* General notes
  + Settings and options
    - IMPORTANT: prevent scientific notation (problematic for user IDs), by running options(scipen=999)
  + All functions are built on rtweet (subject to updates, so keep an eye on it)
  + General speed notes:
    - I get about 1 tweet every .008 seconds
    - I get about 5,000 user IDs (friends or followers) every .5 seconds
    - user info is a bit slower- average of about .7 seconds per call if just getting one user. However, there’s plenty of rate limit (looks like 3 tokens would be sufficient).
      * getting 200 users at a time is also fine, doesn’t break tokens

**First Level Functions (built directly on rtweet version 0.6.0)**

* General notes on first level functions:
  + Most of these functions are just wrappers of the standard rtweet functions. I made them to better deal with rate limits with more tokens. In the standard functions, if no token is specified, the function, will “look around” for a non-exhausted token, and based on my testing, this process is very inefficient as the number of tokens increases. The function speed gradually decreases until the first token regenerates. In my wrappers, I create new variables in the environment that tell the function exactly where to get the token, and cycle through the list of tokens. The testing supports this method, as I find a much better overall speed, and no long delays.
  + An important issue that I’ve found is that even if I specify the token, rtweet will still use the first token to check rate limits, and will automatically switch to the second token to do so if the first token is exhausted. Thus, when starting over on the list of tokens, the functions go back to the 3rd token, not the first, which is often exhausted. I believe the first two tokens are sufficient to handle the rate limit checking.
* get\_friends2
  + dependencies: only rtweet version 0.6.0
  + a slight tweak of the original get\_friends function from rtweet, designed for more efficient use of the tokens.
  + Arguments to get\_friends can be passed through.
  + creates two variables in the global environment to keep track of which token to use, these will then be used any time the function is used again. Note that this separates token use into parallel tracks depending on the API call, which should be more efficient.
  + Errors- This function doesn’t check for participants with 0 friends, or participants who are set to private, or wrong IDs. The function breaks just like the original get\_friends function does.
    - 0 friends- a 1X2 Tibble with an empty list in the user\_id column
    - Private account: prints “private account found”, returns a logical vector with a single NA
    - Wrong ID- prints “mistaken ID”, returns a logical vector with a single NA
  + Returns a tibble that’s identical to the output of the traditional get\_friends function: a tibble with two columns.
    - Key point: the next\_cursor (from rtweet) function works fine on the output.
  + Testing and speed:
    - Speed testing confirms that this is a much better use of tokens.
    - Since this function is faster and doesn’t check rate limits, a larger number of tokens is required to keep it from breaking, if you’re really running it at max speed. This is if you’re asking for 5,000 friends repeatedly with no other calculations.
      * I think ~70 tokens would be enough to keep it from breaking at full speed
    - If running at full speed, should take approximately .5 seconds on average
  + Note: if the page argument is wrong, this function will give the same output as if the user has 0 friends.
* get\_followers2
  + dependencies: packages: rtweet version 0.6.0
  + Basically a wordswap of the get\_friends2 function. Has a few key differences
  + Differences:
    - Normal output: a tibble with 1 column of user IDs
