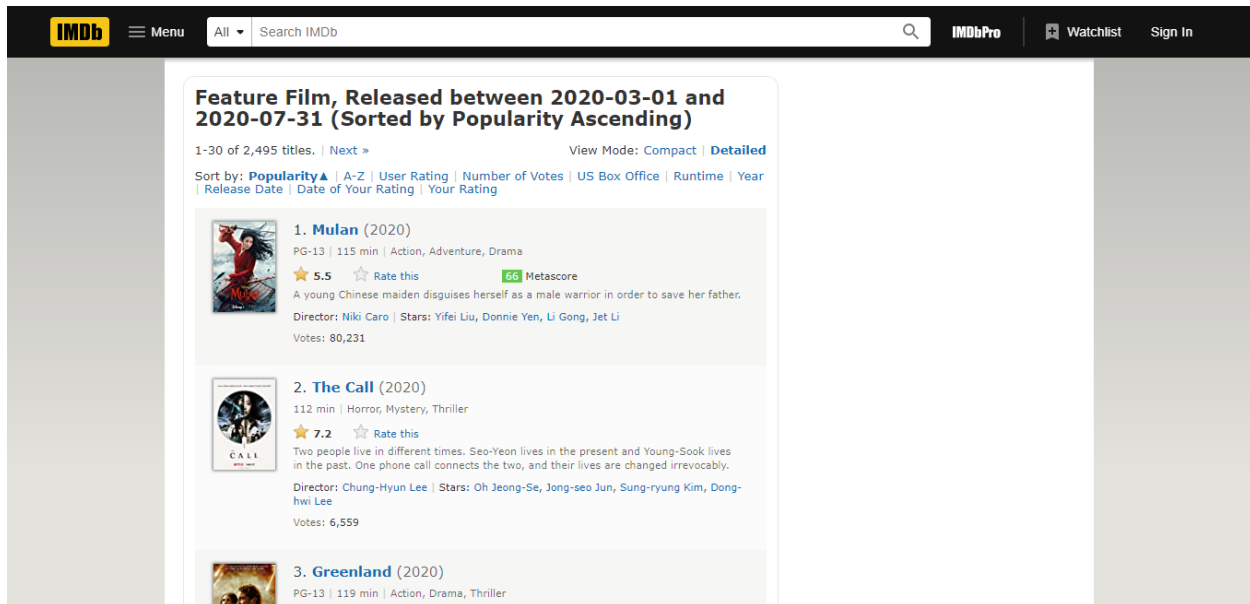


Web Scraping With R

Introduction

Not all websites provide you with an API that you can work with and so the alternative to fetching data from such websites is to get the data through web scraping. The goal of this project is to demonstrate an end to end web scraping workflow with R. The web page we are going to be scraping looks like this



We want to extract data on the movie titles, the year it was released, the runtime, the genre, the user rating, metacore and the vote. At the end of this project we will have dataframe containing values for the aforementioned data points.

```
# loading the libraries
```

```
library(rvest)
library(tidyverse)
library(kableExtra)
```

```
# function to render tibbles as pdf tables
```

```
render_table <- function(table, scale_down=F){
  if(scale_down == T){
    rendered_table <- kbl(table) %>% kable_styling(
      latex_options = c("stripe", "HOLD_position", "scale_down")
    )
  } else{
    rendered_table <- kbl(table) %>% kable_styling(
      latex_options = c("stripe", "HOLD_position")
    )
  }
  return(rendered_table)
}
```

Inspecting The Web Page HTML

The most important thing to do when web scraping is to inspect the html of the web page in your browser. This enables us to determine where we can extract the data we are looking for. To do this all we need to do is right click on the value we want to extract and then click the inspect option. This should open a tab showing the html source code of the web page.

