%GrabTweet: A SAS® Macro to Read JSON Formatted Tweets

Isabel Litton, California Polytechnic State University, San Luis Obispo, CA

Rebecca Ottesen, California Polytechnic State University, San Luis Obispo, CA

ABSTRACT

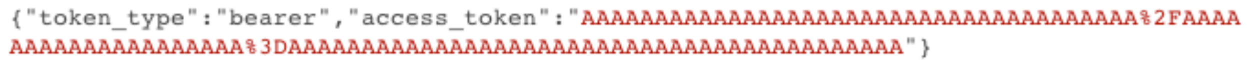
While social networking sites seek to connect people, they have recently flourished as a connection between businesses and consumers. Twitter, in particular, serves as a notable source of raw data businesses can analyze to discover more about consumers and their trends. Twitter recently updated their API to version 1.1, which only supports JSON format. Currently, there is not a lot of published code for SAS 9.3 to directly retrieve the JSON formatted tweets for API version 1.1 and read them into SAS as a cleaned and ready to analyze dataset. This paper describes the application of PROC HTTP within a macro to grab a specified amount of tweets with varying parameters.

INTRODUCTION

With thousands of tweets flying through the internet every second, Twitter is a rich source of information businesses can utilize to learn more about their consumers. In the past few months, Twitter dropped XML support and only supports the JSON format in API v1.1. This new change in format support has inspired a need for SAS code to grab tweets, parse the JSON format, and return a SAS data set. %GrabTweet is a simple macro that downloads recent tweets based on pre-specified queries by users in order to read the data into a SAS data set. In this paper we will provide basic Twitter background information, describe key components of the %GrabTweet macro, and illustrate an example of its use. We hope to provide a basis to obtain twitter data through SAS without having to resort to other programs.

BACKGROUND TWITTER INFORMATION

In order to access Twitter data and use the %GrabTweet macro an application-only authentication is required. An authenticated request is issued on behalf of the application, instead of a user, through an encoded set of credentials (token). Following the instructions on the Twitter Developer's page, there are two steps to obtain a bearer token. Step one involves Base64 encoding a consumer key and consumer secret to calculate a value needed for Step 2. The consumer key and secret strings are given once you have created a Twitter account and signed into the developers' page. Step 2 exchanges the value calculated from the Base64 encoding to obtain a bearer token through a request to POST oauth2/token. Once the request containing Step 1's value is issued, the bearer token is returned and will look similar to below:



{"token_type":"bearer","access_token":"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%2FAAAAAAAAAAAAAAAAAAAAA%3DAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"}

Figure 1. Example bearer token

A bearer token is similar to a password that essentially allows you to authenticate requests to API and start grabbing tweets through PROC HTTP.

For a more in-depth explanation and walk-through of obtaining a bearer token please refer to the following page: https://dev.twitter.com/docs/auth/application-only-auth.

%GRABTWEET MACRO

PROC HTTP

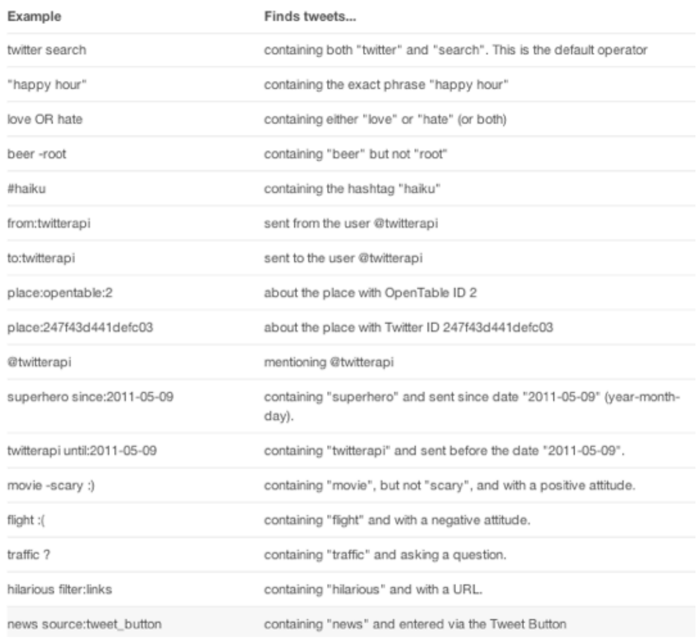
```
filename auth "C:\Users\Administrator\Documents\token.txt";
filename twtOut "C:\Users\Administrator\Documents\Tweets";

PROC HTTP
HEADERIN = auth
METHOD = "get"
URL =
"https://api.twitter.com/1.1/search/tweets.json?q=%23wuss13&max_id=2501261998405181&re
sult_type=recent&count=100"
OUT = twtOut;
```

The key parts of the PROC HTTP code above are the HEADERIN and URL arguments. The HEADERIN argument should reference a file that only contains the string “Authorization: Bearer” followed by the bearer token and nothing else. The file comprising of the bearer token authorizes the HTTP request to Twitter’s API to get tweets.

