# soc128d_notebook_7_webscraping_and_APIs

July 21, 2021

**Sociology 128D: Mining Culture Through Text Data: Introduction to Social Data Science**

## 1 Notebook 7: Web Scraping and APIs

Web scraping is a big topic. There are a lot of reasons someone might want to scrape web content, but the reason applicable to this class is to get data that may be useful for answering questions about some social phenomena.

People who provide web content are typically savvy to the existence of tools for web scraping. You can often find references to automated web scraping in a site's Terms of Use (or equivalent), which often prohibit automated scraping.

I'll just make two points here. First, the desirability of the data on a site is probably positively correlated with how prohibitive it is to scrape it. Second, we should try to be clear about what we mean by "web scraping."

Regarding the second point, we are typically referring to accessing a website's content in a way that's mediated by a tool or set of tools that makes it qualitatively different from browsing the web normally. As we'll see in our first example using the `requests` library, this can be as simple as using a line of Python code to store a web search in memory, rather than rendering it directly in a browser. We can then view what we've scraped (e.g., rendered HTML), which wouldn't be much different from normal browsing. We could also save it, or save some feature or set of features we've extracted from it; and doing this a lot is typically where things become problematic.

At the most basic level, repeatedly scraping a site (or some part of it) means making repeated requests of the site's servers. That can be a problem in itself. The first point above just adds to this: sites may also want to protect their data, and may make it available subject to terms that prohibit automated scraping. Content is also served in different ways. Static websites are much easier to scrape than dynamic ones, which require a different approach.

One compromise many sites make is to offer an application programming interface (API). In this notebook, we're going to keep our focus on getting data that may be useful for answering social research questions. Toward that end, we'll explore scraping static web content with an eye toward getting Twitter user handles for members of the US senate, and we'll then use those handles to get tweets. Finally, we'll use an API to access data from Reddit.

### 1.1 Setup

For this notebook, you'll need to install `beautifulsoup4`, `psaw`, `nest_asyncio`, and `twint`.

If you use Anaconda, you can install `beautifulsoup4` and `async_io` by running the following lines in the Anaconda interpreter:

```
conda install -c anaconda beautifulsoup4
conda install -c conda-forge nest-asyncio
```

Otherwise, you install them using pip. (Depending on your setup, you may need to use pip3 instead.)

```
pip3 install beautifulsoup4
pip3 install nest_asyncio
```

Regardless, you will need to install `psaw` using pip:

```
pip3 install psaw
```

And you will need to install `twint` by executing the following commands from the command line (e.g., the Anaconda interpreter):

```
git clone --depth=1 https://github.com/twintproject/twint.git
cd twint
pip3 install . -r requirements.txt
```

[1]:
```
import datetime as dt
import nest_asyncio
import pandas as pd
import requests
import time
import twint

from bs4 import BeautifulSoup
from IPython.core.display import display, HTML
from psaw import PushshiftAPI

nest_asyncio.apply()
```

## 1.2 Web Scraping with Requests and BeautifulSoup

### 1.2.1 Example 1. Rendering Search Results inside Jupyter

At its most basic level, "scraping the web" is just using a computer to access web content in a different way. The next two cells show how we can use the `requests` library to store the results of a web search in memory (in a variable we'll call results), which we can then render inside the notebook.

We'll use `requests.get()` to get the web content we want to examine. The `requests` library enables us to make HTTP requests, even with authentication.

Running the second cell may change the way the notebook is displayed. You can comment it out and run the cell again if needed.

[2]:
```
url = "https://www.google.com/search?q=weather+stanford"
results = requests.get(url)
```

[3]:
```
# display(HTML(results.text))
```

### 1.2.2 Example 2. Scraping Quotes from a Scraping Sandbox

To get a sense of how scraping static content works, we'll start with a sandbox designed for this purpose. https://toscrape.com/ offers a couple of environments, including a fictional bookstore. Since this is a class on text analysis, we're going to take a look at another page, which displays quotes.

[4]:
```
url = "https://quotes.toscrape.com/"
quotes_page = requests.get(url)
```

[5]:
```
quotes_page.json
```

[5]:
```
<bound method Response.json of <Response [200]>>
```

The first thing to note is that we can interact with the result like it's a string. If you type "quotes_page." (ending with a period) and press the `tab` key, Jupyter will list several attributes you can explore, like the status code and headers.

