# Twitter Sentiment Analysis for the word rugby

## Research Understanding Phase

Brief of the project was to acquire at least a year’s worth of Twitter data, having first tried to use snsscrape to acquire twitter data and having failed the decision was taken to use the archive.org and its historical twitter data.

The archive presented its own difficulties, initially it was easy to write a code snippet to download a range of zip and tar files containing archived twitter data, however the speed of download was very slow by modenr standards, it was suspected that the archive.org throttled file downloads, to test this another download was initiated and the speed of the download compared, both downloads were happening at the same speed, having seen this multiple copies of the download code were setup to run at the same time. The initial code used can be seen in the file DownloadTwitterData.ipynb.

However even splitting the download code into multiple notebooks proved to be very slow, an experiment was made to see how the download would work if the code was converted into a pyhton file, this experiment proved to be more successful, the pyhton file showed up to 3 times as fast at downloading a file, the code can be found in the file getTwitterdata.py; multiple copies of this file were created each with different dates ranges, and the files were set to copying files from the archive.org.

A final experiment was tried, an Azure account was setup using free Azure student credits, then using the Azure software development kit, a python file was written that copied from the Archive.org into Azure blob storage directly, this proved to be the quickest method at getting files from the archive.org. Once the files were in blob stroage, Azure data explorer was used to download the files to a computer hard drive. The downside to this method is that it cost money, all of the free credits were used up in the copying from archive.org and in storage costs for the Azure blob storage. The account and the blob storage are no longer accessible unless credit is added to the account. This meant that only the files downloaded from Azure are available to the project. One other issue with using this method to copy files was that the archive.org uses url redirects and the python package beautifulsoup was needed to find the ulitmate destination url for each zip and tar file.

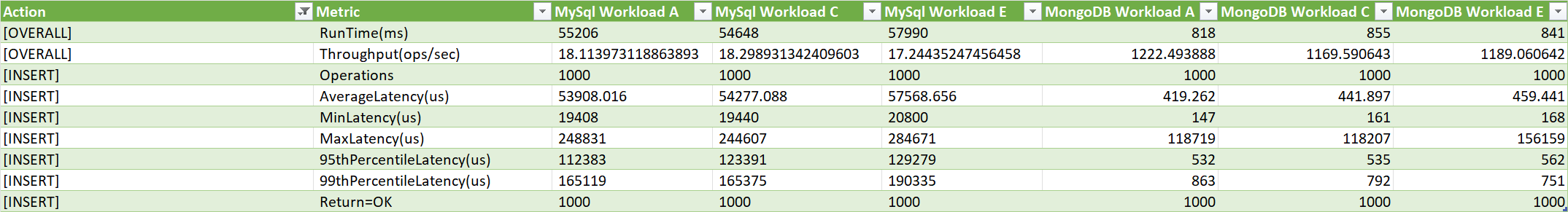
baseurl = "<https://archive.org/download/archiveteam-twitter-stream->" + str\_Year + '-' + str\_Month

r = requests.get(baseurl)  
  
soup = BeautifulSoup(r.content)  
  
soup = soup.find('table')  
  
soup = soup.find\_all('a')  
  
for element in soup:  
  
 dest = 'E:/TwitterStream'  
  
 lnkurl = element.get('href')

### Start of Big Data - MongoDB chosen

The next step was to understand how the files were archived, the zip and tar files had different structures with the zip files having multiple folders in their structure, a brute force method was adapted to get the data out of the files and into a MongoDB database.

MongoDB was chosen as the destination because of the outcome of Yahoo Cloud Serving Benchmark (YCSB) results seen below (citation)



Here we can see the results of a comparison of MongoDB and MySQL over 3 different work loads, in all of them MongoDB comes out best, with the shortest runtimes and highest throughputs across all three work loads. The workloads were chosen to mimic the work that would be happening during this project, work load A gives the results of an update heavy process, with 50% read and write operations, this was measured because it would mimic the initial phase of the project, where data would be written to the database and at the same time data discovery and querying would be started. Workload C is a read only workload and was chosen because once all of the data had been loaded then read times would be become critical for the project. Workload E was chosen as it mimics how a social network is organised and it was assumed that following tweets and retweets might be part of the analysis of the data.

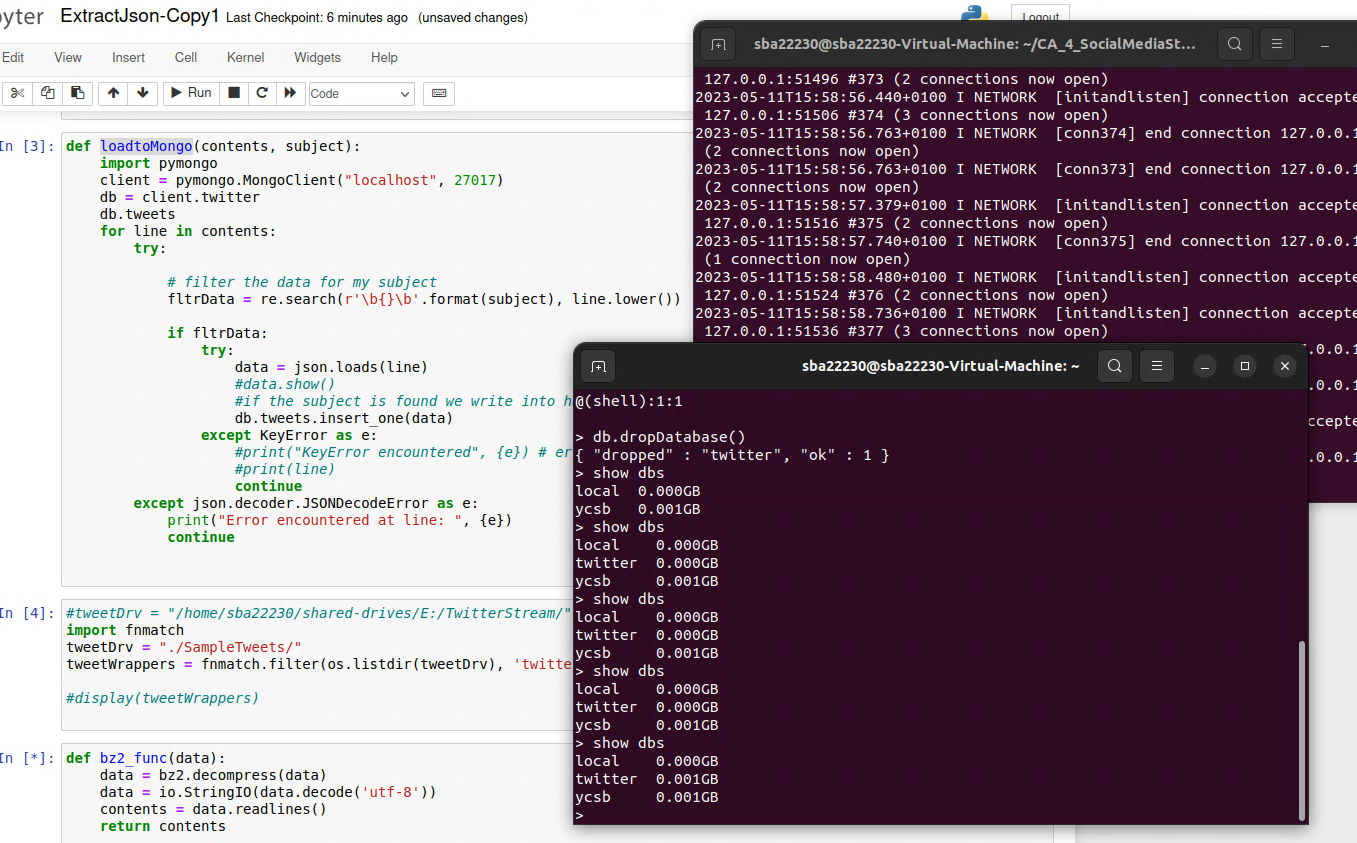
### Data gathering

Once the files were downloading the extraction phase was started, as the files were on a Windows drive external to the Linux VM, a brute force method was used to read the files and iterate through their structures until the JSON snippets were found, then each JSON snippet was read and queried for the word 'Rugby' and snippets that contained the subject were inserted into the MongoDB. Below is the function that was used to query the JSON objects extracted from the compressed files, and each line that contained the subject word was inserted into the MongoDB.

