Introduction to Web Scraping

Lacey Conrad
MSDS 610
Regis University

June, 2020

1 Introduction

Web scraping, also known as web data extraction, is the retrieval of data from a website and then storing it for use in future analysis ("What is web scraping and how does web crawling work?," 2020). Web scraping involves fetching and extracting data from a web page. In the fetching phase, a web page is downloaded. After the web page is fetched data can be extracted from the downloaded page, and is frequently either parsed, reformatted, searched, stored, etc ("Web scraping," 2005). This process enables the creation of business applications, such as for price monitoring, finance, market research, real estate and so on ("What is web scraping and how does web crawling work?," 2020). The general process is fairly straightforward and consists of 3 steps:

1. Scraping - scrapers can be specifically designed for each application depending on the information that needs to be retrieved.

2. Data retrieval in HTML - the data is retrieved in HTML and then parsed to extract raw data and cleaned of noise.

3. Storage - the data is stored in whatever format is needed for the application ("What is web scraping and how does web crawling work?," 2020).

   The exercise this week involved scraping news stories from a website. The example used in the exercise scraped the Denver City website for news stories and used BeautifulSoup to store and search the collected HTML code. To do this, a call was made to the Denver City website to collect the HTML of the website we were interested in scraping. BeautifulSoup was then used to store the HTML code, and searches were performed on stored code in order to extract individual stories so that we may collect the title, date, and website from them. Once I was familiar enough with the `find()` feature in BeautifulSoup, I created a loop that would loop through all the news stories on a page and collect our information (title, url, and date). Then, I made another loop, that parsed through a range of news pages (where each page had multiple stories) and collected the three pieces of information we were looking for. This information was simultaneously inserted into a MongoDB database. When I performed my own webscrape, I chose to scrape the Longmont City news website. I followed a nearly identical procedure to above, the only difference being having to performed another level of searching to produce a list of news stories. This information was also inserted into MongoDB in a manner identical to the above procedure.

2 Written portion - Questions from Week 6

1. What are the ethics of web scraping? How do we know when it's allowed vs not allowed? What are possible legal consequences of unauthorized web scraping?

   Unfortunately, web scraping is fairly easy, and with some Python and tools like BeautifulSoup it is very simple to GET a web page and parse it. Small, several thousand page web scrapes done for weekend projects or in support of a business projects generally are usually not problem spots for unethical scraping. High volume web scraping for commercial use tends to be the form of scraping which may not always be ethically conducted (Densmore, 2019). Sadly, there is much gray areas in the law surrounding web scraping, and a wise first step when web scraping is to make sure you respect any websites rules and terms of service (Bernard, 2017).

Web scraping has, in recent years, seen its reputation drop as a result of unethical and illegal scraping. Here are a few of the ways it is being improperly used:

(a) It is being used by businesses to gain an upper hand against their rivals, with a clear financial motivation behind their scraping uses.
(b) It is being done without any regard to copyright laws, or terms of service.
(c) Some scraping is performed in an abusive manner. Sometimes this means creating an unexpected load on websites being scraped. Other times, this may include circumventing security measures or performing other prohibited operations to reach the data. In many cases, these security measure were put in place to prohibit web scraping (Bernard, 2017).

Web scraping and crawling by themselves are not illegal. The illegality of scraping arises when scraping websites without permission, without consideration to the Terms of Service, or for use in purposes outside the site owner's control. Reading the Terms of Service is one important way to determine if web scraping is legal on the site you are planning to scrape, as well as how the scraping can be legally performed. For example, many of the larger social networks who are frequently scraped have a separate Terms of Service specifically for web scraping. A company is free to pursue legal action against a person who has broken their Terms of Service.

