UNIVERSITY OF WARSAW | FACULTY OF ECONOMIC SCIENCES

WEB SCRAPING AND SOCIAL MEDIA SCRAPING

Project Report

Submitted By:

Natalia Miela

Nurdan Beşli

Mehmet Tiryaki

INTRODUCTION

The focus of our web scraping project is to extract data from the film ranking section of the website filmweb.pl. Filmweb is a popular online platform that provides comprehensive information about movies, including rankings, reviews, and details about individual films. The film ranking page on filmweb.pl presents a curated list of movies based on various criteria such as popularity, user ratings, and reviews.

Our goal is to gather data from this ranking section, which can be used for analysis, research, and other purposes. By scraping this web page, we can collect information about the films' titles, publication years, durations, rankings, and genres, enabling us to gain a deeper understanding of the film industry and trends. The specific webpage we target is located at https://www.filmweb.pl/ranking/film?fbclid=IwAR0qC8KuR4Z159TFqObUBnv3jETlH6yKX7tgtovXJqxrMRs3bG0veCpA4wQ. This webpage presents a ranking of the best movies from around the world.

Our objective is to gather comprehensive data about the top 125 films listed on this webpage. We begin by scraping the links to these films' individual pages. By scraping data from these film pages, our aim is to collect specific information about each film, such as its title, publication year, duration, ranking, and genre. By combining web scraping techniques with the available data on filmweb.pl, we can gather information about the individual films and create a smaller dataset that provides a snapshot of the films' attributes.

SCRAPER MECHANISM

1. Beautiful Soup

BeautifulSoup is a Python library that provides an easy and efficient way to extract data from web pages. It creates a parse tree from the given HTML or XML source, allowing users to navigate and search for specific elements or data within the document. With its intuitive and convenient methods, BeautifulSoup simplifies the process of web scraping and data extraction. Here we would like to introduce the process of using BeautifulSoup to scrap movies data.

First, we define the `scrape_movie_data()` function. This function takes a movie URL as input and utilizes the requests library to send an HTTP request to the movie page. We then use BeautifulSoup to parse the HTML content of the page.

Inside the `scrape_movie_data()` function, we use BeautifulSoup to extract specific details from the parsed HTML. We locate the movie title, ranking, year, genre, and duration by identifying the corresponding HTML elements and their attributes. For example, we use `soup.find()` with appropriate class names or tag names to locate and extract the desired information.

After extracting the movie details, we store them in a dictionary for each movie. The dictionary includes keys such as 'title', 'ranking', 'year', 'genre', and 'duration', with their corresponding values. We initialize an empty list called `films` to store the scraped movie data. The code then sends a GET request to the main ranking page of the movie website using the requests library.

Using BeautifulSoup, we parse the HTML content of the ranking page and locate the movie list. We identify the HTML elements that contain the movie links. We iterate over each movie element in the movie list and extract the movie link using BeautifulSoup. We concatenate the movie link with the website's base URL to obtain the complete movie URL. For each movie URL, we call the `scrape_movie_data()` function to collect the movie details. The extracted details are stored as dictionaries and appended to the `films` list.

After scraping the movie data, we create a pandas DataFrame called `df` from the list of dictionaries. The DataFrame provides a structured tabular format for efficient data manipulation and analysis.

BeautifulSoup provides an intuitive and user-friendly API, making it accessible to both beginners and experienced developers. The library offers various methods for searching and extracting specific tags or elements from a document based on tag names, attributes, or CSS selectors.Moreover, BeautifulSoup seamlessly integrates with libraries like requests and pandas, enabling users to build comprehensive web scraping and data analysis workflows.

However, BeautifulSoup may not be the fastest parsing library, which can be a consideration in scenarios where performance is critical. Parsing large HTML or XML files with BeautifulSoup can consume significant resources, impacting performance and memory usage. And most importantly, BeautifulSoup relies on the website's structure and may require code updates if the structure changes frequently. Thus, it should not be used for dynamic pages.

2. Scrapy

Scrapy is a powerful Python framework designed for efficient web scraping and data extraction tasks. In our project, we utilized Scrapy to scrape movie data from the Filmweb website. The Scrapy implementation consists of two spiders: "link_lists" and "films".

