

Name: Shanmukha Srivathsav Satujoda

NetID: ss3203

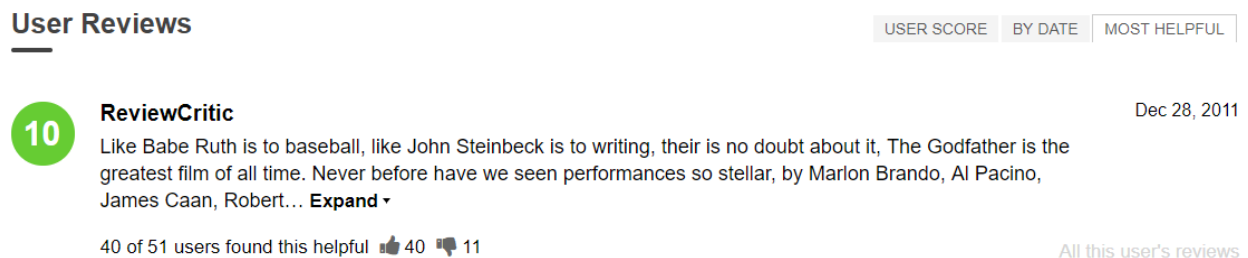
SENTIMENT ANALYSIS ON THE REVIEWS OF “THE GODFATHER” MOVIE

Introduction:

The theme of this project is to come up with a sentiment analysis on the user reviews of the very famous “The Godfather” movie. The reviews are collected from the famous Metacritic Reviews website. The underlying objective however is to showcase scraping techniques using CSS selectors, preprocessing the text (the reviews) using regular expression, coming up with some visualizations to see some interesting trends, performing a sentiment analysis and do a comparison between the sentiment score on a particular review, using a standard libraries and calculating the same manually. In the conclusion section of this report is a list of concepts used to accomplish the above-mentioned task, which are taught in the class.

Scraping:

The following image is a typical example of how a review looks on the Metacritic web page.



The screenshot shows a Metacritic user review interface. At the top, there's a header 'User Reviews' with three tabs: 'USER SCORE', 'BY DATE', and 'MOST HELPFUL'. Below this, a review by 'ReviewCritic' is displayed. The review has a green circular badge with the number '10' and a date of 'Dec 28, 2011'. The review text reads: 'Like Babe Ruth is to baseball, like John Steinbeck is to writing, their is no doubt about it, The Godfather is the greatest film of all time. Never before have we seen performances so stellar, by Marlon Brando, Al Pacino, James Caan, Robert...'. There is an 'Expand' button next to the text. At the bottom of the review, it says '40 of 51 users found this helpful' with a thumbs up icon and the number '40', and a speech bubble icon with the number '11'. A link 'All this user's reviews' is visible at the bottom right of the review section.

Our objective here is to scrape three entities which are used for the analysis. The first one is the rating of the movie represented by a number, the second is the review (which is the text) and the third parameter is the date on which the review has been posted. The reviews are spread across 4 different webpages from the same domain. So, I have created a function `collect_data_metacritic()`, which takes the url as input and returns a data frame as an output with three columns, “Rating”, “Review” and “reviewDate”. In this function we read the url using `read_html(url)`. Once the data from the web page has been parsed, I have used the CSS selector for the required elements and used them as html nodes to gather the data. For every entity mentioned (Rating, Review, reviewDate), the function internally creates an arbitrary data frame with an arbitrary ID which is then used to join the three columns into a

single data frame. The same process is applied for all the other pages which have the reviews. At the end we will have four different data frames with our data (one corresponding to page1, page2, page3 and page4). These four data frames are then merged using a row-wise bind to get the final data frame containing 387 reviews.

The major challenge is to identify the proper CSS tag required to select relevant data. For this I had to review some basics of CSS. Another painfully time-consuming challenge which I encountered during this process was when the scraped data was incomplete. For example, in the image presented above, we can see that the review is not complete. To get the complete review we must click on the “Expand” option which would then give the complete review. The problem might seem minor however figuring what caused the problem was the tricky part.

