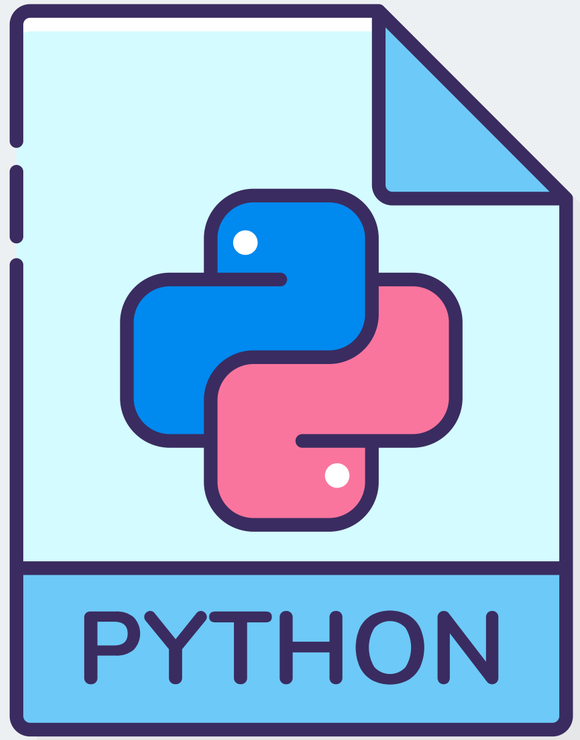
Letterboxd Recommendation System

(simple ver.)



ITM Project Presentation

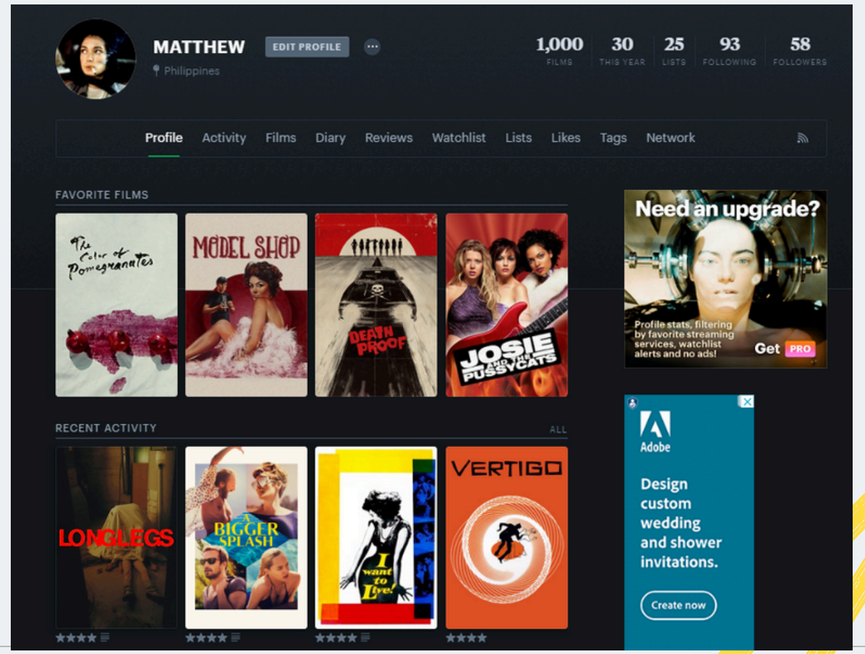
Chong, Gutierrez, Manalili
Section M - Group 6

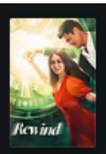
Introduction

What is Letterboxd?

A site that allows users to log the films they have watched in a diary-like format.

Also boasts a follower-following feature that allows users to interact with each other and better engage in film discourse





Rewind 2023

★★½ Watched 11 Jan 2024

Rewind is a tale of sweet contradiction. It searches for the meaning of life in the midst of death and finds agency in putting your faith in someone else. The motif of a ticking clock represents the ticking clocks inside all of us as even in the ability to rewind, we are reminded of the little time we have.

The success of Rewind is a perfect byproduct of the post-COVID world, a period when our time was squandered and our mortality was questioned. Ideated in 2019, I'm sure the creators had different intentions. But with the outbreak in mind, the film takes on an entirely new meaning as a contemplation of what was lost and the lessons that can be found following these losses. Hopefully, the sentiment will only grow to inspire us as we approach the years ahead.

Therefore, to answer the question, is Rewind good? Not really. But has it proven to be the most important Filipino film of the decade so far? Absolutely.

♥ 1 like

NEW FROM FRIENDS ALL ACTIVITY

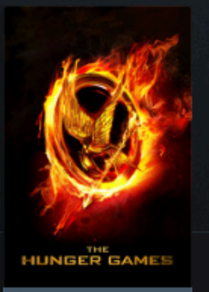
Mike Flanagan Jul 16

Philbert Dy ★★ Jul 17


faye ★★★★ Jul 16

cookie ★★★½ Jul 16

Karsten ★★½ Jul 16



zoë rose bryant ★★★★★ Jul 16

POPULAR REVIEWS WITH FRIENDS MORE

evie

Longlegs 2024

★★★ 1

so happy to see both annabelle and the lady from behind the diner in mulholland drive employed!

