these missing usernames no longer exist on the social media platform; they were blocked from the platform for trolling; or, for whatever reason, their profiles could not be viewed on the website. It doesn't matter what was the cause of this, but I chose to eliminate the rows with missing values in the column 'user' to avoid bias in my research. After that, I adjusted the table's order by date and printed the new shape of the data frame. The printed cell showed that there are no missing values and shape now changed to 1.082,411\* 5.

Because the column 'tweet' contains the most important information for sentiment

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure 2. File upload into HDFS, 19-05-2023.

analysis, it must be treated differently. All values in this column must be cleaned up, and the noise must be removed. I was able to remove tags, hashtags, emails, or website links by generating code using PySpark's <regexp\_replace> function. However, after checking the changes, I found that more string manipulation of the values was required on the column. I generated code again using the <regexp\_replace>, <trim> and <lower> functions. This time I was able to not only remove leading and trailing whitespaces, ampersand characters, different punctuational marks, or non-English text but also replace two or more continuous whitespaces or new lines with a single whitespace and convert the entire string values into lowercase. Then checked the changes, the cleaning was done. After reviewing the changes, the file was uploaded to HDFS so that I would have a backup file in case something went wrong in my VM. It was successfully uploaded into the distributed file system, as seen in Figure 2.

# **3. Pre-processing and Sentiment**

(Jupyter Notebook: 33 - 66)

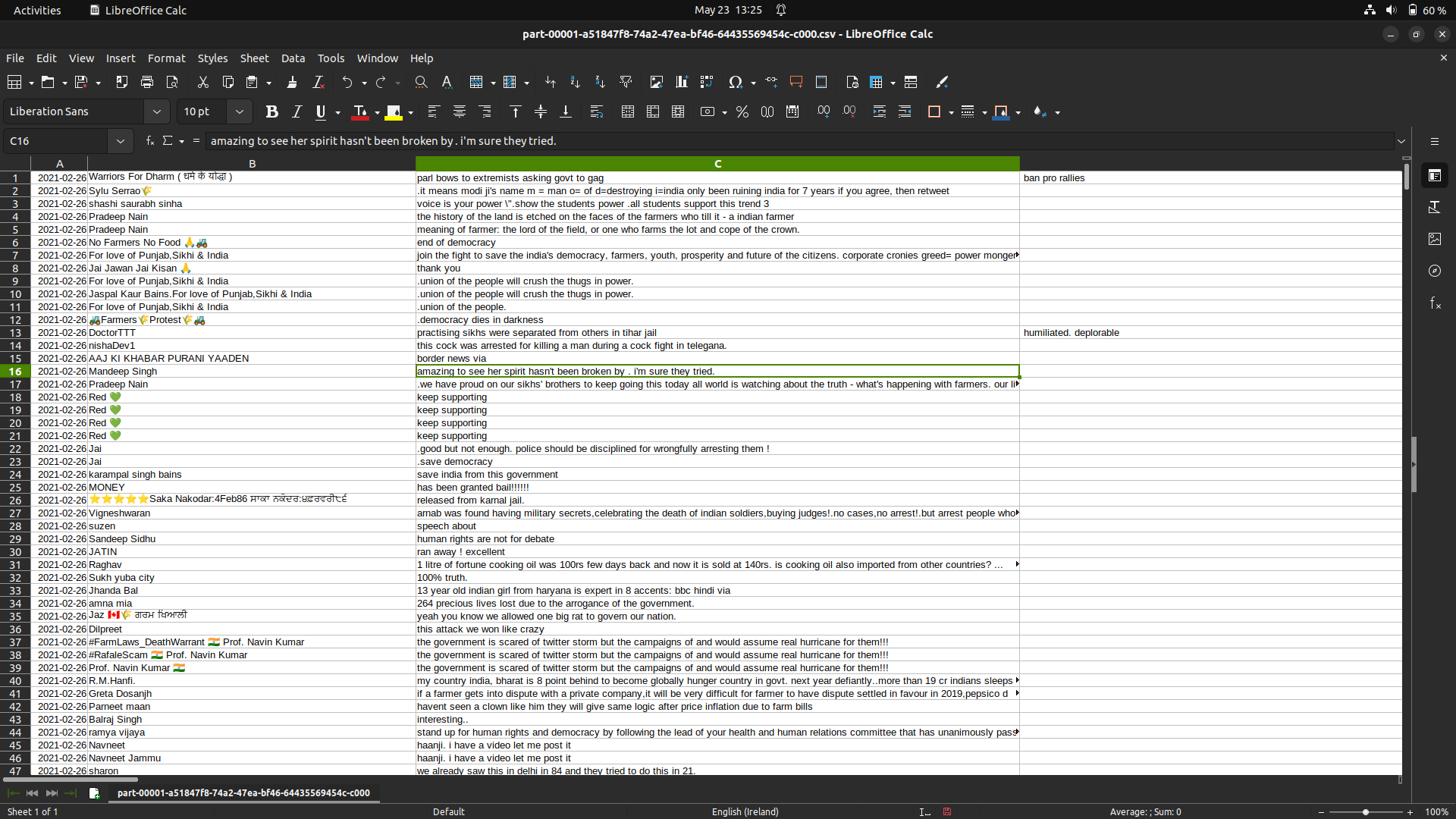
Since I'm using the protest tweet data set, it's obvious that a significant number of tweets will have a negative tone of writing. Most people were on the protesters' side, which can be seen from reading the clean tweets file downloaded from HDFS(Figure 3). It was impossible to determine sentiment and analyze the event.

