How to scrape websites with Python and BeautifulSoup

Justin Yek
Jun 10, 2017 · 8 min read

There is more information on the Internet than any human can absorb in a lifetime. What you need is not access to that information, but a scalable way to collect, organize, and analyze it.

You need web scraping.

Web scraping automatically extracts data and presents it in a format you can easily make sense of. In this tutorial, we'll focus on its applications in the financial market, but web scraping can be used in a wide variety of situations.

If you're an avid investor, getting closing prices every day can be a pain, especially when the information you need is found across several webpages. We'll make data extraction easier by building a web scraper to retrieve stock indices automatically from the Internet.

Getting Started

We are going to use Python as our scraping language, together with a simple and powerful library, BeautifulSoup.

- For Mac users, Python is pre-installed in OS X. Open up Terminal and type `python --version` . You should see your python version is 2.7.x.
- For Windows users, please install Python through the official website.

Next we need to get the BeautifulSoup library using `pip` , a package management tool for Python.

In the terminal, type:

```
easy_install pip
pip install BeautifulSoup4
```

Note: If you fail to execute the above command line, try adding `sudo` in front of each line.

The Basics

Before we start jumping into the code, let’s understand the basics of HTML and some rules of scraping.

HTML tags

If you already understand HTML tags, feel free to skip this part.

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <h1> First Scraping </h1>
        <p> Hello World </p>
    <body>
</html>
```

This is the basic syntax of an HTML webpage. Every `<tag>` serves a block inside the webpage:
1. `<!DOCTYPE html>`: HTML documents must start with a type declaration.
2. The HTML document is contained between `<html>` and `</html>`.
3. The meta and script declaration of the HTML document is between `<head>` and `</head>`.
4. The visible part of the HTML document is between `<body>` and `</body>` tags.
5. Title headings are defined with the `<h1>` through `<h6>` tags.
6. Paragraphs are defined with the `<p>` tag.

Other useful tags include `<a>` for hyperlinks, `<table>` for tables, `<tr>` for table rows, and `<td>` for table columns.

Also, HTML tags sometimes come with `id` or `class` attributes. The `id` attribute specifies a unique id for an HTML tag and the value must be unique within the HTML document. The `class` attribute is used to define equal styles for HTML tags with the same class. We can make use of these ids and classes to help us locate the data we want.

For more information on HTML tags, id and class, please refer to W3Schools Tutorials.

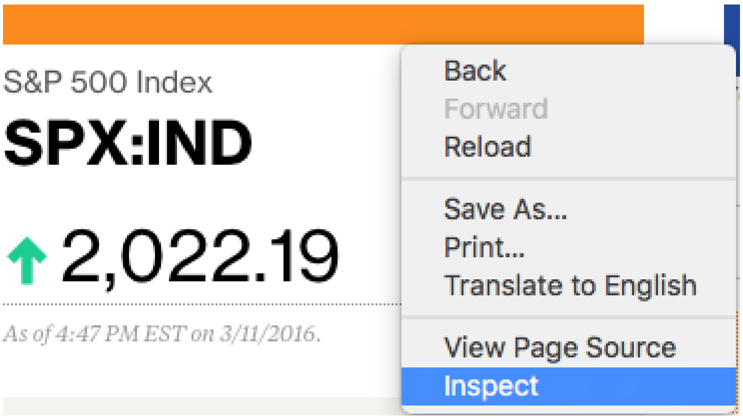
Scraping Rules

1. You should check a website's Terms and Conditions before you scrape it. Be careful to read the statements about legal use of data. Usually, the data you scrape should not be used for commercial purposes.
2. Do not request data from the website too aggressively with your program (also known as spamming), as this may break the website. Make sure your program behaves in a reasonable manner (i.e. acts like a human). One request for one webpage per second is good practice.
3. The layout of a website may change from time to time, so make sure to revisit the site and rewrite your code as needed

Inspecting the Page

Let's take one page from the Bloomberg Quote website as an example.

As someone following the stock market, we would like to get the index name (S&P 500) and its price from this page. First, right-click and open your browser's inspector to inspect the webpage.



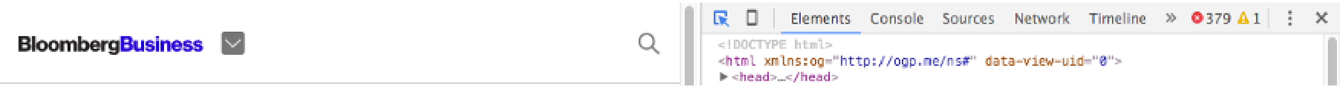
S&P 500 Index

SPX:IND

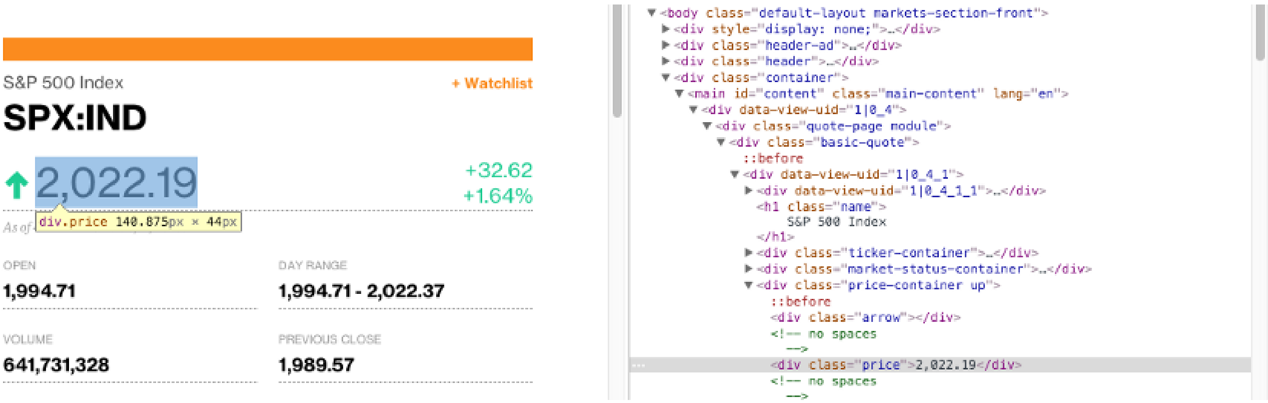
2,022.19

As of 4:47 PM EST on 3/11/2016.

Try hovering your cursor on the price and you should be able to see a blue box surrounding it. If you click it, the related HTML will be selected in the browser console.