♥ Like review 10 likes

Claire Rowden

Monte Carlo 2011

★★★ 2

this era of selena gomez movies is something that can actually be soooo personal to a girl

♥ Like review 9 likes

james

Longlegs 2024

★½ 5

maybe osgood perkins should rethink not watching pearl so he can take notes on how to make a better horror film

♥ Like review 29 likes

becky

The Amazing Spider-Man 2012

★★★ 1

childhood classic for me. rewatched because my tiktok has been filled with andrew garfield edits and i'm in love with him.

i forgot how sad uncle ben's death was wtf ???

anyways andrew is so god damn hot.

edit: changed to a three star because held up to the other spider-man movies it does not compare but i still luv

♥ Like review 1 like

748K 209K 237K 208

WHERE TO WATCH Trailer

Amazon US RENT BUY DISC

Apple TV PH RENT BUY

Google Play ... PH RENT BUY

Google Play ... US RENT BUY

YouTube US RENT BUY

Go PRO to customize this list

All services... JustWatch

POPULAR REVIEWS MORE

Review by **siobhan** ★★★★★ ♥ 7

This review may contain spoilers.

MY FRIENDS WIFE GOT POSSESSED BY A GHOST SO I STALKED HER (GONE WRONG!) (not clickbait)

♥ Like review 6,921 likes

Review by **Leticia Fernandes** ★★★★★ ♥ 22

Justice for Midge

♥ Like review 5,974 likes

Review by **Rida** ★★★★★ 52

This review may contain spoilers.

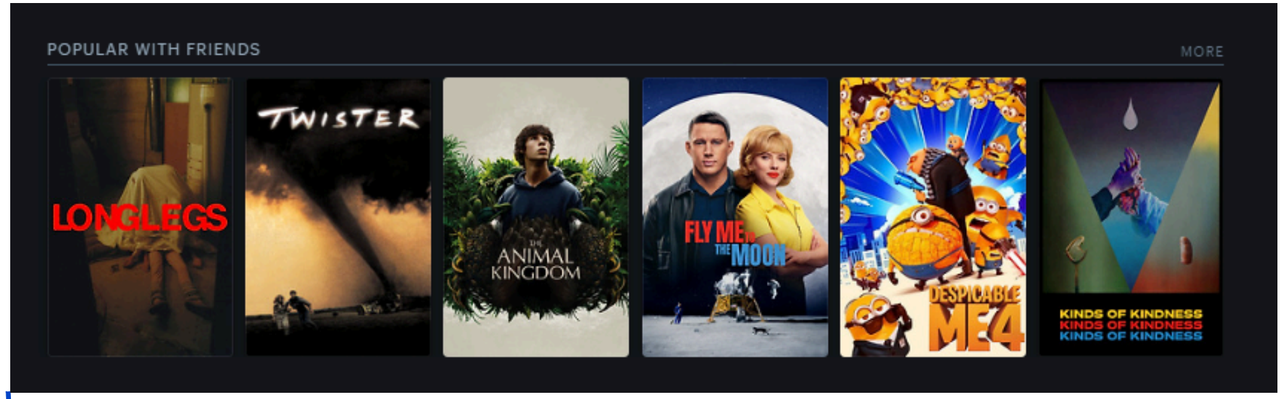
Watching a Hitchcock film is rather like going to a play: you're constantly reminded that everything is just pretend. Even if the experience is great, it never quite makes you realize that the same things can happen to you, that people like these can plausibly exist in real life. But *Vertigo* feels startlingly true, almost confessional, because it's clear that it's Hitchcock's most personal film, the one that came closest to revealing his inner turmoil.