def loadtoMongo(contents, subject):

import pymongo  
client = pymongo.MongoClient("localhost", 27017)  
db = client.twitter  
db.tweets  
for line in contents:  
 try:  
   
 # filter the data for my subject  
 fltrData = re.search(r'\b{}\b'.format(subject), line.lower())  
  
 if fltrData:  
 try:  
 data = json.loads(line)  
 #data.show()  
 #if the subject is found we write into hadoop  
 db.tweets.insert\_one(data)  
 except KeyError as e:  
 #print("KeyError encountered", {e}) # error is encountered mainly due to deleted tweets   
 #print(line)  
 continue  
 except json.decoder.JSONDecodeError as e:  
 print("Error encountered at line: ", {e})  
 continue

Below is a screenshot of the process at work, in the image we can see the size of MongoDB increasing, in the other command window we can multiple connections being opened and closed as the notebook cell is running. To speed up the extraction and loading of data from the shared drive into MongoDB, multiple copies of the notebook were created, and each had different ranges in them so that multiple files were being processed at the same time.



At this stage a decision was taken to extract any and all lines that had the subject word in them, this was done because it would be easier to query for tweets with the subject word in tweet text in the next phase of the project using PySpark; it was also deemed quicker to query the entire tweet in one line rather than query subsections of a tweet.

## Data Understanding Phase

With the tweets in MongoDB all work moved directly onto the Linux VM, the PySpark instance on the virtual machine was connected to the MongoDB and the database was queried. The first part was to understand was the structure of a tweet, once the structure was visualized by the print schema method, a temporary PySpark SQL view was created and then some preliminary data was queried from it.

One issue that was over come was that PySpark tries a number of shortcuts, one of these is the infer schema is set to read only a certain number of rows and infer the data structure from these rows, this caused issues with the query when we went to read the entire database of data, as the inference had chosen the wrong schema for some of the fields incorrectly, the fix for this was to set the infer schema to false, this made PySpark read the entire database and not infer the values of fields from the first set of fields.

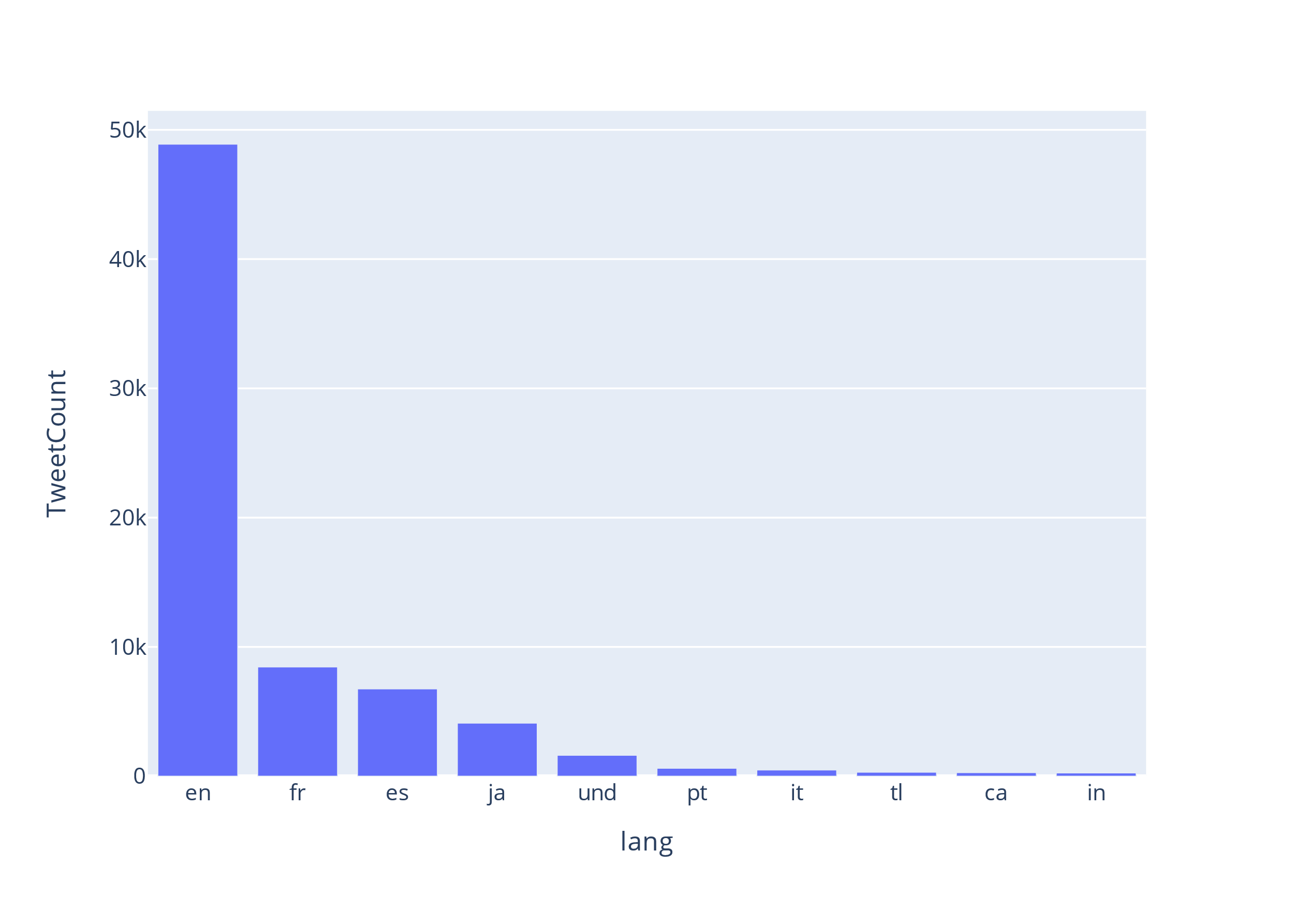
It is easier to use SQL statements and PySpark to clean the data rather than writing queries in MongoDB.

