Muhammad Satrio Pinoto Negoro

Linkedin: https://www.linkedin.com/in/satriopino/

Webscrapping Movie Ratings from IMDb using BeautifulSoup

Web scraping, also known as web harvesting or web data extraction, is the process of automatically extracting data from websites. It involves fetching a web page and extracting data from it. The data can be parsed, searched, reformatted, and copied into a spreadsheet or loaded into a database. Web scraping can be done manually, but in most cases, automated tools are preferred as they can be less costly and work at a faster rate. Web scraping is used for various purposes, including lead generation, price monitoring, market research, and content aggregation. However, some websites use methods to prevent web scraping, such as detecting and disallowing bots from crawling their pages. In response, there are web scraping systems that rely on using techniques in DOM parsing, computer vision, and natural language processing to simulate human browsing to enable gathering web page content for offline parsing.

Dependencies

Actually to follow this module you only need to install beautifulsoup4 with `pip install beautifulsoup4` and you are good to go. But here some libraries that needed to be installed first that I use at bis module :

- beautifulSoup4
- pandas
- matplotlibs

Background

At this project I try to scrap Movie Name, Ratings, Duration, Votes, and Sinopsis Data **IMDb** data center website. IMBD, an acronym for Internet Movie Database, is an online database of information related to films, television series, podcasts, home videos, video games, and streaming content online. It includes information such as cast, production crew, personal biographies, plot summaries, trivia, ratings, and fan and critical reviews. IMDb was first published in 1990 by Col Needham, a computer programmer, and has since grown into a comprehensive resource for movie and TV information. I will try to scrap this sites for educational purpose only.

A lot of you might ask why we need to scrap this data from the sites. I want to make a report and gain some insight from that data and maybe can be useful for others. To do that I need to have the data, and scrapping is a good way to collect the data I don't have from the public.

I will scrap 7 points from this sites. That is Title, Year, Duration, Rating Category, Ratings, and Total Votes.

What is BeautifulSoup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. Beautiful Soup 3 only works on Python 2.x, but Beautiful Soup 4 also works on Python 3.x. Beautiful Soup 4 is faster, has more features, and works with third-party parsers like lxml and html5lib.

Since beautifulsoup used to pull the data out of a HTML, so first I need to pull out the html first. How I do it? I will use default library `request` .

So all this code is doing is sending a GET request to spesific address I give. This is the same type of request your browser sent to view this page, but the only difference is that Requests can't actually render the HTML, so instead you will just get the raw HTML and the other response information.

I'm using the .get() function here, but Requests allows you to use other functions like .post() and .put() to send those requests as well. At this case we will going to the IMDb website, you can click here to follow what exactly that link goes to.

Requesting the Data and Creating a BeautifulSoup

Let's begin with requesting the web from the site with `get` method.

Getting the HTML from the Webpage

In [36]:
```python
url = 'https://www.imdb.com/chart/top/?ref_=nv_mv_250'
```

In [37]:
```python
import requests
```

In [38]:
```python
headers = {
    'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36'
}
url_get = requests.get(url, headers=headers)
```

To visualize what exactly you get from the `request.get` , we can use .content so ee what we exactly get, in here i slice it so it won't make our screen full of the html we get from the page. You can delete the slicing if you want to see what we fully get.

In [39]:
```python
url_get.content[:500]
```

Out[39]:
```
b'<!DOCTYPE html><html lang="en-US" xmlns:og="http://opengraphprotocol.org/schema/" xmlns:fb="http://www.facebook.com/2008/fbml"><head><meta charSe
t="utf-8"/><meta name="viewport" content="width=device-width"/><script>if(typeof uet === \'function\'){ uet(\'bb\', \'LoadTitle\', {wb: 1}); }</scr
ipt><script>window.addEventListener(\'load\', (event) => {\n        if (typeof window.csa !== \'undefined\' && typeof window.csa === \'function\')
{\n            var csaLatencyPlugin = window.csa(\'Content\', {\n            '
```

As we can see we get a very unstructured and complex html, which actually contains the codes needed to show the webpages on your web browser. But we as human still confused what and where we can use that piece of code, so here where we use the beautifulsoup. Beautiful soup class will result a beautifulsoup object. Beautiful Soup transforms a complex HTML document into a complex tree of Python objects. But we'll only ever have to deal with about four kinds of objects: `Tag` , `NavigableString` , `BeautifulSoup` , and `Comment` . But at this project we will only use `BeautifulSoup` .

