Twitter Analytics Using SparkSQL

Group Members: Vlad Dubrovenski, Andrew Wilson, Abdelmoneim Elfagir

University of Missouri-Kansas City

Computer Science 5540: Principles of Big Data Management

Author Note

This document was prepared as a report of the group project phase 2 for CS 5540

Abstract

This paper outlines our semester project for CS5540, Principles of Big Data Management. Our project explores the possibility of analyzing queries using SparkSQL with Scala as the principle coded language. In this paper we discuss the decisions made while designing the project, the actual implementation of the front- and back-ends, and, of course, testing of this interesting project. This paper will also include the illustrations of the software that we used, as well as graphs that we were able to obtain. It will also include a discussion of why we think the design of the front and back end is not only sufficient to handle large amount of data, but also how it can quickly do so. We also believe that the charts, which are the results of our queries, can provide useful insights into the tweet data, which is extremely beneficial in business applications.

Brands and Stocks Analytics Using Twitter

Our project is about data analytics and market fluctuation. While designing it, we wanted to provide people with a tool that could help them analyze real-time popularity of brands and stocks, which is extremely helpful when deciding when to buy or sell something. Our project instructions stipulated that we were required to use Twitter to gather data and then analyze the data using SparkSQL. To achieve maximum performance, we decided that it was best to use Scala. Initially we were deciding between Python and Scala, but after research and consultation from our professor, we decided that it would be best to use Scala due to its efficiency. We also used Python, but only to stream tweet data. As we will understand as we go further, in our project it is not even required to stream tweets, but we instead can use either Local or Hadoop Distributed File System to store the tweets in JSON format and pull them directly to Spark without even touching Python.

Let's take a look at the structure of the project. In order to make it easier to explain it, we prepared the diagram of dataflow that can be found below (Figure 1). As we can see from the diagram, the process of analysis starts from the decision of whether to use the data that was already stored somewhere, or to use the data that comes directly from the Twitter stream. Let's take a look at latter option: the Twitter stream. For this option we used a Python library called Tweepy, which allows us to stream the tweets with a small amount of code. When the user of our software chooses to stream the tweets, he or she will need to enter their developer keys in the program, as well as languages that they prefer to stream the tweets for, key words that the tweets must contain (in our case those are brands and stocks), and the maximum number of tweets they would like to obtain.