[6]:
```
print(quotes_page.text[:500])
```

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Quotes to Scrape</title>
    <link rel="stylesheet" href="/static/bootstrap.min.css">
    <link rel="stylesheet" href="/static/main.css">
</head>
<body>
    <div class="container">
        <div class="row header-box">
            <div class="col-md-8">
                <h1>
                    <a href="/" style="text-decoration: none">Quotes to
Scrape</a>
                </h1>
            </div>
            <div class="col-md
```

[7]:
```
quotes_page.status_code
```

[7]:
```
200
```

[8]:
```
quotes_page.headers
```

[8]:
```
{'Server': 'nginx/1.17.7', 'Date': 'Thu, 22 Jul 2021 01:42:01 GMT', 'Content-
Type': 'text/html; charset=utf-8', 'Transfer-Encoding': 'chunked', 'Connection':
```

```
'keep-alive', 'Vary': 'Accept-Encoding', 'Strict-Transport-Security': 'max-
age=15724800; includeSubDomains', 'Content-Encoding': 'gzip'}
```

We'll use Beautiful Soup to parse the text and find the content we are interested in.

[9]:
```
soup = BeautifulSoup(quotes_page.text, "html.parser")
```

[10]:
```
type(soup)
```

[10]:
```
bs4.BeautifulSoup
```

[11]:
```
print(soup.prettify()[:500])
```

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   Quotes to Scrape
  </title>
  <link href="/static/bootstrap.min.css" rel="stylesheet"/>
  <link href="/static/main.css" rel="stylesheet"/>
 </head>
 <body>
  <div class="container">
   <div class="row header-box">
    <div class="col-md-8">
     <h1>
      <a href="/" style="text-decoration: none">
       Quotes to Scrape
      </a>
     </h1>
    </div>
    <div class="col-md-4">
     <p>
      <a href="/login">
```

We can now search the soup for all kinds of content. If you type "soup." (ending with a period) in a Code cell and press the `tab` key, Jupyter will show different attributes or methods that are available.

[12]:
```
soup.h1
```

[12]:
```
<h1>
<a href="/" style="text-decoration: none">Quotes to Scrape</a>
</h1>
```

[13]:
```
soup.p
```

```
[13]: <p>
      <a href="/login">Login</a>
      </p>
```

```
[14]: soup.a
```

```
[14]: <a href="/" style="text-decoration: none">Quotes to Scrape</a>
```

```
[56]: soup.find_all("a")[:5]
```

```
[56]: [<a class="alert-info" href="#s-lg-guide-main" id="s-lg-public-skiplink">Skip to
      main content</a>,
       <a class="title-header title-header-large" href="https://library.ucsd.edu/">The
      Library</a>,
       <a class="title-logo" href="https://www.ucsd.edu/">
       <img alt="UC San Diego"
      src="https://library.ucsd.edu/assets/libapps/shared/logo-ucsd-header.png"/>
       </a>,
       <a href="https://library.ucsd.edu/research-and-collections/index.html">Research
      &amp; Collections</a>,
       <a href="https://library.ucsd.edu/borrow-and-request/index.html">Borrow &amp;
      Request</a>]
```

Here we print one `div` section (a chunk of the HTML) that shows a single quote and the author.

```
[16]: print(soup.prettify()[600:1538])
```

```
     <div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
      <span class="text" itemprop="text">
       "The world as we have created it is a process of our thinking. It cannot
be changed without changing our thinking."
      </span>
      <span>
       by
       <small class="author" itemprop="author">
        Albert Einstein
       </small>
       <a href="/author/Albert-Einstein">
        (about)
       </a>
      </span>
      <div class="tags">
       Tags:
       <meta class="keywords" content="change,deep-thoughts,thinking,world"
itemprop="keywords"/>
       <a class="tag" href="/tag/change/page/1/">
        change
       </a>
       <a class="tag" href="/tag/deep-thoughts/page/1/">
```

```
    deep-thoughts
   </a>
   <a class="tag" href="/tag/thinking/page/1/">
    thinking
   </a>
   <a class="tag" href="/tag/world/page/1/">
    world
   </a>
  </div>
 </div>
```

The `.find_all()` method can be used for various types of content. Here we use it to get all of the `div` tags containing quotes. We then use `.find_all()` on each result to find the `span` tags nested inside. We use Python's `str.replace()` method to get rid of some unwanted text and print the results.

