Konrad Lewszyk, student id: 437008

IMDB top 250 ranking scraper

For this project I decided to scrape data on all films placed in the top 250 ranking of the IMDB website. IMDB is the most popular online movie database. Every avid movie fan knows the site and can visit it for latest news, trailers, interviews or announcements regarding the movie industry. The ratings the movies are given on the website by the users are widely used as a gauge of a movie's quality, along with other sites like Metacritic or Rotten Tomatoes. IMDB stores the top 250 movies ever ranked in a list, and the term 'imdb top 250' is well known.

As a film fanatic myself, I decided to scrape data on each title in the 250 list. The information I was looking for was title, year of release, budget, box office, genres, directors' names, rating of the movie and finally its runtime.

Before I go on to explain the mechanics and efficiency of each of my scrapers, I want to clarify that for each scraper I started on the homepage of the IMDB, and then scraped the link to get to the 250 ranking.

Beautiful Soup – I start on the homepage of IMDB, and then I iterate through link object in the dropdown menu, look for the one with text 'Top Rated Movies', obtain that link and set it as my new url. Next I select all the movie link elements, and store them in a list. Before I scrape the information on the movies, I create an empty dataframe with appropriate column names. Finally, I iterate through the list, each time setting the particular as my current bs object. For each link I scrape the beforementioned information. I had to make some special cases for particular movie attributes.

For budget, in most cases I was able to scrape the number invested with a filter with a specified string, but some movies turn out to have undisclosed information on budget. This was the case as well with the box office. In such cases I set these attributes to None.

For runtime, some movies did not have this detail specified in the technical section like most movies. In such cases I scraped the information from the header, where runtime was displayed in hour and minutes format. I just had to multiply the hours by 60 and add the minutes.

Scraping the information on directors was a bit tricky as well. Depending on whether there was one director or more than one, the page would specify the section with title "director" or "directors:". Here I also made an exception.

Once I obtained all the information on a movie, I store the data in a dictionary and then append it to the dataframe. Once I iterate through all the movies, my scraped data is ready to be written into a csv file

Scrapy – I wrote three spiders. I used scrapy in a similar way we did in class. I scrape needed links, and feed it to the next spider. My first spider "link_to_ranking_spider" writes a csv file with a row – link to the top 250 ranking. My second scraper named 'links_spider opens that csv file and loads the url. Then the spider scrapes all the movie links in the ranking by using xpath and selecting the href attribute. I write the links to a csv file for my last spider.

In the last of my spiders, the “movie_info_spider” I firstly create a movie class, with 8 attributes set up as scrapy fields. Next I iterate through the list, each time creating a new movie object. I write out an xpath for each of the 8 attributes and just like previously I have to make exceptions for the attributes box office, budget, runtime, directors, but this time I change the xpath. In the end I yield the movie object. Running the spider with spider_name -o filename.csv outputs the data.

Selenium – this scraper has a very similar structure to my beautiful soup scraper, excepts it uses the driver function and searches for information using the xpaths. I start at the homepage, get the xpath for the top 250 ranking link object, scrape the links, create a list, set up an empty dataframe, iterate through the links, add new entries and finally write out my csv file. Same rules applied when it comes to the exception cases.

100 limit – for all the scrapers I set up a Boolean at the start that limits the list length to 100.

Data

The data output from all the scrapers is practically identical. The only difference is that my spiders scraped the data with English titles, while Selenium and Beautiful Soup returned polish titles. Additionally, while selenium and bs maintained the order of the movies by ratings, scrapy did not. The order of the columns also was different in scrapy for some reason.

The data, before limiting to 100, consists of 250 rows and 8 columns with the following column names and data type:

Title – string

Year – integer

Rating – integer

Runtime – integer

Budget - integer

Box_office – integer

Genre – string

Director – string

The only missing values was present for movies that did not disclose all the data

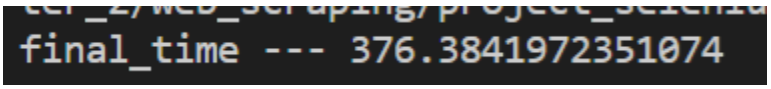
Performance – runtime

Beautiful soup

Scrapy

I tried to measure time with code, and it worked for my first two spiders, but the last one for some reason did not write out the final measured time to my txt file, so I measured the time manually, and all three spiders together scraped the data in 28.15 seconds

Selenium



With 28.15 seconds, scrapy wins by far.

Data Analysis – Here I present that the scraped data can be used for data analysis

```
In [3]: df.head()
```

Out[3]:

	title	year	rating	budget	box_office	runtime	genre	director
0	Skazani na Shawshank	1994	9.3	25000000.0	2.881729e+07	142	Drama	Frank Darabont
1	Ojciec chrzestny	1972	9.2	6000000.0	2.461210e+08	175	Crime, Drama	Francis Ford Coppola
2	Ojciec chrzestny II	1974	9.0	13000000.0	4.803578e+07	202	Crime, Drama	Francis Ford Coppola
3	Mroczny rycerz	2008	9.0	185000000.0	1.005974e+09	152	Action, Crime, Drama, Thriller	Christopher Nolan
4	Dwunastu gniewnych ludzi	1957	9.0	350000.0	9.550000e+02	96	Crime, Drama	Sidney Lumet

```
In [4]: df.tail()
```

Out[4]:

	title	year	rating	budget	box_office	runtime	genre	director
95	Obywatel Kane	1941	8.3	839727.0	1600719.0	119	Drama, Mystery	Orson Welles
96	Dangal	2016	8.4	700000000.0	303723636.0	161	Action, Biography, Drama, Sport	Nitesh Tiwari
97	Idź i patrz	1985	8.3	NaN	20929068.0	142	Drama, Thriller, War	Elem Klimov
98	Brzdąc	1921	8.3	250000.0	26916.0	68	Comedy, Drama, Family	Charles Chaplin
99	Deszczowa piosenka	1952	8.3	2540800.0	1865056.0	103	Comedy, Musical, Romance	Stanley Donen, Gene Kelly

```
In [7]: df.describe()
```

Out[7]:

	year	rating	budget	box_office	runtime
count	100.000000	100.00000	9.400000e+01	9.900000e+01	100.000000
mean	1987.710000	8.51100	8.622642e+07	2.855156e+08	134.640000
std	23.702404	0.20641	2.661120e+08	4.353338e+08	30.529178
min	1921.000000	8.30000	2.500000e+05	9.550000e+02	68.000000
25%	1974.750000	8.40000	3.725000e+06	1.401059e+07	114.500000
50%	1994.000000	8.50000	1.900000e+07	1.012096e+08	130.500000
75%	2004.500000	8.60000	7.000000e+07	3.859164e+08	153.250000
max	2020.000000	9.30000	2.400000e+09	2.797501e+09	229.000000

```
fig, ax = plt.subplots(figsize = (18,5))
width = 0.4
x = np.arange(len(highest_gross.title))
ax.bar(x, highest_gross.budget, width = width)
ax.bar(x + width, highest_gross.box_office, width = width)
ax.ticklabel_format(useOffset=False, style='plain')
ax.set_xticklabels(highest_gross.title)
ax.set_xticks(x + width/2)
ax.xaxis.set_tick_params(rotation=80)
ax.set_title('Box_office to budget comparison');
```

```
<ipython-input-29-c8bfeb82f4e7>:7: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(highest_gross.title)
```