There's Always an API

Sometimes they make you work for it

Hi, I'm Matt Dennewitz

- VP Product, Pitchfork; Dir. Engineering, Wired
- I consult on data for baseball writers and MLB clubs
- · @mattdennewitz on Twitter, Github

Agenda

- · 101
- Your first scrape: Google Docs
- Interlude: HTML, JSON, XML, XPath
- Scaling up: Python
- What happens when the data isn't on the page?
- Advanced topics (time allowing)

What is scraping?

- Extracting information from a document
 - Rows from an HTML table
 - Text from a PDF
 - Images from Craigslist posts or museum websites
 - OCR'ing an image and reading its text
 - Spidering a website like Google

Tools

- Google Docs (surprise!)
- Chrome Developer Tools
- Python
- Scrapy

Strategy

- 1. "What do I want?"
- 2. Case the joint
- 3. Rob it just a little bit
- 4. Move in

"What do I want?"

- Envision the data you want, how you need it
 - "How will I scrape this data?" Script? Crawler?
 - "Do I need to scrape this more than once?"
 - "How do I need to shape the data?"
 - "What do I need to do to the data after I have it?" Clean, verify, cross-link with another data set, ...?
 - "How/to where do I want to output the data?"

Case the joint

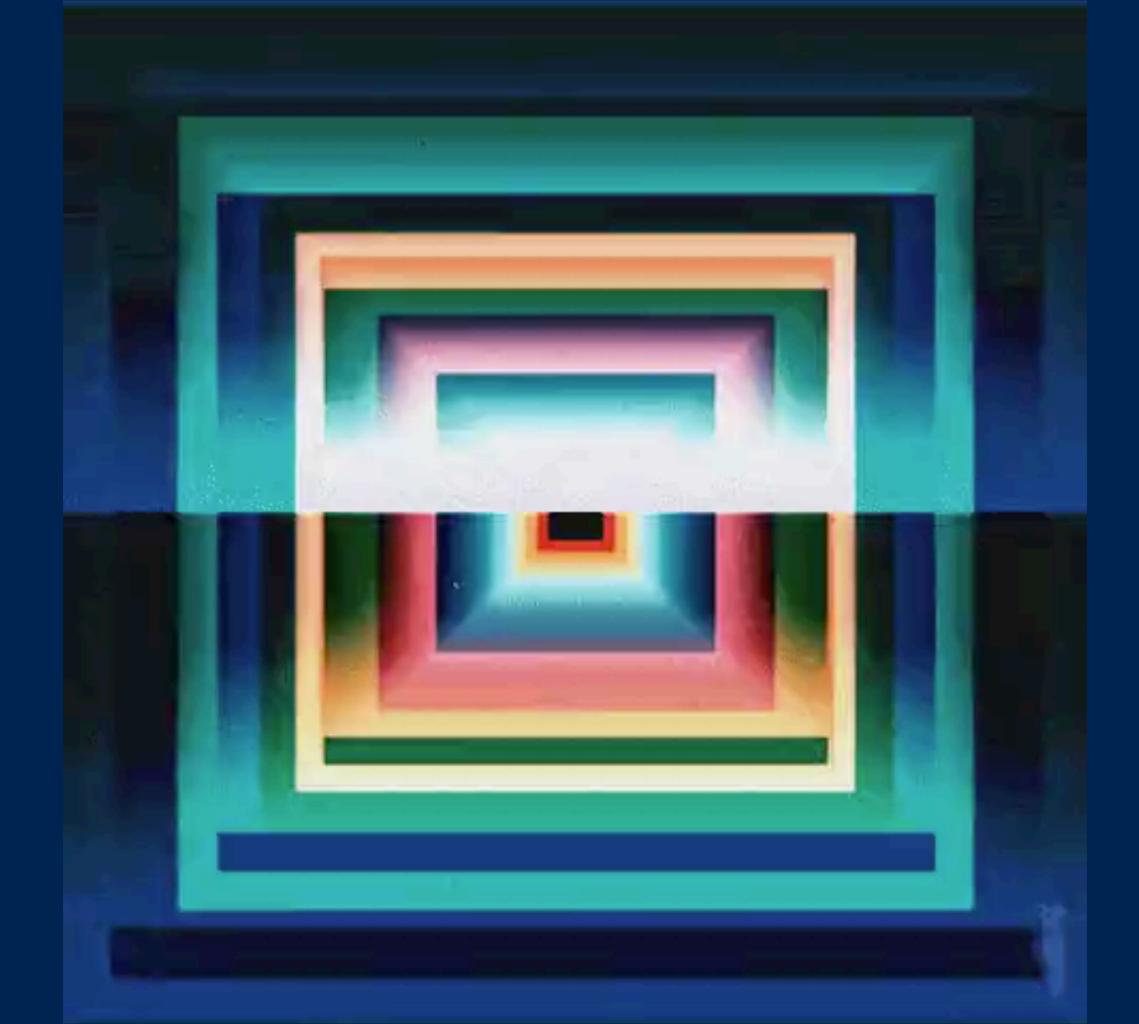
- Does the document seem scrape-ready? Does access come with preconditions?
 - Preconditions: password-protected? Online-only?
 Needs a special decoder?
- Look at how the data is presented in the document. Are there external dependencies, or is it self-contained?
 - External deps: more information on secondary pages, data in other spreadsheets or workbooks

