**Analysing Streaming Tweets with Python and PostgreSQL**

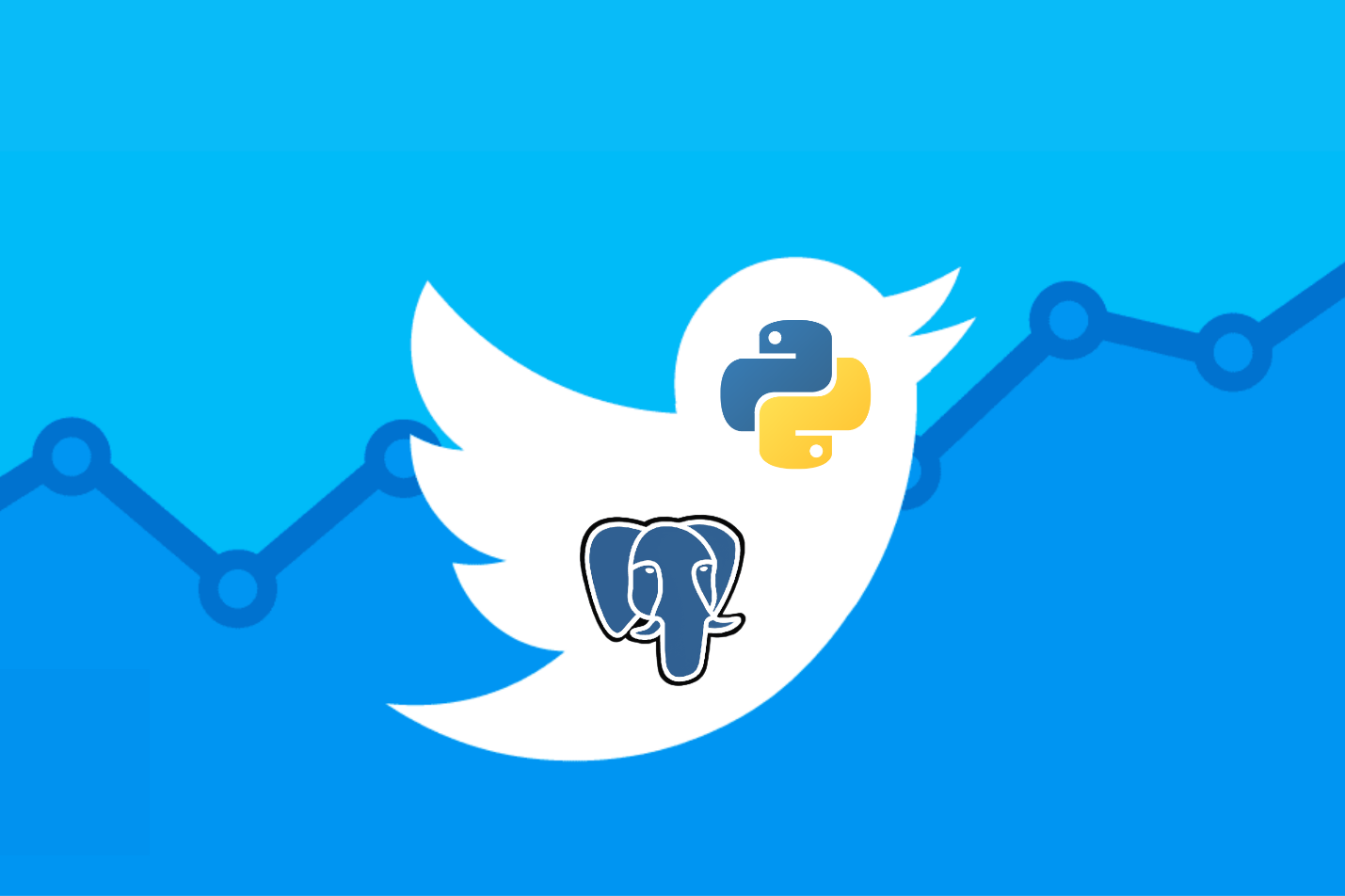
[**ANIRUDDHA BHANDARI**](https://www.analyticsvidhya.com/blog/author/aniruddha/)**, AUGUST 17, 2020**

[**LOGIN TO BOOKMARK THIS ARTICLE**](https://id.analyticsvidhya.com/auth/login/?next=https://www.analyticsvidhya.com/blog/2020/08/analysing-streaming-tweets-with-python-and-postgresql/)**[Click to Download PDF](https://www.analyticsvidhya.com/back-channel/download-pdf.php?pid=69683)**

ArticleVideos

**Introduction**

We are aware of the massive amounts of data being produced each day. This humungous data has lots of insights and hidden trends stored within it that need to be skillfully extracted by data scientists. However, collecting this massive amount of unstructured data is a challenging task in itself. But don’t worry, data engineers are here to your rescue!



[Data engineers](https://www.analyticsvidhya.com/blog/2018/11/data-engineer-comprehensive-list-resources-get-started/) gather the unstructured or semi-structured data from various sources and store them in a structured format in databases. They prepare this data for model building required by the data scientists.  They are the intermediary between unstructured and structured data and play a crucial role in a real-world data science project.

In this article, we will work with a small project of analysing streaming Tweets. From extracting data from the source and storing it into a database to processing that data and preparing it for analysis tasks, we will try to understand the workflow of a data engineer’s task. And to bring closure to this project, we will analyse this data towards the end and see if we can come up with some valuable insights.

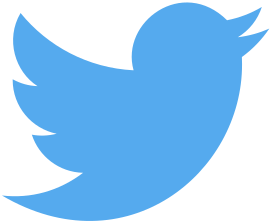
**Table of Contents**

1. What we want to achieve here
2. Installing relevant Libraries
3. PostgreSQL Database
4. Creating Database in PostgreSQL
5. Creating Twitter Developer Account
6. Authenticating the Twitter API
7. Streaming Tweets using Tweepy
8. Streaming Tweets on Keywords
9. Streaming Tweets in Real-Time using StreamListener
10. Creating Tables in PostgreSQL
11. Connecting to the Database
12. Streaming Tweets into Database
13. Analyzing Streamed Tweets

**What we want to achieve here**

Today, the production of data is at a lightning pace. Every domain from stock markets and social media platforms to IoT devices are producing thousands of rows of data in a matter of seconds. But often this data is of immense importance in real-time. For example, analyzing real-time weather data is important for forecasters to predict how the weather will change over time.

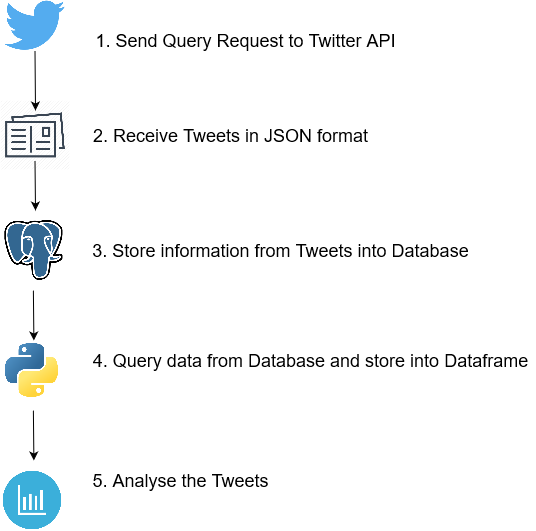
This streamed data can either be analysed in real-time or can be stored and analysed later. Either way, it proves to be immensely important for analysis and deriving insights. Therefore, it becomes imperative for every data scientist to understand how to handle real-time or streaming data. And what better streaming data than Twitter which produces millions of Tweets every hour!



