Web Scraping in Python

DS 6001: Practice and Applications of Data Science

Is Web Scraping Legal?

Web scraping is an exciting tool for collecting data from internet resources, and there are some great tools in Python to make it faster and easier.

Is Web Scraping Legal?

Web scraping is an exciting tool for collecting data from internet resources, and there are some great tools in Python to make it faster and easier.

And web scraping is legal! **For now** . . .

Is Web Scraping Legal?

Web scraping is an exciting tool for collecting data from internet resources, and there are some great tools in Python to make it faster and easier.

And web scraping is legal! **For now** . . .

Right now, there is a court case – HiQ Labs vs. LinkedIn – that will establish whether and under what conditions web scraping is legal.

Is Web Scraping Legal?

Web scraping is an exciting tool for collecting data from internet resources, and there are some great tools in Python to make it faster and easier.

And web scraping is legal! **For now** . . .

Right now, there is a court case – HiQ Labs vs. LinkedIn – that will establish whether and under what conditions web scraping is legal.

HiQ scraped LinkedIn profiles to collect data to build models of employee turnover. LinkedIn issued a cease and desist letter. HiQ sued, and won in lower court. But the case will likely be heard **at the Supreme Court this year**.

Is Web Scraping Ethical?

The legal arguments surrounding HiQ Labs vs. LinkedIn involve data ownership, intellectual property, and anti-trust.

Is Web Scraping Ethical?

The legal arguments surrounding HiQ Labs vs. LinkedIn involve data ownership, intellectual property, and anti-trust.

Beyond the legality, there are ethical issues to consider as well:

Is Web Scraping Ethical?

The legal arguments surrounding HiQ Labs vs. LinkedIn involve data ownership, intellectual property, and anti-trust.

Beyond the legality, there are ethical issues to consider as well:

Many repeated calls (from bots) to a website for the purpose of scraping data might **overwhelm the server**, keeping the website's owners from being able to achieve their purpose for having the website.

Is Web Scraping Ethical?

The legal arguments surrounding HiQ Labs vs. LinkedIn involve data ownership, intellectual property, and anti-trust.

Beyond the legality, there are ethical issues to consider as well:

Many repeated calls (from bots) to a website for the purpose of scraping data might **overwhelm the server**, keeping the website's owners from being able to achieve their purpose for having the website.

Some ethical guidelines:

Is Web Scraping Ethical?

The legal arguments surrounding HiQ Labs vs. LinkedIn involve data ownership, intellectual property, and anti-trust.

Beyond the legality, there are ethical issues to consider as well:

Many repeated calls (from bots) to a website for the purpose of scraping data might **overwhelm the server**, keeping the website's owners from being able to achieve their purpose for having the website.

Some ethical guidelines:

- If a website owner has an API that will provide the same data, always use the API instead.

Is Web Scraping Ethical?

The legal arguments surrounding HiQ Labs vs. LinkedIn involve data ownership, intellectual property, and anti-trust.

Beyond the legality, there are ethical issues to consider as well:

Many repeated calls (from bots) to a website for the purpose of scraping data might **overwhelm the server**, keeping the website's owners from being able to achieve their purpose for having the website.

Some ethical guidelines:

- If a website owner has an API that will provide the same data, always use the API instead.
- Include a user-agent string to provide contact information in case of problems (more on this later).

Is Web Scraping Ethical?

The legal arguments surrounding HiQ Labs vs. LinkedIn involve data ownership, intellectual property, and anti-trust.

Beyond the legality, there are ethical issues to consider as well:

Many repeated calls (from bots) to a website for the purpose of scraping data might **overwhelm the server**, keeping the website's owners from being able to achieve their purpose for having the website.

Some ethical guidelines:

- If a website owner has an API that will provide the same data, always use the API instead.
- Include a user-agent string to provide contact information in case of problems (more on this later).
- Limit the scope and frequency of requests as much as possible.

