Week 6 Technical Lab

Web Scraping

Trevor Thomas

College of Computer and Information Sciences, Regis University

MSDS 610: Data Engineering

Paul Andrus

June 14, 2020

Overview and Objectives

Collecting, cleaning, and formatting data is often one of the most tedious parts of the data science project lifecycle. One of the many methods a data scientist or engineer can use in this early part of the process is web scraping, or the automation of extracting data from a website. With so much data produced by and available on the web, it is an obvious source of material for data scientists and analysts – unfortunately, however, web pages are typically meant for viewing as opposed to data collection, so obtaining the data is not as simple as clicking "download." Web scraping utilizes scripts to pull the HTML source code for a website, parse it, search for desired elements, and ultimately save them to a file or a database.

The objective of this lab is to use a Python script to scrape a Denver Government News website for the title, date, and URL of each news story and store them in a MongoDB database. The tools I will use are Python, various Python libraries such as `BeautifulSoup` and `requests`, and MongoDB. First, I will use the Python interpreter to build the elements of the Python script interactively to better understand how they work and fit together. Then I will create the .py file that will hold the script and run it in my terminal. Finally, I will use the MongoDB shell to query the database and ensure the scraping was successful. Each of these steps is outlined in detail below with a discussion to follow.

Methods and Results

Scrape and Explore Data with Python Interpreter

Before building and running the full script that will ultimately load the Denver News data into a database, I used a Python shell to work with the code interactively. I launched the interpreter with `python` and imported the `requests` and `BeautifulSoup` libraries. Then I needed to obtain the HTML code from the web page and clean it into a more usable format. I used the `requests` library `.get` method to load the HTML into a variable called "page," and then I used the `BeautifulSoup` library to format it in a variable called "clean_page."

```
(base) dpredbeard@dpred:~$ python
Python 3.7.6 | packaged by conda-forge | (default, Jan  7 2020, 20:28:53)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> import requests as req
>>> from bs4 import BeautifulSoup as bs
>>> page = req.get('https://www.colorado.gov/news')
>>> clean_page = bs(page.content)
```

With the HTML loaded in a clean format, I then needed to determine the tag used for each news story element and load those into a new variable. Using Firefox's "inspect element" feature, I determined the stories were under `<div>` tags of class "view-page-item." I used the `.find_all` method to load these into a variable called "divs." After, I called "divs" with a [0] index to ensure this worked properly.

```
>>> divs = clean_page.find_all('div', {'class' : 'view-page-item'})

>>> divs[0]

<div class="view-page-item">
<div class="date margin-b">
<span class="field-content"><span class="date-display-single" content="2020-
06-11T00:00:00-06:00" datatype="xsd:dateTime" property="dc:date">Thursday,
June 11, 2020</span></span></div>
<h3 class="no-padding x-small lighter-text"><span class="field-content"><a
href="https://www.colorado.gov/pacific/cdle/news/press-release-state-labor-
dept-22-billion-unemployment-benefits-paid-march-29th">Press Release: State
Labor Dept: $2.2 Billion in Unemployment Benefits Paid Since March 29th</a></
span></h3>
<div class="margin-b lighter-text smaller-text-views">
<span class="field-content">From Department of Labor and Employment
</span></div>
</div>

>>> divs[0].text

'\n\nThursday, June 11, 2020\nPress Release: State Labor Dept: $2.2 Billion
in Unemployment Benefits Paid Since March 29th\n\nFrom Department of Labor
and Employment  \n'
```

This entry matches the top entry from the web page, so it appears everything worked as expected.

Before moving on to pull all of the news stories into the database, I used `.find` to pull the date, title, and URL for each item, as these will be the data that ultimately go into MongoDB. Again using Firefox's "inspect element," I determined the format for each item so I knew what to pass to `.find()`.

```
>>> divs[0].find('div', {'class' : 'date margin-b'}).text
'\nThursday, June 11, 2020'

>>> divs[0].find('a')
<a href="https://www.colorado.gov/pacific/cdle/news/press-release-state-
labor-dept-22-billion-unemployment-benefits-paid-march-29th">Press Release:
State Labor Dept: $2.2 Billion in Unemployment Benefits Paid Since March
29th</a>

>>> divs[0].find('a').text
'Press Release: State Labor Dept: $2.2 Billion in Unemployment Benefits Paid
Since March 29th'

>>> divs[0].find('a').get('href')
'https://www.colorado.gov/pacific/cdle/news/press-release-state-labor-dept-
22-billion-unemployment-benefits-paid-march-29th'
```

