

Webscraping Text

Emily Malcolm-White



```
#LOAD PACKAGES
library(tidyverse)
library(rvest)
```

Webscraping Text

Let's look at the [top 50 feature films in the first 7 months of 2023 listed on IMBD](https://www.imdb.com/search/title/?title_type=feature&year=2023-01-01,2023-07-01)

```
URL <- read_html("https://www.imdb.com/search/title/?title_type=feature&year=2023-01-01,2023-07-01")
```

Notice that the data for all these films isn't housed inside a `<table>` element!

Titles

For example, check out the first few lines of html code for Oppenheimer:

```
<h3 class="list-item-header">
  <span class="list-item-index unbold text-primary">1.</span>
  <a href="/title/tt15398776/?ref=adv_li_tt"
>Oppenheimer</a>
  <span class="list-item-year text-muted unbold">(2023)</span>
</h3>
```

In this case, we want to look for the class `list-item-header` AND then pull the text inside the `<a>` (link) tag.

```
html_elements(".list-item-header a")
```

Tip

In this case, we want ALL titles so we used `html_elements()`. If we had only wanted the first title we would have used `html_element()`

Scrape IMBD for the titles of the 50 most popular feature films in the first 7 months of 2023.

```
# title_data <- URL %>%
#   html_elements(".list-item-header a") %>%
#   html_text()
#
# title_data
```

Runtime

Scrape IMBD for the runtime of the 50 most popular feature films so far in 2023.

Check out the relevant HTML code for Oppenheimer:

```
<p class="text-muted ">
  <span class="certificate">R</span>
  <span class="ghost">|</span>
  <span class="runtime">180 min</span>
  <span class="ghost">|</span>
  <span class="genre">
Biography, Drama, History
  </span>
</p>
```

In this case, we need to reference the class `text-muted` AND the class `runtime`.

```
# URL %>%
#   html_nodes(".text-muted .runtime") %>%
#   html_text()
```

Alternatively, we could have called class `text-muted` AND the 3rd span, but it's easier and likely more accurate to ask for the class `runtime` in case `runtime` is missing for some reason.

Maybe we want to keep the `min` on the end, but it forces it into being a string rather than a number which makes it difficult to sort or filter.

```
library(readr)
# need this package for parse_number()
```



Figure 1: Artwork by @allisonhorst

```
# runtime_data <- URL %>%
#   html_nodes(".text-muted .runtime") %>%
#   html_text() %>%
#   parse_number() %>% #this picks out only the numbers (and drops characters, in this case)
#   as.numeric()
#
# runtime_data
```

Ratings

Scrape IMBD for the ratings of the 50 most popular feature films in the first 7 months of 2023.

Check out the relevant HTML code for Oppenheimer:

```
<div class="inline-block ratings-imdb-rating" name="ir" data-value="8.6">
  <span class="global-sprite rating-star imdb-rating"></span>
  <strong>8.6</strong>
</div>
```

Let's scrape it!

```
# rating_data <- URL %>%
#   html_elements(".ratings-imdb-rating strong") %>%
#   html_text() %>%
#   as.numeric()
#
# rating_data
```

Warning

Notice that there are only 49 ratings listed, not 50! There is no way to figure out which one is missing besides doing it by hand...

Which one is it?

Once we figure out which one is it is, we should add a blank element for the rating for that movie using the `append` function.

```
rating_data <- append(rating_data, values=FALSE, after=11)
```

It's Killers of the Flower Moon (#32)!

```
#rating_data <- append(rating_data, values=NA, after=31)
```

Notice how it is the correct length (50) now!

Number of Votes

Scrape IMBD for the number of votes of the 50 most popular feature films in the first 7 months of 2023.

Relevant code for Oppenheimer:

```

<p class="sort-num_votes-visible">
  <span class="text-muted">Votes:</span>
  <span name="nv" data-value="391689">391,689</span>
</p>

```

Let's scrape it!

```

# votes_data <- URL %>%
#   html_elements(".sort-num_votes-visible span:nth-child(2)") %>%
#   html_text() %>%
#   parse_number() %>%
#   as.numeric()
#
# votes_data

```

Warning

Same issue as before! We were supposed to have 50 but only got 49. It's Killers of the Flower Moon (#32), again!

```
#votes_data <- append(votes_data, values=NA, after=31)
```

Metascore

Scrape IMBD for the number of votes of the 50 most popular feature films in the first 7 months of 2023.

Relevant code for Oppenheimer:

```

      <div class="inline-block ratings-metascore">
<span class="metascore favorable">88      </span>
      Metascore
      </div>

```

Let's scrape it!

```

# metascore_data <- URL %>%
#   html_elements(".metascore") %>%
#   html_text() %>%
#   parse_number() %>%

```

```
# as.numeric()
#
# metascore_data
```

Warning

Yikes! Now we only have 41 when we should have 50.

We *could* manually go through and figure out which 9 are missing or we could reassess how important the metascore data is to us...

Combining it all together into a data frame!

We can combine all this data into one data frame:

```
# movies <- data.frame(Title = title_data,
# Runtime = runtime_data,
# Rating = rating_data,
# Votes = votes_data
# )
#
# movies
```

Make a list OR Make a plot!

```
# ggplot(movies, aes(x=runtime_data, y=rating_data)) +
#   geom_point() +
#   theme_minimal() +
#   xlab("Runtime (in minutes)") +
#   ylab("IMDB Rating")
```