Paper 119-2022

Simmering Data: Using Beautiful Soup and Python to Scrape Data from Web Pages

Joe Matise, NORC at the University of Chicago

ABSTRACT

Ever look at a table on a web page and wished you had it in a data set? You probably took one look at the source of the web page and then decided it wasn't worth the hassle. SAS has some tools to help, but oftentimes tables have too much complexity to parse without hours of work. Instead, parse the web page on easy mode using Python – even if you don't know any Python! In this paper, we will show an option using a Python library, Beautiful Soup, that allows users to easily navigate even fairly complex web pages, and quickly pull tables into Pandas DataFrames. Once in a Pandas DataFrame, that data can be easily uploaded to SAS to further processing. This paper can be useful for users at any level of SAS programming, and assumes no knowledge of Python.

INTRODUCTION

One of the most frustrating things for a data scientist – or anyone else who works with data – is to find the perfect resource for some data you need, only to discover that the data is only available as a table on a website. No CSV download option, no Excel file available; just text rendered on the screen in tabular format. Maybe it is just one screen of data, and you can just copy and paste it into a spreadsheet or text file; but when it's ten or twenty pages, not to mention thousands, it is impractical to say the least. Select, copy, paste, click next, select, copy, paste… a mind-numbing waste of time.

In this Hands-On Workshop, you will learn how to download the data yourself using Python, and the library Beautiful Soup. First, you will learn how to install the necessary packages to get started parsing your first HTML pages. Next, you will learn how Beautiful Soup helps you read that HTML as if it were a dataset. Finally, you will parse data from a site with a layout of moderate difficulty needing only a few dozen lines of code.

MIS EN PLACE: GATHERING THE REQUIRED TOOLS

First, we will obtain the necessary software and packages needed to run the programs here and easily parse web page data.

PYTHON

For this version of Beautiful Soup, Python 3 is required. We recommend the Anaconda distribution, as that includes most of the tools needed to work with data. If you use a different distribution of Python 3, you may need to install some packages not covered here in order to make full use of Python to parse your data.

To install Anaconda, visit [URL], and download the correct installer for your operating system, then run the installer. Anaconda can install to your User folders on many computers with restricted permissions; follow the prompts and do not select "all users" if given that option if you have a restricted environment. Follow the instructions to activate Conda from the command line in order to enable additional command line options.

SPYDER

Spyder is a simple option for Python IDEs. Conveniently, it comes with Anaconda, so no additional download is required. While being fairly basic in functionality, it has many similar abilities to other IDEs, including Base SAS – syntax highlighting, the ability to open may programs at once, a console that shows the results of a run (similar to the SAS log).

BEAUTIFUL SOUP

Beautiful Soup is a python package, currently on its fourth iteration – BeautifulSoup4 (bs4). You can install BeautifulSoup from the command line, using `pip install bs4` or `conda install bs4` depending on your preference of package manager.

READING THE RECIPE: A BRIEF INTRODUCTION TO HTML AND BEAUTIFULSOUP

HTML, or HyperText Markup Language, is not a programming language, but rather a way of specifying page layout. Basic HTML consists of the information desired to be displayed, and tags embedded in angled brackets (`< >`) that specifies how to display that information – things like font, styling, alignment, and arrangement. Angled bracket markup usually come in pairs - <tag> to open a section and </tag> to close a section. Tags can also contain information inside them that affects how they are displayed, as well as identifying information.

Here is one example. You could save this to a .html file on your computer and view it in a browser, and it would render a page!

```
<html>
<header>
  <title>Your First Webpage</title>
  <h1>Hello, World!</h1>
</header>
<body>
 <p>
    Hello, world!  This is my first <b>web page</b>.  This is a <a
href="http://www.sas.com/">link to another page</a>.
</p>
</body>
</html>
```

HTML TABLES

One arrangement option for HTML pages is the Table. Tables are commonly used to display data on the screen when it has rows and columns. Tables have four key tags: the `<table>` tag, the `<th>` header tag, the `<tr>` row tag, and the `<td>` data element tag.

`<table>`

The `<table>` tag encloses the entire table. The tag sometimes contains useful information for identifying the table of interest, particularly when many tables are represented on the page.

`<th>`

The `<th>` tag encloses the table header, typically (but not always) the first row of the table. It is similar to the `<tr>` tag but allows for different styling for the header (such as bold or background color). It is often useful for identifying columns of interest.

`<tr>`

The `<tr>` tag encloses table rows other than the table header, with one `<tr>` tag per row of data. `<tr>` tags can contain identifying information about the row in them, but often do not.

`<td>`

The `<td>` tag encloses table data elements, the lowest level of the table in most cases, and often contains only a number or string of text. It can also contain identifying information in it that helps identify which table data element is useful.

Table tags are nested like so:

```
<table id="mytable">
  <th id="mytable_h" >
    <td>Column One</td>
    <td>Column Two</td>
  </th>
  <tr id="mytable_r1">
    <td>1</td>
    <td>2</td>
  </tr>
  <tr id="mytable_r2">
    <td>3</td>
    <td>4</td>
  </tr>
</table>
```

Each `<th>` and `<tr>` should contain the same number of `<td>` elements, or the table will not display as expected.