Twitter is a treasure trove of data. A plethora of tweets is sent out every second. Just imagine how much information you can extract from this. If you can analyse streaming tweets, you can understand the feelings of the people about events happening in the present. Businesses can leverage this information about how the public is reacting to their product. Investors can determine how a stock is going to perform and so much more.

Here we plan to stream tweets on Covid-19. Determine people’s sentiment about a disease that has disrupted our lives in an unprecedented way. Also determine what hashtags are being used by people, which keywords are being used, and more.

We will accomplish this by first establishing a connection with the Twitter API. Then we will extract relevant information from tweets and store them into a database. Once we have done that, we can query this database and analyse the streaming tweets. Below is a visual representation of how we are going to achieve this.



Now, before delving into the programming part, let’s first install the relevant libraries for this project.

**Installing relevant Libraries**

We need to install three of the following things for this project:

* PostgreSQL database to store the Tweets  
  Installing PostgreSQL is pretty easy and straightforward. Head over to the [download page](https://www.postgresql.org/download/) and download the application according to your machine’s operating system. Once you have done that, PostgreSQL will be downloaded on your machine along with the pgAdmin tool which is a development platform for PostgreSQL.
* Tweepy library to stream the Tweets  
  [Tweepy](http://docs.tweepy.org/en/latest/) is a great library to access the Twitter API. Using Tweepy you can either stream tweets from your timeline, a specific user’s timeline, or simply search for the specific keyword! All using simple and elegant in-built functions.It is pretty straightforward to install Tweepy. You can either use pip to install Tweepy:

pip install tweepy

Or use Git to clone the repository from [GitHub](https://www.analyticsvidhya.com/blog/2020/05/git-github-essential-guide-beginners/?utm_source=blog&utm_medium=analyse_streaming_tweets).

git clone<https://github.com/tweepy/tweepy.git>

cd tweepy

pip install .

psycopg libraries to access the PostgreSQL database from Python.  
You can install **psycopg** using the pip command.

!pip install psycopg

Alright, now we have downloaded all the relevant libraries for the project. Now let’s understand the properties of PostgreSQL and why we are using it here.

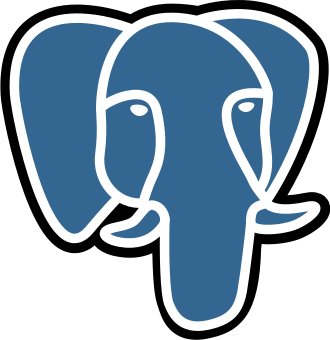
**PostgreSQL Database**

Postgresql is a relational database management system based on SQL. It is free and open-source and works well on all Operating Systems. Further, it has a proven reputation for safely and robustly storing your data and supports both SQL (relational) and JSON(non-relational) based querying.

Postgresql has properties like asynchronous replication, multi-version concurrency control, and many more. Additionally, Postgresql is also highly extensible. You can create your own data types, index types, and more. Besides supporting the primitive data types like boolean, characters, integers, etc., Postgresql also has some additional data types related to geometry like a box, line, polygon, etc.

Postgresql is widely used as a back-end database with many dynamic websites and web applications and provides support for many languages like Python, Java, C++, and more.

The popularity is well documented. In the recent Stack Overflow survey of 2019, Postgresql was the second most popular database amongst the professional developers, ahead of giants like SQL Server, Oracle, and others. And is popularly used by many technology behemoths like Apple, Spotify, Instagram, and more.



So why are we using Postgresql for our purpose here? Being a relational database, we can use Postgresql to define the table structure and the datatypes we want for each of the attributes. It will also ensure that redundant data is not saved multiple times in the database, which can save us a lot of space.

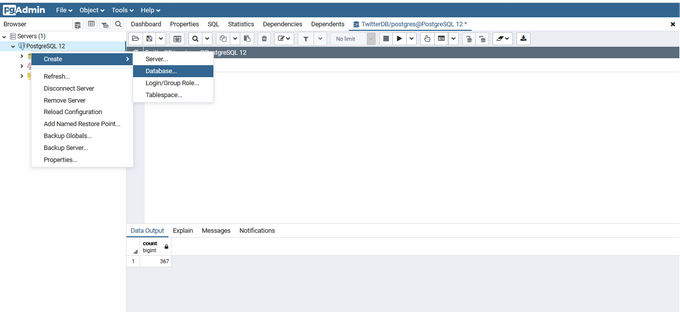
We can utilize the properties of the primary key and foreign key to create multiple related tables in the database for different purposes. Storing data in this structured manner will allow us to query the database according to our requirements.

So let’s install this awesome database!

**Creating Database in PostgreSQL**

Let’s create a database in Postgresql where we will create multiple tables to store data that we receive from Twitter API. To create a database in Postgresql, we are going to use the pgAdmin tool that got installed when we installed Postgresql.

Create a new database and give a name to it. I am going to name our database TwitterDB.