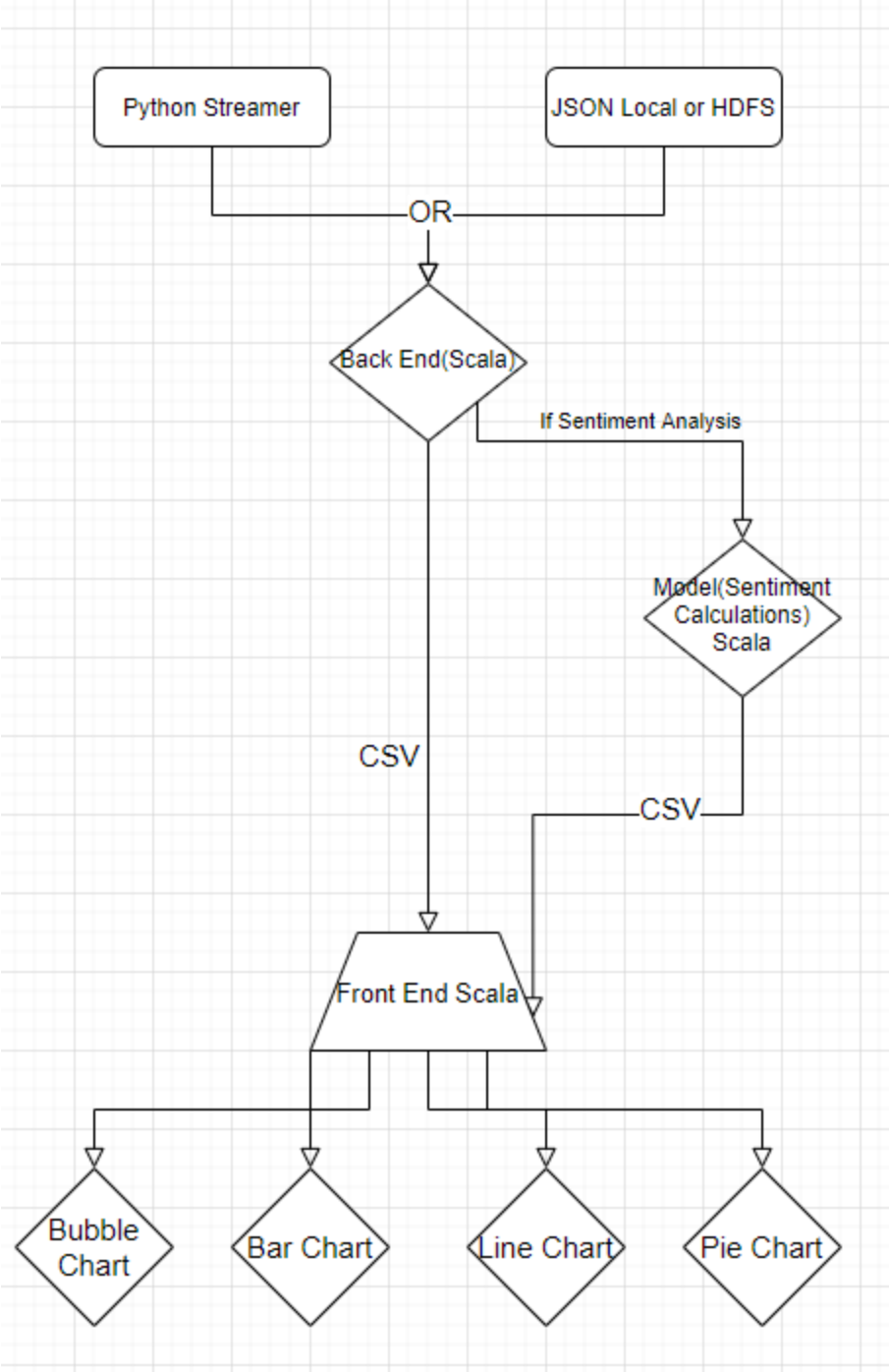

Figure 1(Architecture of the Analytics Tool)

We decided to end streaming after a predetermined number of tweets because it is more effective for a user to know how many tweets he needs in order to perform sufficient analysis for

their use case. Streaming by a predetermined length of time, on the other hand, results in inconsistent numbers of tweets due to differences in tweet subject popularity. The second option of feeding the tweets into the tool is to use HDFS or Local Storage and just to point the program to the JSON file. That way, the processing of this data starts immediately, and the Python 2.7 is not required to run the program.

The second stage of the data flow is the back-end, written completely in Scala. The Scala version is 2.12.10. As I mentioned before, the program uses Spark, which has the version of 2.4.4. And to store the tweets while streaming the tool uses Hadoop 3.2.1. Out back-end consists of two parts, Queries Handler(HandleQueries.scala) and Sentiment Analyzer(Model.scala). Sentiment Analyzer is used only when we analyze the sentiment of the tweets, which is 3 queries out of 10 total. Comma Separated Values (CSV) format is used to move the data between back end and front end. For sentiment analysis the Naïve Bayes Model is used with the training done on a data set called Sentiment140. The model type is mmultinomial ad lambda = 1.0.

The third stage of the data flow is front end, which is also written in Scala. The GUI was tested on Ubuntu and works perfectly there. However, it would be a complex challenge to move this project from Ubuntu to Windows, which is one of the negative aspects of not using GUI. Another one is that the user will not be able to analyze the queries from their mobile device. However, with technology progresses, there are a lot of options for remote desktops for mobile devices, as well as virtual machines that can be used to run Ubuntu. The fxml file is used to be able to build the GUI with SceneBuilder, which increases the performance of development. We had to use Javafx 8 instead of Java 8 to be able to support ScalaFX. Please note that Front End and Back End are separated as requested, and the data is moved with CSV stored locally.

At the end, we were able to obtain 10 charts that can be used to monitor the trends of brands, stocks, and anything related to the market. The first 4 charts are bar charts, which shows the most popular hashtags and URLs by the number of tweets and followers of the user who wrote that tweet. An example of such a chart can be found below Figure 2.