Preprocessing:

Now that the raw data is in a data frame, we now must preprocess it to go with further analysis. The following image illustrates how a typical review looks like in raw format.

```
> temp_data$review[1]
[1] \n                                     \n          usually i like slow
paced films, but at some point a movie should draw interest or emotion. There are almost no story peaks that are exciting, s
uspensful, emotional or whatever. when people die for example, they get killed suddenly out of nowhere, so even before these moments t
s, but at some point a movie should draw interest or emotion. There are almost no story peaks that are exciting, suspenseful,
emotional or whatever. when people die for example, they get killed suddenly out of nowhere, so even before these moments t
he movie builds up no suspense. Many scenes are far too long and forgettable, like both the weddings and many more. Even the
horsehead had no relevance for anything. The acting is solid but never reaches any remarkable levels. Not one of the charac
ters is truly likeable, what makes it difficult to feel for them. In the end i am also a bit disappointed of the camera work
: too many wideshots, too less facial close ups and missing overall aesthetics in look and scenery. I feel bad for rating a
classic movie like this so low, but i cant see the brilliancy of The Godfather.... Expand\n
200 . 1
```

As one can see there are a lot of redundant spaces and special characters in the data. I have created a function preprocessing() which takes a data frame as input and returns a data frame with cleaned columns as output. Inside the function is a for loop which iterates over all the rows of the Review column applying regular expression to clean the data.

Also, the date is a string rather than an actual date. We need to extract the year from the dates which are used in the later stages to do some analysis. For this, I have used as.Date () from R with the format %B-%d-%Y which represents month-day-year format. After that extracting the year becomes relatively easy

The following image shows the regular expression used and its explanation in detail:

```
```{r}
#This function is used for preprocessing the data:
preprocessing<-function(df){
 #direct conversion to numeric data is replacing the rating with level rather than the actual rating
 df$Rating <- as.numeric(as.character(df$Rating))
 df$Review <- (as.character(df$Review))