[17]:
```
for thing1 in soup.find_all(class_="quote"):
    for span in thing1.find_all("span"):
        print(span.text.replace("(about)", ""))
```

```
"The world as we have created it is a process of our thinking. It cannot be
changed without changing our thinking."
by Albert Einstein



"It is our choices, Harry, that show what we truly are, far more than our
abilities."
by J.K. Rowling



"There are only two ways to live your life. One is as though nothing is a
miracle. The other is as though everything is a miracle."
by Albert Einstein



"The person, be it gentleman or lady, who has not pleasure in a good novel, must
be intolerably stupid."
by Jane Austen



"Imperfection is beauty, madness is genius and it's better to be absolutely
ridiculous than absolutely boring."
by Marilyn Monroe



"Try not to become a man of success. Rather become a man of value."
by Albert Einstein
```

```
"It is better to be hated for what you are than to be loved for what you are
not."
by André Gide


"I have not failed. I've just found 10,000 ways that won't work."
by Thomas A. Edison


"A woman is like a tea bag; you never know how strong it is until it's in hot
water."
by Eleanor Roosevelt


"A day without sunshine is like, you know, night."
by Steve Martin
```

### 1.2.3 Example 3. Something Useful: Identifying Twitter Handles of Members of the Senate

As we've noted, at its most basic level scraping is just accessing a site. Here we will scrape a "real" website–but we are only going to make *one* request. Specifically, we'll get the Twitter handles (along with state and party) of each current US senator from a site maintained by the UC San Diego Library.

```
[18]: url = "https://ucsd.libguides.com/congress_twitter/senators"
```

```
[19]: senate_page = requests.get(url)
```

```
[20]: # print(senate_page.text)
```

```
[21]: soup = BeautifulSoup(senate_page.text, "html.parser")
```

You can compare the way the HTML is printed when using `.prettify()` on soup to printing the text from the original result from `requests`.

```
[22]: # print(soup.prettify())
```

If you explore the site in a browser or just scroll through the soup, you can see that the names, states, parties, and Twitter handles of the senators are arranged in a table, which is convenient for us. We'll use `.find_all()` to identify the table.

```
[23]: len(soup.find_all("table"))
```

```
[23]: 2
```

[24]:
```python
tables = soup.find_all("table")
for table in tables:
    print(type(table), len(table))
```

```
<class 'bs4.element.Tag'> 3
<class 'bs4.element.Tag'> 3
```

We can also see that the info we want is inside `tr` tags, which are rows.

[25]:
```python
print(str(tables[0])[:1000])
```

```
<table class="table table-bordered table-striped table-hover table-condensed"
style="border: 1px solid rgb(221, 221, 221);">
<tbody>
<tr>
<td class="ck_border" style="border: 1px solid rgb(221, 221,
221);"><strong>Senator</strong></td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;"><strong>State</strong></td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;"><strong>Party</strong></td>
</tr>
<tr>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221);"><a
href="https://twitter.com/SenatorBaldwin">Baldwin, Tammy</a></td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;">WI</td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;">D</td>
</tr>
<tr>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221);"><a
href="https://twitter.com/SenJohnBarrasso">Barrasso, John</a></td>
<td class="ck_border" style="border: 1px sol
```

The information we want for each senator (name, handle, state, and party) is contained in one row. The handle is in the URL of the `a` tag, while the senator's name is in the text of that tag. The state and party are in additional `td` tags.

[26]:
```python
tables[0].findAll("tr")[1]
```

[26]:
```
<tr>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221);"><a
href="https://twitter.com/SenatorBaldwin">Baldwin, Tammy</a></td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;">WI</td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;">D</td>
</tr>
```

Here we use `enumerate()` with a for loop just to look at the first few results.

This code finds all of the `tr` tags, ignores any without a link (e.g., to a Twitter account), finds all of the elements of the `ck_border` class, and prints the text. This prints the senator's name, state, and party. The `a` tag's attributes are like a dictionary, and the value for the key "href" is the URL to the senator's Twitter.