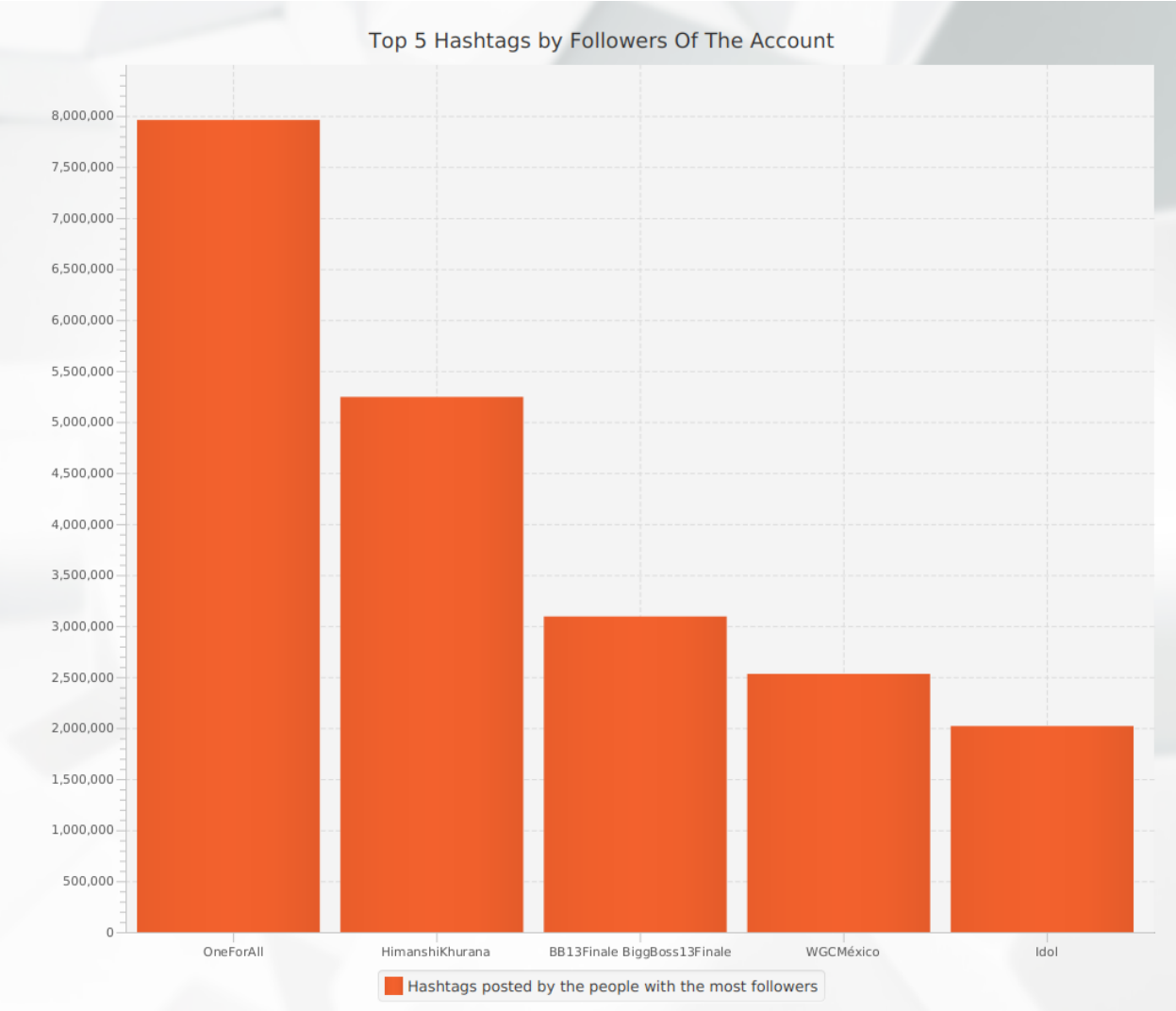

Figure 2 (Bar Chart with Top 5 Hashtags by the Number of Followers)

These four charts can be used to analyze the general trends of the market, what hashtags and URLs are mostly associated with the particular product or stock. The next chart is a Sentiment Pie Chart. It shows the ratio of positive, negative, and neutral tweets using the number of tweets

for each category. The following is an example of Pie Chart for a positive tweet.