Rob it just a little bit

- Prototype using a subset of the information
- Estimate how long scraping will take, determine imperative needs like throttling or a specific OS
- Validate your ideas about the data you wish to extract, correct bugs
 - Writing unit tests

Oceans 1101

- You've created a stable scraper which emits data in the format you want (CSV, JSON, XML, SQL, ...) to the location you want
- You understand its performance characteristics
- · Go!



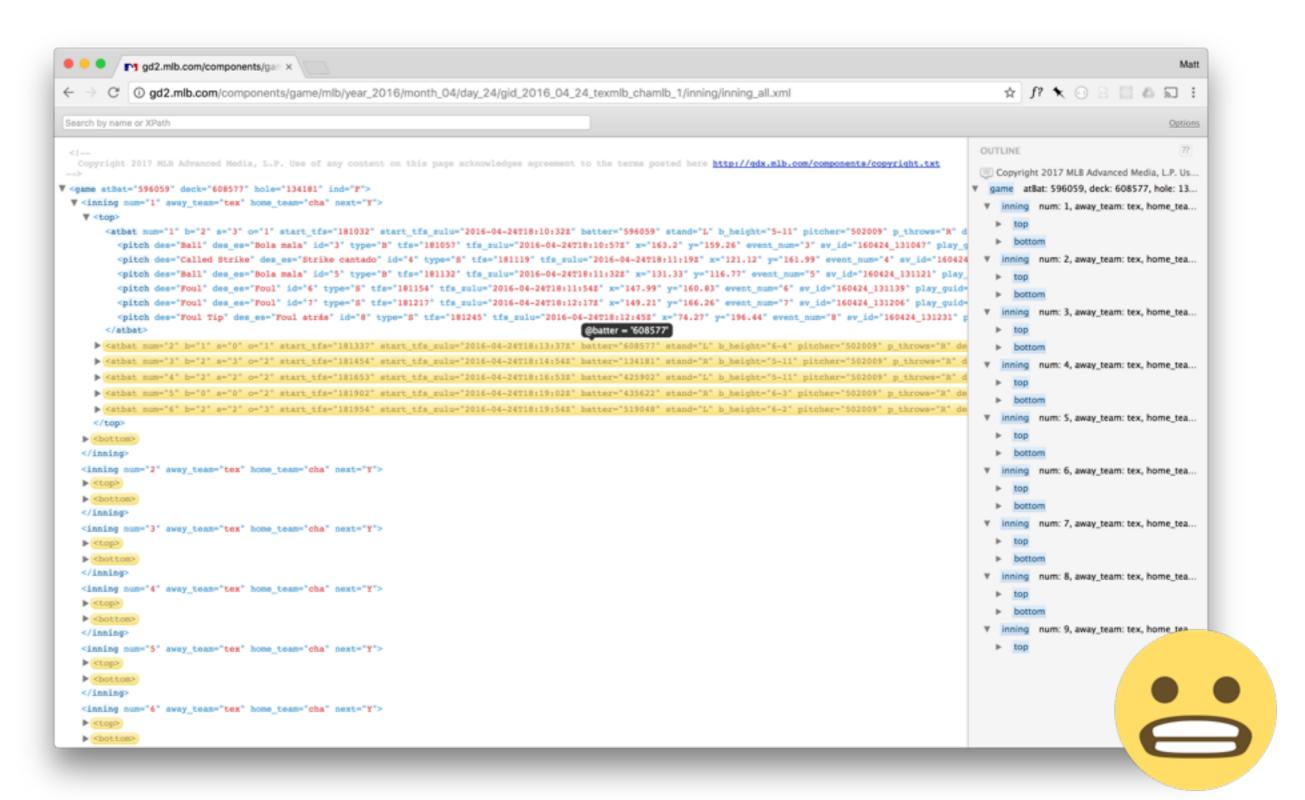
Interlude: formats

- Data is distributed in mercilessly innumerable formats
- The Big Three of web scraping
 - HTML
 - JSON
 - XML

Formats: XML

- eXtensible Markup Language
- Well-structured, self-validating, predictable
- Pedantic, though not with its charms