The "link_lists" spider is responsible for collecting links to movies from the website. It starts by visiting the URL "https://www.filmweb.pl/ranking/film?fbclid=IwAR0qC8KuR4Z159TFqObUBnv3jETlH6yKX7tgtovXJqxrMRs3bG0veCpA4wQ". It uses XPath to select the links from the page by targeting the HTML element with the class "rankingType__title" and extracting the "href" attribute. The spider then appends the extracted links to the base URL "https://www.filmweb.pl" and creates instances of the "Link" item class. Finally, the spider yields these items to return the scraped data.

The website's ranking of films is spread across multiple pages, and the films are loaded dynamically as the user scrolls down the pages. To ensure comprehensive data collection, we employ a technique in our web scraping process that involves visiting multiple pages on the website. This is accomplished through the implementation of the start_requests function, which is a part of the Scrapy framework. This allows us to access and scrape the films beyond the initial page. Each generated URL follows a similar pattern: "https://www.filmweb.pl/ranking/film?page={page_number}". The "page_number" variable represents the specific page we want to scrape.

Within the "start_requests" method, we use Scrapy's "yield" statement along with the "scrapy.Request" function to send the requests to the generated URLs. The "callback=self.parse" argument specifies that the response from each request should be passed to the "parse" method for further processing. Without "start_requests" method, we would only be able to scrape the films on the first page. That is why we create this method to ensure that we scrape all the films available on the website. By employing this approach, we can effectively navigate through multiple pages of the film ranking and extract data from a more extensive collection of films.

The "films" spider, on the other hand, reads a list of URLs from a CSV file named "links.csv". It starts by visiting each URL and extracts various details about the movies. The spider uses XPath to locate specific elements on the page and retrieves information such as the movie name, publication year, duration, ranking, and genre. Similar to the "link_lists" spider, it creates instances of the "Film" item class and yields them to return the scraped data.

Both spiders are part of the Scrapy framework, which simplifies the process of web scraping. They leverage Scrapy's built-in functionalities, such as the ability to make HTTP requests, parse HTML responses using XPath, and store the scraped data in a structured format.

To run the code, follow these steps:

1. Make sure you have Scrapy installed.
2. Create a new Scrapy project.
3. Place the spiders provided in the appropriate "spiders" folder within your Scrapy project directory.
4. Open a terminal or command prompt and navigate to your Scrapy project directory.
5. Execute the command: scrapy crawl link_lists -o links.csv to run the "link_lists" spider and save the extracted links to a CSV file named "links.csv".
6. After the above step, execute the command: scrapy crawl films -o films.csv to run the "films" spider and save the scraped movie data to a CSV file named "films.csv".
7. Verify the generated CSV files ("link_list.csv" and "films.csv") to check if the data has been successfully scraped.

In this project, the scraped data is stored in CSV format for easy access and analysis. At the end of the main Python code file, we utilize the pandas library to read the scraped data into a dataframe. By employing pandas and printing the dataframe, we enhance the clarity and readability of the collected data, facilitating further analysis and interpretation.

Scrapy offers several advantages for web scraping tasks. It provides efficiency and performance, making it suitable for large-scale scraping projects. The framework is scalable and flexible, allowing customization and adaptation to diverse scraping requirements. Additionally, Scrapy benefits from a thriving community and a rich ecosystem with numerous extensions and libraries available.

However, there are some considerations when using Scrapy. It has a steeper learning curve compared to simpler scraping libraries, and familiarity with its concepts may require initial effort, especially for beginners. Scrapy relies on the structure and consistency of the target website, and changes in the website's structure may require updates to the scraping code. For simple and straightforward scraping tasks, the comprehensive feature set of Scrapy may introduce unnecessary complexity.

3. Selenium

Selenium is a widely used open-source framework for automating web browsers. It provides a convenient and powerful toolset for web scraping and automating web-based tasks. In our project, we employed Selenium to create a scraper for extracting movie data from the "filmweb.pl" website. This Python script utilizes Selenium's capabilities to navigate web pages, interact with elements, and extract desired information. By leveraging Selenium's automation capabilities, we were able to efficiently collect movie details without manual intervention.