Exploratory data analysis

1: How many tweets in the DB all together: 72807

2: How many tweets by language

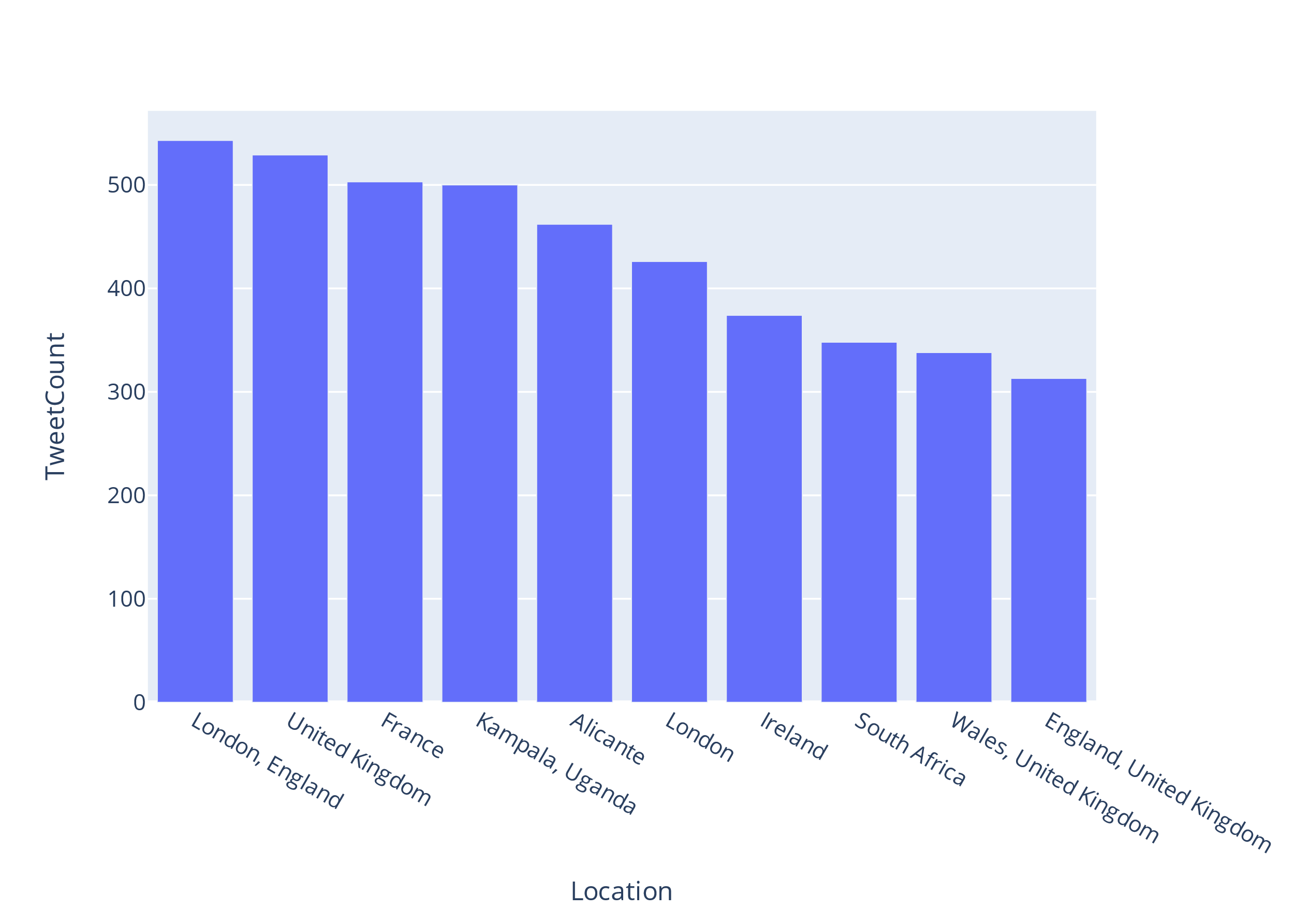
|  |  |
| --- | --- |
| Language | Tweet Count |
| en | 48892 |
| fr | 8450 |
| es | 6754 |
| ja | 4105 |
| und | 1618 |
| pt | 614 |
| it | 480 |
| tl | 309 |
| ca | 274 |
| in | 251 |



Here we can see that the majority of tweets about Rugby are in the English language, followed by French, Spanish and Japanese with a sizable number of tweets having an undefined language.

3: How many tweets by location

|  |  |
| --- | --- |
| Location | Tweet Count |
| null | 24804 |
| London, England | 543 |
| United Kingdom | 529 |
| France | 503 |
| Kampala, Uganda | 500 |
| Alicante | 462 |
| London | 426 |
| Ireland | 374 |
| South Africa | 348 |
| Wales, United Kin... | 338 |



Here we can see the user locations that have tweeted about rugby, note a larger number of users do not have their location set in Twitter as we see 24,804 tweets with null as the location.

The next step was to limit the dataset to English texts and tweets with the rugby in the text

How many tweets in the English language dataset: 48892

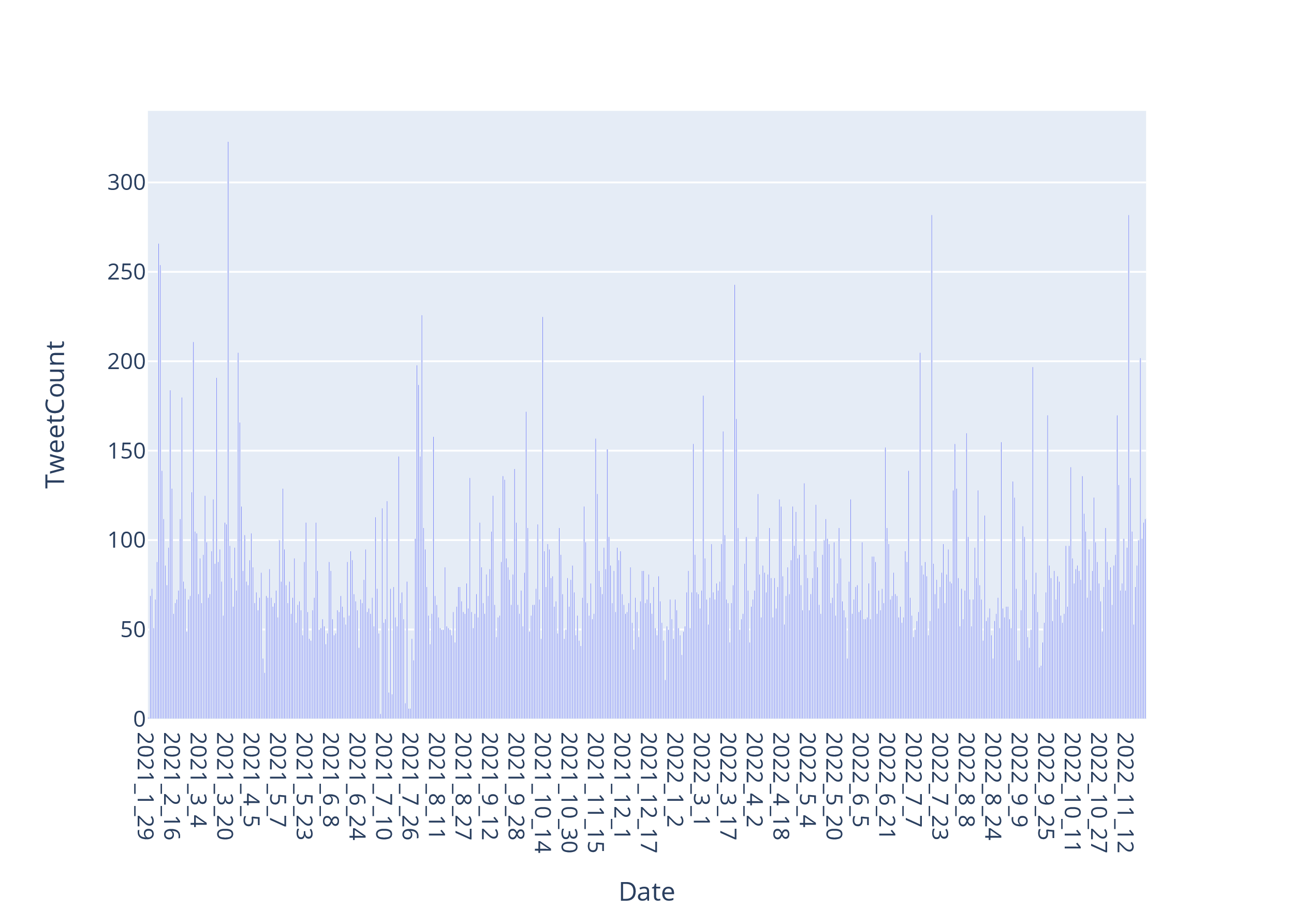
How many English languages are there categorized by user location.

