Chapter 9

Using the Facebook and Twitter APIs

Authors' note: This tutorial combines two tutorials referenced in *The Data Journalist.*

Skills you will learn: This tutorial covers basic use of the Facebook and Twitter APIs.

Facebook

Facebook has changed its Graph API so that information from user timelines as described *in The Data Journalist* is no longer available. Only information from public Facebook pages, that is pages that you can follow, can be obtained via the API. Due to the changing terms of Facebook's API, you should consult with the documentation on the Graph API to determine how you might use it for a journalistic project. You can find the documentation, including how to set up an application and access the service, at https://developers.facebook.com/docs/graph-api

It is important to note that Facebook's terms of service prohibit scraping or other automated data collection, except in extremely narrow circumstances, and only with explicit permission. Please see Facebook's page on the matter at https://www.facebook.com/apps/site_scraping_tos_terms.php

We would not recommend, therefore, attempting to circumvent Facebook's API restrictions through Scraping. You could become subject to legal action, even criminal prosecution, depending on the jurisdiction where you live. We would recommend consulting with qualified legal counsel if you are unsure of the legalities of any project you wish to undertake.

Twitter

Twitter has always been a more public platform than Facebook. While Facebook's model is based on private connections between users, which have to be accepted by both parties, Twitter's default model is more akin to broadcasting in that anyone can follow a Twitter account and receive its

tweets, unless the account holder specifically locks it down. Twitter also has restrictions on use of its APIs, however, and you should be mindful of them and consult with legal counsel if you feel you might be exceeding what Twitter allows. You can read the developer terms at https://dev.twitter.com/overview/terms/agreement-and-policy Specifically, Twitter prohibits any use that would tend to duplicate the functionality of Twitter itself, and has tight restrictions on applications that post content to Twitter. For example, writing a bot that retweets Twitter content is subject to tight restrictions. When collecting data, you are also expected to delete tweets if the original user has deleted them and the change of status has been transmitted through the API. Also, the terms place tight restrictions on providing data you collect to any third party. The terms in effect as of the writing of this tutorial limited such provision, via non-automated downloads, spreadsheets, or PDFs, to either only user IDs or tweet ids, or of up to 50,000 public tweets per user per day. Use for your own analysis should be fine, however. The best approach is to read the current terms and policies in effect, and govern yourself accordingly.

Generally speaking, within these rules, obtaining tweets for research purposes is permitted, though there are rate limits that put a cap on how many tweets you can obtain without paying extra to one of Twitter's partners, such as Gnip.

Twitter has several APIs, including the Search API that is used to search for recent tweets based on search parameters (similar to a search you would conduct via the Twitter application TweetDeck, etc), the REST API, which can be used to obtain tweets based on such things as the unique ID of a tweet or a Twitter user name, and the Streaming API, which is used to collect tweets as they happen.

There is extensive documentation available on Twitter's developer site at https://dev.twitter.com/docs.

We are going to focus on one use case, obtaining tweets from the REST API for one user. Other use cases of the REST API will be similar.

Twitter uses an authorization method known as OAuth and before you can use the API, you need to create an application, and you do that by logging in using your Twitter ID and password at https://apps.twitter.com/. You will be guided through the setup of your application, but at the end of the process you need to have obtained four special pieces of unique identifying information, in two pairs. The first pair consists of the consumer key and consumer secret for your application. Think of these as a username and password that is provided by your application when it contacts the API. The second pair is an access token and access secret. These provide an additional level of security and can be regenerated as necessary. All are long alphanumeric strings.

In no circumstances should you ever share any of these identifying strings, because with them someone would have access to the API in your name. Keep the information in a safe place as you will need it whenever you write a script to access the API. In the example that follows, we'll show you where to enter the information.

Using the API to obtain tweets

We'll walk through the process of writing a simple script to pull down a user's most recent tweets, using the REST API. The script will access the most recent 3,200 tweets for any account whose tweets are public, and save them into a CSV file. The limit of 3,200 is built into the API and applies to anyone making a request for a user's tweets.

Once you have your consumer key and secret and access token and token and secret, you can begin to write your script.