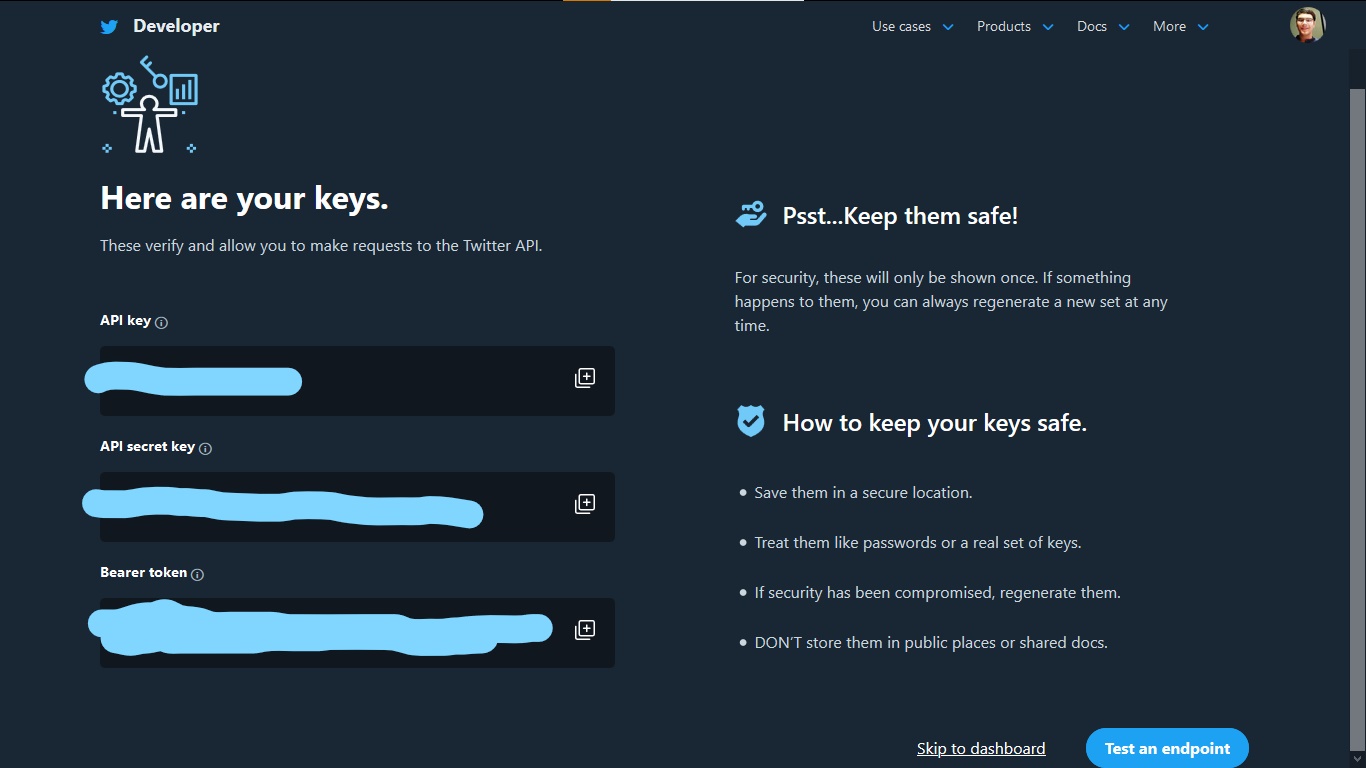


That’s it! You created a database for storing your tweets. Since Postgresql supports Python programming language, we will leverage that capability later on in the article to create tables inside this database. For now, let’s set up our Twitter API!

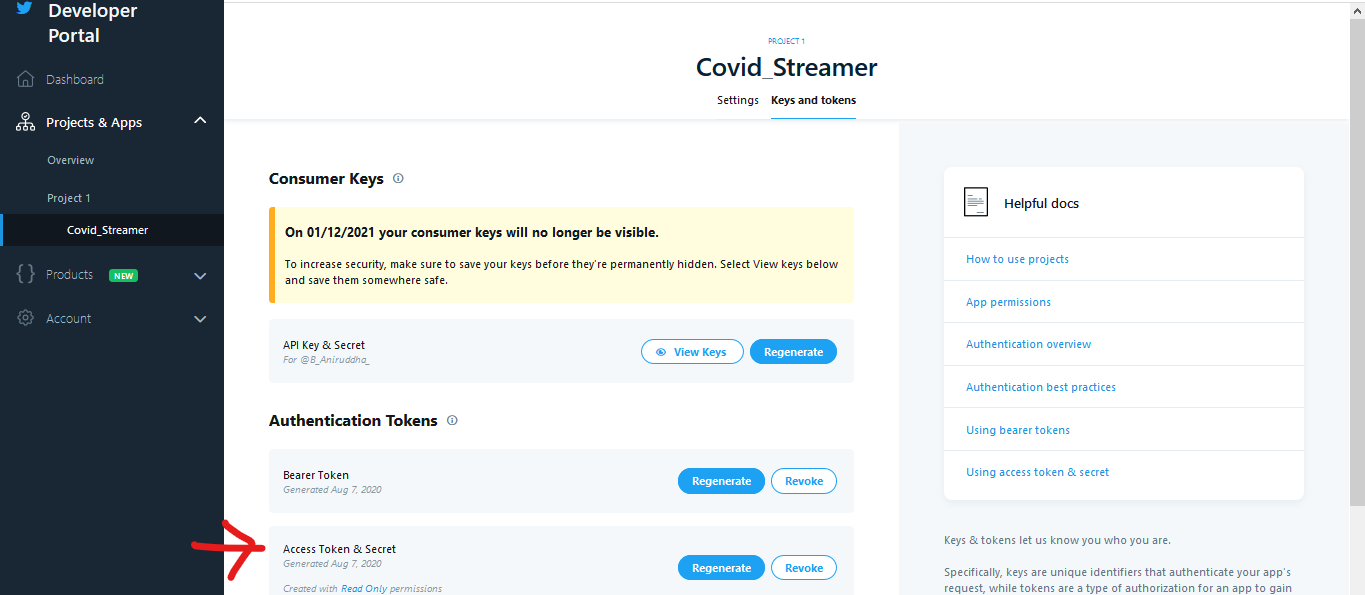
**Creating Twitter Developer Account**

To stream data from Twitter, you will need to head over to the [Twitter developer’s website](https://developer.twitter.com/en/apply-for-access) and register your app to get access to the Twitter APIs. The steps are pretty straightforward. However, you need to provide a good explanation as to what you are going to do with the data you receive from Twitter. Hence, this might take a while, but you should be able to get this done pretty easily.

Once you have registered your app, you will be provided with an API access key and API secret key. Store these keys in a safe place because they will only be generated once and you will need them, again and again, to connect to the API.



Once you have done this, head over to the dashboard and give a name to your app. Then access the Access token and generate Access Token Secret key. You will need these two keys as well to connect to the API.



Alright! Now that you have your API access keys, it’s time to stream those tweets!

**Authenticating the Twitter API**

Authenticating Twitter API with Tweepy is a piece of cake.

Firstly we need to create an OAuthHandler instance that handles the authentication. To do this, you need to pass the API key, API secret key, and Access token that you saved after setting up your Twitter app.

Besides this, you will also need the Access Token Secret key which you can generate by logging into your developer account and generating one for the Twitter app that you created. Once you have done that, you can finally create the API object by passing in the authentication information.

|  |  |
| --- | --- |
|  | # Twitter API authentication |
|  |  |
|  | import tweepy |
|  |  |
|  | api\_key = # api\_key |
|  | api\_secret\_key = # api\_secret\_key |
|  | access\_token = # access\_token |
|  | access\_token\_secret = # access\_token\_secret |
|  |  |
|  | # authorize the API Key |
|  | authentication = tweepy.OAuthHandler(api\_key, api\_secret\_key) |
|  |  |
|  | # authorization to user's access token and access token secret |
|  | authentication.set\_access\_token(access\_token, access\_token\_secret) |
|  |  |
|  | # call the api |
|  | api = tweepy.API(authentication) |

[**view raw**](https://gist.github.com/aniruddha27/49362ce9ee342ffaa835d71d90199416/raw/1152d67aef00f02661d40ccf63c264c7b3e8ef15/authenticate_twitter_api.py)[**authenticate\_twitter\_api.py**](https://gist.github.com/aniruddha27/49362ce9ee342ffaa835d71d90199416#file-authenticate_twitter_api-py) hosted with  by [**GitHub**](https://github.com/)

Now you can finally stream your data using Tweepy!

**Streaming Tweets using Tweepy**

There are a couple of ways to access the data from Twitter. Let’s start with the first one, accessing tweets from our own timeline.