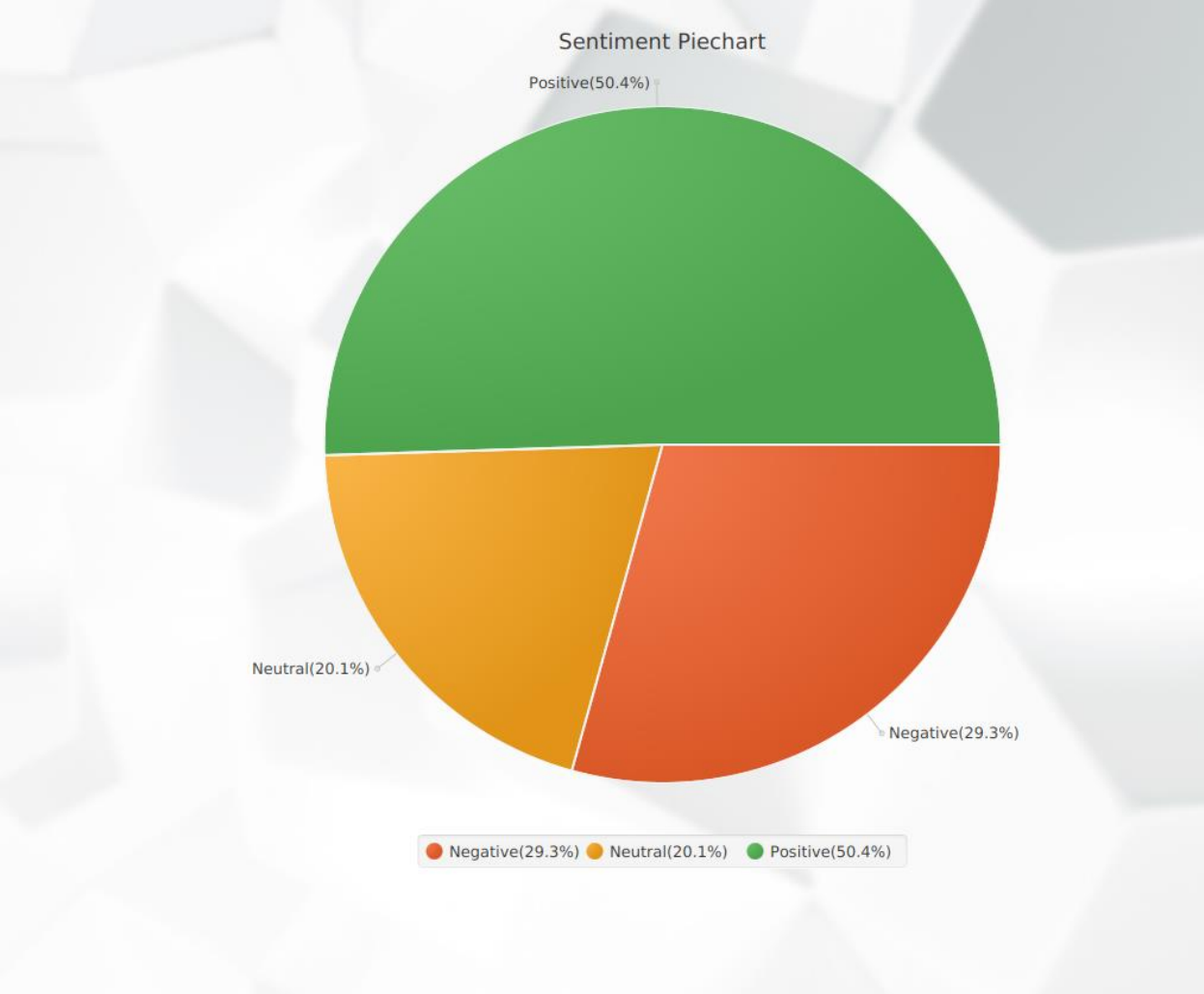

Figure 3 (Positive Tweet Sentiment Pie Chart)

All the analysis of sentiments in the current version can be done only on English tweets, since the model was trained on a labeled English dataset with English stop words. That can be improved, and the possible improvements will be discussed at the end of the paper.