For the purposes of this exercise, we will use Tweepy, which is one of a number of third party modules that you can use to access the API. You can find the documentation for Tweepy here: https://media.readthedocs.org/pdf/tweepy/latest/tweepy.pdf

Because it is a third-party module, you will need to install it with pip. At a prompt in the Mac Terminal or Windows, enter this command:

```
pip install tweepy
```

You will also need the unicodecsv module, which we used in earlier scripts in chapter 9 and various tutorials. If you haven't yet installed it, you can use pip to install it as well:

```
pip install unicodecsv
```

Once Tweepy and unicodecsv are installed, you can begin writing your script. Our goal will be to connect to the Twitter API, retrieve the tweets we want, then extract the information into a csv file that we can import into a database or spreadsheet application.

The first thing we need to do is import the modules we will use. There are four:

```
import tweepy
import time
import json
import unicodecsv
```

We have seen all of these before, except of course, Tweepy.

Next, we need to create variables to hold our consumer key and secret, and our access token and secret. Of course, you will paste in your own keys in place of what you see here.

```
ckey = 'your consumer key goes here'
csecret = 'your consumer secret goes here'
atoken = 'your access token goes here'
asecret = 'your access secret goes here'
```

In the next section of code, we create a new instance of a tweepy OAuthHandler, calling it auth, and giving it the ckey and csecret variables as parameters. We then use the new instance's set_access_token method to set the access token, and provide the atoken and asecret variables as the parameters.

```
auth = tweepy.OAuthHandler(ckey,csecret)
auth.set_access_token(atoken,asecret)
```

Those steps complete, we're ready to fetch some tweets.

In the next line, we create a new instance of the API class of tweepy, calling it api, and give it auth as its parameter. The API class allows you to use any of the REST API methods that are provided by Twitter. This object now has all the abilities it needs to make a request to the Twitter API, and get back a response. So we type in:

```
api = tweepy.API(auth)
```

Next, we will write code to accept some user input—in this case the user can type in a valid Twitter username—,to create a text file to which we will write, and to create a unicodecsv writer object. These will all be familiar to you from previous examples. Make sure to enter a valid user name (sometimes called a Twitter handle), or the script will fail. We haven't added any error handling, but you could write a try/except clause to intercept an error and add a prompt to try again. Note that you could simply assign a single, arbitrary user id to the variable userid, but you would have to change the code every time you wanted to get Tweets for a new user.

```
userid = raw_input("Enter a valid Twitter screen name (@name)")
outFile = open('path to the outfile' + userid +
'Tweets.csv','w')
writer = unicodecsv.writer(outFile,dialect='excel')
```

In the next set of code we set a variable x to the value 0. This is so we can set up a conditional statement to run one of two variations of a call to the API.

We then begin a while loop, and because we use the syntax while True, it will run until we give an explicit break statement or we terminate the process manually.

Within the while loop, we have written an if statement. It tests to see if x contains the value 0. The first time the loop runs, it will contain that value as we just set it two lines above, so the action associated with the if statement will run. The code creates a list called statusList. statusList is populated with tweets, which we obtain using the user_timeline method. We provide two parameters, one is the userid, which is contained in the variable that was populated by user input, and the second is the count of the number of tweets to fetch, in this case 200. This is the maximum number you can fetch at once using this method. We might as well go for the max.

```
x=0


while True:
    if x == 0:
        statusList = api.user_timeline(id=userid,count=200)
```

The next part of the script contains an else statement that will fire if x is not equal to 0, and because we iterate x by 1 near the bottom of the script, this means the else statement will run every subsequent time the while loop executes. Here we make the same request, but add one more parameter, the max_id parameter. This is the unique id of the very last tweet fetched the last time the loop ran, and when this parameter is provided, the API will return only tweets with IDs prior to that tweet. Since the API returns tweets in reverse chronological order, each time the loop runs it will fetch the 200 previous tweets until the 3,200 tweet limit is reached, or if there are fewer than 3,200 available, until all have been fetched.

```
    else:


        statusList =
api.user_timeline(id=userid,count=200,max_id=lastTweetID)
```

The result of the above lines is that statusList will be populated with a Python list of tweet objects. We now need to loop through each of the tweet objects, and extract the detail we want. A for loop is the ideal tool for that task. We will write a loop to iterate through each tweet in the list, and grab the data we want. Inside the for loop, we first create an empty list to hold our extracted data.

```
    for theStatus in statusList:
         statusData = []
```

Continuing inside the code block, we extract from the iterator variable theStatus the element called '_json', which is actually a Python dictionary.