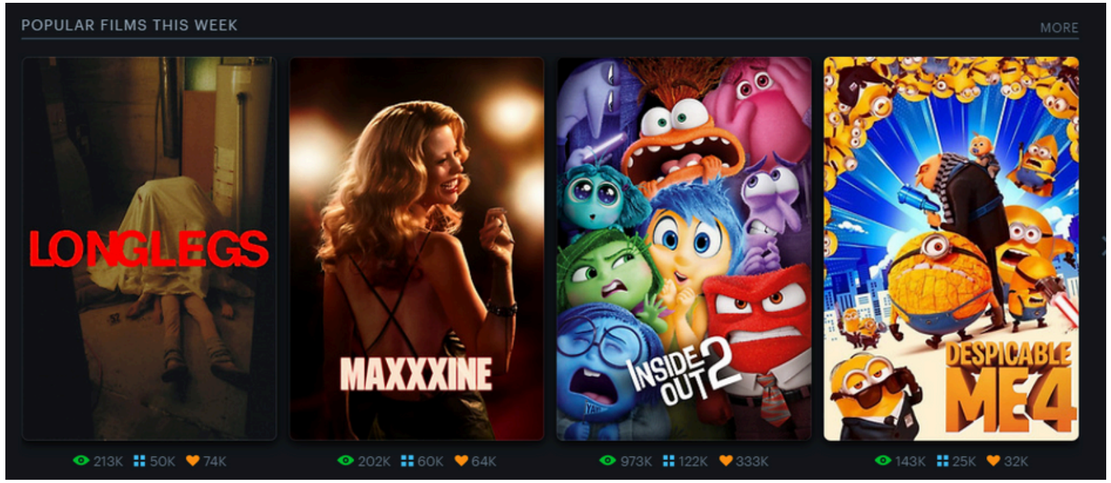
Vertigo has a plot so convoluted that it's fully apparent that it's a mere set-up to get to the heart of the film: a chronicle of the protagonist Scottie's obsession. The film is also considerably sympathetic toward the two slighted women in Scottie's life: his... **more**

♥ Like review 4,657 likes

Understanding the Problem

- Lack of a user-centric movie recommendation feature
- Only recommends based on popularly logged movies



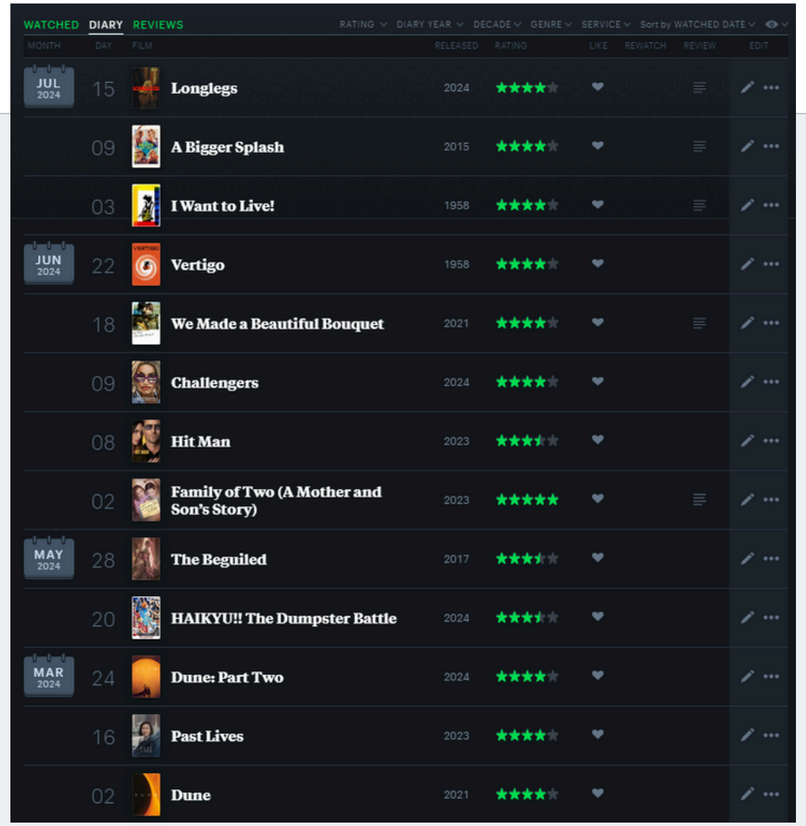


Statement of the Solution

Using diary entries as a basis to create movie recommendations

Objectives

- Extract the complete diary history of letterboxd users
- Use that history to derive movies that best suit the user
- Give substantial information on each recommended film



Data Collection

GETTING FILM URLS IN DIARY

```
base_url = 'https://letterboxd.com'
user_name = input('Enter user name: ')
r = requests.get(f'{base_url}/{user_name}/films/diary/')
soup = BeautifulSoup(r.content, "html.parser")
list_urls = [base_url + a.get('href') for a in soup.select('h3>a[href]')]
if list_urls == []:
    print('No films found')
    sys.exit(1)
## Subsequent Pages
links = soup.find_all('li', class_="paginate-page")
try:
    start_character = str(links[-1]).find('/page/') + 6
    end_character = str(links[-1]).find('/">')
    end_page = int(str(links[-1])[start_character:end_character])
except IndexError:
    end_page = 1
if end_page >1:
    for page in range(2, end_page + 1):
        r = requests.get(f'{base_url}/{user_name}/films/diary/page/{page}')
        soup = BeautifulSoup(r.content, "html.parser")
        new_urls = [base_url + a.get('href') for a in soup.select('h3>a[href]')]
        list_urls.extend(new_urls)
```

- Getting the username
- Save film urls
- Check for films
- Checking multiple diary pages