The string in the URL argument returns a specified amount of the most recent tweets matching a specified query. The user can control the following optional parameters to alter what tweets will be returned:

> **q =** Search term you want tweets to contain. Hashtags are represented by ‘%23’. Other options can be found in Figure 2 from the twitter search page (https://dev.twitter.com/docs/using-search). To construct a query containing these operators, you need to ensure that the operators are properly UTF-8, URL encoded. The proper conversions for the operator symbols can be found at (http://en.wikipedia.org/wiki/URL_encoding).



Example	Finds tweets...
twitter search	containing both "twitter" and "search". This is the default operator
"happy hour"	containing the exact phrase "happy hour"
love OR hate	containing either "love" or "hate" (or both)
beer -root	containing "beer" but not "root"
#haiku	containing the hashtag "haiku"
from:twitterapi	sent from the user @twitterapi
to:twitterapi	sent to the user @twitterapi
place:opentable:2	about the place with OpenTable ID 2
place:247f43d441defc03	about the place with Twitter ID 247f43d441defc03
@twitterapi	mentioning @twitterapi
superhero since:2011-05-09	containing "superhero" and sent since date "2011-05-09" (year-month-day).
twitterapi until:2011-05-09	containing "twitterapi" and sent before the date "2011-05-09".
movie -scary :)	containing "movie", but not "scary", and with a positive attitude.
flight :(	containing "flight" and with a negative attitude.
traffic ?	containing "traffic" and asking a question.
hilarious filter:links	containing "hilarious" and with a URL.
news source:tweet_button	containing "news" and entered via the Tweet Button

Figure 2 Twitter Search Operators

Example searches in both query form and URL encoded form in the macro call are shown in table 1.

Query	Macro call
love OR hate	`%GrabTweet(love%20ORhate, #);`
beer -root	`%GrabTweet(beer%20-root, #);`
from:twitterapi	`%GrabTweet(from%3Atwitterapi, #);`
@twitterapi	`%GrabTweet(%40twitterapi, #);`
superhero since:2011-05-09	`%GrabTweet(superhero%20since2011-05-09, #);`
flight :(	`%GrabTweet(flight%20%3A%28, #);`

Table 1 Query Examples

> **max_id =** Query will return tweets with ID value less than or equal to specified ID. In other words, older tweets than the one specified.
>
> **result_type =** Defines type of search results returned. Result_type can have one of the following values:
>
> > **popular =** only the most popular tweets with regard to search term
> >
> > **recent =** only the most recent tweets with regard to search term
> >
> > **mixed =** both popular and recent tweets
>
> **count =** Amount of tweets returned with a Twitter specified max of 100 and a default value of 15.

It is important to note that these parameters are preceded by an ampersand (&) in the URL and do not represent SAS macro variables. As shown in the URL argument in the PROC HTTP code above, to add an additional parameter the user must add an ampersand (&) before the parameter.

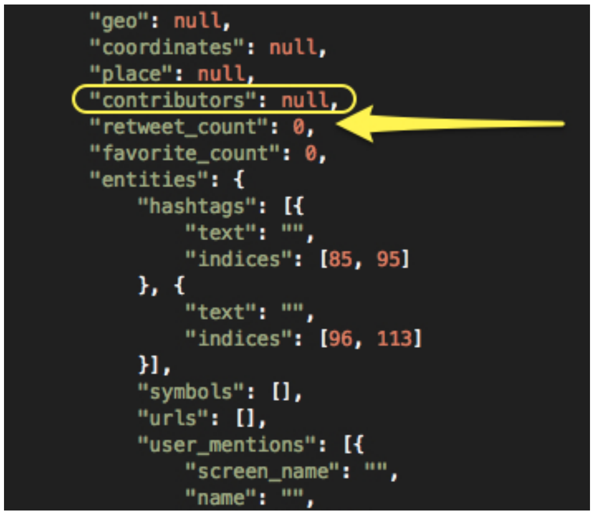
Twitter has limitations in how much data can be downloaded at a time. The max number of tweets in a data pull is 100, although you can repeatedly go back for more data 100 tweets at a time. This is further complicated by the page parameter that only reads relative to the top of the entire list of searched tweets and does not take into account the tweets frequently being added to the list. Because Twitter timelines are constantly changing with tweets added every second, new tweets might be added in between calls to grab more data, which may cause the processing of tweets already read. Since the max_id parameter will only return tweets older than the tweet specified and reads data relative to tweet id, the max_id parameter is the ideal solution to deal with Twitter's real time nature.

Therefore the solution in the %GrabTweet macro is this max_id parameter which is the key for grabbing more data when the user specifies an amount greater than the max value of 100. Using CALL SYMPUT to store the tweet id of the last observation read on the previous data pull allows us to specify that value in the max_id parameter of the URL argument within PROC HTTP for the next data pull. This will return tweets older than the last tweet read. Max_id is a more efficient solution to continuously grab more tweets than the page parameter that was used with the XML format because the query is relative to tweet ids of observations already processed.