This will display a table that looks something like this:

Column One	Column Two
1	2
3	4

Tables like this can be read in and transformed into data sets using the Beautiful Soup package.

BEAUTIFUL SOUP

BeautifulSoup (specifically, bs4) is the package we use to transform HTML text into useable data structures. It does not read the URL directly, but uses another package (in our case, urllib) to download the HTML page.

BeautifulSoup uses an object-oriented framework that allows the user to directly reference the HTML elements as objects in Python, using the familiar dot notation.

For example, if you have defined a BeautifulSoup object `soup`, then you can reference a `<table>` in the document with `soup.table` . Further, you can drill down in the same way – `soup.table.tr.td` for example to drill down to the `<td>` elements, though you need to use filters to limit the components to the correct ones.

Attributes

Attributes inside tags, such as `<table id="mytable">`, can be directly accessed through list notation, so `soup.table["id"]` , and then compared to values, such as:

```
     if (soup.table["id"] == "mytable"):
      print(soup.table)
```

Children and Parents

Tags have the built-in generator `children` and the built-in list `parents`, which allow you to access all the tags nested inside of that tag, or all of the tags that tag is nested inside. Generators allow you to loop through a list one at a time without having to produce the whole list at once, which can be much more efficient when you might have many hundreds of children in one tag.

For example:

```
for tr in soup.table.children:
      for td in tr.children:
            print(td.text)
```

`find` and `find_all`

The best options for finding elements that match particular criteria, most commonly a particular `id`, is the `find()` method. That allows you to directly reference a particular cell in the quickest manner:

```
soup.table.find(id='mytable')
```

There is an equivalent when you want to find a group of tags that all have a similar attribute, such as a particular `class`, or find all of a particular type of tag - the `find_all()` method. That takes several arguments – the name of the tag (`name`), the attributes you want to search for (`attrs`), a string that is present in plain text (`string`) are the most useful. That list can be iterated over, like so:

```
for tr in soup.table.find_all('tr'):
```

`select`

The `select` method allows you to use CSS selectors, which can be very powerful tools for finding specific groups of tags. This returns a list which can be iterated over; if in this example the css class `'table_data_element'` was present on all relevant `<td>` tags:

```
for td in soup.table.select("table_data_element")
```

Many other methods exist, and are viewable in the documentation for BeautifulSoup.

GATHERING THE INGREDIENTS: WORKING AN EXAMPLE, PART I

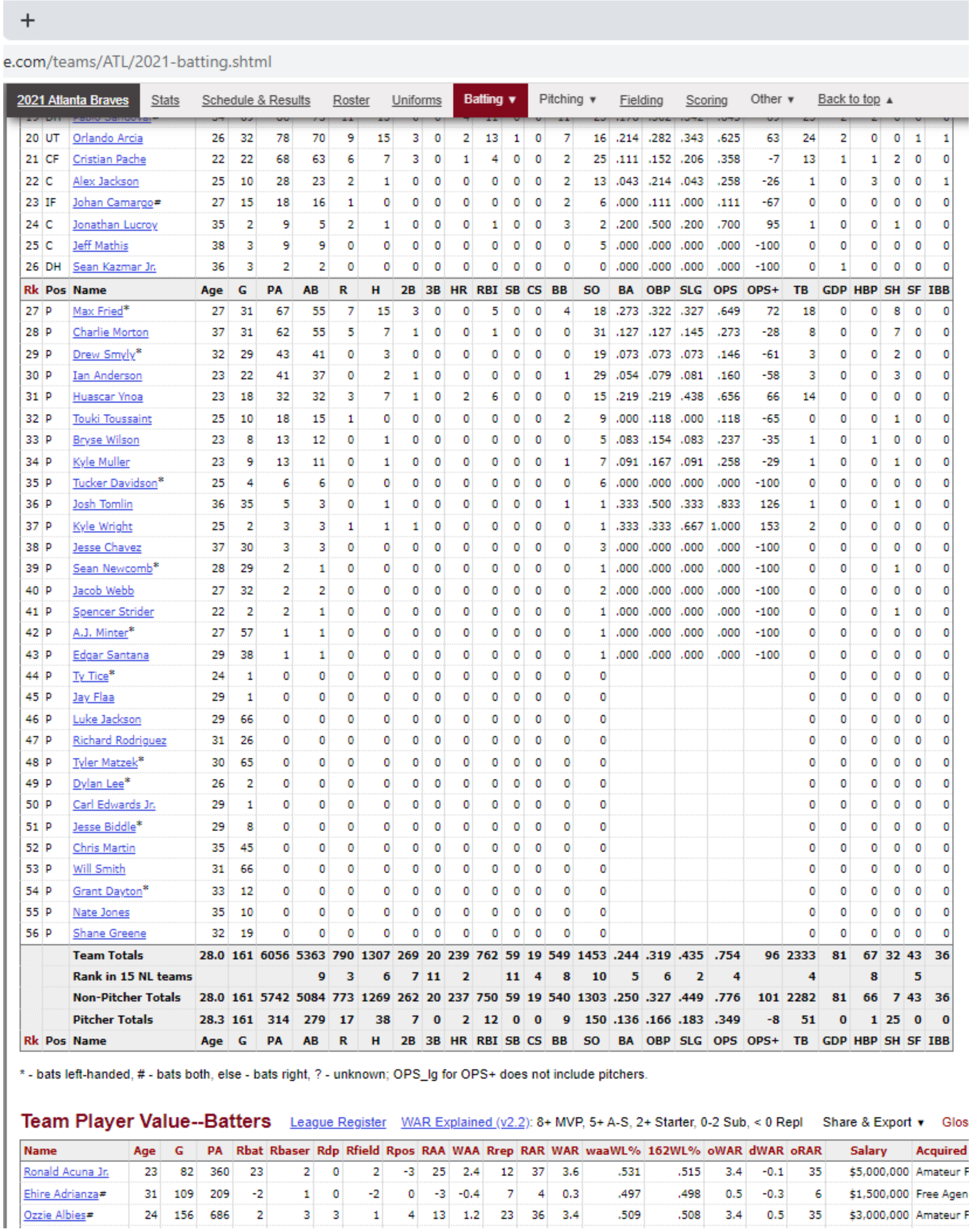
To put this in practice, we will work an example using the website https://baseball-reference.com/ , which contains tables of baseball statistics from over a century of baseball games in Major League Baseball. We'll try to identify the World Series champion that had the worst batting lineup during the regular season.

