# 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')

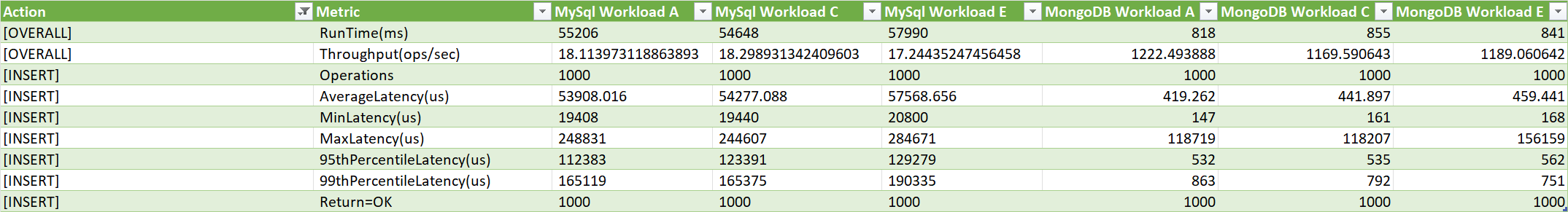
from urllib import request as rq  
import pandas as pd  
import os  
from datetime import datetime as dt  
import calendar  
from bs4 import BeautifulSoup # for web scraping  
import requests # for web scraping

year = {2021, 2022}  
  
for y in year:  
 if y == 2021:  
 month = range(1, 13, 1)  
 else:  
 month = range(1, 12, 1)  
   
 for m in month:  
 str\_Year = str(y)  
 if m < 10:  
 str\_Month = '0' + str(m)  
 else:  
 str\_Month = str(m)  
  
 baseurl = "https://archive.org/download/archiveteam-twitter-stream-" + str\_Year + '-' + str\_Month  
  
# https://archive.org/download/archiveteam-twitter-stream-2021-06/twitter-stream-2021-06-14.zip  
 # resorted to web scraping because there are too many variables to statically code for.   
 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')  
 # only download the files that are zip or tar   
 if lnkurl.endswith('.zip') or lnkurl.endswith('.tar'):  
 dest = dest + '/' + lnkurl  
 lnkurl = baseurl + '/' + lnkurl  
 print('Downloading: ' + lnkurl)  
   
 if os.path.exists(dest):  
 print('File exists: ' + dest)  
 continue  
 else:  
 # download the file  
 try:  
 rq.urlretrieve(lnkurl, dest)  
 except:  
 print('Error: ' + lnkurl)  
 continue  
 print('Downloaded: ' + lnkurl)

### 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 accross 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 proccess, 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.

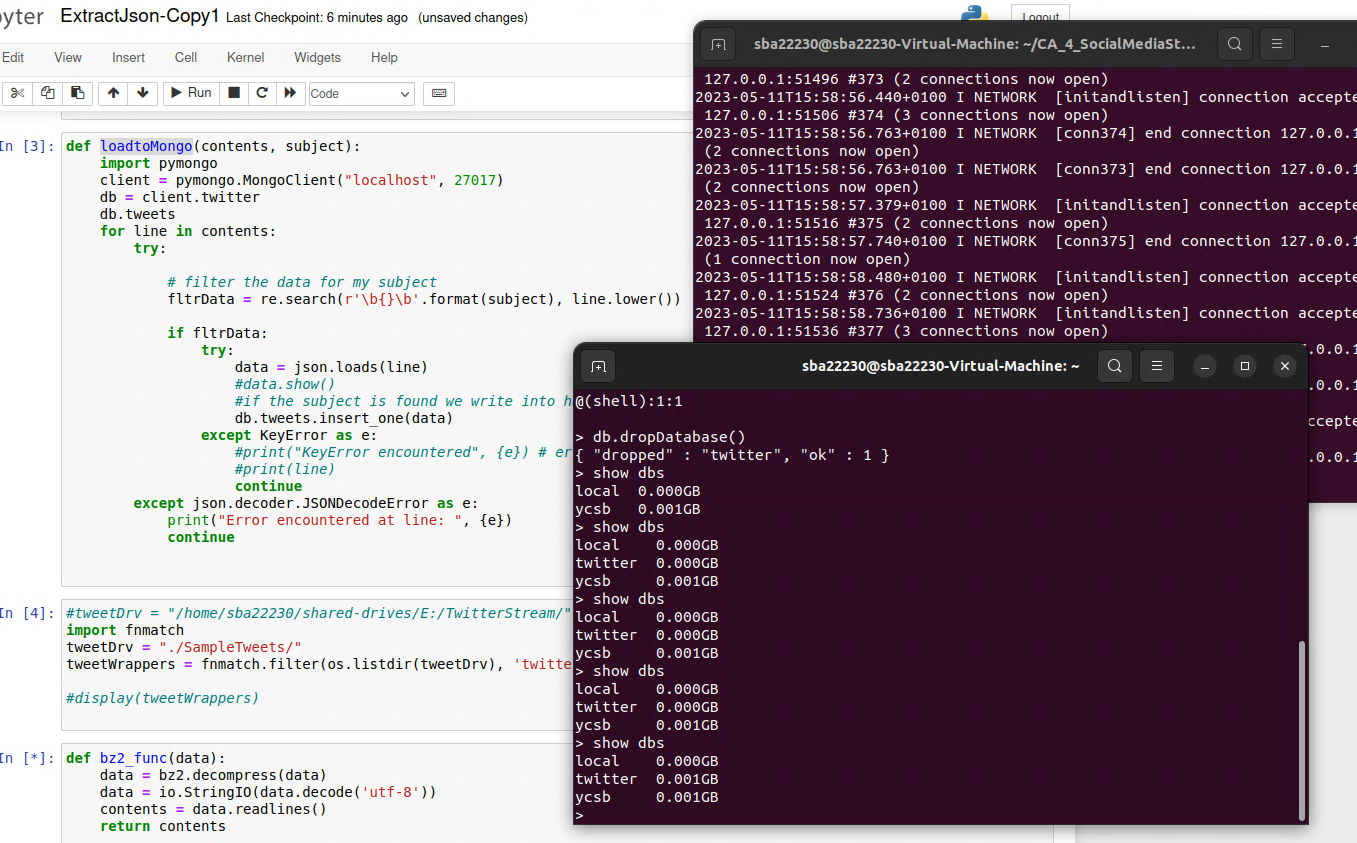
### Data gathering

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 decsion 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 strucuture was visualised 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 inferschema is set to read only a certain number of rows and infer the data strtucture 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 inferschema to false, this made spark 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