Data Collection

GETTING FILM URLS IN DIARY

```
base_url = 'https://letterboxd.com'
user_name = input('Enter user name: ')
r = requests.get(f'{base_url}/{user_name}/films/diary/')
soup = BeautifulSoup(r.content, "html.parser")
list_urls = [base_url + a.get('href') for a in soup.select('h3>a[href]')]
if list_urls == []:
    print('No films found')
    sys.exit(1)
## Subsequent Pages
links = soup.find_all('li', class_="paginate-page")
try:
    start_character = str(links[-1]).find('/page/') + 6
    end_character = str(links[-1]).find('/">')
    end_page = int(str(links[-1])[start_character:end_character])
except IndexError:
    end_page = 1
if end_page >1:
    for page in range(2, end_page + 1):
        r = requests.get(f'{base_url}/{user_name}/films/diary/page/{page}')
        soup = BeautifulSoup(r.content, "html.parser")
        new_urls = [base_url + a.get('href') for a in soup.select('h3>a[href]')]
        list_urls.extend(new_urls)
```

- Getting the username
- Save film urls

Data Collection

GETTING FILM URLS IN DIARY

```
base_url = 'https://letterboxd.com'
user_name = input('Enter user name: ')
r = requests.get(f'{base_url}/{user_name}/films/diary/')
soup = BeautifulSoup(r.content, "html.parser")
list_urls = [base_url + a.get('href') for a in soup.select('h3>a[href]')]
if list_urls == []:
    print('No films found')
    sys.exit(1)
## Subsequent Pages
links = soup.find_all('li', class_="paginate-page")
try:
    start_character = str(links[-1]).find('/page/') + 6
    end_character = str(links[-1]).find('/">')
    end_page = int(str(links[-1])[start_character:end_character])
except IndexError:
    end_page = 1
if end_page >1:
    for page in range(2, end_page + 1):
        r = requests.get(f'{base_url}/{user_name}/films/diary/page/{page}')
        soup = BeautifulSoup(r.content, "html.parser")
        new_urls = [base_url + a.get('href') for a in soup.select('h3>a[href]')]
        list_urls.extend(new_urls)
```

- Check for films
- Checking multiple diary pages

Data Collection

GETTING THE TITLES AND RELEASE YEAR

```
for url in list_urls:
    r = requests.get(url)
    soup = BeautifulSoup(r.content, "html.parser")
    ## Getting the Titles and Release Year
    span = soup.find_all('span', class_="film-title-wrapper")
    span_soup = BeautifulSoup('\n'.join(map(str, span)))
    a_list = list(span_soup.find_all('a'))
    count = 0
    for a in a_list:
        a_str =str(a)
        start_chr = a_str.find('">')+2
        end_chr = a_str.find("</")
        count += 1
        if count == 1:
            titles.append(a_str[start_chr:end_chr])
        if count == 2:
            release_years.append(a_str[start_chr:end_chr])
        if count == 3:
            print('uh oh, Error at titles/year')
            break
```

- For loop
- Titles and Release Year
- Appending to lists

Data Collection

GETTING THE RATINGS

```
## Getting the Ratings
script_str = str(soup.find_all('script', type="application/ld+json"))
start_chr = script_str.find('"ratingValue":')+14
end_chr = script_str.find(',"worstRating"')
if end_chr == -1:
    ratings_list.append('null')
    continue
ratings_list.append(script_str[start_chr:end_chr])
```

- Null

Data Collection

GETTING THE GENRES

```
## Getting the Genres
all_script_tags = soup.find_all("script")
script_list = []
movie_genres = []
for i in all_script_tags:
    script_list.append(str(i))
raw_script_tag = script_list[-1]
for i in all_genres:
    if i in raw_script_tag:
        movie_genres.append(i)
        top_genres[i] += 1
```

- Looping through all genres
- “in”

Data Collection

FORMATTING DATA AND TOP GENRES

```
## Formatting the Data Dictionary
if titles[-1] not in data:
    data[titles[-1]] = [release_years[-1],ratings_list[-1], movie_genres]
```

```
# SORTING GENRE VALUES
sorted_top_genres = sorted(top_genres.items(), key=lambda item: item[1], reverse=True)
```

- Use of -1 index
- movie_genres (reinitialized)

Data Collection

LETTERBOXD BROWSING

```
113  top_3 = [genre for genre, count in sorted_top_genres[:3]]
114
115  soup = BeautifulSoup('', 'html.parser')
116
117  # use webdriver to open letterboxd based on rating and genre
118  driver = webdriver.Chrome()
119
120  # get soup of all combined
121  driver.get(f"https://letterboxd.com/films/genre/{'+'.join(top_3)}/")
122  time.sleep(0.5)
123  html_doc = driver.page_source
124  soup1 = BeautifulSoup(html_doc, 'html.parser')
125  soup.extend(soup1.contents)
```

- use webdriver to fetch dynamic content
- fetch html code for top 3 genres' pages