How Websites Prevent You From Scraping

Every GET request you make sends your IP address to the server of the website you are trying to scrape.

How Websites Prevent You From Scraping

Every GET request you make sends your IP address to the server of the website you are trying to scrape.

If the server registers too many requests from the same IP address, or requests made too quickly, the website might:

- **block** the IP address, or
- require a CAPTCHA.

How Websites Prevent You From Scraping

Every GET request you make sends your IP address to the server of the website you are trying to scrape.

If the server registers too many requests from the same IP address, or requests made too quickly, the website might:

- **block** the IP address, or
- require a CAPTCHA.

Websites might hide the data or API endpoints instead of putting the data into the raw HTML.

How Websites Prevent You From Scraping

Every GET request you make sends your IP address to the server of the website you are trying to scrape.

If the server registers too many requests from the same IP address, or requests made too quickly, the website might:

- **block** the IP address, or
- require a CAPTCHA.

Websites might hide the data or API endpoints instead of putting the data into the raw HTML.

Websites will change their formats frequently to break automated scrapers.

User-Agent Strings

An HTTP header is a parameter that gets sent along with the HTTP request that contains metadata about the request.

User-Agent Strings

An HTTP header is a parameter that gets sent along with the HTTP request that contains metadata about the request.

A user agent header contains contact and identification information about the person making the request.

User-Agent Strings

An HTTP header is a parameter that gets sent along with the HTTP request that contains metadata about the request.

A user agent header contains contact and identification information about the person making the request.

Requests are much more likely to get **blocked** by websites if the request does not specify a header that contains a user agent.

User-Agent Strings

An HTTP header is a parameter that gets sent along with the HTTP request that contains metadata about the request.

A user agent header contains contact and identification information about the person making the request.

Requests are much more likely to get **blocked** by websites if the request does not specify a header that contains a user agent.

A user agent should identify your application, email address, programming language, and platform:

```
headers = {'user-agent': 'Class example (jkropko@virginia.edu)
          (Language=Python 3.8.2; Platform=Mac OSX 10.15.5)'}

r = requests.get("https://spinitron.com/WNRN/",
         headers = headers)
```

Using `BeautifulSoup()`

First download the HTML for a website using the `requests.get()` function, including a user agent string. The raw HTML code contains a series of text fragments that look like this:

```
<tag attribute="value"> Navigable string </tag>
```

Using `BeautifulSoup()`

First download the HTML for a website using the `requests.get()` function, including a user agent string. The raw HTML code contains a series of text fragments that look like this:

```
<tag attribute="value"> Navigable string </tag>
```

Tags specify how the data contained within the page are organized and how the visual elements on this page should look. Tags are designated by opening and closing angle braces, $<$ and $>$.

Using `BeautifulSoup()`

First download the HTML for a website using the `requests.get()` function, including a user agent string. The raw HTML code contains a series of text fragments that look like this:

```
<tag attribute="value"> Navigable string </tag>
```

Tags specify how the data contained within the page are organized and how the visual elements on this page should look. Tags are designated by opening and closing angle braces, $<$ and $>$.

If the tag operates on text that immediately follows, a closing $<$/tag$>$ frames the text. The text in between the opening and closing tag is called the **navigable string**.

Using `BeautifulSoup()`

First download the HTML for a website using the `requests.get()` function, including a user agent string. The raw HTML code contains a series of text fragments that look like this:

```
<tag attribute="value"> Navigable string </tag>
```

Tags specify how the data contained within the page are organized and how the visual elements on this page should look. Tags are designated by opening and closing angle braces, $<$ and $>$.

If the tag operates on text that immediately follows, a closing $<$/tag$>$ frames the text. The text in between the opening and closing tag is called the **navigable string**.

Attributes are listed inside an opening tag to modify the behavior of that tag or to attach relevant data to the tag.

Using `BeautifulSoup()`

Getting Python to register text as a particular coding standard is called **parsing** the code.

Using `BeautifulSoup()`

Getting Python to register text as a particular coding standard is called **parsing** the code.

