SNS Scraping Lab

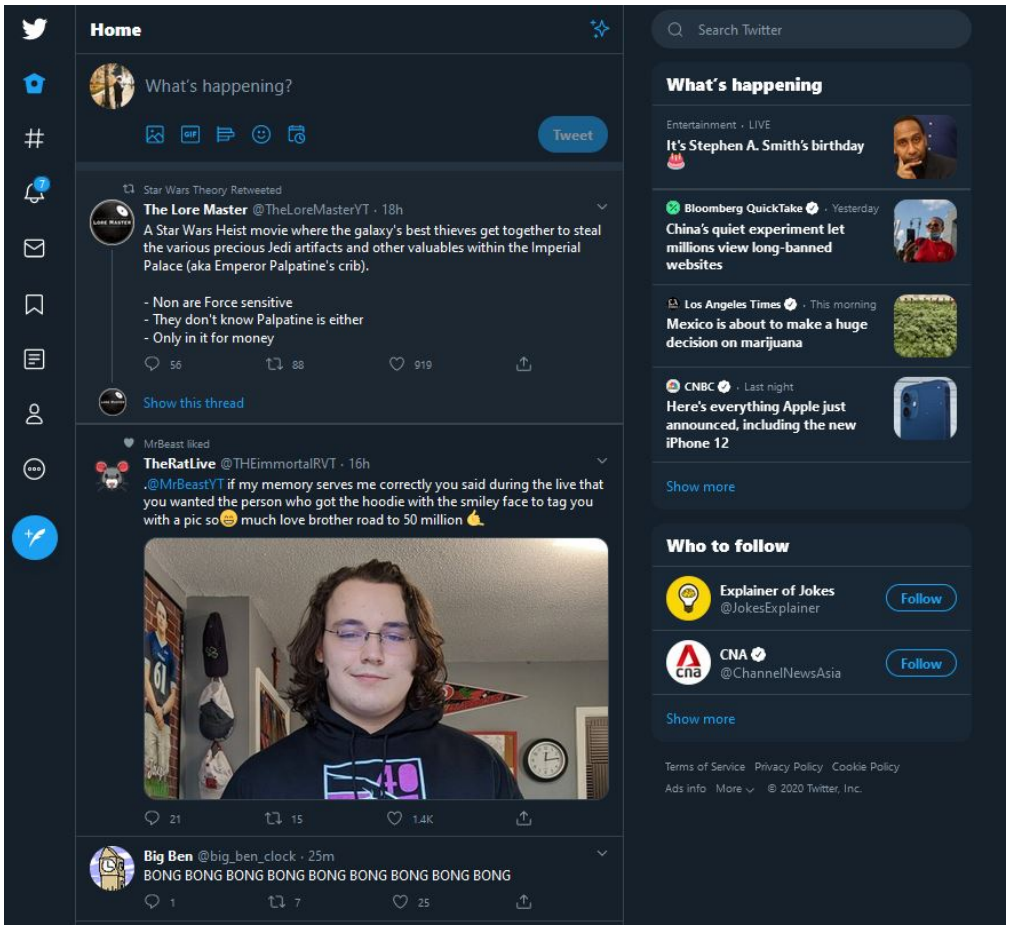
Contents

Please refer to the notebook, "SNS_scraping.ipynb" for detailed codes.

1. Introduction

Social media is one of the major platforms that contain vast behavioural data, such as engagement level and sentiments of people. Combining this with other relevant data, such as company-related information, can bring about amazing insights helpful for companies and researchers of various fields.

Twitter is a social media platform that allows users to microblog - to interact with each other and spread messages through short messages. Twitter is one of the most widely used social media platforms in the world, and users are often posting unfiltered opinions that are able to be retrieved easily.



Twitter offers many information that is available for data scientists. Twitter provides functions to:

1. GET for retrieving data
2. POST for creating data
3. PUT for updating data
4. DELETE for removing data

However, for this lab guide, we will be taking a look at mainly the GET functions as it is the main function for getting Twitter data.

There are 4 main types of data that we can acquire:

1. Tweets
2. Entities (e.g. hashtags, media)
3. Places
4. Users

For this lab guide, we will be taking a look at acquiring the tweets, the trending hashtags, and the users and their tweets.

- Tweets: also known as "status updates", these are the texts that users post.

- Entities: contains metadata and additional contextual information about the tweets (e.g. hashtags, user mentions, links, polls, and attached media such as photos and videos)
- Users: contains the account metadata that describes a particular user of Twitter (e.g. number of followers, number of tweets posted)

In order to access these Twitter data, we need to use Twitter API. This allows users to gather tweets and related metadata. To use this API, users will have to create an app in the Twitter Developers page, acquire the relevant keys for safe access, and initialise the Twitter API modules to access Twitter data. For more detailed information with illustrations, please refer to the detailed lab guide provided.

Due to the complexity of acquiring the keys, I have already acquired one that will solely be used for this lab guide purposes only. It is important that these authorisation keys are NOT TO BE SHARED with people outside of this class. For security reasons, the authorisation key will written on the notebook provided. The authentication keys are included in the **notebook provided.**

2. Initialising Twitter API

Twitter utilises OAuth, which acts as a key that grants the users access to private resources on a website (in this case, Twitter). Twitter API integrates with OAuth to create an authentication code that allows you to access the Twitter data.