Data Collection

LETTERBOXD BROWSING

```
150 # parse out the tags that have the film info
151 initial_poster_lists = soup.find_all('li', class_='listitem poster-container')
152
153 # removes duplicate entries
154 poster_lists = list(OrderedDict.fromkeys(initial_poster_lists))
155
156 # get the url
157 url_list = []
158 for poster in poster_lists:
159     link_tag = poster.find('a')
160     link_url = link_tag['href']
161     url_list.append(f'https://letterboxd.com{link_url}')
162
163 # get the title, date, and rating string
164 title_date_ratingS = []
165 for poster in poster_lists:
166     info_tag = poster.find('a')
167     title_date_rating = info_tag['data-original-title']
168     title_date_ratingS.append(title_date_rating)
```

- extract urls and title-date-rating strings by find_all()

Data Collection

LETTERBOXD BROWSING

```
180  # function to get genre and theme
181  def scrape_url(url):
182      response = requests.get(url).content
183      soup = BeautifulSoup(response, 'html.parser')
184      # Extract the desired data from the soup object
185      genre_theme_tags = soup.find_all('a', {'href': re.compile(r'/films/genre/(.+)')})
186      listt = []
187      for tag in genre_theme_tags:
188          genre_theme = tag.text
189          listt.append(genre_theme)
190      return listt
191
192  # Use concurrent.futures to make all the requests in parallel
193  with concurrent.futures.ThreadPoolExecutor() as executor:
194      genre_theme_lists = list(executor.map(scrape_url, url_list))
```

- define a function to fetch the genres from the urls
- concurrent.futures & map() to make it run simultaneously

Data Cleaning and Preprocessing

LETTERBOXD BROWSING RESULTS

```
170  # get the title and rating, then put in a dictionary
171  title_rating_dict = {}
172  pattern = r"([^\(]+) \((\d{4})\) ([\d.]+)"
173  for info in title_date_ratingS:
174      match = re.search(pattern, info)
175      title = match.group(1)
176      release = match.group(2)
177      rating = match.group(3)
178      title_rating_dict[title] = [release, rating]
```

- use regex to group the info into 3 and store them into a dictionary

```
196  for i in range(len(title_rating_dict)):
197      title_rating_dict[list(title_rating_dict.keys())[i]] += [genre_theme_lists[i]]
```

Data Cleaning and Preprocessing

LETTERBOXD BROWSING RESULTS

```
199  # convert title_rating_dict to dataframe
200  search_results_df = pd.DataFrame(title_rating_dict).T
201  search_results_df = search_results_df.rename(columns={0: 'Release Year', 1: 'Rating', 2: 'Genres'})
202  sorted_search_df = search_results_df.sort_values(by=['Rating'], ascending=False)
203  sorted_search_df.reset_index(inplace=True)
204  sorted_search_df = sorted_search_df.rename(columns={'index':'movie title'})
205
206  sorted_search_df['Rating'] = sorted_search_df['Rating'].astype(float)
207  sorted_search_df['Release Year'] = sorted_search_df['Release Year'].astype(int)
```

- convert dict into DF and the number strings into float/int

Data Cleaning and Preprocessing

FILTERING & RECOMMENDING

```
345  # convert diary dictionary to dataframe
346  diary_df = pd.DataFrame(data).T
347  diary_df = diary_df.rename(columns={0: 'Release Year', 1: 'Rating', 2: 'Genres'})
348  diary_df['Release Year'] = diary_df['Release Year'].astype(int)
349  current_year = datetime.now().year
350  edited_year = pd.cut(diary_df['Release Year'],
351                       bins=[1899, current_year - 21, current_year - 6, current_year],
352                       labels=['old', 'notsorecent', 'recent'])
353  diary_df['Release Year'] = edited_year
```

- convert diary dictionary into DF
- recategorize movies to ‘recent’, ‘not so recent’, and ‘old’ based on Release Year

Data Cleaning and Preprocessing

FILTERING & RECOMMENDING

```
356  def recommend_movies(df):
357      # Filter the DataFrame by Year
358      recent_movies = df[df['Release Year'] >= current_year - 5]
359      notsorecent_movies = df[(df['Release Year'] < current_year - 5) & (
360      old_movies = df[df['Release Year'] < current_year - 20]
361
362      # Filter the DataFrame to only include movies with a rating of 3.5
363      high_rated_movies1 = recent_movies[recent_movies['Rating'] >= 3.5]
364      high_rated_movies2 = notsorecent_movies[notsorecent_movies['Rating'
365      high_rated_movies3 = old_movies[old_movies['Rating'] >= 3.5]
```

- group search results by Release Year and filter out low-rated

Data Cleaning and Preprocessing

FILTERING & RECOMMENDING

```
367     grouped = sorted_diary_df.groupby('Release Year')
368
369     # calculate percentage of movies from the 3 'periods'
370     recent_num = round((int(grouped.get_group('recent').count()['Release Yea
371     notsorecent_num = round((int(grouped.get_group('notsorecent').count()['R
372     old_num = round((int(grouped.get_group('old').count()['Release Year']) /
373
374     # Choose 10 random movies from the filtered DataFrame
375     recommended_movies = pd.concat([high_rated_movies1.sample(n=recent_num),
```