[27]:
```
for i, result in enumerate(soup.find_all("tr")):
    if i < 4:
        if result.a:
            for element in result.find_all(class_="ck_border"):
                print(element.text)
            print(result.a.attrs["href"])
        print()
```

```
Baldwin, Tammy
WI
D
https://twitter.com/SenatorBaldwin

Barrasso, John
WY
R
https://twitter.com/SenJohnBarrasso

Bennet, Michael
CO
D
https://twitter.com/SenatorBennet
```

Now that we have figured out the way the information is structured, we will extract the name, state, party, and Twitter handle for each US senator. We'll create an empty list called senator_data to store the data initially. We'll use a nested for loop just like the one above, for we'll append each senator's name, state, party, and handle to a list called row before appending that row–one per senator–to senator_data.

[28]:
```
senator_data = []

for result in soup.find_all("tr"):
    if result.a:
        row = []
        for element in result.find_all(class_="ck_border"):
            row.append(element.text)
        handle = result.a.attrs["href"]
        handle = handle.replace("https://twitter.com/", "")
        row.append(handle)
        senator_data.append(row)
    else:
        print(result) # show the rows that aren't added to the dataset we're␣
↪making
```

```
<tr>
<td class="ck_border" style="border: 1px solid rgb(221, 221,
221);"><strong>Senator</strong></td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;"><strong>State</strong></td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;"><strong>Party</strong></td>
</tr>
<tr>
<td class="ck_border" style="border: 1px solid rgb(221, 221,
221);"><strong>Senator</strong></td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;"><strong>State</strong></td>
<td class="ck_border" style="border: 1px solid rgb(221, 221, 221); text-align:
center;"><strong>Party</strong></td>
</tr>
```

[29]:
```
senator_data[:5]
```

[29]:
```
[['Baldwin, Tammy', 'WI', 'D', 'SenatorBaldwin'],
 ['Barrasso, John', 'WY', 'R', 'SenJohnBarrasso'],
 ['Bennet, Michael', 'CO', 'D', 'SenatorBennet'],
 ['Blackburn, Marsha', 'TN', 'R', 'MarshaBlackburn'],
 ['Blumenthal, Richard', 'CT', 'D', 'SenBlumenthal']]
```

[30]:
```
len(senator_data)
```

[30]:
```
100
```

Now we will create a pandas dataframe from this list of lists. The `columns` argument lets us name the columns in the resulting dataframe.

[31]:
```
df = pd.DataFrame(senator_data, columns=["senator", "state", "party",␣
↪"twitter_handle"])
```

[32]:
```
df.shape
```

[32]:
```
(100, 4)
```

[33]:
```
df.head()
```

[33]:
```
               senator state party   twitter_handle
0       Baldwin, Tammy    WI     D   SenatorBaldwin
1       Barrasso, John    WY     R  SenJohnBarrasso
2      Bennet, Michael    CO     D    SenatorBennet
```

```
3    Blackburn, Marsha    TN      R  MarshaBlackburn
4  Blumenthal, Richard    CT      D    SenBlumenthal
```

[34]:
```
df.tail()
```

[34]:
```
                 senator state party twitter_handle
95    Warren, Elizabeth    MA     D       SenWarren
96  Whitehouse, Sheldon    RI     D  SenWhitehouse
97        Wicker, Roger    MS     R  SenatorWicker
98           Wyden, Ron    OR     D       RonWyden
99          Young, Todd    IN     R   SenToddYoung
```

[35]:
```
df.to_csv("senate_twitter_dataframe.csv", index=None)
```

### 1.3 Scraping Tweets using `twint`

`twint` describes itself as "an advanced Twitter scraping tool written in Python that allows for scraping Tweets from Twitter profiles without using Twitter's API." `twint` has been featured in plenty of guides to scraping tweets, but there seem to be issues such as the way it handles dates, among other problems. One workaround is to handle some of the configuration in the search string itself using Twitter's search operators, rather than configuring `twint` as intended.

You can see Twitter's standard search operators here.

Here are some helpful thoughts about using (and the limitations of) location data, including tips for finding geocodes and some examples of searching for tweets from particular events.

**Note:** I recommend applying for a Twitter developer account and accessing tweets through the official API. We will use `twint` for this example, but I do not recommend violating Twitter's terms by accessing excessive amounts of data (etc.). I've set the tweet limits low for this notebook for a reason.

