Web Scraping 101 in Python

Learn web scraping and create your own data sets in minutes.

Morten Hegewald
Dec 25, 2019 · 7 min read ★

Use web scraping to fill the holes in your datasets. source: https://pixabay.com/

Web scraping simply means to automatically gather information/data from a website.

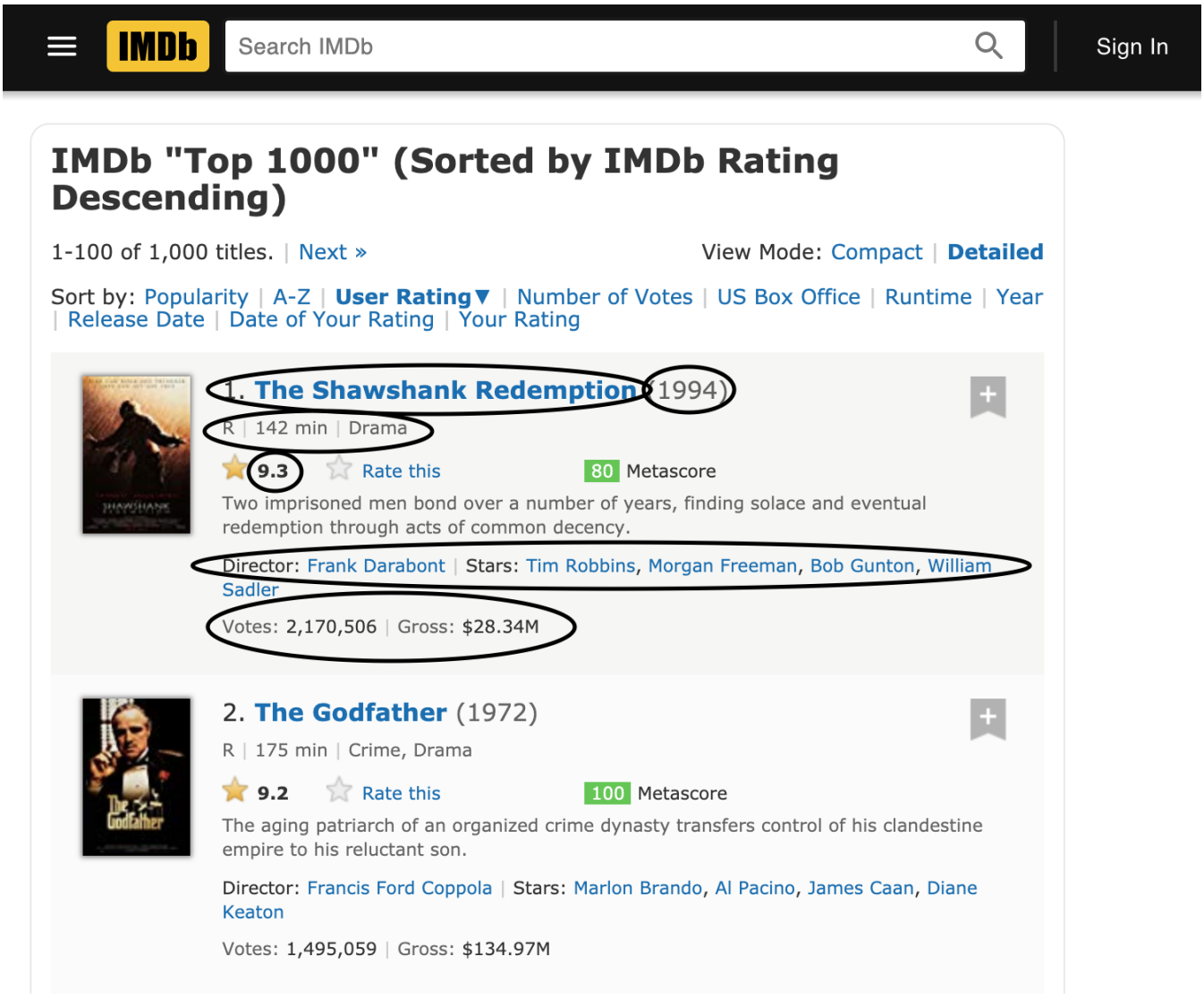
This can be extremely valuable both for an experienced data scientist wanting to add new dimensions to an existing dataset, but also for an inexperienced data scientist in search of interesting datasets to start building their portfolio of projects.

