Statisitical Consulting Final Write Up

Xunqing Zheng & Duja Michael

Extracting and Analyzing the New York Times Movie Reviews Data

```
library(tidyverse)
library(lubridate)
library(stringr)
library(zoo)
library(readr)
library(tidyverse)
library(splitstackshape)
library(stringi)
library(tm)
library(ggplot2)
library(jsonlite)
library(cronR)
library(ggridges)
#library(taskscheduleR)
```

Introduction

As the movie landscape continues to evolve, The New York Times movie reviews continue to offer insightful commentary, providing audiences with a nuanced perspective on the ever-changing and dynamic world of cinema. In order to get a deeper understanding of how the NYTimes is selecting what movies to review, what type of movie each of the staff critics review, and if there are patterns selecting which movies to be reviewed by whom, we decided to analyze all the movie reviews published in NYTimes from Januray 2018 until November 2023. In particular, we are trying to understand:
- Is there a pattern or a time-trend in the number of movies that the NYTimes reviews during this period?
- Is any particular genre more likely to be reviewed by NYTimes?
- Which critics are the most prolific?
- Do particular reviewers/authors specialize in particular genres?
- What's the distribution of IMDb ratings of the movies reviewed each author?
- Are "Critic's Picks" more likely to also receive higher ratings by the public?

Inital Exploration

They journey to land on this topic was a windy one. Initially, we wanted to work with a social media API where we would grab all the comments on a specific movie and run sentiment analysis to get an idea about how the movie is received by the public. We also had a detour in our thinking in mid-October and wanted to shift our project to check the state of democracy in the USA by downloading the comments on all posts by the POTUS and Bernie Sanders accounts to see the prevalence of the calls for a ceasefire in Gaza knowing

that those comments went on deaf ears. All those ideas proved to be unfit for the scope of this project due to the difficulty in extracting comments on a certain post in the APIs of the social media platforms we wanted to use. All the available services to do that were paid ones.

The final pivot we did was to go back to our initial idea of movie reviews and use the NYTimes' API. While working with the NYTimes API, we also attempted to pull Covid related articles but decided not to use that data in our final analysis.

Difficulties with the NYTimes API

The NYTimes API presents a steep learning curve on how it operates. A good understanding of the documentation is required to navigate and extract the desired information efficiently. Throughout the process, we have encountered many difficulties, such as function deprecation, code errors, data consistency, etc.

API Limitations:

The NYTimes API has a daily rate limit of 500 hits per API key, each hit only gets 10 articles, and it also limits users from hitting the API too frequently. An code 400 error message will appear once we reached 200 consecutive request to access the API, sometimes even earlier. These limitations impacted the real time data retrieval and require careful consideration of the API usage patterns. On top of the rate limits, NYTimes also deprecated its movie reviews API, which made our data collection process even more time consuming. We used its article search API as a workaround. The issue with the article search API is that in order to get all the movie reviews, we can't limit the articles to only movie reviews. A lot of other articles were included in our pull and only 28% of the article were actually movie reviews.

Code Errors:

Working with API is never easy, in addition to the API limitation, we also experienced some errors in our codes, which further complicates our data collection process. Although NYTimes deprecated its Movie Review API, it offered a sample url structure to access the movie review from the Article Search API.
- https://api.nytimes.com/svc/search/v2/articlesearch.json?fq=section_name%3A%22Movies" AND type_of_material%3A"Review"&sort=newest&page=0&api-key{your-key}
The first issue we encountered with this URL is with colon. NYTimes used the HTML encoding "%3A" instead of an actual colon in the provided URL. This encoding did not work for us and we kept accessing empty pages. The second issue with this URL is NYTimes added a "page=0" parameter, which limited the access to only the first 10 articles. We did not notice this in our initial API pull, as an result, our initial data only contained 10 unique articles in each data file.

Quality, consistency, and insufficient information:

Another issue we faced with the NYTimes API was the lack of information and consistency in the data provided. For instance, data on the movie name and movie type are provided in a list of lists. Not all reviews had these two pieces of information, and in the cases where the information is provided, it is not arranged in a way that lends itself to systematic extraction. For movie names, we had to supplement the list data with information from the URL in cases where the list was missing that information. For the type of movie, the only categorizations that were consistently provided were whether the movie is a film, a documentary, or an animated film. No consistent information is provided for the genre of the movie.

From the NYTimes API data, we ended up extracting and using movie name, movie type, whether or not it was a critic's pick, the author of the review, and the date of publishing the review.

Data Collection

NYTimes

In order to access the NYTimes API, we first created a developer account with NYTimes to access a API key. The code below sets a stage for all the necessary parameters. The dates are coded automatically referring to past 7 days based on system time for any ongoing, continuous data pull. We hard coded the dates for any massive hitorical data pull.

```
NYTIMES_KEYm <- "hCt4EAjrvjsBswpEqfukn2lLVCkpCgCG" ###need individual access
movie = "Movies" ## section_name
review = "Review" ## type_of_content
begin_date1 = as.POSIXlt(Sys.time()-86400*7)
begin_date2 = paste0(substr(begin_date1,0,4),substr(begin_date1,6,7),substr(begin_date1,9,10))
end_date1 = as.POSIXlt(Sys.time()-86400)
end_date2 = paste0(substr(end_date1,0,4),substr(end_date1,6,7),substr(end_date1,9,10))
```

After coding all the parameters, we used paste0() to combine all of them to form a URL to access the API data.

```
baseurl1 <- paste0("http://api.nytimes.com/svc/search/v2/articlesearch.json?fq=section_name:",movie,
                   "AND type_of_material:",review, "&sort=newest",
                   "&begin_date=",begin_date2,"&end_date=",end_date2,
                   "&facet_filter=true&api-key=",NYTIMES_KEYm)
baseurl1
```