In the code below, we input the authentication codes into the OAuth function and pass that authentication key to the Twitter API through twitter.Twitter function. This is the initialisation step for Twitter API, which creates an object that contains the authenticated API.

```
auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
                           CONSUMER_KEY, CONSUMER_SECRET)

twitter_api = twitter.Twitter(auth=auth)

print(twitter_api)
```

```
<twitter.api.Twitter object at 0x7ff8ecc47080>
```

The first priority of scraping from Twitter would be to gather the tweets. However, we have to first determine the criteria for the tweets to be gathered. Therefore, we can either gather tweets according to the trending topics or we can gather tweets according to a particular user.

We will first take a look at the trending topics.

3. Trending Topics

One of the main features of Twitter is that it allows users to see what are the trending topics. These trending topics can be in the form of hashtags or simple keywords that are being said by a lot of users at a given time. The easiest way to look at the trending topics is through location.

By using Twitter API's **trends.place** function, we can gather information regarding the trending topics in a particular place that we want to look at. For this example, we will take a look at the trending topics of the entire world and the USA.

The locations are determined through Yahoo API's "Where on Earth ID", but as of this moment, it is extremely difficult to attain the IDs as there are numerous procedures. The IDs for the entire world and the USA are shown a lot in other tutorials and websites, and that is why we were able to attain them easily.

- Entire world ID: 1
- USA ID: 23424977

```
WORLD_WOE_ID = 1
US_WOE_ID = 23424977

world_trends = twitter_api.trends.place(_id=WORLD_WOE_ID)
us_trends = twitter_api.trends.place(_id=US_WOE_ID)
```

When we print out the variable "world_trends" containing the trending topics of the world, we see a messy dictionary in a list type of data. Unfortunately, this is the common returned data type for many Twitter API functions. If we take a closer look, we can see that for key: **trends**, we can get the **name** of the trending topic, the **url** for searching the topic, and the **tweet_volume** that indicates the number of tweets that have mentioned this particular topic. At the bottom, **as_of** shows the time when this query has been made and the **locations** that specifies the location of the query made for trending topics.

```
print(world_trends)
```

```
[{'trends': [{'name': '#恋あた', 'url': 'http://twitter.com/search?q=%23%E
6%81%8B%E3%81%82%E3%81%9F', 'promoted_content': None, 'query': '%23%E6%8
1%8B%E3%81%82%E3%81%9F', 'tweet_volume': 47989}, {'name': '#フォロワーと戦う
', 'url': 'http://twitter.com/search?q=%23%E3%83%95%E3%82%A9%E3%83%AD%E3%
83%AF%E3%83%BC%E3%81%A8%E6%88%A6%E3%81%86', 'promoted_content': None, 'qu
ery': '%23%E3%83%95%E3%82%A9%E3%83%AD%E3%83%AF%E3%83%BC%E3%81%A8%E6%88%A
6%E3%81%86', 'tweet_volume': 19797}, {'name': '対戦相手の武器', 'url': 'htt
p://twitter.com/search?q=%E5%AF%BE%E6%88%A6%E7%9B%B8%E6%89%8B%E3%81%AE%E
6%AD%A6%E5%99%A8', 'promoted_content': None, 'query': '%E5%AF%BE%E6%88%A
6%E7%9B%B8%E6%89%8B%E3%81%AE%E6%AD%A6%E5%99%A8', 'tweet_volume': 17881},
```

```
print(us_trends)
```

```
[{'trends': [{'name': '#InternationalAssDay', 'url': 'http://twitter.com/
search?q=%23InternationalAssDay', 'promoted_content': None, 'query': '%23
InternationalAssDay', 'tweet_volume': None}, {'name': '#TuesdayThoughts',
'url': 'http://twitter.com/search?q=%23TuesdayThoughts', 'promoted_conten
t': None, 'query': '%23TuesdayThoughts', 'tweet_volume': 38345}, {'name':
'jimins', 'url': 'http://twitter.com/search?q=jimins', 'promoted_content
': None, 'query': 'jimins', 'tweet_volume': 12959}, {'name': '#tuesdayvib
es', 'url': 'http://twitter.com/search?q=%23tuesdayvibes', 'promoted_cont
ent': None, 'query': '%23tuesdayvibes', 'tweet_volume': 22713}, {'name':
```

For now, let us say that we are only interested in the trending topics. We can gather them simply through the method shown below:

```
for trend in world_trends[0]['trends']:
    print(trend['name'])
```

```
#恋あた
#フォロワーと戦う
対戦相手の武器
#SixTONESと同じ時代刻もう
#SKZ_ALLIN_MV
#あなたを殺したのはだれ
漫画の名シーン
Sexy松
jimins
European Premier League
Andrea Levy
メガ進化
一般人の変態度
hobi
米司法省
SAOコラボ
XY&Z
러브킬러
マッチングアプリ
#生SixTONES
```

Here, we are simply iterating through the dictionary in list of world_trends, and printing the name of the trending topic. We can see that we were able to get only 20 trending topics through this query as Twitter only shows 20 trending topics at a time (Top 20 trending topics).