Let's make Beautiful soup object and feel free to explore the object here.

In [40]:
```python
from bs4 import BeautifulSoup

soup = BeautifulSoup(url_get.content,"html.parser")
print(type(soup))
```

```
<class 'bs4.BeautifulSoup'>
```

Let's see how our beautifulsoup looks like. As you can see, the content is the same with our get_url object but it's tidier. Also beautifulsoup give us method to make it even more prettier, for tidyness purpouse we slice to only see first 500 character.

In [41]:
```python
print(soup.prettify()[:500])
```

```
<!DOCTYPE html>
<html lang="en-US" xmlns:fb="http://www.facebook.com/2008/fbml" xmlns:og="http://opengraphprotocol.org/schema/">
 <head>
  <meta charset="utf-8"/>
  <meta content="width=device-width" name="viewport"/>
  <script>
   if(typeof uet === 'function'){ uet('bb', 'LoadTitle', {wb: 1}); }
  </script>
  <script>
   window.addEventListener('load', (event) => {
        if (typeof window.csa !== 'undefined' && typeof window.csa === 'function') {
            var csaLatencyPlugin = window.csa(
```

Getting right key to extract right information

Now we already have a tidier html, now we should search the lines that we want to use. Let's back to our web page first.

The information that we need are the Date, Inflation value MoM, and Inflation value YoY which contain in the table. To know which part of the code refer to that table, we can just move our cusor there, right click, and inspect element. Then we will see something like this.

From inspect element we know that we need to find the line table with class `baris-scroll` . We can use the find method at our beautifusoup object. Let's also call our object to see what we get.

Finding the right key to scrap the data & Extracting the right information

Find the key and put the key into the `.find()` Put all the exploring the right key at this cell. (please change this markdown with your explanation)

1. Find Title

In [136…
```python
title = soup.find_all("h3", attrs={'class':'ipc-title__text'})[250].text
title.split('. ')[1]
```

Out[136]:
```
'Drishyam'
```

2. Find Year, Duration, Rating Category

In [137…
```python
soup.find_all("span", attrs={'class':'sc-479faa3c-8 bNrEFi cli-title-metadata-item'})[749]
```

Out[137]:
```
<span class="sc-479faa3c-8 bNrEFi cli-title-metadata-item">13+</span>
```

In [ ]:

3. Find Ratings & Total Votes

In [138…
```python
soup.find_all("span", attrs={'class':'ipc-rating-star ipc-rating-star--base ipc-rating-star--imdb ratingGroup--imdb-rating'})[1].text
```

Out[138]:
```
'9.2\xa0(2M)'
```

In [140…
```python
temp = soup.find_all("span", attrs={'class':'ipc-rating-star ipc-rating-star--base ipc-rating-star--imdb ratingGroup--imdb-rating'})[249].text
temp = temp.replace('(', '')
temp = temp.replace(')','')
temp
```

Out[140]:
```
'8.2\xa092K'
```

In [141…
```python
temp.split('\xa0')[0]
```

Out[141]:
```
'8.2'
```

In [142…
```python
temp.split('\xa0')[1]
```

Out[142]:
```
'92K'
```

As you can see get already get the necessary key to extract all needed data. To get the only text information you can add `.text` . Remember you need to only get one information before you use `.text` otherwise it will return error.

Extracting the Information

Now all the `beautiful soup` part is over. All left to do is doing some programming to extract all the data automaticly, you can do this manualy at this part but if your data too long I advice you use loop. I'll show you how to make looping for extracting the data, but before that let's check how long is our data to help our looping process. Since `find_all` will always return data in format list, we will use `len()` to check how long is our list.

