I want to find out if the Zodiac matters, and if people who are deeply into the zodiac do genuinely treat signs differently. Specifically, I want to see what people on twitter who tweet about men based on their zodiac sign. I want to know if the sign matters at all.

My girlfriend, who I love dearly, has what I think is the correct opinion on the Zodiac: Not binding or predictive in any meaningful way, but a fun frame to reflect on your own behavior and thought patterns. My hunch is that this view is held by most Zodiac enthusiasts, but that like with all things there is a spectrum of belief - there are true believers who hold that your Zodiac sign is a meaningful predictor of personality and behavior.

Having no basis to make this assumption, I assume that true Zodiac believers are mostly women or gay men, and mostly on twitter. This would mean that the true believers we're looking for are also complaining about the men in their life who have wronged them in various ways. Being Zodiac believers, we would also expect them to attach this behavior to their men's various signs. My girlfriend tells me that some signs do anecdotally get more pointed complaints more often (gemini, pisces, capricorn, ares - "water signs get a bad rep"). I would like data to see whether or not this is true.

Here is what I would like to know:

- Does sentiment towards the men of each sign differ?

To do this I will have to:

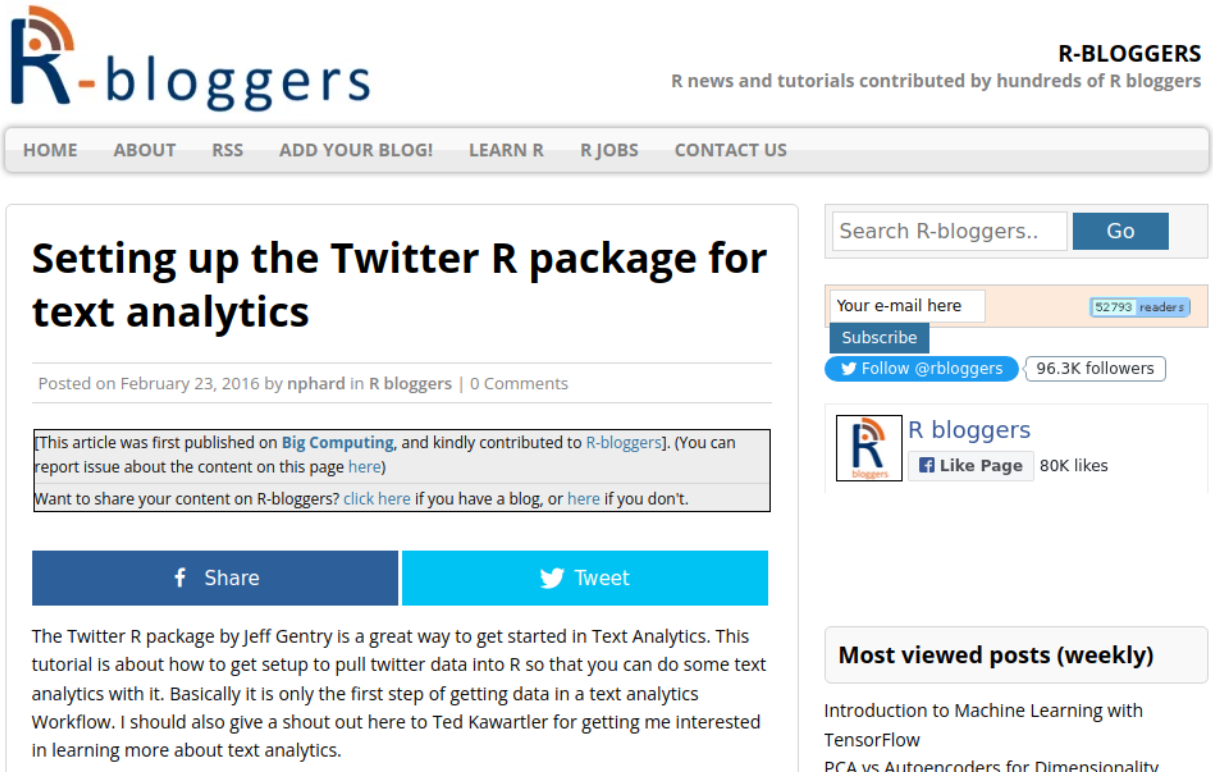
- Collect tweets from Twitter using the search query "[sign] men"
- Filter out words that I am not interested in
- Collect counts of tweets over various time periods
- Conduct a sentiment analysis

I have no idea how to do any of this.

I would like to use the statistical package R. I used it during college, I like RStudio as an IDE, it's all very comfortable and I've only just dipped my toes in Python. First, I need to figure out how I'm going to do this.

STEP 1: Googling how to do any of it.