THE QUESTION

Which team won the World Series with the lowest OPS+ (On Base Percentage Plus Slugging, adjusted for league average and ballpark) in the regular season, since 1908.

IDENTIFY THE PAGE

First, we will look at the page for the most recent World Series champion – the Atlanta Braves from 2021 – and identify the OPS+ stat on that page. That page is found at https://baseball-reference.com/teams/ATL/2021-batting.shtml :



20	UT	Orlando Arcia	26	32	78	70	9	15	3	0	2	13	1	0	7	16	.214	.282	.343	.625	63	24	2	0	0	1	1
21	CF	Cristian Pache	22	22	68	63	6	7	3	0	1	4	0	0	2	25	.111	.152	.206	.358	-7	13	1	1	2	0	0
22	C	Alex Jackson	25	10	28	23	2	1	0	0	0	0	0	0	2	13	.043	.214	.043	.258	-26	1	0	3	0	0	1
23	IF	Johan Camargo#	27	15	18	16	1	0	0	0	0	0	0	0	2	6	.000	.111	.000	.111	-67	0	0	0	0	0	0
24	C	Jonathan Lucroy	35	2	9	5	2	1	0	0	0	1	0	0	3	2	.200	.500	.200	.700	95	1	0	0	1	0	0
25	C	Jeff Mathis	38	3	9	9	0	0	0	0	0	0	0	0	0	5	.000	.000	.000	.000	-100	0	0	0	0	0	0
26	DH	Sean Kazmar Jr.	36	3	2	2	0	0	0	0	0	0	0	0	0	0	.000	.000	.000	.000	-100	0	1	0	0	0	0
Rk	Pos	Name	Age	G	PA	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	BA	OBP	SLG	OPS	OPS+	TB	GDP	HBP	SH	SF	IBB
27	P	Max Fried*	27	31	67	55	7	15	3	0	0	5	0	0	4	18	.273	.322	.327	.649	72	18	0	0	8	0	0
28	P	Charlie Morton	37	31	62	55	5	7	1	0	0	1	0	0	0	31	.127	.127	.145	.273	-28	8	0	0	7	0	0
29	P	Drew Smyly*	32	29	43	41	0	3	0	0	0	0	0	0	0	19	.073	.073	.073	.146	-61	3	0	0	2	0	0
30	P	Ian Anderson	23	22	41	37	0	2	1	0	0	0	0	0	1	29	.054	.079	.081	.160	-58	3	0	0	3	0	0
31	P	Huascar Ynoa	23	18	32	32	3	7	1	0	2	6	0	0	0	15	.219	.219	.438	.656	66	14	0	0	0	0	0
32	P	Touki Toussaint	25	10	18	15	1	0	0	0	0	0	0	0	2	9	.000	.118	.000	.118	-65	0	0	0	1	0	0
33	P	Bryse Wilson	23	8	13	12	0	1	0	0	0	0	0	0	0	5	.083	.154	.083	.237	-35	1	0	1	0	0	0
34	P	Kyle Muller	23	9	13	11	0	1	0	0	0	0	0	0	1	7	.091	.167	.091	.258	-29	1	0	0	1	0	0
35	P	Tucker Davidson*	25	4	6	6	0	0	0	0	0	0	0	0	0	6	.000	.000	.000	.000	-100	0	0	0	0	0	0
36	P	Josh Tomlin	36	35	5	3	0	1	0	0	0	0	0	0	1	1	.333	.500	.333	.833	126	1	0	0	1	0	0
37	P	Kyle Wright	25	2	3	3	1	1	1	0	0	0	0	0	0	1	.333	.333	.667	1.000	153	2	0	0	0	0	0
38	P	Jesse Chavez	37	30	3	3	0	0	0	0	0	0	0	0	0	3	.000	.000	.000	.000	-100	0	0	0	0	0	0
39	P	Sean Newcomb*	28	29	2	1	0	0	0	0	0	0	0	0	0	1	.000	.000	.000	.000	-100	0	0	0	1	0	0
40	P	Jacob Webb	27	32	2	2	0	0	0	0	0	0	0	0	0	2	.000	.000	.000	.000	-100	0	0	0	0	0	0
41	P	Spencer Strider	22	2	2	1	0	0	0	0	0	0	0	0	0	1	.000	.000	.000	.000	-100	0	0	0	1	0	0
42	P	A.J. Minter*	27	57	1	1	0	0	0	0	0	0	0	0	0	1	.000	.000	.000	.000	-100	0	0	0	0	0	0
43	P	Edgar Santana	29	38	1	1	0	0	0	0	0	0	0	0	0	1	.000	.000	.000	.000	-100	0	0	0	0	0	0
44	P	Ty Tice*	24	1	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
45	P	Jay Flaa	29	1	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
46	P	Luke Jackson	29	66	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
47	P	Richard Rodriguez	31	26	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
48	P	Tyler Matzek*	30	65	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
49	P	Dylan Lee*	26	2	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
50	P	Carl Edwards Jr.	29	1	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
51	P	Jesse Biddle*	29	8	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
52	P	Chris Martin	35	45	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
53	P	Will Smith	31	66	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
54	P	Grant Dayton*	33	12	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
55	P	Nate Jones	35	10	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
56	P	Shane Greene	32	19	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
		Team Totals	**28.0**	**161**	**6056**	**5363**	**790**	**1307**	**269**	**20**	**239**	**762**	**59**	**19**	**549**	**1453**	**.244**	**.319**	**.435**	**.754**	**96**	**2333**	**81**	**67**	**32**	**43**	**36**
		Rank in 15 NL teams				**9**	**3**	**6**	**7**	**11**	**2**		**11**	**4**	**8**	**10**	**5**	**6**	**2**	**4**		**4**		**8**		**5**	
		Non-Pitcher Totals	**28.0**	**161**	**5742**	**5084**	**773**	**1269**	**262**	**20**	**237**	**750**	**59**	**19**	**540**	**1303**	**.250**	**.327**	**.449**	**.776**	**101**	**2282**	**81**	**66**	**7**	**43**	**36**
		Pitcher Totals	**28.3**	**161**	**314**	**279**	**17**	**38**	**7**	**0**	**2**	**12**	**0**	**0**	**9**	**150**	**.136**	**.166**	**.183**	**.349**	**-8**	**51**	**0**	**1**	**25**	**0**	**0**
Rk	Pos	Name	Age	G	PA	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	BA	OBP	SLG	OPS	OPS+	TB	GDP	HBP	SH	SF	IBB

* - bats left-handed, # - bats both, else - bats right, ? - unknown; OPS_lg for OPS+ does not include pitchers.