The legal consequences vary a great deal, depending on the scale of the web scraping abuse, and if the law was broken. The website may choose to take technical measures to block scraping, they could send you a cease and desist letter, or they can sue you. This has happened many times, such as the case of Linkedin v. Doe Defendants, where Linkedin sued several people for scraping their site unlawfully. The list of offenses the Doe Defendants in the Linkedin case can also give us an idea of the laws an unlawful web scraping can break:

(a) Violation of the Computer Fraud and Abuse Act (CFAA)
(b) Violation of the California Penal Code
(c) Violation of the Digital Millennium Copyright Act (DMCA)
(d) Breach of contract
(e) Trespass
(f) Misappropriation

The problem that can arise if a website pursues legal action is that you will need to defend yourself in court, and will likely need a lawyer. And even with a lawyer, web scraping law suits (such as those from copyright infringements) can often result in monetary damages. In a more serious case, Andrew Auernheimer was convicted of hacking (he was web scraping), and since he harvested data en mass, it constituted a "brute force attack." This charge is a felony, and each charge of it carries a 15-year prison sentence (Roberts, 2020).

In the United States, there are 3 major legal claims used against unlawful web scraping:

(a) Copyright Infringement
(b) Violation of the Computer Fraud and Abuse Act (CFAA)
(c) Trespass to chattel

These claims all require certain criteria to be met in order to be formally used ("Web scraping," 2005).

2. What are some of the technologies we can use for web scraping?

   (a) Selenium: Selenium can be used to mimic the manner in which a human will browse and interact with a website, allowing a scrape to get the data off of a site that a visitor is likely to see. It can also be used to automate website testing, and provides insight into how a website works (Koshy, 2016).

(b) Boilerpipe: BoilerPipe is designed to extract structured or unstructured data from a website using a Java library. It is able to remove noise from web pages, including HTML tags. It is a very easy to use, rapid scraper that needs little user input (Koshy, 2016).

(c) Nutch: Nutch is used for web crawling, extracting data, and then storing it. The web pages to be traversed and extracted from must be manually entered into Nutch's code (Koshy, 2016).

(d) Watir: Once again, this tool interacts with a website in the same manner a user would (including clicking links, filling out forms, pressing buttons, and so on) (Koshy, 2016).

(e) Celerity: Celerity is an easy to use API that can navigate through web applications. It is also a browser automation tool, which can be set up to run in the background silently (Koshy, 2016).

(f) HTML parsing: Text can be extracted from HTML using parsing with JavaScript (Chapman, 2020).

(g) DOM parsing: Document Object Model (DOM) parsing allows for an in-depth view of a websites structure. A DOM parser can also be used to get nodes containing information, and then use a tool (like XPath) to scrape websites (Chapman, 2020).

(h) Vertical Aggregation: These platforms are typically used by business with access to large scale computing power. It is used to target specific verticals, which are monitored by bots without any human interaction (Chapman, 2020).

(i) XPath: XPath is a query language that is used to navigate XML documents by selecting parameters to search nodes for (Chapman, 2020).

(j) Google Sheets: Very specific amounts of data can be scraped using this tool. It is useful for scraping when specific data or patterns are required from a site. Interestingly, this technology can also be used to check if your site is secure from web scraping (Chapman, 2020).

(k) Text pattern matching: This makes use of the UNIX grep command, and can be used with popular programming languages (Chapman, 2020).

3 Methods and Code

This exercise was performed with Python (in Jupyter notebook) and the database used was MongoDB. The output produced as a result of data verification or to visualize certain steps will be included in the methods section. The database queries will be in the results section.

3.1 Web Scraping Example - Denver, CO News

I began the practice web scraping exercise by using GET to obtain data off the City of Denver's website for news. In a manner not unlike what was used to call an API, I instead called directly to the website that I wanted to scrape. I then saved the scraped data as a BeautifulSoup object by using the BeautifulSoup library. This library allows for data retrieval and markup removal from HTML, in addition to saving the results.

```
import requests as req
import pandas as pd
from bs4 import BeautifulSoup as bs
from pymongo import MongoClient
from tqdm import tqdm_notebook
from pprint import pprint

# Getting the contents of the Denver City new website:
res = req.get('https://www.denvergov.org/content/denvergov/en/city-of-
        denver-home/news.html')

# Saving the contents of the website as a BeautifulSoup object.
soup = bs(res.content)
```