|lang|TweetCount|

| en| 48892|

| fr| 8450|

| es| 6754|

| ja| 4105|

| und| 1618|

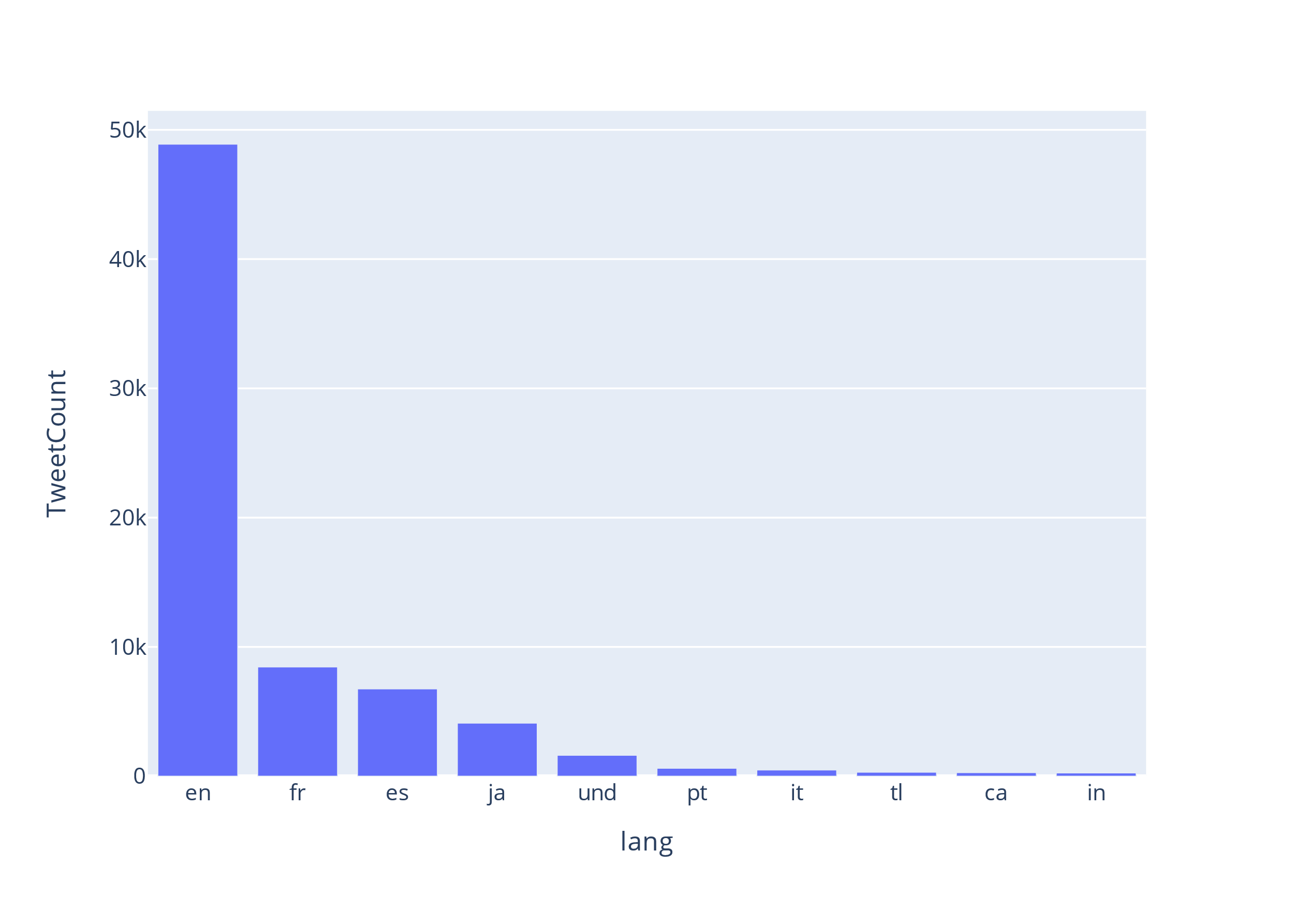
| pt| 614|

| 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|TweetCount|

| null| 24804|

| London, England| 543|

| United Kingdom| 529|

| France| 503|

| Kampala, Uganda| 500|

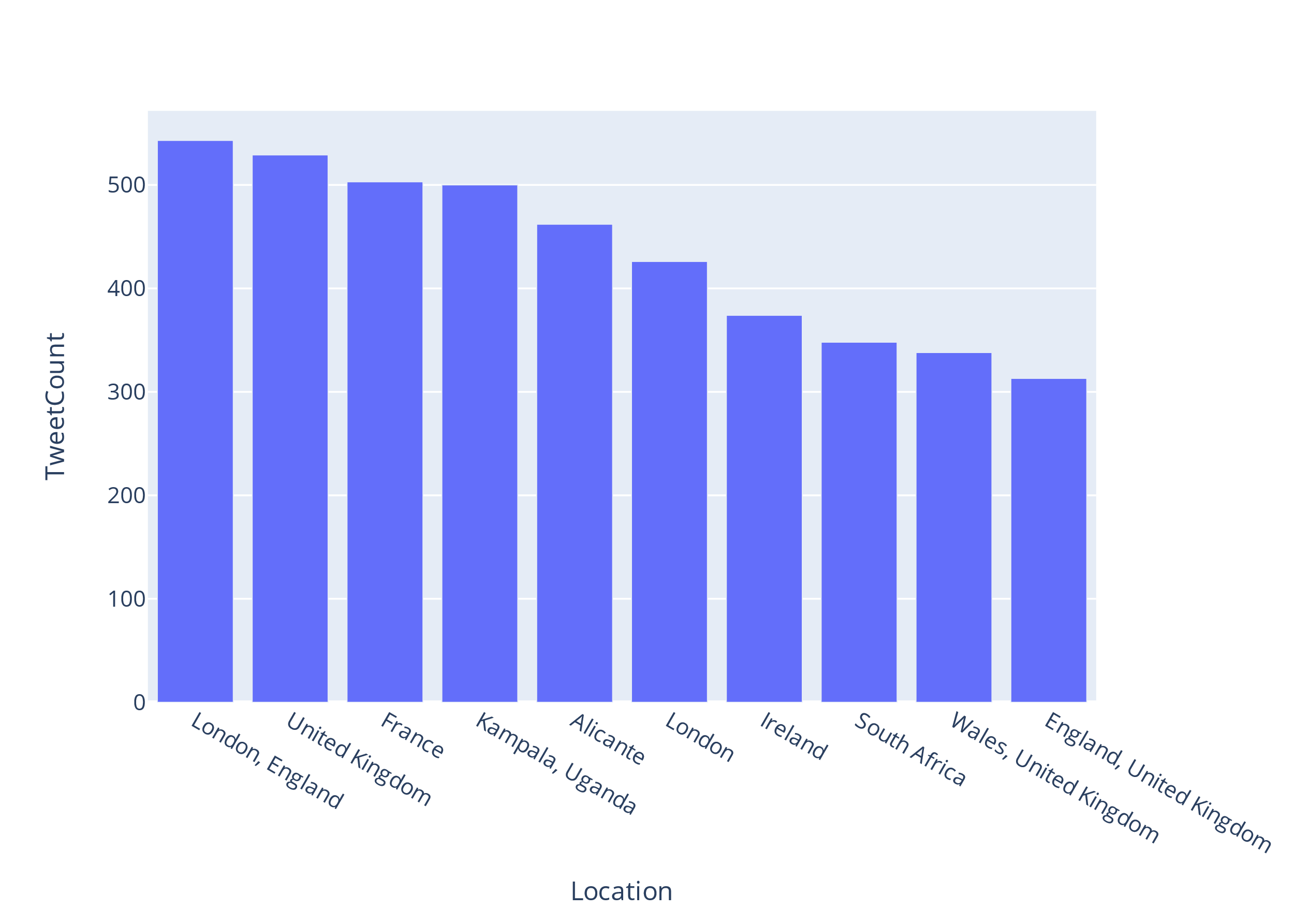
| Alicante| 462|

| 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

Now how many tweets in the English language dataset: 48892

| Location|TweetCount|

| 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|

| Date|TweetCount

|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|

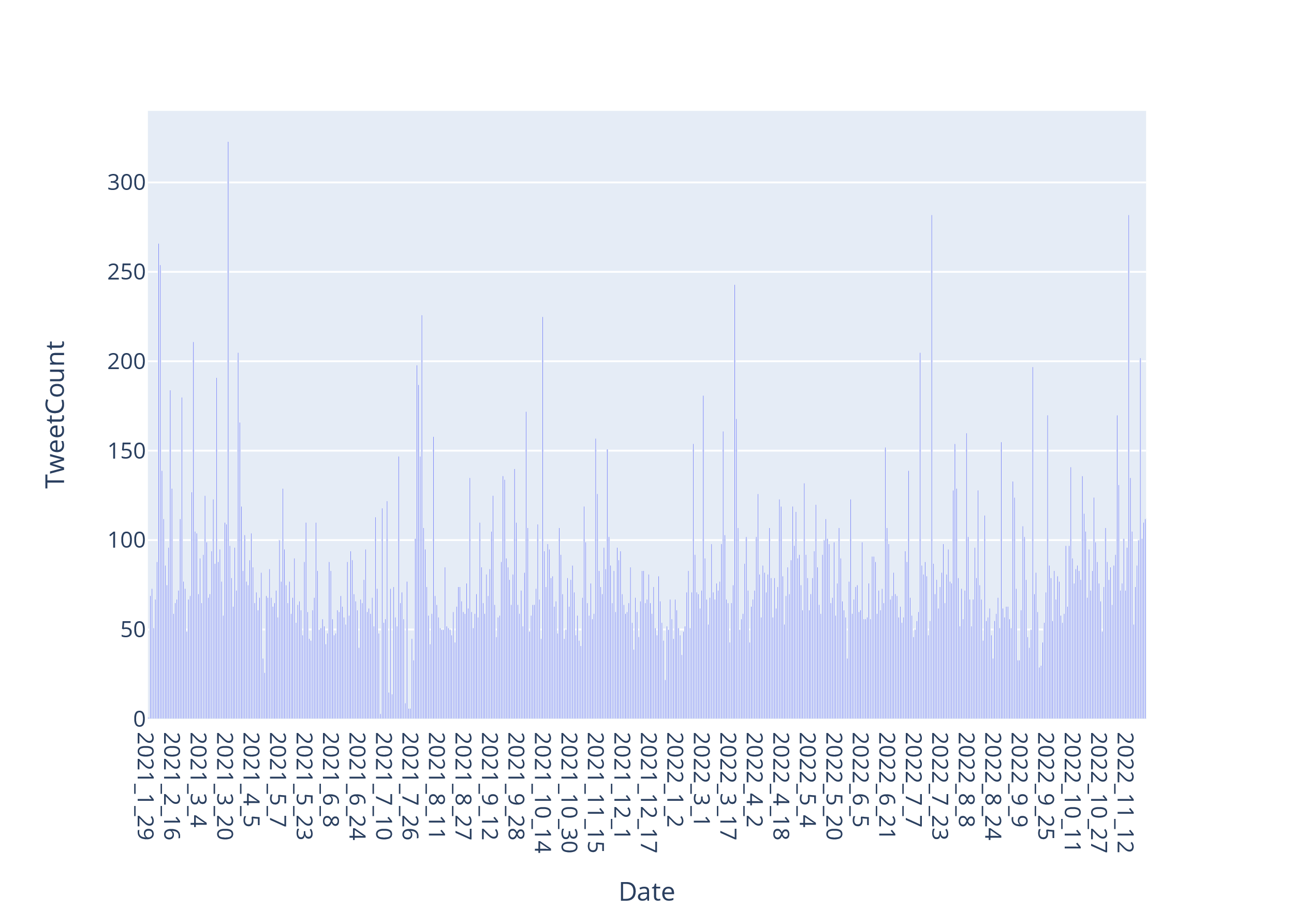
|2021\_2\_16| 65|

|2021\_2\_17| 67|

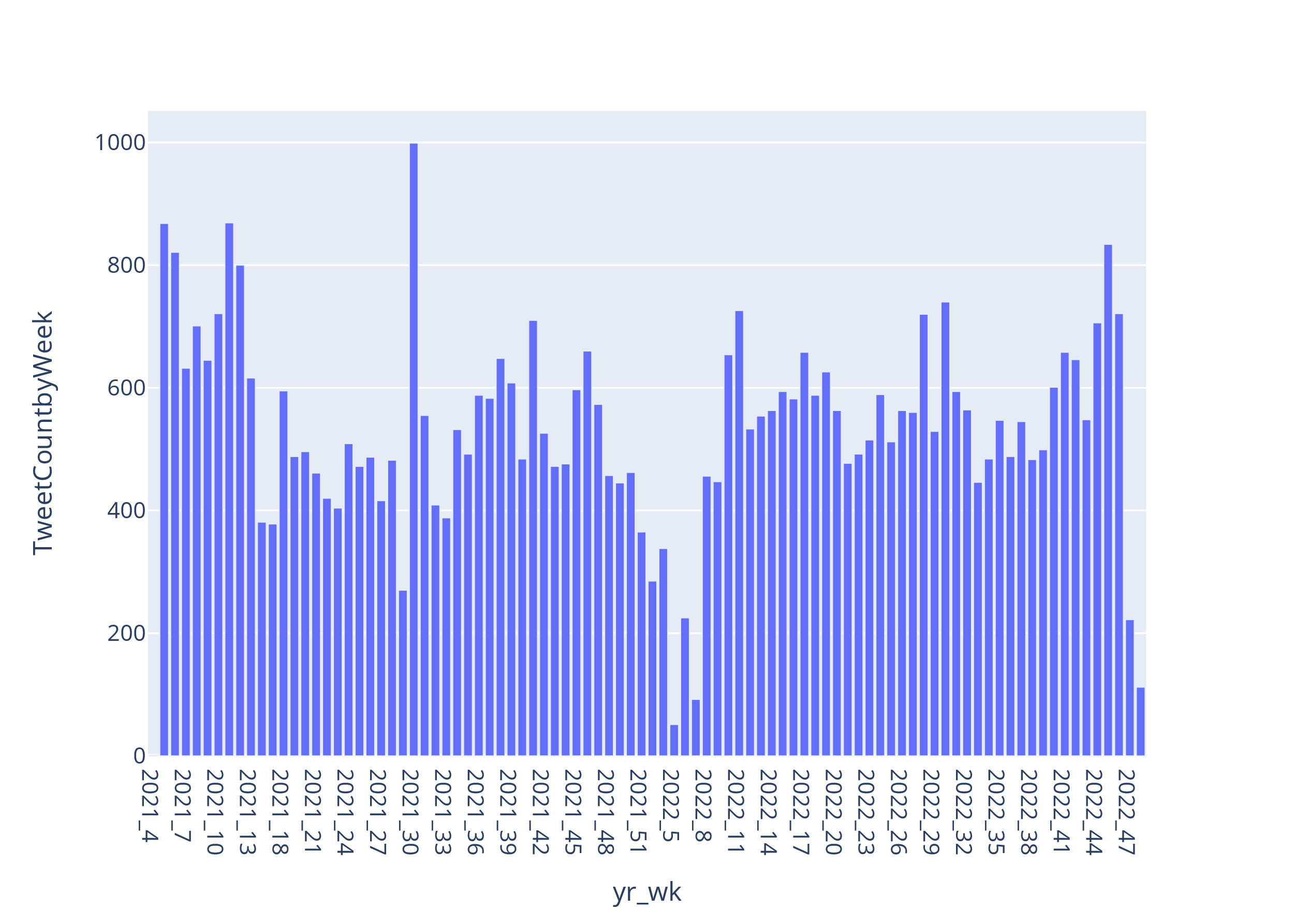
|2021\_2\_18| 72|

|2021\_2\_19| 112|

only showing top 20 rows

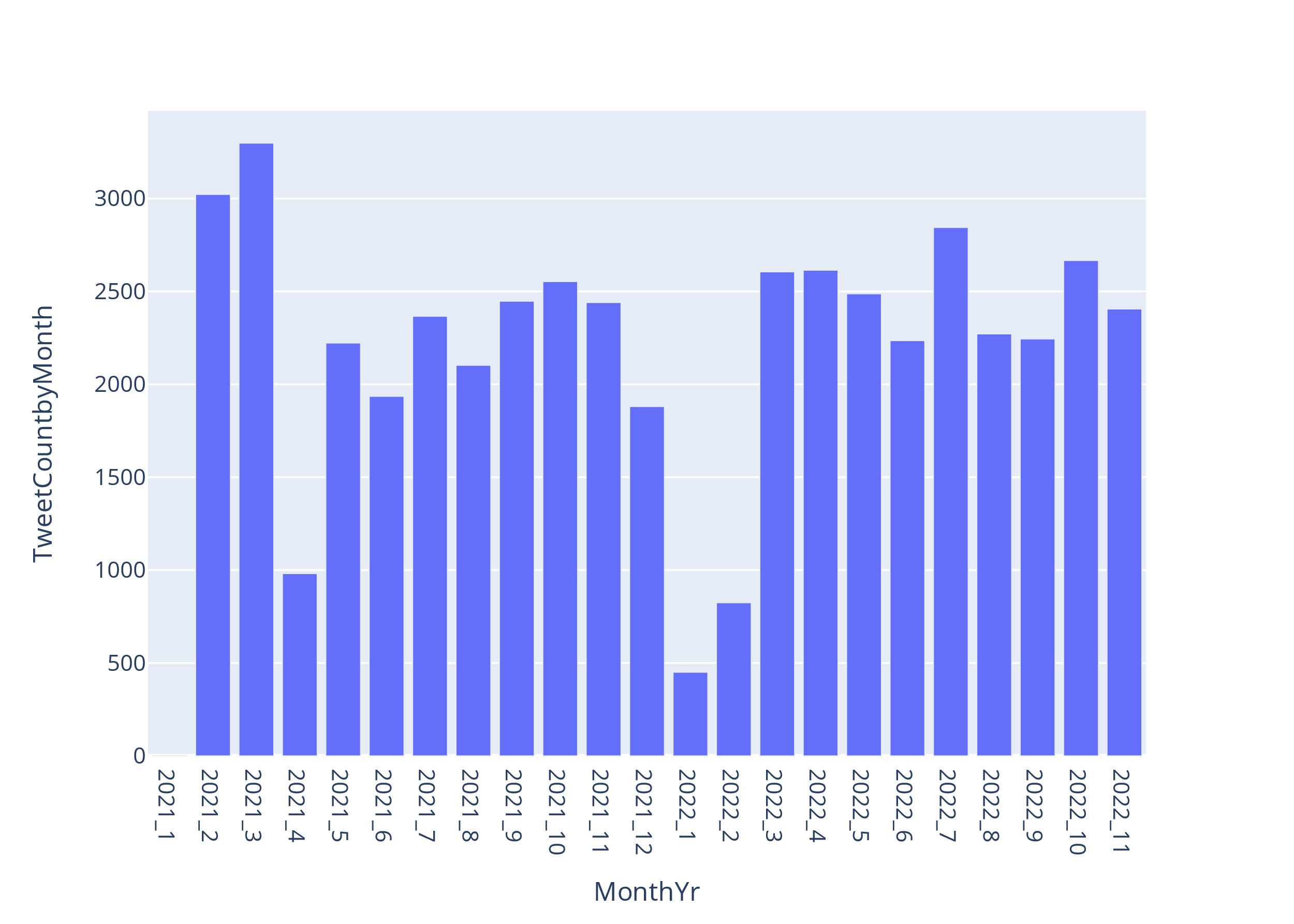