We parse HTML code with `BeautifulSoup()`:

```
from bs4 import BeautifulSoup
wnrn = BeautifulSoup(r.text, 'html')
```

Using `BeautifulSoup()`

Getting Python to register text as a particular coding standard is called **parsing** the code.

We parse HTML code with `BeautifulSoup()`:

```
from bs4 import BeautifulSoup
wnrn = BeautifulSoup(r.text, 'html')
```

The output has methods that allow us to navigate the organizational structure of the HTML code and extract data.

Using `BeautifulSoup()`

Getting Python to register text as a particular coding standard is called **parsing** the code.

We parse HTML code with `BeautifulSoup()`:

```
from bs4 import BeautifulSoup
wnrn = BeautifulSoup(r.text, 'html')
```

The output has methods that allow us to navigate the organizational structure of the HTML code and extract data.

How do you know what tags, etc. to search for?

Using `BeautifulSoup()`

Getting Python to register text as a particular coding standard is called **parsing** the code.

We parse HTML code with `BeautifulSoup()`:

```
from bs4 import BeautifulSoup
wnrn = BeautifulSoup(r.text, 'html')
```

The output has methods that allow us to navigate the organizational structure of the HTML code and extract data.

How do you know what tags, etc. to search for?

I prefer to open the page I am trying to scrape in a browser, "View Page Source", and use search (control + F) to find examples of datapoints I want.

Using `BeautifulSoup()`

Calling a tag as an attribute grabs the first occurrence of that tag:

```
metatag = wnrn.meta
metatag
```

```
<meta charset="utf-8"/>
```

Using `BeautifulSoup()`

Calling a tag as an attribute grabs the first occurrence of that tag:

```
metatag = wnrn.meta
metatag
```

```
<meta charset="utf-8"/>
```

Calling HTML attributes as list elements grabs the value of the attribute:

```
metatag['charset']
```

```
'utf-8'
```

Using `BeautifulSoup()`

To extract the navigable string, use `.string`:

```
titletag = wnrn.title
titletag
```

```
<title>WNRN - Independent Music Radio</title>
```

```
titletag.string
```

```
'WNRN - Independent Music Radio'
```

Using `BeautifulSoup()`

The `.find_next()` method grabs the next tag with the same name. For example:

```
spantag = wnrn.span
spantag
```

```
<span class="artist">Khruangbin &amp; Leon Bridges</span>
```

```
spantag.find_next()
```

```
<div class="info"><span class="release">Texas Sun</span></div>
```

Using `BeautifulSoup()`

To find all occurrences of a tag, organized in a list, use `.find_all()` and provide the tag as the argument:

```
spanlist = wnrn.find_all("span")
spanlist
```

```
[<span class="artist">Khruangbin &amp; Leon Bridges</span>
 <span class="song">Texas Sun</span>,
 <span class="release">Texas Sun</span>,
 <span class="artist">Talking Heads</span>,
 <span class="song">Burning Down The House</span>,
 <span class="release">Speaking In Tongues</span>,
 <span class="artist">Steve Earle</span>,
 <span class="song">You're Still Standin' There</span>,
 <span class="release">I Feel Alright</span>
```

Using `BeautifulSoup()`

If a tag has an associated class attribute, you can call that class in `.find_all()` as well:

```
artistlist = wnrn.find_all("span", "artist")
artistlist
```

```
[<span class="artist">Khruangbin &amp; Leon Bridges</span>
 <span class="artist">Talking Heads</span>,
 <span class="artist">Steve Earle</span>,
 <span class="artist">Rookie</span>,
 <span class="artist">Heartless Bastards</span>
```

Using `BeautifulSoup()`

If a tag has an associated class attribute, you can call that class in `.find_all()` as well:

```
artistlist = wnrn.find_all("span", "artist")
artistlist
```

```
[<span class="artist">Khruangbin &amp; Leon Bridges</span>
 <span class="artist">Talking Heads</span>,
 <span class="artist">Steve Earle</span>,
 <span class="artist">Rookie</span>,
 <span class="artist">Heartless Bastards</span>
```

To find all the tags with a specific attribute, use a logical statement:

```
atags_title = wnrn.find_all("a", title=True)
```

Constructing a Data Frame from HTML Data

We need to build a clean dataframe, which means we need to remove HTML tags and isolate the data.