Per the recommendation in Dr. Nate George's instructional video, I opted to convert the date data type to a datetime object using `pandas`. I imported the library and used `pd.to_datetime`, testing it with the `.month` method.

```
>>> import pandas as pd

>>> pd.to_datetime(divs[0].find('div', {'class' : 'date margin-b'}).text)
Timestamp('2020-06-11 00:00:00')

>>> pd.to_datetime(divs[0].find('div', {'class' : 'date margin-
b'}).text).month
6
```

Use For Loops to Scrape Full Site

Most of the code needed for this step was already written above in the Python interpreter, but I still needed to use two `for` loops to pull each individual news story from each page of the website. The "inner" `for` loop will go through the stories on each page, and the outer loop will cycle through the pages. Still using the interpreter, I built the first test loop to sort through "divs" and save the date, title, and web address to variables called "date," "title," and "address," respectively. I then printed the contents of those variables, and they matched the final entry from the first page of the website, indicating the loop worked.

```
>>> for d in divs:
...     date = pd.to_datetime(d.find('div', {'class' : 'date margin-
      b'}).text)
...     link = d.find('a')
...     title = link.text
...     address = link.get('href')
...

>>> date
Timestamp('2020-06-04 00:00:00')

>>> title
'Colorado Commissioner of Agriculture Statement on the Killing of George
Floyd'

>>> address
'https://www.colorado.gov/pacific/agmain/news/colorado-commissioner-
agriculture-statement-killing-george-floyd'
```

To make the outer loop cycle through the various pages of news stories, I had to determine the website's format for indicating a new page. Following the URL, each page has an added `?page=x` at the end, where x is the page number starting with 0 (so the first page is `?page=0`, second page is `?page=1`, etc.). There were 81 pages of news stories to loop through, so I used the `range()` function combined with the `.format()` method to pull each page.

```
website = 'https://www.colorado.gov/news?page={}'
for i in range(81):
      website_f = website.format(i)
      page = req.get(website_f)
      clean_page = bs(page.content)
      divs = clean_page.find_all('div', {class' : 'view-page-item'})
```

Load Data into MongoDB with Python Script

The only steps left were to import the MongoClient() from pymongo, initialize the database and the collection, add a line of code to the inner `for` loop that adds the date, title, and URL to the database, and then run the script. The final script was as follows:

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

client = MongoClient()
db = client['News']
collection = db['Den_Gov']
```

```
website = 'https://www.colorado.gov/news?page={}'

for i in range(80):
      website_f = website.format(i)
      page = req.get(website_f)
      clean_page = bs(page.content)
      divs = clean_page.find_all('div', {class' : 'view-page-item'})

      for d in divs:
            date = pd.to_datetime(d.find('div', {'class' : 'date margin-
                  b'}).text)
            link = d.find('a')
            title = link.text
            address = link.get('href)

            collection.insert_one({'date' : date, 'URL' : address, 'title' :
                  title})

client.close()
```

I saved this as den_news_scrape.py and ran it in my terminal with the `python` command. Once it finished running, I used `mongo` to open the MongoDB shell and entered `show dbs`. The new "News" database appeared with the Den_Gov collection I had created. I then used `.count()` to see how many records were saved and `.findOne()` to see if the first record matched the top story on the website.

```
(base) dpredbeard@dpred:~$ mongo

> show dbs
News            0.000GB
admin           0.000GB
config          0.000GB
local           0.000GB
parking         1.170GB
weather_test    0.000GB
weather_test_2  0.000GB

> use News
switched to db News

> show collections
Den_Gov

> db.Den_Gov.count()
809

> db.Den_Gov.findOne()
{
      "_id" : ObjectId("5ee6c063b2662bd7965cf2f0"),
      "date" : ISODate("2020-06-11T00:00:00Z"),
      "URL" : "https://www.colorado.gov/pacific/cdle/news/press-release-
state-labor-dept-22-billion-unemployment-benefits-paid-march-29th",
      "title" : "Press Release: State Labor Dept: $2.2 Billion in
Unemployment Benefits Paid Since March 29th"
}
```

The `.count()` function returned 809, which was exactly what I expected – there were 81 pages with 10 records per page, with the last page only containing 9 records. The `.findOne()` function returned the correct record as well. At this stage, I used the MongoDB shell to query the database a bit.