It looks like we have data for most of the days in the data set, but this is misleading because some of the files on the archive.org data set are corrupted and some months are missing entire weeks. As we move further into the analysis we will see these missing data more clearly.



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

Count the number of characters including spaces.

Check for special characters i.e., Hashtags.

Check for the upper case.

Nothing showed for the check on uppercase letters, there were no tweets composed completely of uppercase letters.

Check for numbers.

This check does not really advance our understanding of the data, a lot of twitter names have numbers in them.

Leave only text in the strings, all non alpha numeric characters are removed with the application of the regular expression.

The text variable contains only lower-case text that has been trimmed, that is spaces from the start and the end of the line have been removed, and numbers have been removed.

### Tokenize and stem the tweets.

# stackoverflow ref: https://stackoverflow.com/questions/53579444  
  
# Tokenize text

# Remove stopwords

Decided to stem the words as per this page <https://stackoverflow.com/questions/53579444>

Output the cleaned text into a parquet file for ease of transfer between machines.

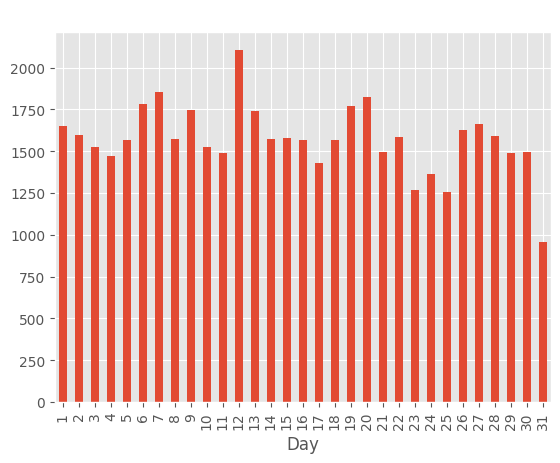
## Building the model

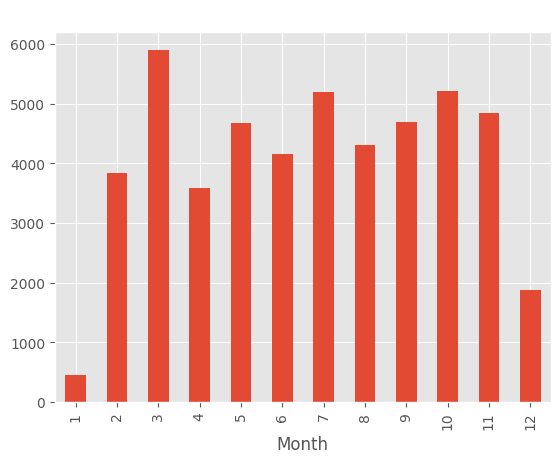
## Modeling Phase

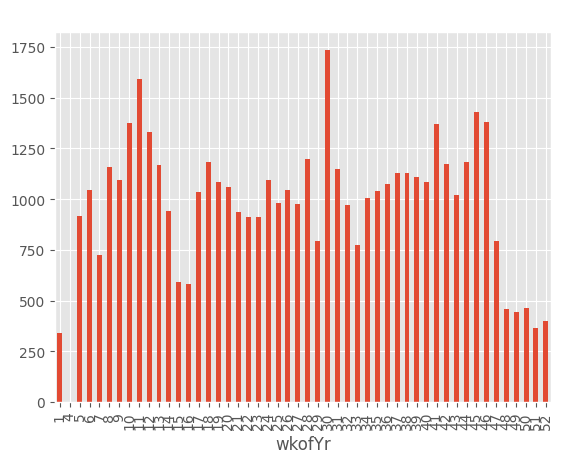
### Start of Time Series Analysis

Time Series predictions

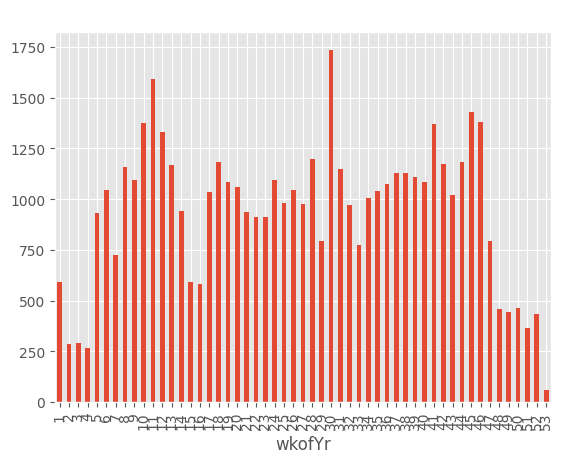
First up join the text that has been analysed for sentiment with the extra tweet information.