In this selenium scraper, a webdriver for Google Chrome is used. The Python script is designed to automate the process of scraping movie data from the website "filmweb.pl". This script is divided into multiple functions to create an efficient workflow.

Here is the breakdown of the functions:

1. get_driver(): This function initializes a Google Chrome webdriver. It uses selenium's Options class to configure the webdriver. The 'headless' option is set, which means that the browser will run in the background and not display a GUI. This is beneficial as it decreases resource use and allows the script to run on servers that don't support GUIs. The function also sets an implicit wait of 10 seconds. An implicit wait tells WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. This 10-second window allows enough time for elements on the webpage to load properly, ensuring that the script does not throw an error if the internet connection is slow or the website server is under heavy load. The configured driver object is then returned for further use in the script.
2. scrape_links(driver, limit_pages=True): This function is responsible for navigating the film ranking pages on "filmweb.pl" and collecting the links to individual film detail pages. It uses a loop that, by default, iterates through the first 5 pages of the ranking (it can be set to scrape up to 500 pages). For each page, it uses the get() method of the driver object to navigate to the page URL. The URL contains a variable component that corresponds to the page number in the ranking, allowing it to loop through the pages sequentially. Once on a page, it locates the film links using their CSS selectors '.efficientPoster.rankingType__poster.EfficientPoster a', which represent the 'href' attributes of the films' poster elements. For each located element, it retrieves the 'href' attribute which is the URL of the film's detail page and appends it to a list. This list of links is then returned by the function.
3. scrape_details(driver, links): This function is designed to visit each link collected by the scrape_links function and extract specific details about each film. The details extracted include the film's name, year, duration, ranking, and genre. It achieves this by using the get() method of the driver to navigate to each link, and then uses the find_element() method with specific CSS selectors to locate and retrieve the text of each detail. Each film's details are stored in a dictionary for structured access, and the dictionaries are appended to a list which is returned by the function.

The script then executes these functions sequentially. It starts by getting the configured driver using get_driver(). This driver is then used to scrape the film links via scrape_links(driver). The links are stored in a list, which is then used as input for the scrape_details(driver, links) function, which extracts and compiles the film details.

The final part of the script converts the list of film details (each item a dictionary) into a pandas DataFrame. The pandas library provides powerful data manipulation and analysis capabilities, and by converting the scraped data into a DataFrame, you can easily perform subsequent analysis or manipulation tasks. For example, you could sort films by year or ranking, or filter them based on genre.

In its final step, the script prints out the DataFrame, which provides a tabular representation of the scraped data. Each row represents a film, and each column represents a specific detail (name, year, duration, ranking, and genre). To execute the script, you can navigate to the directory where the script is located and use the command python3 selenium.py