 #Removing the special characters, leading, trailing white spaces
 for(x in 1:nrow(df)){
 df$Review[x] <- trimws(gsub("\r?\n|\r", " ", temp_data$Review[x]))
 df$Review[x] <- str_replace_all(df$Review[x], "[^[:alnum:]]", " ")
 #Merging multiple white spaces to one white space
 #NODE EXPLANATION
 #-----
 #(?<= look behind to see if there is:
 # -----
 #[\s] any character of: whitespace (\n, \r, \t, \f, and " ")
 # -----
 #) end of look-behind
 # -----
 #\s* whitespace (\n, \r, \t, \f, and " ") (0 or
 # more times (matching the most amount
 # possible))
 # -----
 # | OR
 # -----
 # ^ the beginning of the string
 # -----
 #\s+ whitespace (\n, \r, \t, \f, and " ") (1 or
 # more times (matching the most amount
 # possible))|
 # -----
 # $ before an optional \n, and the end of the
 # string
 df$Review[x] <- gsub("(?<=[\s])\\s*|^\\s+|\\s+$", "", df$Review[x], perl=TRUE)
 }
 return(df)
}
```

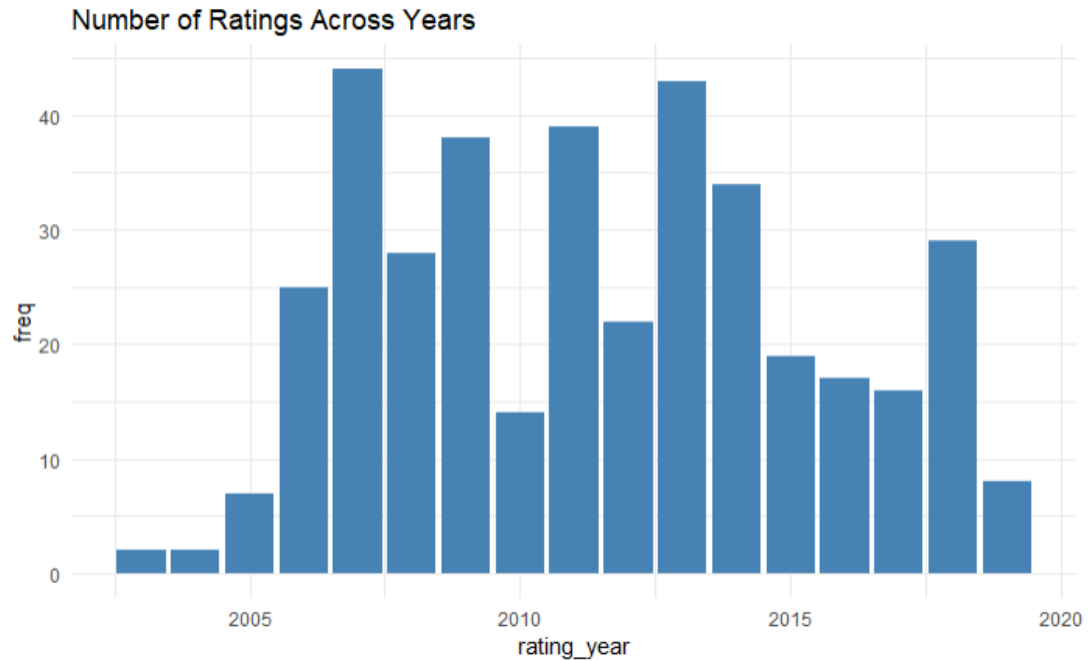
The major challenge which I encountered in this section is coming up with the regular expressions to clean the data. This is achieved after spending a lot of time experimenting with regex and searching online. One very interesting thing which I realized during this section is how the rating variable is considered as a factor variable rather than a text or an integer. That means if I have just used `as.numeric(Rating)`, that would not translate a rating of '3' to a numeric 3 rather it replaces it by its level.

### Visualizations:

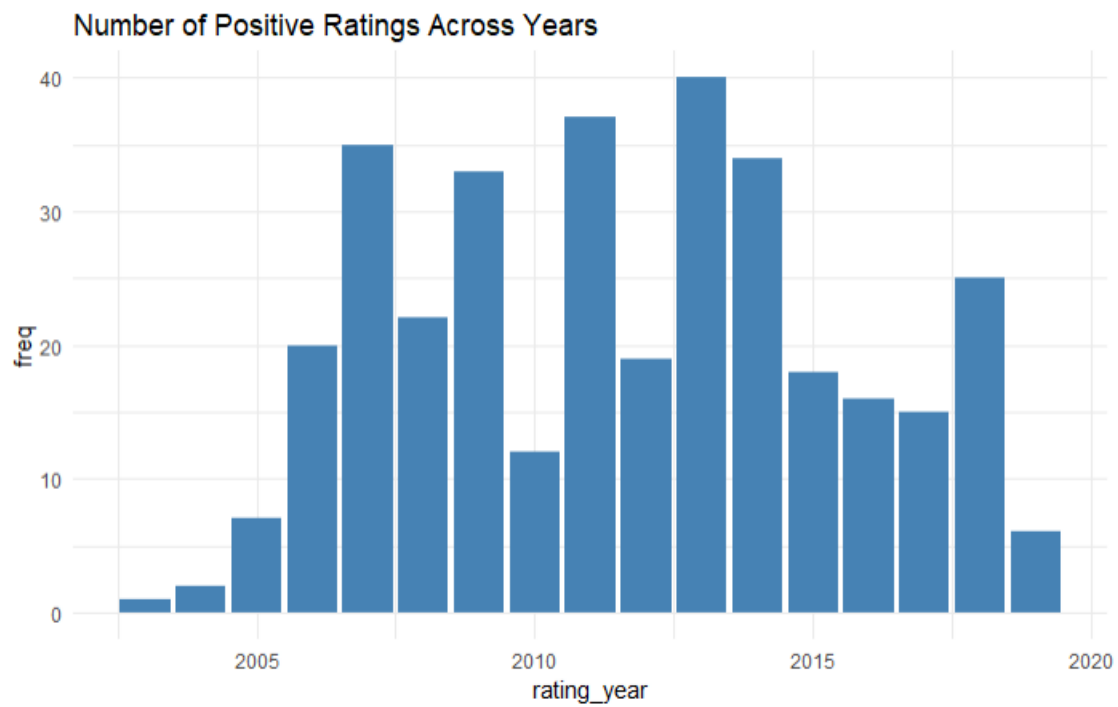
Now that our data is completely ready, we can look at a bunch of visualizations:

The first visualization is looking at how the reviews are distributed across all the years. To achieve this first we have grouped the reviews by year. After grouping by year, we have summarized them according to the frequency which gives us the number of reviews per year. Using ggplot I have created a bar plot which would visualize this data.

