MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

Project Report

on

Emotion Based Movie Recommendation System

Submitted By:

Sanskriti Jha

0901CS191108

Faculty Mentor:

Mr. Mir Shahnawaz Ahmad

Assistant Professor, CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE
GWALIOR - 474005 (MP) est. 1957

MAY-JUNE 2022

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

Project Report

on

Emotion Based Movie Recommendation System

A project report submitted in partial fulfilment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Sanskriti Jha

0901CS191108

Faculty Mentor:

Mr. Mir Shahnawaz Ahmad

Assistant Professor, CSE

Submitted to:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE
GWALIOR - 474005 (MP) est. 1957

MAY-JUNE 2022

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

CERTIFICATE

This is certified that **Sanskriti Jha** (0901CS191108) has submitted the project report titled Mood Based Movie Recommendation System under the mentorship of Prof. Anjula Mehto and Prof. Khushboo Agarwal, in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Computer Science and Engineering from Madhav Institute of Technology and Science, Gwalior.

09/05/22

Mr. Mir Shahnawaz Ahmad
Faculty Mentor
Computer Science & Engineering

09/05/22

Dr. Manish Dixit

Professor and Head,
Computer Science and Engineering

Dr. Manish Dixit
Professor & HOD
Department of CSE
M.I.T.S. Gwalior

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

DECLARATION

I hereby declare that the work being presented in this project report, for the partial fulfilment of requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of Mr. Mir Shahnawaz & Mrs. Khushboo Agarwal, Faculty Mentor, Computer Science & Engineering.

I declare that I have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.

Sanskriti Jha
0901CS191108
III Year, VI Semester
Computer Science and Engineering

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. I am thankful to my institute, **Madhav Institute of Technology and Science** to allow me to continue my disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute. I extend my gratitude to the Director of the institute, **Dr. R. K. Pandit** and Dean Academics, **Dr. Manjaree Pandit** for this.

I would sincerely like to thank my department, **Department of Computer Science and Engineering, for allowing** me to explore this project. I humbly thank **Dr. Manish Dixit**, Professor and Head, Department of Computer Science and Engineering, for his continued support during the course of this engagement, which eased the process and formalities involved.

I am sincerely thankful to my faculty mentors. I am grateful to the guidance of Prof. Anjula Mehto and Prof. Khushboo Agarwal, Faculty Mentor, Computer Science & Engineering, for their continuous support and guidance throughout the project. I am also very thankful to the faculty and staff of the department.

Sanskriti Jha
0901CS191108
III Year, VI Semester
Computer Science and Engineering

ABSTRACT

User generated contents like reviews and comments contain both the information about a given product and also the opinions asserted by the user. With the surge in internet usage, there is a cascade of user generated data such a reviews and comments. People share their experiences, opinions, sentiments and emotions by writing reviews and comments for products they purchase online or after watching a movie, reading books etc. These user generated data contain emotion lexicons such as happiness, sadness, and surprise. Analysis of such emotion can provide a new aspect for recommending new items based on their emotional preferences. In this work, we extract the emotions from this user generated data using the lexical ontology, Python, ML and information from the domain of psychology. These extracted emotions can be used for recommendations. Evaluation on emotion prediction further verifies the effectiveness of the proposed model in comparison to traditional rating-based item similarity model. We further compare this with fuzziness in emotion features.