|  |  |
| --- | --- |
| Location | Tweet Count |
| null | 15507 |
| United Kingdom | 518 |
| London, England | 514 |
| Kampala, Uganda | 467 |
| London | 410 |
| Ireland | 356 |
| South Africa | 327 |
| Wales, United Kin... | 315 |
| England, United K... | 301 |
| Cape Town, South ... | 264 |

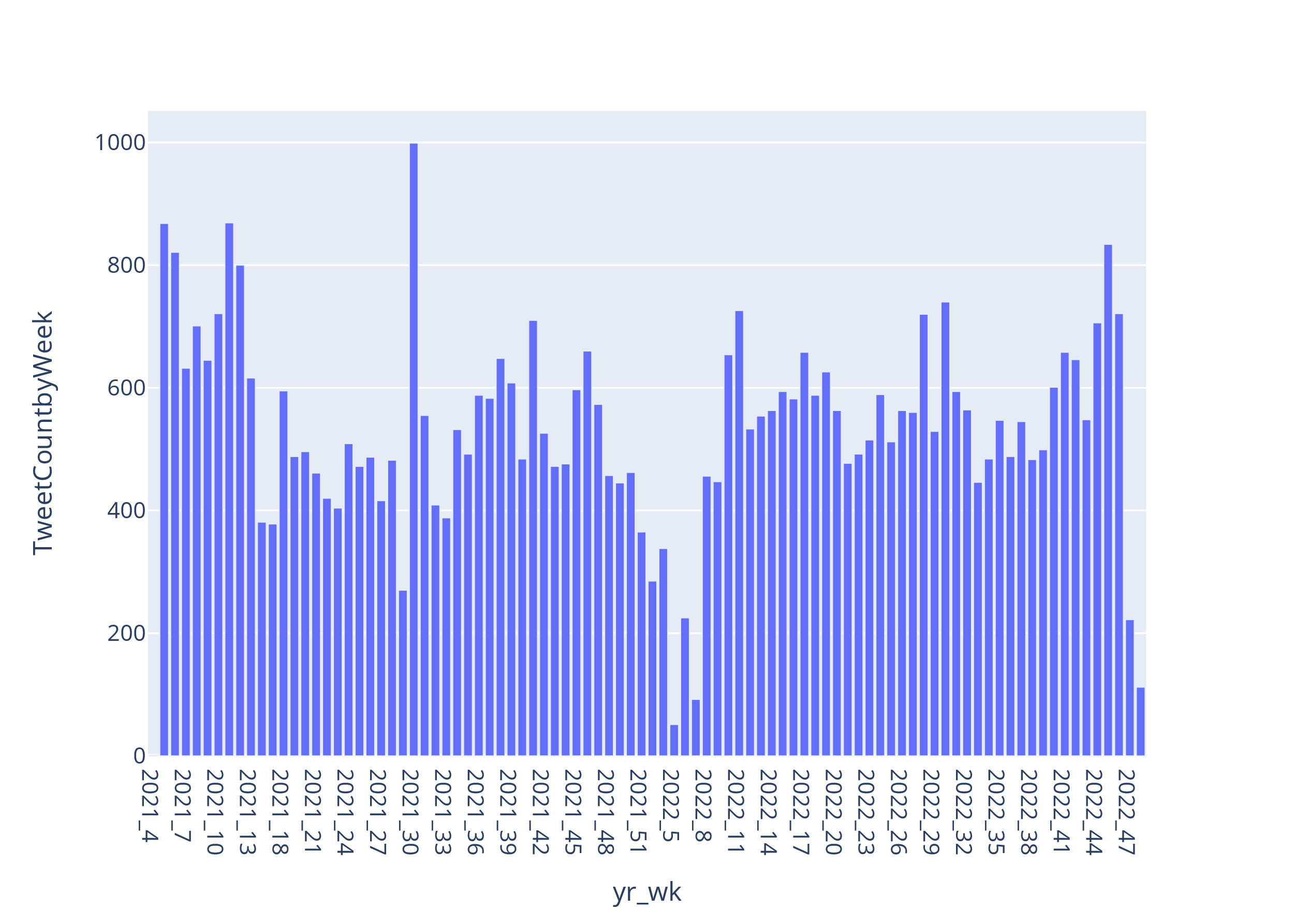
How do the tweet counts break down by day:

|  |  |
| --- | --- |
| Date | Tweet Count |
| 2021\_1\_29 | 1 |
| 2021\_2\_1 | 69 |
| 2021\_2\_2 | 73 |
| 2021\_2\_3 | 51 |
| 2021\_2\_4 | 67 |
| 2021\_2\_5 | 88 |
| 2021\_2\_6 | 266 |
| 2021\_2\_7 | 254 |
| 2021\_2\_8 | 139 |
| 2021\_2\_9 | 112 |
| 2021\_2\_10 | 86 |
| 2021\_2\_11 | 75 |
| 2021\_2\_12 | 96 |
| 2021\_2\_13 | 184 |
| 2021\_2\_14 | 129 |
| 2021\_2\_15 | 59 |
| 2021\_2\_16 | 65 |
| 2021\_2\_17 | 67 |
| 2021\_2\_18 | 72 |
| 2021\_2\_19 | 112 |