First, we'll look at tweets from US senators around April 28, when President Biden addressed a joint session of Congress. Next, we'll look at geotagged tweets.

#### 1.3.1 Example 1. Tweets from US Senators

We'll use the dataframe we created in the previous section to identify the twitter handles of current US senators.

[36]:
```
df = pd.read_csv("senate_twitter_dataframe.csv")
```

[37]:
```
c = twint.Config()
c.Hide_output = True
c.Store_csv = True
c.Output = "senate_tweets.csv"
c.Limit = 10
```

[57]:
```
run_twint = input("Scrape twitter data? ")
```

```
if run_twint in ["yes", "y"]:
    for handle in df.twitter_handle.values:
        searchstr = f"from:{handle} until:2021-04-29 since:2021-04-28"
        c.Search = searchstr
        twint.run.Search(c)
        time.sleep(1)
```

[39]:
```
tweets_df = pd.read_csv("senate_tweets.csv")
```

[40]:
```
tweets_df.date.min(), tweets_df.date.max(), tweets_df.shape
```

[40]:
```
('2021-04-27', '2021-04-28', (379, 36))
```

[41]:
```
tweets_df.head()
```

[41]:
```
                    id      conversation_id  \
0  1387480099763757056  1387480099763757056
1  1387458980025446406  1387458980025446406
2  1387443098502975490  1387443098502975490
3  1387524261951295490  1387524261951295490
4  1387508940645228546  1387508940645228546

                               created_at        date      time  timezone  \
0  2021-04-28 11:53:28 Pacific Daylight Time  2021-04-28  11:53:28      -700
1  2021-04-28 10:29:33 Pacific Daylight Time  2021-04-28  10:29:33      -700
2  2021-04-28 09:26:26 Pacific Daylight Time  2021-04-28  09:26:26      -700
3  2021-04-28 14:48:57 Pacific Daylight Time  2021-04-28  14:48:57      -700
4  2021-04-28 13:48:04 Pacific Daylight Time  2021-04-28  13:48:04      -700

      user_id         username                name  place  … geo source  \
0  1074518754  senatorbaldwin  Sen. Tammy Baldwin    NaN  … NaN    NaN
1  1074518754  senatorbaldwin  Sen. Tammy Baldwin    NaN  … NaN    NaN
2  1074518754  senatorbaldwin  Sen. Tammy Baldwin    NaN  … NaN    NaN
3   202206694  senjohnbarrasso  Sen. John Barrasso    NaN  … NaN    NaN
4   202206694  senjohnbarrasso  Sen. John Barrasso    NaN  … NaN    NaN

  user_rt_id user_rt retweet_id  reply_to  retweet_date  translate trans_src  \
0        NaN     NaN        NaN        []           NaN        NaN       NaN
1        NaN     NaN        NaN        []           NaN        NaN       NaN
2        NaN     NaN        NaN        []           NaN        NaN       NaN
3        NaN     NaN        NaN        []           NaN        NaN       NaN
4        NaN     NaN        NaN        []           NaN        NaN       NaN

  trans_dest
0        NaN
1        NaN
2        NaN
```

```
3         NaN
4         NaN

[5 rows x 36 columns]
```

[42]:
```
tweets_df[["username", "name", "tweet", "likes_count"]].sample(10)
```

[42]:
```
            username                       name  \
281    senjackyrosen       Senator Jacky Rosen
33      senatorbraun        Senator Mike Braun
177   senjohnkennedy              John Kennedy
44         sencapito      Shelley Moore Capito
37    sensherrodbrown            Sherrod Brown
108    senatordurbin       Senator Dick Durbin
86        sentedcruz          Senator Ted Cruz
228   senjeffmerkley      Senator Jeff Merkley
123     senfeinstein  Senator Dianne Feinstein
55        senbobcasey         Senator Bob Casey

                                                tweet  likes_count
281  We have to expand broadband access in communit…          421
33   "Sen. Braun has proposed legislation to elimin…            8
177  What I expect Pres. Biden to say tonight:  1. …          695
44   Students shouldn't have to worry about whether…           15
37   This is what paying workers a living wage look…          201
108  Each of my guests has firsthand experience of …          130
86               This is a crisis.  #BidenBorderCrisis         1712
228  Excited to watch @POTUS's joint address with N…          142
123  Reports that the Biden administration will ban…          144
55   There are some powerful &amp; wealthy people i…          179
```

### 1.3.2 Example 2. Geocoded Data

To take a break from politics, we'll look at tweets sent from near Deer District in Milwaukee on July 20 as up to 65,000 fans celebrated the Bucks' NBA title. The `geocode` argument in searchstr includes the longitude, latitude, and radius. This time, we aren't specifying a username/handle, and we aren't including an actual search term.