```
> db.Den_Gov.find().sort({'date' : 1}).limit(3).pretty();
{
      "_id" : ObjectId("5ee6c091b2662bd7965cf618"),
      "date" : ISODate("2015-01-23T00:00:00Z"),
      "URL" : "https://www.colorado.gov/pacific/dpa/news/capitol-complex-
master-plan-released",
      "title" : "Capitol Complex Master Plan released"
}
{
      "_id" : ObjectId("5ee6c091b2662bd7965cf617"),
      "date" : ISODate("2015-02-13T00:00:00Z"),
      "URL" : "https://www.colorado.gov/pacific/dpa/news/state-controller
%E2%80%99s-office-rulemaking",
      "title" : "State Controller's Office Rulemaking"
}
{
      "_id" : ObjectId("5ee6c091b2662bd7965cf616"),
      "date" : ISODate("2015-03-13T00:00:00Z"),
      "URL" : "https://www.colorado.gov/pacific/dpa/news/division-human-
resources-risk-management-rulemaking",
      "title" : "Division of Human Resources Risk Management Rulemaking "
}
```

This showed the three oldest records – the oldest story, "Capitol Complex Master Plan released" from January 23, 2015, matches the final record on the 81st page. Next, I wanted to test the database, so I went to the 41st page of the website and found the top story was titled "Workshops Across State to Detail Energy Savings Opportunities for Colorado Farmers and Ranchers." I used the `.find()` function to search for this record and confirmed the date, title, and URL match that record. At this point, I was confident the scraping was successful.

```
> db.Den_Gov.find({'title' : 'Workshops Across State to Detail Energy Savings
Opportunities for Colorado Farmers and Ranchers'})

{ "_id" : ObjectId("5ee6c07ab2662bd7965cf480"), "date" : ISODate("2016-11-
18T00:00:00Z"), "URL" :       "https://www.colorado.gov/pacific/energyoffice/
news/workshops-across-state-detail-energy-savings-opportunities-colorado-
farmers-and-ranchers", "title" : "Workshops Across State to Detail Energy
Savings Opportunities for Colorado Farmers and Ranchers" }
```

Discussion

Web scraping is an integral tool for data scientists' ability to obtain new and interesting data upon which they can apply models. The objective of this assignment was to begin using this tool by scraping data from the Denver Government News website and saving it to a MongoDB database. I initially used a Python shell to run commands one at a time and learn how the `BeautifulSoup` library and its objects work. Then I wrote the Python script to iterate through the HTML of each news story on each individual page and pull the relevant objects, using pymongo to ultimately save each item to the MongoDB database. Finally, I used the MongoDB shell to execute queries on the new data to determine whether the process was a success. Each query returned the expected results, indicating everything worked as planned.

Prior to this lab, I had never used a script to scrape a website before, so this relatively straightforward assignment was eye-opening and, candidly, exciting. It had a particular impact on me because of an experience I had when working on my thesis for my undergraduate degree in Economics. I was using logistic regression to predict the success of various mergers and acquisitions based on the change in executive pay from pre to post merger, and all my data had to come from financial filings on SEC.gov. This was not a user-friendly site, and the vast majority of my research time was spent combing through large pdfs for individual numbers on filings, plugging them into an Excel spreadsheet, and repeating. Automating that process would have been more difficult than this week's assignment, but it absolutely could have been done. Had I known about this possibility, I could have easily grown my data-set well beyond my modest 60 observations. During my transition to data science over the past few years, I have been constantly bothered by wondering where I will find new data. Kaggle.com is fantastic, but the interesting data-sets have already been analyzed by individuals far beyond my current skill-set. I can already tell this new method will be one of my primary sources of data going forward, and I very much look forward to using it.