I will start by searching how to scrape twitter with R.



https://www.r-bloggers.com/2016/02/setting-up-the-twitter-r-package-for-text-analytics/

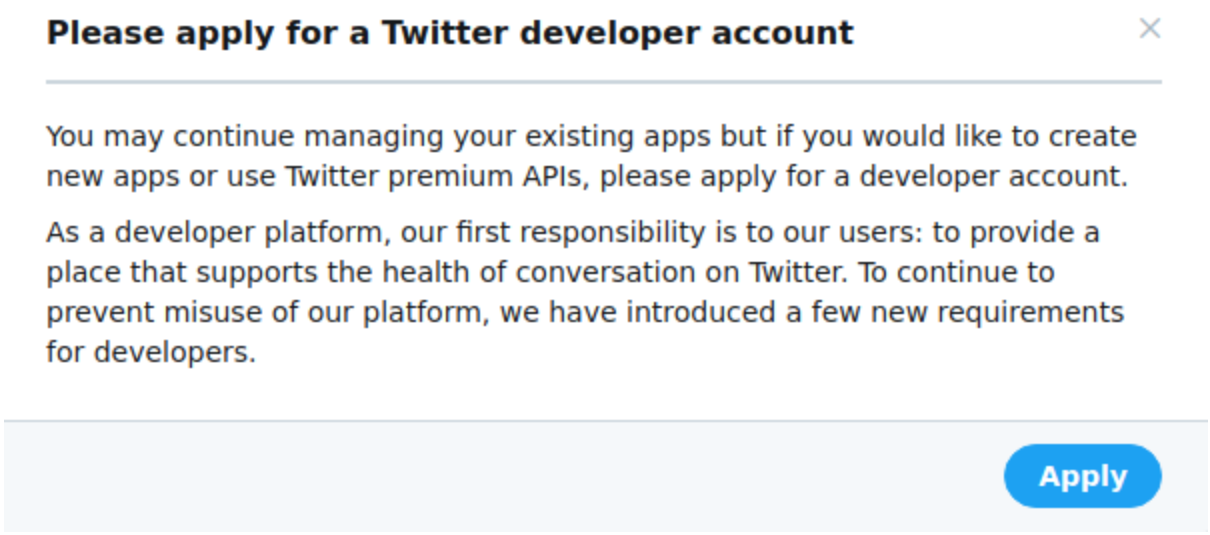
Well, that should work. Thank you Jeff Gentry!

I don't have the twitteR package that the article mentions, so I'll download it real quick. Thankfully twitteR is a CRAN repository, no funny Github stuff.

Now comes the hard part: Remembering the password to the twitter account I made when I was 12. (I was precocious then, and good at lying about my age online.) I have to reset the password. Whatever.

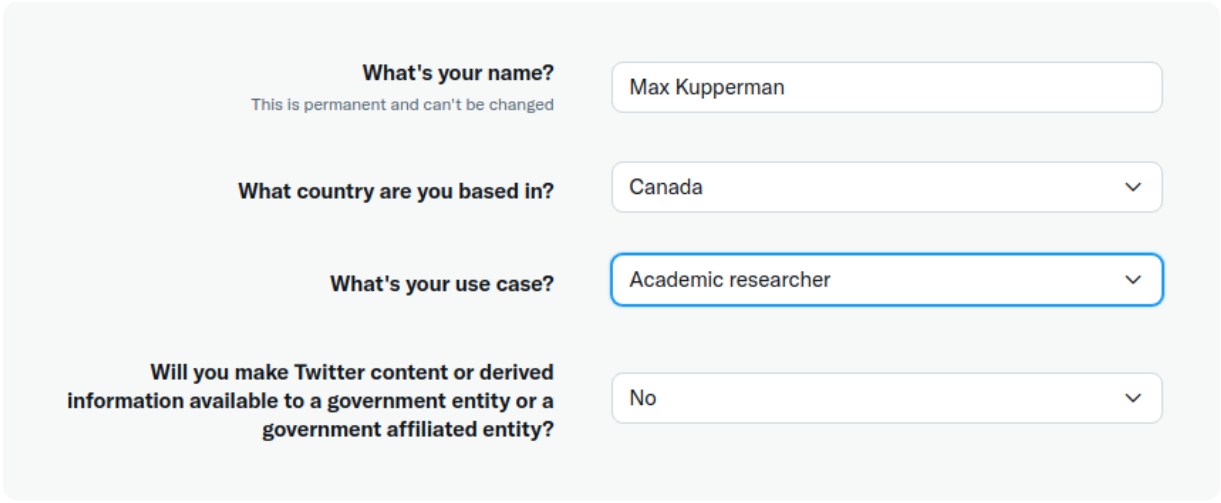
Let's make a twitter app!

STEP 2: Sign up for a twitter dev account



Looks like I'm not the first to do this.

The developer application asks me for my use case. My use case is I'm a moron trying to answer a question about zodiac signs. This is not one of the available options.



I am now an academic researcher.

Wait, this might come back to bite me. I am currently applying for Essential access, which has a cap of giving 500k tweets a month. This should be more than enough for my purposes. I hope it's enough, at least. Maybe it won't be enough.

Let's make a quick estimate to say if this will be enough. There were around 50 tweets with the phrase "gemini men" yesterday, a random Saturday. Let's assume that this is representative of

the number of tweets each [sign] men get each day.. I would like to conduct the same search for all 12 signs for an entire year.