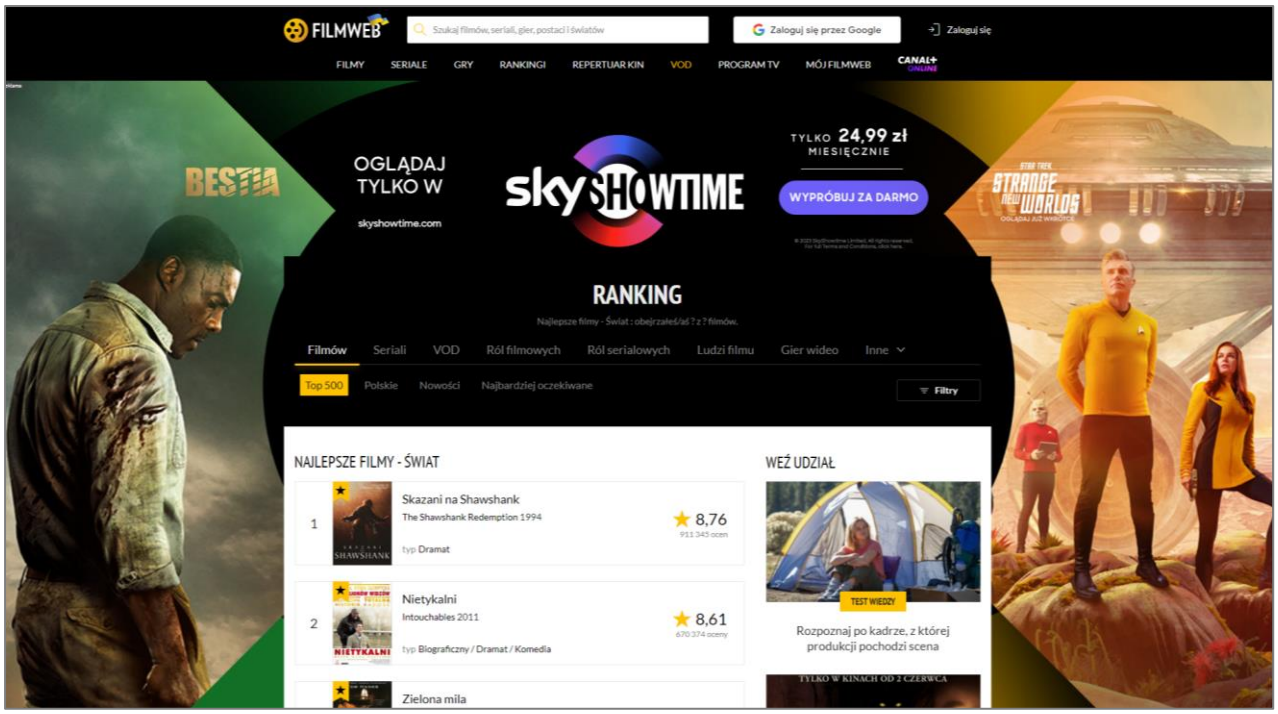
This selenium script efficiently extracts and structures data from "filmweb.pl" and does not require user interaction once it's running, making it suitable for automated data collection tasks. It's designed to scrape public data efficiently and could be extended or modified to handle more complex scraping tasks.

Selenium offers several advantages for web scraping and automation tasks. Firstly, it supports multiple web browsers, including Google Chrome, Firefox, and Microsoft Edge, allowing flexibility in choosing the browser for scraping. Selenium's ability to simulate real user interactions, such as clicking buttons, filling out forms, and scrolling, makes it suitable for scraping websites with dynamic content or requiring user authentication. Additionally, Selenium provides robust