The top screenshot shows the IMDb search results for 'Feature Film, Released between 2020-03-01 and 2020-07-31 (Sorted by Popularity Ascending)'. A right-click context menu is open over the first result, 'Mulan (2020)', with the 'Inspect' option highlighted. The bottom screenshot shows the same page with the browser's developer tools open. The 'Elements' panel shows the HTML structure, and the 'Styles' panel shows the CSS rules for the selected element.

Feature Film, Released between 2020-03-01 and 2020-07-31 (Sorted by Popularity Ascending)

1-30 of 2,495 titles. | Next » View Mode: Compact | Detailed

Sort by: Popularity | A-Z | User Rating | Number of Votes | US Box Office | Runtime | Year | Release Date | Date of Your Rating | Your Rating

1. **Mulan** (2020)
PG-13 | 115 min | Action, Adventure, Drama
★ 5.5 | Rate this | 68 Metascore
A young Chinese maiden disguises herself as a male warrior in order to save her father.
Director: Niki Caro | Stars: Yifei Liu, Donnie Yen, Li Gong, Jet Li
Votes: 80,231

2. **The Call** (2020)
112 min | Horror, Mystery, Thriller
★ 7.2 | Rate this
Two people live in different times. Seo-Yeon lives in the present and Young-Sook lives in the past. One phone call connects the two, and their lives are changed irrevocably.
Director: Chung-Hyun Lee | Stars: Oh Jeong-Se, Jong-seo Jun, Sung-ryung Kim, Dong-hwi Lee
Votes: 6,559

3. **Greenland** (2020)
PG-13 | 119 min | Action, Drama, Thriller

Elements

```
<div class="list-item-image float-left"></div>
<div class="list-item-content">
  <h3 class="list-item-header">
    <span class="list-item-index unbold text-primary">1.
  </span>
  <a href="/title/tt4566758/?ref=adv_li_tt">Mulan</a>
  <span class="list-item-year text-muted unbold">(2020)
</h3>
  <p class="text-muted">
    <span class="certificate">PG-13</span>
    <span class="ghost">|</span>
    <span class="runtime">115 min</span>
    <span class="ghost">|</span>
    <span class="genre">Action, Adventure, Drama
  </p>
</div>
```

Styles

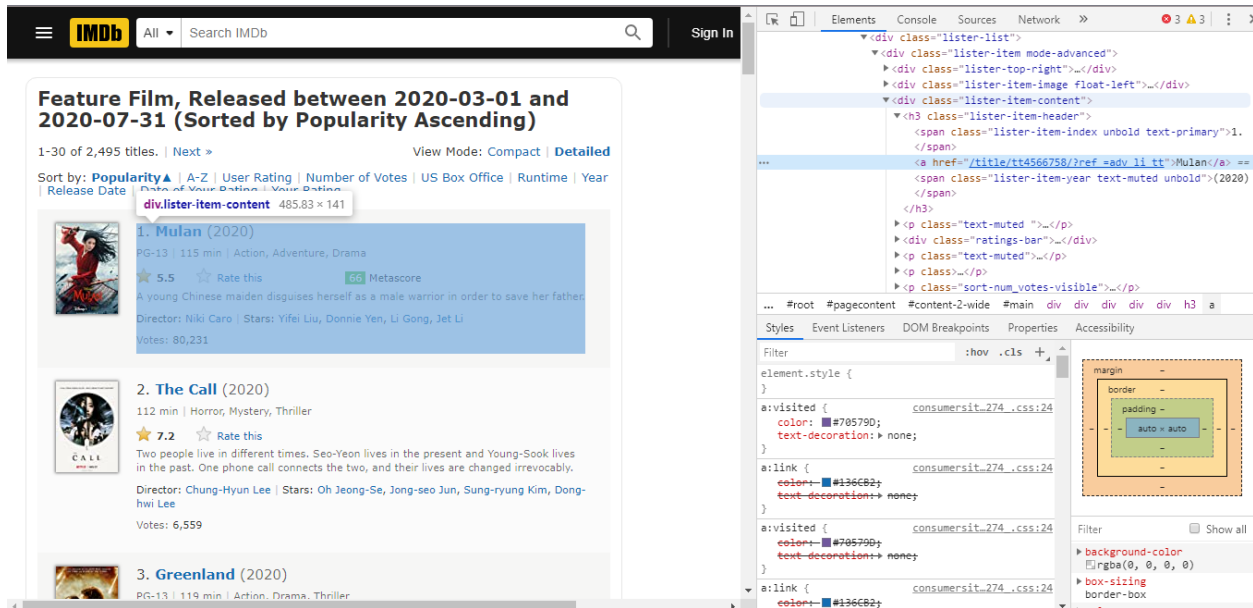
```
element.style {
}
.list-item.mode-advanced>.list-item-content {
  max-width: 84%;
}
.list-item.mode-advanced>.list-item-image, .list-item.mode-advanced>.list-item-content {
  display: inline-block;
  vertical-align: top;
}
.list-item.mode-advanced>.list-item-content {
  max-width: 84%;
}
```