50 tweets/sign/day * 12 signs * 365 day/year = 219k tweets/year

I can probably get around 2 years of tweets per month with this plan. I'd love to do this for all of twitter but I'll stick to 2021 only just to be safe.

I'll also tell twitter that I'm Exploring the API, not doing academic research. Less fun but if I have to apply for a better plan in the future it'll be better to have been honest now.

Alright! Let's send this application and ah fuck I gotta verify my phone number now. God damnit twitter. They do not make this easy.

STEP 3: Create the app

I am now a twitter developer. I have an API Key, API Secret Key, and a Bearer Token. I know vaguely what one of those things are. My career is bright.

My bearer key works. However, there is a problem. Twitter has recently launched their v2 API, but the twitteR scraping guides I have are for their v1 API. I have found the following guide, from Twitter, to work around this:
https://developer.twitter.com/en/docs/tutorials/getting-started-with-r-and-v2-of-the-twitter-api

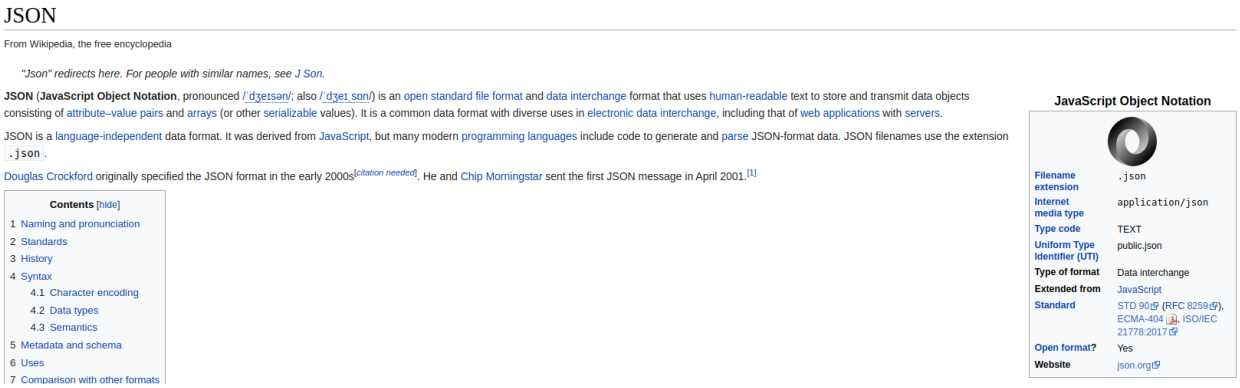
I might have to make the functions from scratch. Who knows.

Let's run the OAuth function!

```
> setup_twitter_oauth(APIkey,APIsecret,AccessToken,AccessSecret)
[1] "Using direct authentication"
Use a local file ('.httr-oauth'), to cache OAuth access credentials between R sessions?

1: Yes
2: No

Selection: 1
Adding .httr-oauth to .gitignore
Error in check_twitter_oauth() : OAuth authentication error:
This most likely means that you have incorrectly called setup_twitter_oauth()'
> setup_twitter_oauth(APIkey,APIsecret,AccessToken,AccessSecret)
[1] "Using direct authentication"
Error in check_twitter_oauth() : OAuth authentication error:
This most likely means that you have incorrectly called setup_twitter_oauth()'
>
```

Shit.

Okay, let's take a look at Twitter's documentation. We'll be working with JSON objects. I don't know what those are, but there's an R package that deals with them. Let's download the package and look up what JSON objects are while they're downloading.



JSON

From Wikipedia, the free encyclopedia

"Json" redirects here. For people with similar names, see J Son.

JSON (**JavaScript Object Notation**, pronounced /ˈdʒeɪsən/; also /ˈdʒeɪˌsɒn/) is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values). It is a common data format with diverse uses in electronic data interchange, including that of web applications with servers.

JSON is a language-independent data format. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data. JSON filenames use the extension `.json`.

Douglas Crockford originally specified the JSON format in the early 2000s[*citation needed*]. He and Chip Morningstar sent the first JSON message in April 2001.[1]

Contents [hide]

1 Naming and pronunciation
2 Standards
3 History
4 Syntax
4.1 Character encoding
4.2 Data types
4.3 Semantics
5 Metadata and schema
6 Uses
7 Comparison with other formats

JavaScript Object Notation

Filename extension	`.json`
Internet media type	`application/json`
Type code	TEXT
Uniform Type Identifier (UTI)	public.json
Type of format	Data interchange
Extended from	JavaScript
Standard	STD 90 (RFC 8259), ECMA-404, ISO/IEC 21778:2017
Open format?	Yes
Website	json.org

It's a fun and fresh way to transfer data between two computers. Sick.

STEP 4: Find out how I'm going to get the tweets

The package has been downloaded, and I've loaded the library. I don't need to call or download the other two packages, httr and dplyr, because they're tidyverse packages and I habitually load the tidyverse on open.

twitteR is definitely depreciated by the update, which sucks. The searchTwitter() function doesn't work. I am going to have to build a function. Jeff Gentry, I assume, has better things to do than update this package.