- get percentage of each ‘period’ in the user’s diary

Data Collection

REFORMATTING TITLE TO ROTTEN TOMATOES URL

```
def movie_reception(function, movie_title, year): #MAIN FUNCTION

    #-----------------------------------------
    year=str(year)
    try:
        movie_title1=movie_title
        unnecessary_punctuation_first = r"$@[].,'#()-\"!?'_;:/…"
        y=0
        while y!=len(unnecessary_punctuation_first):
            movie_title1=movie_title1.replace(unnecessary_punctuation_first[y]," ")
            y+=1
        movie_title1=movie_title1.replace("&",'and')
        movie_copy=movie_title1
        movie_title1=movie_title1+" "+year
        try:
            return function(movie_title1)
        except Exception:
            return function(movie_copy)
    except Exception:
        movie_title2=movie_title
        movie_title2=movie_title2.replace("&",'and')
        movie_copy=movie_title2
        movie_title2=movie_title2+" "+year
        try:
            return function(movie_title2)
        except Exception:
            try:
                return function(movie_copy)
            except Exception:
                if function==get_movie_reviews:
                    return "No reviews found on Rotten Tomatoes"
                else:
                    return "not a valid function"
```

- Considers movies with the same title or with different punctuations
- Using a while function to find the first positive review

Data Collection

OBTAINING URL FROM ROTTEN TOMATOES SEARCH ENGINES

```
def get_rotten_tomatoes_url(movie_name, year):
    year=str(year)
    query = urllib.parse.quote(movie_name)
    url = f"https://www.rottentomatoes.com/search?search={query}"
    response = requests.get(url)
    movie=response.text
    soup = BeautifulSoup(movie, 'html.parser')
    URLS=soup.find_all('a',class_='unset')
    a=[]
    links=[]
    dates=[]
    titles=[]
    x=0
    y=0
    parse_title=["/n"," "]
    for tag in URLS:
        if "thumbnail" in str(tag):
            a.append(tag)
    for tag in a:
        links.append(str(tag)[str(tag).find("https"):(str(tag).find(" slot")-1)])
    datetag=soup.find_all('search-page-media-row')
    for tag in datetag:
        dates.append(str(tag)[str(tag).find("releaseyear")+13:str(tag).find("releaseyear")+17])
    while x!=len(datetag):
        lee=datetag[x].text.replace("\n","")
        lee=lee.replace("  ","")
        titles.append(lee)
        x+=1
    while y!=len(datetag):
        if dates[y]==year and titles[y]==movie_name:
            return links[y]
        y+=1
```

- Searches for the url in the search engine of Rotten Tomatoes
- Uses movie title and release year to identify the appropriate page

Data Collection

GETTING ROTTEN TOMATOES REVIEWS

```
def get_movie_reviews(movie_title):
    unnecessary_punctuation_second = r"&$@[].,'#()-\"!?’_;:/…"
    y=0
    clean_movie=movie_title
    while y!=len(unnecessary_punctuation_second):
        clean_movie=clean_movie.replace(unnecessary_punctuation_second[y],"")
        y+=1
    movie_string=clean_movie.split()
    x=0
    Movie=""
    while x!=len(movie_string):
        Movie+=movie_string[x]
        if x==len(movie_string)-1:
            ""
        else:
            Movie+="_"
        x+=1
    z=0
    url = f"https://www.rottentomatoes.com/m/{Movie}/reviews"
    response = requests.get(url)
    reviews=response.text
    soup = BeautifulSoup(reviews, 'html.parser')
    presented_review=soup.find_all('p',class_='review-text')[z].text
    review_critic=soup.find_all('a',class_='display-name')[z].text
    score=soup.find all('score-icon-critics')
```

- Finding the list of critic reviews on rotten tomatoes by isolating tags
- Formatting movie titles according to url format

Data Collection

FINDING THE MOST POPULAR POSITIVE REVIEW

```
score=soup.find_all('score-icon-critics')
tag=0
number=len(score)
score1=score.copy()
while tag!=number:
    if "4" not in str(score1[tag]):
        score.remove(score1[tag])
    else:
        break
    tag+=1
score
score2=score[z]
senti=str(score2)
parse_critics = ["\n","  "]
punct=0
space=0
while "NEGATIVE" in senti:
    z+=1
    score2=score[z]
    senti=str(score2)
    presented_review=soup.find_all('p',class_='review-text')[z].text
    review_critic=soup.find_all('a',class_='display-name')[z].text
```

- Isolate and create a list of the respective sentiments for all reviews
- Using a while function to find the first positive review

Exploratory Data Analysis

1

RAW DATA

Program's primary application as a Letterboxd data collection service.