HANDLING ORIGINAL TWEETS OF RETWEETS

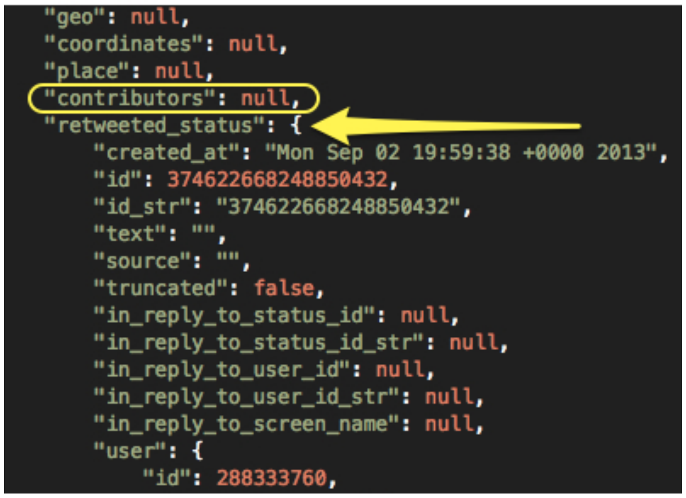
PROC HTTP will return a JSON file containing the most recent tweets in regard to the specified query. When retweets are included, the original tweet and details concerning the original tweet are embedded and will be included as well. Reading the original tweet causes a problem with the max_id parameter because some original tweets of retweets are much older. If an older original tweet was the last observation read, the CALL SYMPUT would use its tweet id as the new value for the max_id parameter when we really wanted the tweet id of the next most recent tweet instead. The subsequent call to grab more tweets would return older tweets relative to the old original tweet of the retweet instead of the retweet, and a section of data would be missed in the process.

In order to distinguish between the original tweet of a retweet and the next most recent tweet, we need to examine the structure of a regular tweet and a retweet. Portions of the JSON formatted regular tweet and retweet are shown below:



```
    "geo": null,
    "coordinates": null,
    "place": null,
    "contributors": null,
    "retweet_count": 0,
    "favorite_count": 0,
    "entities": {
        "hashtags": [{
            "text": "",
            "indices": [85, 95]
        }, {
            "text": "",
            "indices": [96, 113]
        }],
        "symbols": [],
        "urls": [],
        "user_mentions": [{
            "screen_name": "",
            "name": "",
```

Figure 3.1 Part of a regular tweet



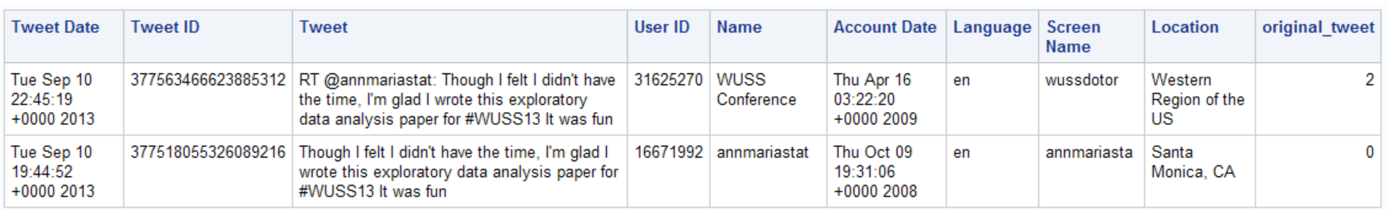
```
    "geo": null,
    "coordinates": null,
    "place": null,
    "contributors": null,
    "retweeted_status": {
        "created_at": "Mon Sep 02 19:59:38 +0000 2013",
        "id": 374622668248850432,
        "id_str": "374622668248850432",
        "text": "",
        "source": "",
        "truncated": false,
        "in_reply_to_status_id": null,
        "in_reply_to_status_id_str": null,
        "in_reply_to_user_id": null,
        "in_reply_to_user_id_str": null,
        "in_reply_to_screen_name": null,
        "user": {
            "id": 288333760,
```

Figure 3.2 Part of a retweet tweet

Notice that following the string ""contributors": null, " are different pairs. For a regular tweet, there is no new object but just another string and its corresponding value. For a retweet, there is a new object called "retweeted_status" which contains members regarding the original tweet. Searching for the phrase "RT" in the text of the tweet is not enough to distinguish between a regular tweet and original tweet because not all retweets have the phrase "RT". In order to distinguish between a tweet and a retweet, we must use the data immediately following this string, ""contributors": null," as shown in the following code.

```
original_tweet = FIND(retweeted, "retweeted_status");
temp = LAG(original_tweet);
IF temp = 0 THEN CALL SYMPUT("tweet_id", tweet_id);
```

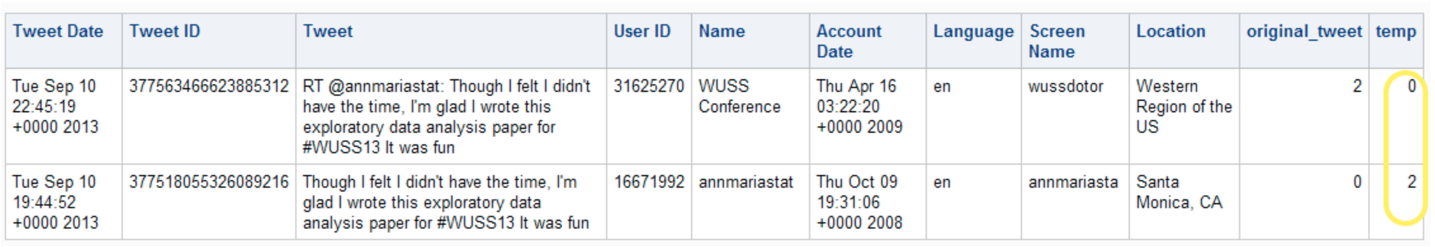
If the FIND function finds the string, “retweeted_status” in the characters following “”contributors”:null,”, the function will return its position (a non-zero integer). The non-zero integer, however, will be associated with the retweet observation instead of the tweet read after the retweet (original tweet) as displayed below.