Formats: XML



Formats: HTML

- Hypertext something something
- XML-like, without the upside
- Needs stronger class of parser to heal broken code
- Less predictable, far more susceptible to changes in the wind

Formats: HTML

```
>
  1.
  <strong><span class="playerdef"><a href="http://</pre>
www.baseballprospectus.com/card/card.php?id=102123">Alex Reyes</
a></span>, RHP, <span class="teamdef"><a href="http://
www.baseballprospectus.com/team_audit.php?team=SLN"
target="blank">St. Louis Cardinals</a></span></strong><br>
 Scouting Report: <a href="http://www.baseballprospectus.com/"</pre>
article.php?articleid=30958">LINK</a>
```

Formats: JSON

- JavaScript Object Notation
- Data objects with simple primitives: int, double, string, boolean, object (key/value pairs), array (untyped), null.
- Requires waaaaaaay less parsing, much easier to serialize
- No schemas, but validation tools exist
- Has taken over for XML in web data transmission

Formats: JSON

```
{
  "prospect_year": "2017",
  "player_id": 643217,
  "player_first_name": "Andrew",
  "player_last_name": "Benintendi",
  "rank": 1,
  "position": "OF",
  "preseason100": 1,
  "preseason20": 1,
  "team_file_code": "BOS",
}
```



Bonus: XPath

- XPath is a way to query XML (and HTML)
- It's got a super goofy syntax
- · Very powerful, essential for scraping the web

Bonus: XPath

```
• XPath: //table/tbody/tr
• HTML (fragment):
<thead>
  NameHRSB
  </thead>
 Mike Trout4040
```

Result:
 Mike Trout
 40
 40
 40
 10

Bonus: XPath

* XPath: //span[@class="playerdef"]/text()

· HTML:

```
1. <strong><span class="playerdef">Eloy
Jiminez</span></strong>, OF, ...
```

• Result: "Eloy Jiminez"

Ok, time to scrape

Google Docs

- Fire up Google Docs, start a new spreadsheet
- IMPORTXML and IMPORTHTML are your friends
- Let's look at IMPORTHTML

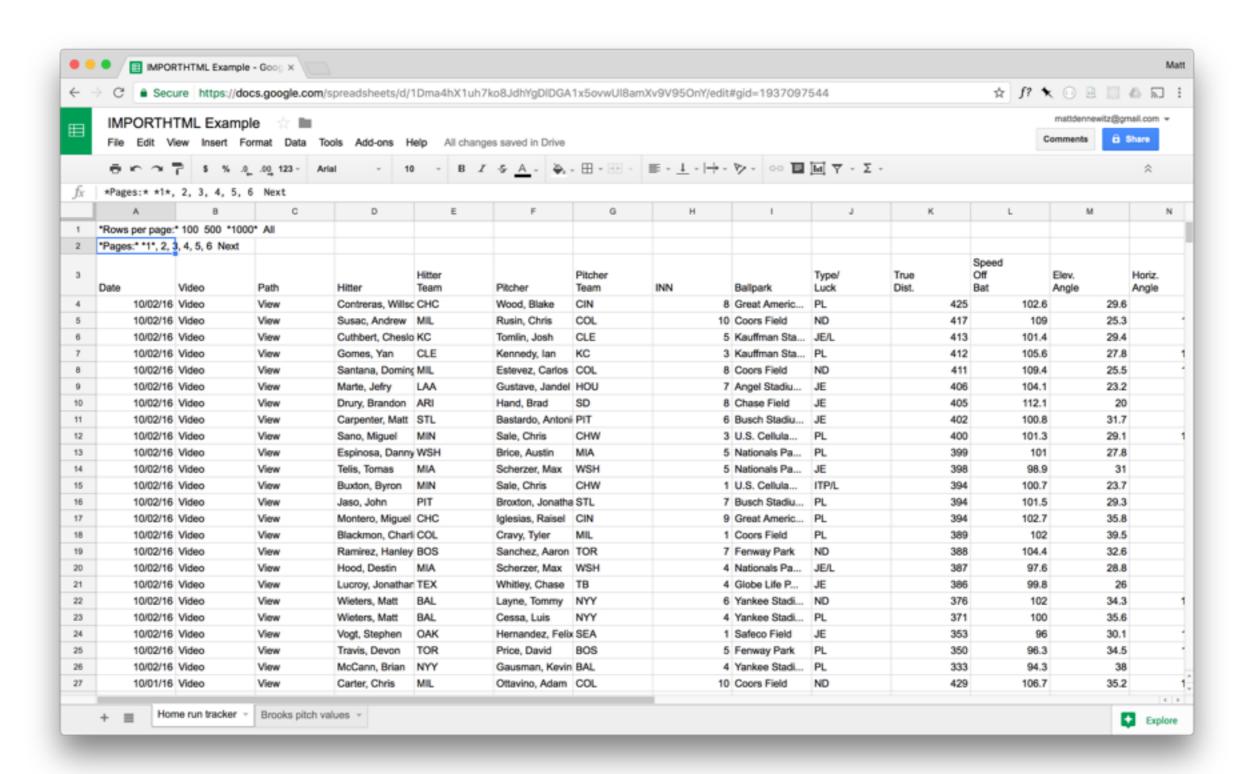
IMPORTHIML

- Allows you put pull in specific list or tabular data from a web page
- Syntax:
 =IMPORTHTML(url, <"list" or "table">,
 [index])