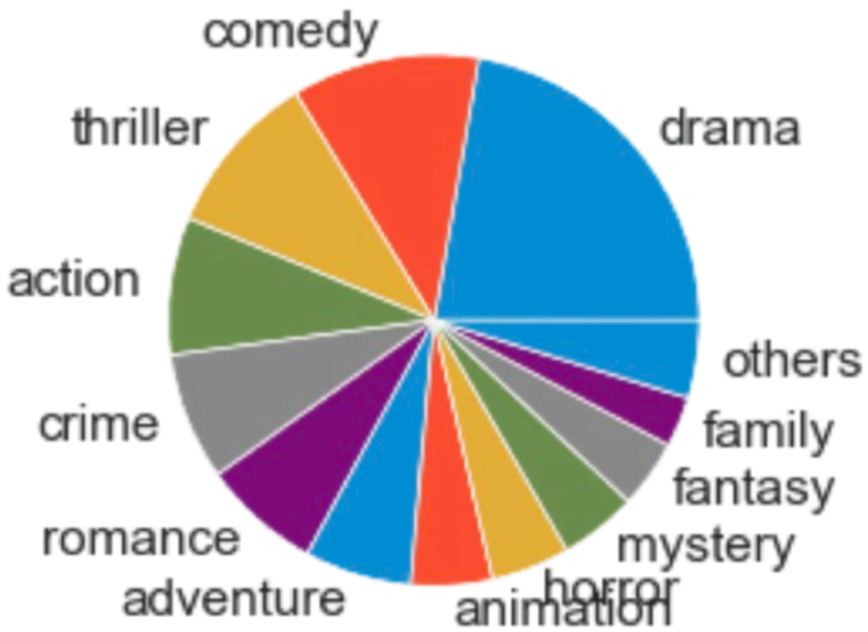
```
{'A Bigger Splash': ['2015', '4', ['drama', 'romance', 'thriller']],
'I Want to Live!': ['1958', '4', ['crime', 'drama']],
'Vertigo': ['1958', '4', ['mystery', 'romance', 'thriller']],
'We Made a Beautiful Bouquet': ['2021', '4', ['drama', 'romance']],
'Challengers': ['2024', '4', ['drama', 'romance']],
'Hit Man': ['2023', '3.5', ['comedy', 'crime', 'romance']],
"Family of Two (A Mother and Son's Story)": ['2023', '5', ['drama']],
'The Beguiled': ['2017', '3.5', ['drama']],
'HAIKYU!! The Dumpster Battle': ['2024', '3.5', ['animation', 'drama']],
'Dune: Part Two': ['2024', '4', ['adventure']],
'Past Lives': ['2023', '4', ['drama', 'romance']],
'Dune': ['2021', '4', ['adventure']],
'Shake, Rattle &amp;amp; Roll Extreme': ['2023', '3', ['horror']],
'Feast': ['2022', '2', ['crime', 'mystery']],
'Tokyo Story': ['1953', '4.5', ['drama']],
'Mean Girls': ['2024', '3.5', ['comedy']],
'Orpheus': ['1950', '3.5', ['drama', 'fantasy', 'romance']],
'The Holdovers': ['2023', '4', ['comedy', 'drama']],
'Sing 2': ['2021', '3', ['animation', 'comedy', 'family', 'music']],
```

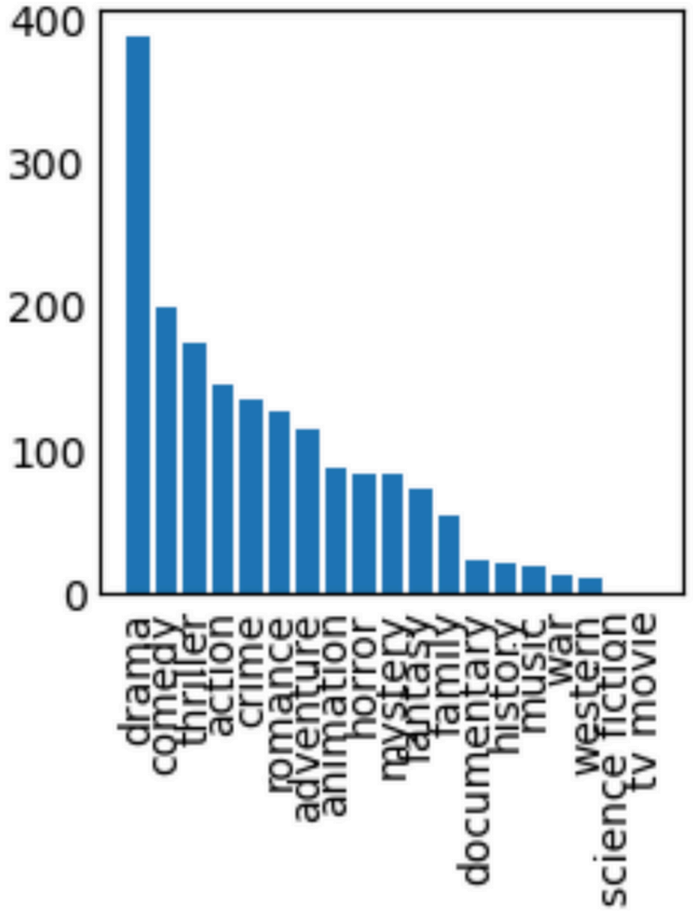
Exploratory Data Analysis

2

TOP RATED GENRES

Genres commonly present in the user's diary page.