only showing top 20 rows

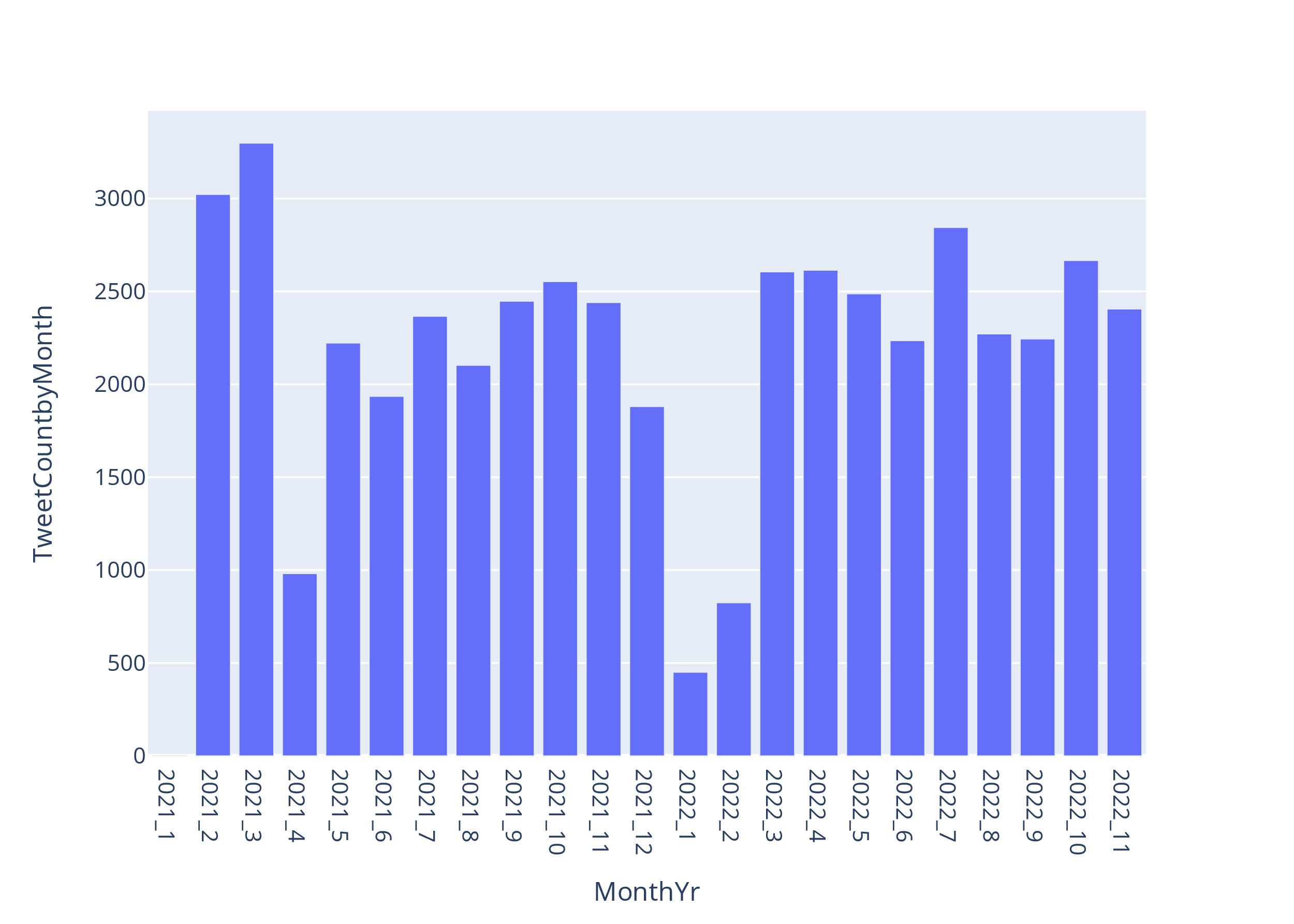


It looks like we have data for most of the days in the data set, but this is misleading because some of the files on the archive.org data set are corrupted, and some months are missing entire weeks. As we move further into the analysis we will see these missing data more clearly.



By Year and week number tweet counts

The plot above show the number of tweets by week number over the years 2021 and 2022, things to note are the gaps at the beginning of 2021 and the gap in May 2022



Plot showing the tweet counts by year and month

## Data Preparation Phase

### Text clean up in PySpark.

The first step was to limit the dataset to tweet id and the tweet text.

|  |  |
| --- | --- |
| id | text |
| 1444274428221734923 | What a game of ru... |
| 1477389703984865287 | RT @labour\_histor... |
| 1477323861788086273 | RT @rugby\_sport\_x... |
| 1477243301812183043 | RT @scarlets\_rugb... |
| 1430144262642061313 | RT @premrugby: We... |

only showing top 5 rows

Count the number of words in the tweets.

|  |  |
| --- | --- |
| Word | count |
| the | 25866 |
| RT | 24751 |
| rugby | 24389 |
| to | 16333 |
| a | 14673 |
| in | 11306 |
| of | 11130 |
| and | 10737 |
| for | 8756 |
| is | 7034 |
| on | 5671 |
| I | 5522 |
| at | 4442 |
| you | 4143 |
| this | 4107 |
| with | 4051 |
|  | 3820 |
| that | 3302 |
| are | 3197 |
| from | 3192 |

only showing top 20 rows

Further analysis was carried out on the text, we counted the number of characters including spaces in each tweet, we checked for upper case letters in the tweets.Nothing showed for the check on uppercase letters, there were no tweets composed completely of uppercase letters. We checked for numbers in the tweets and this check did not advance our understanding of the data, a lot of twitter names have numbers in them.

After these checks were performed it was decided to perform the preliminary text cleaning in PySpark, using a mixture of PySpark SQL queries and some python packages that could run in the PySpark environment. Using PySpark SQL, we were able to remove all non alpha characters, including numeric characters via a regular expression. Then we set all of the tweets to lowercase text and removed but starting and trailing spaces from the text.

The next step in process was to tokenize and stem the tweets, in PySpark, for two reasons: one to learn how to do it, and two to write the code so that it could be used in later projects that might be working with more data. The reason we went down this route was because of a misunderstanding, we misunderstood the brief, originally we thought that we would have to build our own model to categorize the tweets with sentiment labels, when we reread the brief we decided that we needed to label the tweets we had collected with sentiment scores using other prebuilt sentiment models.

## Sentiment Labeling

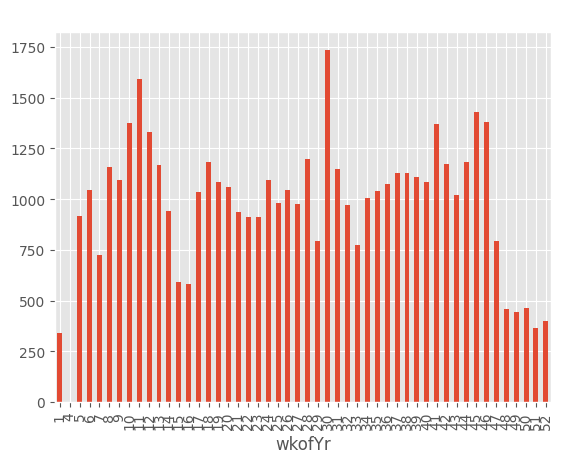
## For the sentiment labeling of the collected Rugby tweets, we decided to use 2 different packages to cateogrise the sentiment of each tweet, the package we used was (*textblob.en.sentiments — TextBlob 0.16.0 documentation*, no date) and Valence Aware Dictionary and sEntiment Reasoner (VADER)(‘Python | Sentiment Analysis using VADER’, 2019), the reason for choosing 2 different packages is because they work differently, TextBlob is using a Naïve Bayes classifier and the model is trained on Movie Reviews (‘Twitter Sentiment Analysis using Python’, 2017; https://twitter.com, 2020), whereas the Vader model is a rule based model using a lexicon and was specifically built for sentiment analysis on social media text (‘Python | Sentiment Analysis using VADER’, 2019). Once the tweets are scored by the two models we built a pseudo Likert Score (‘Likert scale’, 2023) for the tweets, we did this by adding the score from TextBlob and the score from VADER, if the value was greater than 1, we set the Likert value at ‘Very Positive’, if it was greater than 0 and less than 1 we gave it a score of ‘Positive’, if it was 0 we gave it a score of ‘Neutral’ and if it is less than 0 and greater than -1 we gave it a score of ‘Negative’ and if it was less than -1 then we gave a score of ‘Very Negative’, the results of the scores looked like this, as we can see, that the number of positive and very positive tweets out number the negative and very negative tweets, meaning that tweets about Rugby are skewed towards the positive end of the scale, as we worked with the data this scale was not used due to other tests ruling it out.

Number of tweets by Likert Scale:

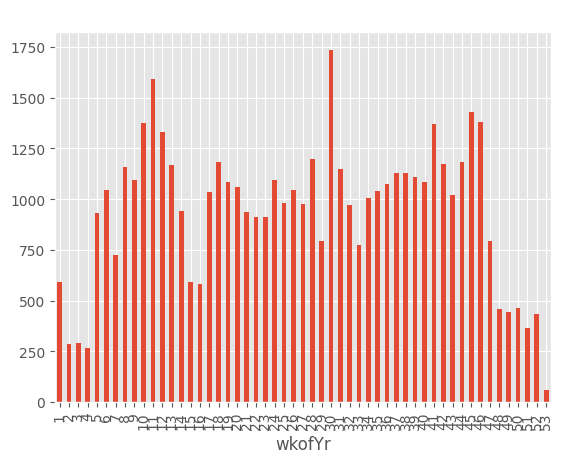
|  |  |
| --- | --- |
| Very Positive | 8372 |
| Positive | 20516 |
| Neutral | 10662 |
| Negative | 9370 |
| Very Negative | 806 |

## Building the model - Modeling Phase

Once the tweets had been given a sentiment score we joined the text and sentiment scores data with other tweet data such as the date and then we plotted out the data.



As we can see January is missing data from a couple of weeks, this is an issue with the data source and can't be rectified by going back to the data. The plan we followed was too sample 1200 rows from the data frame and add them into January 2021. We made this decision as at this stage we had daily tweets and we thought that sampling from the data set would introduce less bias than if we used any other calculated method.