IMPORTHIML

- ESPN Home Run Tracker
- Syntax:
 =importhtml("http://
 www.hittrackeronline.com/?perpage=1000",
 "table", 17)
- "Give me the 16th table on the page" (0-based indexing)

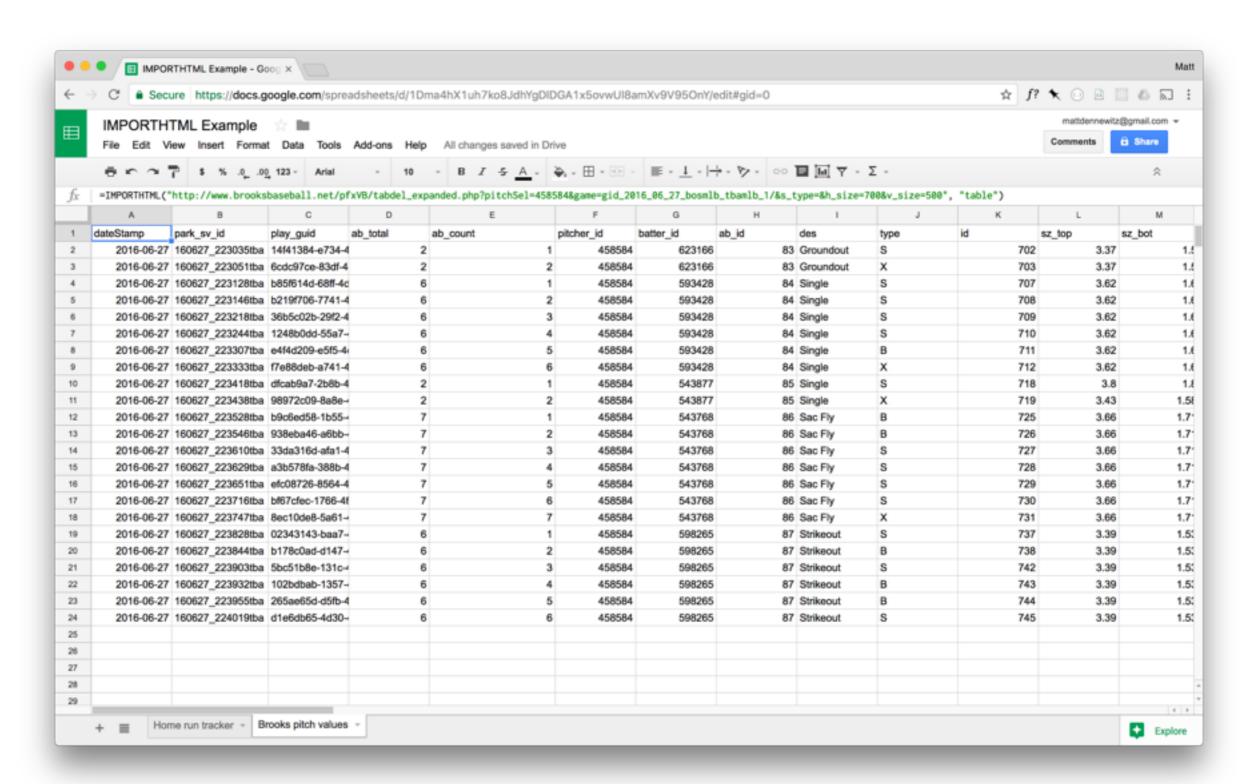
IMPORTHTML



IMPORTHTML

- Brooks Baseball Player Pitch Logs
- Syntax:
 =IMPORTHTML("http://
 www.brooksbaseball.net/pfxVB/
 tabdel_expanded.php?
 - pitchSel=458584&game=gid_2016_06_27_bosm
 lb tbamlb 1/
 - &s_type=&h_size=700&v_size=500",
 "table")

IMPORTHTML



Google Docs

- Useful for pulling in single tables, or keeping everything in a spreadsheet
- Data doesn't always exist in a single place
 - Spread across several pages
 - Spread across several files or APIs