The data we need are nested in different html tags with different CSS classes or ids (CSS selectors). Identifying these tags, their CSS classes or ids will help us extract the right data. It is important to note that css classes are preceded by a dot (.) while ids are preceded by the # symbol.

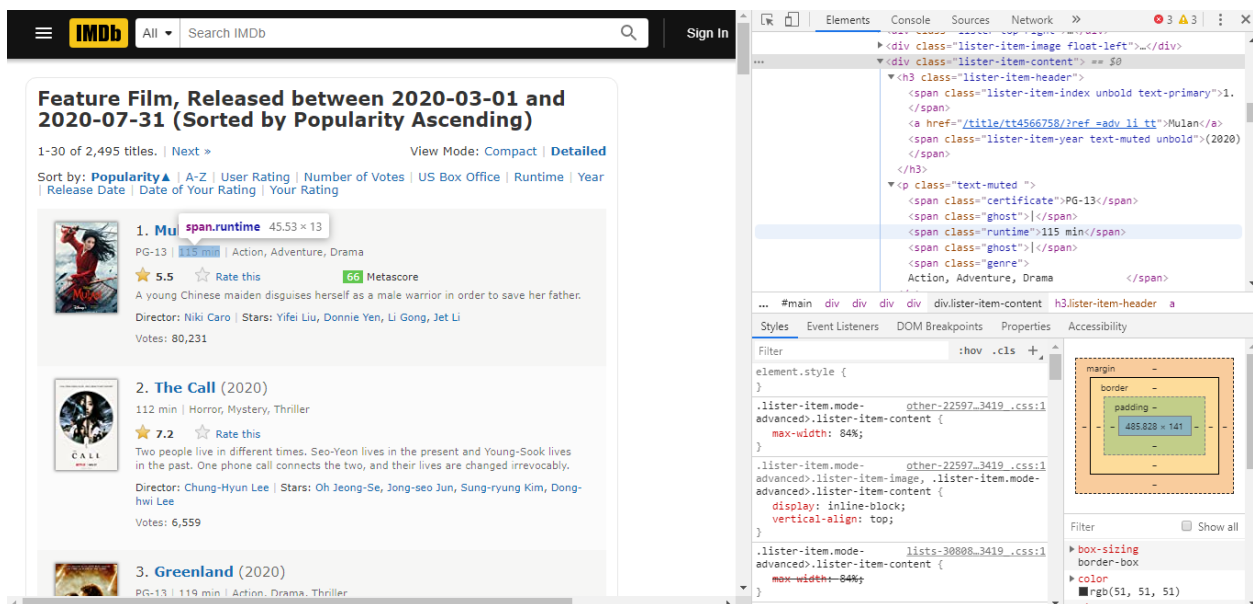
Parsing The HTML

Now that we know how to inspect a web page html, we need to use the `rvest` library to parse the the html tags, CSS selectors containing the required data. The approach we are going to take to extract the data is as follows:

1. We read the web page into an r object using the `read_html()` function.
2. We identify the tag containing data for the movies. This is a `div` tag with a class of `list-item-content`. We use the `html_nodes()` function to extract every element with this `div` and class `list-item-content` and store it in a variable called `movies`.