Through the Jupyter examples and the video accompanying this exercise I know that the individual news stories we want to scrape are stored under the "div class = denver-news-list-item." Using BeautifulSoup, I searched the

HTML for the tag "div class" which corresponded to "denver-news-list-item." Instances of this tag were located, and then stored in an array. I also converted the dates present in the "news list item" to datetime format, so they can be inserted as such in MongoDB.

```
# Finding the tag "div class" that corresponds to "denver-news-list-item"
# which is where the body of each new story is kept.
divs = soup.find_all('div', {'class': 'denver-news-list-item'})

#Viewing the datatype of a member of the divs array.
type(divs[0])

# Viewing the first entry in the divs array
divs[0]

# with the div array, find members of the 'p class'
# that are tagged as "denver-news-list-date," converting
# them to datetime format, and then printing the result.
pd.to_datetime(divs[0].find('p', {'class': 'denver-news-list-date'}).text)
```

Table 1: Here is the output of `type(divs[0])` and `divs[0]` :.

type(divs[0])
bs4.element.Tag
divs[0]
<div class="denver-news-list-item"> <p class="denver-news-list-date">Jun 12, 2020</p> <h3> <a href="/content/denvergov/en/vision-zero/blog/articles/high-line -canal-trail-connections.html">High Line Canal Trail Connections</a> </h3> <img alt="" class="hidden-xs"src="/content/dam/denvergov/Portals/ 705/images/news/high-line-canal-rendering.jpg"/> <p>Imagine taking a walk or riding your bike along the High Line Canal Trail, when suddenly, you come to the intersection of Colorado Boulevard &amp; Hampden Avenue. Two highly-traveled corridors for people who drive - about 77,000 vehicles pass through each day – so, it could be intimidating to cross on foot or on a bike, right? That's why DOTI is working on a project that will create a safer, more convenient connection for people who walk and ride bikes on the High Line Canal Trail!</p> <div style="clear:both;"></div> </div>

To practice using BeautifulSoup to search tags within my scraped document, I practiced searching for various aspects of the `<a>` tag. I also found the end portion of the website address that corresponded to each specific news entry, and used that plus the base web address to recreate a full URL for each news item.

```
# The following is some practice locating elements of
# the downloded webpage, including using the tag 'a'
# to locate the text of divs[0], and then the end portion
# of the url (href):
divs[0].find('a')
divs[0].find('a').text
```

```
divs[0].find('a').get('href')

# Using the base url of the scraped site and
# the href to create a full link to the news story:
BASE_URL = 'https://www.denvergov.org'
BASE_URL + divs[0].find('a').get('href')
```

Table 2: Using BeautifulSoup to search for parts of the news listing stored in the 1st slot of the divs array.

divs[0].find('a').text: 'High Line Canal Trail Connections'

pd.to_datetime(divs[0].find('p', 'class': 'denver-news-list-date').text):
Timestamp('2020-06-12 00:00:00')

divs[0].find('a'): <a href="/content/denvergov/en/vision-zero/blog/articles/high-line-canal-trail-connections.html">High Line Canal Trail Connections</a>
divs[0].find('a').get('href'): '/content/denvergov/en/vision-zero/blog/articles/high-line-canal-trail-connections.html'
BASE_URL: "https://www.denvergov.org"
BASE_URL + divs[0].find('a').get('href'): 'https://www.denvergov.org/content/denvergov/en/vision-zero/blog/articles/high-line-canal-trail-connections.html'

Next, I created a loop that would loop through each item in the divs array, and extract from each news item a date, link, and title. As I did above, I also recreated the web address for each news item using the link and the base URL.

```

# Looping through the content in the array divs and
# setting the contents to variables, to allow for
# easier insertion into a database.  The manner
# in which these variables were created was explained
# previously:
for d in divs:
        date = pd.to_datetime(d.find('p', {'class': 'denver-news-list-date'}).text)
        link = d.find('a')
        href = BASE_URL + link.get('href')
        title = link.text

# Printing out the last iteration of the variables
# title, date, and href.
title
date
href
```