Figure 3. Tweets CSV file downloaded from HDFS, 23-05-2023.

That's why my decision was to choose the proper user and focus only on that user's sentiment change to better understand the protest. And to be honest, I didn’t believe that all of the users who tweeted about the demonstration were real people. I observed some accounts on Twitter.com, but some were active only during the protest, and some didn't even exist. I believe there were bots running tweets behind the scenes to draw individuals' or foreign media's attention to the protest.

I ran the code using the <groupBy> function and glanced for the users with the most tweets. My intuition was correct; some accounts even featured the word 'bot' in their display names; others had similar names and the same quantity of tweets. So, in my opinion, choosing someone with an actual verified account was the best method to examine the sentiment. I debated whether to follow @KuldeepDhaliwal, Minister of Rural Development, or @IndiaToday, India's most prominent news magazine page. Both accounts were genuine and verified, but after reviewing the tweets, I decided to stick with user @IndiaToday because @KuldeepDhaliwal began lately tweeting about the demonstration, and his tweets were simply copies of the same tweets. So, I created a new data frame with the object name <df\_for\_sentiment>, containing only the user's tweets from @IndiaToday. Regarding the execution of the <show> function, it can be seen that there are 3479 tweets on it and three more columns with the date and retweet-like counts.

## Tokenization

(Jupyter Notebook: 39 - 41)

Because the PySpark machine learning library offers a few essential class functions for pre-processing, I decided to execute several NLP techniques in the current data frame. The first step is tokenization, which prepares the text for future processing. By default, the <Tokenizer> class separates text into individual tokens depending on the default delimiter, which in my case is <whitespace>. I executed the code and added a token column. Everything appeared to be placed correctly after reviewing the adjustments.

## Stop word removal

(Jupyter Notebook: 42 - 44)

When I examined the tokens column more closely, I discovered some regularly used words that are inconsequential and have no critical significance. I can reduce noise and concentrate on more important words by deleting these. I loaded the default English stop word list from PySpark's machine learning library and ran code on the token’s column using the <StopWordsRemover>. In order to observe and compare the changes, I added a new column called <clean\_token>.

## Spellcheck and lemmatization on Pandas

(Jupyter Notebook: 45 -55)

I did tokenization and stop word removal on the PySpark data frame however, it lacks libraries for spell checking and lemmatization. I didn't think there are any grammatical problems in the text because it is from one of the largest news publications' official Twitter page. But, just to be sure, and out of curiosity, I decided to try each of those techniques in a Pandas data frame. And before converting my data to Pandas, I experimented with various spell-checking and lemmatization libraries. However, none of them worked with the PySpark data frame. During the execution procedure, there were various JavaScript errors. Some libraries experienced import issues, while others just showed the text that they stopped development for PySpark.

Before beginning the pre-processing in Pandas, I changed the data type of the table and its values. Then, perform a spellcheck on the column <clean\_token> by importing the <aotucorrect> library's <Speler> class and writing a custom function for the application. As a result of the code execution, I got a new column called <corrected\_token> with the grammatically correct values.

The term lemmatization was defined as "a natural language processing technique that reduces words to their base or dictionary form". And it is, in my opinion, one of the most important pre-processing elements. It is critical to perform it right.  Otherwise, it may misinterpret the meaning, resulting in an incorrect sentiment. But fortunately, in my case, the tweets used are formal and official text, so I hope there won't be any bias in this situation. I downloaded the <wordnet> package from the <nltk> library and used a custom function to apply the package's lemmatizer on the values of the <corrected\_token> column.