The next two lines use the dictionary get function to grab the tweet id (id_str) and the tweet text from the dictionary and append them into the statusData list. When extracting the text, we are stripping off white space and replacing any new line and carriage return characters with the empty string. Finally, we write the statusData list to our output file, using the writer's writerow method, and print the statusData list to the screen.

```
         statusDict = thestatus._json
         statusData.append(statusDict.get('id_str'))

statusData.append(statusDict.get('text').strip().replace('\n','')
)
           .replace('\r',''))   #Note: this line is part of the
previous line
         writer.writerow(statusData)
         print statusData
```

The next line checks the length of statusList. If it equals 1, which it does if the last available tweet has been provided, the while loop terminates with the break statement.

```
    if len(statusList) == 1:
         break
```

If the last tweet has not been reached, the script continues to run.

The next line uses Python list slicing syntax to grab the last tweet contained in statusList and put it into a variable called lastTweet. This is the earliest tweet in the particular group of tweets. Next, the json element is extracted, and the unique tweet id (idstr) is extracted using the dictionary get function, and saved in the variable lastTweetID. It will be used the next time the while loop runs, to

populate the max_id parameter and cause Tweepy to collect the previous 200 tweets. In this way, we grab 200 tweets at a time until all are collected. Finally, we call the time.sleep() function to delay two seconds before running the while loop again, and we iterate x by 1.

```
lastTweet = statusList[-1]

    lastDict = lastTweet._json
    lastTweetID = lastDict.get('id_str')

    time.sleep(2)
    x += 1
```

That's all it takes to grab 3,200 tweets of any Twitter user whose account is public.

The completed script looks like this:

```
import tweepy
import time
import json
import unicodecsv

ckey = 'your consumer key goes here'
csecret = 'your consumer secret goes here'
atoken = 'your access token goes here'
asecret = 'your access secret goes here'

auth = tweepy.OAuthHandler(ckey,csecret)
auth.set_access_token(atoken,asecret)
api = tweepy.API(auth)
```

```
userid = raw_input("Enter a valid Twitter screen name (@name)")
outFile = open('path to the outfile' + userid +
'Tweets.csv','w')
writer = unicodecsv.writer(outFile,dialect='excel')



x=0


while True:
    if x == 0:


        statusList = api.user_timeline(id=userid,count=200)


    else:


         statusList =
api.user_timeline(id=userid,count=200,max_id=lastTweetID)


    for thestatus in statusList:


         statusData = []
         statusDict = thestatus._json
         statusData.append(statusDict.get('id_str'))

statusData.append(statusDict.get('text').strip().replace('\n',''
).replace('\r',''))
         writer.writerow(statusData)
         print statusData
```

```
if len(statusList) == 1:
     break


lastTweet = statusList[-1]


lastDict = lastTweet._json
lastTweetID = lastDict.get('id_str')


time.sleep(2)
x += 1
```

This is a simple and useful example of how you can put the Twitter API to work. This script could be modified through the use of a Python list of Twitter user names and a for loop to iterate through the list, to grab tweets from many accounts. In that case, you would obviously extract the user name as well, into your CSV. Our simple example only extracts the tweet id and the text of the tweet. There are many available data elements in the JSON tweet object, including the exact time and date of the tweet. The complete list of fields is available here: https://dev.twitter.com/overview/api/tweets

Twitter rate limits

It is important to keep in mind that Twitter imposes rate limits on the use of its APIs. This is the maximum number of API calls that you can make in a specific period, usually 15 minutes. The limit for the method we used in the script above is a generous 900, meaning you can make 900 calls to the API using the `user_timeline` method within a 15 minute window, or one every second. Make sure you include a delay of at least a second between API calls, when using this method. Other timeline methods have much tighter limits, as low as one call per minute, and you need to incorporate longer delays in your script to ensure you don't exceed the limits. For more information on these rate limits, go to https://dev.twitter.com/rest/public/rate-limits

Other use cases for the REST API will follow the same kind of pattern as seen in this simple script, though you may use a different timeline method. Consult the API documentation, and the Tweepy documentation for more detailed information on the REST api, as well as the search and streaming APIs.