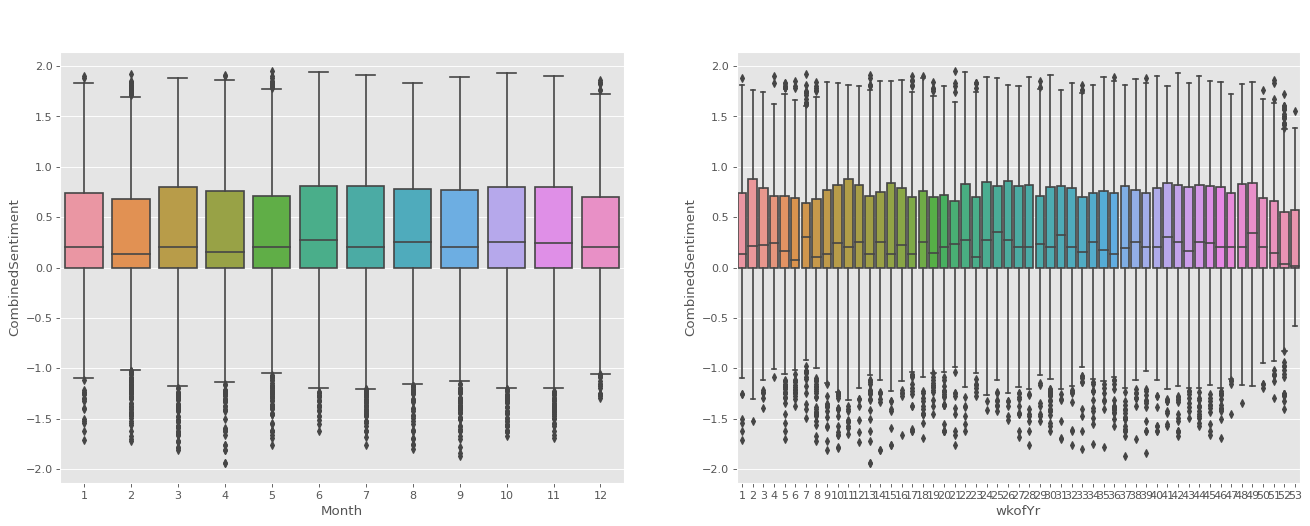
Later on when we had consolidated the data into weeks, we noticed that 5 weeks were missing data, and at that stage we used Pandas interpolation method to fill in the missing data, we chose this because it was fast and straightforward.

Table showing the missing rows of data

txbSentiment vaderSentiment CombinedSentiment  
date   
2021-01-31 00:00:00+00:00 False False False  
2021-02-07 00:00:00+00:00 False False False  
2021-02-14 00:00:00+00:00 False False False  
2021-02-21 00:00:00+00:00 False False False  
2021-02-28 00:00:00+00:00 False False False

### Start of Time Series Analysis

The box plots show that our data has neither seasonality in the month view nor in the weekly view.



The next step was to look at the patterns in the time series, to do this we used Additive Decomposition on this time series for each of the three scores that are in the data set; they the TextBlob score, labelled txbSentiment, VADER score labelled vaderSentiment, and finally the combined score labelled CombinedSentiment. The results are displayed below, and from them it can be seen none of sentiment scores has a trend, and none exhibit seasonality, and for TextBlob score the residuals are showing a more random pattern when compared to the other sentiment scores.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Next step in the analysis was to look at whether these time series data were stationary. To do that we used Augmented Dickey Fuller test (ADF Test)

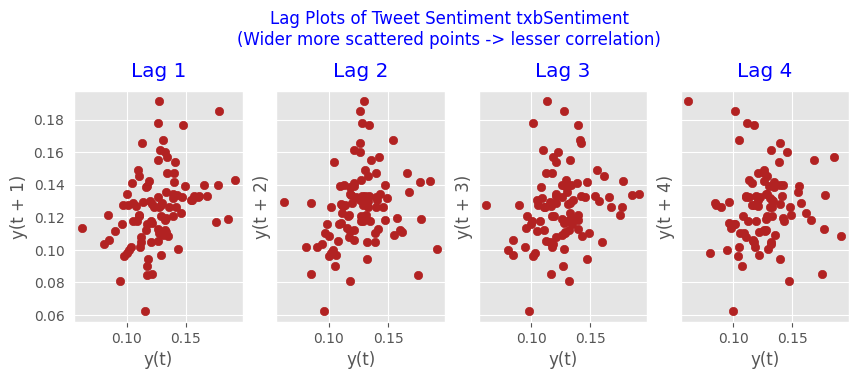
Results:

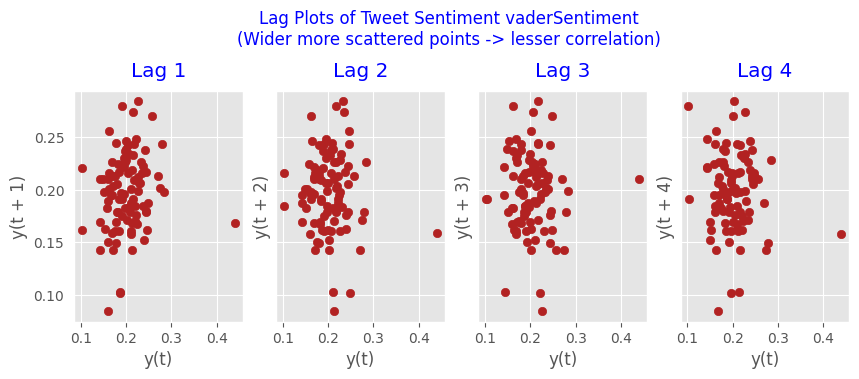
|  |  |  |
| --- | --- | --- |
| TextBlob Sentiment ADF Test ADF Statistic: -6.863973307717687 p-value: 1.5763050593574846e-09 Probably Stationary Critial Values:  1%, -3.5011373281819504 Critial Values:  5%, -2.8924800524857854 Critial Values:  10%, -2.5832749307479226 | Vader Sentiment ADF Test ADF Statistic: -2.412244689099504 p-value: 0.13831021826489986 Probably not Stationary Critial Values:  1%, -3.506057133647011 Critial Values:  5%, -2.8946066061911946 Critial Values:  10%, -2.5844100201994697 | Combined Sentiment ADF Test ADF Statistic: -8.946784801222686 p-value: 8.943946703654746e-15 Probably Stationary Critial Values:  1%, -3.5011373281819504 Critial Values:  5%, -2.8924800524857854 Critial Values:  10%, -2.5832749307479226 |

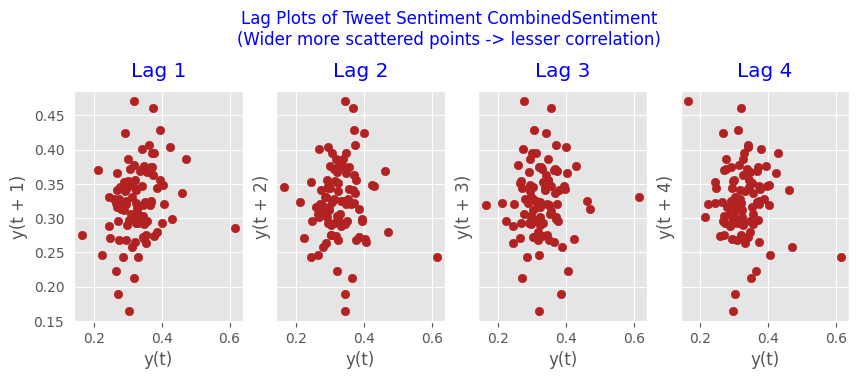
Ideally we want the time series to be stationary, as a stationary time series is easier to predict, and the forecasts are more reliable. (CCT) ADF test is the most commonly used test as can be seen from the table above both TextBlob and Combined Sentiment are probably stationary. The next test was for Autocorrelation and Partial Autocorrelation, as we can see TextBlob is the only series that has point outside of the confidence intervals, indicating that there might be some correlation between the series and its lags (*A Gentle Introduction to Autocorrelation and Partial Autocorrelation - MachineLearningMastery.com*, no date)

|  |  |
| --- | --- |
| TextBlob |  |
| VADER |  |
| Combined |  |

We follow up the ACF plots with scatter plots showing the lag correlation.