To do that you need to use the API object’s[home\_timeline()](http://docs.tweepy.org/en/v3.5.0/api.html#API.home_timeline) function. By default, it returns 20 most recent tweets from your timeline but you can change that using the **count** parameter.

|  |  |
| --- | --- |
|  | # Streaming tweets from home timeline |
|  |  |
|  | public\_tweet = api.home\_timeline(count=5) |
|  |  |
|  | for tweet in public\_tweet: |
|  | print("-->",tweet.text) |

[**view raw**](https://gist.github.com/aniruddha27/4db098163238931d5df91bf1c6f4c794/raw/6b734437bd64af87baef75a9f544d989a22af383/streaming_tweets_home_timeline.py)[**streaming\_tweets\_home\_timeline.py**](https://gist.github.com/aniruddha27/4db098163238931d5df91bf1c6f4c794#file-streaming_tweets_home_timeline-py) hosted with  by [**GitHub**](https://github.com/)

You can even use Tweepy to access tweets from another user’s timeline. For this, you have to use the API object’s[user\_timeline()](http://docs.tweepy.org/en/v3.5.0/api.html#API.user_timeline) function.

|  |  |
| --- | --- |
|  | # Streaming tweets from user timeline |
|  |  |
|  | user = "AnalyticsVidhya" |
|  | public\_tweet = api.user\_timeline(id=user,count=5) |
|  |  |
|  | for tweet in public\_tweet: |
|  | print("-->",tweet.text) |

[**view raw**](https://gist.github.com/aniruddha27/6f45d36d74ba53c7c5e92ec43ea0695b/raw/af9f007dedc2ad92059b3e8240d6d6e193bcfe54/streaming_tweets_user_timeline.py)[**streaming\_tweets\_user\_timeline.py**](https://gist.github.com/aniruddha27/6f45d36d74ba53c7c5e92ec43ea0695b#file-streaming_tweets_user_timeline-py) hosted with  by [**GitHub**](https://github.com/)

Alright now that you have got your hands dirty by streaming tweets, it is important to understand the various data fields that are being retrieved along with the tweet text. Let’s dive deep and unravel the streaming data.

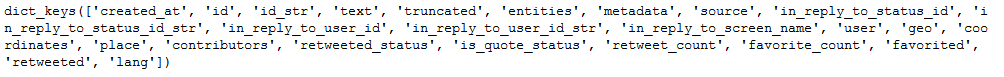
**Streaming Tweets on Keywords**

The Twitter API returns the data in JSON format. So far we had just accessed the text from the tweet using tweet.text. But there is more to what is being returned by Twitter API. To understand that, let’s stream tweets on a specific topic.

Since we are looking to analyse streaming tweets around Covid-19, let’s stream those tweets using the API object’s[search()](http://docs.tweepy.org/en/v3.5.0/api.html#API.search) method which uses the [Twitter Search API](https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets).

|  |  |
| --- | --- |
|  | # Retrieve tweets |
|  | result = api.search(['covid','Covid-19','COVID-19'], lang='en', count=10) |
|  |  |
|  | # JSON keys |
|  | pprint(result[0].\_json.keys()) |

[**view raw**](https://gist.github.com/aniruddha27/6472ca2e77d4ecf36d09a0f58afacab1/raw/2e374d1efa3a87680aac28f1c2b6b9437aa1400f/twitter_search_api.py)[**twitter\_search\_api.py**](https://gist.github.com/aniruddha27/6472ca2e77d4ecf36d09a0f58afacab1#file-twitter_search_api-py) hosted with  by [**GitHub**](https://github.com/)



There is a wealth of information being returned here. As you must have noticed, the API returns the Tweet and related information in a JSON format. Each Tweet JSON contains information about the tweet, the user, entities like hashtags and mentions, and so on. There can be over 150 attributes associated with the Tweet! Let’s try to understand some of the important fields in the Tweet JSON that are returned by the API.

* id – This integer attribute is a unique identifier for the Tweet.
* text – This is the Tweet text that we are looking for. It can contain up to 140 characters.
* user – Contains details about the author of the tweet as a nested field. Information like user id, user creation date, screen name, etc. can all be found in this field.
* entities – This field contains information such as hashtags, mentions, links, etc. These are also present as nested fields.
* truncated – In 2017, Twitter extended the Tweet JSON to include longer messages of up to 280 character length. This field indicated whether the Tweet is shorter than 140 characters or not. If the tweet is longer than 140 characters, it returns True otherwise it returns False.
* extended\_tweet – In case the tweet contains more than 140 characters, the tweet, and the entity information will be truncated in the root-level fields. To access the complete data, you need to access the fields nested within *extended\_tweet*

The following command will let you access the Tweet JSON-

pprint(result[0].user.\_json)

To access hashtags from code, use the following code.

pprint(result[4].entities['hashtags'])

Tweet hashtags

*You can read about the Tweet JSON in detail*[*here*](https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/intro-to-tweet-json)*.*

Now, Tweepy’s search() method is great. But there is a drawback to it. It retrieves Tweets using the Search API which gives results from the past, going as far as 7 days. This means that the Tweets retrieved aren’t actually in real-time. Moreover, the Search API also has a limit of 180 tweets per 15-minute window.

Also, the Tweets retrieved aren’t exhaustive. Meaning, there might be some Tweets, matching the keywords, that might be missing from those that the  API returns. So, how do we stream Tweets in real-time? We use the Streaming API!

**Streaming Tweets in Real-Time using StreamListener**

The[Streaming API](https://developer.twitter.com/en/docs/twitter-api/v1/tweets/filter-realtime/overview) lets you stream Tweets in real-time. It is useful for obtaining a large number of tweets or for creating a live stream of tweets. Unlike the Search API, which pulls data from Twitter, the Streaming API pushes messages to a persistent session.

*“Rather than delivering data in batches through repeated requests by your client app, as might be expected from a REST API, a single connection is opened between your app and the API, with new results being sent through that connection whenever new matches occur. ” —*[*Consuming streaming data*](https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data)

Also, the Streaming API returns data in JSON and with the same fields as the Search API. We will use the Tweepy’s[StreamListener class](https://github.com/tweepy/tweepy/blob/64b42e560a71d3bbcf9cb47f039e158a860eb979/tweepy/streaming.py#L30) to stream real-time tweets. This class has several methods to handle tweets.

The **on\_data()** method in this class which receives the messages and calls the appropriate method. The **on\_status()** method handles the staus of the tweet. Since our only concern is the tweet the user creates, we will be working with the **on\_status()** method.

