Analyzing Movie Ratings

Salvatore Porcheddu

31 3 2021

Contents

Introduction

In this project we will scrape information from **IMDb**[1] about the 30 most popular movies released between March and July 2020, with the goal of comparing ratings from both the professional critics and ordinary people: do these ratings correlate to each other? In other words, **do movies which have been highly rated also tend to receive the highest number of votes from the users**?

Scraping the content

After loading the necessary packages (which was already done in a hidden code chunk), we will read the web page and prepare it for the subsequent scraping; please note that in order to avoid server instability problems, we will use a copy of the original page, stored in an external server. The URL will be stored in a variable, created in a hidden code chunk.

```
wp_content <- read_html(URL)

str(wp_content)
```

```
## List of 2
##  $ node:<externalptr>
##  $ doc :<externalptr>
##  - attr(*, "class")= chr [1:2] "xml_document" "xml_node"
```

[1] **IMDb**, which stands for **Internet Movie Database**, is a website containing information related to films, television programs, home videos, video games and so on. Read more about it on its Wikipedia page.

The output is a list of two elements, which are respectively the head and the body of the page.

The first thing that we will extract are the movies' titles and release years; the extraction will be performed using CSS selectors and the `rvest` R package.

```
title_selector <- ".lister-item-header a"

title <- wp_content %>%
  html_elements(title_selector) %>%
  html_text

title
```

```
##  [1] "Mulan"
##  [2] "The Call"
##  [3] "Greenland"
##  [4] "Don't Listen"
##  [5] "Unhinged"
##  [6] "Ava"
##  [7] "The Hunt"
##  [8] "Ghosts of War"
##  [9] "Hamilton"
## [10] "The Old Guard"
## [11] "The Secret: Dare to Dream"
## [12] "The Outpost"
## [13] "Extraction"
## [14] "Train to Busan Presents: Peninsula"
## [15] "Greyhound"
## [16] "The King of Staten Island"
## [17] "A Quiet Place Part II"
## [18] "Bloodshot"
## [19] "The Dark and the Wicked"
## [20] "Arkansas"
## [21] "The Rental"
## [22] "Trolls World Tour"
## [23] "Sputnik"
## [24] "Eurovision Song Contest: The Story of Fire Saga"
## [25] "Inheritance"
## [26] "Spenser Confidential"
## [27] "The Tax Collector"
## [28] "The Way Back"
## [29] "The Silencing"
## [30] "Archive"
```

```
year_selector <- ".lister-item-year"

year <- wp_content %>%
  html_elements(year_selector) %>%
  html_text()

year
```

```
##  [1] "(2020)"      "(2020)"      "(2020)"      "(2020)"      "(2020)"
##  [6] "(IV) (2020)" "(II) (2020)" "(2020)"      "(2020)"      "(2020)"
```

```
## [11] "(2020)"      "(2020)"      "(2020)"      "(2020)"      "(2020)"
## [16] "(2020)"      "(2020)"      "(2020)"      "(2020)"      "(2020)"
## [21] "(2020)"      "(2020)"      "(2020)"      "(2020)"      "(I) (2020)"
## [26] "(2020)"      "(2020)"      "(2020)"      "(2020)"      "(2020)"
```

The `year` vector has a few problems that need to be addressed:

```
# 1) getting rid of the parentheses and of other non-number characters
year <- year %>%
  stringr::str_replace_all("[()]+", "") %>%
  stringr::str_replace_all("[IV]", "") %>%
  stringr::str_trim()

# 2) converting the years from strings to numeric
year <- as.numeric(year)

year
```

```
##  [1] 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020
## [16] 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020
```

Next, we will extract the movies' runtimes and genres, following a similar process to what we have done for titles and years:

```
# Let's start with the runtimes:

runtime_selector <- ".runtime"

runtime <- wp_content %>%
  html_elements(runtime_selector) %>%
  html_text()

runtime
```

```
##  [1] "115 min" "112 min" "119 min" "97 min"  "90 min"  "96 min"  "90 min"
##  [8] "94 min"  "160 min" "125 min" "107 min" "123 min" "116 min" "116 min"
## [15] "91 min"  "136 min" "97 min"  "109 min" "95 min"  "117 min" "88 min"
## [22] "90 min"  "113 min" "123 min" "111 min" "111 min" "95 min"  "108 min"
## [29] "93 min"  "109 min"
```