```
<!DOCTYPE html>
<html xmlns:og="http://ogp.me/ns#" data-view-uid="0">
▶<head>…</head>
```



S&P 500 Index

+ Watchlist

SPX:IND

2,022.19

+32.62
+1.64%

As of div.price 140.875px × 44px

OPEN
1,994.71

DAY RANGE
1,994.71 - 2,022.37

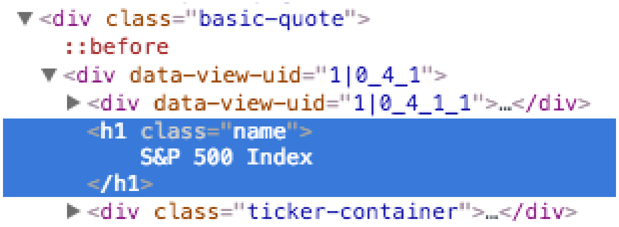
VOLUME
641,731,328

PREVIOUS CLOSE
1,989.57

```
▼<body class="default-layout markets-section-front">
  ▶<div style="display: none;">…</div>
  ▶<div class="header-ad">…</div>
  ▶<div class="header">…</div>
  ▼<div class="container">
    ▼<main id="content" class="main-content" lang="en">
      ▼<div data-view-uid="1|0_4">
        ▼<div class="quote-page module">
          ▼<div class="basic-quote">
              ::before
            ▼<div data-view-uid="1|0_4_1">
              ▶<div data-view-uid="1|0_4_1_1">…</div>
                <h1 class="name">
                    S&P 500 Index
                </h1>
              ▶<div class="ticker-container">…</div>
              ▶<div class="market-status-container">…</div>
              ▼<div class="price-container up">
                  ::before
                  <div class="arrow"></div>
                  <!-- no spaces
                    -->
                  <div class="price">2,022.19</div>
                  <!-- no spaces
                    -->
```

From the result, we can see that the price is inside a few levels of HTML tags, which is `<div class="basic-quote">` → `<div class="price-container up">` → `<div class="price">`.

Similarly, if you hover and click the name “S&P 500 Index”, it is inside `<div class="basic-quote">` and `<h1 class="name">`.



```
▼<div class="basic-quote">
    ::before
  ▼<div data-view-uid="1|0_4_1">
    ▶<div data-view-uid="1|0_4_1_1">…</div>
      <h1 class="name">
          S&P 500 Index
      </h1>
    ▶<div class="ticker-container">…</div>
```

Now we know the unique location of our data with the help of `class` tags.

Jump into the Code

Now that we know where our data is, we can start coding our web scraper. Open your text editor now!

First, we need to import all the libraries that we are going to use.