The trending topics for USA are shown below:

```
for trend in us_trends[0]['trends']:
    print(trend['name'])
```

```
#InternationalAssDay
#TuesdayThoughts
jimins
#tuesdayvibes
#MeToobin
#minecraftmanhunt
Tiffany Trump
Madam Vice President
Happy Birthday Kamala
Michael Steele
John Ross
PragerU
Champions League
Tony Katz
Marilyn Monroe
ERIC NAM
Happy Birthday Senator Harris
Madam VP
Troy Aikman
14 Days
```

4. Querying Tweets

We have found the trending topics in Twitter for some specific time. However, the end goal is to gather the tweets and related data. By using the trending topics as queries, we can gather the tweets that include the trending topics. Let's try to get tweets that contain the hashtag: #TuesdayThoughts using **search.tweets(q=' ', count=).** This function takes in the query term at the argument "q", and takes in the number of tweets to gather in the argument "count".

```
searched = twitter_api.search.tweets(q='#TuesdayThoughts', count=100)
```

```
print(searched)
```

```
{'statuses': [{'created_at': 'Tue Oct 20 14:51:26 +0000 2020', 'id': 1318
565495868620801, 'id_str': '1318565495868620801', 'text': 'RT @paulbiagi:
⭐⭐⭐⭐⭐One of the best alien encounter stories I have read.\n"#AmReadin
g #BookReview #mustread #reading #books #IARTG  #BookBoo…', 'truncated':
False, 'entities': {'hashtags': [{'text': 'AmReading', 'indices': [74, 8
4]}, {'text': 'BookReview', 'indices': [85, 96]}, {'text': 'mustread', 'i
ndices': [97, 106]}, {'text': 'reading', 'indices': [107, 115]}, {'text':
'books', 'indices': [116, 122]}, {'text': 'IARTG', 'indices': [123, 12
9]}], 'symbols': [], 'user_mentions': [{'screen_name': 'paulbiagi', 'name
': 'paulbiagi', 'id': 1033101962986774528, 'id_str': '1033101962986774528
', 'indices': [3, 13]}], 'urls': []}, 'metadata': {'iso_language_code': '
en', 'result_type': 'recent'}, 'source': '<a href="https://mobile.twitte
r.com" rel="nofollow">Twitter Web App</a>', 'in_reply_to_status_id': Non
e, 'in_reply_to_status_id_str': None, 'in_reply_to_user_id': None, 'in_re
ply_to_user_id_str': None, 'in_reply_to_screen_name': None, 'user': {'id
': 872131125187141638, 'id_str': '872131125187141638', 'name': 'Andria St
one', 'screen_name': 'andria_mavrek', 'location': 'Florida, USA', 'descri
ption': 'Political Activist 🌊 World Traveler & Novelist of The EDGE Tril
ogy: Edge Of The Future, Edge Of The Stars & Edge Of The Rings. DMs ONLY
```

The result for the query returns a dictionary containing 2 keys. The 2 keys are "statuses" and "search_metadata". Statuses contain the actual tweets collected and the

corresponding metadata. Search_metadata contains the information about the query itself, which is not that useful at the moment. We will mainly focus on the "statuses" key.

We have to be aware of the data structure when navigating through the collected tweets.

```
print(searched['statuses'][0]['text'])
```

```
RT @paulbiagi: ⭐⭐⭐⭐⭐One of the best alien encounter stories I have re
ad.
"#AmReading #BookReview #mustread #reading #books #IARTG  #BookBoo…
```

```
len(searched['statuses'])
```

```
100
```

It seems that we have gathered 100 tweets. We have set the limit to 100 tweets, and 100 tweets were available at the time.

In the key, "statuses" refers to the actual tweets and the corresponding metadata. For example, in each status, "text" section contains the actual tweet, "id" section contains the id number for that tweet, "entities" contains the entities for the particular tweet, such as hashtags.

Now, let's try to extract 10 tweets among the tweets that we have gathered. Along with the actual tweets, let's try to also extract the **number of favourites,** and the **number of retweets.**

Number of favourites refers to the number of times the tweet was favourited by another use. This is a feature of Twitter that allows users to "like" the tweet.

Retweets refers to a user relaying the tweet to their Twitter profile, allowing their followers to see the particular tweet.

We will first store all statuses in the "statuses" variable for easier iteration. Then, now, we will iterate through first 10 of these tweets gathered, and print the tweet, the number of favourites, and the number of retweets.

```
for i in range(10):
    print()
    print(statuses[i]['text'])
    print('Favourites: ', statuses[i]['favorite_count'])
    print('Retweets: ', statuses[i]['retweet_count'])
```

```
RT @paulbiagi: ⭐⭐⭐⭐⭐One of the best alien encounter stories I have re
ad.
"#AmReading #BookReview #mustread #reading #books #IARTG  #BookBoo…
Favourites:  0
Retweets:  1

अगर आप जिंदगी में सुख-शांति और खुशियां चाहते हो तो ,रोज़ाना भगवान के नाम का सिमरन करें और
उस पर दृढ़ विश्वास रखें ,… https://t.co/GDAmS4n3E6
Favourites:  0
Retweets:  0

If you always want happiness and peace in life, then chant the name of Go
d daily and have #FirmFaithInGod he will s… https://t.co/xwl1LeEzIA
Favourites:  0
Retweets:  0

#TuesdayThoughts
भगवान का नाम सभी सुखों की खान है।अगर आप मालिक की रहमतों को हासिल करना चाहते हैं तो… ht
tps://t.co/QDKsv87VVo
Favourites:  0
Retweets:  0

RT @SeemaIn98413785: https://t.co/1BIlOLdDnf
```

We can see from the above code that we can separate and extract the tweets and their corresponding metadata. So, if necessary, we can store each of them as a separate variable and make them into features for later use.