## Polarity scoring

(Jupyter Notebook: 56 - 60)

From the results of all four previous pre-processing procedures, I got another column of tokens called <last\_token>, which will be used for polarity scoring. Polarity scoring is a sentiment analysis technique that assigns a numerical score or value to a piece of text to represent its sentiment polarity or sentiment strength. For this case, I'm using Natural Language Toolkit's pre-trained machine learning model for PySpark, <TheSentimentIntesityAnalyzer>, to analyze the text and determine the intensity value ranging from -1 to 1.

I generated the code using a custom function, applied it to my PySpark data frame, and retrieved the polarity values as an outcome. Then I created another code to convert these numbers into positive and negative sentiments. I picked only two sentiments because the writing is formal, and the text structure is similar to a report. In this case, in my opinion, being highly nuanced and capturing subtle sentiments is unnecessary.

## Sentiment analysis model in Pyspark

(Jupyter Notebook: 61 - 66)

This section was added to clarify the outcomes of the prior pre-processing steps. By training and testing the NLP model using the current output data frame, I can determine whether or not the work done thus far has been valued.

I used tweets as an input feature and tweet sentiments as an output-dependent variable. I had to make a decision on which model to use. But, after some research and consideration, I decided on Logistic Regression for this topic because it is the most widely used and one of the simplest algorithms for solving binary problems and for NLP in general.

As I mentioned before, my input and output values are complete. I'm using the <last\_token> column for the independent variable and the 'sentiment' column for the dependent variable. It is ready for training with only minor modifications and changes. I converted the tweet's sentiment into 1s and 0s using the <when> and <else> PySpark functions. I also successfully converted values from tweet tokens to numerics using the HashingTF function of PySpark's machine-learning library. Then, I divided the data by 70/30 and trained the model with PySpark's Logistic Regression library. And lastly, I tested the model on the test dataset.

The model performed well, correctly predicting 937 out of 1100 variables with an accuracy of 0.86. I can conclude that the pre-processing step I performed were successful with the dataset, and I can use this trained Logistic Regression model to predict tweets on the farmers' protest. But I'm not saying it's certain since adjustments have to be made depending on unaccounted factors, even if it's been trained.

# **EDA and Forecast**

(Jupyter Notebook: 67 - 90)

I expected to use the entire dataset for my data visualization, assigning polarity and sentiment to it. However, as I commented in the Jupyter Notebook file, the conversion took a few hours and was never completed. That's why I chose to use only the dataset of the user @IndiaToday.

## EDA

(Jupyter Notebook: 68 - 80)

Because I lacked knowledge about data visualization on the PySpark data frame, I used the Pandas data frame for IDE. I began by creating a new object named <df>, which was the Pandas data frame transformed from PySpark. Then I renamed the columns and updated the data types to make it easier to examine and started exploration.

The first graph I made using the <df> table is a line graph illustrating the frequency of tweets by the user @IndiaToday. The data covered the period from November 2020 to November 2021. I can see from cell 71 that the tweet frequency at the beginning of the time was quite zig-zagged, hitting a high of 115 and a low of 0 tweets per day. But after February 2021, it became more consistent, averaging between 0 and 20 per day for the rest of the period.

The second graph I generated was a two-line graph showing the frequency of negative and positive tweets of the user @IndiaToday from November 2020 to November 2021. I observed that the frequency of each sentiment increased dramatically from November 2020 to December 2020, reaching its peak. The highest number of positive tweets, 400 per month, were tweeted in December 2020, whereas the highest number of negative tweets, almost 1000 per month, were tweeted during the same period. After December 2020, both tweet counts fell gradually, reaching a low in March 2021. There is a slight variation from March 2021 to November 2021, with both averaging 0 to 40 tweets each month. And it is clear that the number of negative tweets was always greater than the number of good tweets over the provided time period.

The third pie chart displayed the total amount of sentiment tweets by the user @IndiaToday between November 2020 and November 2021. The pie chart showed that negative sentiment outnumbered positive sentiment, with values 2513 and 966, respectively, and it was almost three times greater.

The top three liked tweets, and top three retweeted tweets were printed in the two following cells. The most liked tweet was posted on February 2nd, 2021, receiving 5413 likes. This tweet also received the most retweets, with 1332 times. I looked up this tweet and found it was about Barbados star Rihanna.

The next cell listed the most frequently used words from the tweets. The top five most often used words were farmer, protest, watch, delhi, and law, with counts of 2157, 636, 560, 475, and 404, respectively. However, these words were not counted from the tweets but from the token column.