[43]:
```
c = twint.Config()
c.Hide_output = True
c.Store_csv = True
c.Output = "geo_tweets.csv"
c.Limit = 1000
searchstr = "until:2021-07-21 since:2021-07-19 geocode:43.045110,-87.
 ↪915820,5km" # within 5km of Deer District
c.Search = searchstr
twint.run.Search(c)
```

[44]:
```
geo_df = pd.read_csv("geo_tweets.csv")
```

[45]:
```
geo_df.date.min(), geo_df.date.max(), geo_df.shape
```

[45]:
```
('2021-07-20', '2021-07-20', (1000, 36))
```

[46]:
```
geo_df.head()
```

[46]:
```
                    id      conversation_id  \
0  1417635421417267203  1417522585936568323
1  1417635401251053573  1417635401251053573
2  1417635400454184962  1417635400454184962
3  1417635389548941312  1417635389548941312
4  1417635385665011714  1417635385665011714

                                created_at        date      time  timezone  \
0  2021-07-20 16:59:57 Pacific Daylight Time  2021-07-20  16:59:57      -700
1  2021-07-20 16:59:52 Pacific Daylight Time  2021-07-20  16:59:52      -700
2  2021-07-20 16:59:52 Pacific Daylight Time  2021-07-20  16:59:52      -700
3  2021-07-20 16:59:49 Pacific Daylight Time  2021-07-20  16:59:49      -700
4  2021-07-20 16:59:48 Pacific Daylight Time  2021-07-20  16:59:48      -700

              user_id        username             name place  … geo source  \
0  856597620306968576    tweetiestate      tweetiestate   NaN  … NaN    NaN
1  998242960646049797  foxconnaerials  Foxconn Aerials   NaN  … NaN    NaN
2           146943128      danmolloytv       Dan Molloy   NaN  … NaN    NaN
3           705336188       tinker_pix      FlutterBy   NaN  … NaN    NaN
4           368905822        njanczak7             nate   NaN  … NaN    NaN

  user_rt_id user_rt retweet_id  reply_to  retweet_date  translate trans_src  \
0        NaN     NaN        NaN        []           NaN        NaN       NaN
1        NaN     NaN        NaN        []           NaN        NaN       NaN
2        NaN     NaN        NaN        []           NaN        NaN       NaN
3        NaN     NaN        NaN        []           NaN        NaN       NaN
4        NaN     NaN        NaN        []           NaN        NaN       NaN

  trans_dest
0        NaN
1        NaN
2        NaN
3        NaN
4        NaN

[5 rows x 36 columns]
```

[47]:
```
geo_df[["username", "tweet", "likes_count"]].sample(10)
```

```
[47]:           username                                               tweet  \
      235      mrmillymike  @DrKarateChop So you know you're on the right …
      443     jasonfechner                    #Bucks in…  https://t.co/AbZmvtoOIB
      902       jsarles414                         BUCKS IN SIX FOR THE CULTURE
      847        ctown3721               There are children out in these streets.
      976  spectrumnews1wi  Traffic coming into the city is INSANE! #Game6…
      436    goddessblair8  My old subs are being disappointing and poor. …
      578       chefgleon1  Just had the pleasure of finally meeting State…
      390        bebravent  Director of Sales - Menomonee Falls, WI  https…
      927          mvlii89                      @TALLY4K  https://t.co/AQbRRvA3bs
      185       jeffbricco  @jimmyfk Over 90 minutes before game time. Pac…

           likes_count
      235            1
      443            1
      902            0
      847            1
      976            3
      436            1
      578           72
      390            0
      927            0
      185            3
```

### 1.4 Scraping Reddit Content using psaw

Another amazing resource for social media data is pushshift.io, which archives vast amounts of data and makes it easily accessible. We'll use the `psaw` library to access content from the pushshift.io Reddit API.

