Mapping Social Media

Introduction

Most of us are on social media be it Facebook, Twitter, Instagram or WhatsApp. We all celebrate different events that also take place in our day to day lives, be it a birthday, an internationally recognised day or even political posts appear on social media.

In this tutorial we are not going to do anything politically related. Instead, we are going to use the power of **Python** to find out *where* people were taking part in the just ended *#GlobalSurveyorsDay* celebrations.

So I have personally selected one social media platform which I know is where we can actually get to find most of the data that we are looking for only for this exercise.

What Do You Need?

In order to be able to follow along through this step-by-step guide, you are going to need the following:

- Twitter Developer Account

We are going to need this account to be able to use the *Twitter API* which we will integrate with our Python code to give us what we need.

- An ArcGIS Organizational Account

Since this is all about *Where*, we defintely need that Geographical Information Systems platform and what better place to represent this than in an Online environment where anybody can view and leave a comment.

- Jupyter Notebooks (or any code editor)

Of course you definitely need a place to write your code and execute. I prefer these Jupyter Notebook but if you have ArcGIS Pro, it comes embedded with these too so you can just open them in there. Don't like notebooks? Well then I guess you can use code editors like Visual Studio Code for this exercise then.

Let's Get to Work

So I also left a few things above because those do not have anything to do with our programming which we are about to indulge into.

We are going to need the following Python libraries to execute our task well enough:

- tweepy

An easy-to-use Python library for accessing the Twitter API.

- dotenv

For keeping our secret credential safe in a place where you can't see them.

- pandas

For handling our data in a manner that we want.

- re

For using Regular expressions for some conversions and detection.

Installation of Packages

Just like any other foreign packages, these need to be installed and we are just going to do that here.

1. Installing Tweepy

Its simple. Just run the command below in the terminal.

```
pip install tweepy
```

2. Installing dotenv

Well I am not going to steal the show on this one because I actually had to google in order for me to see how its done. You can read the official post from Medium which explains how its done.

They are several methods to do so but I prefered using the **python-dotenv** methods where you just run:

`pip install python-dotenv`

So in the end my `.env` file looked like this:

```
.env
twitter_consumer_key=<my_consumer_key>
twitter_consumer_secret=<my_consumer_secret>
twitter_access_key=<my_access_key>
twitter_access_secret=<my_access_secret>
```

I know you probably wondering where I got the four values I need to place in that `.env` file from. Remember when I said we need a Twitter Developer account? Yes, above.

Register for a Twitter Dev Account.

Okay so here it is:

- first you need a normal user twitter account. You can go and create one if you did not have any.
- then you need to register for a developer account. You can apply on their developers site
- you will need to read the guidelines and accept the terms and conditions too
- finally, wait for verification. It normally takes a day or two to get verified and you ae ready to go.
- in case they reject your application, you need to read the policies and figure out which one your application violated and re-apply.

Assuming you now have that Dev Account, lets get to focus more on this tutorial.

Login

As usual, like in any other previous tutorials that I have released, we need to login to get started.

In [1]:

```
from arcgis.gis import GIS
import tweepy

gis = GIS("https://arcgis.com", "kumbirai_matingo")
```

```
Enter password: ••••••••
```

Import Libraries

We are going to import our libraries here for usage.

- os
- dotenv

Do not forget to `load_dotenv` otherwise those secret keeys will not be detected.

After importing, we need to create variables for our secrets so that we can reference them later on by just calling the name of the variable.

To call the value from our `.env` file, we need to call in the `os.getenv()` method and pass in the variable name we assigned in the `.env` file as an argument.

In [2]:

```
# Twitter Dev Credentials
import os
from dotenv import load_dotenv # add this line

load_dotenv() # add this line

consumer_key = os.getenv('twitter_consumer_key')
consumer_secret = os.getenv('twitter_consumer_secret')
```

```
access_key = os.getenv('twitter_access_key')
access_secret = os.getenv('twitter_access_secret')
```

In this step, let us create our authentication system. We call in the `OAuthHandler()` method, pass in the *secrets* as parameters and we also do the same with the `set_access_token()` but now we will be using the keys to set the access tokens.

Then we tell the **API** that we have authenticated with these parameters can you verify and give me access to Twitter data.

If you keys and secrets are correct, you will not see any errors below.

In [3]:

```
# Twitter authentication
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)

# Creating an API object
api = tweepy.API(auth)
```

Global Surveyors Day Tweets

The code below might be a bit confusing but here is an explanation.

- we created a variable for every expression that we will need later on
- tell tweepy to call the `Cursor()` method which helps the **api** to perform a search using the keywords passed in the `q` expression.
- we use the extended `tweet_mode` is we need to get all the data associated with a single tweet.
- the `pages()` basically handles pagination in case we have soo much tweets we need to extract. For this tutorial I will not be paginating anything.