The runtimes are expressed as strings with a number and the "min" abbreviation to indicate that they are represented as minutes. We will now parse these strings as numeric:

```
runtime <- runtime %>%
  readr::parse_number()

runtime
```

```
##  [1] 115 112 119  97  90  96  90  94 160 125 107 123 116 116  91 136  97 109  95
## [20] 117  88  90 113 123 111 111  95 108  93 109
```

```r
# Now let's deal with the genres

genre_selector <- ".genre"

genre <- wp_content %>%
  html_elements(genre_selector) %>%
  html_text() %>%
  stringr::str_trim() # This is to remove leading or trailing whitespaces from
                      # the result

genre
```

```
##  [1] "Action, Adventure, Drama"     "Horror, Mystery, Thriller"
##  [3] "Action, Drama, Thriller"      "Drama, Horror, Thriller"
##  [5] "Action, Thriller"             "Action, Crime, Drama"
##  [7] "Action, Horror, Thriller"     "Horror, Thriller, War"
##  [9] "Biography, Drama, History"    "Action, Adventure, Fantasy"
## [11] "Drama, Romance"               "Action, Drama, History"
## [13] "Action, Thriller"             "Action, Horror, Thriller"
## [15] "Action, Drama, History"       "Comedy, Drama"
## [17] "Drama, Horror, Sci-Fi"        "Action, Drama, Sci-Fi"
## [19] "Horror"                       "Crime, Drama, Thriller"
## [21] "Horror, Thriller"             "Animation, Adventure, Comedy"
## [23] "Drama, Horror, Sci-Fi"        "Comedy, Music"
## [25] "Drama, Mystery, Thriller"     "Action, Comedy, Crime"
## [27] "Action, Crime, Drama"         "Drama, Sport"
## [29] "Action, Crime, Thriller"      "Drama, Sci-Fi, Thriller"
```

Now let's go further and extract the rating for each movie and the respective metascore.

```r
rating_selector <- ".ratings-imdb-rating strong"

rating <- wp_content %>%
  html_elements(rating_selector) %>%
  html_text() %>%
  as.numeric() # Converting the result to a number

rating
```

```
##  [1] 5.5 7.2 6.3 6.1 6.0 5.3 6.5 5.5 8.6 6.6 6.4 6.8 6.7 5.4 7.0 7.1 5.7 6.2 5.9
## [20] 5.7 6.1 6.4 6.5 5.5 6.2 4.7 6.7 6.1 6.3
```

```r
metascore_selector <- ".metascore"

metascore <- wp_content %>%
  html_elements(metascore_selector) %>%
  html_text() %>%
  stringr::str_trim() %>%
  as.numeric()

metascore
```

```
##  [1] 66 40 39 50 38 90 70 32 71 56 51 64 67 44 72 55 63 51 61 50 31 49 22 66 67
```

Finally, let's extract the number of votes:

```
votes_selector <- ".sort-num_votes-visible > span:nth-child(2)"

votes <- wp_content %>%
  html_elements(votes_selector) %>%
  html_text() %>%
  readr::parse_number() # The strings are numbers written with a thousands
                        # separator, that we can remove by parsing them as
                        # numbers

votes
```

```
##  [1]  80231   6559  27482   4879  25316  20882  62277   4886  48663 117211
## [11]   5420  17001 151319  16815  60885  29278  57763   3701   7951  13965
## [21]  14792  10516  69186   7837  65768   7416  28300   9355   9271
```

Working with the data

Unfortunately, our `user_rating`, `metascore` and `votes` vectors don't share the same length as the other vectors that we have extracted before: this is because some movies are not rated.

We will fix this problem by doing the following:

- we will add NA values into the `metascore` vector for the movies with indices 2, 3, 4, 17, and 29;
- we will remove the movie with the index 17 from all the vectors with length of 30 (our new `metascore` vector included), because this movie lacks rating, metascore and number of votes.

