**Scrape Tabular Data with Python**

**How to scrape NBA players’ data using BeautifulSoup, Selenium, and Pandas in Python.**

https://towardsdatascience.com/scrape-tabular-data-with-python-b1dd1aeadfad

One of the bottlenecks in executing a machine learning project is the dataset assembling.

The ways of data assembling vary a lot with the type of data, of which scraping the tabular datasets from the web is one of the most typical sources. I’ve used it for a long time to get as much data as I need efficiently.

I have been writing about machine learning techniques for a while using the NBA players’ stats as the raw data. One of the most frequently asked questions to me is whether I could share the data because people would like to play with it.

There’s an old saying in Asia,

Give a man a fish, and you feed him for a day; teach a man to fish, and you feed him for a lifetime.

So, in this post, I would like to share with you how to scrape tabular data from the web with Python.

Besides the standard steps, I will also introduce the practical **issues** I met and the **solutions** to solve them.

**Prepare the tools**

pip install beautifulsoup4   
pip install selenium  
pip install requests  
pip install pandas

The codes above show the installation of four required python packages to do the job, and most of the time they are sufficient.

After successfully installing these packages, simply import them into the python environment.

import requests  
from bs4 import BeautifulSoup  
from selenium import webdriver  
import pandas as pd

If you are using *Google Chrome* as your default browser, please make sure that the ***chromedriver*** is executable in your ***PATH***. If not, you may run into the same problem as I did, with the error message below.

Message: ‘chromedriver’ executable needs to be in PATH

To solve the problem, simply download the ***chromedriver*** from <https://sites.google.com/a/chromium.org/chromedriver/home> and put it in an executable path or adding its current location to the system PATH variable.

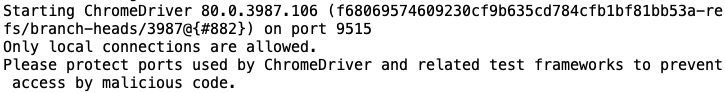
To add its location to PATH, open the ~/.bash\_profile by typing

vi ~/.bash\_profile

Then add the following line to the end of the file,

# add chromedriverexport PATH="/Users/yourdirectorytoCHROMEDIRVER:$PATH"

To check whether ***chromedriver*** is executable, just type *chromedriver* in the terminal and you can see the message from the package,



run chromedriver by [Yufeng](https://medium.com/@jianan.jay.lin)

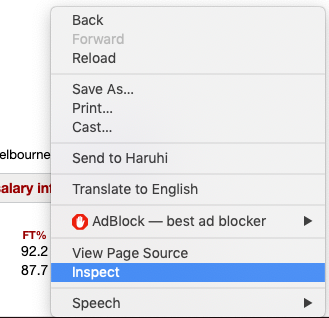
Till now, we have all the tools required.

**Inspect the webpage**

First, we need to find a target page. I will use the NBA player’s stats page from [***basketball-reference***](https://www.basketball-reference.com/) as the example of this tutorial. Here is the URL of my target page:

URL = <https://www.basketball-reference.com/players/i/irvinky01.html>

To scrape data from a specific webpage, we need to know its structure via a developer’s view. Right-click the page and left-click the ‘***Inspect***’ button as below.



Inspect view of the page by [Yufeng](https://medium.com/@jianan.jay.lin)

Then you will see the script of the webpage on the right side as shown below.



Inspect view of the page by [Yufeng](https://medium.com/@jianan.jay.lin)

The information shown above is hardly human-readable but after carefully checking, you may find some patterns.

For example, the tables on the page always start with **<table …>** and end with **</table>** (highlighted by blue in the plot above). And these tables are exactly what I want from the page.

**Scrape the tables**

Now, we are going to scrape those tables from the page using Beautifulsoup. The standard way of getting all the tables from the page is,

page = requests.get(URL)  
soup = BeautifulSoup(page.content, 'html.parser')  
tables = soup.find\_all("table")

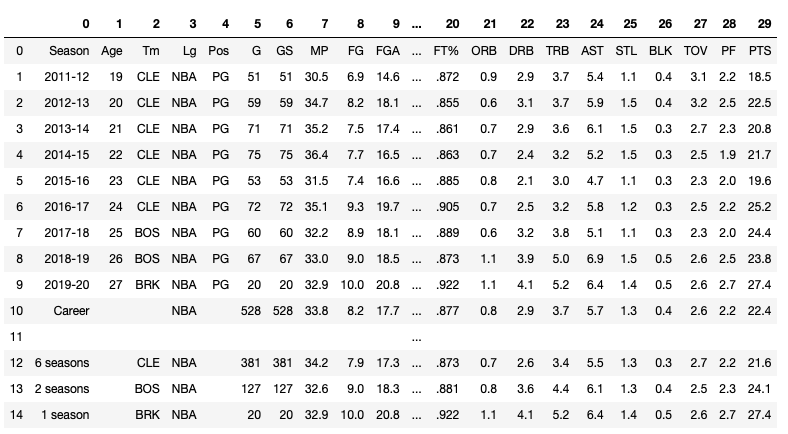
where ***requests.get(URL)*** is basically getting the information from the page and ***BeautifulSoup(page.content, ‘html.parser’)*** is to parse the information.