From the texts we have gathered, let's now try to extract the tweet, the user name of the one who tweeted, and the hashtags involved in the tweet.

```
status_texts = [status['text'] for status in statuses]

screen_names = [user_mention['screen_name']
                for status in statuses
                    for user_mention in status['entities']['user_mentions']]

hashtags = [hashtag['text'] for status in statuses
                for hashtag in status['entities']['hashtags']]
```

Through this process alone, we can now proceed with extremely important and interesting analysis such as text vectorisation. We will explore such analysis later on.

As expected, we have gathered 100 tweets. So far, we have gathered tweets that contained the trending topics. What if we want to collect tweets that contained different key words other than the trending topics? What if we wanted to see the tweets that contained the keyword: "COVID"?

Easy, we can do this with the same procedure as how we gathered tweets with the trending topics.

```
searched_2 = twitter_api.search.tweets(q='COVID', count=100)
```

In the variable "searched_2", we have gathered recent tweets that contain the keyword, "COVID". We can do the same procedure as above, gathering the tweets and their corresponding metadata for further analysis.

```
# storing the tweets and their info into statuses_2 variable
statuses_2 = searched_2['statuses']
```

```
len(statuses_2)
```

```
100
```

```
status_texts_2 = [status['text'] for status in statuses_2]

screen_names_2 = [user_mention['screen_name']
                for status in statuses_2
                    for user_mention in status['entities']['user_mentions']]

hashtags_2 = [hashtag['text'] for status in statuses_2
                for hashtag in status['entities']['hashtags']]
```

Now, we have gathered the tweets containing the terms "COVID", and we have extracted the tweets, their user names, and their hashtags. For the next section, to make the computations easier, we will focus only on the first 10 tweets gathered.

```
status_10 = status_texts_2[:10]
len(status_10)
```

```
10
```

```
status_10
```

```
['RT @Ambrosia_Ijebu: You cannot Curfew a state or country indefinitely.
Covid should have taught us all that. You must still SHOW WORKINGS o…',
 'RT @mlizza_: I've seen Twitter really change peoples life, so I'm gonna
give it a try. My husband opened up this restaurant called Cocina E…',
 'RT @nuernberg_de: Die aktuellen Zahlen aus den Krankenhäusern in #Nürnb
erg:\n🏥 #COVID19-Patienten Normalstation: 35 (+5)\n🏥 COVID-19-Patient
e…',
 'RT @FINALLEVEL: FYI: Somebody else I know just died from Covid. That ma
kes 8 people dead that I knew personally.. This shit is not over. St…',
 'RT @FruitloopBirb: Each chair in this picture represents 10 people who
died from Covid-19 in the United States.\n\nIf each chair represented…',
 'Zé Felipe revela que ele e Virginia Fonseca testaram positivo para a Co
vid-19...\nhttps://t.co/2rz7twE0nL',
 'RT @polliforero: 🔴 AHORA. El séptimo Juzgado de Garantía de Santiago a
ccedió a la solicitud de la defensa del exministro de Salud, Jaime Ma…',
 'Acho que não estou mesmo com covid',
 'RT @physorg_com: How COVID-19 #cough #clouds travel in the presence and
absence of face masks: study https://t.co/nMmX2y0m98',
 'RT @Mochievous: In the US during COVID some folks pushed for lawyers to
be labelled as essential workers &amp;amp; this happened in some states. \n
I…']
```

5. Text Vectorisation

Since we have now gathered the tweets according to some query that we want, we can now perform text vectorisation methods to quantify the texts. The words in the text will be encoded as integers or floating-point values for use as input. By quantifying the texts, we can make them into usable features for further analysis, such as machine learning usage.

We will take a look at 2 common methods of vectorisation: Count Vectorisation and TF-IDF Vectorisation.

Count Vectorisation

Count vectorising refers to a method of tokenising a collection of text documents and building a vocabulary of known words while encoding new documents using that vocabulary built.

The method of creation is as follows:

1. Create an instance of the CountVectorizer class
2. Call the fit() function in order to learn a vocabulary from one or more documents
3. Call the transform() function on one or more documents as needed to encode each as a vector