With this base url, we were able to access the data from the NYTimes API, divided the total article number by 10 to get the maximum pages we need to pull within the time frame we set. Then, we created an empty list vector to store all the data from the API. Since each time we can only access 1 page or 10 articles, a for loop is used to hit the API and store data over pages. The for loop was the most time consuming part of the code. After multiple tests to minimized potential error, we had to change the wait time between each API hit from our initial 5 seconds to 12 second.

```
initialQuery <- fromJSON(baseurl1)
maxPages <- round((initialQuery$response$meta$hits[1] / 10)-1)

pages_2023 <- vector("list",length=maxPages)

for(i in 0:maxPages){
  nytSearch <- fromJSON(paste0(baseurl1, "&page=", i), flatten = TRUE) %>% data.frame()
  pages_2023[[i+1]] <- nytSearch
  Sys.sleep(12)
}
```

Lastly, we store the data as .Rdata file for future analysis.

```
MovieFile <- rbind_pages(pages_2023)
save(MovieFile,file=paste0(end_date2,"Movie.Rdata"))
```

Throughout the process, we learned that accessing recent data is a lot easier than historical data due to volumes. For historical data, we had to carefully check the date range to make sure each time we ran the for loop, it's within 200 pages to avoid potential errors.

IMDb

Besides its paid API, IMDb has free datasets that are available for use for non-commercial purposes. We utilize this data to supplement the NYTimes information. In particular, this data adds information on the genre of the movie, the IMDb average rating, and the number of ratings. The code below basically creates a function to go to the IMDb file download page, get the TSV files, then store the dataset into R Global Enviroment for analysis.

```
imdbTSVfiles <- function(fileName){
  url <- paste0("https://datasets.imdbws.com/",fileName,".tsv.gz")
  tmp <- tempfile()
  download.file(url, tmp)

  assign(fileName,
         readr::read_tsv(
           file = gzfile(tmp),
           col_names = TRUE,
           quote = "",
           na = "\\N"),
         envir = .GlobalEnv)
}

imdbTSVfiles("title.basics")
imdbTSVfiles("title.ratings")
```

Automation of the data pipeline

In an effort to access all the data in a more sustainable way for long term purpose, we also tried to automate the data pull with cronR for Mac and taskscheduleR for Windows. Unfortunately, both attempts were not successful for NYTimes due to issues with our laptops' security systems. We did successfully automate the IMDb data pull as we save the data to R directly instead of as a separate file to our local disk.

```
# This code chunk is for display only. We don't want to actually schedule any task.

#mac
cmd1 = cron_rscript("~/StatConsultation/MovieAPI.R")
cron_add(cmd1, "daily", at="6AM")

cmd2 = cron_rscript("~/StatConsultation/imbddata.R")
cron_add(cmd1, "daily", at="3PM")

#Window
taskschedulerAddin()
```

Data manipulation

NYTimes data manipulation

1. All the files that were extracted from the API and saved as .Rdata files were loaded and appended using the following code:

```
################
# loading API movie data
###############

# appending the data
Movie2018H1 <- load("Movie2018H1.Rdata")
Movie2018H1 <- MovieFile
Movie2018H2 <- load("Movie2018H2.Rdata")
Movie2018H2 <- MovieFile
Movie2019H1 <- load("Movie2019H1.Rdata")
Movie2019H1 <- MovieFile
Movie2019H2 <- load("Movie2019H2.Rdata")
Movie2019H2 <- MovieFile
Movie2020H1 <- load("Movie2020H1.Rdata")
Movie2020H1 <- MovieFile
Movie2020H2 <- load("Movie2020H2.Rdata")
Movie2020H2 <- MovieFile
load("Movie2021H1.Rdata")
load("Movie2021H2.Rdata")
load("Movie2022H1.Rdata")
load("Movie2022H2.Rdata")
load("Movie2023H1.Rdata")
load("Movie2023H2.Rdata")


merged_all <- rbind(Movie2018H1, Movie2018H2,
                    Movie2019H1, Movie2019H2,
                    Movie2020H1, Movie2020H2,
                    Movie2021H1, Movie2021H2,
                    Movie2022H1, Movie2022H2,
                    Movie2023H1, Movie2023H2)
```

2. Given that the API provides movie and book reviews, we filter out all the observations that are not movie reviews using information from the URL column:

```
# keeping only movies using the /movies/ pattern in response.docs.web_url
merged <- merged_all %>%
  filter(str_detect(response.docs.web_url, "/movies"))
# we have 4268 movies, now keep only unique ones
```

This gives a dataset with 4268 rows (movie reviews).

3. Only keep the unique reviews since some of the reviews might have been pulled more than once due to an overlap of the time period in the API call. This leaves us with 4142 unique movie reviews.

```
merged_unique <- merged %>% distinct(response.docs.snippet, .keep_all = TRUE)
```

4. Create columns with variables of interest including review author, critic's pick, review date, movie type, and movie name:

```
# Add author name (removing "By")
stopwords = c("By")
merged_unique$author <-gsub("By","",as.character(merged_unique$response.docs.byline.original))

# clean critic's pick column
merged_unique$criticpick <- ifelse(is.na(merged_unique$response.docs.headline.kicker),0,1)

# clean date column
merged_unique <- merged_unique %>%
  mutate (month = month(response.docs.pub_date),
          year = year(response.docs.pub_date))
merged_unique$monthyear <- as.yearmon(paste(merged_unique$year, merged_unique$month), "%Y %m")

# loop to extract movie type and movie name
merged_unique <- merged_unique %>% mutate(type=0,
                                          movie_name=0)
for (i in 1:nrow(merged_unique)) {
  typetable=as.data.frame(merged_unique$response.docs.keywords[i])
  merged_unique$type[i]=ifelse(any(typetable$value=="Documentary Films and Programs"),"Documentary Films
                               ifelse(any(typetable$value=="Movies"),"Movies",
                                      ifelse(any(typetable$value=="Animated Films"),"Animated Films",NA
  merged_unique$movie_name1[i] <- typetable[typetable$name=="creative_works", "value"][1]
}

#table(merged_unique$type, useNA = "always") #12 movies have no type information

#Extract movie name from URL
merged_unique$movie_name2 <- str_match(merged_unique$response.docs.web_url,
                                       "/movies/\\s*(.*?)\\s*-review")[,2]

#Use name from URL if the information is not there in the response.docs.keywords list
merged_unique$movie_name <- ifelse(is.na(merged_unique$movie_name1),
                                   merged_unique$movie_name2,merged_unique$movie_name1)

check3 <- merged_unique %>% filter(is.na(movie_name))
#only 3 have unidentifiable name from either source. We proceed without them.
```