Tweet Date	Tweet ID	Tweet	User ID	Name	Account Date	Language	Screen Name	Location	original_tweet
Tue Sep 10 22:45:19 +0000 2013	377563466623885312	RT @annmariastat: Though I felt I didn't have the time, I'm glad I wrote this exploratory data analysis paper for #WUSS13 It was fun	31625270	WUSS Conference	Thu Apr 16 03:22:20 +0000 2009	en	wussdotor	Western Region of the US	2
Tue Sep 10 19:44:52 +0000 2013	377518055326089216	Though I felt I didn't have the time, I'm glad I wrote this exploratory data analysis paper for #WUSS13 It was fun	16671992	annmariastat	Thu Oct 09 19:31:06 +0000 2008	en	annmariasta	Santa Monica, CA	0

Figure 4.1 Sample tweets with original_tweet variable

In order to use the value of original_tweet to flag for the max_id of regular tweets, the value must be brought down from the first observation to the second observation in Figure 4.1. The LAG function becomes necessary to associate the value of original_tweet in the previous observation to the current observation. As shown in Figure 3.2, retweets will always have the object, “retweeted_status”, with its members describing the original tweet of the retweet. Therefore, a retweet and its corresponding original tweet will always be next to each other, making the LAG function an efficient solution to always identify an original tweet from a regular tweet. The lagged variable, temp, now relates the non-zero integers of original_tweet to the observation representing original tweets of retweets as illustrated in Figure 4.2.



Tweet Date	Tweet ID	Tweet	User ID	Name	Account Date	Language	Screen Name	Location	original_tweet	temp
Tue Sep 10 22:45:19 +0000 2013	377563466623885312	RT @annmariastat: Though I felt I didn't have the time, I'm glad I wrote this exploratory data analysis paper for #WUSS13 It was fun	31625270	WUSS Conference	Thu Apr 16 03:22:20 +0000 2009	en	wussdotor	Western Region of the US	2	0
Tue Sep 10 19:44:52 +0000 2013	377518055326089216	Though I felt I didn't have the time, I'm glad I wrote this exploratory data analysis paper for #WUSS13 It was fun	16671992	annmariastat	Thu Oct 09 19:31:06 +0000 2008	en	annmariasta	Santa Monica, CA	0	2

Figure 4.2 Sample tweets with lagged original_tweet variable

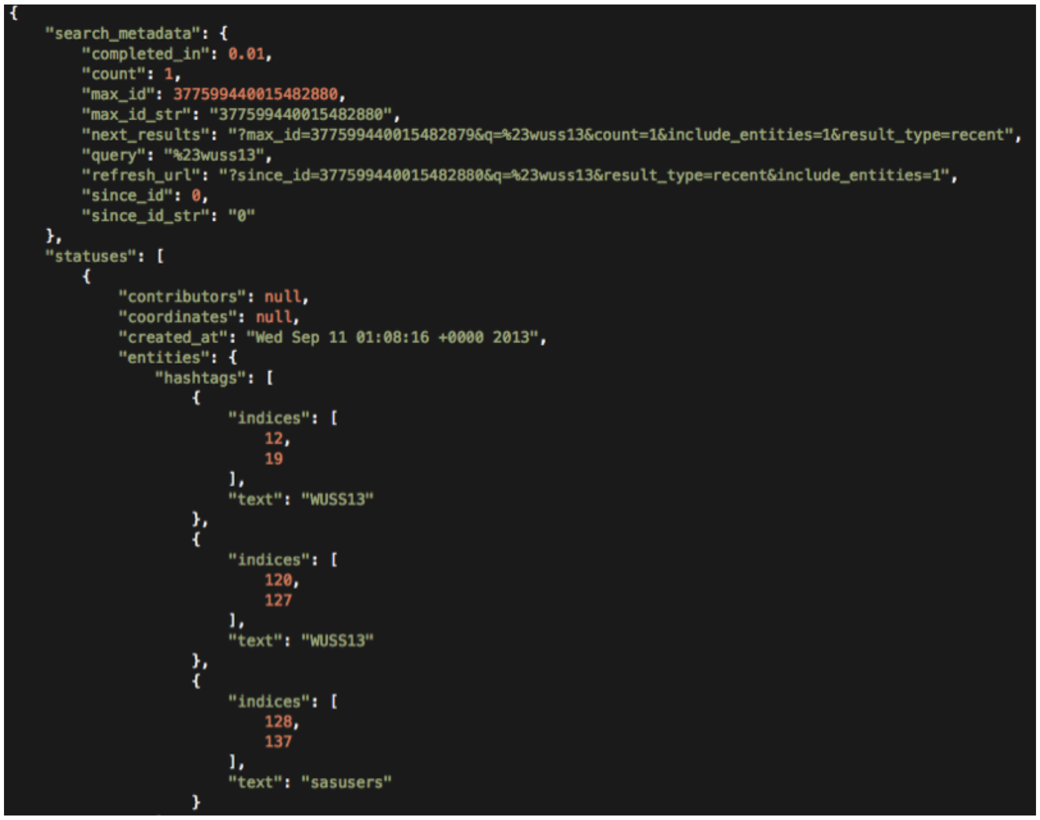
Now the variable, temp, can be used in a subsetting IF statement to flag observations corresponding to original tweets.