Constructing a Data Frame from HTML Data

We need to build a clean dataframe, which means we need to remove HTML tags and isolate the data.

A very useful tool is a list comprehension:

```
newlist = [ expression for item in oldlist if condition ]
```

Constructing a Data Frame from HTML Data

We need to build a clean dataframe, which means we need to remove HTML tags and isolate the data.

A very useful tool is a list comprehension:

```
newlist = [ expression for item in oldlist if condition ]
```

We create a new list by

- iteratively performing operations on the elements of an existing list (`oldlist`).

Constructing a Data Frame from HTML Data

We need to build a clean dataframe, which means we need to remove HTML tags and isolate the data.

A very useful tool is a list comprehension:

```
newlist = [ expression for item in oldlist if condition ]
```

We create a new list by

- iteratively performing operations on the elements of an existing list (`oldlist`).
- `item` represents one item of the existing list,

Constructing a Data Frame from HTML Data

We need to build a clean dataframe, which means we need to remove HTML tags and isolate the data.

A very useful tool is a list comprehension:

```
newlist = [ expression for item in oldlist if condition ]
```

We create a new list by

- iteratively performing operations on the elements of an existing list (`oldlist`).
- `item` represents one item of the existing list,
- `expression` is the Python code we would use on a single element of the existing list, except we replace the name of the element with `item`,

Constructing a Data Frame from HTML Data

We need to build a clean dataframe, which means we need to remove HTML tags and isolate the data.

A very useful tool is a list comprehension:

```
newlist = [ expression for item in oldlist if condition ]
```

We create a new list by

- iteratively performing operations on the elements of an existing list (`oldlist`).
- `item` represents one item of the existing list,
- `expression` is the Python code we would use on a single element of the existing list, except we replace the name of the element with `item`,
- and `condition` sets a filter: only certain elements are transformed and placed into the new list.

Constructing a Data Frame from HTML Data

For example, to extract the navigable string from every element of artistlist:

```
artists = [a.string for a in artistlist]
```

```
['Khruangbin & Leon Bridges',
 'Talking Heads',
 'Steve Earle',
 'Rookie',
 'Heartless Bastards',
 'Leo Kottke',
 'Stray Fossa',
 'The Raconteurs',
 'My Morning Jacket',
 'Drive-By Truckers',
 'Boy and Bear',
 'Andy Jenkins']
```

Constructing a Data Frame from HTML Data

To construct a clean data frame, we create a dictionary that combines these cleaned lists and passes this dictionary to the `pd.DataFrame()` function:

```
mydict = {'time':times,
           'artist':artists,
          'song':songs,
          'album':albums}
wnrn_df = pd.DataFrame(mydict)
wnrn_df
```

	time	artist	song	album
0	6:43 AM	Khruangbin & Leon Bridges	Texas Sun	Texas Sun
1	6:39 AM	Talking Heads	Burning Down The House	Speaking In Tongues
2	6:36 AM	Steve Earle	You're Still Standin' There	I Feel Alright
3	6:31 AM	Rookie	Sunglasses	Rookie
4	6:26 AM	Heartless Bastards	Parted Ways	Arrows
5	6:23 AM	Leo Kottke	Stolen	Try And Stop Me
6	6:20 AM	Stray Fossa	It's Nothing	(Single)

Building a Spider

A spider is a web scraper that follows links on a page automatically and scrapes from those links as well.

Building a Spider

A spider is a web scraper that follows links on a page automatically and scrapes from those links as well.