support for JavaScript-heavy websites, as it can execute JavaScript code and retrieve dynamically generated content. This makes it a powerful tool for scraping modern web applications.

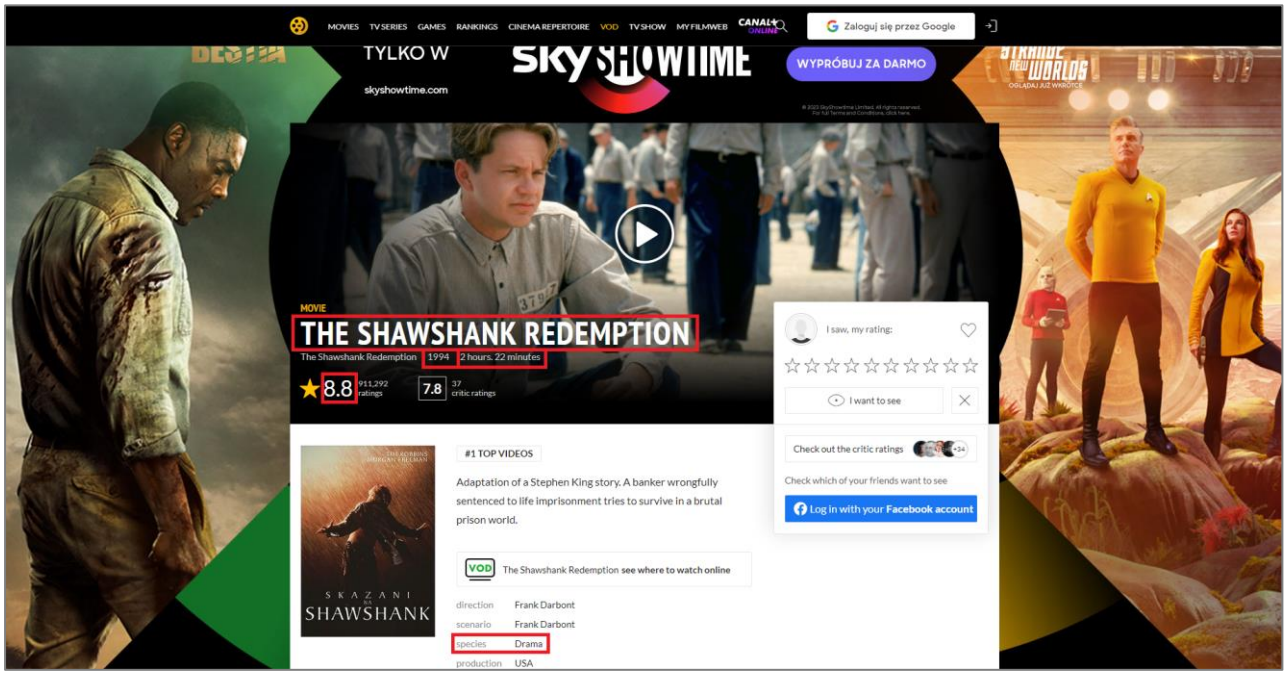
However, there are some considerations when using Selenium. One notable aspect is its reliance on web browsers, which adds overhead in terms of resource consumption and browser-specific configurations. Selenium requires a compatible WebDriver for the chosen browser, and these drivers need to be installed and maintained. Moreover, as Selenium interacts with web browsers, the scraping process is slower compared to pure HTML parsing libraries like BeautifulSoup. Selenium is best suited for scenarios where website interaction or JavaScript execution is necessary, rather than simple static HTML scraping.