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
x = vectorizer.fit_transform(status_10)

print(vectorizer.get_feature_names())
```

```
['10', '19', '2rz7twe0nl', '35', 'absence', 'accedió', 'acho', 'ahora', '
aktuellen', 'all', 'ambrosia_ijebu', 'amp', 'and', 'as', 'aus', 'be', 'ca
lled', 'cannot', 'chair', 'change', 'clouds', 'co', 'cocina', 'com', 'cou
gh', 'country', 'covid', 'covid19', 'curfew', 'de', 'dead', 'defensa', 'd
el', 'den', 'die', 'died', 'during', 'each', 'el', 'ele', 'else', 'essent
ial', 'estou', 'exministro', 'face', 'felipe', 'finallevel', 'folks', 'fo
nseca', 'for', 'from', 'fruitloopbirb', 'fyi', 'garantía', 'give', 'gonna
', 'happened', 'have', 'how', 'https', 'husband', 'if', 'in', 'indefinite
ly', 'is', 'it', 'jaime', 'just', 'juzgado', 'knew', 'know', 'krankenhäus
ern', 'la', 'labelled', 'lawyers', 'life', 'ma', 'makes', 'masks', 'mesmo
', 'mlizza_', 'mochievous', 'must', 'my', 'nmmx2y0m98', 'normalstation',
'not', 'nuernberg_de', 'não', 'nürnberg', 'of', 'opened', 'or', 'over', '
para', 'patiente', 'patienten', 'people', 'peoples', 'personally', 'physo
rg_com', 'picture', 'polliforero', 'positivo', 'presence', 'pushed', 'que
', 'really', 'represented', 'represents', 'restaurant', 'revela', 'rt', '
salud', 'santiago', 'seen', 'shit', 'should', 'show', 'so', 'solicitud',
'some', 'somebody', 'st', 'state', 'states', 'still', 'study', 'séptimo',
'taught', 'testaram', 'that', 'the', 'this', 'to', 'travel', 'try', 'twit
ter', 'united', 'up', 'us', 've', 'virginia', 'who', 'workers', 'workings
', 'you', 'zahlen', 'zé']
```

The result shows the representing words that are included in all the tweets. We can transform this into an array to make this into numerical data. The numerical data simply shows how many times a certain word has appeared in each text (tweet).

```
print(x.toarray())
```

```
[[0 0 0 ... 2 0 0]
 [0 0 0 ... 0 0 0]
 [0 1 0 ... 0 1 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

0 refers to the word not being present in the text, and 1 refers to the word being present in the text. This data can now be used as a feature to be used as an input for machine learning or other analytical purposes.

TF-IDF Vectoriser

Using CountVectorizer is very simple but may be too simple for usage. For example, some words like "the" might appear many times and their counts may not be meaningful in the encoded vectors. Therefore, the more popular method of calculating word frequencies is TF-IDF (Term Frequency-Inverse Document Frequency)

- Term frequency: summarises how often a given word appears within a document
- Inverse document frequency: downscales words that appear a lot across documents

Basically, TF-IDF are word frequency scores that try to highlight words that are more interesting.

TfidfVectorizer will tokenise documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. The flow of usage is basically the same as CountVectorizer.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# create the transform
vectorizer = TfidfVectorizer()

# tokenise and build vocab
vectorizer.fit(status_10)
```

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features
=None,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=Non
e,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)
```

We can print out the learnt vocabulary as such:

```
print(vectorizer.vocabulary_)
```

```
{'rt': 112, 'ambrosia_ijebu': 10, 'you': 146, 'cannot': 17, 'curfew': 28,
'state': 124, 'or': 92, 'country': 25, 'indefinitely': 63, 'covid': 26, '
should': 117, 'have': 57, 'taught': 129, 'us': 140, 'all': 9, 'that': 13
1, 'must': 82, 'still': 126, 'show': 118, 'workings': 145, 'mlizza_': 80,
've': 141, 'seen': 115, 'twitter': 137, 'really': 107, 'change': 19, 'peo
ples': 98, 'life': 75, 'so': 119, 'gonna': 55, 'give': 54, 'it': 65, 'try
': 136, 'my': 83, 'husband': 60, 'opened': 91, 'up': 139, 'this': 133, 'r
estaurant': 110, 'called': 16, 'cocina': 22, 'nuernberg_de': 87, 'die': 3
4, 'aktuellen': 8, 'zahlen': 147, 'aus': 14, 'den': 33, 'krankenhäusern':
71, 'in': 62, 'nürnberg': 89, 'covid19': 27, 'patienten': 96, 'normalstat
ion': 85, '35': 3, '19': 1, 'patiente': 95, 'finallevel': 46, 'fyi': 52,
'somebody': 122, 'else': 40, 'know': 70, 'just': 67, 'died': 35, 'from':
50, 'makes': 77, 'people': 97, 'dead': 30, 'knew': 69, 'personally': 99,
'shit': 116, 'is': 64, 'not': 86, 'over': 93, 'st': 123, 'fruitloopbirb':
51, 'each': 37, 'chair': 18, 'picture': 101, 'represents': 109, '10': 0,
'who': 143, 'the': 132, 'united': 138, 'states': 125, 'if': 61, 'represen
ted': 108, 'zé': 148, 'felipe': 45, 'revela': 111, 'que': 106, 'ele': 39,
'virginia': 142, 'fonseca': 48, 'testaram': 130, 'positivo': 103, 'para':
94, 'https': 59, 'co': 21, '2rz7twe0nl': 2, 'polliforero': 102, 'ahora':
7, 'el': 38, 'séptimo': 128, 'juzgado': 68, 'de': 29, 'garantía': 53, 'sa
ntiago': 114, 'accedió': 5, 'la': 72, 'solicitud': 120, 'defensa': 31, 'd
el': 32, 'exministro': 43, 'salud': 113, 'jaime': 66, 'ma': 76, 'acho':
6, 'não': 88, 'estou': 42, 'mesmo': 79, 'com': 23, 'physorg_com': 100, 'h
ow': 58, 'cough': 24, 'clouds': 20, 'travel': 135, 'presence': 104, 'and
': 12, 'absence': 4, 'of': 90, 'face': 44, 'masks': 78, 'study': 127, 'nm
mx2y0m98': 84, 'mochievous': 81, 'during': 36, 'some': 121, 'folks': 47,
'pushed': 105, 'for': 49, 'lawyers': 74, 'to': 134, 'be': 15, 'labelled':
73, 'as': 13, 'essential': 41, 'workers': 144, 'amp': 11, 'happened': 56}
```

