**WEB SCIENCE COURSEWORK REPORT: 2418094**

Table of Contents

[INTRODUCTION: TWITTER 2](#_Toc530372504)

[SOFTWARE DESCRIPTION 2](#_Toc530372505)

[TIME DESCRIPTION 3](#_Toc530372506)

[ACTUAL CODE ACCESS 3](#_Toc530372507)

[DATA CRAWL: TWITTER 4](#_Toc530372508)

[STREAMING API 4](#_Toc530372509)

[ENHANCING CRAWLING USING STREAMING AND REST API 5](#_Toc530372510)

[GEO TAGGED DATA COLLECTION FOR GLASGOW FOR SAME TIME PERIOD 6](#_Toc530372511)

[DATA ACCESS STRATEGY AND RESTRICTIONS FROM TWITTER AND TWIT MODULE 7](#_Toc530372512)

[BASIC DATA ANALYTICS: TWITTER 8](#_Toc530372513)

[TOTAL DATA COLLECTED 8](#_Toc530372514)

[TWEETS FROM GLASGOW 9](#_Toc530372515)

[TOTAL REDUNDANT DATA 9](#_Toc530372516)

[RE-TWEETS AND QUOTES 10](#_Toc530372517)

[ENHANCING THE GEO TAGGED DATA: TWITTER 11](#_Toc530372518)

[GROUPING OF TWEETS: CLUSTERING 11](#_Toc530372519)

[GEO-LOCATION ASSIGNMENT TO CLUSTERS 15](#_Toc530372520)

[EVLUATION OF METHOD 17](#_Toc530372521)

[DATA CRAWLING: TUMBLR 19](#_Toc530372522)

[INTRODUCTION: DATA COLLECTION MECHANISM 19](#_Toc530372523)

[ACTUAL CODE ACCESS 19](#_Toc530372524)

[DATA RESTRICTIONS 19](#_Toc530372525)

[DATA COLLECTION APPROACH 19](#_Toc530372526)

[CODE SAMPLE 20](#_Toc530372527)

[DATA ANALYSIS 21](#_Toc530372528)

[BIBLIOGRAPHY 23](#_Toc530372529)

# INTRODUCTION: TWITTER

## SOFTWARE DESCRIPTION

The software developed for Twitter web crawler uses JavaScript as the programming language with Node.js (v8.12.0) (NPM version: 6.4.1) as the run-time environment for execution of the program. The data collected is stored in the MongoDB which is preferred due to its flexibility and ease of use to store unstructured data in a JSON format. The various node package manager (NPM) modules have been used, all of MIT license, which are helpful is collecting Twitter data and performing the analysis on the collecting tweets. Various methodologies and filters have been used for collection of the Twitter data.

This software uses two APIs for collecting data from Twitter, namely REST API and STREAMING API.

The REST API is used to collect keyword filtered data with a maximum limit of 100 tweets per call. This restriction is due to Twitter restriction of the number of tweets per call, present in the NPM module used. The REST API calls have been made every 10 seconds for a period of 5 minutes (for collecting sample data) or 1 hour (for collecting more data). Some errors were encountered in making these many calls, which are Limit Message errors (from the error logs). After these warnings, the connection is automatically restored and tweets keep coming for the specified time duration.

The second type of API used is STREAMING API. This API opens an endless stream of tweets where real-time tweets keep coming for the specified duration of time (5 minutes or 1 hour). Few filters have been used in these streaming API, for instance, keyword filtering (same set of keywords used for REST API), location filtering (in bounding box coordinates) and one stream without any filtering of data.

The various NPM modules/libraries have been used for the software which are Body Parser, Cors, Express, Minhash, Mongoose, Plotly, Twit, Winston and Winston Daily Rotate File, Stopword (NPM, 2014).

After the data collection is done, the software performs analytics on the collected data which will be discussed further in the report. The analysis includes counting of total tweets, redundant tweets, geo-tagged tweets, etc. Then, the software collects all texts from the tweets and forms clusters according to the Minhas LSH algorithm and gives out information about geo-tagged and user profile based statistics from the created clusters. Various graphs have been generated as a part of the analysis whose URL can be found out from the final output logs of the software.

The output of the running of the software can be found in the directory Twitter-Crawler-Logs and the final/major outputs of the analytics can be found in the directory Twitter-Crawler-Final-Output-Log, both located in the crawl-server directory.

## TIME DESCRIPTION

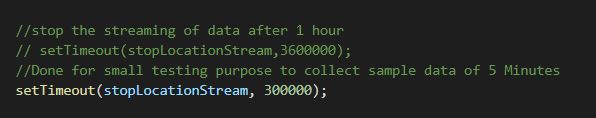
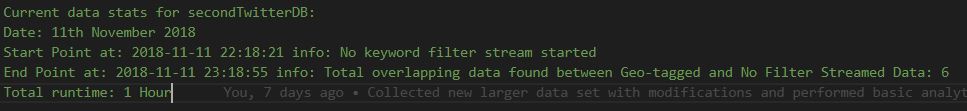
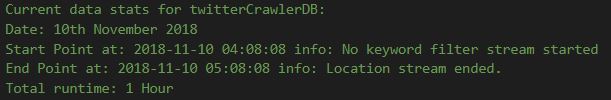
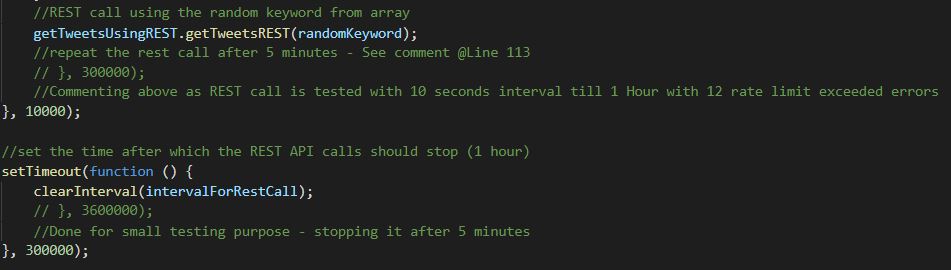
The data collected from various methods has been done so for different time durations. The REST API and STREAMING API have been used three times overall, running for 1 hour each for 2 times, once at approximately 04:08:08 on 10th November 2018 till 05:08:08 and the other time on 22:18:21 till 23:18:55 on 11th November 2018.

It was also run for multiple times for 10 minutes duration, 6 times to compare the amount of data collected for each slot over an hour.

The REST API calls have been made at 5 minutes interval for 1 hour as well as at 10 seconds interval for 1 hour, 10 minutes and 5 minutes duration. The STREAMING API calls have been used for 1 hour, 10 minutes and 5 minutes durations as well.

Finally, the software also collected data for a 5-minute window which is submitted as a sample data along with the code (sampleData.tar – in the directory crawl-server/model).