5. Clean up the movie names sufficiently so that they can be effectively merged with the IMDb data

```
# remove (Movie) ; replace dashes with spaces ; remove white space ; lower case all for movie names
merged_unique$movie_name <-str_replace(merged_unique$movie_name, " \\s*\\([^\\)]+\\)", "")
merged_unique$movie_name <-str_replace_all(merged_unique$movie_name,'-',' ')
merged_unique$movie_name <- trimws(merged_unique$movie_name)
merged_unique$movie_name <- tolower(merged_unique$movie_name)
# replace & with "and", remove ":"
merged_unique$movie_name <-str_replace_all(merged_unique$movie_name,':','')
merged_unique$movie_name <-str_replace_all(merged_unique$movie_name,'&','and')
```

6. Keep relevant columns only and get the number of reviews per month and year

```
api_unique <- merged_unique %>% dplyr::select (response.docs.pub_date, month, year, monthyear,
                                   response.docs.news_desk, type, author, criticpick,
```

```
                                    movie_name, movie_name1, movie_name2) %>%
  group_by(monthyear) %>%
  mutate(n_month=n()) %>%
  ungroup() %>% group_by(year) %>%
  mutate(n_year=n()) %>%
  ungroup() %>%
  group_by(monthyear) %>%
  arrange(monthyear)
```

IMDb data preparation

1. Keep movies from 2017 onwards only since the NYTimes movie reviews start in 2017 and it is very unlikely for the NYTimes to review movies from over a year before and merge this information with that from the ratings dataset by movie ID (tconst).

```
titles <- title.basics %>%
  filter(startYear>2016,
         titleType=="movie"|titleType=="tvMovie")
ratings <- title.ratings
joined <- left_join(titles,ratings, by="tconst") %>%
  rename(movie_name = primaryTitle,
         release_year=startYear) %>%
  mutate(release_year=as.numeric(release_year))
```

2. Clean up movie names in preparation for the merge with NYTimes data

```
# Clean up movie names: replace dashes with spaces ; remove white space ; lower case all for movie name
joined$movie_name <-str_replace_all(joined$movie_name,'-',' ')
joined$movie_name <- trimws(joined$movie_name)
joined$movie_name <- tolower(joined$movie_name)
joined$movie_name <-str_replace_all(joined$movie_name,':','')
joined$movie_name <-str_replace_all(joined$movie_name,'&','and')
```

Merging the two data sources

The code chunk below shows the merge and the filtering out of non-unique matches from the two data sources

```
withratings <- left_join(api_unique, joined, by="movie_name") #5350
```

```
## Warning in left_join(api_unique, joined, by = "movie_name"): Detected an unexpected many-to-many rela
## i Row 3 of `x` matches multiple rows in `y`.
## i Row 47319 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
# keep only the joins where release date is BEFORE review date
withratings <- withratings %>% filter(release_year <= year) %>%
  group_by(movie_name) %>% mutate(n_matches=n(),
                                  unclear=ifelse(n_matches>1,1,0)) #4456 row left
table(withratings$n_matches, useNA = "always") # 3232 uniquely matched out of 4142
```

```
##
##    1    2    3    4    5    6    7    8    9   10   12 <NA>
## 3235  516  264  156  115   48   49   16   27   10   12    0
```

```
3232/4142
```

```
## [1] 0.7802994
```

```
#keep only matched movies
matched <- withratings %>% filter(n_matches==1)
```

3232 out of 4142 movies were uniquely matched with the IMDb data. A random sub-sample of those were manually checked, and they were all true matches. Our match rate is roughly 80%, which is not bad.

Transforming the genre data to a usable format

```
# get the genres:
genre <- unique(word(matched$genres, sep=","))
length(genre) #19 genres represented
```

```
## [1] 19
```