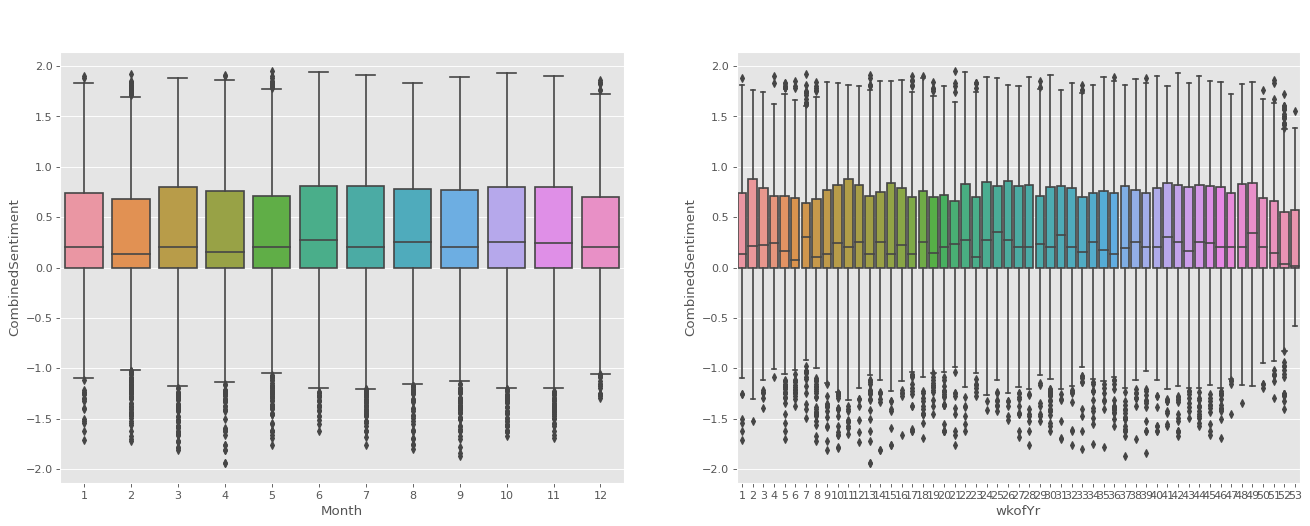




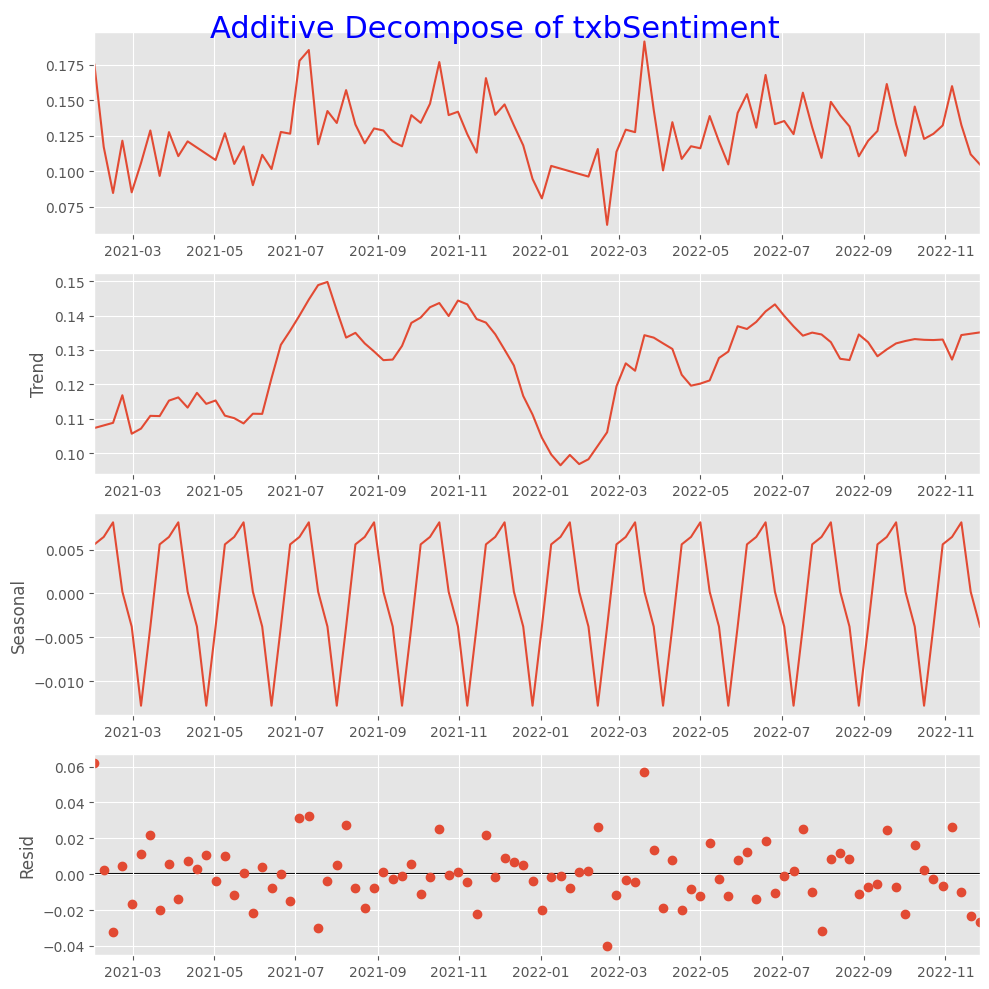
As we can see January is missing data from a couple of weeks This is an issue with the data source and can't be recitified by going back to the data The plan is too sample 1000 rows from the data frame and add them in january Also I am going to drop the year value as other months are missing days



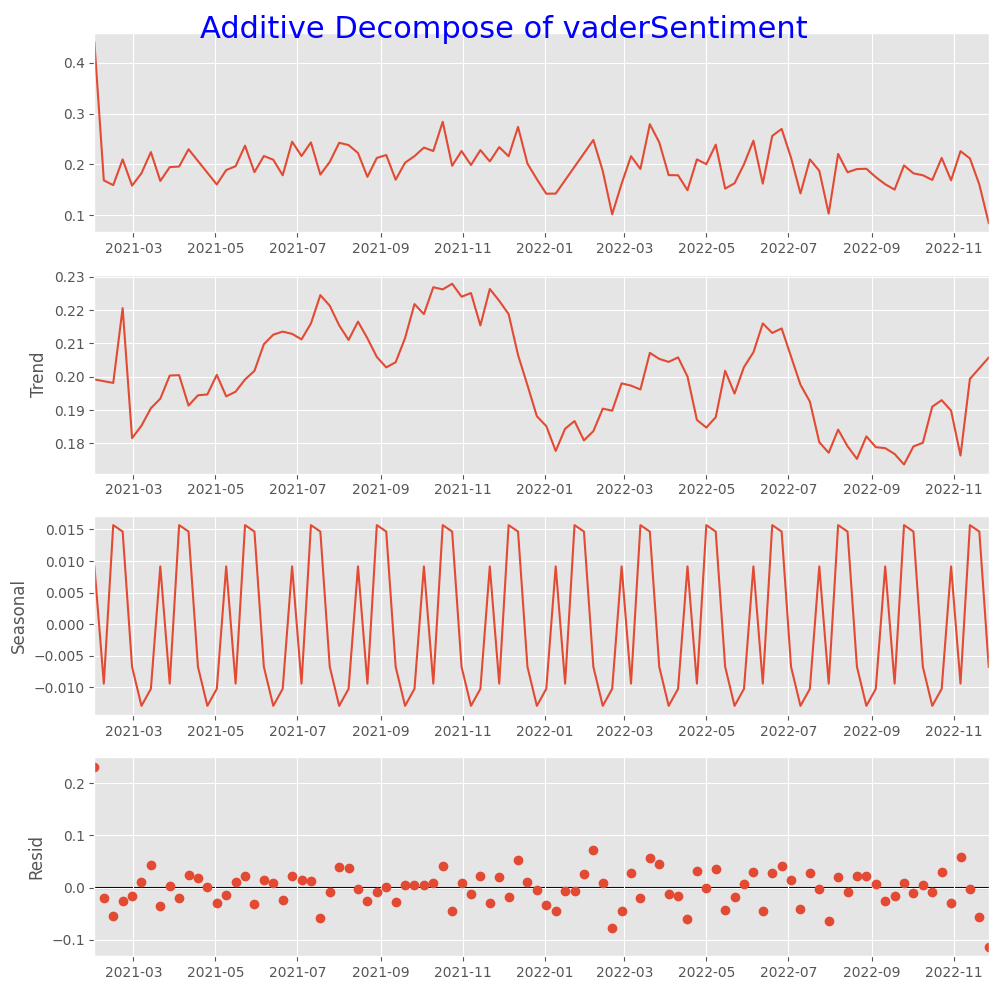
Get ready for the time series model.