Code Sample For Rest Call Timer, first and second run of the software, and timer for the streaming data:



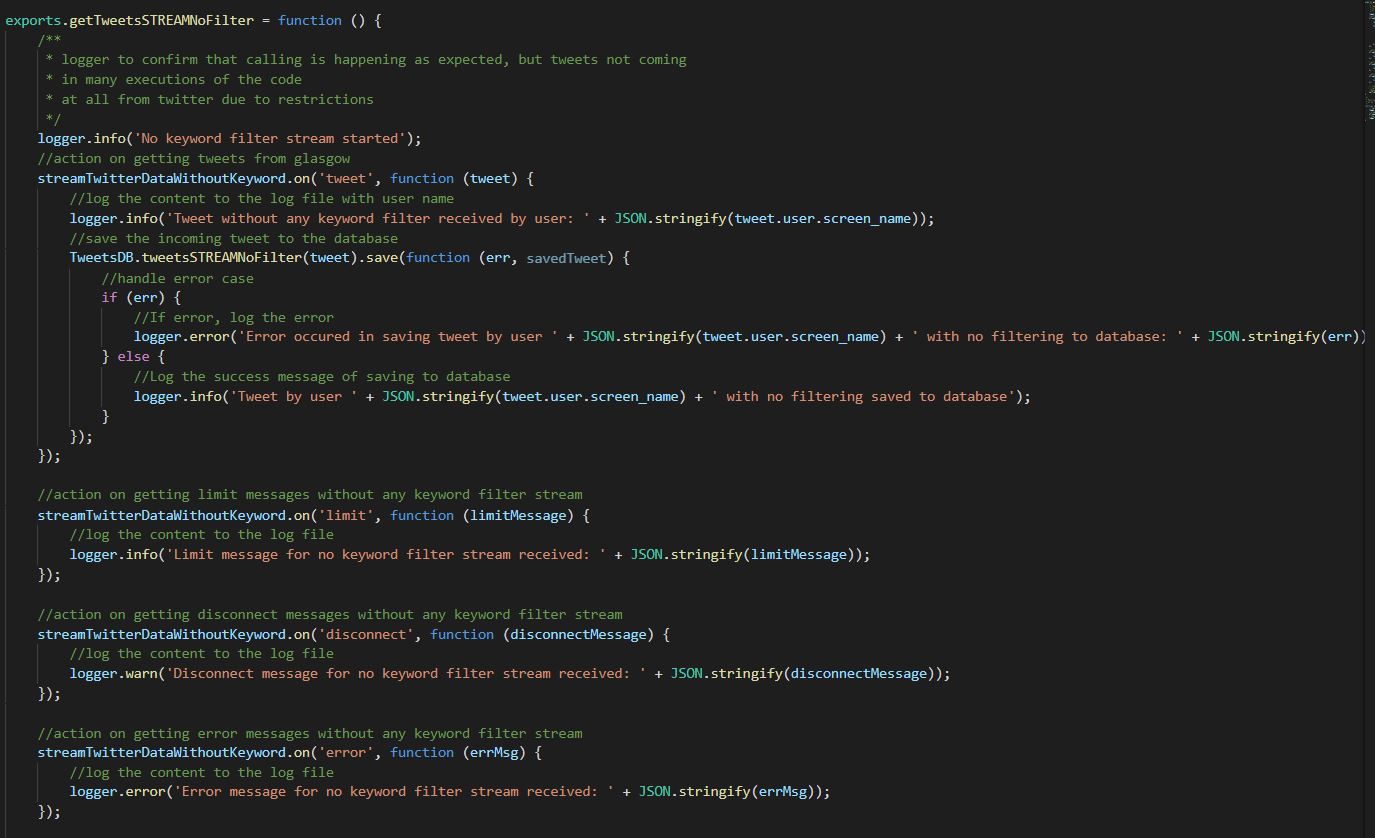
## ACTUAL CODE ACCESS

The actual code base for this software can be found out at <https://github.com/Kinshuk1993/twitterWebScience>.

# DATA CRAWL: TWITTER

## STREAMING API

The streaming API used in this software have been of the NPM library TWIT. I have used streaming endpoints provided by Twitter, namely statuses/sample.



The sample end-point is used for streaming of data without any filter of keyword or location. The filter end-point is used for streaming of data with keyword filtering and the location filtering data. The keyword filtering uses an array of keywords which have been termed as trending topics on Google and Twitter by the google search (Hudgens, 2016) (Mondovo, 2018).

This streaming API without any filter starts as soon as the program starts and runs for as long as it is needed. This is because one can modify the time period it should run for in the code by commenting out. The time period is mentioned in milliseconds.

The data collected using this is stored in the database in a collection named as tweetsstreamnofilters. The total data collected using no filter stream for one hour in 2 different runs were 118151 and 136885 on 10th and 11th November 2018 respectively.

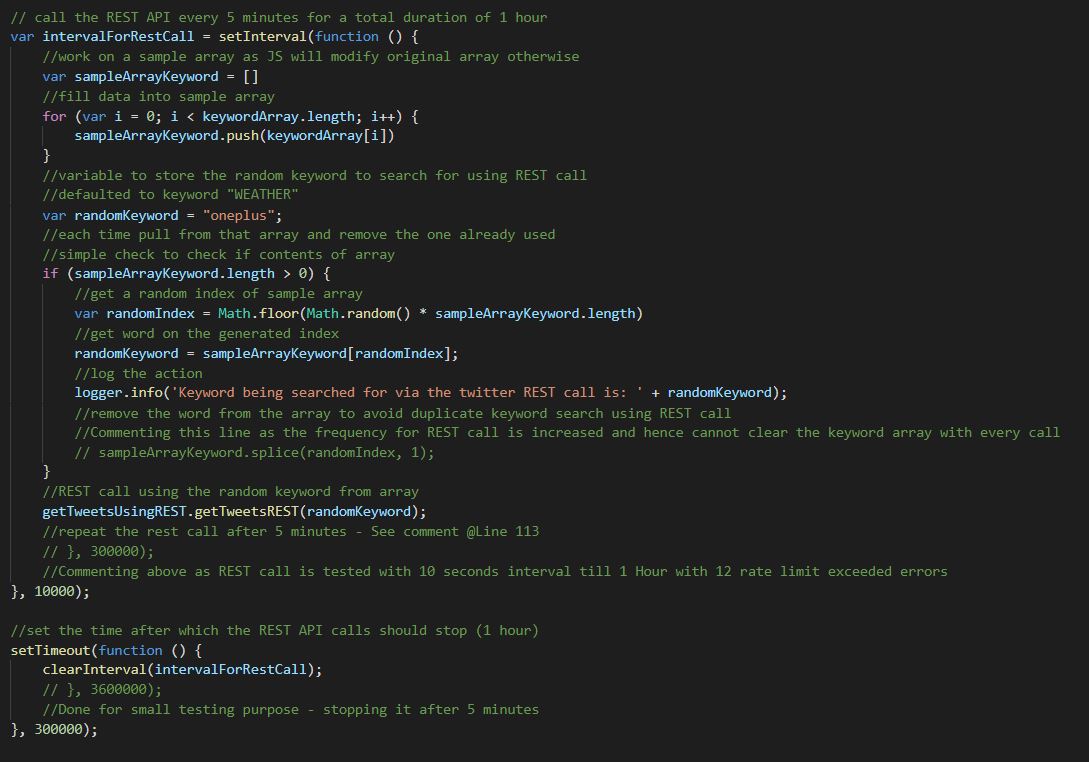
## ENHANCING CRAWLING USING STREAMING AND REST API

The data collected using no filter stream has been further enhanced by using rest and enhanced streaming API. The streaming endpoint used here is statuses/filter and search/tweets are used for REST API.