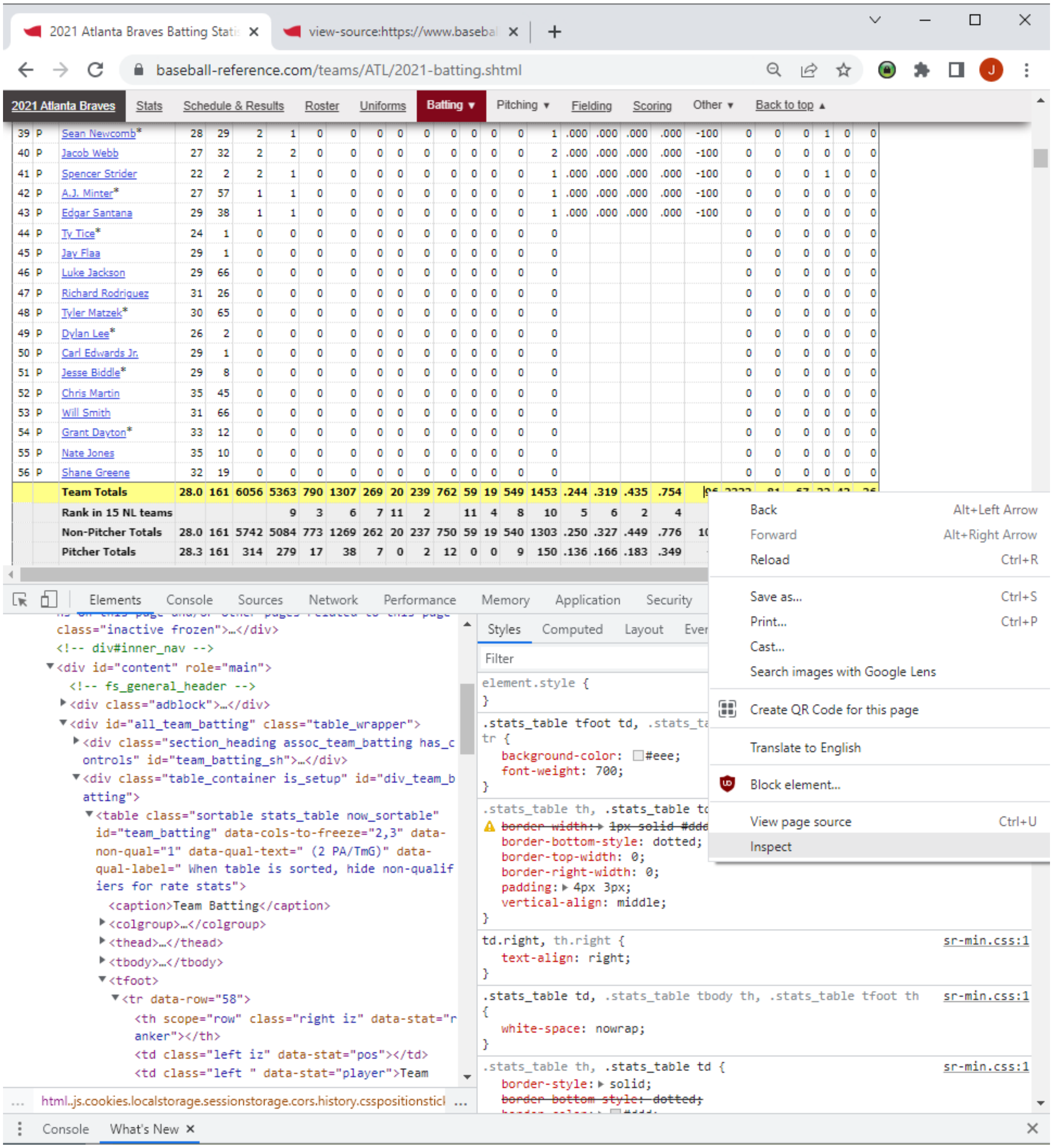
Team Player Value--Batters

League Register WAR Explained (v2.2): 8+ MVP, 5+ A-S, 2+ Starter, 0-2 Sub, < 0 Repl Share & Export ▾ Glos

Name	Age	G	PA	Rbat	Rbaser	Rdp	Rfield	Rpos	RAA	WAA	Rrep	RAR	WAR	waaWL%	162WL%	oWAR	dWAR	oRAR	Salary	Acquired
Ronald Acuna Jr.	23	82	360	23	2	0	2	-3	25	2.4	12	37	3.6	.531	.515	3.4	-0.1	35	$5,000,000	Amateur F
Ehire Adrianza#	31	109	209	-2	1	0	-2	0	-3	-0.4	7	4	0.3	.497	.498	0.5	-0.3	6	$1,500,000	Free Agen
Ozzie Albies#	24	156	686	2	3	3	1	4	13	1.2	23	36	3.4	.509	.508	3.4	0.5	35	$3,000,000	Amateur F

INSPECT THE HTML

By using the appropriate menu option or hotkey to inspect the source of the web page, we can find the specific items that will be needed here. See below:



39	P	Sean Newcomb*	28	29	2	1	0	0	0	0	0	0	0	0	0	1	.000	.000	.000	.000	-100	0	0	0	1	0	0
40	P	Jacob Webb	27	32	2	2	0	0	0	0	0	0	0	0	0	2	.000	.000	.000	.000	-100	0	0	0	0	0	0
41	P	Spencer Strider	22	2	2	1	0	0	0	0	0	0	0	0	0	1	.000	.000	.000	.000	-100	0	0	0	1	0	0
42	P	A.J. Minter*	27	57	1	1	0	0	0	0	0	0	0	0	0	1	.000	.000	.000	.000	-100	0	0	0	0	0	0
43	P	Edgar Santana	29	38	1	1	0	0	0	0	0	0	0	0	0	1	.000	.000	.000	.000	-100	0	0	0	0	0	0
44	P	Ty Tice*	24	1	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
45	P	Jay Flaa	29	1	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
46	P	Luke Jackson	29	66	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
47	P	Richard Rodriguez	31	26	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
48	P	Tyler Matzek*	30	65	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
49	P	Dylan Lee*	26	2	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
50	P	Carl Edwards Jr.	29	1	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
51	P	Jesse Biddle*	29	8	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
52	P	Chris Martin	35	45	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
53	P	Will Smith	31	66	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
54	P	Grant Dayton*	33	12	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
55	P	Nate Jones	35	10	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
56	P	Shane Greene	32	19	0	0	0	0	0	0	0	0	0	0	0	0						0	0	0	0	0	0
		Team Totals	28.0	161	6056	5363	790	1307	269	20	239	762	59	19	549	1453	.244	.319	.435	.754	96	2222	81	67	22	42	26
		Rank in 15 NL teams				9	3	6	7	11	2		11	4	8	10	5	6	2	4							
		Non-Pitcher Totals	28.0	161	5742	5084	773	1269	262	20	237	750	59	19	540	1303	.250	.327	.449	.776	10						
		Pitcher Totals	28.3	161	314	279	17	38	7	0	2	12	0	0	9	150	.136	.166	.183	.349	-						