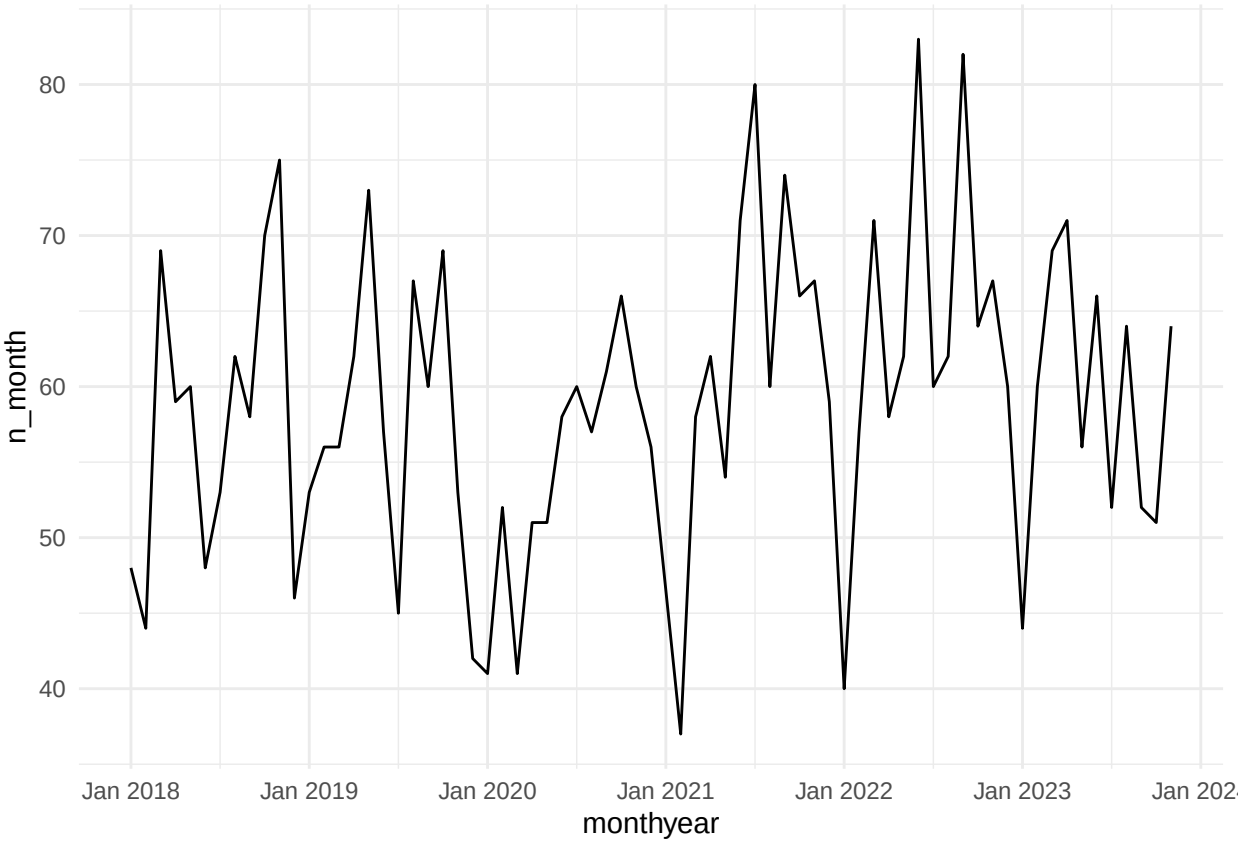
```
# make them into indicator columns:
matched$Documentary <- grepl("Documentary", matched$genres)
matched$Drama <- grepl("Drama", matched$genres)
matched$Animation <- grepl("Animation", matched$genres)
matched$Comedy <- grepl("Comedy", matched$genres)
matched$Crime <- grepl("Crime", matched$genres)
matched$Action <- grepl("Action", matched$genres)
matched$Adventure <- grepl("Adventure", matched$genres)
matched$Biography <- grepl("Biography", matched$genres)
matched$Horror <- grepl("Horror", matched$genres)
matched$Mystery <- grepl("Mystery", matched$genres)
matched$Animation <- grepl("Animation", matched$genres)
matched$Thriller <- grepl("Thriller", matched$genres)
matched$SciFi <- grepl("Sci-Fi", matched$genres)
matched$Fantasy <- grepl("Fantasy", matched$genres)
matched$Family <- grepl("Family", matched$genres)
matched$Musical <- grepl("Musical", matched$genres)
matched$History <- grepl("History", matched$genres)
matched$Music <- grepl("Music", matched$genres)
matched$Romance <- grepl("Romance", matched$genres)
```

Analysis and Visualizations

To visualize the time trends in the number of reviews published by the NYTimes during the period of analysis, we use the following code and inspect the resulting plot.

```
#review trends
api_unique %>%
  arrange(monthyear) %>%
  group_by(monthyear, year) %>%
  summarise(n_month=n()) %>%
  ggplot() +
  aes(x = monthyear, y = n_month) +
  geom_line()+
  scale_color_gradient() +
  theme_minimal()
```

```
## `summarise()` has grouped output by 'monthyear'. You can override using the
## `.groups` argument.
```



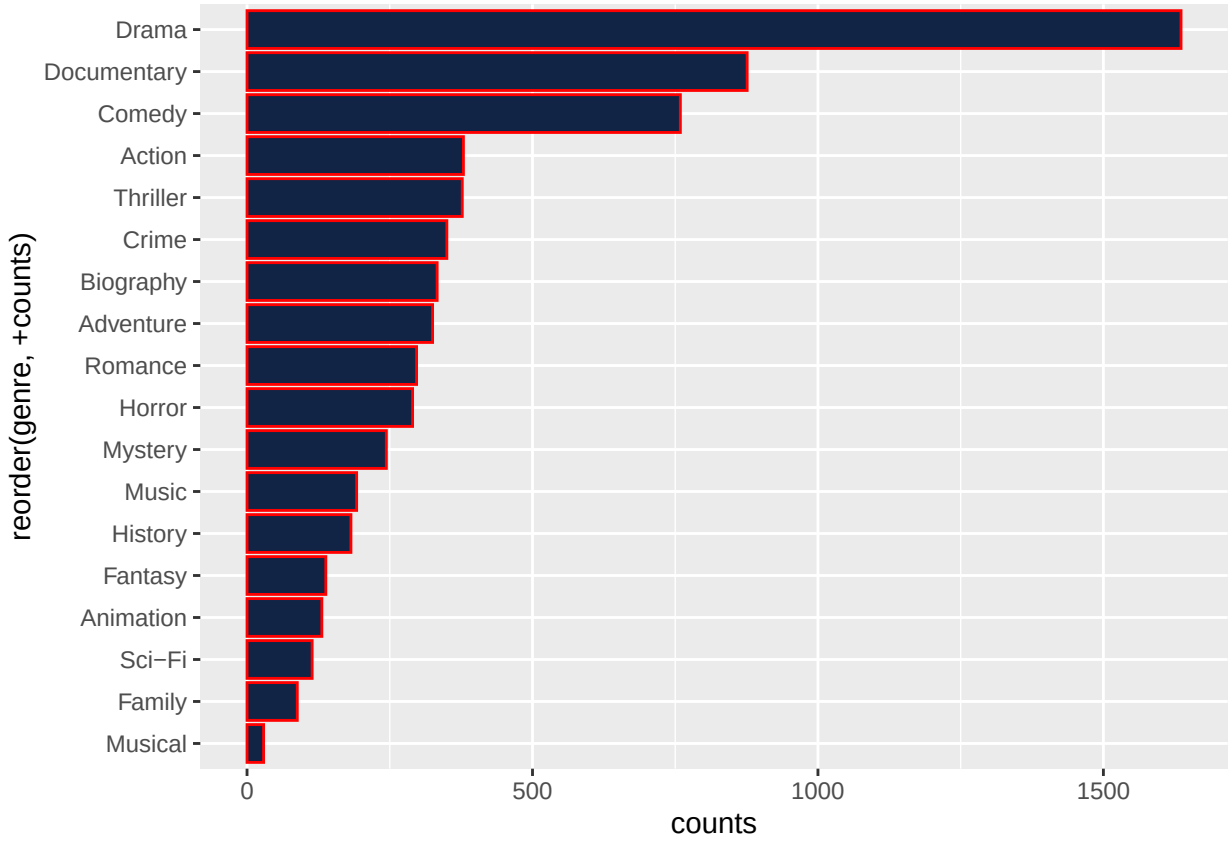
The plot shows strong seasonal trends, but there is no discernible longer term pattern.