The vocabulary simply shows the unique integer index for each word in the document.

```
print(vectorizer.idf_)
```

```
[2.70474809 1.78845736 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.29928298 2.70474809 2.70474809
 2.70474809 2.70474809 1.2006707  2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.29928298
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.29928298 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.29928298
 2.70474809 2.70474809 1.78845736 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.29928298 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.29928298 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 1.2006707  2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.29928298
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.29928298
 2.01160091 1.78845736 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.29928298 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809]
```

The inverse document frequencies are printed above. We can now transform the entire document that we have prepared using TF-IDF method.

```
vector = vectorizer.transform([status_10[0]])
```

```
print(vector.shape)
```

```
(1, 149)
```

This document is now encoded as a 149-element sparse array. When we convert this to an array:

```
print(vector.toarray())
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.21905699 0.21905699 0.
  0.         0.         0.         0.         0.         0.21905699
  0.         0.         0.         0.         0.         0.
  0.         0.21905699 0.09724207 0.         0.21905699 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.21905699 0.         0.
  0.         0.         0.         0.21905699 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.21905699 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.21905699 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.09724207 0.
  0.         0.         0.         0.21905699 0.21905699 0.
  0.         0.         0.         0.         0.21905699 0.
  0.21905699 0.         0.         0.21905699 0.         0.18621845
  0.         0.         0.         0.         0.         0.
  0.         0.         0.18621845 0.         0.         0.
  0.         0.21905699 0.43811398 0.         0.        ]]
```

We get the TF-IDF score for each word that is normalised to values between 0 and 1. This encoded document vector can now be used directly as features for most of machine learning problems.

Word Cloud

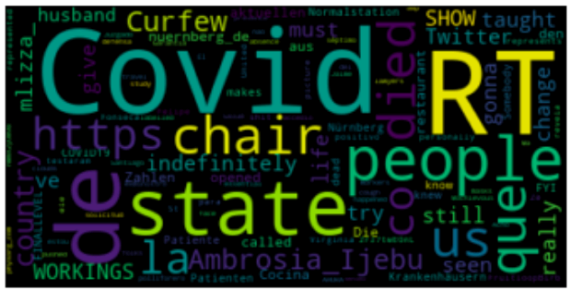
We can also visualise the commonly appearing terms in the document. Using the latest 10 tweets that we have prepared, let us create a Word Cloud.

```
# combine all strings in the list into one string
status_text = ', '.join(status_10)
```

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

wordcloud = WordCloud().generate(status_text)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



With bigger text document, we can get more interesting word clouds. You can explore more with bigger dataset on your own to gather more interesting insights.

6. Accessing User Information

So far, we have learnt how to find out the trending topics, how to use query terms to gather tweets, and how to use those tweets gathered to gain insights. There is another method of gathering tweets and useful information: accessing user information.

By accessing user information, we can gain useful information such as:

1. Location
2. Follower count
3. Date of creation
4. Number of tweets

And many more.

Getting User Information

Using **users.lookup(screen_name=),** we can access a user's information. In the argument "screen_name", we have to input the Twitter ID name that we want to access. The Twitter ID name is the name in the form of "@SocialWebMining", where the @ signifies that it is an ID. The output is a nested dictionary in a list, requiring careful navigation.

As an example, let us try to access Elon Musk's Twitter account. Elon Musk's Twitter ID is @elonmusk.


```
account = twitter_api.users.lookup(screen_name='elonmusk')
```

```
print(account)
```

```
[{'id': 44196397, 'id_str': '44196397', 'name': 'Elon Musk', 'screen_name
': 'elonmusk', 'location': '', 'description': '', 'url': None, 'entities
': {'description': {'urls': []}}, 'protected': False, 'followers_count':
39448993, 'friends_count': 97, 'listed_count': 56489, 'created_at': 'Tue
Jun 02 20:12:29 +0000 2009', 'favourites_count': 6929, 'utc_offset': Non
e, 'time_zone': None, 'geo_enabled': False, 'verified': True, 'statuses_c
ount': 12639, 'lang': None, 'status': {'created_at': 'Mon Oct 19 04:05:54
+0000 2020', 'id': 1318040653658423296, 'id_str': '1318040653658423296',
```

We can now extract necessary information into variables of our choice. For example, let us extract the **name**, **description,** and **followers_count.**

```
account_name = account[0]['name']

account_description = account[0]['description']

account_followers = account[0]['followers_count']
```

```
print(account_name)
print(account_description)
print(account_followers)
```

```
Elon Musk

39448993
```

Unfortunately for Elon Musk, there is no description available. However, using these methods, we can now create more features should we need this information.