Google Docs

- Useful for pulling in single tables, or keeping everything in a spreadsheet
- Data doesn't always exist in a single place
 - Spread across several pages
 - Spread across several files or APIs
- Automate as much as you can

Python time

- Beautiful language. Transcendental even.
- Robust ecosystem for handling data parsing, cleaning, making net requests, etc
- A+ community
- Runs anywhere



Python time

- I'm going to use two non-standard packages today:
 - 1xml, for HTML parsing and cleaning
 - requests, for HTTP fetching



- 1. "What do I want?"
- 2. Case the joint
- 3. Rob it just a little bit
- 4. Move in

- "What do I want?": prospect rankings from BP, MLB, Baseball America
- 2. Case the joint
- 3. Rob it just a little bit
- 4. Move in

- "What do I want?": prospect rankings from BP, MLB
- 2. **Case the joint**: BP has dirty HTML. MLB loads a JSON file.
- 3. Rob it just a little bit
- 4. Move in

- "What do I want?": prospect rankings from BP, MLB
- 2. **Case the joint**: BP has dirty HTML. MLB loads a JSON file.
- 3. **Rob it just a little bit**: Get a feel for BP and BA's HTML structure, examine MLB's JSON file.
- 4. Move in

Strategy (again)

- "What do I want?": prospect rankings from BP, MLB.
- 2. **Case the joint**: BP has dirty HTML. MLB loads a JSON file.
- 3. **Rob it just a little bit**: Get a feel for BP and BA's HTML structure, examine MLB's JSON file.
- 4. Move in: Write three scripts, one for each.

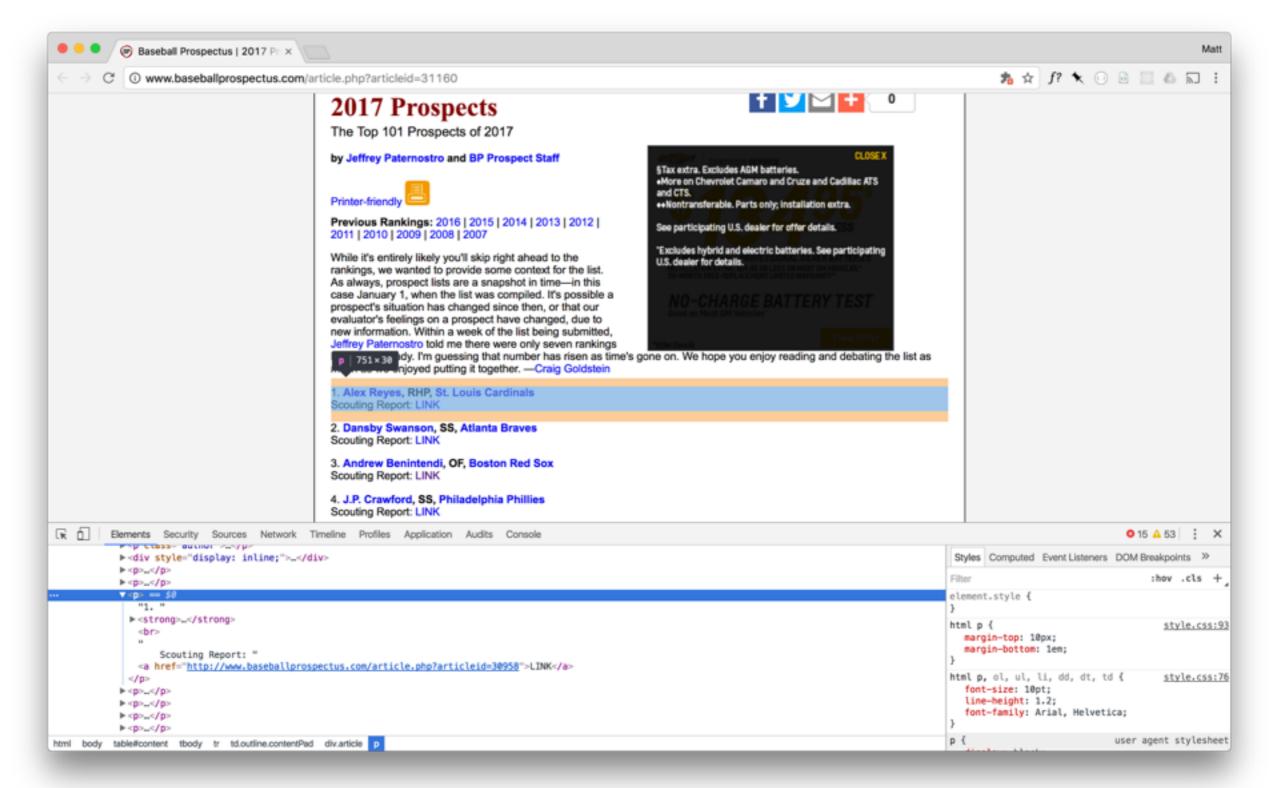
Strategy (again)