Next, we turn the the distribution of the movies reviewed across genres and create the visualization below to summarize the data.

```
#distribution of reviewed movies across genres
genre <- genre[-14]
counts <- c(sum(matched$Documentary==1), sum(matched$Drama==1),
              sum(matched$Comedy==1), sum(matched$Crime==1),
              sum(matched$Action==1), sum(matched$Adventure==1),
              sum(matched$Biography==1), sum(matched$Horror==1),
              sum(matched$Mystery==1), sum(matched$Animation==1),
```

```
                sum(matched$Thriller==1), sum(matched$SciFi==1),
                sum(matched$Fantasy==1), sum(matched$Family==1),
                sum(matched$Musical==1), sum(matched$History==1),
                sum(matched$Music==1), sum(matched$Romance==1))
genre_table=as.data.frame(cbind(genre,counts))
genre_table$counts <- as.numeric(genre_table$counts)
genre_table <- genre_table[order(genre_table$counts, decreasing = TRUE), ]

p <- ggplot(genre_table, aes(x = reorder(genre, +counts), y = counts)) +
  geom_bar(stat="identity", color='red',fill="#112446") +
  coord_flip()
p
```



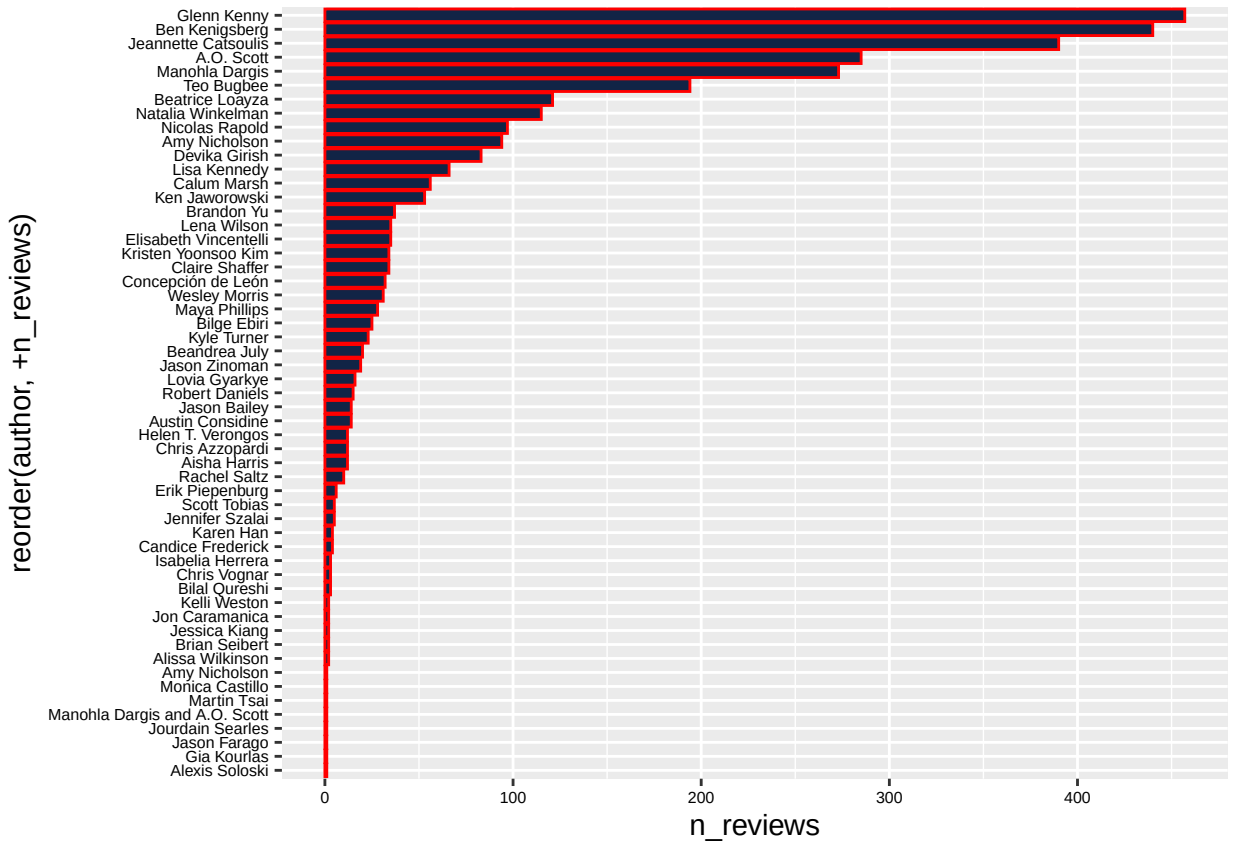
The genre most reviewed is drama, with over 1600 movie reviews, followed by documentary (~870) and then comedy (~760). Musicals are reviewed at the lowest frequency - only 30 musicals were reviewed during the period of analysis.