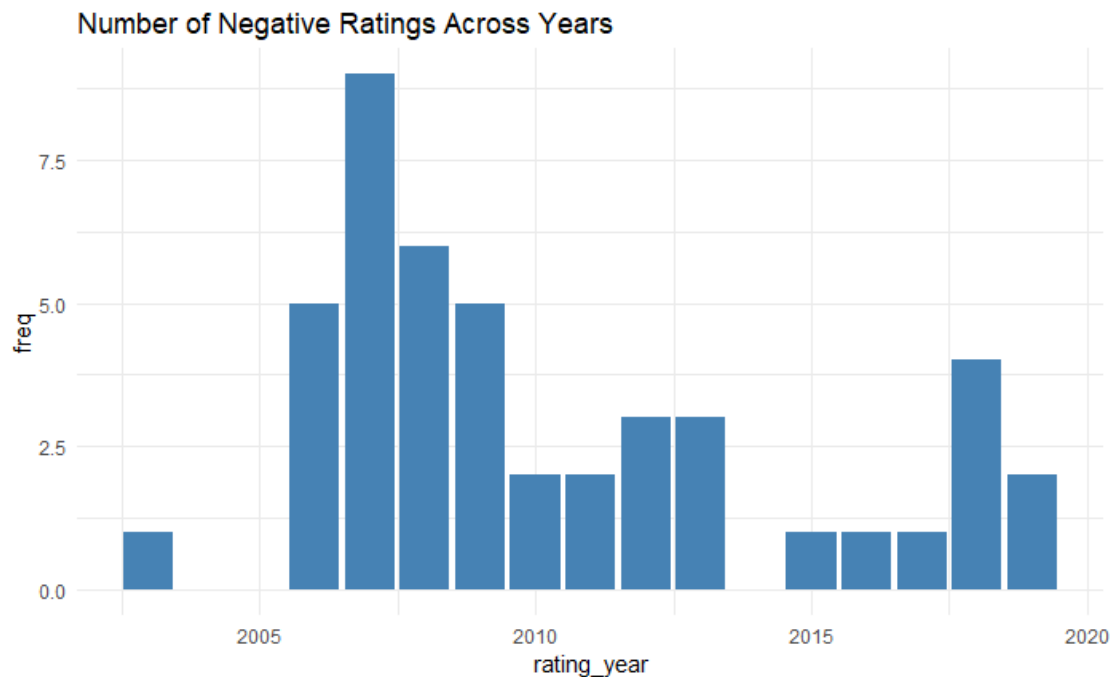
The following graph represents how the data is distributed:



In a similar fashion we can obtain the frequencies of both the positive rating and negative ratings. For positive ratings, I have used a threshold of 7.5/10 and anything below 7.5 is considered as negative rating. The pictorial representations of both the positive and negative ratings is given below.



Negative rating across years:



### Sentiment Analysis:

For this section I have done two things. Calculate the sentiment score using the 'sentimentr' package and calculate the sentiment score using `get_sentiments()` for any particular review. However, before proceeding we have to make sure that all the stop words have been removed. For this we have to `unnest()` all the words in a review and use `anti_join` on the `stop_words` tibble inbuilt in R. Then we can use the `count()` function to see the frequency of the words.

A similar approach can be applied to get the top bigrams of the entire review section using the `bigram` parameter in `unnest_tokens()` function. After obtaining the frequency of both monograms and bigrams, a simple visualization of the most frequently used words can be done using a word cloud. The top ten words, bigrams and the word cloud of hundred words look as represented below.

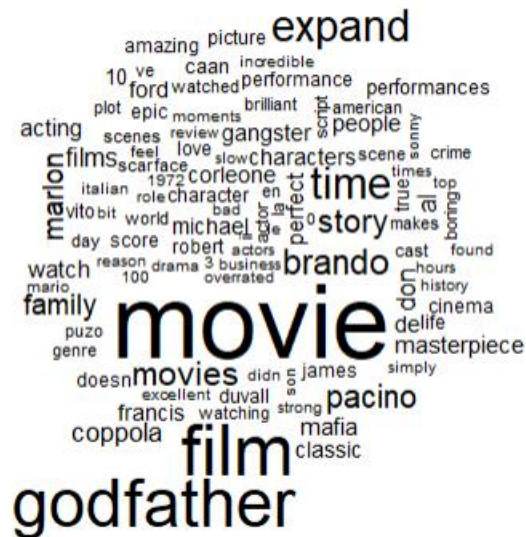
Top 10 words:

word	n
movie	597
film	389
godfather	361
time	206
expand	203
brando	151
movies	139
pacino	132
story	125
family	107

Top 10 bigrams:

word1 < chr>	word2 < chr>	n < int>
marlon	brando	102
al	pacino	92
francis	ford	63
ford	coppola	62
james	caan	39
robert	duvall	35
don	vito	31
brando	al	24
mario	puzo	22
vito	corleone	22

Word Cloud for the top 100 words:

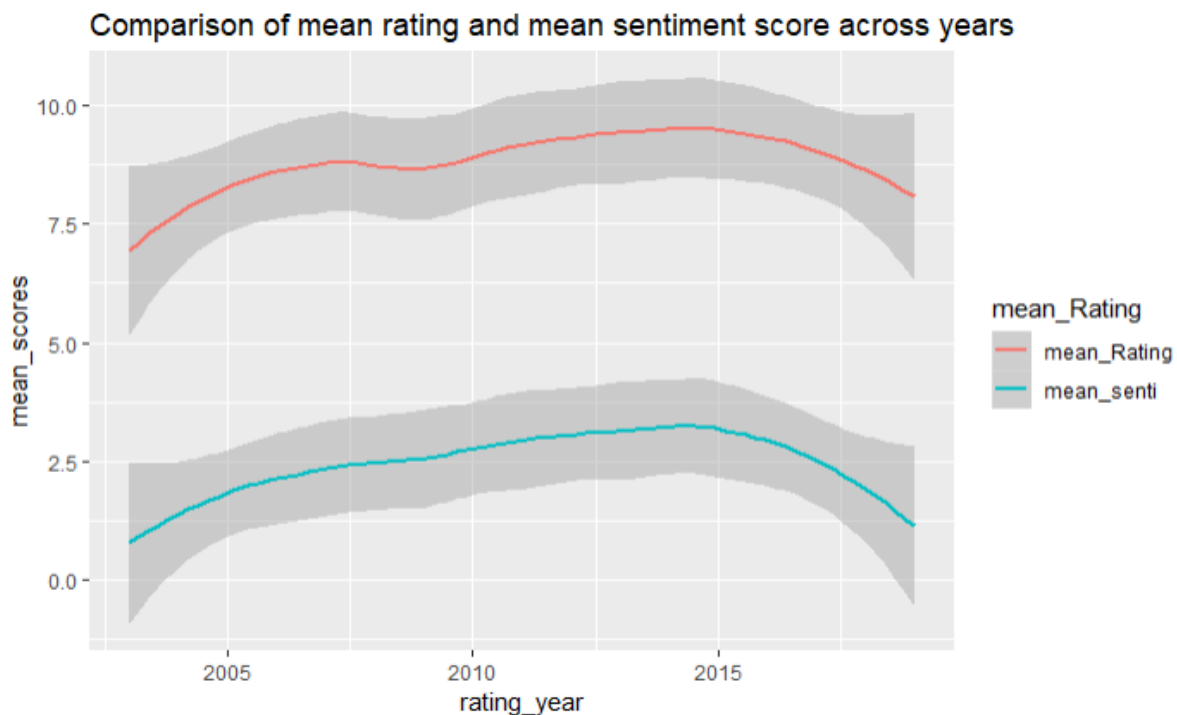