Table 3: The date, title, and URL acquired from a loop through the divs array, using BeautifulSoup to search for tag elements.

href: 'https://www.denvergov.org/content/denvergov/en/denver-city-council/news/2020/week-of-june-15–2020-meeting-schedule-and-agendas.html'
date: Timestamp('2020-06-11 00:00:00')
title: 'Week of June 15, 2020 Meeting Schedule and Agendas'

Now that I had my scraped data and a loop that was able to extract relevant information from each news item, I was ready to insert the data I was extracting into MongoDB. To enable my scraper to collect even more news data I created another loop which iterated through multiple pages of news stories (In the past examples, I was looking at single page, which contained multiple news stories linked off of it). I then performed the same BeautifulSoup search to located the distinct news stories, and looped through that, this time inserting the collected information into a MongoDB database.

```
# Starting this time with the Denver City Government
# website with pagination left open, in order to be
# filled in later by a loop
PAGE_URL = 'https://www.denvergov.org/content/denvergov/en/
        city-of-denver-home/news.html?page={}'

# Connecting to MongoDB and creating connections to the
# news database and the denver collection:
client = MongoClient()
db = client['news']
coll = db['denver']

# tqdm was used to show a bar that showed progress
# towards completing currently executing code.
# The loop is set to iterate through
# 70 pages of news on the website, during which
# each page will have its contents obtained through
# GET, then stored as a BeautifulSoup object.
# Similar to the initial procedure I followed,
# I then used BeautifulSoup to locate ' div class'
# with the title 'denver-news-list-item,' and saved
# that to an array 'divs.'
for page in tqdm_notebook(range(1, 71)):
        url = PAGE_URL.format(page)
        res = req.get(url)
        soup = bs(res.content)
        divs = soup.find_all('div', {'class': 'denver-news-list-item'})

# Similar to the loop describe previously to obtain
# date, title, and url from each news story,
# however, in this case, each component is also inserted into
# the MongoDB we set up for this database:
        for d in divs:
                date = pd.to_datetime(d.find('p', {'class':
                        'denver-news-list-date'}).text)
                link = d.find('a')
                href = BASE_URL + link.get('href')
                title = link.text

                coll.insert_one({'date': date,
                                                'link': href,
                                                'title': title})

client.close()
```

To verify data insertion into MongoDB, I queried the database to send me the first item in the database.

```
from pprint import pprint

client = MongoClient()
db = client['news']
coll = db['denver']

# To make sure our data was
# inserted, I queried the collection
# using \code{find_one()} to print the
# first entry in the database:
pprint(coll.find_one())
```

3.2 Longmont, CO City News Scrape

Similar to the Denver City news example, I decided to web scrape the news pages on the Longmont City website. While the structure to the website was similar, the underlying HTML was not. I had to spend some time figuring out what tags I needed to call to obtain the data I wanted (news title, date, website, etc).

I used the developer tools provided with Google Chrome to inspect a news page out of the website. The following was what I observed to be connected to the stories on any given page within the news pages.

```
<ul class="list-main clearfix">
  <li>
    <div class="item-img">
      <img alt="City facilities closures extended through April 26,2020" class=
      "item-img" src="/Home/ShowPublishedImage/23431/637218634813800000"/>
    </div>
    <h2>
      <a class="item-title" href="/Home/Components/News/
      News/9891/3?npage=4">
      Closure of City Buildings Extended Through April 26
      </a>
        </h2>
        <p class="item-intro">
          Based on continuing developments related to COVID-19,
          the City is extending the closure of all public buildings
          through April 26, 2020. In alignment with Gov. Jared
          Polis, the City is also recommending residents to
          wear cloth face coverings when they go out of the
          house for essential trips.
        </p>
        <p class="item-date">
          04/07/2020 1:48 PM
        </p>
  </li>
```

I then used GET to acquire the code from page 2 of the Longmont City news:

```
# Requesting webpage data from the Longmont, CO city website:
res = req.get('https://www.longmontcolorado.gov/news/-npage-2')
```

```python

# Saving the data as a BeautifulSoup object, and then
# printing it using prettify():
soup = bs(res.content)
print(soup.prettify())
```

I also used prettify from BeautifulSoup to more easily looked at the code received from the GET command.