- Fields to export:
 - Name
 - Rank
 - List type ("BP", "MLB", ...)
 - System ID (MLBAM ID, BP player ID, ...)

 http://www.baseballprospectus.com/article.php? articleid=31160

- First thing to do is inspect the source
 - Is there a pattern in the HTML you can engineer for, or an attribute you can target?
- Let's head to the console! Right click on the one of the capsules, and click "Inspect"





- Yes! Player data is in a paragraph tag, , which contains a with class "playerdef"
 - Get used to talking like this
- Using XPath, we can target that and walk up to its parent element, , which gives us access to the whole player capsule

- Beware: the "playerdef" class could be used anywhere.
 We need to find a reasonable scope for our XPath.
- Luckily for us, player capsules are in a <div> with class "article", and that structure appears only once per article page across BP.
- XPath: //div[@class="article"]// span[@class="playerdef"]/..
- What else?

- Code: https://github.com/mattdennewitz/sloan-scraping/blob/master/bp-
 top-101-2017.py
- Output: https://github.com/mattdennewitz/
 sloan-scraping/blob/master/bp-2017.csv

• • •	1, less				
	+	+	+		
rank	id	name	url		
1	102123	Alex Reyes	http://www.baseballprospectus.com/card/card.php?id=102123		
2	107168	Dansby Swanson	http://www.baseballprospectus.com/card/card.php?id=107168		
3	105574	Andrew Benintendi	http://www.baseballprospectus.com/card/card.php?id=105574		
4	102559	J.P. Crawford	http://www.baseballprospectus.com/card/card.php?id=102559		
5	105432	Yoan Moncada	http://www.baseballprospectus.com/card/card.php?id=105432		
6	102432	Austin Meadows	http://www.baseballprospectus.com/card/card.php?id=102432		
7	104023	Victor Robles	http://www.baseballprospectus.com/card/card.php?id=104023		
8	103203	Amed Rosario	http://www.baseballprospectus.com/card/card.php?id=103203		
9	104176	Eloy Jimenez	http://www.baseballprospectus.com/card/card.php?id=104176		
10	100261	Lucas Giolito	http://www.baseballprospectus.com/card/card.php?id=100261		
11	106765	Brendan Rodgers	http://www.baseballprospectus.com/card/card.php?id=106765		
12	100634	Lewis Brinson	http://www.baseballprospectus.com/card/card.php?id=100634		
13	104042	Rafael Devers	http://www.baseballprospectus.com/card/card.php?id=104042		
14	70795	Tyler Glasnow	http://www.baseballprospectus.com/card/card.php?id=70795		
15	104180	Gleyber Torres	http://www.baseballprospectus.com/card/card.php?id=104180		
16	102429		http://www.baseballprospectus.com/card/card.php?id=102429		
17	70551	Robert Gsellman	http://www.baseballprospectus.com/card/card.php?id=70551		
18	100988	Manuel Margot	http://www.baseballprospectus.com/card/card.php?id=100988		
19	101384	Josh Hader	http://www.baseballprospectus.com/card/card.php?id=101384		
20	108571		http://www.baseballprospectus.com/card/card.php?id=108571		
21	103209	Willy Adames	http://www.baseballprospectus.com/card/card.php?id=103209		
22	103893		http://www.baseballprospectus.com/card/card.php?id=103893		
23	107180		http://www.baseballprospectus.com/card/card.php?id=107180		
24	105921		http://www.baseballprospectus.com/card/card.php?id=105921		
25	! !	Josh Bell	http://www.baseballprospectus.com/player_search.php?search_name=Josh+Bell		
26	102519		http://www.baseballprospectus.com/card/card.php?id=102519		
27	104816		http://www.baseballprospectus.com/card/card.php?id=104816		
28	103071		http://www.baseballprospectus.com/card/card.php?id=103071		
29	109123	Jason Groome	http://www.baseballprospectus.com/card/card.php?id=109123		
30	101728		http://www.baseballprospectus.com/card/card.php?id=101728		
31	105454		http://www.baseballprospectus.com/card/card.php?id=105454		
32	70946	Amir Garrett	http://www.baseballprospectus.com/card/card.php?id=70946		
	68865	Jorge Alfaro	http://www.baseballprospectus.com/card/card.php?id=68062		
34	1 104004	Francisco Mejia	http://www.baseballprospectus.com/player_search.php?search_name=Francisco+Mejia		
35	104004		http://www.baseballprospectus.com/card/card.php?id=104004		
36		Michael Kopech	http://www.baseballprospectus.com/card/card.php?id=104824		
37	10/04/	Kyle Tucker Jose De Leon	http://www.baseballprospectus.com/card/card.php?id=107047		
38 39	 101100	Yohander Mendez	http://www.baseballprospectus.com/player_search.php?search_name=Jose+De+Leon http://www.baseballprospectus.com/card/card.php?id=101198		
•	101130	Tollalluel Melluez	neep.//www.baseballprospectos.com/card/card/card-lollso		