I am now looking at one of Twitter's sample functions in their github.

https://github.com/twitterdev/Twitter-API-v2-sample-code/blob/main/Tweet-Lookup/get_tweets_with_bearer_token.r

21 lines (13 sloc) | 425 Bytes Raw Blame

```r
require(httr)

bearer_token <- ""

headers <- c(`Authorization` = sprintf('Bearer %s', bearer_token))

params <- list(`tweet.fields` = 'created_at')

ids <- '1293593516040269825'

url_ids <-
  sprintf('https://api.twitter.com/2/tweets?ids=%s', ids)

response <-
  httr::GET(url = url_ids,
            httr::add_headers(.headers = headers),
            query = params)


obj <- httr::content(response, as = "text")
print(obj)
```

Let's try to understand what's going on here.

- httr is a package that allows R to process HTTP files.
- The bearer token is my bearer token. So far so good.
- They're making me make the object headers to add by bearer token to the URL later on with add_headers(). That's important, it'll let me use the API.
- ids is an object for a tweet id. Each tweet on twitter is numbered sequentially in order. This tweet id is Twitter's official twitter account announcing v2 of the twitter API.
- url_ids joins the id into the twitter API with sprintf(), a proxy for C. Whoever wrote this is probably a C programmer first. The more native R way would be str_glue() or something.
- response calls the tweet from the API and calls it using httr::GET()
- We put the tweet into a text object

This is fine, but I don't want to collect one tweet, I want many tweets whose IDs I don't know.

Let's try another one:
https://github.com/twitterdev/Twitter-API-v2-sample-code/blob/main/Full-Archive-Search/full-archive-search.r

26 lines (20 sloc) | 550 Bytes Raw Blame

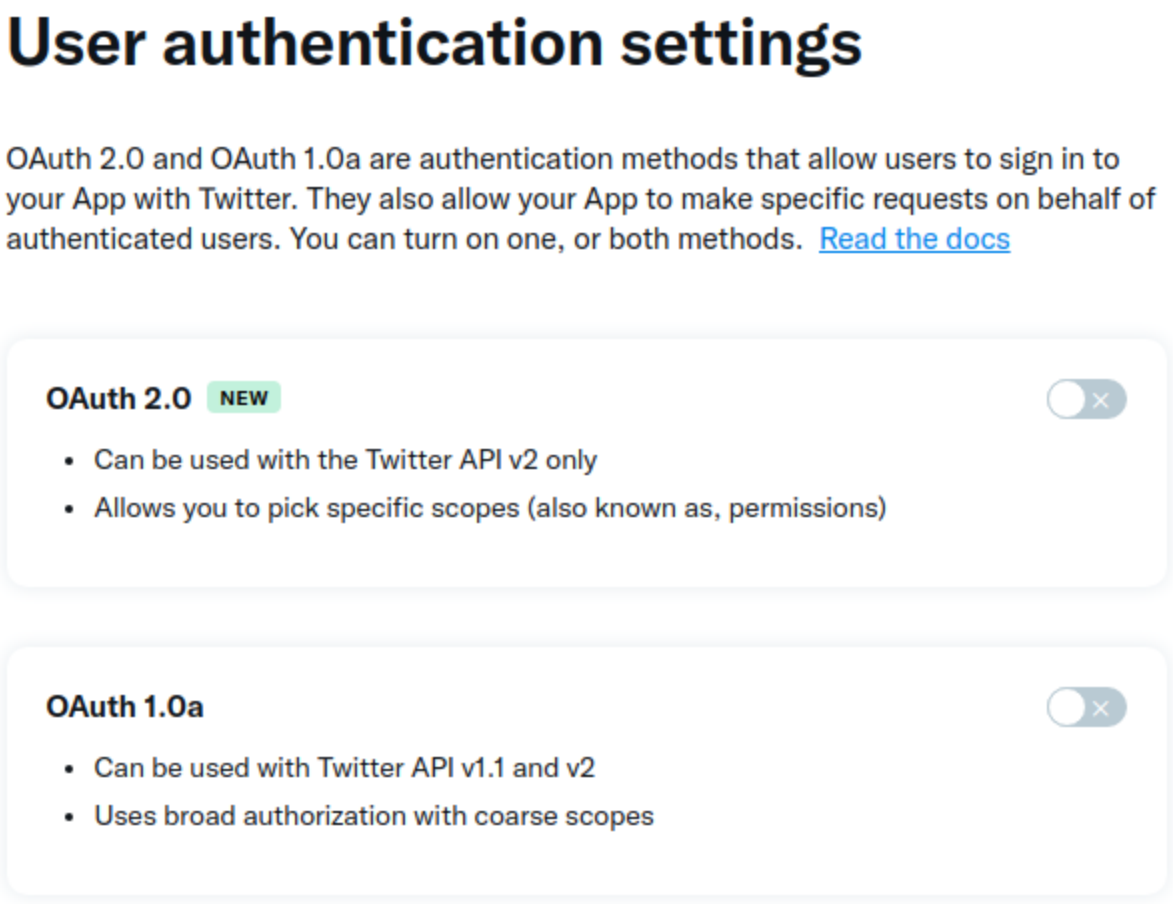
```r
library(httr)

# Replace the bearer token below with
bearer_token = ""

headers = c(
  `Authorization` = sprintf('Bearer %s', bearer_token)
)

params = list(
  `query` = 'from:twitterdev lang:en',
  `max_results` = '10',
  `tweet.fields` = 'created_at,lang,context_annotations'
)

response <- httr::GET(url = 'https://api.twitter.com/2/tweets/search/all', httr::add_headers(.headers=headers), query = params)

fas_body <-
  content(
    response,
    as = 'parsed',
    type = 'application/json',
    simplifyDataFrame = TRUE
  )

View(fas_body$data)
```

What does this do?