Table 4: Here is the output of `print(soup.prettify())` :.

print(soup.prettify())
<!DOCTYPE html>
<html lang="en">
<head id="Head1">
<meta charset="utf-8"/>
<meta content="IE=9; IE=8; IE=7; IE=EDGE" http-equiv="X-UA-Compatible"/>
<title>
Citywide News | City of Longmont, Colorado
</title>
<meta content="width=device-width" name="viewport"/>
<link href="/DefaultContent/Default/StyleBundleDesignTheme.cssbnd?v=ioinOC033 D9ASRJPXUIzdsDMlIoz6Jcys2ud3eut2IE1" rel="stylesheet"/>
<link href="/Project/Contents/Main/StyleBundleDesignTheme.cssbnd?v=YPSNzqJiALyzwk _w29mdizcCHgwYvXsH3goKXXHYayM1" rel="stylesheet"/>
<link href="/Areas/Admin/Content/StyleBundleFrontendExtra.cssbnd?v=TXLLavBbQdlTrg6s ukdO1rojFtp3vRD061rBA3Rt7YM1" rel="stylesheet"/>
...

When using the Chrome developer tools, I was able to figure out that I would be searching for 'ul' instead of 'div.' This took me a little time, since I thought we had to search for "div." I located the tag 'list-main' which proved to be the tag associate with all the news stories on page 2.

```python
# Using the tag <ul class="list-main"> to
# extract from the data all of the all of the news
# stories and prettifying the results:
divs = soup.find('ul', {'class': 'list-main'})
print(divs.prettify())

# Finding all children of the divs BeautifulSoup
# object with the tag 'li' and then
# displaying the 3rd member of that array:
news = divs.findChildren('li')
news[2]
```

I inspected divs, and noticed that I still received a dump of data which didn't look like what I was working with in the example. It wasn't until I asked a web developer friend how I could pull individual stories out of divs, when they told me that I had to actually go one step deeper into my data to pull out the stories and their information. Since each story was contained within an `<li>` tag, I went about extracting the story from the list item. I had to look up methods to use to find the sub-tags of divs, and when looking at some BeautifulSoup examples, I noticed the `findChildren()` function. I arrived at the code previously displayed, which first extracts all the news stories, and then splits the news stories into children, which are each individual story. I now had an array similar to the one created in the example.

Here is some of the output of `divs.prettify()` which searched for tags of `<ul class="list-main">`. As is evident in this figure, a news story is contained within the tags `<li>` and `</li>`, making it fairly likely these are the tags I may need to use to search for individual stories.

Table 5: A brief example of the output of `print(divs.prettify())`. Observing a greater portion of this data shows that while we have individual news stories, we also have our data stored as one mass.

print(divs.prettify())
<ul class="list-main clearfix">
<li>
<div class="item-img">
<img alt="MedicareBasics_1080px" class="item-img"
src="/Home/ShowPublishedImage/24633/637254951996430000"/>
</div>
<h2>
<a class="item-title" href="/Home/Components/News/News/10986/3?npage=2">
Medicare Basics offered virtually in May and June
</a>
</h2>
<p class="item-intro">
Choose from a virtual evening option on Tuesday May 26th, or
connect during the daytime on Monday, June 15th to get your Medicare questions answered.
</p>
<p class="item-date">
05/20/2020 8:00 AM
</p>
</li>
...

Here is an example of a list item within the `<ul class="list-main">`. I made several tags that I would end up using in the future bold.