EXAMPLE

```
%GrabTweet(%23wuss13, 10);
```

In this example the search_term parameter is passed, “%23wuss13, which means the macro will grab tweets containing the phrase “#wuss13”. The number 10 is passed for the count parameter so only 10 observations will be returned.

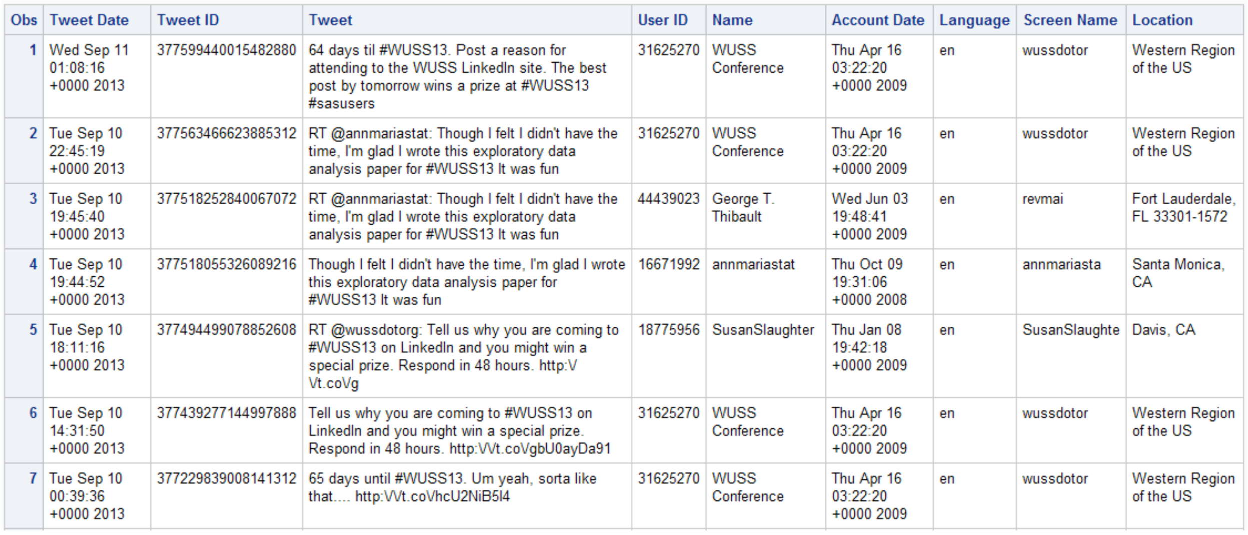
Part of the JSON file returned by PROC HTTP containing the tweets is shown below in figure 5.



```
{
    "search_metadata": {
        "completed_in": 0.01,
        "count": 1,
        "max_id": 377599440015482880,
        "max_id_str": "377599440015482880",
        "next_results": "?max_id=377599440015482879&q=%23wuss13&count=1&include_entities=1&result_type=recent",
        "query": "%23wuss13",
        "refresh_url": "?since_id=377599440015482880&q=%23wuss13&result_type=recent&include_entities=1",
        "since_id": 0,
        "since_id_str": "0"
    },
    "statuses": [
        {
            "contributors": null,
            "coordinates": null,
            "created_at": "Wed Sep 11 01:08:16 +0000 2013",
            "entities": {
                "hashtags": [
                    {
                        "indices": [
                            12,
                            19
                        ],
                        "text": "WUSS13"
                    },
                    {
                        "indices": [
                            120,
                            127
                        ],
                        "text": "WUSS13"
                    },
                    {
                        "indices": [
                            128,
                            137
                        ],
                        "text": "sasusers"
                    }
```

Figure 5. JSON file containing tweets

Once the JSON file is read into SAS through a series of FIND, INDEX, and SUBSTR functions, the output should look similar to figure 6.