Possibly the hardest to make and the most resource-consuming chart is the Sentiment Bubble Chart, which we describe next. First, let's take a look at the Figure 4, which not only

shows the diagram but also the whole GUI of the tool. The Twitter developer keys are hidden by the opaque gray box.

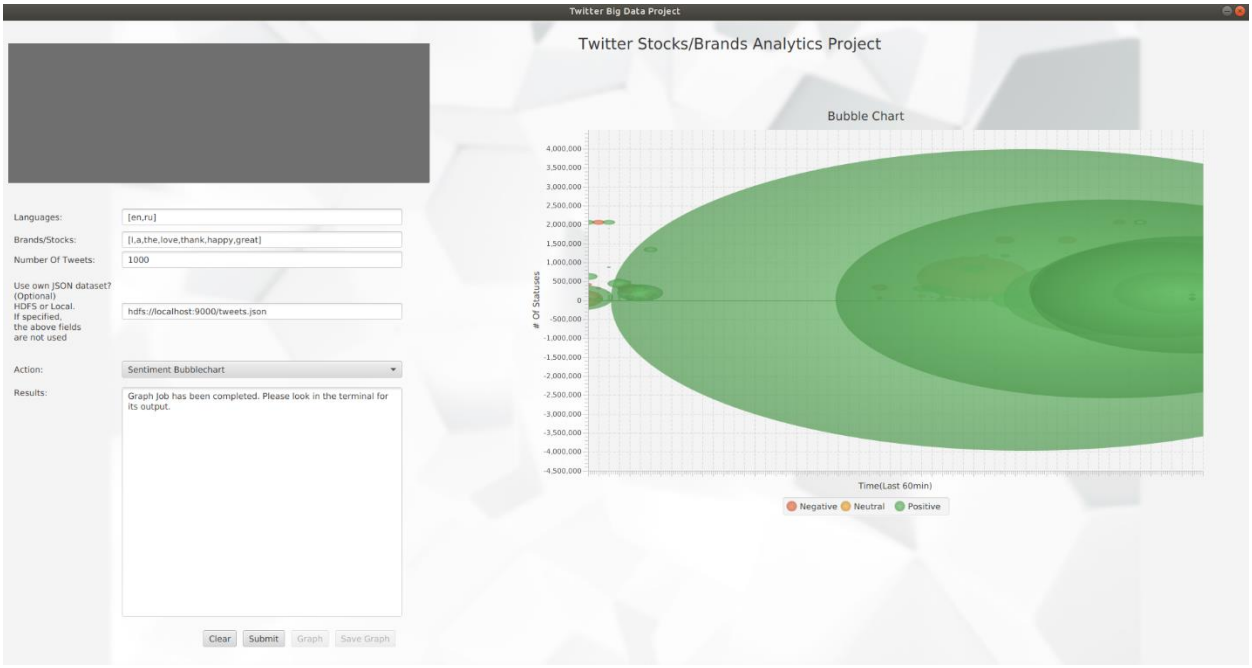

Figure 4(Positive Tweet Sentiment Bubble Chart)

This diagram is built using 100,000 tweets. The x-axis represents time, in minutes, for demonstration purposes. It would be more efficient to instead chart the x-axis by year, month, and date; however, for the demonstration purposes we use the last 60 minutes. The radius of each "bubble" is the number of followers of the user that wrote the particular tweet. This is very useful to measure the influence this user has compare to other users (more followers-more influence). The color of the bubble is the tweet sentiment, and the y axis is the number of tweets the person has written before he wrote this tweet.

Let's take a look at the Figure 4 one more time. As we can see, there is a field called "Results" in the GUI. This field displays messages that indicate a successful task, or any errors

the user encounters. First we have to press the Submit button to analyze the tweet, and then we have to press the Graph button to put it in Chart form. The Analytics process is done with the Scala Future interface, which provides a way to run threads in parallel. That was ideal for our case, so we even have a waiting bar displayed, which does not show progress since it is pointless in this type of application (different machines will have different results), but it shows that the program is still running.