For this example, we'll get posts to r/WallStreetBets from the last week of January, 2021. During this time, there was a lot of excitement about the rise of the GameStop stock–and then trading was halted on some platforms, such as Robinhood.

First, create an instance of the `PushShiftAPI()` class.

```
[48]: api = PushshiftAPI()
```

We'll use the helper function get_results() to turn the results we get into a list.

```
[49]: def get_results(subreddit: str, start_epoch, before_epoch, limit=10):
          res = list(api.search_submissions(after=start_epoch,
                                            before=before_epoch,
                                            subreddit=subreddit,
                                            limit=limit))
          return res
```

```
[50]: wsb = []
```

```
year = 2020
month = 1
days = range(24,31)

epochs = []

for day in days:
    start_epoch=int(dt.datetime(year, month, day).timestamp())
    try:
        before_epoch=int(dt.datetime(year, month, day+1).timestamp())
    except:
        before_epoch=int(dt.datetime(year, month+1, 1).timestamp()) # first day␣
 ↪of next month

    epochs.append((start_epoch, before_epoch))
    res = get_results("WallStreetBets", start_epoch, before_epoch)
    wsb.append(res)
    time.sleep(1)
```

[51]:
```
wsb_flat = [post for sublist in wsb for post in sublist] # turn list of lists␣
 ↪into list of posts
```

[52]:
```
wsb_df = pd.DataFrame([post.d_ for post in wsb_flat])
```

[53]:
```
wsb_df.head()
```

[53]:
```
  all_awardings  allow_live_comments                author  \
0            []                False           praisomnisf
1            []                False  WarmingSpiritualism
2            []                False         perfectentry1
3            []                False                RLaG69
4            []                False         cheeseburger-

  author_flair_css_class author_flair_richtext author_flair_text  \
0                   None                    []              None
1                   None                    []              None
2                   None                    []              None
3                   None                    []              None
4                   None                    []              None

  author_flair_type author_fullname  author_patreon_flair  author_premium  \
0              text     t2_3g6gzsv5                 False           False
1              text      t2_cfv4pgt                 False           False
2              text      t2_ngkjp0s                 False           False
3              text     t2_318efk89                 False            True
4              text     t2_15wnnhpe                 False           False
```

```
  … removed_by_category  media_metadata  thumbnail_height  thumbnail_width  \
0 …                 NaN             NaN               NaN              NaN
1 …                 NaN             NaN               NaN              NaN
2 …                 NaN             NaN               NaN              NaN
3 …                 NaN             NaN               NaN              NaN
4 …                 NaN             NaN               NaN              NaN

  post_hint preview media media_embed  secure_media  secure_media_embed
0       NaN     NaN   NaN         NaN           NaN                 NaN
1       NaN     NaN   NaN         NaN           NaN                 NaN
2       NaN     NaN   NaN         NaN           NaN                 NaN
3       NaN     NaN   NaN         NaN           NaN                 NaN
4       NaN     NaN   NaN         NaN           NaN                 NaN

[5 rows x 70 columns]
```

[54]:
```
wsb_df.shape
```

[54]:
```
(70, 70)
```

[55]:
```
wsb_df[["author", "title", "selftext", "score"]]
```

[55]:
```
                   author                                              title  \
0             praisomnisf  The mainstream media is failing me, who do you…
1   WarmingSpiritualism                                          Priced In
2          perfectentry1                  Ebay Earnings After the Bell Tuesday
3                 RLaG69                    Follow the government pump and dump
4          cheeseburger-                  Is Bloomberg always so doom and gloom?
..                    …                                                  …
65            praisomnisf                                  Bears versus Bulls
66           Noahnovanoah                  Oh what I beautiful ride it has been.
67              wsb_itch                  How to get away with insider trading
68           Noahnovanoah                  Oh what a beautiful ride it has been.
69                 LVXSIT                  JPow Networth? I'd eat his ass too

                                              selftext  score
0                                                           1
1                                                           1
2   It's pretty difficult to find a major brand na…      1
3   Does anyone know where those fuckers in upper …      1
4   It seems to me if you purchase a Bloomberg ter…      1
..                                                 …    …
65                                           [removed]      1
66                                                          1
67  Hear me out, first you get insider information…      1
68                                                          1
69  Would this man really tank the economy? Get a …      1
```

```
[70 rows x 4 columns]
```