Table 6: The results of using BeautifulSoup to search for `divs.findChildren('li')` and storing the results in an array. The delineation of each story by <li> is now evident.

news
<li> <div class="item-img"> <img alt="city logo, thumbnail" class="item-img" src="/Home/ShowPublishedImage/23213/637199826038600000"/> </div> <h2> **<a class**="item-title" href="/Home/Components/News/News/10984/3? npage=2">Amenities Begin Reopening in Longmont Parks **</a>** </h2> **<p class="item-intro">** Several City of Longmont park amenities are reopening with specific requirements in place for public health and safety. **</p>** **<p class="item-date">** 05/18/2020 5:13 PM **</p>** </li> ...

I then used several of the techniques from the Denver News example to extract parts of a news story out of my news array. This enabled me to search for the date, title, and recreated the full URL the story was found at.

```
# Practicing searching through the list-item to find
# the title, date, and href of each news story:
pd.to_datetime(news[1].find('p', {'class': 'item-date'}).text)
news[2].find('a')
news[2].find('a').text
news[2].find('a').get('href')

# Reconstructing the website each news story can be
# found at by using the sites base url, the stories
# href, and concatenating them.  I also checked to make
# sure the concatenated site and the actual site matched:
BASE_URL = 'https://www.longmontcolorado.gov'
url = BASE_URL + news[2].find('a').get('href')
url
url2 = 'https://www.longmontcolorado.gov/Home/Components/News/News/10984/3?npage=2'
url == url2
```

Table 7: Using BeautifulSoup to search for parts of the news listing stored in the 3rd slot of the news array.

news[2].find('a').text: 'Amenities Begin Reopening in Longmont Parks'

pd.to_datetime(news[2].find('p', 'class': 'item-date').text):
Timestamp('2020-05-18 17:13:00')

news[2].find('a'):
<a class="item-title" href="/Home/Components/News/News/10984/3?
npage=2">Amenities Begin Reopening in Longmont Parks</a>

news[2].find('a').get('href'): '/Home/Components/News/News/10984/3?npage=2'
url: 'https://www.longmontcolorado.gov/Home/Components/News/News/10984/3?npage=2'
url == url2: True

Similar to the Denver News example, I created a loop to move through my array, and then print out the most recent href, date, and title.

```
# A loop to loop through the news array, pulling out
# the data (converted to datetime), url, and title,
# and then printing them:
for n in news:
        date = pd.to_datetime(n.find('p', {'class': 'item-date'}).text)
        link = n.find('a')
        href = BASE_URL + link.get('href')
        title = link.text

href
date
title
```

Table 8: The date, title, and URL extracted by BeautifulSoup from the news array using a loop.

href: 'https://www.longmontcolorado.gov/Home/Components/News/News/10946/3?npage=2'
date: Timestamp('2020-05-01 09:25:00')
title: 'This Week in Longmont - May 1, 2020'

Having worked the kinks out of most of my code, which now loops through the scraped data from a website and extracts the title, date, and URL from each news story, I was ready to increase the loop to looping through my divs variable, as well as inserting this information into MongoDB.

```
# Requesting data from page 4 of the Longmont City news,
# turning it into a BeautifulSoup object, and then
# searcing for news articles by using the tag
# <ul class="list-main">.  I also printed out this
# variable to make sure my data looked correct.
page = 'https://www.longmontcolorado.gov/news/-npage-4'
res = req.get(page)
soup = bs(res.content)
divs = soup.find('ul', {'class': 'list-main'})
divs
```

The following shows connecting to the MongoDB database, followed by insertion of the data in my news array into the database. This was very similar to the Denver example, however, since I had to located the children of my initial search item in BeautifulSoup, I had to account for this by adding an additional loop:

```
# Connecting to MongoDB, and establishing a
# connection to the news database and the
# longmont collection:
client = MongoClient()
db = client['news']
coll = db['longmont']

# This loop searches through the BeautifulSoup
# object divs and pulls out each story, and
# extracts the title, url, and date from them
# and then inserts these three items into
# the MongoDB collection connected from
# above:
divs = soup.find('ul', {'class': 'list-main'})
for d in divs:
        news = divs.findChildren('li')
        for n in news:
                link = n.find('a')
                title = link.text
                date = pd.to_datetime(n.find('p', {'class': 'item-date'}).text)
                href = BASE_URL + link.get('href')
                coll.insert_one({'date': date,
                                                'link': href,
                                                'title': title})
                #print(title)
                #print(date)
                #print(href)
client.close()
```