In this short article, we'll go through a simple example of **how to write a Python script to automate turning information on a website into a dataset ready for your data**

It's worth mentioning that even though information/data is available online it's not necessarily legal to use it for commercial purposes — check with the website you intend to scrape before you do so.

. . .

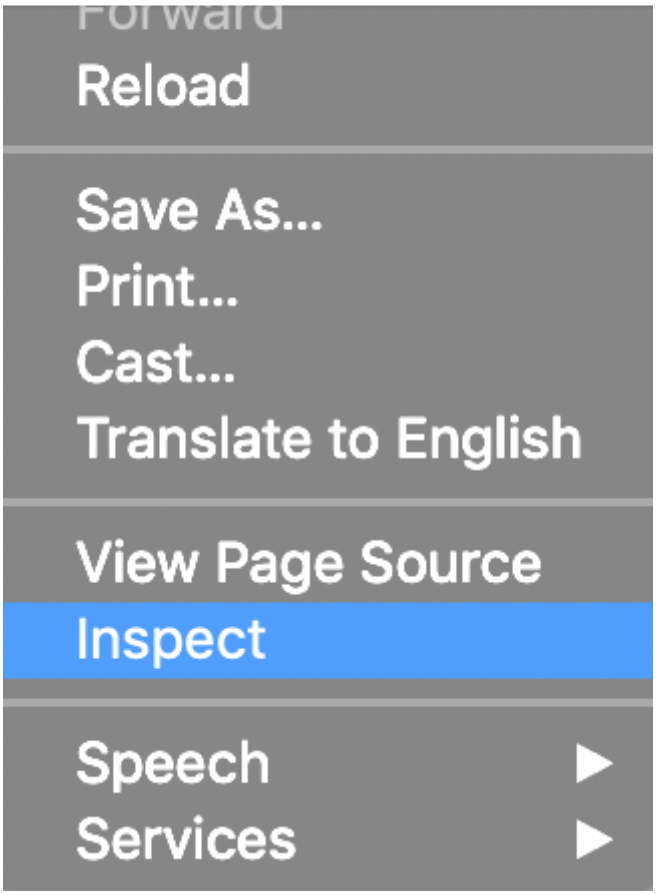
For this tutorial on web scraping we'll go ahead and create a dataset from the 100 highest user rated movies on IMDB:



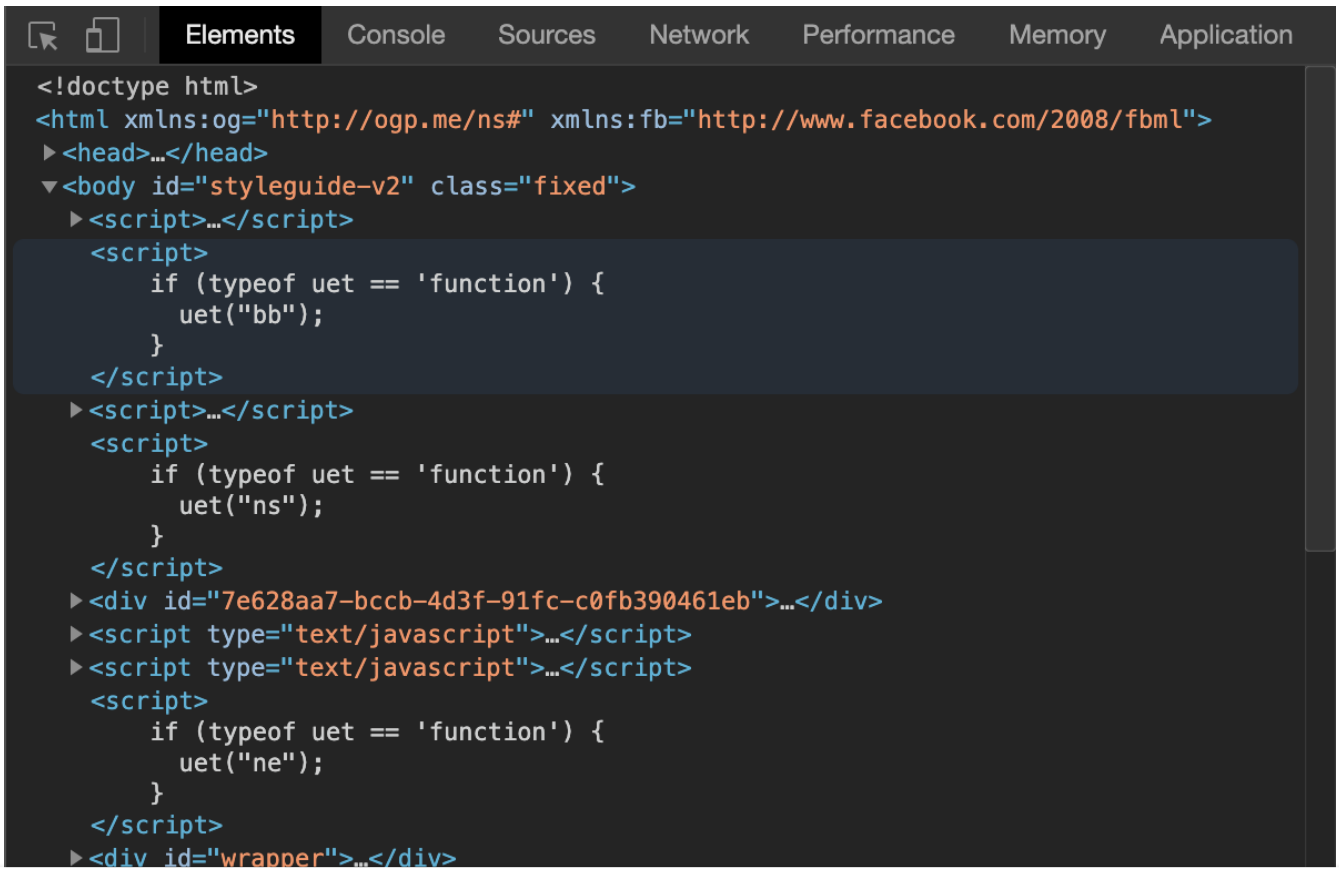
The webpage holds several interesting pieces of information for us to extract