Obs	Tweet Date	Tweet ID	Tweet	User ID	Name	Account Date	Language	Screen Name	Location
1	Wed Sep 11 01:08:16 +0000 2013	377599440015482880	64 days til #WUSS13. Post a reason for attending to the WUSS LinkedIn site. The best post by tomorrow wins a prize at #WUSS13 #sasusers	31625270	WUSS Conference	Thu Apr 16 03:22:20 +0000 2009	en	wussdotor	Western Region of the US
2	Tue Sep 10 22:45:19 +0000 2013	377563466623885312	RT @annmariastat: Though I felt I didn't have the time, I'm glad I wrote this exploratory data analysis paper for #WUSS13 It was fun	31625270	WUSS Conference	Thu Apr 16 03:22:20 +0000 2009	en	wussdotor	Western Region of the US
3	Tue Sep 10 19:45:40 +0000 2013	377518252840067072	RT @annmariastat: Though I felt I didn't have the time, I'm glad I wrote this exploratory data analysis paper for #WUSS13 It was fun	44439023	George T. Thibault	Wed Jun 03 19:48:41 +0000 2009	en	revmai	Fort Lauderdale, FL 33301-1572
4	Tue Sep 10 19:44:52 +0000 2013	377518055326089216	Though I felt I didn't have the time, I'm glad I wrote this exploratory data analysis paper for #WUSS13 It was fun	16671992	annmariastat	Thu Oct 09 19:31:06 +0000 2008	en	annmariasta	Santa Monica, CA
5	Tue Sep 10 18:11:16 +0000 2013	377494499078852608	RT @wussdotorg: Tell us why you are coming to #WUSS13 on LinkedIn and you might win a special prize. Respond in 48 hours. http:\/\/t.co\/g	18775956	SusanSlaughter	Thu Jan 08 19:42:18 +0000 2009	en	SusanSlaughte	Davis, CA
6	Tue Sep 10 14:31:50 +0000 2013	377439277144997888	Tell us why you are coming to #WUSS13 on LinkedIn and you might win a special prize. Respond in 48 hours. http:\/\/t.co\/gbU0ayDa91	31625270	WUSS Conference	Thu Apr 16 03:22:20 +0000 2009	en	wussdotor	Western Region of the US
7	Tue Sep 10 00:39:36 +0000 2013	377229839008141312	65 days until #WUSS13. Um yeah, sorta like that.... http:\/\/t.co\/hcU2NiB5l4	31625270	WUSS Conference	Thu Apr 16 03:22:20 +0000 2009	en	wussdotor	Western Region of the US

Figure 6. Partial output of GrabTweet(%23wuss2013, 10)

CONCLUSION

Twitter has become a social media giant with millions of active users tweeting valuable data every second, and what better way to handle a data giant than with SAS? The %GrabTweet macro strives to connect SAS with Twitter API v1.1 to utilize Twitter's lucrative source of data. The macro applies the NODUPKEY option in PROC SORT in order to remove duplicate tweets due to the use of the max_id parameter in the Twitter search URL as well as identical original tweets of retweets. The tweet used for the max_id parameter will be the last observation from one data pull as well as the first observation from the subsequent data pull, so using the NODUPKEY will result in one copy of this observation. Some regular tweets may be retweets of the same tweet, which means the original tweet will be the same for those retweets, causing duplicate original tweets to be read. Once again, the NODUPKEY option will remove all duplicates of those original tweets.

Finally, there are two important caveats that must be addressed if the user chooses to use this macro to include original tweets of retweets. One, the date of the tweet variable will appear strange due to the older dates of original tweets mixed with the similar recent dates and times of regular tweets. The user should pay extra attention to the tweet date variable to make sure the observations fall in the correct timeframe for analysis. Second, the macro may return more observations than the target amount specified due to the inclusion of these original tweets. The user should not expect the number of observations returned to exactly match the target amount specified.

The output of tweets can be transformed into useful information for businesses through text mining and sentiment analysis. Future improvement of the %GrabTweet macro include adding more user specified parameters such as geocode location and language, as well as more options for searching such as dates and frequency of data download.

REFERENCES

- Choy, Murphy. Shim, Kyong Jin. 2013. "Efficient extraction of JSON information in SAS® using the SCANOVER function." Proceedings of the SAS Global 2013 Conference. Available at http://support.sas.com/resources/papers/proceedings13/296-2013.pdf.
- Garla, Satish. Chakraborty, Goutam. 2011. "%GetTweet: A New SAS® Macro To Fetch and Summarize Tweets." Proceedings of the SAS Global 2011 Conference. Available at: http://support.sas.com/resources/papers/proceedings11/324-2011.pdf.
- Munoz, John. "SAS and Twitter- how to harness SAS to grab data from Twitter in 2 easy steps." August 12, 2010. Available at http://bzintelguru.com/blog/sas-and-twitter-how-to-harness-sas-to-grab-data-from-twitter-in-2-easy-steps/.
- "Application-only Authentication". March 11th, 2013. Available at https://dev.twitter.com/docs/auth/application-only-auth.
- "GET search/tweets". March 7th, 2013. Available at https://dev.twitter.com/docs/api/1.1/get/search/tweets.
- "Percent-encoding". September 16th, 2013. Available at http://en.wikipedia.org/wiki/URL_encoding.
- "SAS-Problem-5:Number of Observations in a Dataset". September 17, 2009. Available at http://www.analytics-tools.com/2009/09/sas-problem-5number-of-observations-in.html.
- "Using the Twitter Search API." Available at https://dev.twitter.com/docs/using-search.
- "Working with Timelines". September 7th, 2012. Available at https://dev.twitter.com/docs/working-with-timelines.

