Introduction to Data Science

Web Scraping in Python

Joanna Bieri DATA101

Important Information

- Email: joanna_bieri@redlands.edu
- Office Hours: Duke 209 Click Here for Joanna's Schedule

Announcements

In NEXT WEEK - Data Ethics This week you should be reading your book or articles.

Web Scraping Ethical Issues

There are some things to be aware of before you start scraping data from the web.

- Some data is private or protected.
- Some websites have rules against scraping and will cut of service to users who are clearly scraping data.
- The line between web scraping and plagiarism can be very blurry.
- Ethics are different depending on if you are using the data for a personal project vs if you are using the project for your business or website

Web Scraping Ethical Issues

The Ethical Scraper (from https://towardsdatascience.com/ethics-in-web-scraping-b96b18136f01):

I, the web scraper will live by the following principles:

- If you have a public API that provides the data I'm looking for, I'll use it and avoid scraping all together.
- I will always provide a User Agent string that makes my intentions clear and provides a way for you to contact me with questions or concerns.
- I will request data at a reasonable rate. I will strive to never be confused for a DDoS attack.
- I will only save the data I absolutely need from your page. If all I need it OpenGraph meta-data, that's all I'll keep.

Web Scraping Ethical Issues

The Ethical Scraper (from https://towardsdatascience.com/ethics-in-web-scraping-b96b18136f01):

I, the web scraper will live by the following principles:

- I will respect any content I do keep. I'll never pass it off as my own.
- I will look for ways to return value to you. Maybe I can drive some (real) traffic to your site or credit you in an article or post.
- I will respond in a timely fashion to your outreach and work with you towards a resolution.
- I will scrape for the purpose of creating new value from the data, not to duplicate it.

Using pandas to get table data.

We have already briefly seen this in action!

If the data on the website you are interested in is already written in a table then Pandas can grab that data and save it to a data frame.

Using pandas to get table data.

ACES DATA =
https://www.basketball-reference.com/wnba/teams/SAS/players.html

	Rk	Player	From	To	Yrs	Unnamed: 5	G
0	1	Danielle Adams	2011	2015	5	NaN	155
1	2	Elisa Aguilar	2002	2002	1	NaN	28
2	3	Kayla Alexander	2013	2017	5	NaN	154
3	4	Lindsay Allen	2018	2020	2	NaN	45
4	5	Chantelle Anderson	2005	2007	3	NaN	68
...	...	...	...	...	...	...	...
146	147	Nevriye Yilmaz	2004	2004	1	NaN	7
147	148	Jackie Young	2019	2024	6	NaN	199
148	149	Sophia Young-Malcolm	2006	2015	9	NaN	301
149	150	Tamera Young	2018	2019	2	NaN	67
150	151	Shanna Zolman	2006	2009	3	NaN	87

Using pandas to get table data.

What are these unnamed columns?

```
columns = ['Unnamed: 5', 'Unnamed: 22','Unnamed: 26']
ACES[columns]
```

	Unnamed: 5	Unnamed: 22	Unnamed: 26
0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	NaN	NaN	NaN
...	...	...	...
146	NaN	NaN	NaN
147	NaN	NaN	NaN
148	NaN	NaN	NaN
149	NaN	NaN	NaN

Dealing with NaNs

- First NaNs are a strange data type (**np.nan**) they are considered a float - like a decimal.
- In most raw data sets NaN means no data was given for that observation and variable, but be careful NaN can also happen if you do a calculation and accidentally divide by zero.

Dealing with NaNs

- **.isna()** creates a mask for whether or not there is a NaN in each row of the data.
- **.fillna()** will replace NaN in your data set with whatever you put inside the parenthesis
- **.dropna()** will drop all rows that contain NaN - becareful with this command. You want to keep as much data as possible and .dropna() might delete too much!

Dealing with NaNs

```
ACES.dropna(inplace=True)
ACES
```

Rk	Player	From	To	Yrs	Unnamed: 5	G	MP	FG	FGA	...

Oh No! We deleted our data!!!!

We got rid of any row that contains NaN, but if we look at our unnamed columns, they are all NaNs!

Be careful when removing NaNs!

Sometimes you get errors!

- You can try installing the packages that python asks for, but some websites use pretty advanced coding.
- When you see a 404 forbidden error, this means that the website is trying to stop you from scraping and you would need to use even more advanced techniques!

```
website = 'https://www.scrapethissite.com/pages/simple/'
df = pd.read_html(website)
```

```
HTTPError: HTTP Error 403: Forbidden
```

Using Beautiful Soup to get HTML code

Websites are built using html code. That code tells the web browser (FireFox, Chrome, etc) what to display. Websites can be very simple (just html) to much more complicated (java script $+$). When you load a website you can always see the source code.