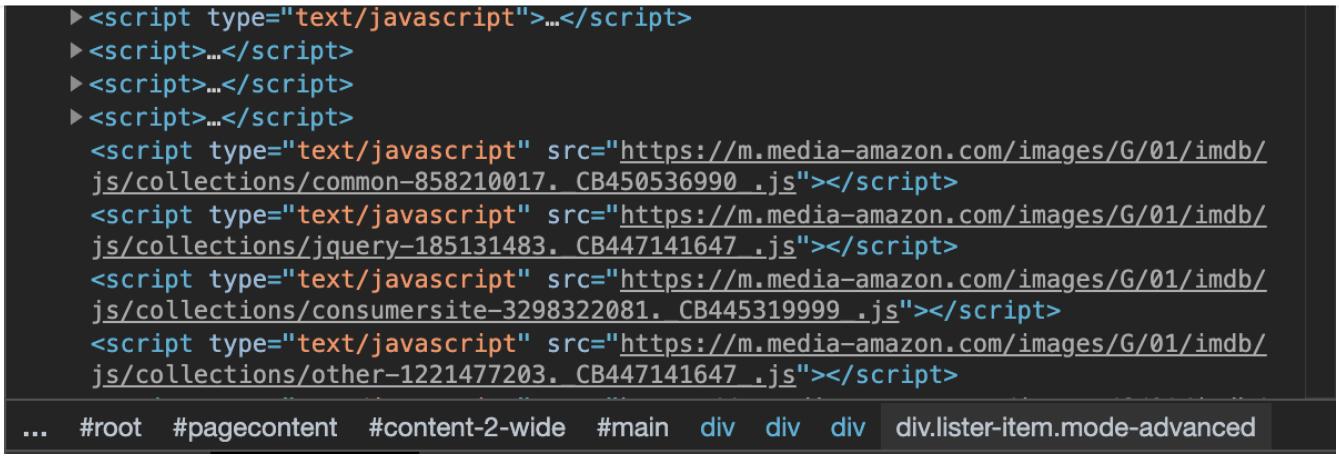
We can scrape this webpage by parsing the HTML of the page and extracting the information we desire for our dataset.

To do this we can right-click anywhere on the page and select "Inspect".



Once this is done the console should pop up displaying the underlying HTML of the part of the page we originally right-clicked:





This can seem a bit overwhelming to parse visually, so to make things easier for us we can make use of the arrow button in the top left corner.

Clicking this button and subsequently clicking any piece of the original webpage will move the console view directly to the HTML for that part of the page.