ACKNOWLEDGMENTS

The authors would like to thank the Cal Poly Statistics department for the opportunity to conduct summer research.

RECOMMENDED READING

- Twitter Developers Documentation for Rest API v1.1

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name:	Isabel Litton
Enterprise:	California Polytechnic State University
E-mail:	ilitton@calpoly.edu

Name:	Rebecca Ottesen
Address:	1 Grand Avenue, Statistics Dept
City, State ZIP:	San Luis Obispo, CA, 93407-0405
Work Phone:	805-756-2709

Fax:	805-756-2700
E-mail:	rottesen@calpoly.edu
Web:	http://statweb.calpoly.edu/rottesen/

APPENDIX

```
/*******************************************************************************
Macro: GrabTweet
Parameters:
       Search_Term = Word or phrase you want tweets to contain. Represent hashtags (#)
       with %23.
       Target_Amount = Number of tweets you want returned.
Requirements:
       Bearer token from Twitter Developers' Page to authorize requests to Twitter API
       (Not provided)
       %Grab_Tweet_100 Macro (Provided)
       %Grab_Tweet_GT_100 Macro (Provided)
Output Variables:
       created_at1 = Date of tweet
       tweet_id = Tweet ID
       text = Tweet
       user_id = User ID
       name = Name of account
       created_at2 = Date account was created
       lang = Language
WARNING: If original tweets are included, the macro will return more observations than
the specfied target amount.
******************************************************************************/


%MACRO grab_tweet_100(search_term, target_amount);
/*Specifies the input token file for authorization to access API 1.1*/
filename auth "C:\Users\Administrator\Documents\token.txt";
/*Specifies the output JSON file that will contain the tweets*/
filename twtOut "C:\Users\Administrator\Documents\data\Tweets";

/*Sets the following parameters to use in the GET search/tweets url:
         COUNT = number of tweets to grab
         TYPE = what type of search tweets to grab. Options include the following:
              --> Popular = returns only the most popular tweets with regard to search
              term
              --> Recent = returns only the most recent tweets with regard to search
              term
              --> Mixed = popular and real time tweets
*/
%IF &target_amount < 100 %THEN %LET num_tweet = %NRSTR(&count=)&target_amount;
%ELSE %LET num_tweet = %NRSTR(&count=100);
%LET type = %NRSTR(&result_type=recent);

/*Issues GET search/tweet URL with search term and number of tweets specified by user
to output JSON file*/
PROC HTTP
HEADERIN = auth
METHOD = "get"
URL = "https://api.twitter.com/1.1/search/tweets.json?q=&search_term&type&num_tweet"
OUT = twtOut;
RUN;

DATA tweet (DROP= text_start text_end userid_end name_end retweeted original_tweet
temp);
INFILE "C:\Users\Administrator\Documents\data\Tweets" LRECL = 1000000 TRUNCOVER
SCANOVER DLM=',' DSD;
INFORMAT created_at1 $30. tweet_id $18. text $140. user_id $15. name $185. screen_name
$185. location $185. created_at2 $30. lang $2. retweeted $18.;
INPUT @'"created_at":' created_at1
      @'"id":' tweet_id
      @'"text":' text
      @'"user":{"id":' user_id
      @'"name":' name
      @'"screen_name":' screen_name
      @'"location":' location
      @'"created_at":' created_at2
      @'"lang":' lang
      @'"contributors":null,' retweeted @@;
text_start = INDEX(text, '"')+1;
text_end = INDEX(text,'","');
IF text_end ~= 0 THEN text = SUBSTR(text,text_start,text_end-1);
IF text_end = 0 THEN text = SUBSTR(text, text_start);
userid_end = INDEX(user_id, ',"');
IF userid_end ~= 0 THEN user_id = SUBSTR(user_id,1,userid_end-1);
name_end = INDEX(name, '","s');
IF name_end = 1 THEN DO;
       name_end = 2;
       name = 'NA';
END;
IF name_end ~IN(0,1) THEN name = SUBSTR(name, 1, abs(name_end-1));

/*The macro is suppose to return only the most recent tweets, but the file contains
the original tweets of retweets, so we must distinguish between original tweets and
retweets using the variable Original_tweet.
Retweets will have the string "retweeted_status" as a member while non-tweets will
have a different string.
Since original_tweet will be a nonzero integer for retweets instead of the original
tweets, the variable must be lagged so that "retweeted_status" will be part of the
observation representing original tweets.
Now that original_tweet is a nonzero integer for original tweets, these observations
can be flagged, and regular tweets(whose value of temp will be zero) can be used in
the CALL SYMPUT function to grab the tweet_id for the max_id URL parameter.*/
original_tweet = FIND(retweeted, "retweeted_status");
temp = LAG(original_tweet);

/*Optional code to delete original tweets of retweets.*/