सार

उपयोगकर्ता द्वारा तैयार की गई सामग्री जैसे समीक्षाओं और टिप्पणियों में किसी दिए गए उत्पाद के बारे में जानकारी और उपयोगकर्ता द्वारा दी गई राय दोनों शामिल हैं। इंटरनेट के उपयोग में वृद्धि के साथ, उपयोगकर्ता द्वारा तैयार किए गए डेटा जैसे समीक्षाओं और टिप्पणियों का एक झरना है। लोग अपने अनुभव, राय, भावनाओं और भावनाओं को उन उत्पादों के लिए समीक्षा और टिप्पणियां लिखकर साझा करते हैं जो वे ऑनलाइन खरीदते हैं या एक फिल्म देखने, किताबें पढ़ने आदि के बाद। इन उपयोगकर्ता उत्पन्न डेटा में खुशी, उदासी और आश्चर्य जैसे भावनात्मक शब्दकोष शामिल हैं। इस तरह की भावनाओं का विश्लेषण उनकी भावनात्मक प्राथमिकताओं के आधार पर नई वस्तुओं की सिफारिश करने के लिए एक नया पहलू प्रदान कर सकता है। इस काम में, हम लेक्सिकल ऑन्कोलॉजी, पायथन, एमएल और मनोविज्ञान के क्षेत्र से जानकारी का उपयोग करके इस उपयोगकर्ता द्वारा उत्पन्न डेटा से भावनाओं को निकालते हैं। इन निकाली गई भावनाओं का उपयोग सिफारिशों के लिए किया जा सकता है। भावनात्मक भविष्यवाणी पर मूल्यांकन पारंपरिक रेटिंग-आधारित आइटम समानता मॉडल की तुलना में प्रस्तावित मॉडल की प्रभावशीलता की पुष्टि करता है। हम आगे इसकी तुलना इमोशनल फीचर्स में फजीनेस से करते हैं।

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER – 1: PROJECT OVERVIEW

1.1 **Introduction**: This is a Python - Emotion based movie recommendation system that implemented text-retrieval techniques and Graphical User Interface.

1.2 **Objective**: The objective of E- MRS is to provide adapted and personalized suggestions to users using a combination of collaborative filtering and content-based techniques. The recommendation is based on inferences about a user's emotions and preferences, as well as opinions of other similar users.

1.3 **Scope**: Recommender systems help E-commerce sites to increase their sales. A movie recommendation system named E-MRS, based on collaborative filtering approach makes use of the information provided by users, analyzes them and then recommends the movie that is best suited to the user at that time using k-means algorithm.

1.4 **Project Features**:

- Literature Review.
- Design of process.
- Source code separately.
- Outcome of source code.
- Final working and output.

1.5 **System Requirement**:

- Windows 7/8/10/11.
- Internet Conectivity.
- AMD 64-bit Processor.
- Software – Python 3.10.4
- 8 GB RAM, 512 GB ROM.

CHAPTER – 2: LITERATURE REVIEW

2.1 Knowledge Required:

1. A recommendation system is a subclass of Information filtering Systemsthat seeks to predict the rating or the preference a user might give to anitem. In simple words, it is an algorithm that suggests relevant items to users.

2. This is a Python-based movie recommendation system that implemented text-retrieval techniques and Graphical User Interface.

3. One special thing about this system is that its recommendations were tailored around users' emotion of the moment.

4. The main objective of this project is to fill this gap by making traditional recommender system more user-driven.

5. One of the underlying targets of movies is to evoke emotions in their viewers. IMDb offers all the movies for all genre. Therefore, the movie titles can be scraped from the IMDB list to recommend to the user.

6. IMDB does not have an API, for accessing information on movies and TV Series. Therefore, we have to perform scraping. Scraping is used for accessing information from a website which is usually done with APIs.

7. This project would have – Choice of emotions, list of movies according to the emotion(s), any movie's detailed info, movie's summary and similar movie recommendation according to the output movie.