    - Abnormal outputs:
      * Wrong ID: a 1X1 tibble filled with NA
      * Private account: a single logical vector with NA, prints “private account”
      * 0 followers: a 5X1 tibble filled with NA
  + Note: default N is 5000, not max of 75,000.
  + Testing and speed:
    - If running at full speed, each call (75,000 followers) takes about 8.75 seconds
  + Caution: In this code, I detect rate limit issues by checking the number of rows in the output from the get\_followers function. Usually, this is 0 when there’s a rate limit problem. However, I’ve had some instances where the getfollowers function gets an error when it hits the rate limit instead. I can’t pinpoint what makes it do this.
* get\_tweets
  + Overview: this is a function that returns all of a specified user’s tweets within the specified time frame. It also handles a number of abnormal situations.
    - Unlike the get\_followers2 and get\_friends2 functions, this function does NOT deal with tokens and the rate limit, because some testing I did suggested that this was best left up to the default rtweet method
  + dependencies- packages: rtweet version 0.6.0, tibble
  + input- user ID or screen name, as well as a beginning date and end date (must be in proper date format).
  + Note that the function gathers tweets from the start day and stop day as well as all tweets in between
  + Normal output- a dataframe of tweets (a tibble) with 42 columns.
  + Abnormal output
    - Private account: single NA value
    - 0 tweets in the time frame (or ever)- a dataframe (tibble) with 0 rows and 42 columns. Can be merged with other tweets dataframes without a problem.
  + Warnings: the Twitter API will only allow access to the most recent 3,220 (approx) tweets from a given user. If this limit is reached, this function will give inaccurate results (without breaking), and this should be checked if it’s a concern.
    - Additionally, this function will not detect if the account wasn’t in existence during the specified time frame.
  + Note on rate limits: in using the get\_timeline function, this function sets check=FALSE so that the rate limit is only checked once (not twice) every time the function is run. This is because the bottleneck for this function is the rate\_limit checking, not the actual getting of timelines.
  + Testing
    - Did quick test on 100 people, with no problems handling different types of accounts, and numbers of tweets up to
* getmentionees
  + Overview: This is a fairly simple function that just takes a set of tweets and counts which users are mentioned and how often. Doesn’t make calls to the Twitter API.
  + dependencies: none, but will only process tweets data from rtweets functions or those described here
  + Input: a tibble of tweets (either from the get\_tweets or get\_timeline functions).
  + Output: a tibble with two columns: “user\_ID” (a character vector listing each user), and “count” (a numeric vector listing how many times they were mentioned).
  + Note: mentions include replies and retweets as well, but not quote tweets.
  + Note: the output will include self mentions (e.g., via “ego retweets”)
  + Note: mentions of deleted accounts can cause confusion in tweets data. They won’t be included in the mentions user ID column, but will in the text column. This function relies on the mentions user ID column, so it won’t count them
* limitcheck
  + simple function for checking the rate limit of specified tokens
  + input is just a numeric vector specifying the token numbers to check (default is all of them)
  + output: for each token, the function prints out any rate limits that have been used at all in the last 15 minutes
* reformat\_tweets
  + another simple function that reformats the outputs get\_tweets and get\_timeline so that they can be printed into a normal csv