- Loads httr, same as before
- Inputs bearer token, same as before
- The search parameters are different! It's restricting the search to being from @twitterdev, in English, 10 tweets, and includes the fields created_at, language, and whatever context_annotations is.
- Response generates the actual search
- fas_body must be a cleaning function. Don't know quite what it does yet. If I had to guess it makes the JSON data readable to us.
- View() to see results. Straightforward.

The problem is the sample code doesn't give me a way to get results based on a search term. Very frustrating!

Let's look at the resources we have before us. There's Postman, a service twitter seems very well connected to that could give me the search terms I'm looking for, but I don't want to sign up for another account if I don't have to. Maybe I can use the OAuth function from earlier?



I am a moron. How did I not see this earlier? I'm going to use 1.0 OAuth, since I only intend to make the one call.

I have enabled OAuth 1.0 using my linkedin profile as my website. One day I will have the money to be more professional than this, but here I stand.

Let's try that twitteR call from earlier.

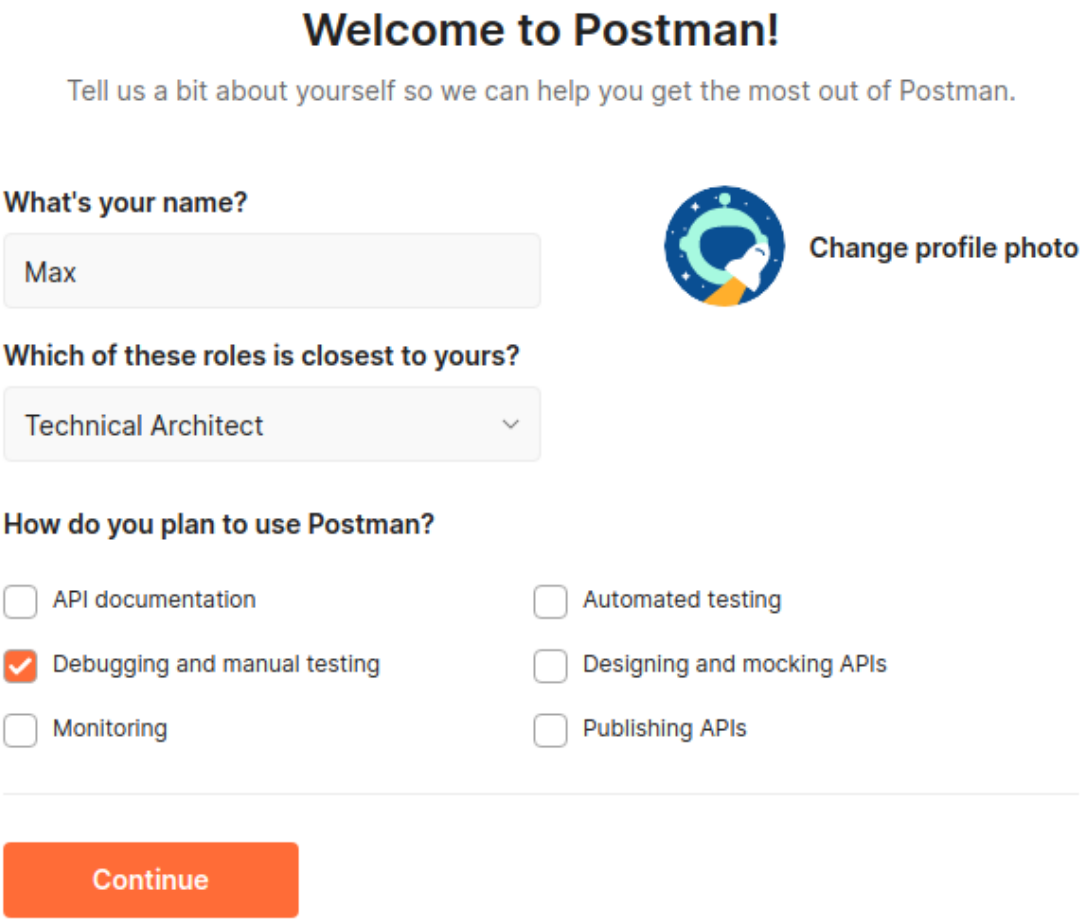
```
> sampleSearch <- searchTwitter("chris boucher", n=2)
Error in twInterfaceObj$doAPICall(cmd, params, "GET", ...) :
  Forbidden (HTTP 403).
> setup_twitter_oauth(APIkey,APIsecret,AccessToken,AccessSecret)
[1] "Using direct authentication"
Error in check_twitter_oauth() : OAuth authentication error:
This most likely means that you have incorrectly called setup_twitter_oauth()'
```

Fuck. Back to the drawing board? Let me google that error code first.

https://github.com/geoffjentry/twitteR/issues/137

Calling additional libraries doesn't help. rtweet is mentioned as an alternative library in the comments, but its search functions rely on twitteR.