Finally, I added one last loop to allow my code to scrape the news data from several of the Longmont City websites news pages. Initially, I was just scraping the 4th page of news (or the 2nd, in the beginning). I limited this scrape to the first 4 news pages, since that should show that my scraper does work. I also inserted this information into MongoDB.

```
client = MongoClient()
db = client['news']
coll = db['longmont']

PAGE_URL = 'https://www.longmontcolorado.gov/news/-npage-{}'

# This code performs largely the same as the previous
# code block, however, this code employs a loop
# that loops through the first 3 pages of the Longmont
# city news feed.
for page in tqdm_notebook(range(1,5)):
        divs = soup.find('ul', {'class': 'list-main'})
        for d in divs:
                news = divs.findChildren('li')
                for n in news:
                        link = n.find('a')
                        title = link.text
                        date = pd.to_datetime(n.find('p', {'class':
                                'item-date'}).text)
                        href = BASE_URL + link.get('href')
                        coll.insert_one({'date': date,
                                                        'link': href,
                                                        'title': title})
```

To make sure I had data in my new database, I queried the database with a `find_one()` command.

```
client = MongoClient()
db = client['news']
coll = db['longmont']

# To make sure the collection has
# entries in it and they look right,
# I queried the database to return
# a single entry.
pprint(coll.find_one())
```

I also performed one other query to enable me to save several of the database entries into a dataframe. The query was simply to show the first ten objects in my database; I added no other search parameters.

```
sort = coll.find({},{"date":1, "link":1, "title":1}).limit(10)
df = pd.io.json.json_normalize(sort)
df
```

4 Results and Output

First, here is the result of calling the MongoDB database for the `www.denver.gov` and displaying the first entry:

Table 9: The results of `find_one()` on the data scraped from `www.denver.gov`

`find_one()`
{'_id': ObjectId('5ee27867ffe6e51ec0f3a45e'), 'date': datetime.datetime(2020, 6, 11, 0, 0), 'link': 'https://www.denvergov.org/content/denvergov/en/denver-board-of-ethics/newsroom/2020/board-meeting-2020.html', 'title': 'Board Meeting: July 8, 2020'}

Similar to the `find_one` call for the `www.denver.gov` data, here is the result for the `longmontcolorado.gov` scraped data:

Table 10: The results of `find_one()` on the data scraped from `www.longmontcolorado.gov`

`find_one()`
{'_id': ObjectId('5ee109028a7c0daa7c9913ec'), 'date': datetime.datetime(2020, 5, 18, 17, 30), 'link': 'https://www.longmontcolorado.gov/Home/Components/News/News/10996/3?npage=2', 'title': 'Los servicios comienzan a reabrir en los parques de Longmont'}

Finally, I created a simple MongoDB query to return a limited amount of my Longmont database entries. I then created a dataframe using Pandas; the results are displayed below.

Table 11: A dataframe created from a MongoDB query to return example database entries from the `www.longmontcolorado.gov` scraped data.

date	link	title
2020-05-18 17:13:00	https://www.longmontcolorado.gov/Home/...	Amenities Begin Reopening in...
2020-05-15 14:45:00	https://www.longmontcolorado.gov/Home/...	Waste Services Unveils Sustainably...
2020-05-15 12:21:00	https://www.longmontcolorado.gov/Home/...	Longmont Peace Officer Memorial
2020-05-15 10:32:00	https://www.longmontcolorado.gov/Home/...	This Week in Longmont - May 15, 2020
2020-05-14 15:30:00	https://www.longmontcolorado.gov/Home/...	Transfort Resumes Partial and ...
2020-05-12 13:20:00	https://www.longmontcolorado.gov/Home/...	Longmont Museum Summer Camps ...