Let's use that functionality to navigate to the title of the top movie on the webpage. You might be seeing different articles, but the underlying HTML should contain the same patterns. In my case, I'm directed to the following HTML:

```
<h3 class="lister-item-header">…</h3>
```

For this exercise we'll want to extract data points for each movie, so to achieve this we'll start by identifying the main HTML tag for each movie. If I navigate upwards in the HTML tree I can observe that each movie on the page is contained within a tag of the form:

```
<div class="lister-item mode-advanced">
```

We can do a similar thing for any piece of information about each movie we'd like to extract.

Let's start looking at the code.

. . .

With that in mind we have our development environment set up!

Start by open up a terminal session and run the following command to install the packages we'll need:

```
pip install beautifulsoup4 requests pandas
```

- beautifulsoup4: Allows us to parse the HTML of the site and convert it to a BeautifulSoup object, which represents the HTML as a nested data structure.
- requests: The package that allows us to connect the site of choice.
- pandas: The goto Python package for dataset manipulation

We can then import these:

```
import bs4
import pandas as pd
import requests
```

Let's connect to the IMDB top 100 movies webpage, extract the HTML behind it and convert it to a BeautifulSoup object:

```
url = 'https://www.imdb.com/search/title/?
count=100&groups=top_1000&sort=user_rating'

def get_page_contents(url):
    page = requests.get(url, headers={"Accept-Language": "en-US"})
    return bs4.BeautifulSoup(page.text, "html.parser")


soup = get_page_contents(url)
```

After taking a look at the IMDB webpage, we'll set out to extract (all highlighted in the above screenshot of the page):

- Movie title

- Runtime
- Audience rating
- Genre
- IMDB rating
- Number of votes
- Box office earnings
- Director
- Primary actors

Extracting text: Movie titles and release year

Since we've already located the HTML tag containing each movie card, we can get a list of all distinct movies and their corresponding HTML by:

```
movies = soup.findAll('h3', class_='lister-item-header')
```

The `findAll` method creates a list where each entry contains the HTML that's captured within the `h3` tag and `list-item-header` class. By taking a deeper look at the first movies HTML and see that the movie title can be found under the first `a` tag.

To capture this attribute we can loop through all movies and either call `findAll` and grab the first element of the list, or we can use the `find` method which automatically grabs the first tag it finds. Thus, we can construct a list of all movie titles (with a little help from list comprehensions for efficiency) through:

```
titles = [movie.find('a').text for movie in movies]
```

Release years can be found under the tag `span` and class `lister-item-year text-muted unbold`. To grab these, we can follow a similar approach as before:

Extracting runtime, audience rating and genre all follow a similar approach as the above examples.

. . .

Extracting Numerical Values:

In the case of IMDB ratings, number of votes and box office earnings we can see that while these may be available as string values, can also grab the actual numerical values from the `data-value` attribute within each respective tag.

Here's the IMDB rating of The Godfather:

```
<div class="inline-block ratings-imdb-rating" name="ir" data-value="9.2">
```

To grab the `9.2` value from the `data-value` attribute we simply need parse the dictionary that the `find` method returns. In this case we can grab the correct value through:

```
movie.find('div', 'inline-block ratings-imdb-rating')['data-value']
```

In the case of number of votes and earnings we don't have a class attribute to filter for. Here are the number of votes and estimated box office earnings for The Godfather:

```
<span class="text-muted">Votes:</span>
<span name="nv" data-value="1495059">1,495,059</span>
<span class="ghost">|</span>
<span class="text-muted">Gross:</span>
<span name="nv" data-value="134,966,411">$134.97M</span>
```

In this case we'll use a dictionary to filter for the attribute `name='nv'` in our `find` method. However, because there are no apparent distinction between the number of

```
votes = movie.findAll('span' , {'name' : 'nv'})[0]['data-value']
earnings = movie.findAll('span' , {'name' : 'nv'})[1]['data-value']
```

. . .

Nested Values

In the case where the data we need is located within multiple levels of generic tags, we'll need to dig into this nested structure to extract what we need.

In the case of the movie directors and actors we'll need to do just that. From inspecting the HTML we see that the director information is located within an initial `p` tag and thereafter an `a` tag — both without class attributes making it necessary to unnest the data. We'll do this by calling `find` and `findAll` repeatedly.

Since the director is the 1st `a` tag, we can extract this information through:

```
director = movie.find('p').find('a').text
```

and, since the actors always correspond to the remaining `a` tags, we can grab these through:

```
actors = [actor.text for actor in movie.find('p').findAll('a')[1:]]
```

. . .

