

# Scraping Premier League Football Data with Python

We've already seen in previous scraping articles how we can identify parts on a web page and scrape them into dataframe. We would strongly recommend taking a look through our introductory piece on scraping before pressing forward here.

Another previous article stored player images for match/scouting reports, pre-match presentations, etc. This ti we will be looking to collate the heights, appearances and weights of each Premier League player from the offi site. Let's get our modules imported and run through our process:

In [1]:

```
from lxml import html
import requests
import pandas as pd
import numpy as np
import re
```

Take a look at a player page from the Premier League site. There is loads of information here, but we are intere in collecting the apps, height and weight data.

We could do this manually for each player of each team, but hopefully we can also scrape through a list of eac player in each team, and a list of each team in the league to automate the process entirely. Subsequer r p for the code is going to be something like this:

- Scrape the clubs page and make a list of each team page
- Scrape each team's page for players and make a list of each player
- Scrape each player page and take their height, weight and apps number
  - Save this into a table for later analysis

## Read the Clubs page and list each team

Our intro article goes through this in much more depth, so take a look there if any of the code below is confusi

In short, we need to download the html of the page and identify the links pointing towards the teams. We then this into a list that we can use later.

Take a look through the code and try to follow along. A reminder that more detail is here if you need it!

In [2]:

```
#Take site and structure html
page = requests.get('https://www.premierleague.com/clubs')
tree = html.fromstring(page.content)
```

In [3]:

```
#Using the page's CSS classes, extract all links pointing to a team
linkLocation = tree.cssselect('.indexItem')

#Create an empty list for us to send each team's link to
teamLinks = []

#For each link...
for i in range(0,20):

    #...Find the page the link is going to...
    temp = linkLocation[i].attrib['href']
```

```
    #...Add the link to the website domain...
    temp = "http://www.premierleague.com/" + temp

    #...Change the link text so that it points to the squad list, not the page overvie
    temp = temp.replace("overview", "squad")

    #...Add the finished link to our teamLinks list...
    teamLinks.append(temp)
```

## Read through each team's list of players and create a link for each one

Our process here is very similar to the first step, now we are just looking to create a longer list of each player, r each team.

The main difference is that we will create two links, as the data that we need is across both the player overview page, and the player stats page.

Once again, if anything here is confusing, check out the intro to scraping piece! You might also want to check ou the for loop page as we have a nested for loop in this part of the code!

In [4]:

```
#Create empty lists for player links
playerLink1 = []
playerLink2 = []

#For each team link page...
for i in range(len(teamLinks)):

    #...Download the team page and process the html code...
    squadPage = requests.get(teamLinks[i])
```

```
squadTree = html.fromstring(squadPage.content)

#...Extract the player links...
playerLocation = squadTree.cssselect('.playerOverviewCard')

#...For each player link within the team page...
for i in range(len(playerLocation)):

    #...Save the link, complete with domain...
    playerLink1.append("http://www.premierleague.com/" + playerLocation[i].attrib[

    #...For the second link, change the page from player overview to stats
    playerLink2.append(playerLink1[i].replace("overview", "stats"))
```

## Scrape each player's page for their age, apps, height and weight data

If you have been able to follow along with the previous steps, you'll be absolutely fine here too. The steps are v similar again, just this time we are looking to store data, not links.

We will start this step by defining empty lists for the datapoints we intend to capture. Afterwards, we'll work through each player link to save the player's details. We will also add a little line of code to add in some blank d the site is missing any details – this should allow us to run without any errors. After collecting each player's dat will simply add it to the lists.