Time to try signing up for Postman.



I'm a technical architect now, baby!

I've found Twitter's postman profile, and I'm coming up short. What I want is a way to search the text of tweets, but they are not giving me an easy way to do this.

Maybe I should start by running their code and seeing if it works.

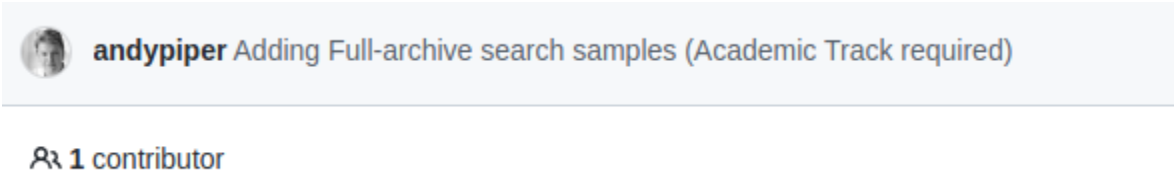
fas_body	list [7]	List of length 7
client_id	character [1]	'23050948'
detail	character [1]	'When authenticating requests to the Twitter API v2 endpoints, you must use keys ...
registration_url	character [1]	'https://developer.twitter.com/en/docs/projects/overview'
title	character [1]	'Client Forbidden'
required_enrollment	character [1]	'Standard Basic'
reason	character [1]	'client-not-enrolled'
type	character [1]	'https://api.twitter.com/2/problems/client-forbidden'

Well, that might explain a thing or two. Let me check OAuth again.

That didn't work either. Let me try that first block of code, the one that I ruled out


```
[1] "{\"data\":[{\"created_at\":\"2020-08-12T17:01:42.000Z\",\"id\":\"1293593516040269825\",\"text\":\"It's finally here! \\uD83E\\uDD41 Say hell
o to the new #TwitterAPI.\\n\\nWe're rebuilding the Twitter API v2 from the ground up to better serve our developer community. And today's launch
 is only the beginning.\\n\\nhttps://t.co/32VrwpGaJw https://t.co/KaFSbjWUA8\"}]}"
```


That did what it was supposed to do, and I didn't have any errors. Huh. I must be restricted from doing a larger search.



Academic Track Required. Yep. Yep yep yep yep yep. Shit.

Andy Piper, bless them, has added a third block of code that I should test: https://github.com/twitterdev/Twitter-API-v2-sample-code/blob/main/Recent-Search/recent-search.r

It defaults to running only on @twitterdev, but I tested the code on @ebruenig, and ....

```r
require(httr)

headers = c(
  `Authorization` = sprintf('Bearer %s', bearer_token)
)

params = list(
  `query` = 'from:ebruening',
  `max_results` = '10',
  `tweet.fields` = 'created_at,lang,conversation_id'
)


response <- httr::GET(url = 'https://api.twitter.com/2/tweets/search/recent', httr::add_headers(.headers=headers), query = params)


recent_search_body <-
  content(
    response,
    as = 'parsed',
    type = 'application/json',
    simplifyDataFrame = TRUE
  )

View(recent_search_body$data)
```

it works!

I've made this into a function, searchTwitterTextAndTimestamp(), for ease of use. The name reflects that I don't really care about anything in the tweet except for the text of the tweet and the timestamp. After tweaking it, I think we're ready to roll.

I ran the function to generate the most recent 10 tweets with the phrase "scorpio men". The results are beautiful:

	created_at	id	text
1	2022-01-10T00:03:53.000Z	1480329513162706947	🛒 Sleek sellouts! 🤓. Order Scorpio Pink Tourmaline...
2	2022-01-09T23:58:48.000Z	1480328231085494276	Scorpio And Pisces Men 🙌
3	2022-01-09T23:58:02.000Z	1480328037837144064	Scorpio men are such stalkers
4	2022-01-09T23:45:36.000Z	1480324910086070274	scorpio men do be crying a lot. it's so much.
5	2022-01-09T23:40:50.000Z	1480323710577876993	scorpio and aries men https://t.co/LC9x9l74sW
6	2022-01-09T23:26:44.000Z	1480320164604395522	@miwkluvinpanda Mal kolchi mea scorpio men😤
7	2022-01-09T23:24:37.000Z	1480319630006796291	Leo shower u with love and attention, this is true. I ...
8	2022-01-09T23:22:44.000Z	1480319157023477765	Scorpio men are not for the weak 😭😭😭
9	2022-01-09T23:15:01.000Z	1480317213525655561	Capricorn WOMEN,cancer WOMEN,LEO WOMEN e S...
10	2022-01-09T23:10:25.000Z	1480316058208768007	Y'all lied to me about Scorpio men. The dick is bom...