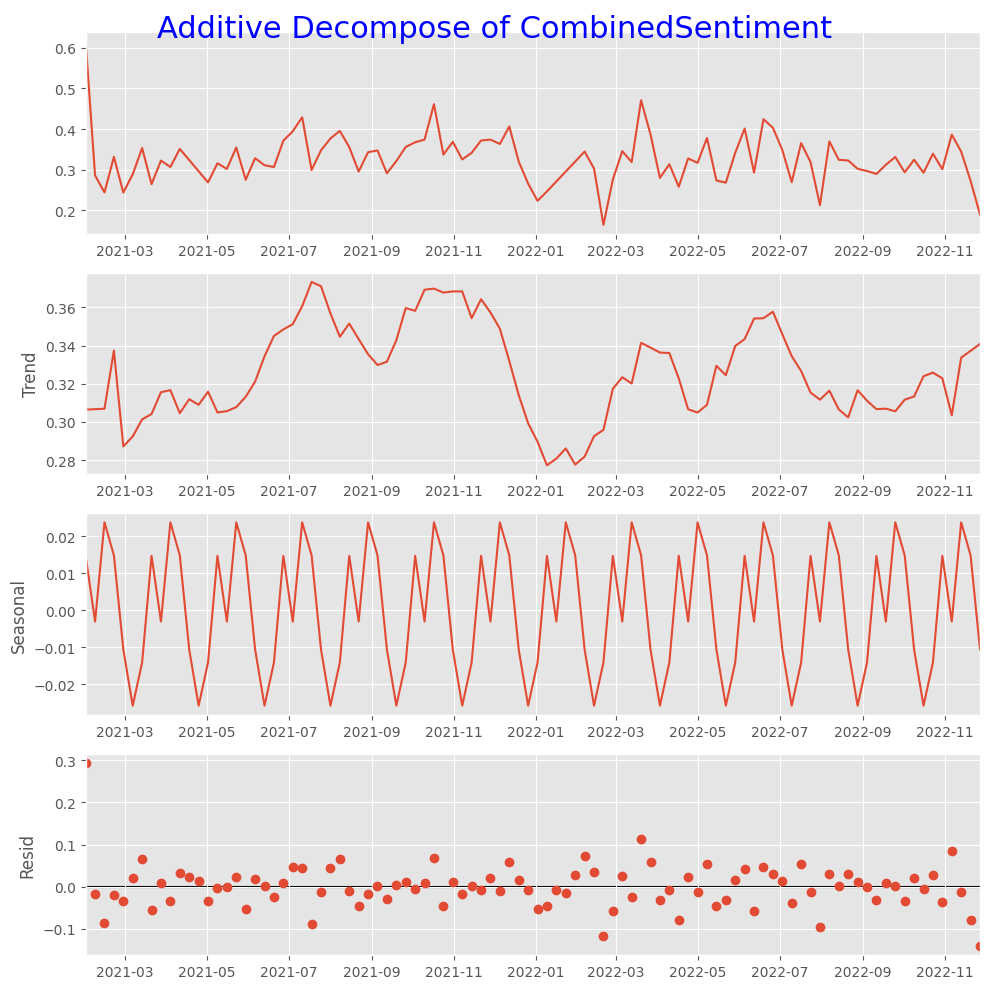
txbSentiment vaderSentiment CombinedSentiment  
date   
2021-01-31 00:00:00+00:00 False False False  
2021-02-07 00:00:00+00:00 False False False  
2021-02-14 00:00:00+00:00 False False False  
2021-02-21 00:00:00+00:00 False False False  
2021-02-28 00:00:00+00:00 False False False



seas trend resid actual\_values  
date   
2021-01-31 00:00:00+00:00 0.005592 0.107286 0.062122 0.175000  
2021-02-07 00:00:00+00:00 0.006439 0.108037 0.002598 0.117075  
2021-02-14 00:00:00+00:00 0.008101 0.108789 -0.032274 0.084615  
2021-02-21 00:00:00+00:00 0.000207 0.116812 0.004470 0.121490  
2021-02-28 00:00:00+00:00 -0.003770 0.105619 -0.016746 0.085103



seas trend resid actual\_values  
date   
2021-01-31 00:00:00+00:00 0.009143 0.199181 0.232076 0.440400  
2021-02-07 00:00:00+00:00 -0.009454 0.198657 -0.020507 0.168695  
2021-02-14 00:00:00+00:00 0.015691 0.198132 -0.054506 0.159317  
2021-02-21 00:00:00+00:00 0.014673 0.220568 -0.025261 0.209980  
2021-02-28 00:00:00+00:00 -0.006787 0.181587 -0.016448 0.158353



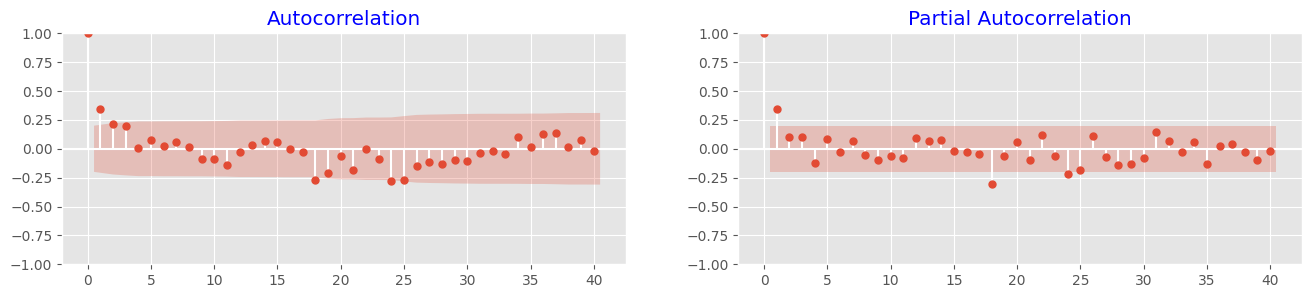
seas trend resid actual\_values  
date   
2021-01-31 00:00:00+00:00 0.014735 0.306467 0.294198 0.615400  
2021-02-07 00:00:00+00:00 -0.003015 0.306694 -0.017909 0.285770  
2021-02-14 00:00:00+00:00 0.023792 0.306921 -0.086781 0.243932  
2021-02-21 00:00:00+00:00 0.014880 0.337380 -0.020790 0.331470  
2021-02-28 00:00:00+00:00 -0.010557 0.287206 -0.033194 0.243456

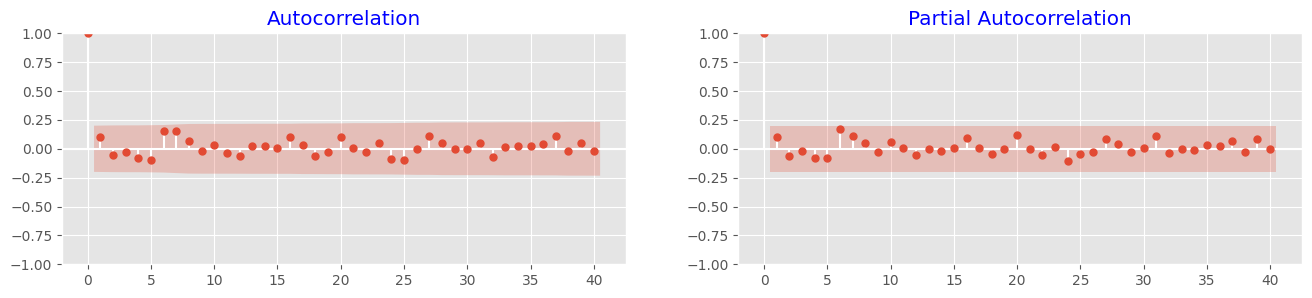
Test for Stationary Seasonality using Augmented Dickey Fuller test (ADH Test)

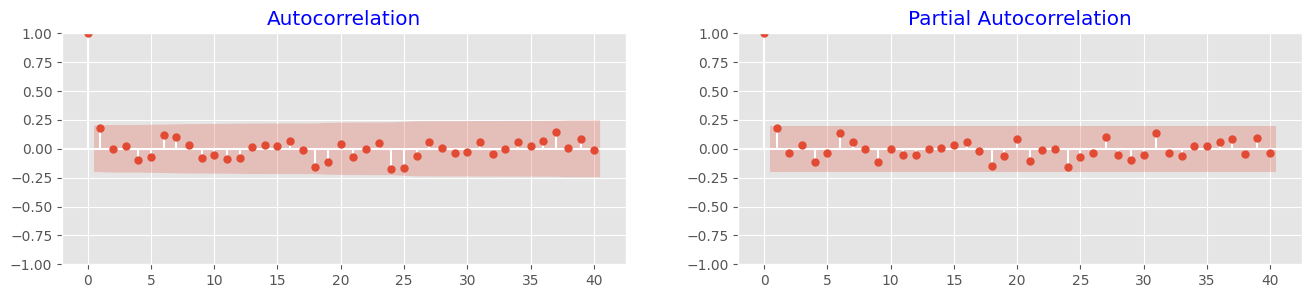
TextBlob Sentiment ADF Test  
ADF Statistic: -6.863973307717687  
p-value: 1.5763050593574846e-09  
Probably Stationary  
Critial Values:  
 1%, -3.5011373281819504  
Critial Values:  
 5%, -2.8924800524857854  
Critial Values:  
 10%, -2.5832749307479226  
  
 Vader Sentiment ADF Test  
ADF Statistic: -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

No trend is visible in the previous charts in this data.