TECHNICAL DESCRIPTION OF OUTPUT

Our web scraping project from filmweb.pl focuses on extracting five key details about the top 125 films listed in the film ranking section of the website.



The scraping process involves gathering the page links of the top 125 films from the URL: https://www.filmweb.pl/ranking/film?fbclid=IwAR0qC8KuR4Z159TFqObUBnv3jETlH6yKX7tgtovXJqxrMRs3bG0veCpA4wQ.



For each film link, we extract the following information:

- Title: The title of the film (stored as object).
- Publication Year: The year the film was released (stored as integer).
- Duration: The duration of the film (stored as object).
- Ranking: The ranking score of the film (stored as object).
- Genre: The genre(s) of the film (stored as object).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125 entries, 0 to 124
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   duration          125 non-null    object
 1   genre             125 non-null    object
 2   movie_name        125 non-null    object
 3   publication_year  125 non-null    int64
 4   ranking           125 non-null    object
dtypes: int64(1), object(4)
memory usage: 5.0+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125 entries, 0 to 124
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   link    125 non-null    object
dtypes: object(1)
memory usage: 1.1+ KB
```

The DataFrame has a total memory usage of approximately 5.0 KB. The scraped data from filmweb.pl does not contain any null variables. The DataFrames for both film links and film details consists of 125 film records, and each column in the DataFrame has a non-null count of 125. This indicates that all the variables in the dataset have been successfully extracted and populated with valid values.

The table below shows the information of the first 10 film links in the DataFrame.

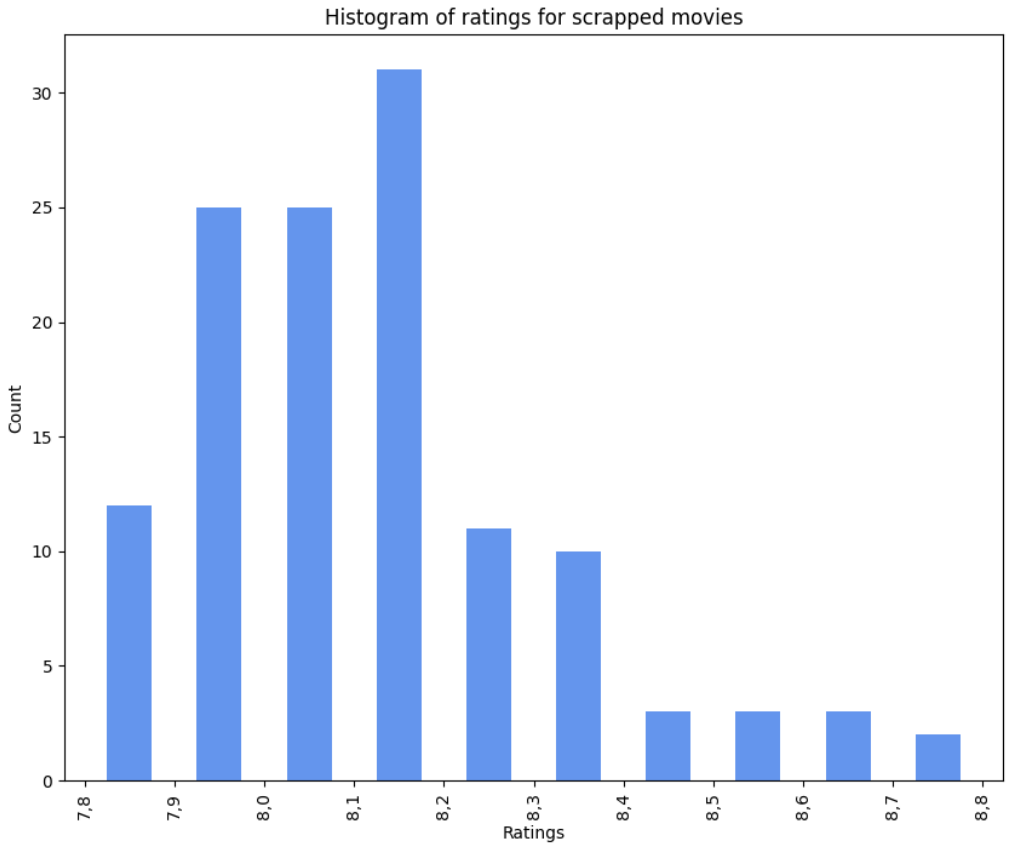
	link
0	https://www.filmweb.pl/film/Skazani+na+Shawsha...
1	https://www.filmweb.pl/film/Nietykalni-2011-58...
2	https://www.filmweb.pl/film/Zielona+mila-1999-862
3	https://www.filmweb.pl/film/Ojciec+chrzestny-1...
4	https://www.filmweb.pl/film/Dwunastu+gniewnych...
5	https://www.filmweb.pl/film/Forrest+Gump-1994-998
6	https://www.filmweb.pl/film/Lot+nad+kuku%C5%82...
7	https://www.filmweb.pl/film/Ojciec+chrzestny+I...
8	https://www.filmweb.pl/film/W%C5%82adca+Pier%C...
9	https://www.filmweb.pl/film/Lista+Schindlera-1...

The table below shows the information of the first 10 movies in the DataFrame.

	duration	genre	movie_name	publication_year	ranking
0	2 godz. 22 min.	Dramat	Skazani na Shawshank	1994	8,8
1	1 godz. 36 min.	Dramat sądowy	Dwunastu gniewnych ludzi	1957	8,7
2	2 godz. 13 min.	Dramat,Psychologiczny	Lot nad kukułczym gniazdem	1975	8,5
3	2 godz. 55 min.	Dramat,Gangsterski	Ojciec chrzestny	1972	8,6
4	3 godz. 21 min.	Fantasy,Przygodowy	Władca Pierścieni: Powrót króla	2003	8,4
5	3 godz. 8 min.	Dramat	Zielona mila	1999	8,6