Getting User's Tweets

Getting a user's information is good, but it would be much better to get the tweets of a user. Fortunately, the Twitter API provides such function:
statuses.user_timeline(screen_name= , count=)

```
user_tweets = twitter_api.statuses.user_timeline(screen_name='elonmusk', count=100)
```

We have gathered latest 100 tweets of Elon Musk. In a list, there are 100 dictionaries, each dictionary containing the tweet and its corresponding information. We can do similar methods to extract the necessary information. We will now try to extract 10 tweets of Elon Musk.

We will first try to navigate through the data structure and print out the latest 10 tweets.

```
for i in range(10):
    print()
    print(user_tweets[i]['text'])
```

```
@PPathole @toadmeister Yes. We also have to consider population life-mont
hs lost from lockdowns &amp;amp; other restriction… https://t.co/0Ct2sVAngy

@toadmeister Sweden at zero deaths Oct 15 https://t.co/Gy9a20uMmX

@JoshTownsend_96 Yes

@juanjacobs @jwangARK Firmware is probably a slightly more accurate descr
iption, but yes

RT @Tesla: There's no place like home..especially if it has a Solar Roof.

This Kansas ranch got theirs installed in 4 days.

@PPathole @flyerandyp @mojosusan Mobi-C has too much mobility, fusion has
too little. We need Mobi-D haha.

@flyerandyp @mojosusan Actually, it’s my neck … https://t.co/2vCoAxsrV3

@flyerandyp @mojosusan Maybe I can be helpful. Had Mobi-C disc put in at
C5-C6, however 1st surgery failed to remov… https://t.co/Czykfyo0IX

@jwangARK Pretty accurate. It’s not so much DNA sequencing as it is using
synthetic DNA/RNA to fix bugs in our code… https://t.co/y6GcySVGhM

@SamTalksTesla London to Beijing would be cool
```

Now, we will save all of Elon Musk's tweets into a variable since we know now how to navigate through the data structure. Furthermore, we will only use 10 tweets, same as before, for computational ease.

```
user_10
```

```
['@PPathole @toadmeister Yes. We also have to consider population life-mo
nths lost from lockdowns &amp;amp; other restriction… https://t.co/0Ct2sVAngy
',
 '@toadmeister Sweden at zero deaths Oct 15 https://t.co/Gy9a20uMmX',
 '@JoshTownsend_96 Yes',
 '@juanjacobs @jwangARK Firmware is probably a slightly more accurate des
cription, but yes',
 "RT @Tesla: There's no place like home..especially if it has a Solar Roo
f.\n\nThis Kansas ranch got theirs installed in 4 days.",
 '@PPathole @flyerandyp @mojosusan Mobi-C has too much mobility, fusion h
as too little. We need Mobi-D haha.',
 '@flyerandyp @mojosusan Actually, it’s my neck … https://t.co/2vCoAxsrV3
',
 '@flyerandyp @mojosusan Maybe I can be helpful. Had Mobi-C disc put in a
t C5-C6, however 1st surgery failed to remov… https://t.co/Czykfyo0IX',
 '@jwangARK Pretty accurate. It’s not so much DNA sequencing as it is usi
ng synthetic DNA/RNA to fix bugs in our code… https://t.co/y6GcySVGhM',
 '@SamTalksTesla London to Beijing would be cool']
```

Vectoriser with User Tweets

Same as before, we can try vectorising Elon Musk's tweets using Count Vectoriser, TF-IDF Vectoriser, and visualising through Word Cloud.

CountVectorizer:

```
vectorizer = CountVectorizer()
x = vectorizer.fit_transform(user_10)
print(vectorizer.get_feature_names())
```

```
['0ct2svangy', '15', '1st', '2vcoaxsrv3', 'accurate', 'actually', 'also',
'amp', 'as', 'at', 'be', 'beijing', 'bugs', 'but', 'c5', 'c6', 'can', 'co
', 'code', 'consider', 'cool', 'czykfyo0ix', 'days', 'deaths', 'descripti
on', 'disc', 'dna', 'especially', 'failed', 'firmware', 'fix', 'flyerandy
p', 'from', 'fusion', 'got', 'gy9a20ummx', 'had', 'haha', 'has', 'have',
'helpful', 'home', 'however', 'https', 'if', 'in', 'installed', 'is', 'it
', 'joshtownsend_96', 'juanjacobs', 'jwangark', 'kansas', 'life', 'like',
'little', 'lockdowns', 'london', 'lost', 'maybe', 'mobi', 'mobility', 'mo
josusan', 'months', 'more', 'much', 'my', 'neck', 'need', 'no', 'not', 'o
ct', 'other', 'our', 'place', 'population', 'ppathole', 'pretty', 'probab
ly', 'put', 'ranch', 'remov', 'restriction', 'rna', 'roof', 'rt', 'samtal
kstesla', 'sequencing', 'slightly', 'so', 'solar', 'surgery', 'sweden', '
synthetic', 'tesla', 'theirs', 'there', 'this', 'to', 'toadmeister', 'too
', 'using', 'we', 'would', 'y6gcysvghm', 'yes', 'zero']
```

```
print(x.toarray())
```

```
[[1 0 0 ... 0 1 0]
 [0 1 0 ... 0 0 1]
 [0 0 0 ... 0 1 0]
 ...
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 0 0]]
```