The next two charts are similar to the Sentiment Pie Chart, but they also show the ratio of languages that were used to pull the tweets. There are two charts about languages: one is for number of tweets and another one is about the influence of the language, about the followers of the user who wrote the tweet in the particular language.

The last two charts are about timeline of the tweets. One of them shows the timeline of 3 lines on the Line Chart for sentiment, positive, neutral, negative. The y-axis is the number of followers of the user who wrote the particular tweet. This chart can be found below, Figure 5. And the last chart is also a line graph, but this time it illustrates the most influential languages. Since the number of languages can differ, we decided to use a Java list to store Scala data frames that we obtain for each language. Then we convert each Java list into a Scala iterator, and for each of the data frame stored, we basically plotted a line on the chart. We used a lamda function to convert each line of the data frame to the format of the Line Chart. The result can be found below as well, Figure 6

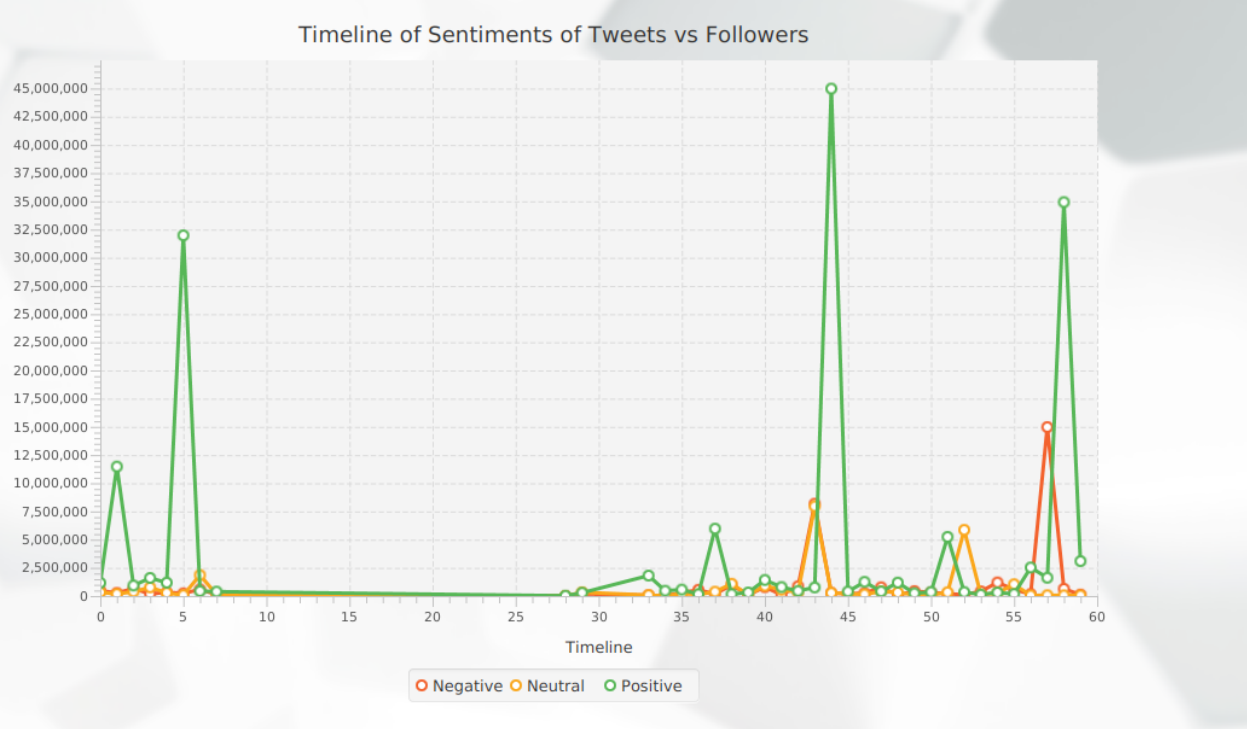

Figure 5(Timeline of Sentiments vs Followers)