As for the number of reviews per author, the information is summarized in the barchart below.

```
#number of reviews per author
auth_rate <- matched %>% group_by(author) %>%
  summarise(mean_imdb_rating=mean(averageRating, na.rm=TRUE),
            n_reviews=n())
auth_rate <- na.omit(auth_rate)

p <- ggplot(auth_rate, aes(x = reorder(author, +n_reviews), y = n_reviews)) +
  geom_bar(stat="identity", color='red',fill="#112446") +
```

```
  coord_flip() +
  theme(axis.text.x = element_text(color = "black", size = 6),
        axis.text.y = element_text(color = "black", size = 6))
p
```



Glenn Kenny and Ben Kanigsberg are the two most prolific movie reviewers at the NYTimes during our analysis period. It is important to note that there is a good number of reviewers who did less than 10 reviews each. Those reviewers are excluded from further analyses and the rest of the plots for clarity.

We turn next to the question of whether authors/reviewers tend to specialize in a genre or not. We do that by calculating the percentage of movies by author that are tagged as a specific genre. The plot below show the most frequently reviewed genre by author.

```
#genre per author
genres_author <- matched %>% group_by(author) %>%summarise(n=n(),
                                                           p_pick=mean(criticpick),
                                                           p_rating <- mean(averageRating, na.rm=TRUE),
                                                           p_Documentary=mean(Documentary),
                                                           p_Drama=mean(Drama),
                                                           p_Animation=mean(Animation),
                                                           p_Comedy=mean(Comedy),
                                                           p_Crime=mean(Crime),
                                                           p_Action=mean(Action),
                                                           p_Adventure=mean(Adventure),
                                                           p_Biography=mean(Biography),
                                                           p_Horror=mean(Horror),
                                                           p_Mystery=mean(Mystery),
```

```
                                                  p_Thriller=mean(Thriller),
                                                  p_SciFi=mean(SciFi),
                                                  p_Fantasy=mean(Fantasy),
                                                  p_Family=mean(Family),
                                                  p_Musical=mean(Musical),
                                                  p_History=mean(History),
                                                  p_Music=mean(Music),
                                                  p_Romance=mean(Romance)) %>%
  filter(n>9)
na.omit(genres_author)
```

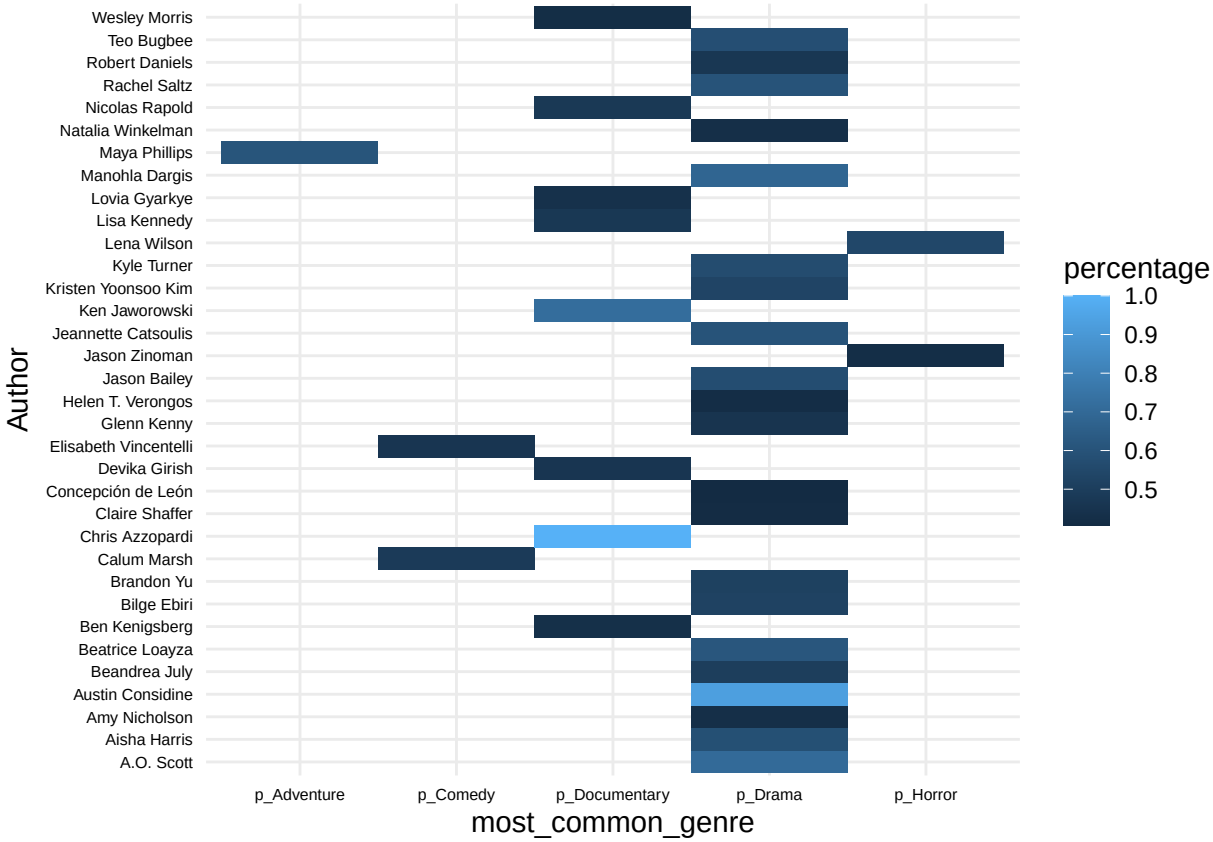
```
## # A tibble: 34 x 22
##    author      n p_pick p_rating <- mean(ave~1 p_Documentary p_Drama p_Animation
##    <chr>   <int>  <dbl>                 <dbl>         <dbl>   <dbl>       <dbl>
##  1 " A.O.~   285 0.291                  6.78         0.119   0.702      0.0211
##  2 " Aish~    12 0.167                  6.58         0.0833  0.583      0
##  3 " Amy ~    94 0.213                  6.23         0.0957  0.426      0.0851
##  4 " Aust~    14 0.286                  6.34         0       0.929      0
##  5 " Bean~    20 0.25                   6.42         0.35    0.5        0.05
##  6 " Beat~   121 0.149                  6.50         0.190   0.612      0.0496
##  7 " Ben ~   440 0.114                  6.55         0.430   0.395      0.0523
##  8 " Bilg~    25 0.28                   6.31         0.24    0.52       0.12
##  9 " Bran~    37 0.162                  6.21         0.162   0.514      0.0270
## 10 " Calu~    56 0.0893                 6.1          0.214   0.268      0.161
## # i 24 more rows
## # i abbreviated name: 1: `p_rating <- mean(averageRating, na.rm = TRUE)`
## # i 15 more variables: p_Comedy <dbl>, p_Crime <dbl>, p_Action <dbl>,
## #   p_Adventure <dbl>, p_Biography <dbl>, p_Horror <dbl>, p_Mystery <dbl>,
## #   p_Thriller <dbl>, p_SciFi <dbl>, p_Fantasy <dbl>, p_Family <dbl>,
## #   p_Musical <dbl>, p_History <dbl>, p_Music <dbl>, p_Romance <dbl>
```

```
genre_dist <- genres_author %>%
  mutate(maxx = pmax(p_Documentary, p_Drama,p_Comedy, p_Crime,
                     p_Action,p_Adventure,p_Biography,p_Horror,
                     p_Mystery,p_Animation,p_Thriller,p_SciFi  ,
                     p_Fantasy,p_Family,p_Musical ,p_History,
                     p_Music,p_Romance))

genre_max <- genres_author %>%
  dplyr::select(p_Documentary, p_Drama,p_Comedy, p_Crime,
         p_Action,p_Adventure,p_Biography,p_Horror,
         p_Mystery,p_Animation,p_Thriller,p_SciFi  ,
         p_Fantasy,p_Family,p_Musical ,p_History,
         p_Music,p_Romance) %>%
  rowwise %>%
  mutate(Max = names(.)[which.max(c(p_Documentary, p_Drama,p_Comedy, p_Crime,
                                    p_Action,p_Adventure,p_Biography,p_Horror,
                                    p_Mystery,p_Animation,p_Thriller,p_SciFi  ,
                                    p_Fantasy,p_Family,p_Musical ,p_History,
                                    p_Music,p_Romance))]) %>%  ungroup
author_n_max <- cbind(genres_author$author,genre_max,genre_dist$maxx)

colnames(author_n_max)[20:21] <- c("most_common_genre","percentage")
```

```
colnames(author_n_max)[1] <- "Author"
ggplot(author_n_max) +
  aes(
    x = Author,
    y = most_common_genre,
    fill = percentage
  ) +
  geom_tile() +
  scale_fill_gradient() +
  coord_flip() +
  theme_minimal() +
  theme(axis.text.x = element_text(color = "black", size = 6),
        axis.text.y = element_text(color = "black", size = 6))
```