Using Beautiful Soup to get HTML code

- Right Click - view page source

This is what beautiful soup downloads. For static (simple) sites this code is immediately available. More complicated sites might require Python to open the webpage, let the content render, and then download the code.

How to get data from static sites:

You should already have the packages bs4 and requests but if you get an error try running:

```
!conda install -y bs4
!conda install -y requests
```

How to get data from static sites:

```
import requests
from bs4 import BeautifulSoup

website = 'https://www.scrapethissite.com/pages/simple/'

raw_code = requests.get(website)
html_doc = raw_code.text
soup = BeautifulSoup(html_doc, 'html.parser')
```

This is HTML code

```
<!DOCTYPE html>

<html lang="en">
<head>
<meta charset="utf-8"/>
<title>Countries of the World: A Simple Example | Scrape T
<link href="/static/images/scraper-icon.png" rel="icon" ty
<meta content="width=device-width, initial-scale=1.0" name
<meta content="A single page that lists information about
<link crossorigin="anonymous" href="https://maxcdn.bootstr
<link href="https://fonts.googleapis.com/css?family=Lato:4
<link href="/static/css/styles.css" rel="stylesheet" type=
<meta content="noindex" name="robots"/>
<link href="https://lipis.github.io/flag-icon-css/css/flag
</head>
<body>
```

This is HTML code

WHAT A MESS!

The information in soup is ALL of the code and unless you are awesome at reading HTML, this is indecipherable. We need to be able to find specific parts of this to extract the data.

Extracting data from HTML:

We will use the **soup.find_all()** function.

Here is the simplified function signature:

```
soup.find_all(name=None,attrs={})
```

You can type soup.find_all? and run it to see all the information about additional arguments and advanced processes.

Extracting data from HTML:

Here is how we will mostly use it, but there are much more advanced things you can do:

```
soup.find_all( <type of section>, <info> )
```

Extracting data from HTML:

The **.find_all()** function searches through the information in soup to match and return only sections that match the info. Here are some important types you might search for:

Extracting data from HTML:

- 'h2' - this is a heading
- 'div' - this divides a block of information
- 'span' - this divides inline information
- 'a' - this specifies a hyperlink
- 'li' - this is a list item
- class_= - many things have the class label (notice the underscore!)
- string= - you can also search by strings.

Using Developer tools:

To figure out what data to extract I suggest you use developer tools on the website to find what you need. Navigate to the website:

Scrape This Site - Website

I really like Brave Browser or Google Chrome for this, but most browsers with have **More Tools/Developer Tools** where you can see the code.

Search for all the country names

The names of the country are inside

```
<h3 class="country-name">
```

So lets search for this:

```
result = soup.find_all('h3',class_="country-name")
```

This is still a mess

We can see the country names but they are surrounded by other junk. Here is how we will handle this.

1. We will start and EMPTY data frame using

   ```
   DF = pd.DataFrame()
   ```

2. We will add our **soup.find_all** results as a column of the data frame
3. We will fix the data to strip off all of the unneeded text.

How to get the text

What this returns is a list of all the text that is inside a link block of code. If we just want to look at the text we can!

```
result[0].text.lstrip().rstrip()
```

Put this into a data frame

```
DF = pd.DataFrame()
DF['country']=result
DF['country'] =DF['country'].apply(lambda x: x.text.rstrip().l
DF
```

Put this into a data frame

	country
0	Andorra
1	United Arab Emirates
2	Afghanistan
3	Antigua and Barbuda
4	Anguilla
...	...
245	Yemen
246	Mayotte
247	South Africa
248	Zambia
249	Zimbabwe

Lets try again and this time add the country capital

```
result = soup.find_all('span',class_="country-capital")
DF['capital']=result
DF['capital'] = DF['capital'].apply(lambda x: x.text)
DF
```

	country	capital
0	Andorra	Andorra la Vella
1	United Arab Emirates	Abu Dhabi
2	Afghanistan	Kabul
3	Antigua and Barbuda	St. John's
4	Anguilla	The Valley
...	...	...
245	Yemen	Sanaa
246	Mayotte	Mamoudzou
247	South Africa	Pretoria
248	Zambia	Lusaka

Where can you get more practice

Here is a website dedicated to allowing students to practice webscraping:

www.scrapethissite.com

There are “sandbox” websites that are indended for scraping. The only thing the site asks is:

Please be Well-Behaved

Just like any site you'd scrape out in the wild wild west (www), please be mindful of other users trying to access the site. From a technical standpoint, you must observe the following rules:

- Clients may only make a maximum of one request per second
- Clients must send an identifying user agent
- Clients must respect this site's robots.txt file

Any client that violates the rules above or otherwise tries to interfere with the site's operation will be subject to a temporary or permanent ban.

Be a good web scraping citizen.