TfidfVectorizer:

```
vectorizer = TfidfVectorizer()

vectorizer.fit(user_10)

print(vectorizer.vocabulary_)
```

```
{'ppathole': 76, 'toadmeister': 99, 'yes': 105, 'we': 102, 'also': 6, 'ha
ve': 39, 'to': 98, 'consider': 19, 'population': 75, 'life': 53, 'months
': 63, 'lost': 58, 'from': 32, 'lockdowns': 56, 'amp': 7, 'other': 72, 'r
estriction': 82, 'https': 43, 'co': 17, '0ct2svangy': 0, 'sweden': 92, 'a
t': 9, 'zero': 106, 'deaths': 23, 'oct': 71, '15': 1, 'gy9a20ummx': 35, '
joshtownsend_96': 49, 'juanjacobs': 50, 'jwangark': 51, 'firmware': 29, '
is': 47, 'probably': 78, 'slightly': 88, 'more': 64, 'accurate': 4, 'desc
ription': 24, 'but': 13, 'rt': 85, 'tesla': 94, 'there': 96, 'no': 69, 'p
lace': 74, 'like': 54, 'home': 41, 'especially': 27, 'if': 44, 'it': 48,
'has': 38, 'solar': 90, 'roof': 84, 'this': 97, 'kansas': 52, 'ranch': 8
0, 'got': 34, 'theirs': 95, 'installed': 46, 'in': 45, 'days': 22, 'flyer
andyp': 31, 'mojosusan': 62, 'mobi': 60, 'too': 100, 'much': 65, 'mobilit
y': 61, 'fusion': 33, 'little': 55, 'need': 68, 'haha': 37, 'actually':
5, 'my': 66, 'neck': 67, '2vcoaxsrv3': 3, 'maybe': 59, 'can': 16, 'be': 1
0, 'helpful': 40, 'had': 36, 'disc': 25, 'put': 79, 'c5': 14, 'c6': 15, '
however': 42, '1st': 2, 'surgery': 91, 'failed': 28, 'remov': 81, 'czykfy
o0ix': 21, 'pretty': 77, 'not': 70, 'so': 89, 'dna': 26, 'sequencing': 8
7, 'as': 8, 'using': 101, 'synthetic': 93, 'rna': 83, 'fix': 30, 'bugs':
12, 'our': 73, 'code': 18, 'y6gcysvghm': 104, 'samtalkstesla': 86, 'londo
n': 57, 'beijing': 11, 'would': 103, 'cool': 20}
```

```
print(vectorizer.idf_)
```

```
[2.70474809 2.70474809 2.70474809 2.70474809 2.29928298 2.70474809
 2.70474809 2.70474809 2.70474809 2.29928298 2.29928298 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 1.6061358
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.01160091 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.29928298 2.70474809 2.70474809 2.70474809
 2.70474809 1.6061358  2.70474809 2.01160091 2.70474809 2.29928298
 2.01160091 2.70474809 2.70474809 2.29928298 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.29928298 2.70474809 2.01160091 2.70474809 2.70474809 2.29928298
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.29928298 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809 2.70474809
 2.70474809 2.70474809 1.78845736 2.29928298 2.70474809 2.70474809
 2.29928298 2.70474809 2.70474809 2.01160091 2.70474809]
```

```
print(vector.toarray())
```

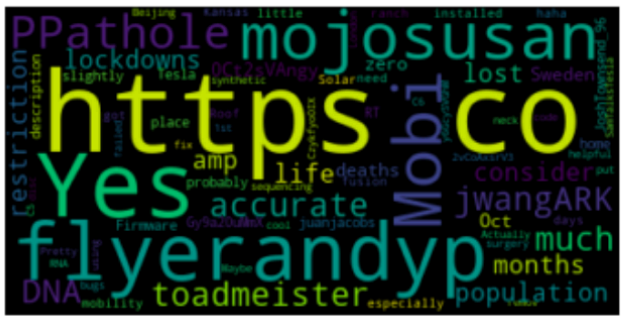
```
[[0.24351474 0.         0.         0.         0.         0.
  0.24351474 0.24351474 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.14460413
  0.         0.24351474 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.24351474 0.         0.         0.
  0.         0.         0.         0.24351474 0.         0.
  0.         0.14460413 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.24351474
  0.         0.         0.24351474 0.         0.24351474 0.
  0.         0.         0.         0.24351474 0.         0.
  0.         0.         0.         0.         0.         0.
  0.24351474 0.         0.         0.24351474 0.20700977 0.
  0.         0.         0.         0.         0.24351474 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.16101896 0.20700977 0.         0.
  0.20700977 0.         0.         0.18110909 0.        ]]
```

Word Cloud

```
user_combined = ', '.join(user_10)
```

```
wordcloud = WordCloud().generate(user_combined)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



Therefore, now we have taken a look at some simplistic methods in exploring Twitter and gathering tweets. There are many more complex and advanced methods that can be used that requires more programming and API understanding and expertise.

7. Exercise:

- **Gather Twitter data regarding a topic of your own interest and perform the same exploration as the guide.**
- **Gather tweets of 5 Twitter users and perform the same exploration as the guide.**