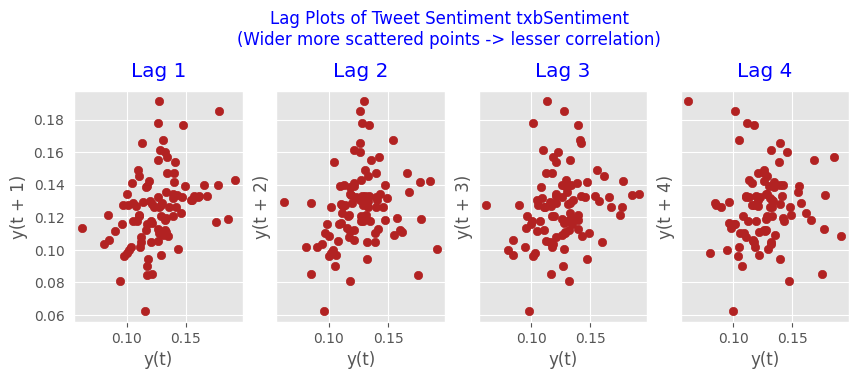
let’s check for seasonality.

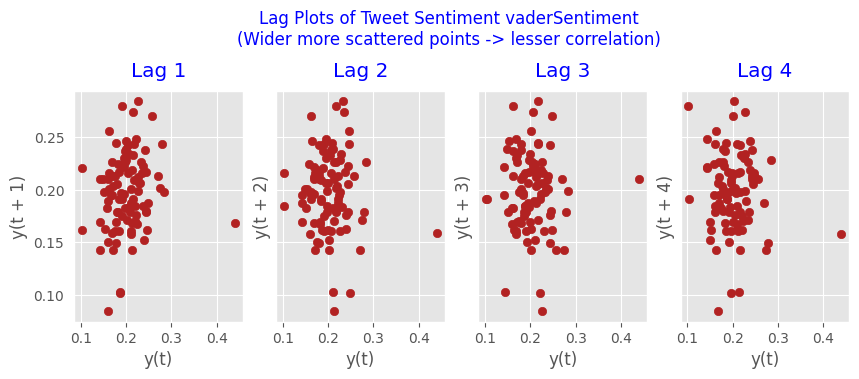


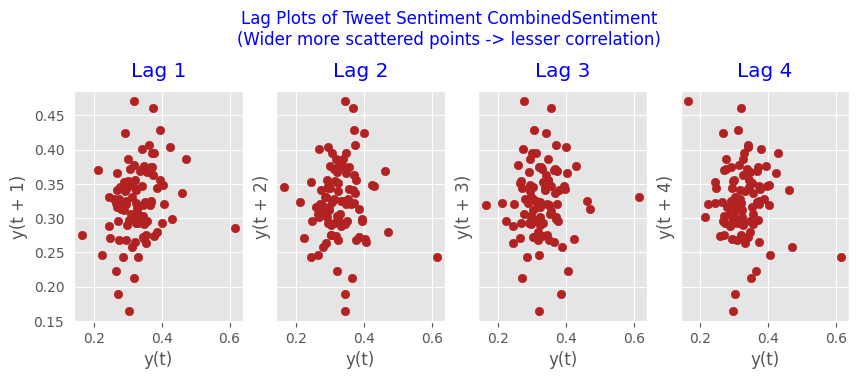




Scatter plots showing the lag correlation.







Entropy

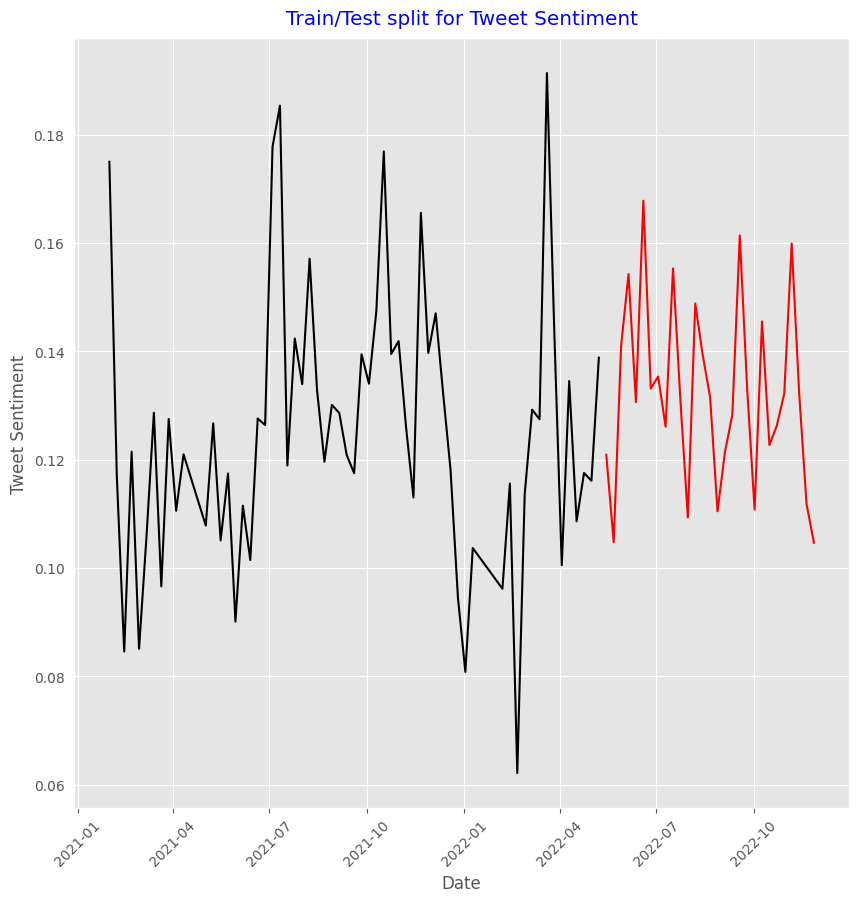
grangercausalitytests(df[['txbSentiment', 'month']], maxlag=4)

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 at last!