Let's collect the data into lists, before we put it into a dataframe and save it as a spreadsheet:

In [5]:

```
#Create lists for each variable
Name = []
Team = []
Age = []
Apps = []
HeightCM = []
WeightKG = []
```

```
#Populate lists with each player

#For each player...
for i in range(len(playerLink1)):

    #...download and process the two pages collected earlier...
    playerPage1 = requests.get(playerLink1[i])
    playerTree1 = html.fromstring(playerPage1.content)
    playerPage2 = requests.get(playerLink2[i])
    playerTree2 = html.fromstring(playerPage2.content)

    #...find the relevant datapoint for each player, starting with name...
    tempName = str(playerTree1.cssselect('div.name')[0].text_content())

    #...and team, but if there isn't a team, return "BLANK"...
    try:
        tempTeam = str(playerTree1.cssselect('.table:nth-child(1) .long')[0].text_cont
    except IndexError:
        tempTeam = str("BLANK")

    #...and age, but if this isn't there, leave a blank 'no number' number...
    try:
        tempAge = int(playerTree1.cssselect('.pdcol2 li:nth-child(1) .info')[0].text_c
    except IndexError:
        tempAge = float('NaN')

    #...and appearances. This is a bit of a mess on the page, so tidy it first...
    try:
        tempApps = playerTree2.cssselect('.statappearances')[0].text_content()
        tempApps = int(re.search(r'\d+', tempApps).group())
    except IndexError:
```

```
        tempApps = float('NaN')

    #...and height. Needs tidying again...
    try:
        tempHeight = playerTree1.cssselect('.pdcol3 li:nth-child(1) .info')[0].text_cc
        tempHeight = int(re.search(r'\d+', tempHeight).group())
    except IndexError:
        tempHeight = float('NaN')

    #...and weight. Same with tidying and returning blanks if it isn't there
    try:
        tempWeight = playerTree1.cssselect('.pdcol3 li+ li .info')[0].text_content()
        tempWeight = int(re.search(r'\d+', tempWeight).group())
    except IndexError:
        tempWeight = float('NaN')


    #Now that we have a player's full details - add them all to the lists
    Name.append(tempName)
    Team.append(tempTeam)
    Age.append(tempAge)
    Apps.append(tempApps)
    HeightCM.append(tempHeight)
    WeightKG.append(tempWeight)
```

## Saving our lists to a dataframe

You'll have noticed that if the data wasn't available, we add a blank item to the list instead. This is really importa as it keeps all of our lists at the same length and means that player data is all in the same row.

We can now add this to a dataframe, made ridiculously easy through the pandas module. Let's create it and ch out our data:

In [6]:

```
#Create data frame from lists
df = pd.DataFrame(
    {'Name':Name,
     'Team':Team,
     'Age':Age,
     'Apps':Apps,
     'HeightCM':HeightCM,
     'WeightKG':WeightKG})

#Show me the top 3 rows:

df.head()
```

Out[6]:

| | AGE | APPS | HEIGHTCM | NAME | TEAM | WEIGHTKG |
|---|---|---|---|---|---|---|
| **0** | 29 | 25 | 183.0 | David Ospina | Arsenal | 80.0 |
| **1** | 35 | 432 | 196.0 | Petr Cech | Arsenal | 90.0 |
| **2** | 23 | 0 | 198.0 | Matt Macey | Arsenal | 81.0 |
| **3** | 23 | 0 | 195.0 | Dejan Iliev | Arsenal | 87.0 |
| **4** | 24 | 22 | 183.0 | Sead Kolasinac | Arsenal | 85.0 |

In [7]:

```
#Show me Karius' height:

df[df['Name']=="Loris Karius"]["HeightCM"]
```

Out[7]:

```
272    189.0
Name: HeightCM, dtype: float64
```

Everything seems to check out, so you're now free to use this data in Python for analysis or visualisation, or you may want to export it for use elsewhere, with the '.to_csv' function:

In [8]:

```
df.to_csv("EPLData.csv")
```

One slight caveat with this dataset is that it includes players on loan – you may want to exclude them. Check ou data analysis course out to learn about cleaning your datasets.

## Summary

In this article, we've covered a lot of fundamental Python tasks through scraping, including for loops, lists and d frames – in addition to increasingly complex ideas like processing html and css classes. If you've followed along great work! But there's no reason not to go back over these topics to make sure you've got a decent understan of them.

Next up, you might want to take a look at visualising some of the age data to check out team profiles!

PREV POST

← How much does it cost to fill the Panini World Cup album? Simulations in Python

NEXT POST

Searching and Downloading Kaggle Datasets in Command Line

## 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

---