*IF temp = 2 THEN DELETE;

/*In order to grab more tweets and return older tweets (Tweets with IDs lower than
specified tweet_id), we need to store the value of tweet_id in a macro variable to be
used in %Grab_Tweet_GT_100. The stored value of &tweet_id will be used as the max_id
parameter in the GET search/tweets url.*/
IF temp = 0 THEN CALL SYMPUT("tweet_id", tweet_id);
RUN;

%MEND;

%MACRO grab_tweet_gt_100(search_term, remain_tweet);
filename auth "C:\Users\Administrator\Documents\token.txt";
filename twtOut "C:\Users\Administrator\Documents\data\Tweets";


/*Sets parameters to use in the GET search/tweets url. Compared to grab_tweet_100,
this macro will also set the max_id parameter in the GET search/tweet url. Setting the
max_id parameter as the last tweet_id read will return tweets older than the specified
tweet. This is the key parameter in order to keep regrabbing more tweets that were not
previously grabbed. */

%GLOBAL tweet_id;
%IF &remain_tweet < 100 %THEN %LET num_tweet = %NRSTR(&count=)&remain_tweet;
%ELSE %LET num_tweet = %NRSTR(&count=100);
%LET id = %NRSTR(&max_id=)&tweet_id;
%LET type = %NRSTR(&result_type=recent);
```

```
PROC HTTP
HEADERIN = auth
METHOD = "get"
URL =
"https://api.twitter.com/1.1/search/tweets.json?q=&search_term&id&type&num_tweet"
OUT = twtOut;
RUN;

DATA tweet (DROP= text_start text_end userid_end name_end retweeted original_tweet
temp);
INFILE "C:\Users\Administrator\Documents\data\Tweets" LRECL = 1000000 TRUNCOVER
SCANOVER DLM=',' DSD;
INFORMAT created_at1 $30. tweet_id $18. text $140. user_id $15. name $185. screen_name
$185. location $185. created_at2 $30. lang $2. retweeted $18.;
INPUT @'"created_at":' created_at1
      @'"id":' tweet_id
      @'"text":' text
      @'"user":{"id":' user_id
      @'"name":' name
      @'"screen_name":' screen_name
      @'"location":' location
      @'"created_at":' created_at2
      @'"lang":' lang
      @'"contributors":null,' retweeted @@;
text_start = INDEX(text, '"')+1;
text_end = INDEX(text,'","');
IF text_end ~= 0 THEN text = SUBSTR(text,text_start,text_end-1);
IF text_end = 0 THEN text = SUBSTR(text, text_start);
userid_end = INDEX(user_id, ',"');
IF userid_end ~= 0 THEN user_id = SUBSTR(user_id,1,userid_end-1);
name_end = INDEX(name, '","s');
IF name_end = 1 THEN DO;
       name_end = 2;
       name = 'NA';
END;
IF name_end ~IN(0,1) THEN name = SUBSTR(name, 1, abs(name_end-1));
original_tweet = FIND(retweeted, "retweeted_status");
temp = LAG(original_tweet);
*IF temp = 2 THEN DELETE;
IF temp = 0 THEN CALL SYMPUT("tweet_id", tweet_id);
RUN;

%MEND;

%MACRO grabtweet(search_term, target_amount);
DATA main;
LENGTH created_at1 $30 tweet_id $18 text $140 user_id $15 name $185 created_at2 $30
lang $2 screen_name $185 location $185;
DELETE;
RUN;

/*DO UNTIL loop executes until the number of observations matches amount specified by
the user.
Within the loop, the number of observations and remaining number of observations left
to grab are calculated.
Also, a counter (i) is created in order to specify that the first iteration grabs the
first 100 tweets to create the base data set (if target amount is > 100).
The following iterations grab the remaining tweets if target amount is > 100.
PROC SORT with the NODUPKEY option is needed because the tweets whose id was used for
the max_id parameter is duplicated (once as the last observation and once as the first
observation of the subsequent set of tweets).
*/
```

```
%LET i = 0;
%LET nobs = 0;
%DO %UNTIL (&nobs = &target_amount);
       %LET i=%EVAL(&i+1);

       %LET dsid=%SYSFUNC(OPEN(main));
       %LET nobs=%SYSFUNC(ATTRN(&dsid,nobs));
       %IF &dsid > 0 %THEN %LET RC=%SYSFUNC(CLOSE(&dsid));
       %LET remaining = &target_amount - &nobs + 1;
       %LET remain_tweet = %SYSEVALF(&remaining, int);
       %IF &remain_tweet < 0 %THEN %RETURN;

       %IF &i = 1 %THEN %grab_tweet_100(&search_term, &target_amount);

       %ELSE %grab_tweet_gt_100(&search_term, &remain_tweet);

       PROC APPEND BASE = main DATA = tweet force;
       RUN;

       PROC SORT DATA = main NODUPKEY;
       BY DESCENDING tweet_id ;
       RUN;

/*Recalculates the number of observations in the data set after appending and sorting
the new file of tweets. The recalculation is needed to compare the number of
observations before and after appending the new set of tweets.*/
       %LET dsid2=%SYSFUNC(OPEN(main));
       %LET nobs2=%SYSFUNC(ATTRN(&dsid2,nobs));
       %LET RC2=%SYSFUNC(CLOSE(&dsid2));

/*%ABORT statement is necessary to break out of the DO UNTIL loop if there are less
tweets than the target amount specified.*/
       %IF &nobs = &nobs2 %THEN %ABORT;

%END;

%MEND;
```