In [4]:

```
# Search GlobalSurveyorsDay tweets
gsd = tweepy.Cursor(api.search, q="GlobalSurveyorsDay", tweet_mode='extended').pages()
```

Let's play around with Loops

- we create an empty dictionary in which we are going to store each and every tweet
- create a **for** loop which goes through each and every result.
- create another loop (this time nested) which will store the paginated tweets. In our case we do not have any paginated tweets.
- append each tweet after the other and store this information in JSON format

In [5]:

```
all_tweets = []
for page in gsd:
    for tweet in page:
        all_tweets.append(tweet._json)
```

Let's just check to see how many tweets in total do we actually have.

In [6]:

```
len(all_tweets)
```

Out[6]:

```
14
```

Remember the **regular expression** we talked about? Import it here as below

- create a function which is going to identify URL's in every `tweet_text`
- the `tweet text` stores the data which the user who sent out the tweet actually said.

- so the expression is going to find anything that starts with an *https* within the tweet.
- the identified URLs are placed in an HTML anchor tag

If you are not familiar with HTMl an anchor tag stores this clickable links like this one which go somewhere on the web. The HTML notation is as below:

```
<a href="https://somewebsite.com/">A clickable statement</a>
```

Its actually a nice way for our viewers to see the original source of the tweets.

- lastly we truncuate the length of the tweet to avoid long letters. If the user wants to view more that's what the link will be for.

In [7]:

```
import re
def handle_tweet_text(tweet_text):

    for url in re.findall(r'(https?://\S+)', tweet_text):
        tweet_text = tweet_text.replace(url, f'<a href="{url}">{url}</a>')

    if len(tweet_text) >= 356:
        tweet_text = f'{tweet_text[:350]} . . .'

    return tweet_text
```

Now that we have refined our `tweet_text`, let's create a new list which this time is going to be structured. We are going to be storing only the items we want to represent for our project.

We have chosen the following items:

- id of the tweet
- date when the tweet was created
- text
- hashtags
- mentioned users
- url
- location (which is basically the reason why we are here)
- retweet count (how many people retweeted that particular tweet)
- the screen name of the user
- their profile image
- favourite count (the number of users who liked that tweet)

This is just a list of what we want, at the end of this tutorial, I am going to include a JSON which contains everything that you can extract out of twitter. *Just hang on*

We call the **for** loop again to get the data that we need and store it in a list.

In some cases we use `try` and `except` because its not certain if we are going to find anything and we wouldn't want to run into any errors if nothing exists. So the *except* function helps us in placing a default if nothing is found.

The we *append* after each and every loop into our list.

In [8]:

```
all_tweets_refined = []
for tweet in all_tweets:
    temp = {}
    temp['id'] = tweet['id']
    #fetch created time
    temp['created_at'] = tweet['created_at']
    #fetch text
    temp['text'] = handle_tweet_text(tweet['full_text'])
    #fetch hashtags for the tweet
    try:
        hashtags_list = tweet['entities']['hashtags']
        hashtags = [hashtag['text'] for hashtag in hashtags_list]
        if len(hashtags_list)>1:
            temp['hashtags'] = ",".join(hashtags)
        else:
            temp['hashtags'] = hashtags_list[0]['text']
```

```
    except:
        temp['hashtags'] = ''
    #fetch user mentions
    mentions_list = [mention['screen_name'] for mention in tweet['entities']['user_mentions']]
    temp['mentions'] = ",".join(mentions_list)
    #fetch url for the tweet
    try:
        temp['url'] = tweet['entities']['urls'][0]['url']
    except:
        temp['url'] = ''
    #Fetch location (if available) for tweet
    try:
        temp['location'] = tweet['user']['location']
    except:
        temp['location'] = None
    #fetch number of times tweet has been retweeted
    temp['retweet_count'] = tweet['retweet_count']
    #who posted the tweet?
    temp['screen_name'] = tweet['user']['screen_name']
    #The profile picture of the source account
    temp['profile_image_url'] = tweet['user']['profile_image_url']
    # The number of people who like the tweet
    temp['favorite_count'] = tweet['favorite_count']
    all_tweets_refined.append(temp)
```

Let's create a *DataFrame*.

Remember **Pandas**? We are going to use it here to create that *DataFrame* and convert it into a *CSV* file.

In [9]:

```
import pandas as pd
```

We are going to store the DataFrame in a variable named *tweets_df* and then we call the `DataFrame()` method and pass in our list as a parameter.

- included below is how you can convert different date and time zones to suit your needs.

In the `tz_convert()` where I placed **None** you can replace that with the timezone that you need.

For example: I can use UTC and my code for that line will look like so;

```
tweets_df['created_at'] = tweets_df['created_at'].map(lambda t: t.tz_convert('UTC'))
```

You can check outline for different timezones and their representational parameters.