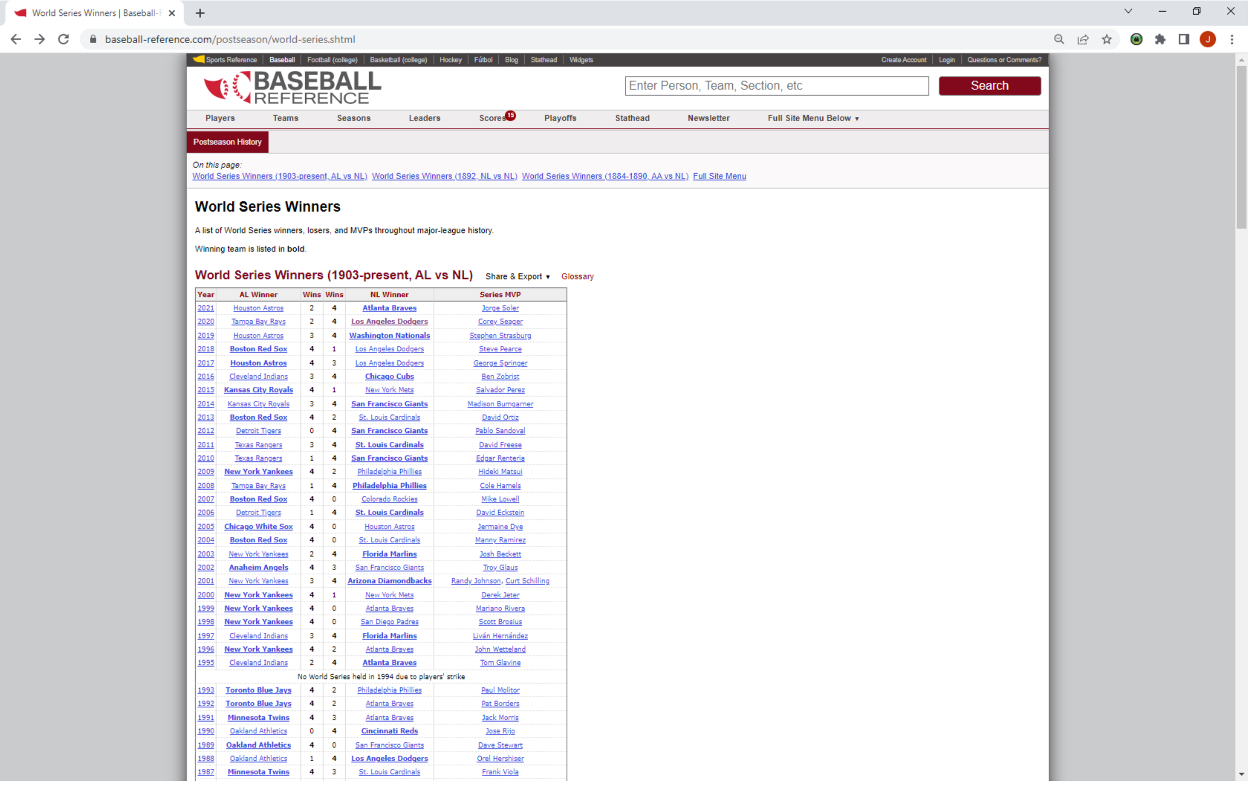
```
ns on this page and/or other pages related to this page
class="inactive frozen">…</div>
<!-- div#inner_nav -->
▼<div id="content" role="main">
  <!-- fs_general_header -->
 ▶<div class="adblock">…</div>
 ▼<div id="all_team_batting" class="table_wrapper">
   ▶<div class="section_heading assoc_team_batting has_c
    ontrols" id="team_batting_sh">…</div>
   ▼<div class="table_container is_setup" id="div_team_b
    atting">
     ▼<table class="sortable stats_table now_sortable"
      id="team_batting" data-cols-to-freeze="2,3" data-
      non-qual="1" data-qual-text=" (2 PA/TmG)" data-
      qual-label=" When table is sorted, hide non-qualif
      iers for rate stats">
        <caption>Team Batting</caption>
       ▶<colgroup>…</colgroup>
       ▶<thead>…</thead>
       ▶<tbody>…</tbody>
       ▼<tfoot>
         ▼<tr data-row="58">
            <th scope="row" class="right iz" data-stat="r
            anker"></th>
            <td class="left iz" data-stat="pos"></td>
            <td class="left " data-stat="player">Team
```

```
... html..js.cookies.localstorage.sessionstorage.cors.history.csspositionsticl ...
```

```
element.style {
}
.stats_table tfoot td, .stats_ta
tr {
  background-color: □#eee;
  font-weight: 700;
}
.stats_table th, .stats_table tc
⚠ ~~border-width: ▸ 1px solid #ddd~~
  border-bottom-style: dotted;
  border-top-width: 0;
  border-right-width: 0;
  padding: ▸ 4px 3px;
  vertical-align: middle;
}
td.right, th.right {                                        sr-min.css:1
  text-align: right;
}
.stats_table td, .stats_table tbody th, .stats_table tfoot th   sr-min.css:1
{
  white-space: nowrap;
}
.stats_table th, .stats_table td {                          sr-min.css:1
  border-style: ▸ solid;
  ~~border-bottom-style: dotted;~~
```