### Getting Sentiment score for any particular review:

Now that the basic text processing has finished, we can move on to the sentiment analysis. For any particular review, the first step is to split the entire string and converting into a word vector. Then use `antijoin()` this word vector on to the standard stop word vector to weed out the stop words. Using the `get_sentiments` we can get a sentiment score for all the words in a particular string. By summing over all the sentiment scores of the word vector we get the sentiment score of the entire review. The whole process can be packaged in the form of a function and can be applied to all the reviews. However, this is long and tedious. The next approach is a more elegant way of doing the same.

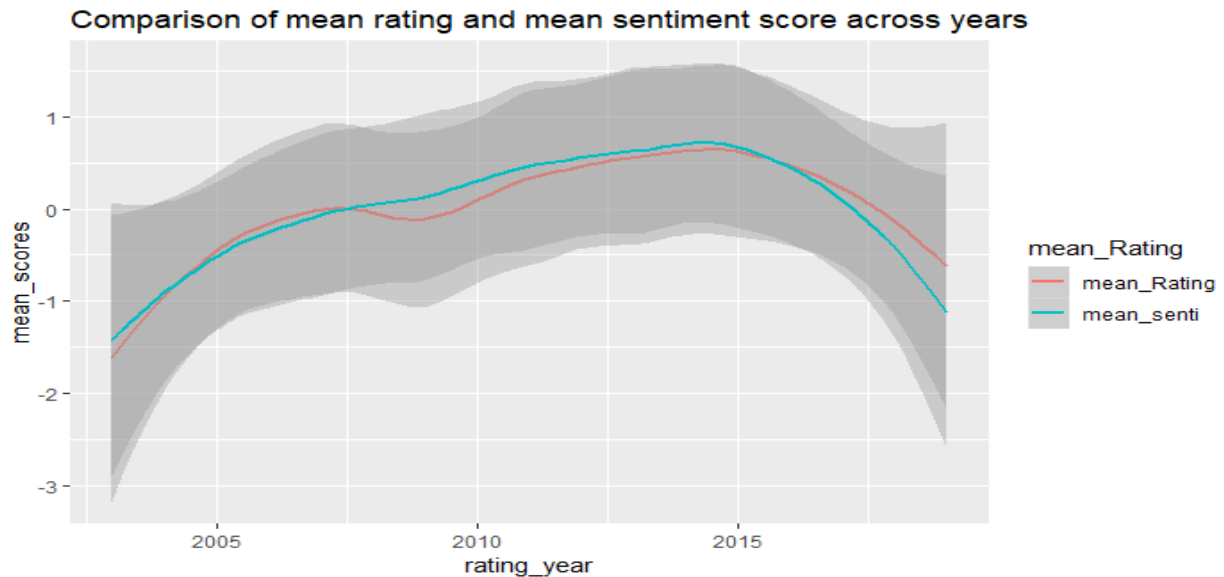
## Getting sentiment score using inbuilt packages in R:

Using the `sentiment()` function we can get the sentiment score of a sentence directly. I have made use of this feature to calculate the sentiment score for all the reviews by applying `sentiment(Metacritic_Data$Review)`. Hypothetically speaking the sentiment score must approximately match the user rating. We can validate this visually by plotting the average user rating across all years and average sentiment score across all years on the same plot. The following graph is obtained by doing so:



The green line represents the average rating and the red line represents the average sentiment score across years. We can see that there is a clear disparity. However, that disparity has occurred because both the ratings are not normalized i.e., the scale of both ratings are different. To overcome this all we have to do is just normalize both the columns average rating and average sentiment score and then plot the same.

The following visualization represents the comparison after normalizing:



We can now see a clear correspondence in the trends between the average sentiment score (red line) and average rating (green line).

**Comparison between rating, sentiment score calculated manually, and sentiment score calculated using sentimentr package:**