We need to create a custom class that will inherit the**StreamListener class** and overrides the **on\_status()** method. Here we will write our own code to extract the relevant data from the tweet. We will also override the **on\_error()** method that handles any errors that occur during the connection. We are concerned about error 420 which occurs when we exceed the limit on the number of attempts to connect to the Streaming API.

|  |  |
| --- | --- |
|  | class MyStreamListener(tweepy.StreamListener): |
|  |  |
|  | def \_\_init\_\_(self, time\_limit=300): |
|  | self.start\_time = time.time() |
|  | self.limit = time\_limit |
|  | super(MyStreamListener, self).\_\_init\_\_() |
|  |  |
|  | def on\_connect(self): |
|  | print("Connected to Twitter API.") |
|  |  |
|  | def on\_status(self, status): |
|  |  |
|  |  |
|  | # Tweet ID |
|  | tweet\_id = status.id |
|  |  |
|  | # User ID |
|  | user\_id = status.user.id |
|  | # Username |
|  | username = status.user.name |
|  |  |
|  |  |
|  | # Tweet |
|  | if status.truncated == True: |
|  | tweet = status.extended\_tweet['full\_text'] |
|  | hashtags = status.extended\_tweet['entities']['hashtags'] |
|  | else: |
|  | tweet = status.text |
|  | hashtags = status.entities['hashtags'] |
|  |  |
|  | # Read hastags |
|  | hashtags = read\_hashtags(hashtags) |
|  |  |
|  | # Retweet count |
|  | retweet\_count = status.retweet\_count |
|  | # Language |
|  | lang = status.lang |
|  |  |
|  |  |
|  | # If tweet is not a retweet and tweet is in English |
|  | if not hasattr(status, "retweeted\_status") and lang=="en": |
|  | # Connect to database |
|  | dbConnect(user\_id, username, tweet\_id, tweet, retweet\_count, hashtags) |
|  |  |
|  | if (time.time() - self.start\_time) > self.limit: |
|  |  |
|  | print(time.time(), self.start\_time, self.limit) |
|  | return False |
|  |  |
|  | def on\_error(self, status\_code): |
|  | if status\_code == 420: |
|  | # Returning False in on\_data disconnects the stream |
|  | return False |

[**view raw**](https://gist.github.com/aniruddha27/9e7c5ae250c6f0239f940b8315b3518c/raw/3b9ff3a0e77da462b4ff74cb53c2f4adee5b8822/tweepy_streamlistener.py)[**tweepy\_streamlistener.py**](https://gist.github.com/aniruddha27/9e7c5ae250c6f0239f940b8315b3518c#file-tweepy_streamlistener-py) hosted with  by [**GitHub**](https://github.com/)

As you can see, I have put the code to read the tweet inside the if-else loop. This is so because if the tweet is an extended tweet, exceeding 140 characters, it will be truncated. To retrieve the full tweet, in that case, would mean that we have to access the tweet from the *extended\_tweet* field rather than reading from the root-level *text* field.

We will be only connecting and saving the tweet to the database when the tweet is in English and is not a retweet, since our only concern is about new tweets here.

Also, I have included a code to pass False as the output when the time exceeds the time\_limit. This I have done to make sure the connection doesn’t stay alive forever. I will only be keeping it alive for 300 seconds. You can change this according to your needs.

Since the hashtags field dictionary (containing hashtags and their character indexing in the tweet) is saved within a list in the entities field, I have written a custom method **read\_hashtags()** to extract only the hashtags and save them in a Python list.

|  |  |
| --- | --- |
|  | # Extract hashtags |
|  | def read\_hashtags(tag\_list): |
|  | hashtags = [] |
|  | for tag in tag\_list: |
|  | hashtags.append(tag['text']) |
|  | return hashtags |

[**view raw**](https://gist.github.com/aniruddha27/d7af158d4d3e432a0dfdcbf440f8bcae/raw/5baeb3d6b3561cbd252c9bcb3b8e91d9b75df653/hashtags.py)[**hashtags.py**](https://gist.github.com/aniruddha27/d7af158d4d3e432a0dfdcbf440f8bcae#file-hashtags-py) hosted with  by [**GitHub**](https://github.com/)

Finally, to connect to the database and save the tweets, I will write a custom**dbConnect()** method which we will discuss in one of the succeeding sections. First, let’s set up those tables in the database!

**Creating Tables in PostgreSQL**

Now that we know how to stream tweets using the Twitter API, it time to define those tables inside the TwitterDb database that we created using pgAdmin.

Remember that I said Postgresql supports the Python Programming. Well, how does it do that? It provides a very popular database adapter called **psycopg** that let us enjoy the best of both the worlds, Python and Postgresql!

Let’s first write up the commands for creating the tables. Since Postgresql support SQL language, the syntax should be pretty easy to grasp. The important part however is understanding the various tables and how they relate to each other.

*Check out*[*this article*](https://www.analyticsvidhya.com/blog/2020/07/difference-between-sql-keys-primary-key-super-key-candidate-key-foreign-key/?utm_source=blog&utm_medium=analyse_streaming_tweets)*to understand the various keys in SQL.*

We are going to create three tables in the database.

The first table will be **TwitterUser**. It will contain user-specific information.

* user\_id – Unique user id. This will be the PRIMARY KEY for the table.
* user\_name – User screen name

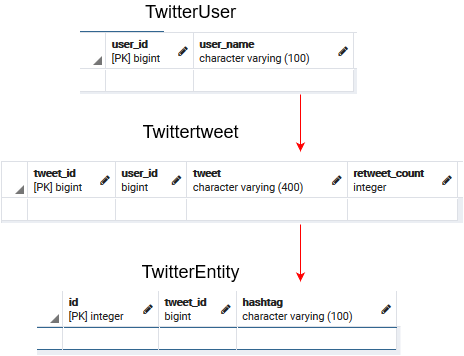
The second table will be  **TwitterTweet**. It will contain tweet specific information.

* tweet\_id – Unique tweet id. This will be the PRIMARY KEY for the table.
* tweet – Tweet text
* retweet\_count – Number of times the tweet was retweeted
* user\_id – To relate the user information with their tweet information, we are going to include the user\_id from the TwitterUser table as a FOREIGN KEY in this table.

The third table will be **TwitterEntity**. It will contain entity information. Specifically, we will be storing hashtags from the tweet but you can store other entity fields as well.

* id – Unique to represent the rows. It will be of SERIAL type which will make sure that this field increments automatically as new rows are added to the table. We don’t have to provide this value, the database will handle it on its own (perks of using Postgresql ;))
* hashtag – It will store the hashtags and will be of VARCHAR type.
* tweet\_id – To relate the table to the TwitterTweet table, we will include the tweet\_id as FOREIGN KEY in the table.

Below is a visualization of the tables and how they are related to each other.



Since the tables are related to each other, it is important to create the table in order otherwise we will get an error. First, we will create the TwitterUser table, then TwitterTweet, and finally TwitterEntity table. Let’s write these commands in a simple tuple.

|  |  |
| --- | --- |
|  | # Table creation |
|  | commands = (# Table 1 |
|  | '''Create Table TwitterUser(User\_Id BIGINT PRIMARY KEY, User\_Name TEXT);''', |
|  | # Table 2 |
|  | '''Create Table TwitterTweet(Tweet\_Id BIGINT PRIMARY KEY, |
|  | User\_Id BIGINT, |
|  | Tweet TEXT, |
|  | Retweet\_Count INT, |
|  | CONSTRAINT fk\_user |
|  | FOREIGN KEY(User\_Id) |
|  | REFERENCES TwitterUser(User\_Id));''', |
|  | # Table 3 |
|  | '''Create Table TwitterEntity(Id SERIAL PRIMARY KEY, |
|  | Tweet\_Id BIGINT, |
|  | Hashtag TEXT, |
|  | CONSTRAINT fk\_user |
|  | FOREIGN KEY(Tweet\_Id) |
|  | REFERENCES TwitterTweet(Tweet\_Id));''') |

[**view raw**](https://gist.github.com/aniruddha27/8b0cd3a181d99fccd44806f0d462d933/raw/d341beeef0dc32dda42d030a3e17ef0849c6378f/table_creation_sql.py)[**table\_creation\_sql.py**](https://gist.github.com/aniruddha27/8b0cd3a181d99fccd44806f0d462d933#file-table_creation_sql-py) hosted with  by [**GitHub**](https://github.com/)

Now, let’s make a connection to the Postgresql server using **psycopg**. For that, you need to import the psycopg module and call the **connect()** method. In the method, you need to pass some important arguments.