Here we see that the <table> is id="team_batting", the <tr> of interest is the first <tr> element inside of a <tfoot> element, and the specific <td> has an attribute data-stat="onbase_plus_slugging_plus". These tell us all of the information we need to find the statistic on a single page. Now, we need to find how to get the page for each World Series Champion.

The World Series Champions are visible on a single page, with links to the team statistics pages:


World Series Winners | Baseball-
baseball-reference.com/postseason/world-series.shtml
Sports Reference | Baseball | Football (college) | Basketball (college) | Hockey | Fútbol | Blog | Stathead | Widgets
Create Account | Login | Questions or Comments?
BASEBALL REFERENCE
Enter Person, Team, Section, etc
Search
Players
Teams
Seasons
Leaders
Scores 15
Playoffs
Stathead
Newsletter
Full Site Menu Below ▾
Postseason History
On this page:
World Series Winners (1903-present, AL vs NL) World Series Winners (1892, NL vs NL) World Series Winners (1884-1890, AA vs NL) Full Site Menu
Share & Export ▾
Glossary

World Series Winners

A list of World Series winners, losers, and MVPs throughout major-league history.

Winning team is listed in **bold**.

World Series Winners (1903-present, AL vs NL)

Year	AL Winner	Wins	Wins	NL Winner	Series MVP
2021	Houston Astros	2	4	**Atlanta Braves**	Jorge Soler
2020	Tampa Bay Rays	2	4	**Los Angeles Dodgers**	Corey Seager
2019	Houston Astros	3	4	**Washington Nationals**	Stephen Strasburg
2018	**Boston Red Sox**	4	1	Los Angeles Dodgers	Steve Pearce
2017	**Houston Astros**	4	3	Los Angeles Dodgers	George Springer
2016	Cleveland Indians	3	4	**Chicago Cubs**	Ben Zobrist
2015	**Kansas City Royals**	4	1	New York Mets	Salvador Perez
2014	Kansas City Royals	3	4	**San Francisco Giants**	Madison Bumgarner
2013	**Boston Red Sox**	4	2	St. Louis Cardinals	David Ortiz
2012	Detroit Tigers	0	4	**San Francisco Giants**	Pablo Sandoval
2011	Texas Rangers	3	4	**St. Louis Cardinals**	David Freese
2010	Texas Rangers	1	4	**San Francisco Giants**	Edgar Renteria
2009	**New York Yankees**	4	2	Philadelphia Phillies	Hideki Matsui
2008	Tampa Bay Rays	1	4	**Philadelphia Phillies**	Cole Hamels
2007	**Boston Red Sox**	4	0	Colorado Rockies	Mike Lowell
2006	Detroit Tigers	1	4	**St. Louis Cardinals**	David Eckstein
2005	**Chicago White Sox**	4	0	Houston Astros	Jermaine Dye
2004	**Boston Red Sox**	4	0	St. Louis Cardinals	Manny Ramirez
2003	New York Yankees	2	4	**Florida Marlins**	Josh Beckett
2002	**Anaheim Angels**	4	3	San Francisco Giants	Troy Glaus
2001	New York Yankees	3	4	**Arizona Diamondbacks**	Randy Johnson, Curt Schilling
2000	**New York Yankees**	4	1	New York Mets	Derek Jeter
1999	**New York Yankees**	4	0	Atlanta Braves	Mariano Rivera
1998	**New York Yankees**	4	0	San Diego Padres	Scott Brosius
1997	Cleveland Indians	3	4	**Florida Marlins**	Liván Hernández
1996	**New York Yankees**	4	2	Atlanta Braves	John Wetteland
1995	Cleveland Indians	2	4	**Atlanta Braves**	Tom Glavine
No World Series held in 1994 due to players' strike					
1993	**Toronto Blue Jays**	4	2	Philadelphia Phillies	Paul Molitor
1992	**Toronto Blue Jays**	4	2	Atlanta Braves	Pat Borders
1991	**Minnesota Twins**	4	3	Atlanta Braves	Jack Morris
1990	Oakland Athletics	0	4	**Cincinnati Reds**	Jose Rijo
1989	**Oakland Athletics**	4	0	San Francisco Giants	Dave Stewart
1988	Oakland Athletics	1	4	**Los Angeles Dodgers**	Orel Hershiser
1987	**Minnesota Twins**	4	3	St. Louis Cardinals	Frank Viola

From the source of that page, it can be identified that the World Series champion is bolded with the tag <strong>, and appear on rows with a <th> element (which is used somewhat unusually here). We will need to extract both the year and the page link to the champion from this.