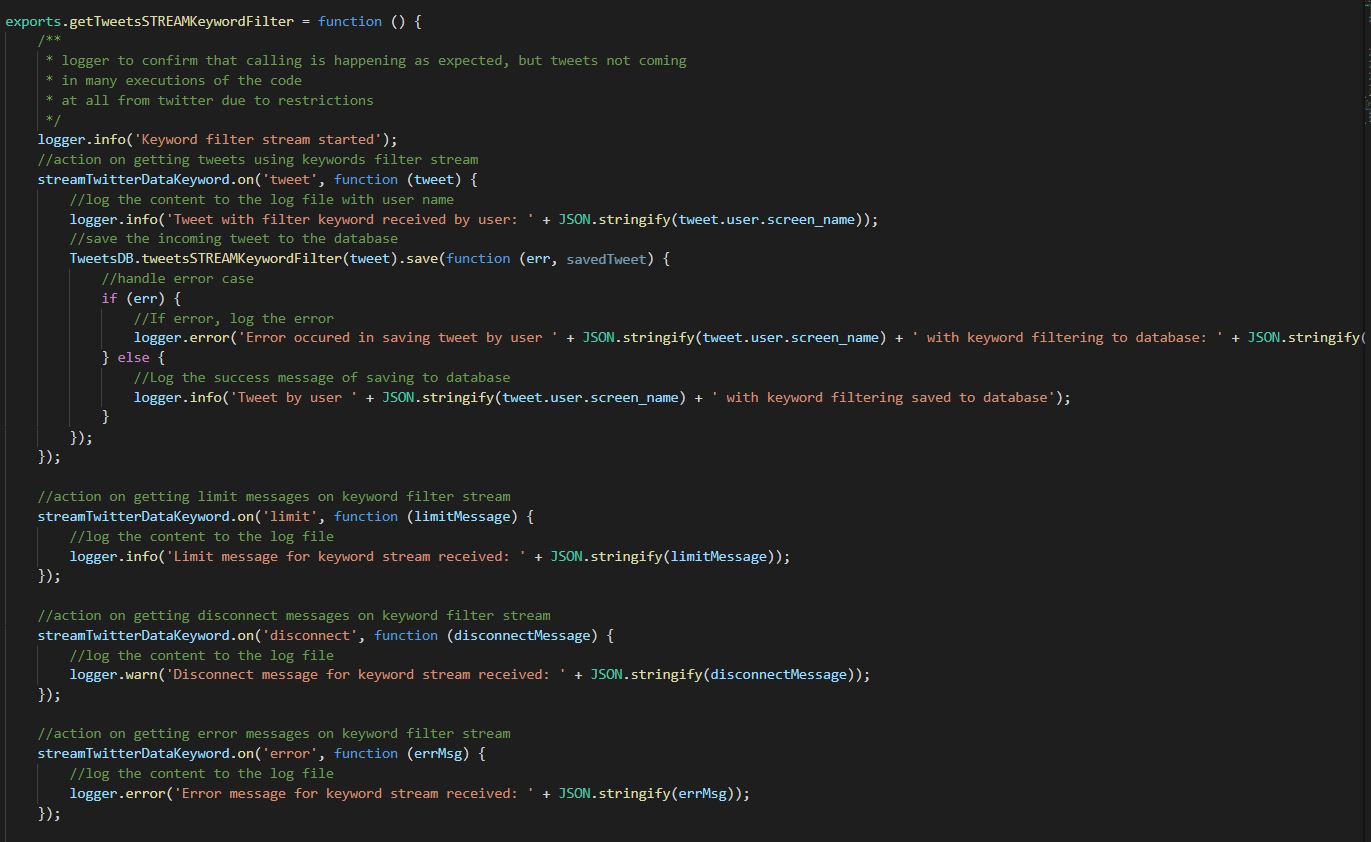
An array of most common and trending keywords have been used for making rest and streaming calls.

The REST API uses random keywords from the array for each call whereas the streaming API uses all of those keywords for filtering out tweets containing those keywords.

REST calls are made in two ways, one every 5 minutes for 1 hour, where only 1287 tweets were collected in an hour and the other at every 10 seconds for 1 hour where 33792 tweets were collected.

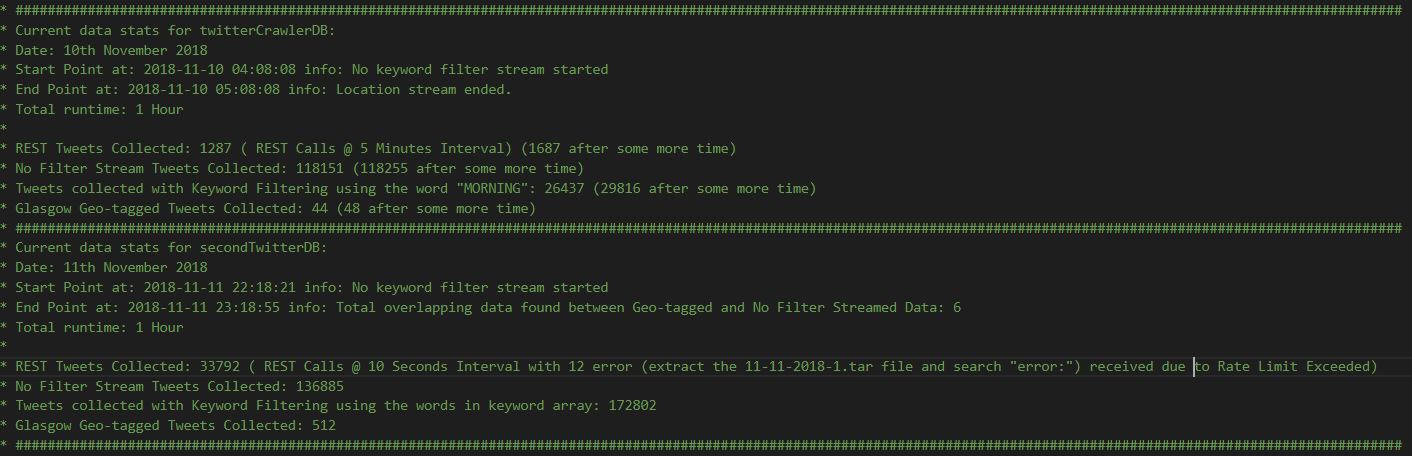


The data collected using streaming API are also done in two ways, one with only the keyword “Morning” which yielded 26437 tweets and for the second time, with the keyword array filter, which resulted in 172802 tweets. Both times, streaming was open for a 1-hour duration.



For further analysis purpose, the REST and STREAMING calls were made for 10 minutes duration as well as for 5 minutes duration (for sample data submission).

The enhancement made is clear from the amount of data collected which can be seen from the below screenshot (from app.js – captured during the development).