3. We create empty vectors for each data point, loop through every of the element in `movies`, pass in the appropriate `html` tag or `CSS` selector into the `html_node()` function to extract data contained within that tag or `CSS` selector. The text data is extracted using `html_text()` function. We clean the extracted data using `stringr` functions or `readr` `parse_number()` function for numeric data. This data is then added to the appropriate vector we created. The reason behind using a `for` loop is so that elements without any data is represented as `NA` when parsed. For example the runtime data is contained in a `span` tag with class `runtime`, so to extract the runtime data we pass the argument `span.runtime` into the `html_node()` function. Extracting the runtime will look like this : `movies %>% html_node("span.runtime") %>% html_text()`.



4. The vectors are combined into a tibble, so we can visualise the result.

Extracting the data

```
# reading the web page
web_content <- read_html(
  "http://dataquestio.github.io/web-scraping-pages/IMDb-DQgp.html"
)

# storing the movies data
movies <- web_content %>% html_nodes("div.lister-item-content")

# creating vectors to store the different data points
titles <- c()
years <- c()
runtimes <- c()
genres <- c()
metascores <- c()
user_ratings <- c()
votes <- c()

# looping through movies and adding each data to the appropriate vector
for (i in 1:length(movies)){
  title <- movies[i] %>% html_node("a") %>% html_text()
  year <- movies[i] %>% html_node(".lister-item-year") %>% html_text() %>% parse_number()
  runtime <- movies[i] %>% html_node("span.runtime") %>% html_text()
  genre <- movies[i] %>% html_node("span.genre") %>% html_text() %>% str_replace("\n", "") %>% str_trim()
  user_rating <- movies[i] %>% html_node(".ratings-imdb-rating") %>% html_text() %>% parse_number()
  metascore <- movies[i] %>% html_node(".ratings-metascore") %>% html_text() %>% parse_number()
  vote <- movies[i] %>% html_node(".sort-num_votes-visible") %>% html_text() %>% parse_number()

  titles <- c(titles, title)
  year <- c(years, year)
  runtimes <- c(runtimes, runtime)
  genres <- c(genres, genre)
  user_ratings <- c(user_ratings, user_rating)
  metascores <- c(metascores, metascore)
  votes <- c(votes, vote)
}

# combining the vectors into a tibble
movies_df <- tibble(
  title = titles,
  year = years,
  runtime = runtimes,
  genre = genres,
  user_rating = user_ratings,
  metascore = metascores,
  votes = votes
)

movies_df %>% head() %>% render_table()
```