I would like to congratulate tweet #10 on having recently received some bomb dick. Way to go! I am making a note here to add lang:en to the search filters. I have no interest in reading non-English Zodiac tweets. Maybe another time.

STEP 5: Build a machine

It is time to conduct a sample sentiment analysis on these 10 tweets. For this I am consulting this guide:
https://utstat.toronto.edu/~nathan/teaching/sta4002/Class1/scrapingtwitterinR-NT.html

It is clear from the outset that I know nothing about sentiment analysis. It is time to learn a bit about it.

There are a bunch of different sentiment analysis standards, four of which the package tidytext supports. I think I'm going to use AFINN, which assigns numerical sentiment values to each word. What I want to do is:

- Collate a bunch of tweets
- Strip the individual words from each
- Get their AFINN values
- Sum the values up

We'll be able to thus assign a numerical value to the sentiment of man-sign on twitter. If I make a list with each word in each tweet from a large enough sample, I'll also be able to tell what the most common words are.

After some trial and error, I've written some code that does exactly this:

```
#Tweet Stripper
require(tidyverse)
require(tidytext)

afinn <- get_sentiments("afinn")


text_amalgamator <- function(data){
scorp_words <- data$text
scorp <- str_replace_all(scorp_words, "[^[:alnum:]]", " ")
scorp <- paste(scorp, collapse = " ")
scorp <- str_split(scorp," ") %>% unlist() %>% as.data.frame()
colnames(scorp) <- c("word")
bad <- c("RT", " ", "t", "co", "", "https","and","are","the","with","be")
scorp <- scorp %>% filter(., !(word %in% bad))
scorp$word <- tolower(scorp$word)
scorp_count <- scorp %>% count(word)
scorp_count

scorp_count <- merge.data.frame(scorp_count,afinn)
scorp_count <- mutate(scorp_count, score = n * value)
return(scorp_count) }
```

I'm very excited about this. It spits out exactly what I want it to: a list of words, their frequency, the AFINN sentiment value per word, and a net score of frequency times sentiment. Interestingly the sentiment for the test sample is 3, pretty neutral. Wonder if that will hold.

Let's get some results!

STEP 6: Get meaningful results

I have run into a minor issue, which is that twitter is only letting me pull 100 tweets at a time. I think that'll be enough. I hope it will be, anyway. So much for a 219k tweet dataset!

I've decided to call this machine the sentiment cannon. It sounds nice to me. I've wrapped the two functions above into another function:

```
60 twitter_sentiment <- function(query){
61   tweets <- searchTwitterTextAndTimestamp(query,100)
62   sentis <- text_amalgamator(tweets)
63   return(sentis)
64 }
```

I'm making the query 100 each time for best results. Maybe I'll get better API access in the future and be able to scrape more at once. Who knows.

I've generated a couple tables as tests and made the following observations:

1. AFINN's list does seem to capture the sentiment of the tweets accurately!
2. The most common sentiment-valued words for any sign are "yes" and "no". Might be worth including in an analysis as a separate measurable.

I would like to get the following information from each of these tables:

1. Query name
2. Number of "sentiment words" (words assigned a sentiment score)
3. Number of times Yes appears
4. Number of times No appears
5. Overall sentiment score
6. List of most common words

First, I need a list of the zodiac signs, and of each sign + "men".

```
1 zodiac_signs <- tibble(c("aries","taurus","gemini","cancer","leo","virgo","libra","scorpio","capricorn","aquarius","pisces"))
2 colnames(zodiac_signs) <- "signs"
3
4 zodiac_men_list <- mutate(zodiac_signs, sign_men = str_glue("{signs} men"))
5
```

	signs	sign_men
1	aries	aries men
2	taurus	taurus men
3	gemini	gemini men
4	cancer	cancer men
5	leo	leo men
6	virgo	virgo men
7	libra	libra men
8	scorpio	scorpio men
9	capricorn	capricorn men
10	aquarius	aquarius men
11	pisces	pisces men

Simple enough.

I'm going to write a function that takes one of these inputs and spits out the 6 data points we're looking for above.

```
79 conduct_sentiment <- function(query){
80   gem_tibble <- tibble(query)
81   colnames(gem_tibble) <- "query"
82   gem_sent <- twitter_sentiment(query)
83   gem_tibble$word_count <- sum(gem_sent$n)
84   gem_tibble$yes_count <- count_tester(gem_sent, "yes")
85   gem_tibble$no_count <- count_tester(gem_sent,"no")
86   gem_tibble$score <- sum(gem_sent$score)
87   gem_sent <- arrange(gem_sent,desc(n))
88   gem_tibble$common_words <- filter(gem_sent, n > 1)$word %>% paste(.,collapse = ", ")
89   return(gem_tibble)
90 }
```