8. Tools required – Python, ML, Jupyter, VS Code.

9. Libraries required – Numpy, Pandas, Scikit-Learn.

2.2 Competitor Analysis:

Table 2.2.1

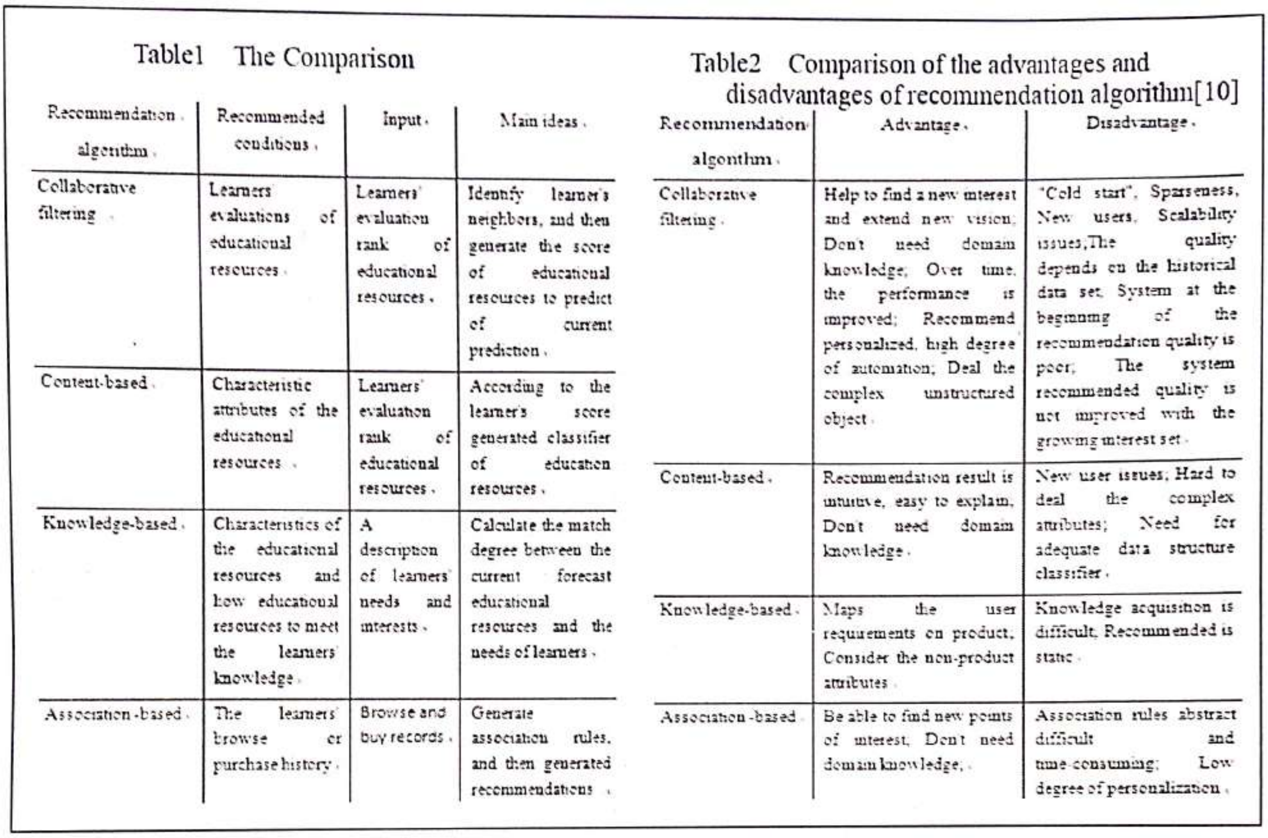


Table1 The Comparison

Recommendation algorithm	Recommended conditions	Input	Main ideas
Collaborative filtering	Learners' evaluations of educational resources	Learners' evaluation rank of educational resources	Identify learner's neighbors, and then generate the score of educational resources to predict of current prediction
Content-based	Characteristic attributes of the educational resources	Learners' evaluation rank of educational resources	According to the learner's score generated classifier of education resources
Knowledge-based	Characteristics of the educational resources and how educational resources to meet the learners' knowledge	A description of learners' needs and interests	Calculate the match degree between the current forecast educational resources and the needs of learners
Association-based	The learners' browse or purchase history	Browse and buy records	Generate association rules, and then generated recommendations

