Introduction to Scraping Data from Transfermarkt

Before starting the article, I'm obliged to mention that web scraping is a grey area legally and ethicaly in lots of circumstances. Please consider the positive and negative effects of what you scrape before doing so!

Warning over. Web scraping is a hugely powerful tool that, when done properly, can give you access to huge, cl data sources to power your analysis. The applications are just about endless for anyone interested in data. As a professional analyst, you can scrape fixtures and line-up data from around the world every day to plan scoutin assignments or alert you to youth players breaking through. As an amateur analyst, it is quite likely to be your source of data for analysis.

This tutorial is just an introduction for Python scraping. It will take you through the basic process of loading a p locating information and retrieving it. Combine the knowledge on this page with for loops to cycle through a sit and HTML knowledge to understand a web page, and you'll be armed with just about any data you can find.

Let's fire up our modules & get started. We'll need requests (to access and process web pages with Python) anc beautifulsoup (to make sense of the code that makes up the pages) so make sure you have these installed.

In [1]:

```
import requests
from bs4 import BeautifulSoup

import pandas as pd
```

Our process for extracting data is going to go something like this:

1. Load the webpage containing the data.
2. Locate the data within the page and extract it.
3. Organise the data into a dataframe

For this example, we are going to take the player names and values for the most expensive players in a particu
year. You can find the page that we'll use here.

The following sections will run through each of these steps individually.

Load the webpage containing the data

The very first thing that we are going to do is create a variable called 'headers' and assign it a string that will tell
website that we are a browser, and not a scraping tool. In short, we'll be blocked if we are thought to be scrapir

Next, we have three lines. The first one assigns the address that we want to scrape to a variable called 'page'.

The second uses the requests library to grab the code of the page and assign it to 'pageTree'. We use our heade
variable here to inform the site that we are pretending to be a human browser.

Finally, the BeautifulSoup module parses the website code into html. We will then be able to search through th
the data that we want to extract. This is saved to 'pageSoup', and you can find all three lines here:

In [2]:

```
headers = {'User-Agent':
           'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chr

page = "https://www.transfermarkt.co.uk/transfers/transferrekorde/statistik/top/plus/0
pageTree = requests.get(page, headers=headers)
pageSoup = BeautifulSoup(pageTree.content, 'html.parser')
```

Locate the data within a page & extract it

To fully appreciate what we are doing here, you probably need a basic grasp of HTML – the language tl uc a webpage. As simply as I can put it for this article, HTML is made up of elements, like a paragraph or a link, tha the browser what to render. For scraping, we will use this information to tell our program what information to

Take another look at the page we are scraping. We want two things – the player name and the transfer value.

The player name is a link. This is denoted as an 'a' tag in HTML, so we will use the 'find_all' function to look for a the a tags in the page. However, there are obviously lots of links! Fortunately, we can use the class given to the players' names specifically on this page to only take these ones – the class name is passed to the 'find_all' funct as a dictionary.

This function will return a list with all elements that match our criteria.

If you're curious, classes are usually used to apply styles (such as colour or border) to elements in HTML.

The code to extract the players names is here:

In [3]:

```
Players = pageSoup.find_all("a", {"class": "spielprofil_tooltip"})


#Let's look at the first name in the Players list.
Players[0].text
```

Out[3]:

```
'Luís Figo'
```

Looks like that works! Now let's take the values.

As you can see on the page, the values are not a link, so we need to find a new feature to identify them by.

They are in a table cell, denoted by 'td' in HTML, so let's look for that. The class to highlight these cells specifical 'rechts hauptlink', as you'll see below.

Let's assign this to Values and check Figo's transfer value:

In [4]:

```
Values = pageSoup.find_all("td", {"class": "rechts hauptlink"})


Values[0].text
```

Out[4]:

```
'£54.00m'
```

That's a lot of money! Even in today's market! But according to the page, our data is correct. Now all we need tc is process the data into a dataframe for further analysis or to save for use elsewhere.

Organise the data into a dataframe

This is pretty simple, we know that there are 25 players in the list, so let's use a for loop to add the first 25 playe and value to new lists (to ensure that no stragglers elsewhere in the page jump on). With these new lists, we'll jı create a new dataframe with them:

In [5]:

```
PlayersList = []
ValuesList = []

for i in range(0,25):
    PlayersList.append(Players[i].text)
    ValuesList.append(Values[i].text)

df = pd.DataFrame({"Players":PlayersList,"Values":ValuesList})

df.head()
```

Out[5]:

	PLAYERS	VALUES
0	Luís Figo	£54.00m
1	Hernán Crespo	£51.13m
2	Marc Overmars	£36.00m

	PLAYERS	VALUES
3	Gabriel Batistuta	£32.54m
4	Nicolas Anelka	£31.05m

And now we have a dataframe with our scraped data, pretty much ready for analysis!

Summary

This article has gone through the absolute basics of scraping, we can now load a page, identify elements that w want to scrape and then process them into a dataframe.

There is more that we need to do to scrape efficiently though. Firstly, we can apply a for loop to the whole prog above, changing the initial webpage name slightly to scrape the next year – I'll let you figure out how!

You will also need to understand more about HTML, particularly class and ID selectors, to get the most out of scraping. Regardless, if you've followed along and understand what we've achieved and how, then you're in a g place to apply this to other pages.

The techniques in this article gave the data for our joyplots tutorial, why not take a read of that next?

PREV POST

Joyplots in Python with Joypy

NEXT POST

Scraping Lists Through Transfermarkt and Saving Images

MOST RECENT TUTORIALS

Find us on Twitter!

- → Introducing Pass Elo – Using Elo ratings to measure passers and passes in the 2018 World Cup
- → Introduction to K-Means with Python – Clustering Shot Creators in the Premier League
- → Introduction to Simple Linear Regression in Python