We then can apply the ***find\_all*** function to the parsed information in ***soup***. ***soup.find\_all(“table”)*** is collecting all the blocks of information that start with **<table>** and end with **</table>**.

For each of the *table* in the variable **tables**, usually, the table headers start with **<th>** and all the table cells of the rows start with **<td>**. So, the table can be extracted and converted to **pandas** data frame in the following code.

table = tables[0]  
tab\_data = [[cell.text for cell in row.find\_all(["th","td"])]  
 for row in table.find\_all("tr")]  
df = pd.DataFrame(tab\_data)

The generated data frame is as below,



df from beautifulsoup by [Yufeng](https://medium.com/@jianan.jay.lin)

To move the first row to the headers, simply type

df.columns = df.iloc[0,:]  
df.drop(index=0,inplace=True)

To get all the tables of the page in the same way as the first table (**tables[0]**), I created a dictionary and use the attribute ‘id’ of each table as the key within the for-loop.

tabs\_dic = {}  
   
for table in tables:  
 tab\_name = table['id']  
   
 tab\_data = [[cell.text for cell in row.find\_all(["th","td"])]  
 for row in table.find\_all("tr")]  
 df = pd.DataFrame(tab\_data)  
 df.columns = df.iloc[0,:]  
 df.drop(index=0,inplace=True)  
   
 #df = df.loc[df.Season != ""]  
 tabs\_dic[tab\_name] = df

I was able to extract the table id by **table[‘id’]** because *‘id’* is an attribute of the table, which is ***‘per\_game’*** as shown below,

https://miro.medium.com/max/60/1*ySopNAE-SQ0sMgXKklwpkw.png?q=20

https://miro.medium.com/max/623/1*ySopNAE-SQ0sMgXKklwpkw.png

table attributes by [Yufeng](https://medium.com/@jianan.jay.lin)

In this way, I should have all the tables in *pandas data frame* format deposited in the large dictionary **tabs\_dic**.

However, I met a problem…

**The issue of scraping multiple tables**

When I first tried to investigate the table numbers on the player’s page, I found this issue where I only scraped **ONE** table from it!!!

I did use **find\_all(“table”)** as shown in the previous code, how can I miss all the other tables except the first one?!

I checked the types of the tables on the page and I found that the tables that escaped from my scraping were those within javascript.

Then, selenium was used to solve the problem.

driver = webdriver.Chrome()  
driver.get(URL)  
soup = BeautifulSoup(driver.page\_source,'html')  
driver.quit()  
tables = soup.find\_all('table')

The codes above solved the problem and successfully acquired 75 tables then.

I was confused again because 75 seemed too many to me. I checked the webpage over and over again and found even though some of the blocks start with <table>, they only deposit one or two values and are not the tables I want.

So, I further revised the code to select several classes of tables on the page based on the ***class*** attribute of the tables.

tables = soup.find\_all('table',{"class":["row\_summable sortable stats\_table now\_sortable","suppress\_all sortable stats\_table now\_sortable","sortable stats\_table now\_sortable","suppress\_glossary sortable stats\_table now\_sortable"]})

where the ***class*** information was passed to the function in a dictionary format.

I finally got 22 tables in total.

**The function of scraping NBA players’ stats from basketball-reference**

To require the players’ stats is one of my most frequent operations, so I wrote a large function for my future use.

def get\_all\_tabs\_of\_player\_page(URL = '<https://www.basketball-reference.com/players/i/irvinky01.html'>):  
 driver = webdriver.Chrome()  
 driver.get(URL)  
 soup = BeautifulSoup(driver.page\_source,'html')  
 driver.quit()  
 tables = soup.find\_all('table',{"class":["row\_summable sortable stats\_table now\_sortable","suppress\_all sortable stats\_table now\_sortable","sortable stats\_table now\_sortable","suppress\_glossary sortable stats\_table now\_sortable"]})  
 tabs\_dic = {}  
   
 for table in tables:  
 tab\_name = table['id']  
   
 tab\_data = [[cell.text for cell in row.find\_all(["th","td"])] for row in table.find\_all("tr")]  
 df = pd.DataFrame(tab\_data)  
 df.columns = df.iloc[0,:]  
 df.drop(index=0,inplace=True)  
   
 tabs\_dic[tab\_name] = df  
   
 return tabs\_dic

Webpage scraping is a powerful skill, especially for those who are interested to apply machine learning techniques to ***some interesting fields***.

For example, the scraping skill is fundamental to all the following posts I wrote about machine learning and basketball.