```
##{r}
#Final comparison between the mean rating and mean sentiment score:
#The final sentiment score of the entire review
print("The final sentiment score of the entire review calculated manually is: ")
final_sentiment_score
print("The final sentiment score of the entire review calculated using sentimentr is: ")
metacritic_data$SentimentScore[1]
print("The user rating is: ")
metacritic_data$Rating[1]
##
```

```
[1] "The final sentiment score of the entire review calculated manually is: "
[1] -4
[1] "The final sentiment score of the entire review calculated using sentimentr is: "
[1] -3.097187
[1] "The user rating is: "
[1] 3
```

## Conclusions:

The major takeaway from this project is how we can scrape the data and preprocess the data for any kind of analysis. In the process of doing so, I have tried to use as many concepts as possible, taught during the class. Some of the concepts that are used are pipes, creating functions for reusability, scraping websites, using regular expressions for cleaning, joins, bindings, grouping by, summarizing, aggregating, tidyverse-unnesting, tokenizing, using mutates to create new columns dynamically, using gather to change data from wide to long format, ggplots for visualizations, creating word clouds from monograms, obtaining bigrams, getting sentiment scores and performing sentiment analysis.



**Github Link:**

<https://github.com/SrivathsavSatujoda/SentimentAnalysisOnMovieReviews.git>