- What did we do?
 - Inspected the page
 - Found critical path to data, wrote supporting XPaths
 - Scripted collecting and outputting the data

http://m.mlb.com/prospects/2017

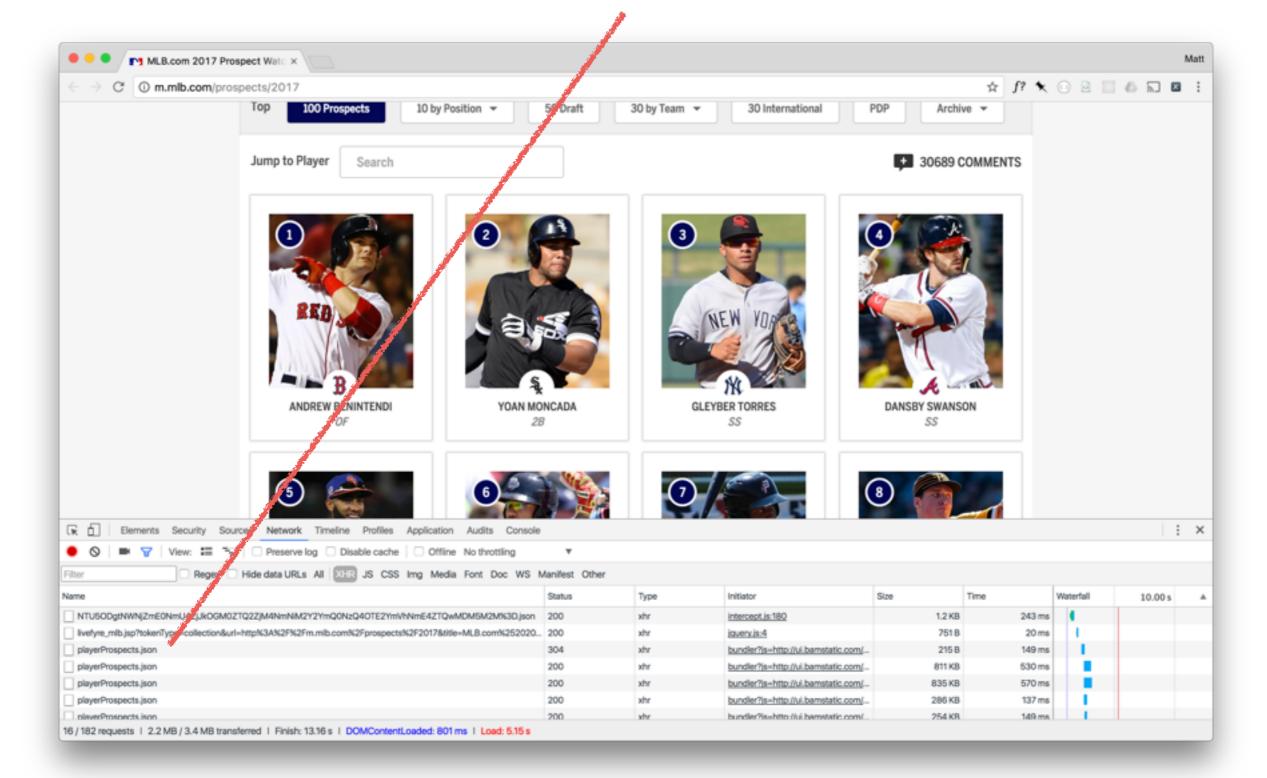
- Again, start by inspecting the source
- Try to find "Benintendi" or "Moncada"



- Again, start by inspecting the source
- Try to find "Benintendi" or "Moncada" in the HTML
- · 'uhh"

- Websites love to load data asynchronously.
 - LOVE to
- Let's head to the Inspector's Network panel to poke around and find the source
 - In Chrome: Ctrl+Shift+I (Windows) or Cmd+Opt+I (Mac), then select "Network"

- Websites love to load data asynchronously.
 - LOVE to
- Let's head to the Inspector's Network panel to poke around and find the source
 - In Chrome: Ctrl+Shift+I (Windows) or Cmd+Opt+I (Mac), then select "Network"
- Let's start by looking under "XHR", the typical place to look for dynamically loaded data



- "playerProspects.json" looks promising
- We know it's a JSON file
- The filename is a pretty dead giveaway
- When we open it up, it has a ton of prospect data