```
# import libraries
import urllib2
from bs4 import BeautifulSoup
```

Next, declare a variable for the url of the page.

```
# specify the url
quote_page = ‘http://www.bloomberg.com/quote/SPX:IND'
```

Then, make use of the Python urllib2 to get the HTML page of the url declared.

```
# query the website and return the html to the variable ‘page’
page = urllib2.urlopen(quote_page)
```

Finally, parse the page into BeautifulSoup format so we can use BeautifulSoup to work on it.

```
# parse the html using beautiful soup and store in variable `soup`
soup = BeautifulSoup(page, ‘html.parser’)
```

Now we have a variable, `soup` , containing the HTML of the page. Here’s where we can start coding the part that extracts the data.

Remember the unique layers of our data? BeautifulSoup can help us get into these layers and extract the content with `find()` . In this case, since the HTML class `name` is unique on this page, we can simply query `<div class="name">` .

```
# Take out the <div> of name and get its value
name_box = soup.find(‘h1’, attrs={‘class’: ‘name’})
```

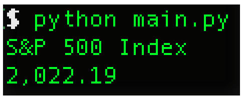
After we have the tag, we can get the data by getting its `text` .

```
name = name_box.text.strip() # strip() is used to remove starting
and trailing
print name
```

Similarly, we can get the price too.

```
# get the index price
price_box = soup.find(‘div’, attrs={‘class’:’price’})
price = price_box.text
print price
```

When you run the program, you should be able to see that it prints out the current price of the S&P 500 Index.



```
$ python main.py
S&P 500 Index
2,022.19
```

Export to Excel CSV

Now that we have the data, it is time to save it. The Excel Comma Separated Format is a nice choice. It can be opened in Excel so you can see the data and process it easily.

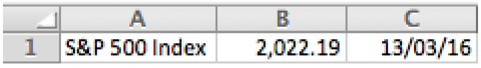
But first, we have to import the Python csv module and the datetime module to get the record date. Insert these lines to your code in the import section.

```
import csv
from datetime import datetime
```

At the bottom of your code, add the code for writing data to a csv file.

```
# open a csv file with append, so old data will not be erased
with open(‘index.csv’, ‘a’) as csv_file:
 writer = csv.writer(csv_file)
 writer.writerow([name, price, datetime.now()])
```

Now if you run your program, you should able to export an `index.csv` file, which you can then open with Excel, where you should see a line of data.



	A	B	C
1	S&P 500 Index	2,022.19	13/03/16

So if you run this program everyday, you will be able to easily get the S&P 500 Index price without rummaging through the website!

Going Further (Advanced uses)

Multiple Indices

So scraping one index is not enough for you, right? We can try to extract multiple indices at the same time.

First, modify the `quote_page` into an array of URLs.

```
quote_page = [‘http://www.bloomberg.com/quote/SPX:IND',
‘http://www.bloomberg.com/quote/CCMP:IND']
```

Then we change the data extraction code into a `for` loop, which will process the URLs one by one and store all the data into a variable `data` in tuples.

```
# for loop
data = []
for pg in quote_page:
 # query the website and return the html to the variable ‘page’
 page = urllib2.urlopen(pg)

# parse the html using beautiful soap and store in variable `soup`
 soup = BeautifulSoup(page, ‘html.parser’)

# Take out the <div> of name and get its value
 name_box = soup.find(‘h1’, attrs={‘class’: ‘name’})
 name = name_box.text.strip() # strip() is used to remove starting
and trailing