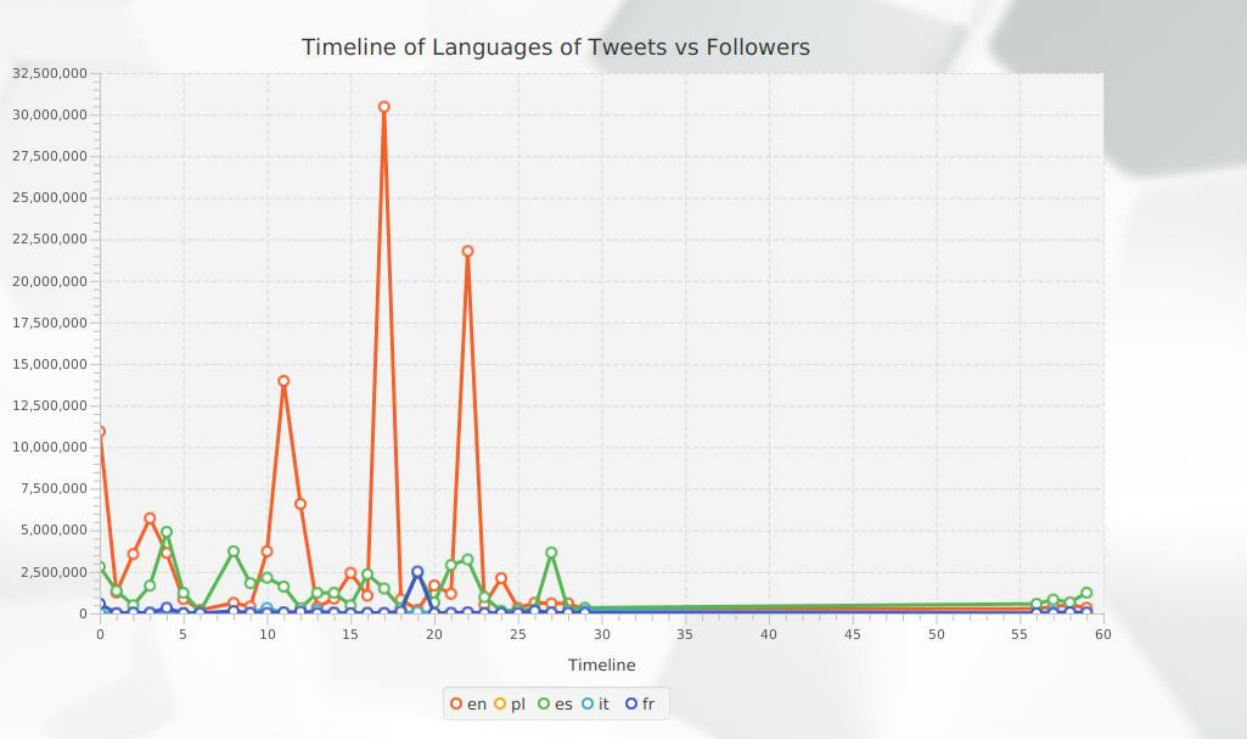

Figure 6(Timeline of Languages vs Followers)

Finally, let's discuss the results of this project and our approach. As was mentioned prior, one of the negative aspects of using ScalaFX is the poor platform compatibility. However, there are some benefits of using ScalaFX with SparkSQL instead of a web front end. For such a task where we want to display charts dynamically with different data, it is beneficial to not use a server, which decreases the performance of the machine since the server itself uses RAM, which is crucial for the analysis like that. Also, since we decided to perform the analysis separately on each query, the time for analysis of 100,000 of tweets of relevant data is low. And it is even less when we perform it again, since we don't have to reload spark session which takes a lot of time. It is already loaded after the first query and we don't have to wait for it to load again, which is a great performance boost. However, for the tasks like heat map, that would be beneficial to use web front end, since ScalaFX does not provide map. It does, however, provides a Web View that can be utilized for this purpose, but we did not get to test it. Overall, we believe we achieved great results and learned many new things about SparkSQL and Scala, that we did not know before.