One Function to Rule Them All

We've now outlined how we'd go about extracting all the data, but how about we construct a function that can be applied across any piece of information we'd want to extract?

In addition to this, we'll create a few conditions to return `None` in the event that a movie doesn't have a genre, box office earnings, etc.

If we do this, extracting all the information we need an turning it into a clean pandas data frame can be done through the following script:

```
import bs4
import pandas as pd
import requests


def numeric_value(movie, tag, class_=None, order=None):
    if order:
        if len(movie.findAll(tag, class_)) > 1:
            to_extract = movie.findAll(tag, class_)[order]['data-value']
        else:
            to_extract = None
    else:
        to_extract = movie.find(tag, class_)['data-value']

    return to_extract


def text_value(movie, tag, class_=None):
    if movie.find(tag, class_):
        return movie.find(tag, class_).text
    else:
        return


def nested_text_value(movie, tag_1, class_1, tag_2, class_2, order=None):
    if not order:
        return movie.find(tag_1, class_1).find(tag_2, class_2).text
    else:
        return [val.text for val in movie.find(tag_1, class_1).findAll(tag_2, class_2)[order]]


def extract_attribute(soup, tag_1, class_1='', tag_2='', class_2='',
                      text_attribute=True, order=None, nested=False):
    movies = soup.findAll('div', class_='lister-item-content')
    data_list = []
    for movie in movies:
        if text_attribute:
```

```
            else:
                data_list.append(text_value(movie, tag_1, class_1))
        else:
            data_list.append(numeric_value(movie, tag_1, class_1, order))

    return data_list


titles = extract_attribute(soup, 'a')
release = extract_attribute(soup, 'span', 'lister-item-year text-muted unbold')
audience_rating = extract_attribute(soup, 'span', 'certificate')
runtime = extract_attribute(soup, 'span', 'runtime')
genre = extract_attribute(soup, 'span', 'genre')
imdb_rating = extract_attribute(soup, 'div', 'inline-block ratings-imdb-rating', False)
votes = extract_attribute(soup, 'span' , {'name' : 'nv'}, False, 0)
earnings = extract_attribute(soup, 'span' , {'name' : 'nv'}, False, 1)
directors = extract_attribute(soup, 'p', '', 'a', '', True, 0, True)
actors = extract_attribute(soup, 'p', '', 'a', '', True, slice(1, 5, None), True)


df_dict = {'Title': titles, 'Relase': release, 'Audience Rating': audience_rating,
           'Runtime': runtime, 'Genre': genre, 'IMDB Rating': imdb_rating,
           'Votes': votes, 'Box Office Earnings': earnings, 'Director': directors,
           'Actors': actors}
df = pd.DataFrame(df_dict)
```

	Title	Relase	Audience Rating	Runtime	Genre	IMDB Rating	Votes	Box Office Earnings	Director	Actors
0	The Shawshank Redemption	(1994)	R	142 min	\nDrama	\n\n9.3\n	2,170,529	2,170,529	Frank Darabont	[Tim Robbins, Morgan Freeman, Bob Gunton, Will...
1	The Godfather	(1972)	R	175 min	\nCrime, Drama	\n\n9.2\n	1,495,081	1,495,081	Francis Ford Coppola	[Marlon Brando, Al Pacino, James Caan, Diane K...
2	The Dark Knight	(2008)	PG-13	152 min	\nAction, Crime, Drama	\n\n9.0\n	2,154,722	2,154,722	Christopher Nolan	[Christian Bale, Heath Ledger, Aaron Eckhart, ...
3	The Godfather: Part II	(1974)	R	202 min	\nCrime, Drama	\n\n9.0\n	1,046,642	1,046,642	Francis Ford Coppola	[Al Pacino, Robert De Niro, Robert Duvall, Dia...
4	The Lord of the Rings: The Return of the King	(2003)	PG-13	201 min	\nAdventure, Drama, Fantasy	\n\n8.9\n	1,544,248	1,544,248	Peter Jackson	[Elijah Wood, Viggo Mortensen, Ian McKellen, O...
5	Pulp Fiction	(1994)	R	154 min	\nCrime, Drama	\n\n8.9\n	1,704,367	1,704,367	Quentin Tarantino	[John Travolta, Uma Thurman, Samuel L. Jackson...

There are several data outputs here that need a bit of cleaning before any meaningful work can be done, but for the exercise of extracting a data set through web scraping our work is complete.