- Here, we have a JSON file
- Let's inspect the structure to find exactly what attributes we would like to scrape
- Fast-forward: the "prospect_players" key has prospects for all teams! And it has the Top 100 under the "prospects" key.

```
{
  "prospect_year": "2017",
  "player_id": 643217,
  "player_first_name": "Andrew",
  "player_last_name": "Benintendi",
  "rank": 1,
  "position": "OF",
  "preseason100": 1,
  "preseason20": 1,
  "team_file_code": "BOS",
}
```

- Using Python's out-of-box JSON parser, we can easily parse this file and extract players
- Code: https://github.com/mattdennewitz/ sloan-scraping/blob/master/mlbtop-100-2017.py
- Output: https://github.com/mattdennewitz/
 sloan-scraping/blob/master/mlb-2017.csv

• • •				1.
rank	id	name	url	¦
	·	· +	+	i
1	643217	Andrew Benintendi	http://m.mlb.com/player/643217	7/
2	660162	Yoan Moncada	http://m.mlb.com/player/660162	
3	650402	Gleyber Torres	http://m.mlb.com/player/650402	
4	621020	Dansby Swanson	http://m.mlb.com/player/621020	
5	642708	Amed Rosario	http://m.mlb.com/player/642708	
6	641487	J.P. Crawford	http://m.mlb.com/player/641487	
7	645302	Victor Robles	http://m.mlb.com/player/645382	
8	607192	Tyler Glasnow	http://m.mlb.com/player/607192	2/
9	648457	Austin Meadows	http://m.mlb.com/player/640457	
10	645277	Ozzie Albies	http://m.mlb.com/player/645277	7/
11	608337	Lucas Giolito	http://m.mlb.com/player/608337	7/
12	641355	Cody Bellinger	http://m.mlb.com/player/641355	5/
13	650391	Eloy Jimenez	http://m.mlb.com/player/650391	1/
14	621052	Alex Reyes	http://m.mlb.com/player/621052	2/
15	663898	Brendan Rodgers	http://m.mlb.com/player/663898	3/
16	656629	Michael Kopech	http://m.mlb.com/player/656629	3/
17	646240	Rafael Devers	http://m.mlb.com/player/646240)/
18	621446	Lewis Brinson	http://m.mlb.com/player/621446	3/
19	666160	Mickey Moniak	http://m.mlb.com/player/666160	9/
20	642564	Francis Martes	http://m.mlb.com/player/642564	4/
21	642715	Willy Adames	http://m.mlb.com/player/642715	5/
22	605548	Bradley Zimmer	http://m.mlb.com/player/605548	3/
23	622534	Manuel Margot	http://m.mlb.com/player/622534	4/
24	648449	Clint Frazier	http://m.mlb.com/player/640449	3/
25	659262	Anderson Espinoza	http://m.mlb.com/player/659262	2/
26	669222	Nick Senzel	http://m.mlb.com/player/669222	2/
27	605137	Josh Bell	http://m.mlb.com/player/605137	7/
28	664023	Ian Happ	http://m.mlb.com/player/664023	
29	641786	Kyle Lewis	http://m.mlb.com/player/641786	3/
30	641999	Corey Ray	http://m.mlb.com/player/641999	3/
31	641703	Brent Honeywell	http://m.mlb.com/player/641703	3/
32	670867	Kevin Maitan	http://m.mlb.com/player/670867	
33	592254	Jose De Leon	http://m.mlb.com/player/592254	4/
34		Vladimir Guerrero Jr		
35		Kyle Tucker	http://m.mlb.com/player/663656	
36		Tyler O'Neill	http://m.mlb.com/player/641933	
37		Blake Rutherford	http://m.mlb.com/player/666164	
38		Josh Hader	http://m.mlb.com/player/623352	
39	608348	Carson Kelly	http://m.mlb.com/player/608348	3/
:				

Recap

- We've used the four step approach to plan for consistent output across disparate systems
- We've used tools like the Inspector to probe for data
- We've written very simple yet powerful scripts in Python to download prospect lists
- We've streamlined the data into a consistent shape
- Our scripts are easily reusable

Next steps

- Since we were clever and included system IDs, we can tie it all together using a baseball player ID registry
 - Chadwick Register
 - Smart Fantasy Baseball
 - Crunchtime

Tools

Hopefully there's time to talk about this

Tools

- requests: A beautiful HTTP library
- Ixml: A beautiful XML and HTML parsing library.
 Tricky to install on Windows, binaries are available.
- BeautifulSoup: another A+ HTML parser
- Scrapy: a very robust Python framework for crawling websites

Code

The code and output from this session is online
 at: https://github.com/mattdennewitz/2017-
 sloan-data-scraping

Thanks!

- Questions?
- If we have some time left, we could try a bit of live coding
- If you have very specific scraping questions, find me after and let's talk