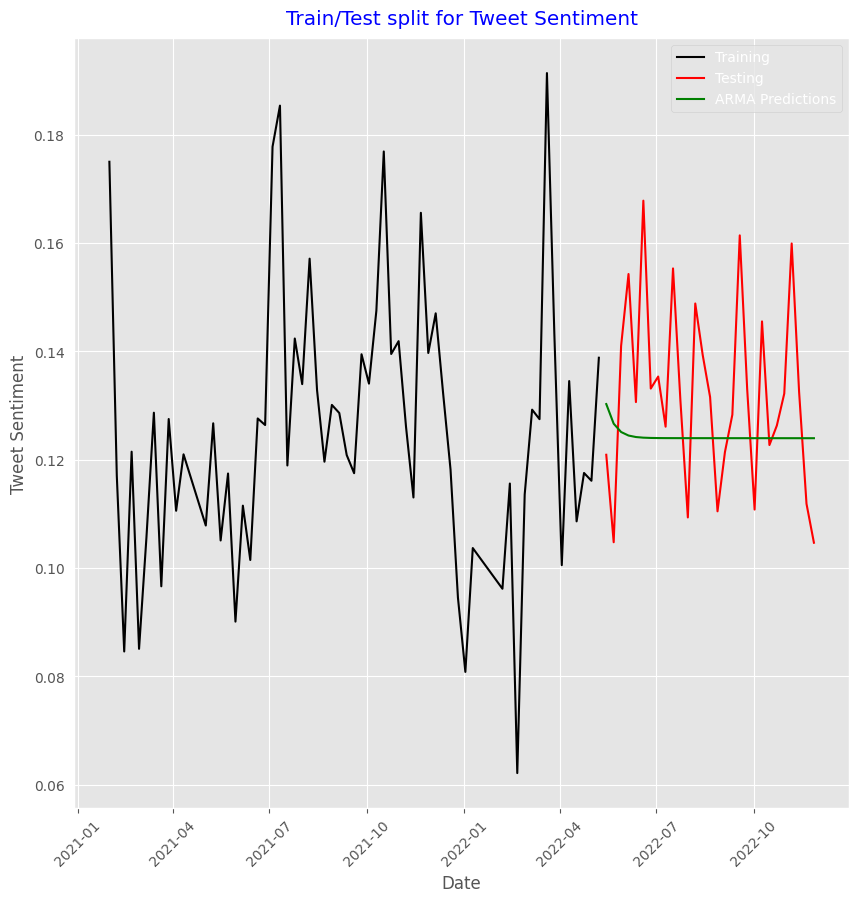
## Evaluation Phase

### Grid Search for the best model

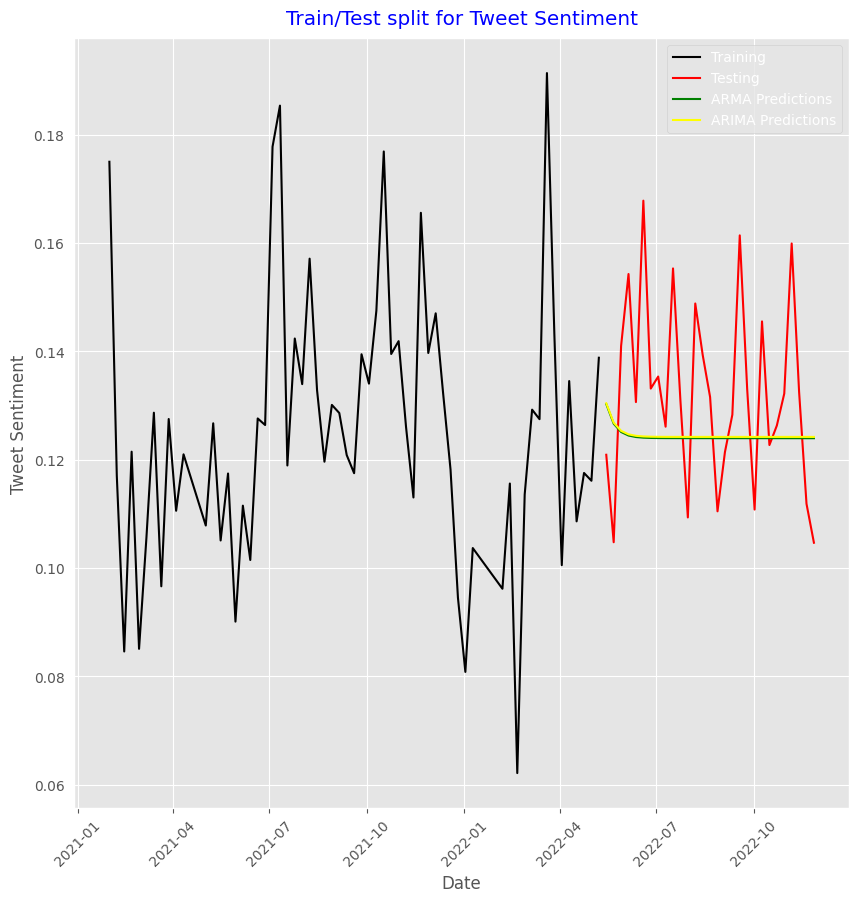
Taken from here <https://towardsdev.com/auto-arima-hyperparameter-search-ab991a21c2bd>

# Grid Search

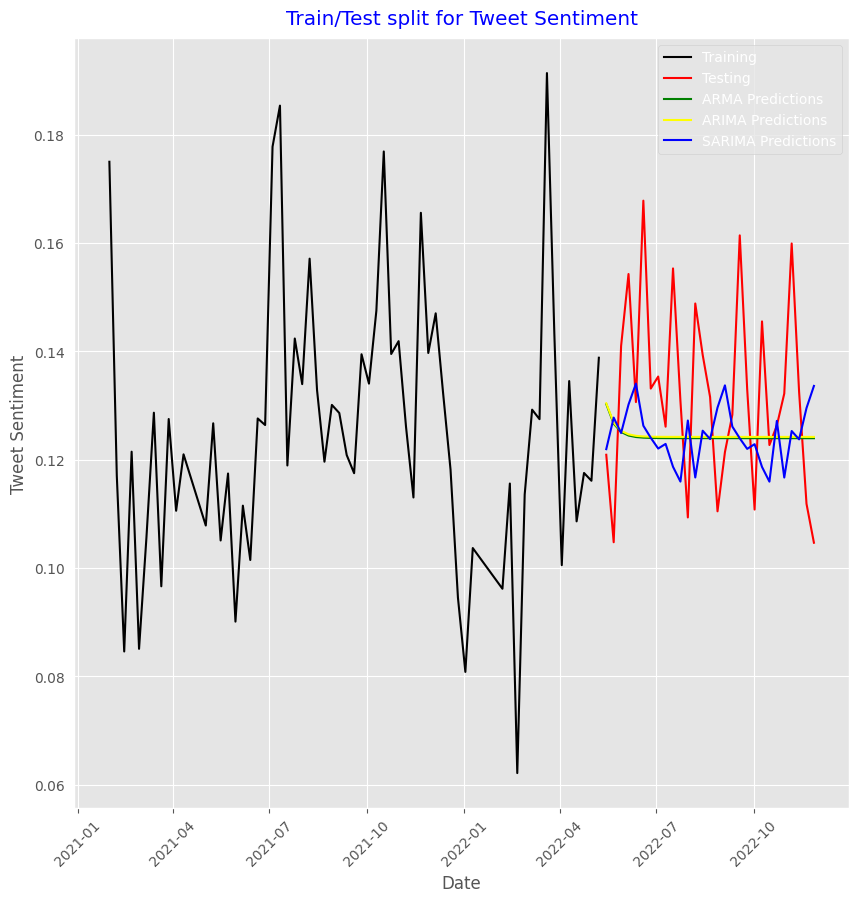
Arma



RMSE for ARMA model: 0.031672380658061355



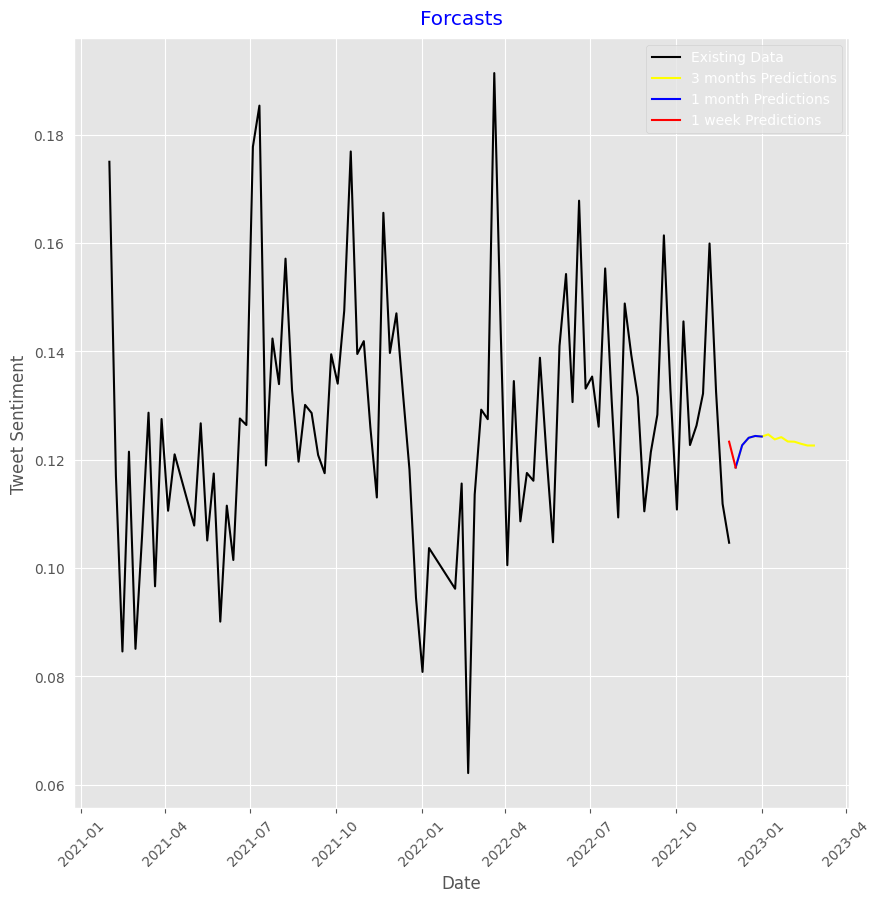
ARIMA RMSE: 0.03185068818081811



SARIMA RMSE: 0.031421142238327336

## Deployment Phase

### SARIMAX model wins the right predict txbSentiment.



# 

# Dashboard