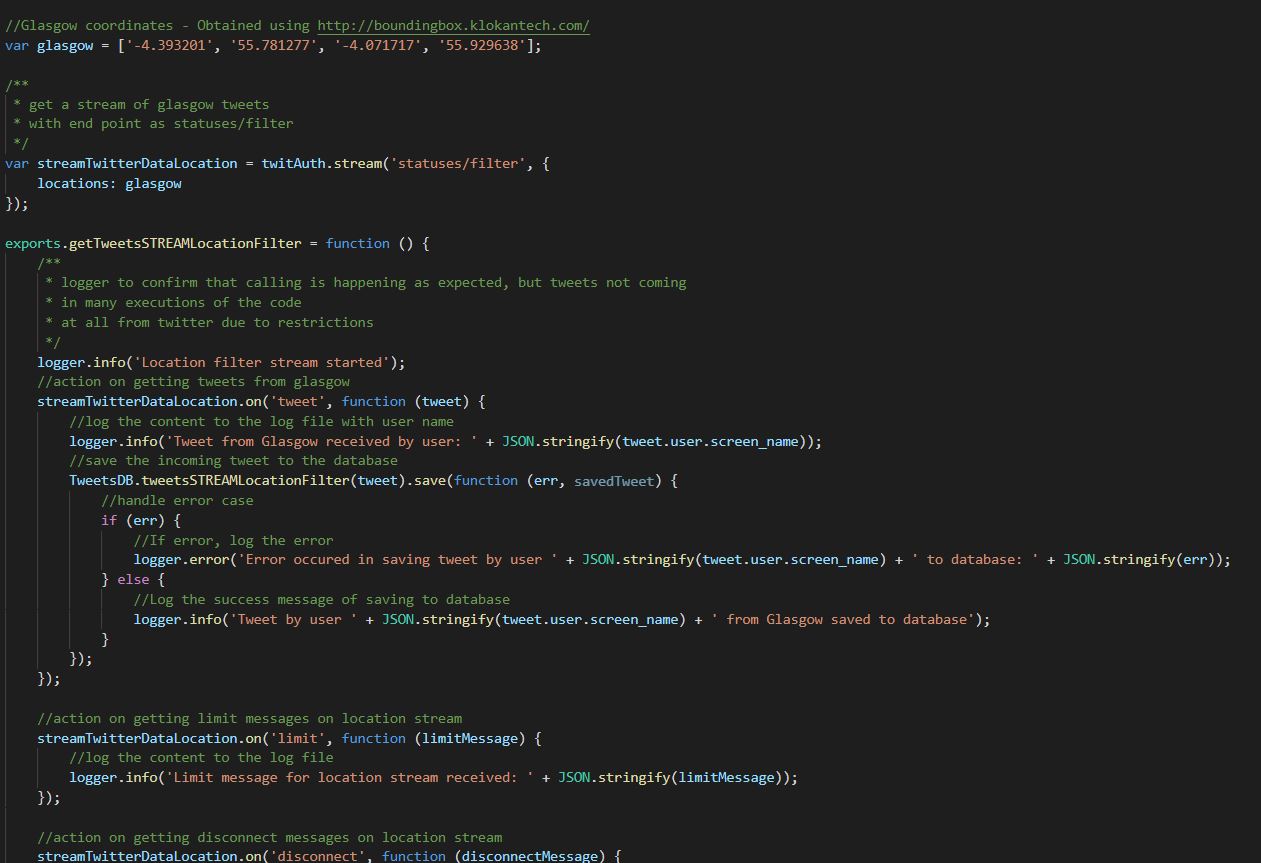


## GEOTAGGED DATA COLLECTION FOR GLASGOW FOR SAME TIME PERIOD

The streaming API is used for one more purpose which is to filter the incoming data according to the location coordinates for Glasgow. These coordinates have been used to create a bounding box for Glasgow, hence enabling incoming tweets to be from Glasgow location only.

The coordinates have been collected from the google search from <http://boundingbox.klokantech.com/> (Klokantech, 2018).

These coordinates have then been used in the code with endpoint as statuses/filter with the parameter as the coordinates obtained. This stream is also run similar to other streams (keyword and no filter) and also for same time duration.



The data collected during two periods of 1 hour on 2 days are 44 and 512 respectively. The sample data submitted along with the code has 64 tweets during the time it was collected.

## DATA ACCESS STRATEGY AND RESTRICTIONS FROM TWITTER AND TWIT MODULE

The data access from Twitter is done via the Twit Module which is installed via npm (ttezel, 2018).

Twitter has a restriction of a maximum number of REST calls that can be done in a 15-minute window, which is handled by the module as well by the software developed.

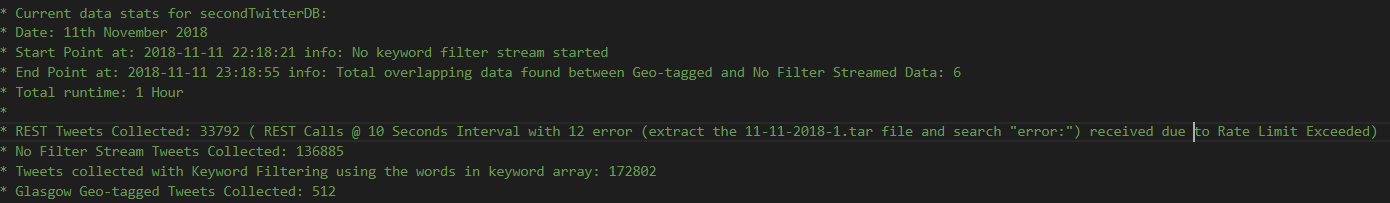
The way in which the Twit module handles the restriction is that, whenever the warning of limit exceeded message comes, it re-establishes the connection to Twitter and tries to fetch tweets again till it successfully gets a response from Twitter. In this software, the code is done in such a way that it makes the rest calls every 10 seconds (which initially was 5 minutes interval). This shift of 5 minutes to 10 seconds is done so that more data can be collected from the REST API calls.

The streaming API do not have such restrictions, as, during the testing of the API, it went on for 3 hours without stopping or throwing any error in the logs.

One restriction in using all 4 ways of collecting data (1 REST, 3 Streams) is that Twitter sometimes does not send data on either of the APIs and one has to terminate the program manually and start again – sometimes keep doing it for 4-5 times to get all tweets from all 4 API calls. But once that starts happening, it works flawlessly without stopping till manually stopped.

The software uses an array of top trending keywords (obtained after google search – for both Twitter and Google). This increases the possibility of gathering a very large amount of data, especially considering the different REST calls going through every time with a different keyword search. The same keyword array is also used for streaming tweets with keyword filtering, where tweets matching any keyword present in the array is received.

This approach has actually enhanced the tweet count and the amount of data collected as is evident from the below screenshot where the amount of data collected using keyword filtering stream is more than no filter stream data count.



# BASIC DATA ANALYTICS: TWITTER

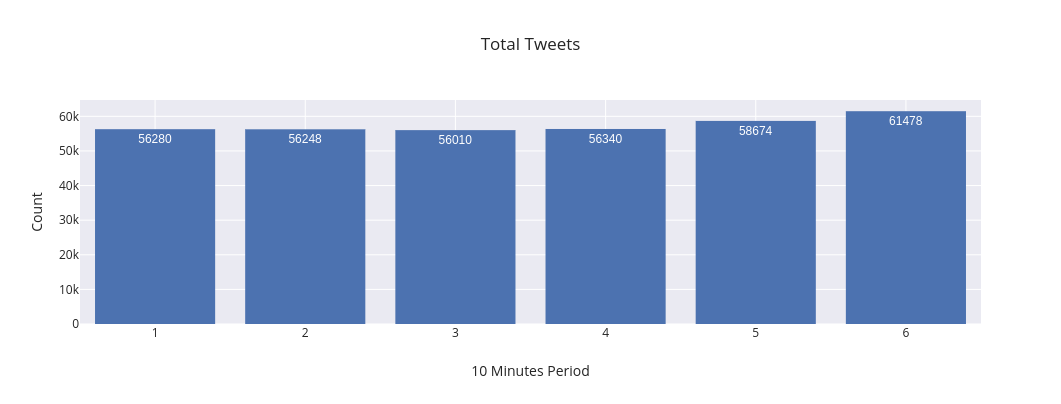
The amount of data collected varies on the time and day of the week where the data is collected. It also varies according to the type of filtering used, a number of keywords used for filtering and also on the number of API calls done in a fixed period of time.