The result that we see is expected, since the genre reviewed most by the NYTimes is drama. However, we do see some reviewers specializing in horror, documentary, and comedy reviews. In the plot above, specialization is defined the most common genre the author writes about (it also means that in all of these cases, more than 40% of the reviews they wrote belonged to that specific genre). The color gradient indicates the percentage of reviews they wrote that belonged to that most common genre, with the lighter color indicating more specialization.

Next, we turn to the distribution of IMDb ratings for the movies that each author reviewed, visualized as a ridgeplot.

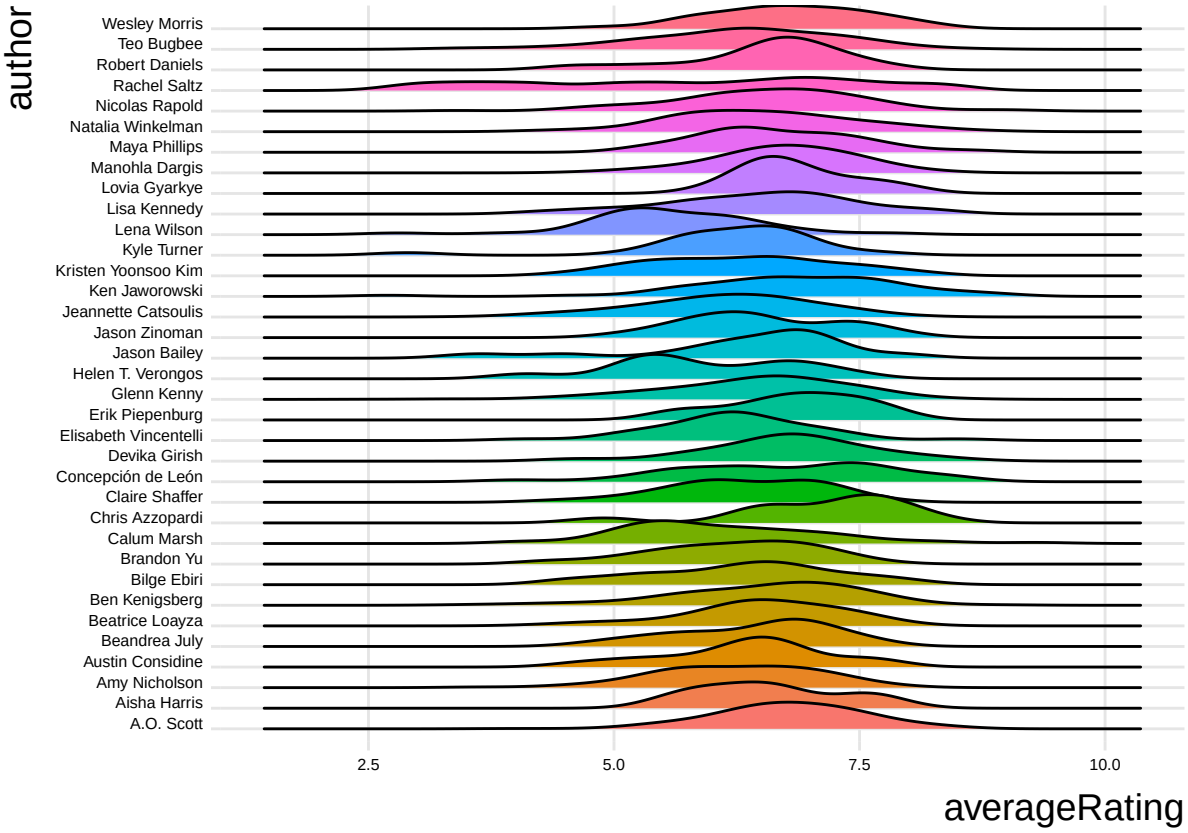
```
#average imdb rating per author (those with more than 10 reviews)
auth_rate <- matched %>% group_by(author) %>%
  summarise(mean_imdb_rating=mean(averageRating, na.rm=TRUE),
```

```
            n_reviews=n()) %>%
  filter(n_reviews>9)
auth_rate <- na.omit(auth_rate)
# p <- ggplot(auth_rate, aes(x = reorder(author, +mean_imdb_rating), y = mean_imdb_rating)) +
#   geom_bar(stat="identity", color='red',fill="#112446") +
#   coord_flip()
# p

ridge <- matched %>% group_by(author) %>% mutate(n=n()) %>% filter(n>5)
ggplot(ridge, aes(x = averageRating, y = author, fill = author)) +
  geom_density_ridges() +
  theme_ridges() +
  theme(legend.position = "none") +
  theme(axis.text.x = element_text(color = "black", size = 6),
        axis.text.y = element_text(color = "black", size = 6))
```

```
## Picking joint bandwidth of 0.354
```

```
## Warning: Removed 16 rows containing non-finite values
## (`stat_density_ridges()`).
```