* host – Database server address
* port – The port number on which the server is running. By default, it is 5432.
* database – Name of the database name you want to connect to.
* user – The username to authenticate.
* password – Password to authenticate the user.

import psycopg2

# Connection to database server

conn = psycopg2.connect(host="localhost",database="TwitterDB",port=5432,user=<user>,password=<password>)

The **connect()** method creates a new Database session and returns an instance of the connection class. Now we need to generate a Cursor object using the **cursor()** method which allows us to execute queries against a database using the **execute()** method.

|  |  |
| --- | --- |
|  | # Create cursor to execute SQL commands |
|  | cur = conn.cursor() |
|  |  |
|  | # Execute SQL commands |
|  | for command in commands: |
|  | # Create tables |
|  | cur.execute(command) |
|  |  |
|  | # Close communication with server |
|  | conn.commit() |
|  | cur.close() |
|  | conn.close() |

[**view raw**](https://gist.github.com/aniruddha27/7bff74b294db9a7ad014dbe73f874ab2/raw/8542595e64de0cab5bc114dde766f289dbc3ccc9/cursor_psycopg.py)[**cursor\_psycopg.py**](https://gist.github.com/aniruddha27/7bff74b294db9a7ad014dbe73f874ab2#file-cursor_psycopg-py) hosted with  by [**GitHub**](https://github.com/)

Don’t forget to commit the changes and close connection after use so that the database doesn’t get locked!

Now its time to define that custom **dbConnect()** function, that we created in the **MyStreamListener** class, that will take the data from the streaming tweets and save to our database.

**Connecting to the Database**

We have already seen how to connect to the database and how to execute SQL commands from Python using psycopg. Now we will do those together. Whenever a tweet needs to be saved to the database, we will first open a connection to the database, execute the SQL command to the database, commit the changes, and finally close the connection. This is what is happening in the **dbConnect()** method below.

|  |  |
| --- | --- |
|  | # Insert Tweet data into database |
|  | def dbConnect(user\_id, user\_name, tweet\_id, tweet, retweet\_count, hashtags): |
|  |  |
|  | conn = psycopg2.connect(host="localhost",database="TwitterDB",port=5432,user=<user>,password=<password>) |
|  |  |
|  | cur = conn.cursor() |
|  |  |
|  | # insert user information |
|  | command = '''INSERT INTO TwitterUser (user\_id, user\_name) VALUES (%s,%s) ON CONFLICT |
|  | (User\_Id) DO NOTHING;''' |
|  | cur.execute(command,(user\_id,user\_name)) |
|  |  |
|  | # insert tweet information |
|  | command = '''INSERT INTO TwitterTweet (tweet\_id, user\_id, tweet, retweet\_count) VALUES (%s,%s,%s,%s);''' |
|  | cur.execute(command,(tweet\_id, user\_id, tweet, retweet\_count)) |
|  |  |
|  | # insert entity information |
|  | for i in range(len(hashtags)): |
|  | hashtag = hashtags[i] |
|  | command = '''INSERT INTO TwitterEntity (tweet\_id, hashtag) VALUES (%s,%s);''' |
|  | cur.execute(command,(tweet\_id, hashtag)) |
|  |  |
|  | # Commit changes |
|  | conn.commit() |
|  |  |
|  | # Disconnect |
|  | cur.close() |
|  | conn.close() |

[**view raw**](https://gist.github.com/aniruddha27/cd5764bbac6c8ec29ff3bf7a9675e223/raw/fe373389baa326a05162af0cb1c60bdef37a6c83/tweets_into_database.py)[**tweets\_into\_database.py**](https://gist.github.com/aniruddha27/cd5764bbac6c8ec29ff3bf7a9675e223#file-tweets_into_database-py) hosted with  by [**GitHub**](https://github.com/)

There are however two important things to point out here.

Firstly, I have used the[ON CONFLICT](https://www.postgresqltutorial.com/postgresql-upsert/) command while inserting information into the TwitterUser table that is storing information about the author of the Tweet. This is so because a user can create multiple tweets but we don’t want to store the information about the same user multiple times. That’s the beauty of RDBMS!

Secondly, I have used a for-loop for inserting information into the TwitterEntity table. This is so because the hashtags parameter received by this function is a Python list that we generated in our custom **MyStreamListener** class. Since RDBMS needs to maintain atomicity, we need to insert each of these hashtags as new rows.

Once we are done adding values to the tables, we commit and close the connection.

**Streaming Tweets into Database**

We have set up how we will stream tweets in real-time and how we will store them in the database. Now, it is time to call the Streaming API and store those tweets in the database. To do that we need to first create an instance of our**MyStreamListener()** class. This instance along with our API authentication will be passed to**tweepy. Stream()** method. Finally, we can filter the keywords we want to receive by using the **filter()** method on the instance returned by tweepy.Stream().

|  |  |
| --- | --- |
|  | # Streaming tweets |
|  |  |
|  | myStreamListener = MyStreamListener() |
|  | myStream = tweepy.Stream(auth=api.auth, listener=myStreamListener, |
|  | tweet\_mode="extended") |
|  | myStream.filter(track=['Covid','covid-19']) |

[**view raw**](https://gist.github.com/aniruddha27/979ecc27c94e16a5681c2712641ea6c4/raw/2e82f4c6442b51dfda32d5afb8a79087ab95eb68/streaming_tweets.py)[**streaming\_tweets.py**](https://gist.github.com/aniruddha27/979ecc27c94e16a5681c2712641ea6c4#file-streaming_tweets-py) hosted with  by [**GitHub**](https://github.com/)

The **tweet\_mode** argument specifies whether you want to access the tweets in extended mode or not.

Its time to analyse the streamed tweets, now that we have successfully streamed and stored them!

**Analysing streamed tweets**

Having finished with the tricky part already, analysing tweets should be a piece of cake! I am going to do three things here-

* Check out the most commonly occurring words in the tweets.
* Determine the sentiment of the tweets.
* Chart out the count of the hashtags that we retrieved from the tweets.

Let’s start out by importing the relevant libraries.

from nltk.stem import WordNetLemmatizer

import numpy as np

from textblob import TextBlob

import re

from wordcloud import WordCloud

import matplotlib.pyplot as plt

import seaborn as sns

The first thing to do when working with text data is to preprocess the text. So let’s do that!