The scatter plots show us that TextBlob sentiment has no discernible pattern at any of the lags. At this point we will work exclusively with the TextBlob sentiment score, it has passed the various tests above.

“The more regular and repeatable patterns a time series has, the easier it is to forecast. The ‘Approximate Entropy’ can be used to quantify the regularity and unpredictability of fluctuations in a time series” Next is to look at the entropy of the data. The approximate entropy of the data is 1.7749 – which is not especially small number.

Below are the results of the Granger Causality tests, these show that for both weekly or monthly value that we can accept the null hypothesis that Month and Week both do not Granger cause the TextBlob sentiment score.

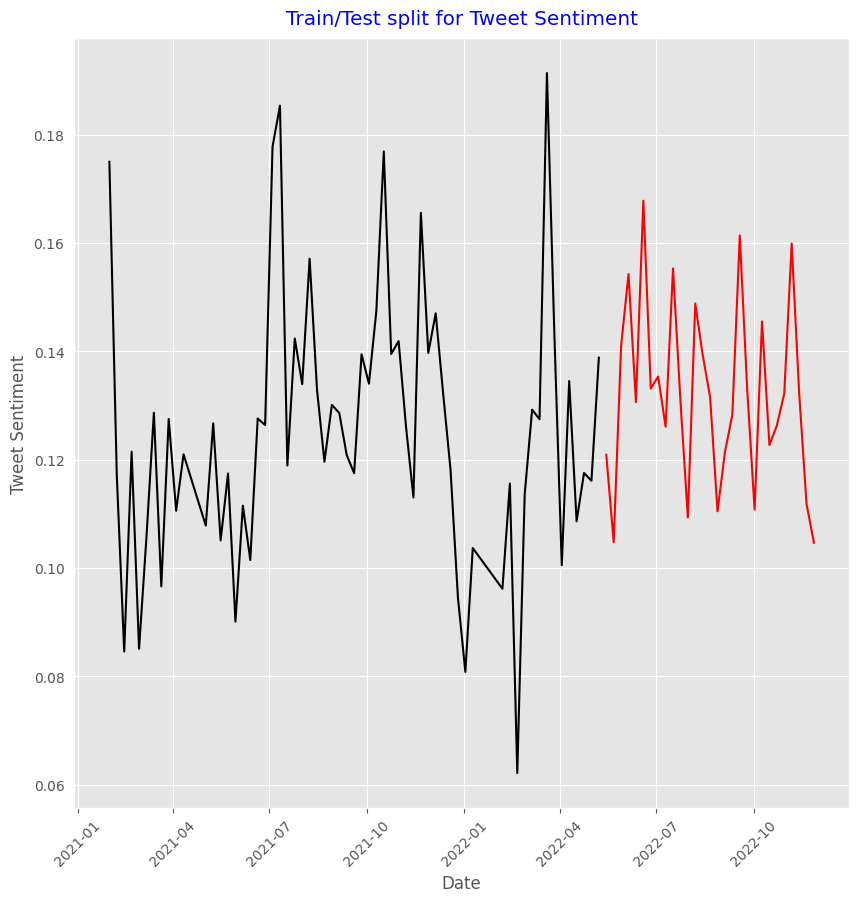
grangercausalitytests(df[['txbSentiment', 'month']], maxlag=4)

Granger Causality  
number of lags (no zero) 1  
ssr based F test: F=4.9300 , p=0.0289 , df\_denom=92, df\_num=1  
ssr based chi2 test: chi2=5.0907 , p=0.0241 , df=1  
likelihood ratio test: chi2=4.9590 , p=0.0260 , df=1  
parameter F test: F=4.9300 , p=0.0289 , df\_denom=92, df\_num=1  
  
Granger Causality  
number of lags (no zero) 2  
ssr based F test: F=1.4996 , p=0.2288 , df\_denom=89, df\_num=2  
ssr based chi2 test: chi2=3.1676 , p=0.2052 , df=2  
likelihood ratio test: chi2=3.1154 , p=0.2106 , df=2  
parameter F test: F=1.4996 , p=0.2288 , df\_denom=89, df\_num=2  
  
Granger Causality  
number of lags (no zero) 3  
ssr based F test: F=0.5518 , p=0.6483 , df\_denom=86, df\_num=3  
ssr based chi2 test: chi2=1.7901 , p=0.6171 , df=3  
likelihood ratio test: chi2=1.7731 , p=0.6208 , df=3  
parameter F test: F=0.5518 , p=0.6483 , df\_denom=86, df\_num=3  
  
Granger Causality  
number of lags (no zero) 4  
ssr based F test: F=0.6273 , p=0.6444 , df\_denom=83, df\_num=4  
ssr based chi2 test: chi2=2.7812 , p=0.5951 , df=4  
likelihood ratio test: chi2=2.7400 , p=0.6022 , df=4  
parameter F test: F=0.6273 , p=0.6444 , df\_denom=83, df\_num=4

grangercausalitytests(df[['txbSentiment', 'week']], maxlag=4)

Granger Causality  
number of lags (no zero) 1  
ssr based F test: F=3.6819 , p=0.0581 , df\_denom=92, df\_num=1  
ssr based chi2 test: chi2=3.8019 , p=0.0512 , df=1  
likelihood ratio test: chi2=3.7278 , p=0.0535 , df=1  
parameter F test: F=3.6819 , p=0.0581 , df\_denom=92, df\_num=1  
  
Granger Causality  
number of lags (no zero) 2  
ssr based F test: F=1.1017 , p=0.3368 , df\_denom=89, df\_num=2  
ssr based chi2 test: chi2=2.3271 , p=0.3124 , df=2  
likelihood ratio test: chi2=2.2988 , p=0.3168 , df=2  
parameter F test: F=1.1017 , p=0.3368 , df\_denom=89, df\_num=2  
  
Granger Causality  
number of lags (no zero) 3  
ssr based F test: F=0.3143 , p=0.8150 , df\_denom=86, df\_num=3  
ssr based chi2 test: chi2=1.0196 , p=0.7965 , df=3  
likelihood ratio test: chi2=1.0140 , p=0.7979 , df=3  
parameter F test: F=0.3143 , p=0.8150 , df\_denom=86, df\_num=3  
  
Granger Causality  
number of lags (no zero) 4  
ssr based F test: F=0.7039 , p=0.5915 , df\_denom=83, df\_num=4  
ssr based chi2 test: chi2=3.1210 , p=0.5378 , df=4  
likelihood ratio test: chi2=3.0693 , p=0.5463 , df=4  
parameter F test: F=0.7039 , p=0.5915 , df\_denom=83, df\_num=4