Finding row length of Date.

In [62]:
```python
title_row = soup.find_all("h3", attrs={'class':'ipc-title__text'})
len(title_row)
```

Out[62]:
```
263
```

BUT the actual Title Data only exist in index list [1:251]

In [68]:
```python
second_info = soup.find_all("span", attrs={'class':'sc-479faa3c-8 bNrEFi cli-title-metadata-item'})
len(second_info)
```

Out[68]:
```
750
```

In [69]:
```python
third_info = soup.find_all("span", attrs={'class':'ipc-rating-star ipc-rating-star--base ipc-rating-star--imdb ratingGroup--imdb-rating'})
len(third_info)
```

Out[69]:
```
250
```

Now we know the length of our data, now here what we will do for the looping process.

In [143…
```python
Title = []
Year = []
Duration = []
Rating_Category = []
Ratings = []
Total_Votes = []
```

In [144…
```python
for i in range(1,251):
    judul = soup.find_all("h3", attrs={'class':'ipc-title__text'})[i].text
    judul = judul.split('. ')[1]

    Title.append(judul)
```

In [145…
```python
n = 0
while n < 750:
    tahun = soup.find_all("span", attrs={'class':'sc-479faa3c-8 bNrEFi cli-title-metadata-item'})[n].text
    n+=1

    durasi = soup.find_all("span", attrs={'class':'sc-479faa3c-8 bNrEFi cli-title-metadata-item'})[n].text
    n+=1

    ratcat = soup.find_all("span", attrs={'class':'sc-479faa3c-8 bNrEFi cli-title-metadata-item'})[n].text
    n+=1

    Year.append(tahun)
    Duration.append(durasi)
    Rating_Category.append(ratcat)
```

In [146…
```python
for i in range(0,250):
    base = soup.find_all("span", attrs={'class':'ipc-rating-star ipc-rating-star--base ipc-rating-star--imdb ratingGroup--imdb-rating'})[i].text
    base = base.replace('(', '')
    base = base.replace(')','')
    rating = base.split('\xa0')[0]
    total_votes = base.split('\xa0')[1]

    Ratings.append(rating)
    Total_Votes.append(total_votes)
```

Then after we fix our list a bit, as usual we will input it to pandas' dataframe.

Creating data frame & Data wrangling

Put the array into dataframe

In [147…
```python
data = {
    'Title': Title,
    'Year': Year,
    'Duration': Duration,
    'Rating_Category': Rating_Category,
    'Ratings': Ratings,
    'Total_Votes': Total_Votes
}
```

In [179…
```python
import pandas as pd

df = pd.DataFrame(data)
df
```

Out[179]:

	Title	Year	Duration	Rating_Category	Ratings	Total_Votes
0	The Shawshank Redemption	1994	2h 22m	18+	9.3	2.8M
1	The Godfather	1972	2h 55m	18+	9.2	2M
2	The Dark Knight	2008	2h 32m	R	9.0	2.8M
3	The Godfather: Part II	1974	3h 22m	18+	9.0	1.3M
4	12 Angry Men	1957	1h 36m	SU	9.0	841K
...	...	...	...	...	...	...
245	Les quatre cents coups	1959	1h 39m	Not Rated	8.1	125K
246	Aladdin	1992	1h 30m	G	8.0	453K
247	Persona	1966	1h 25m	Not Rated	8.1	128K
248	Dances with Wolves	1990	3h 1m	PG-13	8.0	284K
249	Drishyam	2015	2h 43m	13+	8.2	92K

250 rows × 6 columns

Let's check our dataframe data types to see if our data is useable.