We are going to do the following things to preprocess the tweets:

* Remove tweet URLs.
* Remove any mentions that were present in the tweet.
* Remove all non-alphanumeric characters from the tweet.
* Lemmatize the words in the tweet.
* Finally, convert the tweet to the lowercase.

|  |  |
| --- | --- |
|  | # Cleaning the tweets |
|  | def preprocess(tweet): |
|  |  |
|  | # remove links |
|  | tweet = re.sub(r'http\S+', '', tweet) |
|  | # remove mentions |
|  | tweet = re.sub("@\w+","",tweet) |
|  | # alphanumeric and hashtags |
|  | tweet = re.sub("[^a-zA-Z#]"," ",tweet) |
|  | # remove multiple spaces |
|  | tweet = re.sub("\s+"," ",tweet) |
|  | tweet = tweet.lower() |
|  | # Lemmatize |
|  | lemmatizer = WordNetLemmatizer() |
|  | sent = ' '.join([lemmatizer.lemmatize(w) for w in tweet.split() if len(lemmatizer.lemmatize(w))>3]) |
|  |  |
|  | return sent |

[**view raw**](https://gist.github.com/aniruddha27/8d112b87ff4014b80f606dc68080066d/raw/d847792b1cc3be153504f68fba661722ce105aac/preprocess_tweets.py)[**preprocess\_tweets.py**](https://gist.github.com/aniruddha27/8d112b87ff4014b80f606dc68080066d#file-preprocess_tweets-py) hosted with  by [**GitHub**](https://github.com/)

I am going to create a separate function to execute the SQL queries on the database.

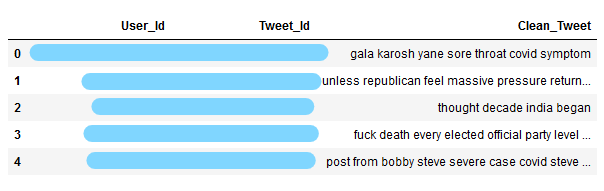
|  |  |
| --- | --- |
|  | # Connecting to the Database |
|  | def DbConnect(query): |
|  |  |
|  | conn = psycopg2.connect(host="localhost",database="TwitterDB",port=5432,user=<user>,password=<password>) |
|  | curr = conn.cursor() |
|  |  |
|  | curr.execute(query) |
|  |  |
|  | rows = curr.fetchall() |
|  |  |
|  | return rows |

[**view raw**](https://gist.github.com/aniruddha27/70099e5c18917e1b8449a67bf60ca87d/raw/8923f5f597acf3db6b39c4ff5a6581cad92e4b32/dbconnect.py)[**dbconnect.py**](https://gist.github.com/aniruddha27/70099e5c18917e1b8449a67bf60ca87d#file-dbconnect-py) hosted with  by [**GitHub**](https://github.com/)

Then, let’s write up the SQL queries to retrieve the tweets from the database and save the resulting data into a dataframe for analyzing.

|  |  |
| --- | --- |
|  | # Create |
|  | data\_tweet = DbConnect("SELECT User\_Id, Tweet\_Id, Tweet FROM TwitterTweet;") |
|  |  |
|  | df\_tweet = pd.DataFrame(columns=['User\_Id','Tweet\_Id','Clean\_Tweet']) |
|  |  |
|  | for data in data\_tweet: |
|  | index = len(df\_tweet) |
|  | df\_tweet.loc[index,'User\_Id'] = data[0] |
|  | df\_tweet.loc[index,'Tweet\_Id'] = data[1] |
|  | df\_tweet.loc[index,'Clean\_Tweet'] = preprocess(data[2]) |
|  |  |
|  | df\_tweet.head() |

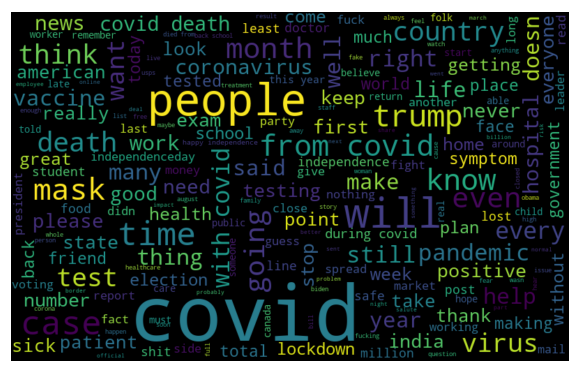
[**view raw**](https://gist.github.com/aniruddha27/c0321317928e030224016bc65ca543b4/raw/900f9f56b5149e4e0fc265ae98839eb5e52ff3c4/sql_query_tweets.py)[**sql\_query\_tweets.py**](https://gist.github.com/aniruddha27/c0321317928e030224016bc65ca543b4#file-sql_query_tweets-py) hosted with  by [**GitHub**](https://github.com/)



Let’s create a word cloud of the most commonly occurring words in the tweets.

|  |  |
| --- | --- |
|  | # Most commomly occuring words |
|  | def keywords(): |
|  | all\_words = ' '.join([text for text in df\_tweet['Clean\_Tweet']]) |
|  | wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(all\_words) |
|  |  |
|  | plt.figure(figsize=(10, 7)) |
|  | plt.imshow(wordcloud, interpolation="bilinear") |
|  | plt.axis('off') |
|  | plt.show() |
|  |  |
|  | keywords() |

[**view raw**](https://gist.github.com/aniruddha27/9f7e822b21411439c46bdb1e419407e4/raw/3c86a6a9facc1da3bd7a4d68df153cabbe21304f/wordcloud.py)[**wordcloud.py**](https://gist.github.com/aniruddha27/9f7e822b21411439c46bdb1e419407e4#file-wordcloud-py) hosted with  by [**GitHub**](https://github.com/)

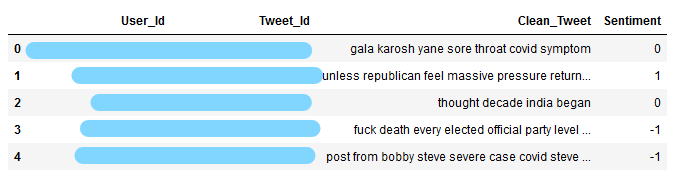


We can notice words like “Covid”, “mask”, “pandemic”, “virus”, “lockdown”, etc. occuring quite frequently in the tweets (as expected).