The I use the `head()` function to display the first *5* rows in my dataframe.

In [10]:

```
tweets_df = pd.DataFrame(all_tweets_refined)
tweets_df['created_at'] = pd.to_datetime(tweets_df['created_at'])
tweets_df['created_at'] = tweets_df['created_at'].map(lambda t: t.tz_convert(None))
tweets_df.head(5)
```

Out[10]:

	id	created_at	text	hashtags	
0	1376839696991461379	2021-03-30 10:12:18	RT @LPSminneapolis: .@NSPSINC @NCEES @GetKidsi...	NationalSurveyorsWeek,GlobalSurveyorsDay	LPSminneapolis,NSPSINC,NCE
1	1376768912042299393	2021-03-30 05:31:02	Long days, bad weather, over time, complex pro...	GlobalSurveyorsDay	
2	1376308722788564992	2021-03-28 23:02:24	RT @NCEES: Today is #GlobalSurveyorsDay. Learn...	GlobalSurveyorsDay	
3	1375760237941166084	2021-03-27	We have chosen to dwell much on the	GlobalSurveyorsDay,GlobalSurveyorsDay,Python,A...	

	id	created_at	text	hashtags
4	1375557398312591363	2021-03-26 21:16:55	Long days, bad weather, over time, complex pro...	GlobalSurveyorsDay,PureSurvyeing

Convert DataFrame to CSV file

- call the `to_csv()` function and assign a path where we would like to store our csv and **DO NOT** forget to give it a name.

I have a folder named **data** in this projects directory, so that's where I chose to store my csv file.

In [11]:

```
# save the dataframe as a CSV file
gsd_tweets = tweets_df.to_csv('data\global_surveyors_day.csv')
gsd_tweets
```

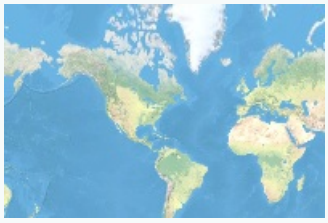
Publish to ArcGIS Online

I will not get into detail about this step since I have touched on this one a few tutorials back. You can refer to this publication.pdf) in case you have forgotten what this process of publication means or if you need explanation on why and what.

In [12]:

```
surveyors_day_tweets = gis.content.add({}, 'data/global_surveyors_day.csv')
surveyors_day_tweets
```

Out[12]:



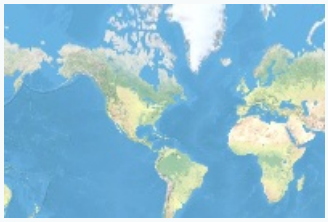
global_surveyors_day

CSV by kumbirai_matingo
Last Modified: April 01, 2021
0 comments, 0 views

In [13]:

```
# created a hosted layer
surveyors_day_tweets.publish()
```

Out[13]:



global_surveyors_day

Table Layer by kumbirai_matingo
Last Modified: April 01, 2021
0 comments, 0 views

Now that we have published our data and it's ready for mapping, let me deliver my promise to you.

That JSON

Earlier on I had mentioed about other things you can extract from twitter. This code below is going to help us see what exactly can we get from twitter.

I am going to extract some information from my twitter account

This time we do the same thing like we did with the *search* query above but instead, we change a few things.

- we tell the API to get from a `user_timeline`
- define a screename (in this case that's my twitter account username)

The `.items()` function tells the API how many items we want to return. Leaving it blank will extract all information from my timeline and if I have soo many tweets, you might exhaust and go beyong the API's limit. Its always wise to specify how many items you want.

In [14]:

```
my_tweets = tweepy.Cursor(api.user_timeline, screename="surveyor_jr", tweet_mode='extended').items(
10)
```

Creating a list. Just like we did before

In [15]:

```
retrieved_tweets = []
for tweet in my_tweets:
    retrieved_tweets.append(tweet._json)
```

Let's filter everything and review the contents of a **single** tweet.

Check it out for yourself.

retrieved_tweets[0]

So that's it for now. That is how you can get to extract data from twitter and use the data for various use cases such as analysis or anything you might think of.

You can also watch the live video based on this tutorial straight from our **Youtube** channel. Be sure to like and subscribe. Leave a comment also in case you need more explanation.

If you didn't face any errors along the way and managed to get the data onto the ArcGIS Organizational Platform then;

CONGRATULATIONS

You've managed to complete your step in **#DataMining** with GIS

For anyone having trouble or fails to understand this tutorial, I am reachable via LinkedIn. Just send me a direct message and I will be sure to respond to any questions relating to the tutorials that you might have.

About Author

- 3rd Year BSc Hons in Surveying & Geomatics
- Interested in GIS for Health and Land Administration, Spatial Data Science and Programming
- You can download my Resume online too. Just click here