Our strategy will be to iterate over the winners from the World Series Champion page, and then for each winner load its team page, extract the OPS+, and load that into a list. Once we have every OPS+ loaded into the list (along with the year and team), we can a statistical routine to identify the minimum.

STIRRING THE SOUP: WORKING AN EXAMPLE, PART II

Now that we have identified the HTML pages, tags, and other information needed to solve the problem, we will write a Python script to accomplish this. In order to do that, we need to break the problem down into a few pieces.

First, we will write a function to read an individual team page. Then, we will write a function to parse the team list page that has each World Series winner on it, and call the first function to read the data for that team. Finally, we will call that function, store the results, and determine the record with the lowest OPS+.

PARSING A TEAM PAGE

To parse a single team's page, we identify on the team page the specific item we chose – the team_`batting` table, the `tfoot` element in that table, and the `td` element in that table footer that has the `data-stat` attribute `onbase_plus_slugging_plus`. Then, we store the text found in that element.

```
def ops_tot(team_str):
    response = urlopen('https://www.baseball-reference.com' + team_str)
                   # Construct the URL from the parameter
    html_doc = response.read()
    soup = BeautifulSoup(html_doc, 'html.parser')
                   # Read the web page for that URL and parse it

    table = soup.find('table',id='team_batting')
                  # Look for the table with the team_batting id
    if table != None:
      for tfoot in table.tfoot:
                  # Look for the table footer
        td = tfoot.find_all('td')
                  # Find all of the td elements and save in a list
        for t in td:
                  # Iterate over that list and look for the data-stat we want
          if t['data-stat'] == 'onbase_plus_slugging_plus' and t.text != '':
            return(t.text)
                  # Return the text from the relevant table cell
```

PARSING THE LIST OF TEAMS

To parse the page that lists the teams, we find the table world_series_winners_al_nl, and then we examine each tr element. For each table row, we look at whether it has a th element. If it does, we extract the year from the text in the th element, and then find any cells that are bolded via the strong tag. Then, we look inside the strong tag , and see if it has a href attribute inside (which is used for links). If it does, then we can extract that url as well as identify the team name (the contents of the cell text). Once we have that, we call the team page function and add the returned value to a list.

```
def ws_winners():
    response = urlopen('https://www.baseball-reference.com/postseason/world-
series.shtml')
    html_doc = response.read()
    soup = BeautifulSoup(html_doc, 'html.parser')
    stats = []                 #initialize the list to store our stats in
    table = soup.find('table',id='world_series_winners_al_nl')
    if table != None:
        for row in table.find_all('tr'):
            ws_stats = {}                        #initialize our dictionary
            if row.contents[0].name == 'th':
                     #Look for th at the start (eliminates non-data rows)
                year = row.contents[0].text      #grab the year
                strong_cells = row.find_all('strong')
                     #Look for cells with <strong> tag. WS winner is bolded!
                for cell in strong_cells:
                    try:       #need try-except because of bolded Wins
                               #try lets us test things and ignore errors
                        link = cell.find('a')['href']
                     #find the value of the href attribute in the anchor tag>
                        ws_stats['team'] = cell.text
                     #the team name
                        ws_stats['ops+'] = int(ops_tot(link))
                     #call the earlier function to read the OPS+
                        ws_stats['year'] = year   #save the year
                        stats.append(ws_stats)    #append to the main list
                        print(ws_stats)
                    except(TypeError):
                        pass                      #ignore errors

    return(stats)   #returns the list of results
```

RETURNING THE RESULT

To compute the final result, we call the function to parse the list of teams (and thus parse each team), and then determine which row has the minimum. To do that, we use a clever Python syntax element called a *lambda*, which lets us write a mini-function that in this case is what we tell Python to take the minimum of (the OPS+ value) when we pass it the entire list element (the team name, year, and OPS+).

```
final_stats = ws_winners()

minOPS = min(final_stats,key=lambda x:x['ops+'])
            #key=lambda x:x['ops+'] identifies what to take the min of
print(minOPS)
```

TASTING THE RESULTS

It is not a miracle that the result is…

```
{'team': 'New York Mets', 'ops+': 84, 'year': '1969'}
```

CONCLUSION

Parsing well-formatted web pages using BeautifulSoup in python can be as simple as baking a cake with a box mix, as long as you do your prep work first. Find the element you want to read, trace its parentage, then write a line of code for each element in that tree.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Joe Matise
NORC at the University of Chicago
matisejoe@gmail.com