To build a spider:

Building a Spider

A spider is a web scraper that follows links on a page automatically and scrapes from those links as well.

To build a spider:

1. Find the URLs for the additional webpages that contain relevant data. Store these URLs in a list.

Building a Spider

A spider is a web scraper that follows links on a page automatically and scrapes from those links as well.

To build a spider:

1. Find the URLs for the additional webpages that contain relevant data. Store these URLs in a list.

2. Put all of the code to scrape one page into a **single function** that takes a URL as input and outputs a data frame.

Building a Spider

A spider is a web scraper that follows links on a page automatically and scrapes from those links as well.

To build a spider:

1. Find the URLs for the additional webpages that contain relevant data. Store these URLs in a list.

2. Put all of the code to scrape one page into a **single function** that takes a URL as input and outputs a data frame.

3. Loop over the URLs, applying the function to each one, and append these data frames together.

Building a Spider

1. Find the URLs for the additional webpages that contain relevant data. Store these URLs in a list.

Building a Spider

1. Find the URLs for the additional webpages that contain relevant data. Store these URLs in a list.

```
recent = wnrn.find("div", "recent-playlists")
recent_atags = recent.find_all("a")
wnrn_url = [pl['href'] for pl in recent_atags if "/pl/" in pl['href']]
wnrn_url
```

```
['/WNRN/pl/10646133/WNRN-4-4-20-5-00-AM',
 '/WNRN/pl/10646058/WNRN-4-4-20-4-02-AM',
 '/WNRN/pl/10645824/WNRN-4-4-20-3-01-AM',
 '/WNRN/pl/10645245/WNRN',
 '/WNRN/pl/10644264/WNRN']
```

Building a Spider

2. Put all of the code to scrape one page into a **single function** that takes a URL as input and outputs a data frame.

Building a Spider

2. Put all of the code to scrape one page into a **single function** that takes a URL as input and outputs a data frame.

```
def wnrn_spider(url):
    headers = {'user-agent': 'Class example (jkropko@virginia.edu)'}
    r = requests.get(url, headers=headers)
    wnrn = BeautifulSoup(r.text, 'html')
    artistlist = wnrn.find_all("span", "artist")
    songlist = wnrn.find_all("span", "song")
    albumlist = wnrn.find_all("span", "release")
    timelist = wnrn.find_all("td", "spin-time")
    artists = [a.string for a in artistlist]
    songs = [a.string for a in songlist]
    albums = [a.string for a in albumlist]
    times = [a.string for a in timelist]
    mydict = {'time':times, 'artist':artists,
                'song':songs, 'album':albums}
    wnrn_df = pd.DataFrame(mydict)
    return wnrn_df
```

Building a Spider

3. Loop over the URLs, applying the function to each one, and append these data frames together.

Building a Spider

3. Loop over the URLs, applying the function to each one, and append these data frames together.

```
wnrn_total_playlist = wnrn_df
for w in wnrn_url:
    moredata = wnrn_spider('https://spinitron.com/' + w)
    wnrn_total_playlist = wnrn_total_playlist.append(moredata)
wnrn_total_playlist
```

	time	artist	song	album
0	6:43 AM	Khruangbin & Leon Bridges	Texas Sun	Texas Sun
1	6:39 AM	Talking Heads	Burning Down The House	Speaking In Tongues
2	6:36 AM	Steve Earle	You're Still Standin' There	I Feel Alright
3	6:31 AM	Rookie	Sunglasses	Rookie
4	6:26 AM	Heartless Bastards	Parted Ways	Arrows
...	...	...	...	...
53	11:43 PM	Waxahatchee	Lilacs	Saint Cloud
54	11:47 PM	Nathaniel Rateliff	And It's Still Alright	And It's Still Alright
55	11:51 PM	Spoon	Hot Thoughts	Hot Thoughts
56	11:55 PM	Alejandro Escovedo	Sister Lost Soul	Real Animal
57	11:58 PM	Vagabon	Water Me Down	Vagabon