Then let’s check out the sentiment of the tweets using the[Textblob](https://www.analyticsvidhya.com/blog/2018/02/natural-language-processing-for-beginners-using-textblob/?utm_source=blog&utm_medium=analyse_streaming_tweets) library. We will be using the **sentiment()** method to call upon the tweet and check out the polarity of the sentiment. Further, the polarity of -1 means that the tweet is a negative statement and 1 means that the tweet is a positive statement (a blessing in these tough times!).

|  |  |
| --- | --- |
|  | # Sentiment analysis using Textblob |
|  | def sentiment(tweet): |
|  |  |
|  | analysis = TextBlob(tweet) |
|  | if analysis.sentiment.polarity > 0: |
|  | return 1 |
|  | elif analysis.sentiment.polarity == 0: |
|  | return 0 |
|  | else: |
|  | return -1 |
|  |  |
|  | df\_tweet['Sentiment'] = df\_tweet['Clean\_Tweet'].apply(sentiment) |
|  |  |
|  | df\_tweet.head(20) |

[**view raw**](https://gist.github.com/aniruddha27/7d1a63ddaed056009d6f7ad88a9714fb/raw/2ce9ceaf754a0a0dd4457a9e8683e522c8f75077/tweet_sentiment.py)[**tweet\_sentiment.py**](https://gist.github.com/aniruddha27/7d1a63ddaed056009d6f7ad88a9714fb#file-tweet_sentiment-py) hosted with  by [**GitHub**](https://github.com/)

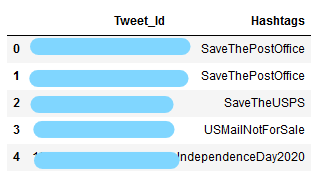


Now let’s do the final thing we wanted to achieve in this analysis – determining the most commonly occurring hashtags!

Firstly, we need to query and retrieve the hashtags from our TwitterEntity table in the database.

|  |  |
| --- | --- |
|  | # Querying hashtags from database |
|  | data\_tags = DbConnect("SELECT Tweet\_Id, Hashtag FROM TwitterEntity;") |
|  |  |
|  | df\_tags = pd.DataFrame(columns=['Tweet\_Id','Hashtags']) |
|  |  |
|  | for data in data\_tags: |
|  | index = len(df\_tags) |
|  | df\_tags.loc[index,'Tweet\_Id'] = data[0] |
|  | df\_tags.loc[index,'Hashtags'] = data[1] |
|  |  |
|  | df\_tags.head(20) |

[**view raw**](https://gist.github.com/aniruddha27/964a836b35d8dc6a156f593bf272b0dd/raw/f7ea37a6781f88942a86d31b9ae99bd60967908f/sql_query_hashtags.py)[**sql\_query\_hashtags.py**](https://gist.github.com/aniruddha27/964a836b35d8dc6a156f593bf272b0dd#file-sql_query_hashtags-py) hosted with  by [**GitHub**](https://github.com/)

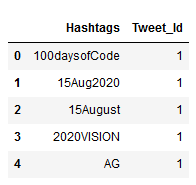


We have 277 unique hashtags here, that’s almost 75% of the total number of tweets we mined from Twitter! People surely use hashtags a lot in their tweets.

Let’s create a[pivot table](https://www.analyticsvidhya.com/blog/2020/03/pivot-table-pandas-python/?utm_source=blog&utm_medium=analyse_streaming_tweets) to determine the count of the unique hashtags from the tweets.

|  |  |
| --- | --- |
|  | # Unique hashtag counts |
|  | table = df\_tags.pivot\_table(index="Hashtags",values='Tweet\_Id',aggfunc=len) |
|  |  |
|  | # Convert pivot table to dataframe |
|  | df\_pivot = pd.DataFrame(table.to\_records()) |
|  |  |
|  | df\_pivot.head() |

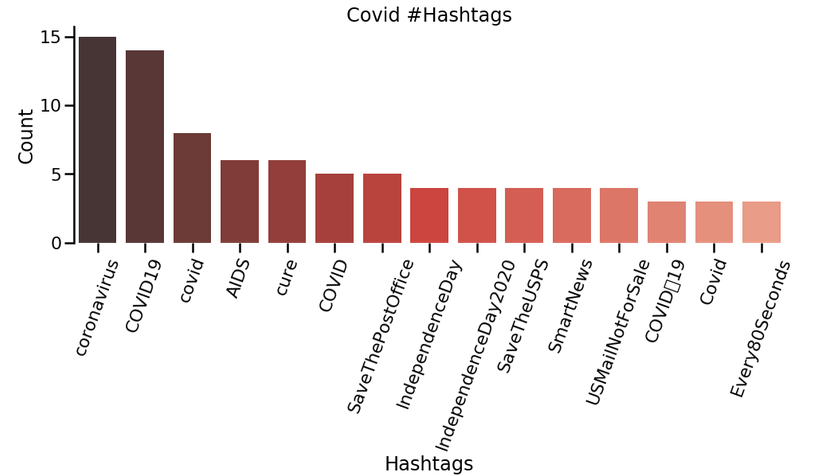
[**view raw**](https://gist.github.com/aniruddha27/bf71ad077f01bd8a4d9169abf4122d88/raw/b66d88a5af682beed2fbd63cbe3b3de1ace0b1e9/hastags_pivot_table.py)[**hastags\_pivot\_table.py**](https://gist.github.com/aniruddha27/bf71ad077f01bd8a4d9169abf4122d88#file-hastags_pivot_table-py) hosted with  by [**GitHub**](https://github.com/)



Finally, let’s visualize these hashtags.

|  |  |
| --- | --- |
|  | # Plotting hashtags counts |
|  |  |
|  | data = df\_pivot.nlargest(columns="Tweet\_Id", n = 15) |
|  |  |
|  | # Creating bar graph |
|  | plt.figure(figsize=(16,5)) |
|  | ax = sns.barplot(data=data, x= "Hashtags", y = "Tweet\_Id", palette=("Reds\_d")) |
|  |  |
|  | # Altering the visual elements |
|  | sns.set\_context("poster") |
|  | ax.set(ylabel = 'Count') |
|  | ax.set\_xticklabels(labels=ax.get\_xticklabels(),rotation=70) |
|  |  |
|  | plt.title('Covid #Hashtags') |
|  |  |
|  | ax.spines['top'].set\_visible(False) |
|  | ax.spines['right'].set\_visible(False) |
|  | ax.spines['bottom'].set\_visible(False) |
|  |  |
|  | # Output plot |
|  | plt.show() |

[**view raw**](https://gist.github.com/aniruddha27/60659eb05a9845951bbba4b88c6a6bbc/raw/6878f2e7da6b4f3b2aeb37b28e4e2778fb590b64/visualize_hashtags.py)[**visualize\_hashtags.py**](https://gist.github.com/aniruddha27/60659eb05a9845951bbba4b88c6a6bbc#file-visualize_hashtags-py) hosted with  by [**GitHub**](https://github.com/)



As you can see, although COVID-19 is mostly used hashtag in the tweets, there are other events that people want to talk about in their tweets as well. Twitter is truly THE place to understand the present feelings of the people. Well done Twitter!

**Endnotes**

To summarize, we learned to stream tweets in real-time, store them in a database, and finally analyse streaming tweets. This was really a fun project to work on, hope you were able to get a glimpse of the responsibilities of a data engineer! Surely, there were a lot of other things we could have retrieved and analysed from the tweets, but I leave it up to you to experiment with them and come up with your own mind-boggling analysis.

*You can check out this*[*amazing article*](https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/?utm_source=blog&utm_medium=analyse_streaming_tweets)*and this super awesome*[*free course*](https://courses.analyticsvidhya.com/courses/twitter-sentiment-analysis?utm_source=blog&utm_medium=analyse_streaming_tweets)*on Twitter analysis that we have created for you.*

I hope you have understood each and every concept in this article. In case you have any queries, reach out to me in the comments below.