## TOTAL DATA COLLECTED

The total amount of data collected using all the REST calls and all 3 STREAM types is different for different runs. Total data collected during the first run was 145919. This was the data when keyword used in REST call was “Morning” and calls were made at 5 minutes interval for 1 hour.

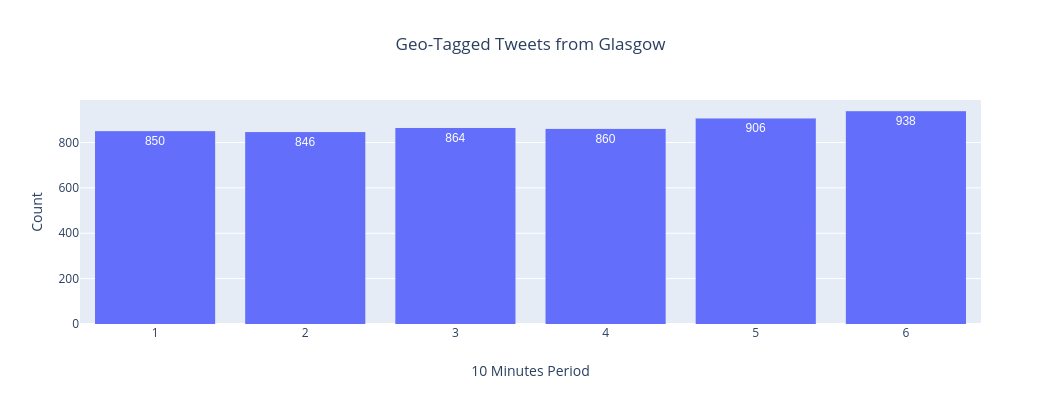
During the second run, the total data collected was 343991 when the rest calls were made at 10 seconds interval for 1 hour and the keyword array was used for filtering keyword search.

During the 10 minutes interval, all data were collected for 6 times and the following graph was obtained:



## TWEETS FROM GLASGOW

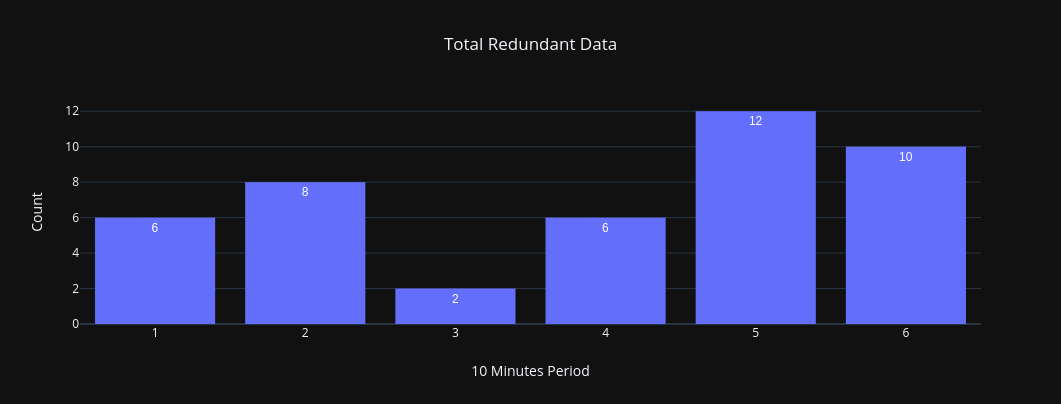
The total data collected from Glasgow over the different 10-minute interval in an hour using all 4 APIs are shown below.



It may be concluded from the above graph that not many users are present who have enabled geo-information in their tweets. This conclusion is done on comparing the data of total tweets received.

## TOTAL REDUNDANT DATA

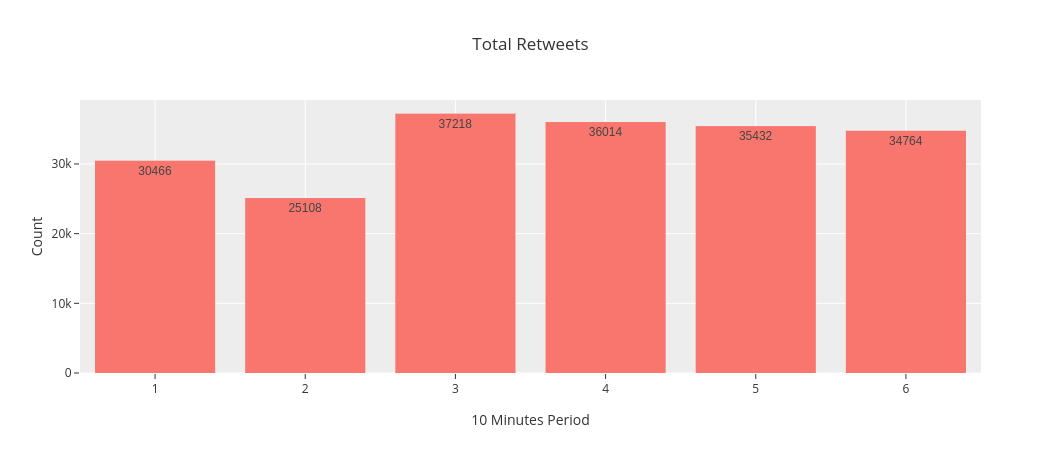
Since the data collection mechanism involves 4 different ways, namely, REST calls, and 3 streams (unfiltered, keyword filtered and location filtered), it is very much possible that the same tweet may have been captured by two or more APIs. The redundant data collected is depicted from the below graph:



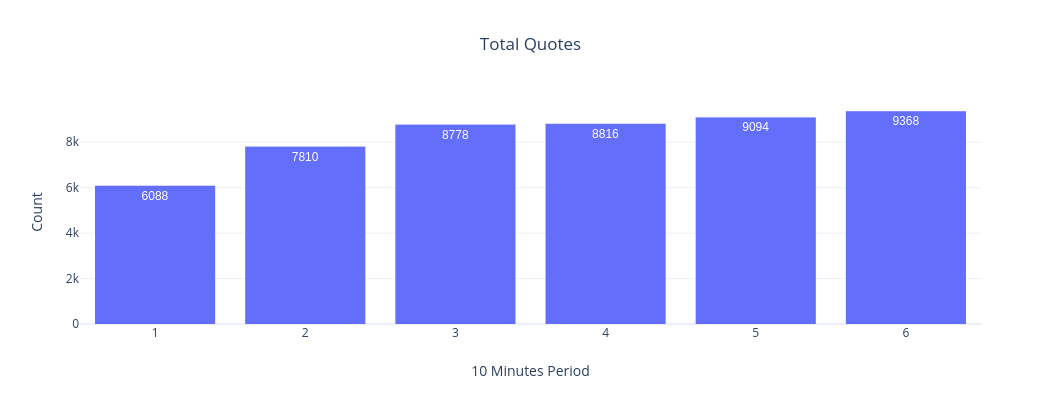
From the above graph, it may be concluded that the mechanism to collect data works well, as the redundant data collected is almost negligible, especially considering the total tweets collected for every 10 minute period. Hence, most of the data collected from this software are unique in nature.