# get the index price
 price_box = soup.find(‘div’, attrs={‘class’:’price’})
 price = price_box.text

# save the data in tuple
 data.append((name, price))
```

Also, modify the saving section to save data row by row.

```
# open a csv file with append, so old data will not be erased
with open(‘index.csv’, ‘a’) as csv_file:
 writer = csv.writer(csv_file)
 # The for loop
 for name, price in data:
 writer.writerow([name, price, datetime.now()])
```

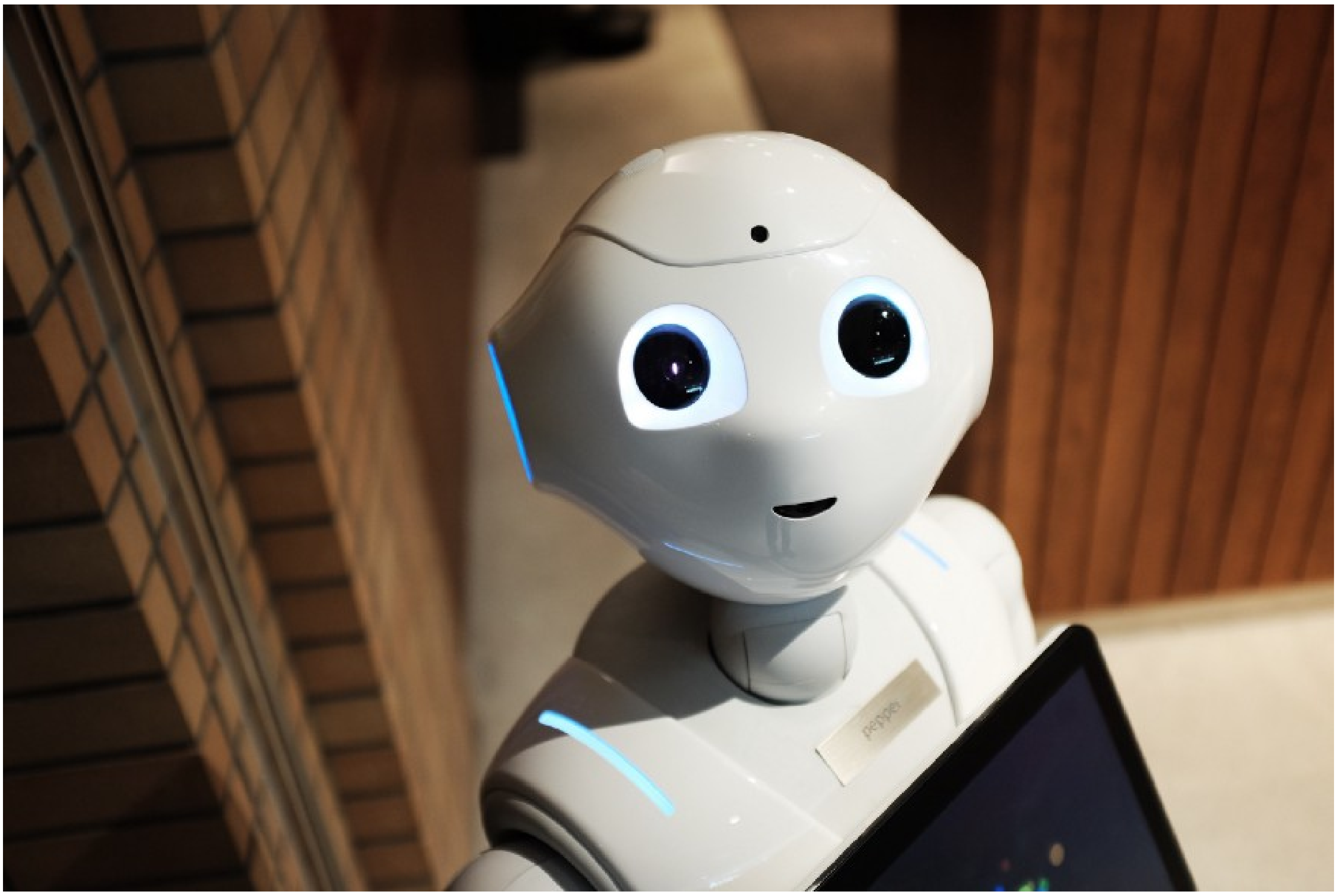
Rerun the program and you should be able to extract two indices at the same time!

Advanced Scraping Techniques

BeautifulSoup is simple and great for small-scale web scraping. But if you are interested in scraping data at a larger scale, you should consider using these other alternatives:

1. Scrapy, a powerful python scraping framework
2. Try to integrate your code with some public APIs. The efficiency of data retrieval is much higher than scraping webpages. For example, take a look at Facebook Graph API, which can help you get hidden data which is not shown on Facebook webpages.
3. Consider using a database backend like MySQL to store your data when it gets too large.

Adopt the DRY Method



DRY stands for “Don’t Repeat Yourself”, try to automate your everyday tasks like this person. Some other fun projects to consider might be keeping track of your Facebook friends’ active time (with their consent of course), or grabbing a list of topics in a forum and trying out natural language processing (which is a hot topic for Artificial Intelligence right now)!

If you have any questions, please feel free to leave a comment below.

References

http://www.gregreda.com/2013/03/03/web-scraping-101-with-python/

http://www.analyticsvidhya.com/blog/2015/10/beginner-guide-web-scraping-beautiful-soup-python/

This article was originally published on Altitude Labs' blog and was written by our software engineer, Leonard Mok. Altitude Labs is a software agency that specializes in personalized, mobile-first React apps.

Web Development Software Development Python Programming Life Hacking