The WordCloud plots were included as the final element in EDA. There are three plots in total, and they show the most commonly used words in all tweets, all positive, and all negative tweets. You can find them in my Jupyter Notebook’s Figures 4, 5, and 6.

## Forecasting using ARIMA model

(Jupyter Notebook: 81 - 90)

I wanted to experiment with my data using the ARIMA model for forecasting because it is one of the most popular and powerful forecasting methods. ARIMA, which stands for AutoRegressive Integrated Moving Average, captures data patterns, trends, and seasonality by combining prior values, differences, and errors. The ARIMA model has the advantage of being versatile and able to handle a wide range of time series data as long as they are univariate. However, before using it, I had to double-check my data to ensure the model could be applied to it.

First, I prepared the data for the analysis, so I converted my prior PySpark data frame into Pandas and made some minor changes. Such as removing all unnecessary columns, renaming column names, changing the data type to an appropriate one, and removing the last five digits from date column values. After reviewing the results, I saw that the new data consisted of one column called "polarity" and an index called "date."

The next step is to examine the data. I plotted the data to see how it looked, and the plot seemed pretty stationary. I did the augmented Dickey-Fuller (ADF) test on the data to confirm my hypothesis, and the P-value result and the ADF values indicated the same. The ADF test number was -6.5, which is way lower than zero, and the P-value of 1.2288184623591962e-08 is very little, less than the significance level. And it confirms my hypothesis: the data is stationary, allowing me to apply the ARIMA model to it.

I need to identify the best ARIMA parameters before training the model. I used the <auto\_arima> function from the <pmdarima> library, and the results showed that the optimal order for the number of autoregressive terms (p), differences (d), and moving average terms (q) is (1,1,1), with the lowest AIC 2921.637. Moreover, the seasonality value (0,0,0) from the test proved again that the ARIMA model can be applied to the data set because there is no seasonality in it.

I had some difficulty with the splitting and training parts. The values are fine because they range from -1 to 1, so no normalization or transformation techniques are required. However, the frequency of tweets, or the index values of this data set, is challenging. Figure 2 shows that tweets were quite active at the start of the period, then dropped practically to 0 and stayed constant, ranging from 0 to 20 tweets per day until the end of the period. There were several gaps between the records; in other words, there was no consistent interval ranging from a fraction of a second to a daily, weekly, or monthly period. I couldn't do anything in this case. Perhaps I could use forward or backward filling methods, or any other technique, to fill the blank gap. However, in my opinion, it would make my project worthless. So, I decided to maintain the data as is but test it with fewer values, giving 30 values for prediction.

The ARIMA model was then trained and tested with the parameters (1,1,1). The outcome only displayed the average of my historical data on forecasting test values. And when I plotted it, I got a straight line. Based on my knowledge, there is no information to learn when the data collection lacks any seasonality or trends. As a result, the model cannot be accurate. I refer it to the the date's blank gap. And if the protest continued, I could suggest that the average sentiment of tweets from user @IndiaToday would be negative, with a score around -0.203491.

# **References**

Brockwell, P.J. et al. 1991, *Time Series: Theory and Methods.* Springer Science & Business Media.

Helmke, M. 2016, *Ubuntu Unleashed 2017 Edition (Includes Content Update Program)*. Sams Publishing.

Illowsky, B. and Dean, S. 2017, *Introductory Statistics*.

McKinney, W. 2017, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython.* “O’Reilly Media, Inc.”

Tunstall, L., Von Werra, L. and Wolf, T. 2022, *Natural Language Processing with Transformers.* “O’Reilly Media, Inc.”

Vrbka, J. 2021, *Using Artificial Neural Networks for Timeseries Smoothing and Forecasting: Case Studies in Economics*. Springer Nature.

<https://www.kaggle.com/datasets/prathamsharma123/farmers-protest-tweets-dataset-csv> (assessed May 13rd, 12:14)

<https://creativecommons.org/publicdomain/zero/1.0/> (assessed May 13rd, 12:42)

<https://www.kaggle.com/prathamsharma123> (assessed May 13rd, 12:57)

<https://slate.com/news-and-politics/2020/12/india-farmer-protests-modi.html> (assessed May 13rd, 13:02)

<http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm> (assessed May 24th, 11:12)

<https://sparkbyexamples.com/pyspark/pyspark-to_date-convert-timestamp-to-date/#:~:text=PySpark%20timestamp%20(%20TimestampType%20)%20consists%20of,be%20yyyy%2DMM%2Ddd%20> (assessed May 24th, 11:13)

<https://twitter.com/kuldeepsinghaap?lang=en> (assessed May 25th, 15:33)