**Second Level Functions**

* recipmentioners
  + Overview: identifies the contacts mentioned by a given user, and how many mentions have been given and received between the user and each of his contacts during the timeframe. Adds all tweets to the archive
  + Dependencies- rtweet, get\_tweets, getmentionees, tibble.
  + Additional arguments:
    - preset.contacts – list of users to limit the data collection to (helpful to speed up sociogram construction)
    - startday and stopday (passed to the get\_tweets function)
  + Normal output: a tibble, with four columns, contact (ID of contact), mentions\_given, mentions\_received, and min (the minimum of the previous two columns). Note that the results are limited to contacts who reciprocated at least once (i.e., min>0).
  + Abnormal outputs:
    - private account, suspended account, incorrect ID: a single NA value
    - 0 tweets (during time period), never tweeted, no mentionees or reciprocated mentions: a tibble with 0 columns or rows.
  + notes on the archive
    - this function requires a pre-existing archive of tweets in the environment (can be 0 rows)
    - it updates and references this archive throughout
  + ID’s and screen names
    - Since screen names are easily changeable, users are always identified by ID’s. However, the function accepts a screen name for the original user
  + Abnormal contacts: this function ignores all contacts who are private, who have since deleted their account, or whose account is suspended.
  + Testing:
    - Confirmed that specifying the timeframes works (checked archive$created\_at)
    - Confirmed that the function handles abnormal accounts correctly
* recipfollowers
  + Overview: this is a function built to identify a user’s reciprocated following relationships, and handle the rate limit as well as tricky kinds of accounts.
  + Dependencies- rtweet, get\_follower2, get\_friends2
  + Normal output- character vector of user IDs, where the length of the vector is the number of reciprocated followers.
  + Abnormal outputs-
    - Mistaken ID- prints “mistaken ID” and returns NA value.
    - Private account- prints “private account found” returns NA value, and appends the user ID to the environment variable privateaccounts (which is created if it doesn’t already exist).
    - 0 friends or 0 followers- prints “0 friends or followers”, and returns a character vector of length 0.
    - Large account (by default more than 300,000 followers or 20,000 friends)- prints “large account found”, returns NA value, and appends the user ID to the environment variable largeaccounts (created if it doesn’t exist).
  + Testing and speed-
    - at 72 tokens, ran fine over 2,740 users (2 hours) without breaking, mean computation of 2.67 seconds, max of 15 seconds.
    - Later, ran overnight (>14,000 users), without breaking.
    - Accuracy testing.
      * Tested on 1,019 most recent followers of jonfavs- correctly identified private accounts (n=71), large accounts (n=3), and accounts with 0 reciprocators (n=128). I spot-checked two other accounts and they were accurate.
        + Miscalculation on one account that’s since been deleted, and I suspect that this was not a problem on my code.
      * Tested on 2,661 oldest followers of RadioLab.
        + Correctly identified all private accounts (n=285), large accounts (n=2), and accounts with 0 reciprocators (n=128).
        + I looked at 1,000 of these accounts that had 200 or fewer reciprocators, and the function was perfectly accurate, though there was one issue that seemed due to the API (may have been a change in following since the first function was run).
        + Looked at 5 accounts with large numbers of reciprocators (1,174-9,674), and found two cases that were one value off, three that were accurate. I’m confident that the discrepancies are from Twitter, not my code (yes, really).

**Third Level Functions**

* recipfollowers\_network
  + This is a higher level function that creates a sociogram representing a 1.5 ego network for the user, with a tie defined as reciprocated following.
  + dependencies: rtweet, get\_followers2, get\_friends2, recipfollowers
  + Normal output is a list that includes:
    - user- the user ID (not screen name)
    - ncontacts- the number of contacts the person has (not always identical to number of contacts in sociogram, see below)
    - funcstart- time when function started
    - funcstop- time when function finished
    - error- “none” if there are no problems
    - sociogram- a sociogram mapping the user’s network, in the form of a matrix. Row and column names are user IDs and are the same. The first user ID belongs to the original user whose ego network this is. Internal data are character vectors. Each cell’s value describes the tie between the users. Options are “Connected”, “Unconnected”, “Self” (on the matrix’s diagonal), and NA.
    - If the original user has more than 500 contacts, 500 of their contacts are randomly sampled and used in the sociogram.
  + Abnormal output-
    - If the original user has a private account, the function returns a single NA for ncontacts and sociogram, and “Private Account” for error
    - If the original user has a large account (over 500,000 followers or over 50,000 friends), the function returns a single NA for ncontacts and sociogram, and “Large Account” for error
    - If the original user has a mistaken ID, the function currently breaks with an error
    - If they have 0 reciprocating friends, it returns intuitive values, no error message.
  + Handling abnormal contacts- the recipfollowers function is run on each contact, using default settings. If this returns an NA (see the recipfollowers documentation for reasons why), the function assesses that contact’s connections using the info from the rest of the contacts. Thus, the only NA values in the sociogram are due to the confluence of two of these abnormal contacts.
  + Testing
    - Test 1: Hit an error after running the function on users for about an hour. Switched tokens and returned this message: “ Error in if (!is\_json(rsp)) { : argument is of length zero”. The error also showed up when running the limitcheck for the first token afterwards. The problem doesn’t appear to be the user (378971584), as I ran it again later and got their data. All tokens (including the one in use at the time) are working fine now.
      * My best guess is that this problem occurred at the first level function, either get\_friends2 or get\_followers2, and the way they’re using tokens.
    - Test 2: Ran all night without a problem.
      * Setting the max at 500 contacts, mean run time per user was 470 seconds.