For the first step we are going to use a custom function that allows us to insert values into a vector at the specified positions:

```
append_vector <- function(vector, inserted_indices, values){

  ## Creating the current indices of the vector
  vector_current_indices <- 1:length(vector)

  ## Adding small amount of values (between 0 and 0.9) to the `inserted_indices`
  new_inserted_indices <- inserted_indices +
    seq(0, 0.9, length.out = length(inserted_indices))

  ## Appending the `new_inserted_indices` to the current vector indices
  indices <- c(vector_current_indices, new_inserted_indices)

  ## Ordering the indices
  ordered_indices <- order(indices)

  ## Appending the new value to the existing vector
  new_vector <- c(vector, values)

  ## Ordering the new vector wrt the ordered indices
  new_vector[ordered_indices]
}
```

```r
metascore <- append_vector(metascore, c(1, 1, 1, 13, 24), NA)

metascore
```

```
##  [1] 66 NA NA NA 40 39 50 38 90 70 32 71 56 51 64 67 NA 44 72 55 63 51 61 50 31
## [26] 49 22 66 NA 67
```

Note that the positions chosen for the positions vector in the `append_vector` function do not correspond to the final indices, as they are determined based on the initial `metascore` vector, which had length less than 30.

Now we can proceed and remove the seventeenth element from each vector that has length 30 (i.e. the `title`, `year`, `runtime`, `genre` and `metascore` vectors):

```r
title <- title[-17]
year <- year[-17]
runtime <- runtime[-17]
genre <- genre[-17]
metascore <- metascore[-17]

# The new `title` vector, for example, is this
title
```

```
##  [1] "Mulan"
##  [2] "The Call"
##  [3] "Greenland"
##  [4] "Don't Listen"
##  [5] "Unhinged"
##  [6] "Ava"
##  [7] "The Hunt"
##  [8] "Ghosts of War"
##  [9] "Hamilton"
## [10] "The Old Guard"
## [11] "The Secret: Dare to Dream"
## [12] "The Outpost"
## [13] "Extraction"
## [14] "Train to Busan Presents: Peninsula"
## [15] "Greyhound"
## [16] "The King of Staten Island"
## [17] "Bloodshot"
## [18] "The Dark and the Wicked"
## [19] "Arkansas"
## [20] "The Rental"
## [21] "Trolls World Tour"
## [22] "Sputnik"
## [23] "Eurovision Song Contest: The Story of Fire Saga"
## [24] "Inheritance"
## [25] "Spenser Confidential"
## [26] "The Tax Collector"
## [27] "The Way Back"
## [28] "The Silencing"
## [29] "Archive"
```

Now every vector has a length of 29, although the `metascore` vector still contains 4 NA values.

Building a dataframe and visualizing it

It is now possible to combine the vectors into a dataframe without any errors or other issues:

```
imdb_df <- tibble::tibble(title, year, runtime, genre, rating, metascore, votes)

# Here are the first six rows of the new dataframe
head(imdb_df)
```

```
## # A tibble: 6 x 7
##   title         year runtime genre                      rating metascore votes
##   <chr>        <dbl>   <dbl> <chr>                       <dbl>     <dbl> <dbl>
## 1 Mulan         2020     115 Action, Adventure, Drama      5.5        66 80231
## 2 The Call      2020     112 Horror, Mystery, Thriller     7.2        NA  6559
## 3 Greenland     2020     119 Action, Drama, Thriller       6.3        NA 27482
## 4 Don't Listen  2020      97 Drama, Horror, Thriller       6.1        NA  4879
## 5 Unhinged      2020      90 Action, Thriller              6          40 25316
## 6 Ava           2020      96 Action, Crime, Drama          5.3        39 20882
```

To improve the clarity of the visualizations that we will create in the following paragraph, let's remove the decimals in the `user_rating` column by flooring the numbers:

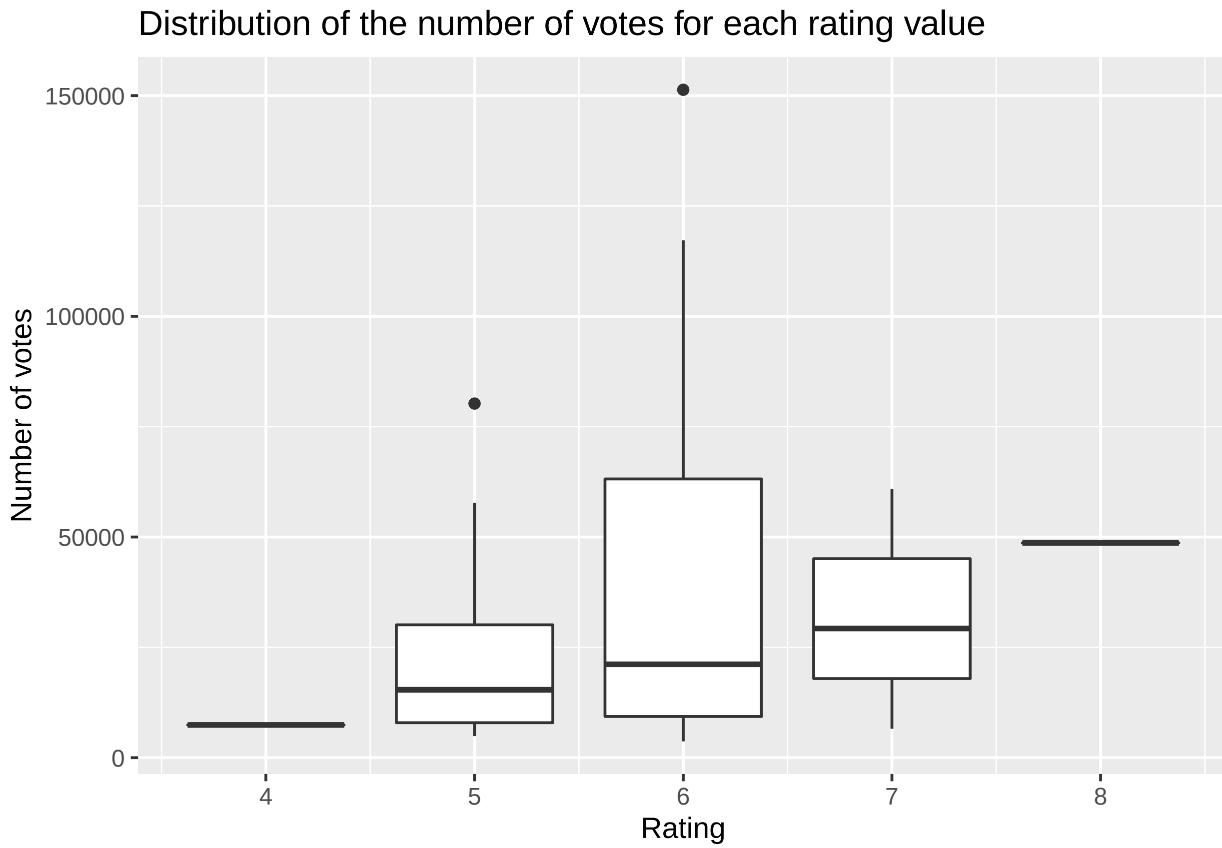
```
imdb_df <- imdb_df %>%
  mutate(rating = floor(rating))

imdb_df$rating
```

```
##  [1] 5 7 6 6 6 5 6 5 8 6 6 6 6 5 7 7 5 6 5 5 6 6 6 5 6 4 6 6 6
```

We can now plot our data in order to answer our initial question: *do movies which have been highly rated also tend to receive the highest number of votes from the users*?

The answer can be easily seen with a **boxplot** with the `user_rating` on the x-axis and the `votes` on the y-axis:

```
ggplot(imdb_df, aes(x=rating, y=votes, group=rating)) +
  geom_boxplot() +
  labs(title = "Distribution of the number of votes for each rating value",
       x = "Rating", y = "Number of votes")
```



We can see that indeed as the rating increases, the median number of votes also increases, although the maximum number of votes was received by a movie with a rating of 6 (*Extraction*).

Conclusion

In this project, we have scraped data about some popular movies from 2020, combined them into a dataframe and visualized them. Our analysis has allowed us to find out that **highly-rated movies also tend to receive the highest number of votes from the people**.