5 Analysis

Web scraping on the surface looked very easy. However, once I began to scrape my first web pages I realized how difficult it can be. Every web page is different, and while it may look obvious where to focus your attention on to extract data, it can be difficult to find precisely how to extract that data. Although part of me is speaking as someone with virtually no experience working with HTML, and I imagine any amount of experience working with HTML will make scraping easier.

Additionally, I began to feel more comfortable working with BeautifulSoup as I worked through this exercise, and I can see it helping a great deal in locating the extractable data. From the reading I did to answer this weeks written questions, it seems as though there are other tools similar to BeautifulSoup, so I imagine these will represent help for specific web scraping tasks.

6 Conclusion

What started off looking like an easy assignment was actually, for me, fairly difficult. It was evident after beginning my own web scrape how different websites are from each other. Some have wonderfully organized and easy to read HTML, while others have code that I couldn't even begin to figure out how to use. That was largely my struggle with this assignment; as someone with almost no web programming experience, I didn't know how to look at the HTML code to ascertain what tags I needed. I spent a day going through web scraping tutorials and managed to get a better idea. I also fortunately have many programming friends, and had to ask a little guidance from one who is a web developer when I was stuck on creating my news item array. While I see web scraping as a very valuable tool, I wonder if certain data scientists focus particularly on this, since it seems rather difficult (and has a learning curve), or if it just gets way easier with time and experience.

After reading about the ethics and legality of web scraping, I am now a bit afraid to use it. While I always perform due-diligence and make sure anything I do is legal and ethical, it seems that there is still a great deal of gray area around what unethical web scraping actually is. Some uses are obviously unethical, such as in cases of mass data scraping for profit. I am sure within the next few decades it will become more clear how (or if) to use web scraping. My original degrees were in Biology and Chemistry, and one of the things I notice about Computer

Science is how new it is comparatively speaking. As such a new science, it frequently feels like ethics and law are still being worked out at a basic level. Even if the ethics and law of Computer Science was somehow able to be established quickly after its discovery, that leaves its sub-disciplines to work out their ethics and law (this base law does not mean every sub-discipline will have the same "base law" exclusively, after all). Since Data Science is one of the newer branches of Computer Science, it seems like many times when we learn a new technology we do have to fully consider the ethical ramifications, as they have not fully been laid out, discussed, experienced, or encountered.

Lastly, I have caught myself thinking of "this is data on the internet, everyone can see it, clearly it is available for my use in any way I want to use it." I now realize that this is dangerous, since in no way does viewing something represent ownership (this doesn't work outside of computer systems, after all). While data on the internet doesn't feel like real, physical data in front of you, many of the laws that pertain to ownership still pertain to websites. For example, copyright infringement is a very relevant problem for web scrapers, as is trespassing. I had never considered that you could trespass on another person computer, but after reading several unethical web scraping examples, I can now see how.

7 References

1. Bernard, B. (2017, April 24). *Web scraping and crawling are perfectly legal, right?* Benoit Bernard. `https://benbernardblog.com/web-scraping-and-crawling-are-perfectly-legal-right/`

2. Chapman, R. (2020, February 11). *Top 10 web scraping techniques.* LimeProxies. `https://limeproxies.com/blog/top-10-web-scraping-techniques/`

3. Densmore, J. (2019, July 23). *Ethics in web scraping.* Medium. `https://towardsdatascience.com/ethics-in-web-scrapi`

4. Koshy, J. (2016, August 19). *5 technologies to master if you want to scrape the web.* PromptCloud. `https://www.promptcloud.com/blog/technologies-for-web-scraping/`

5. Roberts, E. (2020, January 27). *Is web scraping illegal? Depends on what the meaning of the word is.* Blog. `https://www.imperva.com/blog/is-web-scraping-illegal/`

6. Web scraping. (2005, September 17). *Wikipedia, the free encyclopedia.* Retrieved June 12, 2020, from `https://en.wikipedia.org/wiki/Web_scraping`

7. *What is web scraping and how does web crawling work?* (2020, January 16). Scrapinghub. `https://www.scrapinghub.com/what-is-web-scraping`