## RE-TWEETS AND QUOTES

The number of retweets collected for 10 minute period over one hour is evident from the below bar graph:



And, the number of tweets collected which have quotes in them are analyzed as below:



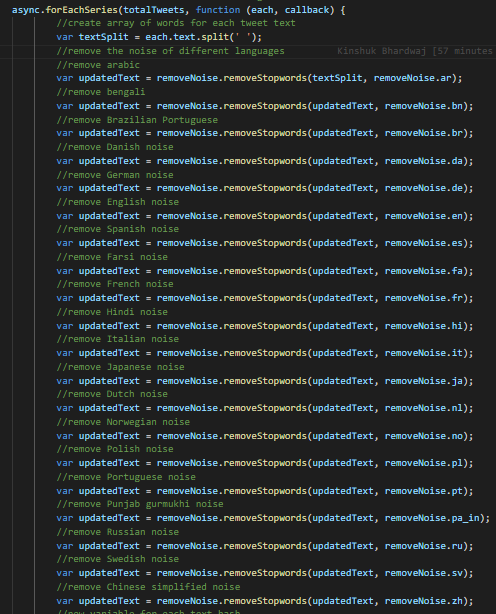
From the above 2 graphs, it is conclusive that, almost half of the total tweets or more (in some cases) are retweets and not the new tweets. Also, a fair amount of tweets have quotes contained in them when the total tweets for the same period are taken into consideration.

# ENHANCING THE GEOTAGGED DATA: TWITTER

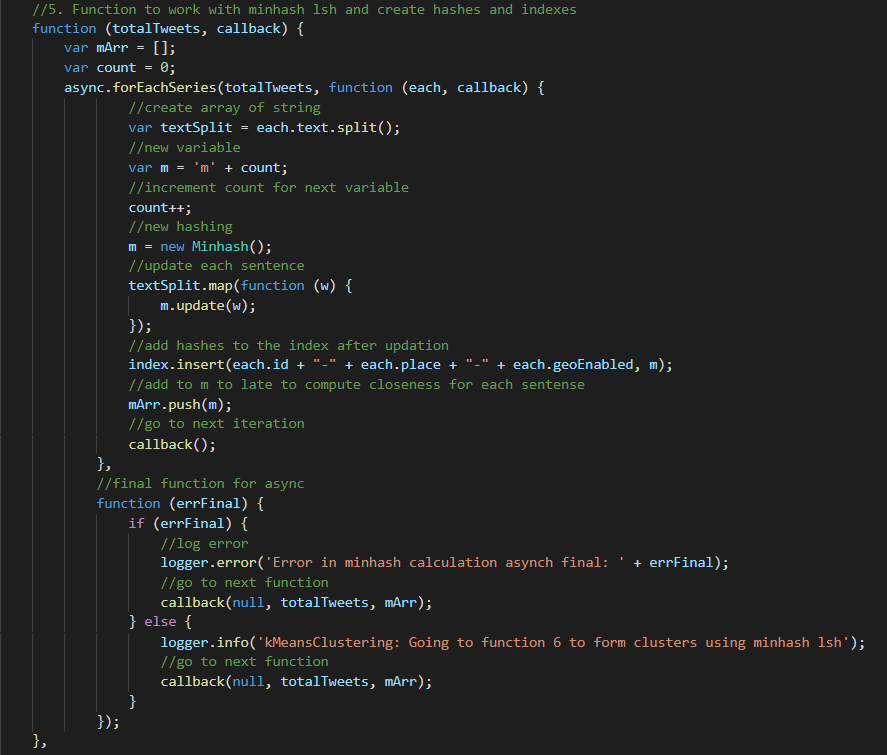
## GROUPING OF TWEETS: CLUSTERING

The main idea of collecting all tweets is to group them in a way that the similar tweets according to the text are placed together in a single group. This clustering is done by this software by an algorithm called Minhash Locality Sensitive Hashing. This algorithm is generally used for finding the similarity between sets, which are the texts in our use case. It is mainly used for finding a similar set of text and clustering them together. Jaccard similarity coefficient and minimum hash values are used to find the 2 similar text samples (Wikipedia, n.d.).

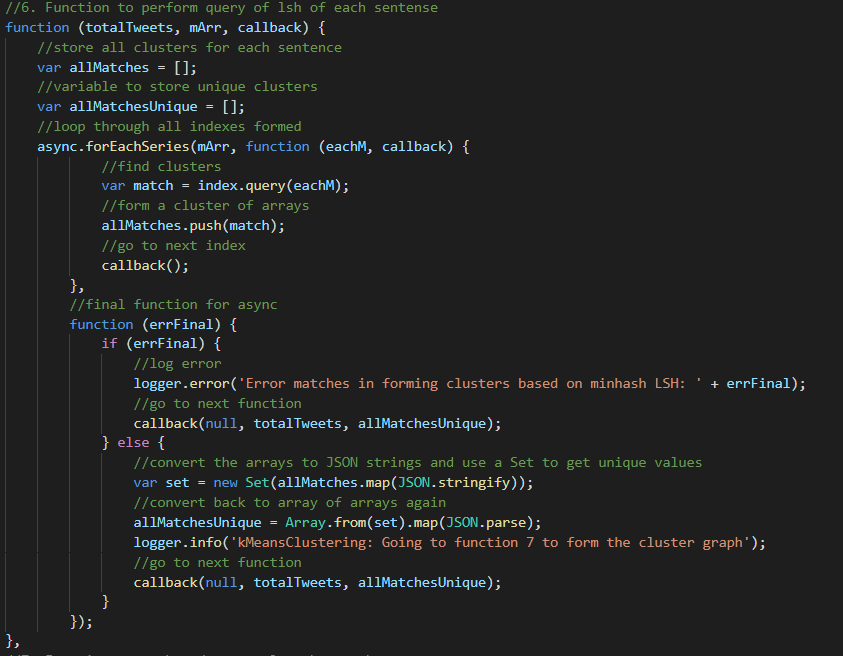
We use an implementation of this algorithm provided by the “Minhash” npm module. This software first takes all texts and removes all emojis and then creates an array of each text split by the space. The npm module named as “Stopword” is also used to remove the noisy words from the word array (Fergie, 2018). This is done for different languages to ensure the clusters formed are only done on the basis of important words.



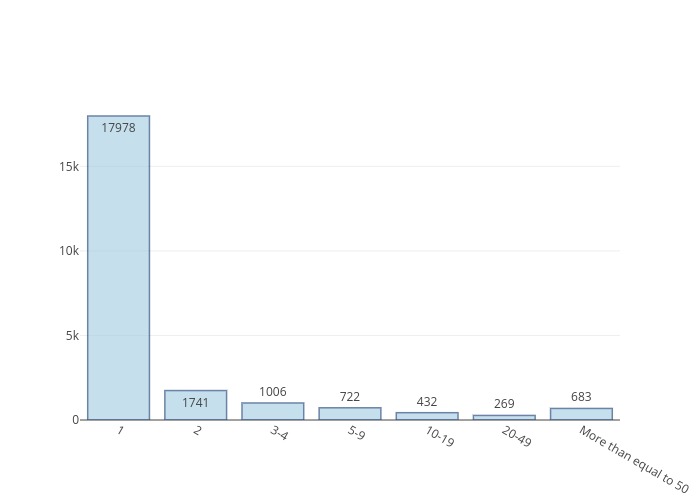
It then creates a hash for each set and updates it based on the existing hashes. Then all hashes are saved in an array. Then each of the text hash created is queried in the LSH index to find and group similar text sets into a single cluster. If a text has no similarity, then it is placed in a cluster with just one element.