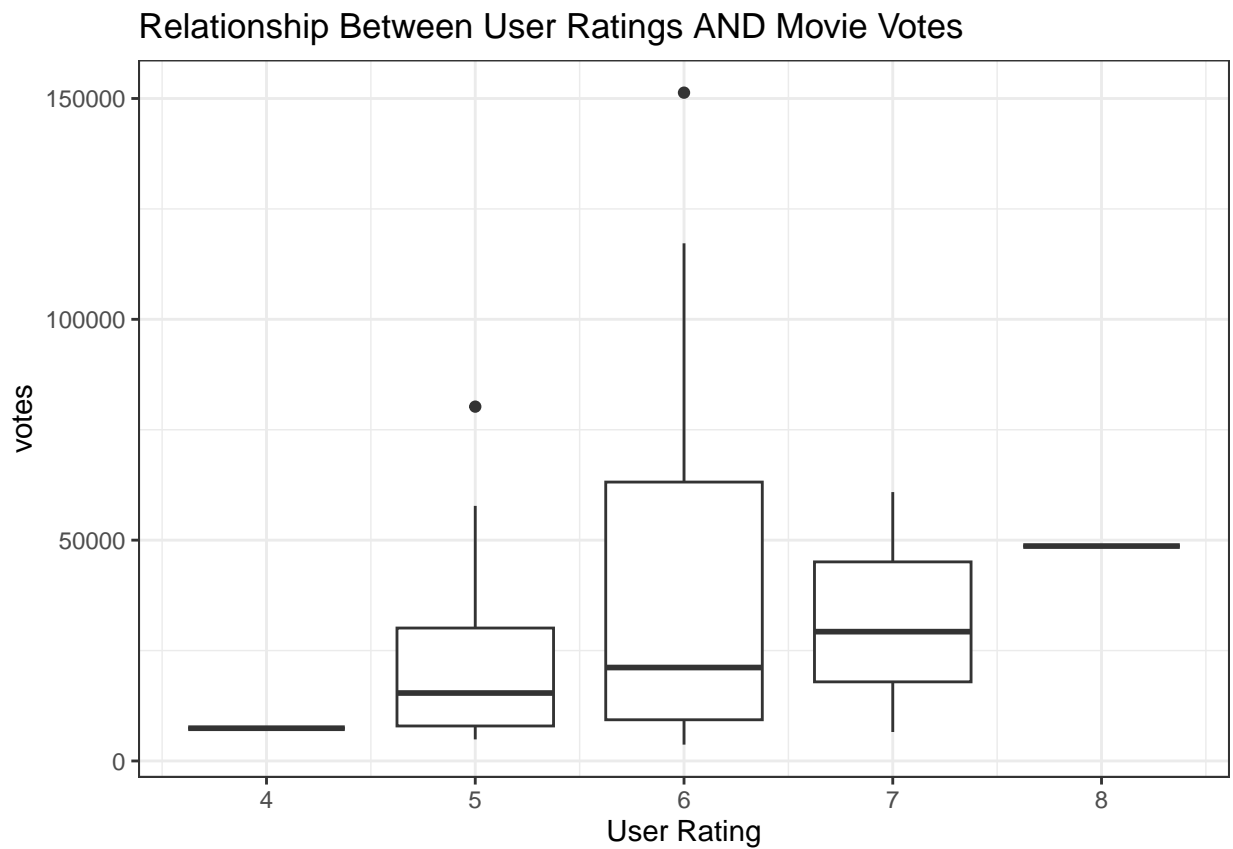
title	runtime	genre	user_rating	metascore	votes
Mulan	115 min	Action, Adventure, Drama	5.5	66	80231
The Call	112 min	Horror, Mystery, Thriller	7.2	NA	6559
Greenland	119 min	Action, Drama, Thriller	6.3	NA	27482
Don't Listen	97 min	Drama, Horror, Thriller	6.1	NA	4879
Unhinged	90 min	Action, Thriller	6.0	40	25316
Ava	96 min	Action, Crime, Drama	5.3	39	20882

Visualising the data

We want to find out if movies with higher user rating get higher votes.

```
movies_df <- movies_df %>% mutate(
  user_rating = floor(user_rating)
)

movies_df %>% ggplot(
  aes(x = user_rating, y = votes, group = user_rating)
) +
  labs(
    title = "Relationship Between User Ratings AND Movie Votes",
    x = "User Rating"
  ) +
  geom_boxplot() +
  theme_bw()
```



If we consider the median number of votes for each of the user rating, we can see that the number of votes

tend to increase as user rating increases.

Conclusion

We wanted to demonstrate an end to end web scraping work flow in this project and we were able to achieve this by doing the following:

- inspecting the web page and getting familiar with the html structure of the page
- Parsing html using the rvest library and using functions from such as `stringr::str_replace()` and `readr::parse_number()` to get our required result.

While I used a for loop in this project to be able to deal with NA values, one draw back to this method is that when scraping large number of pages, it isn't the fastest solution. It is also worth keeping in mind web scraping ethics when scraping web pages.