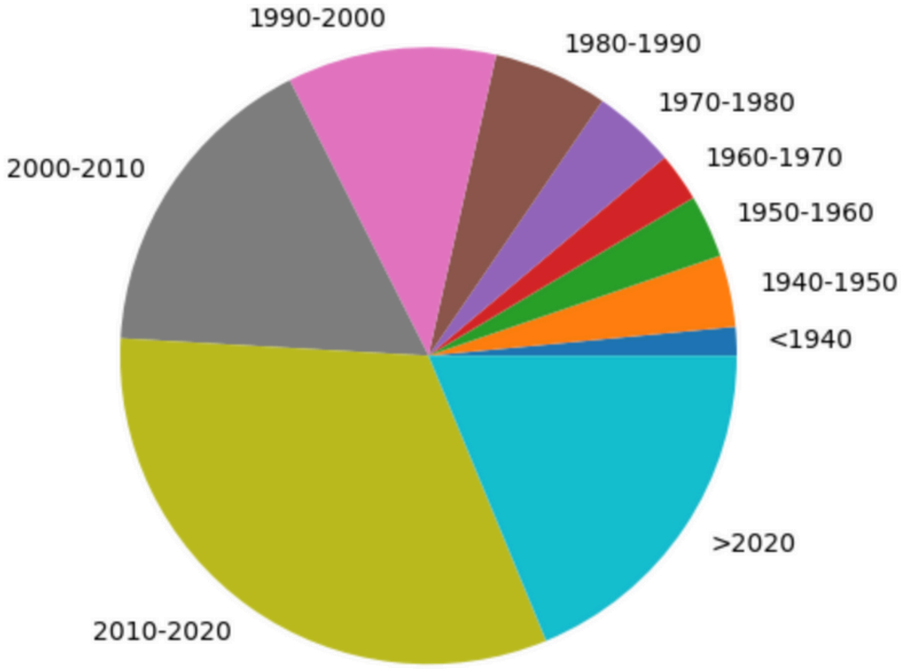


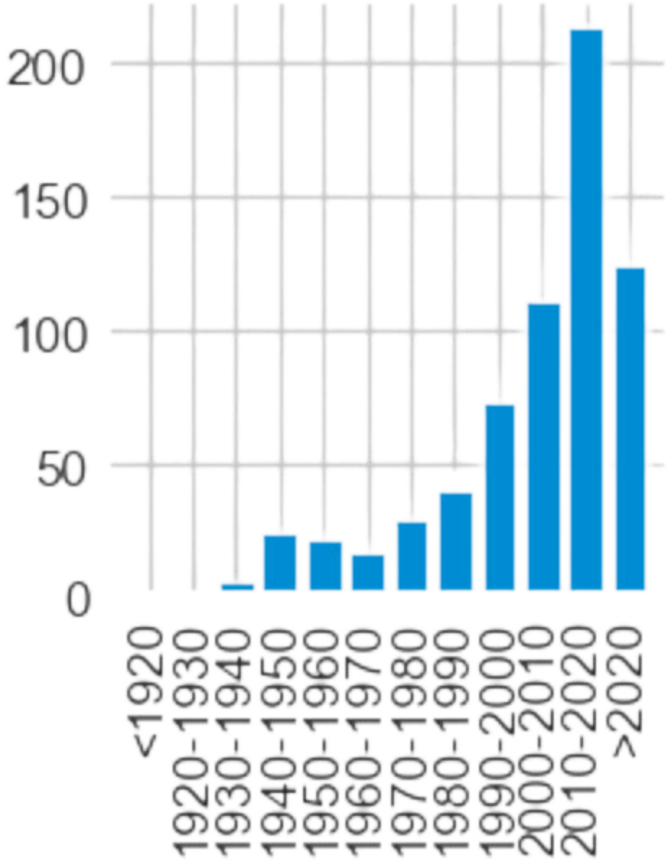
Exploratory Data Analysis

3

RELEASE YEAR OF FILMS

Films reviewed by the user falling under a certain time period.





Reflection and Iteration

Limitations

- lack of knowledge about better approach to shorten the run-time
- Doesn't take into account casts and directors when filtering data

Exploration

- Expand the scope to include other users for better analysis and recommendation
- Use machine learning to better find similarities and give recommendation
- More flexibility in preferences if the user would like to add inputs

Back to Overview

Thank you for listening!