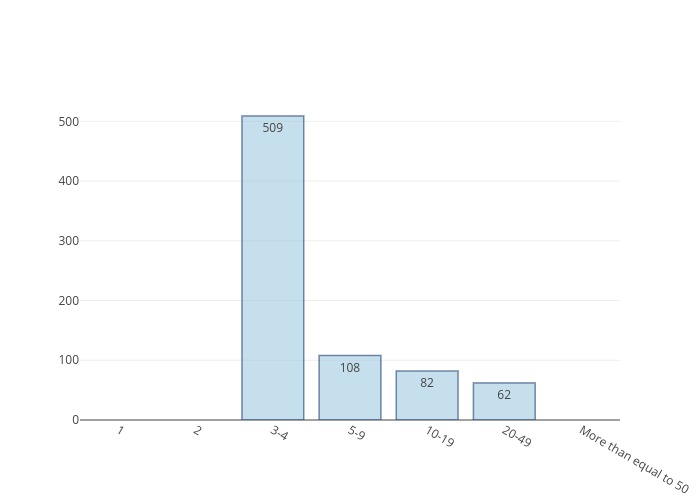
After the clustering is done, duplicate ones containing the same indexes are removed and unique ones are analyzed. The following graph shows the distribution of the clusters made with the number of elements in it.



The below graph depicts that most of the groups have no similar tweets. Others have the size of the cluster as two or three and few have more than 10 as the size of the cluster.

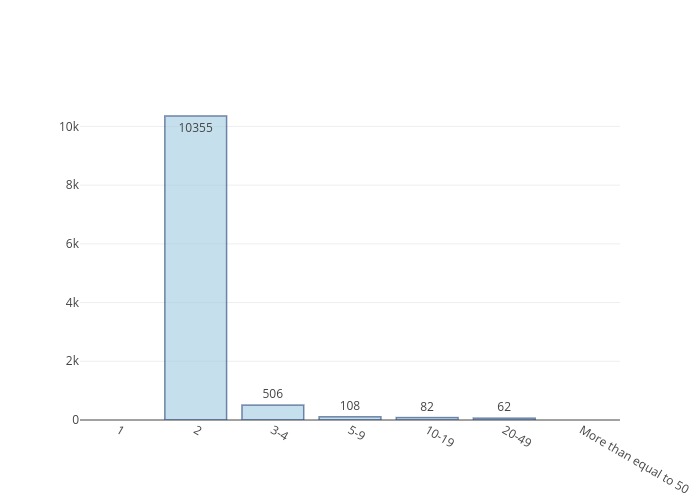


The next graph is showing the number of geo-tagged clusters among the clusters formed and their distribution among various sizes of the cluster.



It may be concluded that very few users are there who tweet with their geolocation enabled. This is in line with the conclusion made earlier with the total geo-location enabled data collected.

The following graph shows the distribution of the user profile geo information (the users who have geo\_enabled as true in their profile, meaning their profile location) among the clusters formed.



From the above 2 graphs, it may be concluded that a large number of users have their profile location on, but their geo-location for tweets is not enabled. This again coincides with the conclusion that very few geo-tagged tweets exist from the total tweets collected.

Thus, it is very much possible that profile information may be enabled for a user, which for example may be shown as Glasgow, but the geo-tagged location shows the user tweeting from another location in the world, for instance, London.

## GEO-LOCATION ASSIGNMENT TO CLUSTERS

The assignment of a geo-location to different clusters depends on the fact that if any tweet from the cluster has a location present within it or not. If no information is present, that cluster is not assigned any location.

The method used by this software to assign a cluster a location is right now using the given information in a way that it gives out the information about if a cluster belongs to the location Glasgow or not. The logic used in assigning this information is that, if a tweet is from Glasgow, then its place information would be Glasgow, which in turn may or may not have any coordinates assigned to it. A cluster is said to be belonging to the location Glasgow only on the following conditions:

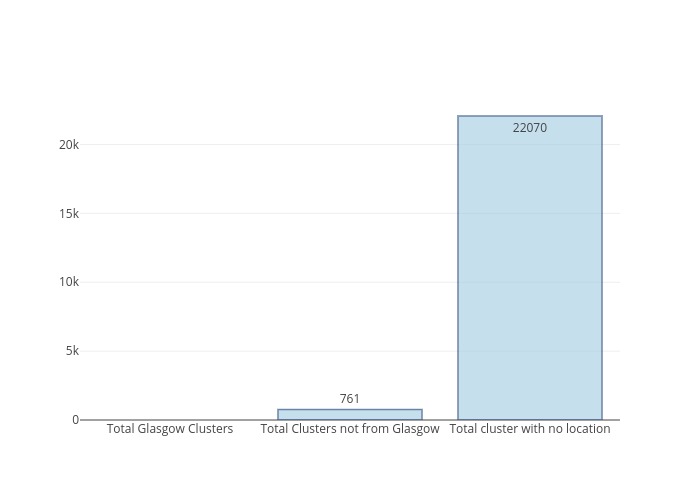
1. If all tweets in that cluster are from Glasgow.
2. If more than 50% of the tweets in a cluster belong to location Glasgow.
3. If one or more tweets in a cluster are from Glasgow and all other tweets have no location information at all.

The code implementation can be seen in the below screenshot:



Using this methodology, this software analyses the clusters formed and finds information about the number of Glasgow clusters, a number of clusters which are not from Glasgow, but some other location and all those clusters which have no location information at all.

The below graph is obtained on plotting the result of the method used by this software to assign each cluster a location:



Based on the graph generated, it can be assumed that no clusters belong to the location, Glasgow. In fact, most of the clusters have no location assigned to them at all, which is also seconds the previous observations that the maximum tweets have no geolocation data present in them.

Also, from the above graph, a number of clusters are present which are having a geo-located tweet in them, but they can be from any other part of the world, as tweets obtained using the different APIs may result in the collection of data from all over the world.

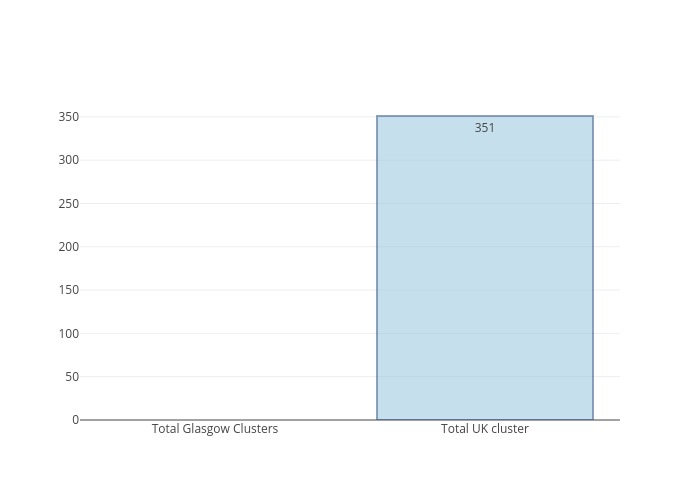
Finally, it can also be concluded that people may be talking about the same topics, but they may not be from the same location at that point of time as incoming tweets are distributed from all different parts of the world.

## EVALUATION OF METHOD

The above method is evaluated on a basis that, since the number of clusters belonging to Glasgow location came out to be zero, so, clusters are tried to be assigned a country – the United Kingdom. A similar approach is used for assignment of a country to each cluster as discussed in the section GEO-LOCATION ASSIGNMENT TO CLUSTERS.

The code snippet is similar to the one present in the previous section and is implemented on the exact same conditions, but this time, for the country as the United Kingdom.

The following graph is obtained for clusters assigned a location United Kingdom:



This method has an advantage that even if the exact coordinate of the tweet is not present, a rough idea about the user location may be estimated to a certain extent and hence assigned a location.