6	3 godz. 15 min.	Dramat,Wojenny	Lista Schindlera	1993	8,4
7	2 godz. 22 min.	Dramat,Komedia	Forrest Gump	1994	8,5
8	3 godz. 20 min.	Dramat,Gangsterski	Ojciec chrzestny II	1974	8,5
9	1 godz. 52 min.	Biograficzny,Dramat,Komedia	Nietykalni	2011	8,6

ELEMENTARY DATA ANALYSIS

We aimed to analyze the trend of production years and rankings in our dataset by creating a histogram.

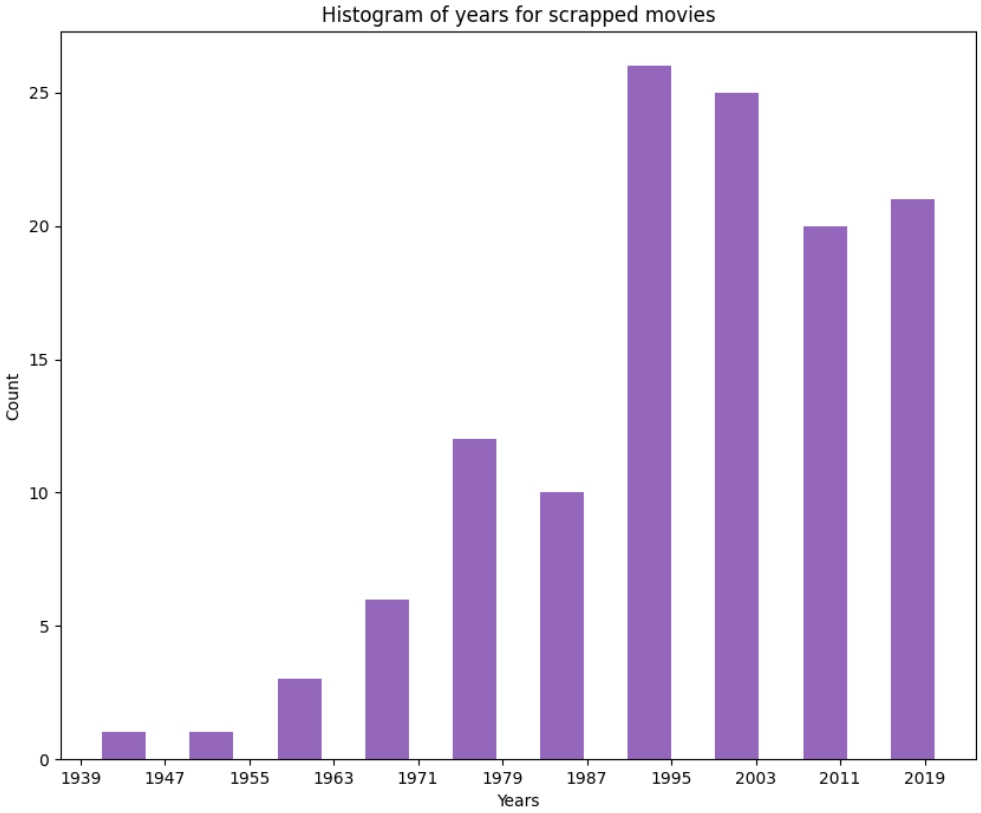


The histogram represents the rankings of the top 125 liked films from around the world. The x-axis denotes the film rankings, while the y-axis represents the frequency or the number of films falling into each ranking category.

From the histogram, we can observe that the majority of films in this dataset received high ratings, with a significant number of movies having a ranking of 8.0, 8.1, or 7.9. This indicates that the films in this selection are generally well-received and highly regarded by viewers.

There is a gradual decline in the frequency of films as we move towards higher rankings, suggesting that fewer movies achieve exceptional ratings. However, we still see a notable presence of films with rankings ranging from 8.2 to 8.8, indicating a considerable number of highly acclaimed films among the top 125.

Overall, the histogram demonstrates a positive trend in the rankings of these films, with the majority falling into the higher rating categories. This suggests that the top liked films from around the world, as represented in this dataset, are of high quality and have garnered significant praise from viewers.



The histogram provides a visual representation of the distribution of movies based on their release years in the film ranking data from Filmweb.pl. The x-axis denotes the film production year, while the y-axis represents the frequency or the number of films falling into each year range. By comparing the different time periods, we can observe the following trends:

- The years between 1939 and 1955 have a relatively low number of movies, with only 2 films released during this period.
- The number of movies starts to increase from 1956 to 1971, with a gradual growth in the film industry.
- The period between 1972 and 1987 shows a significant rise in the number of movies, indicating a period of prolific film production.
- From 1988 to 2003, the number of movies remains relatively high, suggesting a continued active film industry.
- The years between 2004 and 2011 show a slight decrease compared to the previous period but still maintain a considerable number of movies.
- Finally, from 2012 to the present, we observe a relatively stable number of movies, indicating a consistent output in recent years.

Overall, the histogram demonstrates the changing landscape of film production over time, highlighting periods of growth and stability in the industry.

TASK DIVISION

Task	Responsible
Beautiful Soup	Natalia Miela
Scrapy	Nurdan Beşli
Selenium	Mehmet Tiryaki
Project Finalization & Report	All