In [180…
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Title            250 non-null    object
 1   Year             250 non-null    object
 2   Duration         250 non-null    object
 3   Rating_Category  250 non-null    object
 4   Ratings          250 non-null    object
 5   Total_Votes      250 non-null    object
dtypes: object(6)
memory usage: 11.8+ KB
```

Usual stuff, we can clean the data or save it to csv let's do a bit cleaning so we can do a bit of visualisation. We will change the inflation to float datatype, but before we can do that we need to change the "," to "." first. To do this we can use the help of `str.replace()` . Then lastly let's fix our period data type.

In [181…
```python
def convert_duration(duration_str):
    hours, minutes = 0, 0
    if 'h' in duration_str:
        hours = float(duration_str.split('h')[0])
    if 'm' in duration_str:
        minutes = float(duration_str.split('m')[0].split()[-1])
    return hours * 60 + minutes

df['Duration'] = df['Duration'].apply(convert_duration)
```

In [182…
```python
def convert_total_votes(votes_str):
    multiplier = 1
    if votes_str[-1] == 'K':
        multiplier = 1000
    elif votes_str[-1] == 'M':
        multiplier = 1e6
    return int(float(votes_str[:-1]) * multiplier)

df['Total_Votes'] = df['Total_Votes'].apply(convert_total_votes)
```

In [195…
```python
df[['Duration', 'Ratings', 'Total_Votes']] = df[['Duration', 'Ratings', 'Total_Votes']].astype('float')
df['Rating_Category'] = df['Rating_Category'].astype('category')
```

In [202…
```python
a = [df['Rating_Category'].nunique()]
a
```

Out[202]:
```
[22]
```

In [196…
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Title            250 non-null    object
 1   Year             250 non-null    object
 2   Duration         250 non-null    float64
 3   Rating_Category  250 non-null    category
 4   Ratings          250 non-null    float64
 5   Total_Votes      250 non-null    float64
dtypes: category(1), float64(3), object(2)
memory usage: 10.8+ KB
```

The results of the data after cleaning the data.

In [203…
```python
df.head(15)
```

Out[203]:

	Title	Year	Duration	Rating_Category	Ratings	Total_Votes
0	The Shawshank Redemption	1994	142.0	18+	9.3	2800000.0
1	The Godfather	1972	175.0	18+	9.2	2000000.0
2	The Dark Knight	2008	152.0	R	9.0	2800000.0
3	The Godfather: Part II	1974	202.0	18+	9.0	1300000.0
4	12 Angry Men	1957	96.0	SU	9.0	841000.0
5	Schindler's List	1993	195.0	R	9.0	1400000.0
6	The Lord of the Rings: The Return of the King	2003	201.0	A	9.0	1900000.0
7	Pulp Fiction	1994	154.0	17+	8.9	2200000.0
8	The Lord of the Rings: The Fellowship of the Ring	2001	178.0	R	8.8	2000000.0
9	The Good, the Bad and the Ugly	1966	161.0	D	8.8	795000.0
10	Forrest Gump	1994	142.0	21	8.8	2200000.0
11	Fight Club	1999	139.0	D17+	8.8	2300000.0
12	The Lord of the Rings: The Two Towers	2002	179.0	A	8.8	1700000.0
13	Inception	2010	148.0	D	8.8	2500000.0
14	Star Wars: Episode V - The Empire Strikes Back	1980	124.0	PG	8.7	1400000.0

To export the Web-Scrapping result regarding the inflation rate, we can use "variable that contain result data".to_csv("Name of file", index=False)

In [204…
```python
df.to_csv("IMDb Top 250 Movies Data.csv", index=False)
```

Check the Output Data

In [205…
```python
pd.read_csv("IMDb Top 250 Movies Data.csv").head()
```

Out[205]:

	Title	Year	Duration	Rating_Category	Ratings	Total_Votes
0	The Shawshank Redemption	1994	142.0	18+	9.3	2800000.0
1	The Godfather	1972	175.0	18+	9.2	2000000.0
2	The Dark Knight	2008	152.0	R	9.0	2800000.0
3	The Godfather: Part II	1974	202.0	18+	9.0	1300000.0
4	12 Angry Men	1957	96.0	SU	9.0	841000.0