Actually, on second thought, I'm going to want to write a function that does all of this 12 times for each of the sign men. Let me write this as a function that takes the query name and sentiment table instead - that'll stop me from pulling too many tweets from twitter.

```
analyse_sentiment <- function(query,sent_table){
  gem_tibble <- tibble(query)
  colnames(gem_tibble) <- "query"
  gem_tibble$word_count <- sum(sent_table$n)
  gem_tibble$yes_count <- count_tester(sent_table, "yes")
  gem_tibble$no_count <- count_tester(sent_table,"no")
  gem_tibble$score <- sum(sent_table$score)
  sent_table <- arrange(sent_table,desc(n))
  gem_tibble$common_words <- filter(sent_table, n > 1)$word %>% paste(.,collapse = ", ")
  return(gem_tibble)
}
```

Sick. Fun fact: I stopped writing as I was doing this a little while back. I'm pretending that this is what I'm currently doing even though it's not. I'm lying to you!

I did a first run of this where I just spit out the sentiment values we were looking for above, but after showing the results to my girlfriend she asked if I could make a version that collected the tweets and sentiment word tables as well. She didn't ask for those things explicitly, but that's what she wanted. I came up with this:

```
many_sentiments <- function(query_list){
  results_tibble <- tibble(query=character(),word_count=integer(),
                           yes_count=integer(),no_count=integer(),
                           score=integer(),common_words=character())
  tweet_table_list <- list(results_tibble)
  names(tweet_table_list) <- c(query_list[1])
  sentiment_table_list <- list(results_tibble)
  names(sentiment_table_list) <- c(query_list[1])
  for (item in query_list){
    tweets <- searchTwitterTextAndTimestamp(item,100)
    sentis <- text_amalgamator(tweets)
    new_row <- analyse_sentiment(item,sentis)
    results_tibble <- rbind(results_tibble,new_row)
    tweet_table_list[[item]] <- tweets
    sentiment_table_list[[item]] <- sentis
  }
  master_list <- list(results_tibble,tweet_table_list,sentiment_table_list)
  names(master_list) <- c("sentiments","tweets","word lists")
  return(master_list)
}
```

I really like this function. I made some late additions that I think make it a lot more professional. This is why I like it:

- It spits out all the data you need in one nice list
- It names the items in the list accordingly, so if you're looking at it in View() it's easy to navigate and clear what the data is
- The names are automatic based on the search query
- It only asks twitter to generate tweets once per query

- It can take multiple and single search queries
- It does essentially everything I've been working on up to this point in a single function.

RESULTS

Guys, I fucked it. After trying the function on the list of men a couple times, I've realized I can't really get good data.

What are the issues?

First, I can only draw 100 tweets per query at a time. This is not enough, as it turns out. It only reflects about 1-2 hours of tweets for the most popular searches.

Second, AFINN doesn't care much about context. This is particularly a problem for Cancer Men, where most of the tweets are about men with cancer, the group of diseases. Other individual words (like, love, fuck, dick, etc) that AFINN notes as positive or negative may not have the same connotation in context.

Third, I have to manually enter in filtered words for any given query or group of queries and I don't want to do that shit.

Here is a sample output from after I took some filters off so that I could run a similar test for taylor swift albums:

	query	word_count	yes_count	no_count	score	common_words
1	aries men	141	12	9	45	like, love, yes, no, lol, lowest, terrible, attracted, ba...
2	taurus men	162	13	6	37	like, yes, love, best, cancer, no, apologizing, bad, lol...
3	gemini men	141	11	7	4	yes, like, best, cancer, no, hate, avoid, lol, shit, god,...
4	cancer men	249	4	4	-111	cancer, die, risk, like, want, lmao, myth, no, yes, ba...
5	leo men	135	3	7	-12	weakness, love, no, like, bitches, fuck, gift, great, h...
6	virgo men	151	4	8	13	like, lol, no, hate, love, best, gift, great, hell, yes, ca...
7	libra men	130	11	7	48	yes, love, like, no, active, hell, lol, worst, best, canc...
8	scorpio men	147	3	3	-17	love, like, fucking, fight, nasty, stop, best, crazy, fav...
9	sagittarius men	143	12	7	52	amazing, bad, yes, no, like, cancer, worst, chances, ...
10	capricorn men	50	0	2	-20	weakness, like, lol, ass, death, good, love, no, obses...
11	aquarius men	161	13	8	16	like, yes, love, no, hate, shit, cancer, chances, fuck, ...
12	pisces men	145	3	5	-20	hate, like, love, cancer, gift, great, no, shit, dick, fuc...

If I could get enough tweets to run this a couple times, filter out the problem words, and still have enough data to draw meaningful conclusions from I would. As it stands, I can't do that, so nothing meaningful here.

With that said, I've created a pretty good machine. I may come back, fine tune this, and make a shiny app out of it. Who knows.

WHAT HAVE WE LEARNED?

1. The twitter API is easy as shit broski
2. Twitter is also the devil. I'm not paying them for more tweets.
3. I am a god of R programming
4. paste(collapse = "") is how you combine all items of a list together
5. Sentiment analyses are very difficult and I'm not doing it right
6. I like doing this
7. Maybe I should've required the tweets to be english after all