Finally, we turn to the question of whether movies chosen as "critic's pick" tend to be rating more generously on IMDb. The plot below shows the ratings distribution by critic's pick status.

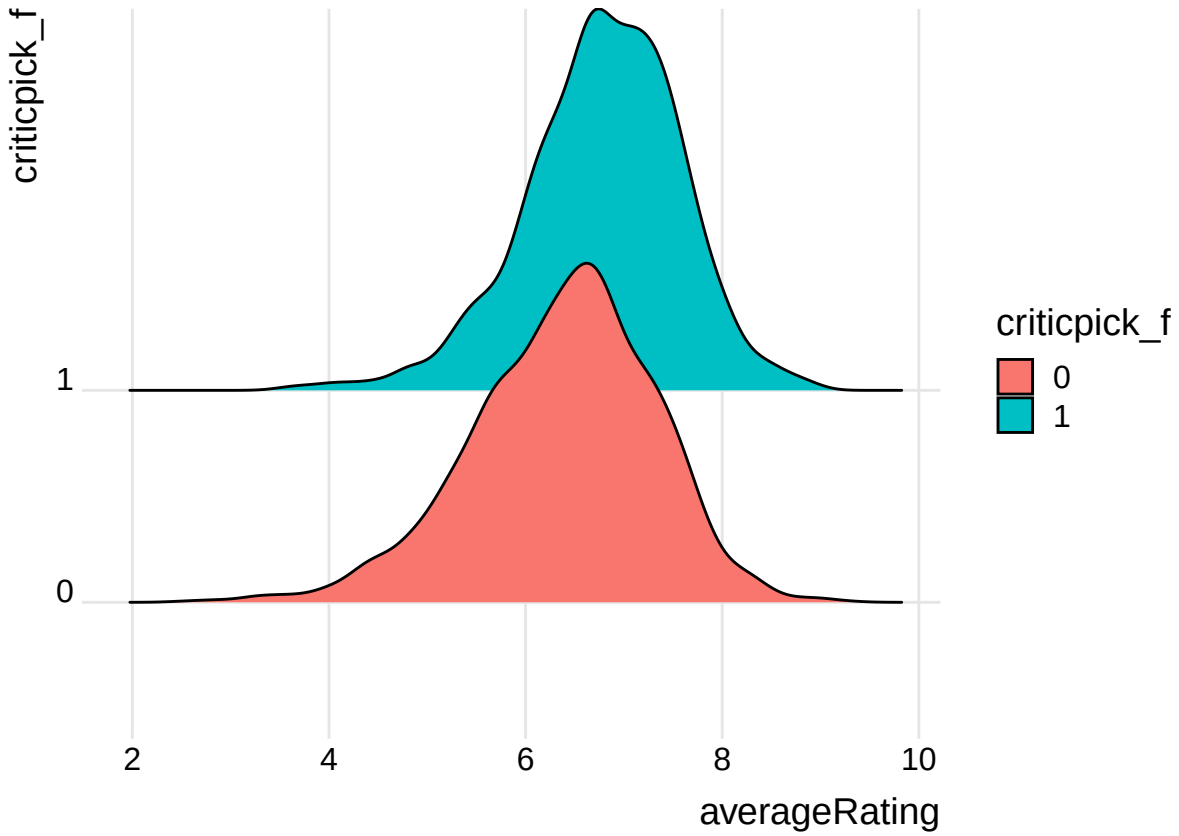
```
#critics pick ratings vs non critics pick
matched %>% group_by(criticpick) %>% summarise(avg_imdb_rating=mean(averageRating, na.rm=TRUE))
```

```
## # A tibble: 2 x 2
##   criticpick avg_imdb_rating
##        <dbl>           <dbl>
## 1          0            6.38
## 2          1            6.78
```

```
matched$criticpick_f=as.factor(matched$criticpick)
ggplot(matched, aes(x = averageRating, y = criticpick_f, fill = criticpick_f)) +
  geom_density_ridges() +
  theme_ridges()
```

```
## Picking joint bandwidth of 0.175
```

```
## Warning: Removed 17 rows containing non-finite values
## (`stat_density_ridges()`).
```



The plot shows that indeed, the distribution of IMDb ratings for the critic's pick movies is to the right of that of the movies that did not receive that title.

Conclusion

By doing this project, we had a chance to work with an API, collect data from the real world, clean these collected data, and then use it for analysis. We learned that data from the real world is very messy. As

systematic as we want it to be, we have encountered many variances in data recording, likely a result of human errors. With the data collected, there are many other things we can do with it but haven't have a chance. A next step we have for self exploration is to fit the data through a sentiment analysis.