### Predicting Sentiment

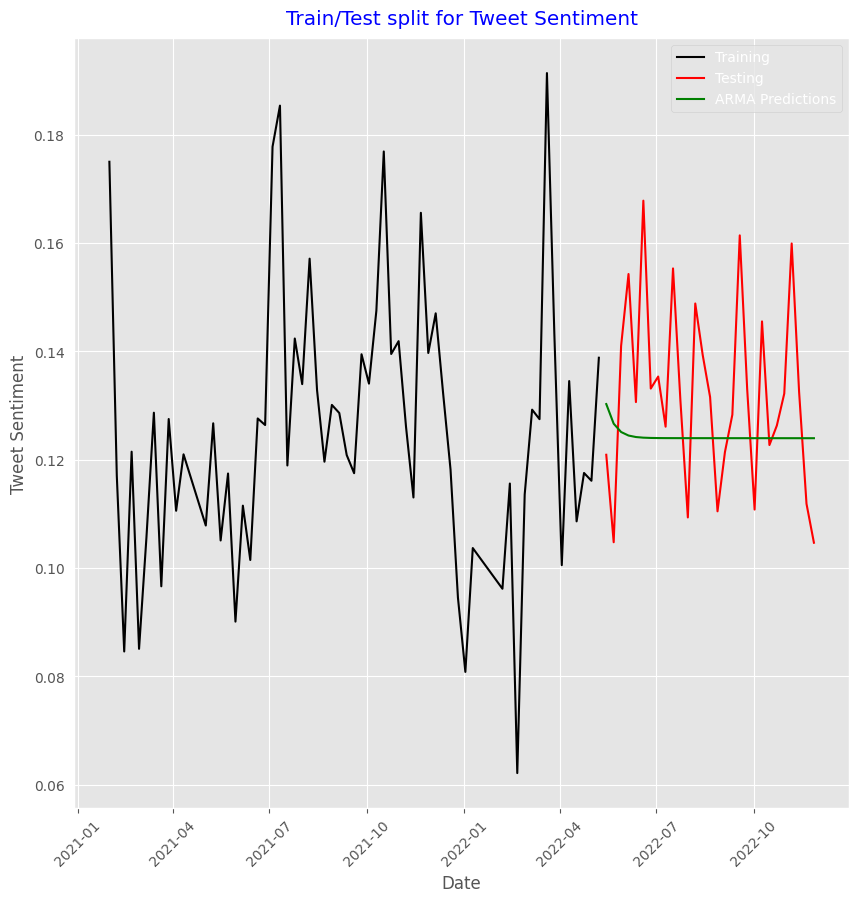


## Evaluation Phase

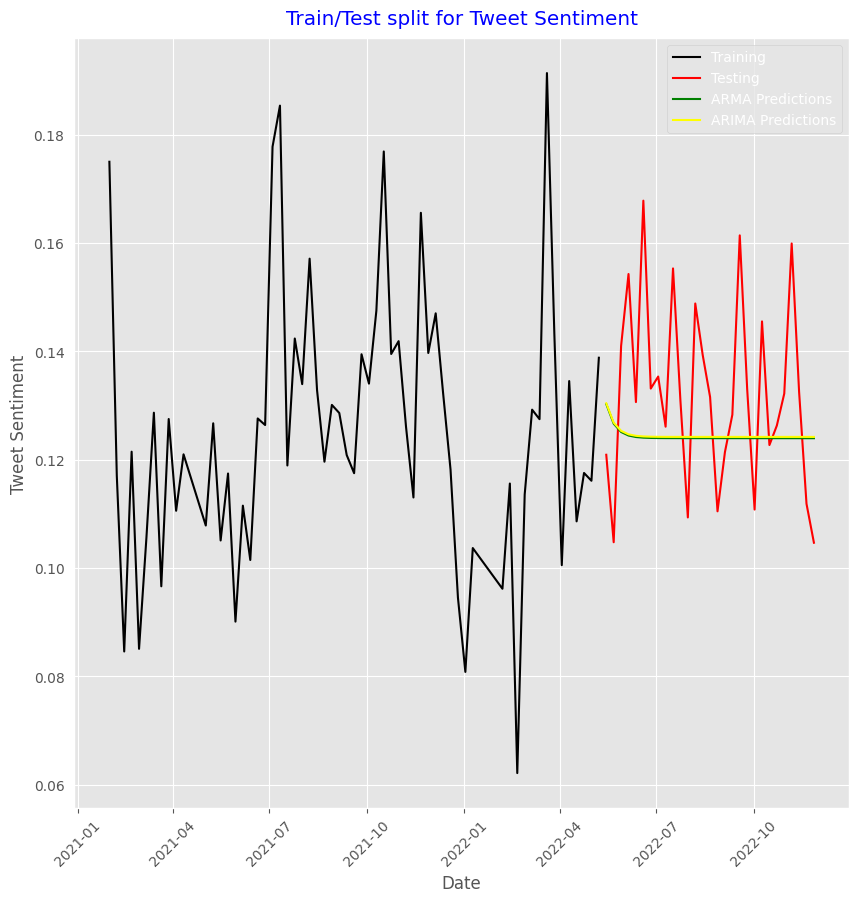
### Grid Search for the best model

For the evaluation phase, we set up 3 different time series models, and we used a grid search pattern documented at the web page <https://towardsdev.com/auto-arima-hyperparameter-search-ab991a21c2bd> the results of the Grid search are in the notebook and the best models with their predictions are shown below.

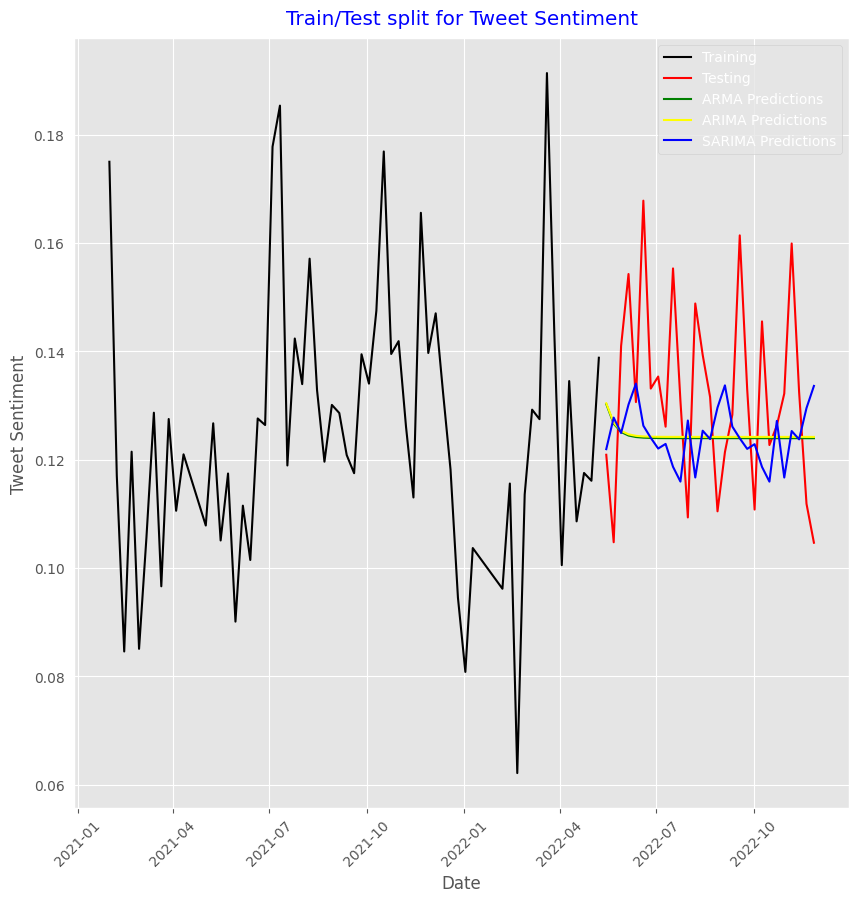
Arma



RMSE for ARMA model: 0.031672380658061355



ARIMA RMSE: 0.03185068818081811



SARIMA RMSE: 0.031421142238327336

## Deployment Phase

### Based on root mean squared error (RMSE) the Sarima model is the most accurate and it is the model we used in our dashboard. We also looked at Akaike’s Information Criteria (AIC) measure as well, the AIC penalizes too many features being added to a model, how in most of the grid search, both scores minimized on the same values.

Below is an example of the models predictions for 1 week, 1 month and 3 months out.