The limitation of this method is that it might not work for locations mentioned in other languages like Spanish, Japanese, etc. It is also quite possible that even if the tweet belongs to the United Kingdom, the actual tweet might be about a different location (like Spain) and yet the tweet’s actual location (which the user has chosen not to be shown) is totally in a third location (for example the U.S).

# DATA CRAWLING: TUMBLR

## INTRODUCTION: DATA COLLECTION MECHANISM

Along with the submission of the Twitter crawler, another software is submitted, which crawls the social media site named Tumblr (<https://www.tumblr.com/>).

The software uses a similar setup of JavaScript and Node.js (v8.12.0) (NPM version: 6.4.1) as the runtime environment. Tumblr has exposed official module names as “tumble.js” NPM module to access the different APIs of Tumblr (Tumblr, 2018).

The various NPM modules/libraries have been used for the software which are Body Parser, Cors, Express, Mongoose, Minhash, Tumblr.js, Winston and Winston Daily Rotate File (NPM, 2014).

## ACTUAL CODE ACCESS

The actual code base for this software can be found out at <https://github.com/Kinshuk1993/tumblrCrawler>.

## DATA RESTRICTIONS

The official Tumblr APIs and modules have a restriction that for every request made, only a maximum of 20 posts of any blog may be exposed to the developer, irrespective of the total number of the posts that particular blogs contain.

The restriction is such that the blogs returned for multiple calls for a blog return the exact same posts (which has been verified by the post id, which is a unique identifier). Multiple things have been tried, for example, putting a timer to the REST calls and trying to pass different parameters to collect different posts, but they return the exact same data for every iterative run of the software.

This restriction of maximum 20 can also be verified by looking at the response coming from the REST API call.

Also, few of the APIs like blogFollowers(), blogQueue(), blogDrafts() always throw an error when data is tried to be obtained using these. The assumption may be made that these APIs have either been deprecated or the names have been changed which Tumblr has not updated in their official JavaScript module.

## DATA COLLECTION APPROACH

This software collects the currently authenticated user and then goes through each of the blogs that he/she follows explores the blog’s data to crawl information in a restricted way. This restricted crawling is due to restriction from Tumblr end.

No timer approach has been used like the one used in Twitter as Tumblr returns the exact same data for one API call.

For data submission purpose, the authenticating user is only following 13 blogs, of which, one is named as staff, which is present by default for every Tumblr account.

## CODE SAMPLE

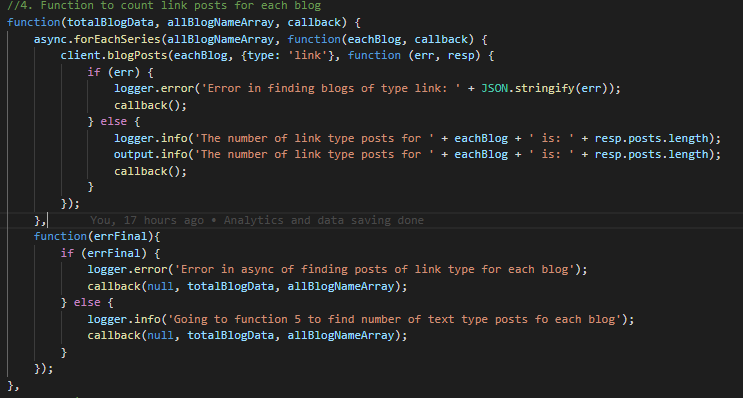
The code in the below screenshot is used to save the details of each blog data that the current authenticating user is following in the database and then do analytics on the data collected.



The below code is used to find the total posts (photo, links, and texts) of each blog the current user is following:



The following is the code sample to count the total number of link type posts for each blog. A similar approach has been followed and implemented to count the number of photo and text type posts for each blog the current user follows.

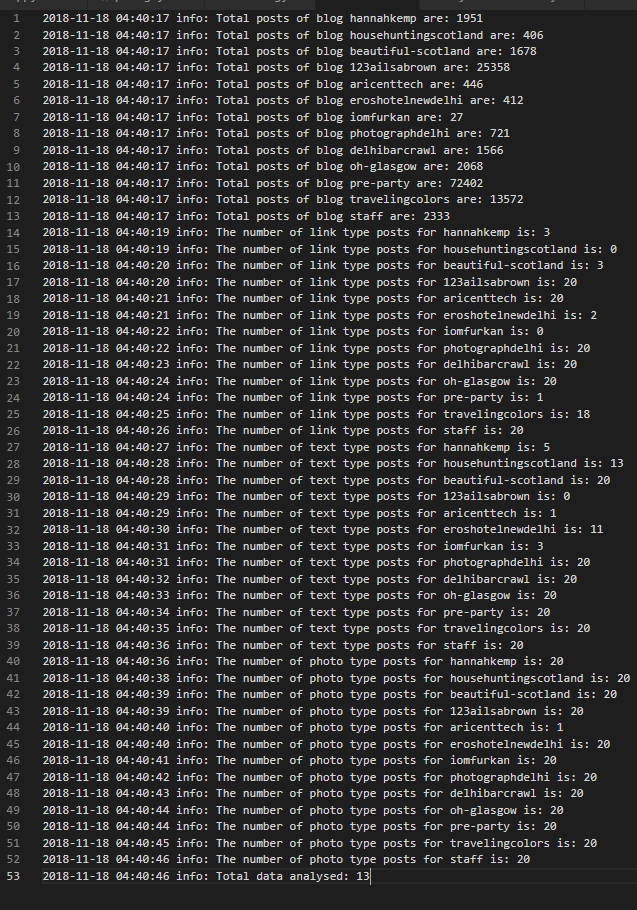


## DATA ANALYSIS

The data collected is for the authenticating user. The following analysis is done for the data collected:

1. The total number of blogs the authenticating user follows.
2. For each of the blog that the user follows, count the total number of posts each blog contains.
3. For each of the blog the user follows, count the total number of “Link” type posts each blog contains (maximum being 20).
4. For each of the blog the user follows, count the total number of “Text” type posts each blog contains (maximum being 20).
5. For each of the blog the user follows, count the total number of “Photo” type posts each blog contains (maximum being 20 posts).

The following screenshot of output logs shows the above analysis done on the data collected:



From the above data counting and analysis, it may be concluded that the authenticating user has a liking towards the blog posts of type “Photo” and is not much inclined towards “Link” or “Text” based blogs.

# BIBLIOGRAPHY

1. Fergie, E. a. (2018). *https://www.npmjs.com/package/stopword*. Retrieved from https://www.npmjs.com/package/stopword.
2. Hudgens, R. (2016). *https://www.siegemedia.com/seo/most-popular-keywords*. Retrieved from Siege Media.
3. Klokantech. (2018). *http://boundingbox.klokantech.com/*. Retrieved from Bounding box Klokantech.
4. Mondovo. (2018). *https://www.mondovo.com/keywords/twitter-keywords*. Retrieved from Mondovo.
5. NPM, I. (2014). *https://www.npmjs.com/*. Retrieved from NPM.
6. ttezel. (2018). *https://www.npmjs.com/package/twit*. Retrieved from https://www.npmjs.com/package/twit.
7. Tumblr. (2018). *https://github.com/tumblr/tumblr.js*. Retrieved from https://github.com/tumblr/tumblr.js.
8. Wikipedia. (n.d.). *https://en.wikipedia.org/wiki/MinHash*. Retrieved from https://en.wikipedia.org/wiki/MinHash.