Table2 Comparison of the advantages and disadvantages of recommendation algorithm[10]

Recommendation algorithm	Advantage	Disadvantage
Collaborative filtering	Help to find a new interest and extend new vision; Don't need domain knowledge; Over time, the performance is improved; Recommend personalized, high degree of automation; Deal the complex unstructured object	"Cold start", Sparseness, New users, Scalability issues;The quality depends on the historical data set. System at the beginning of the recommendation quality is poor; The system recommended quality is not improved with the growing interest set
Content-based	Recommendation result is intuitive, easy to explain, Don't need domain knowledge	New user issues; Hard to deal the complex attributes; Need for adequate data structure classifier
Knowledge-based	Maps the user requirements on product, Consider the non-product attributes	Knowledge acquisition is difficult, Recommended is static
Association-based	Be able to find new points of interest; Don't need domain knowledge;	Association rules abstract difficult and time-consuming; Low degree of personalization

CHAPTER – 3 (PRELIMINARY DESIGN)

3.1 Flow Charts:

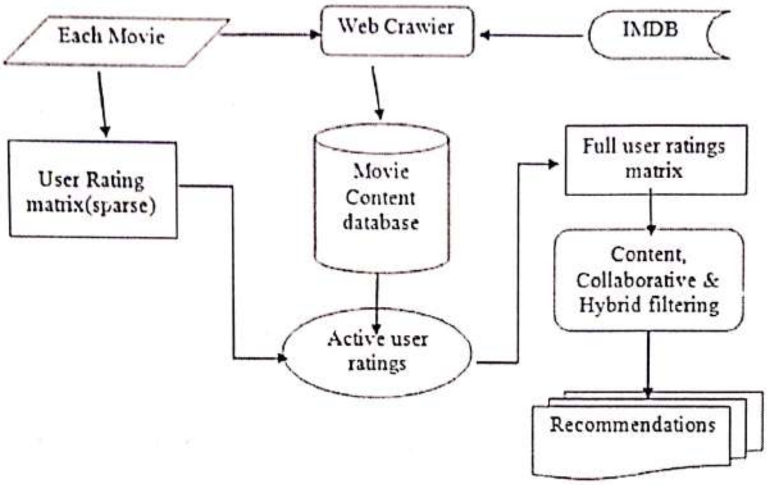


Fig 3.1.1 Work flow

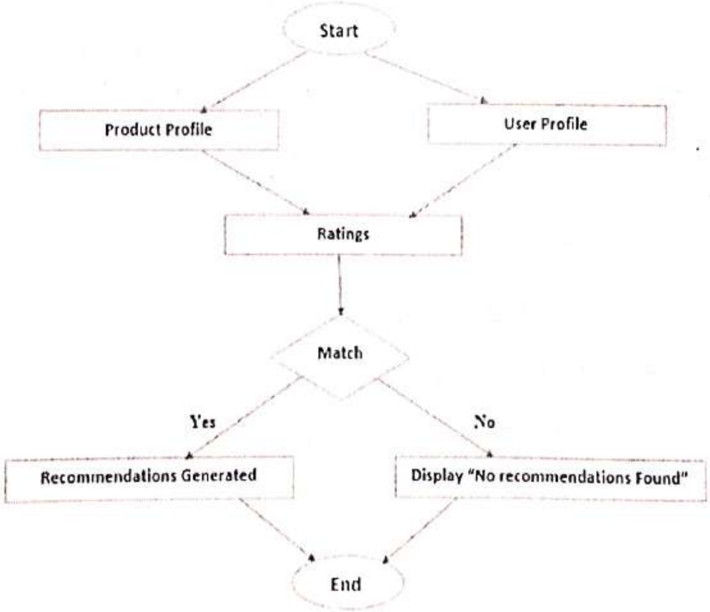


Fig 3.1.2 DFD Diagram

3.2 Emotion Description:

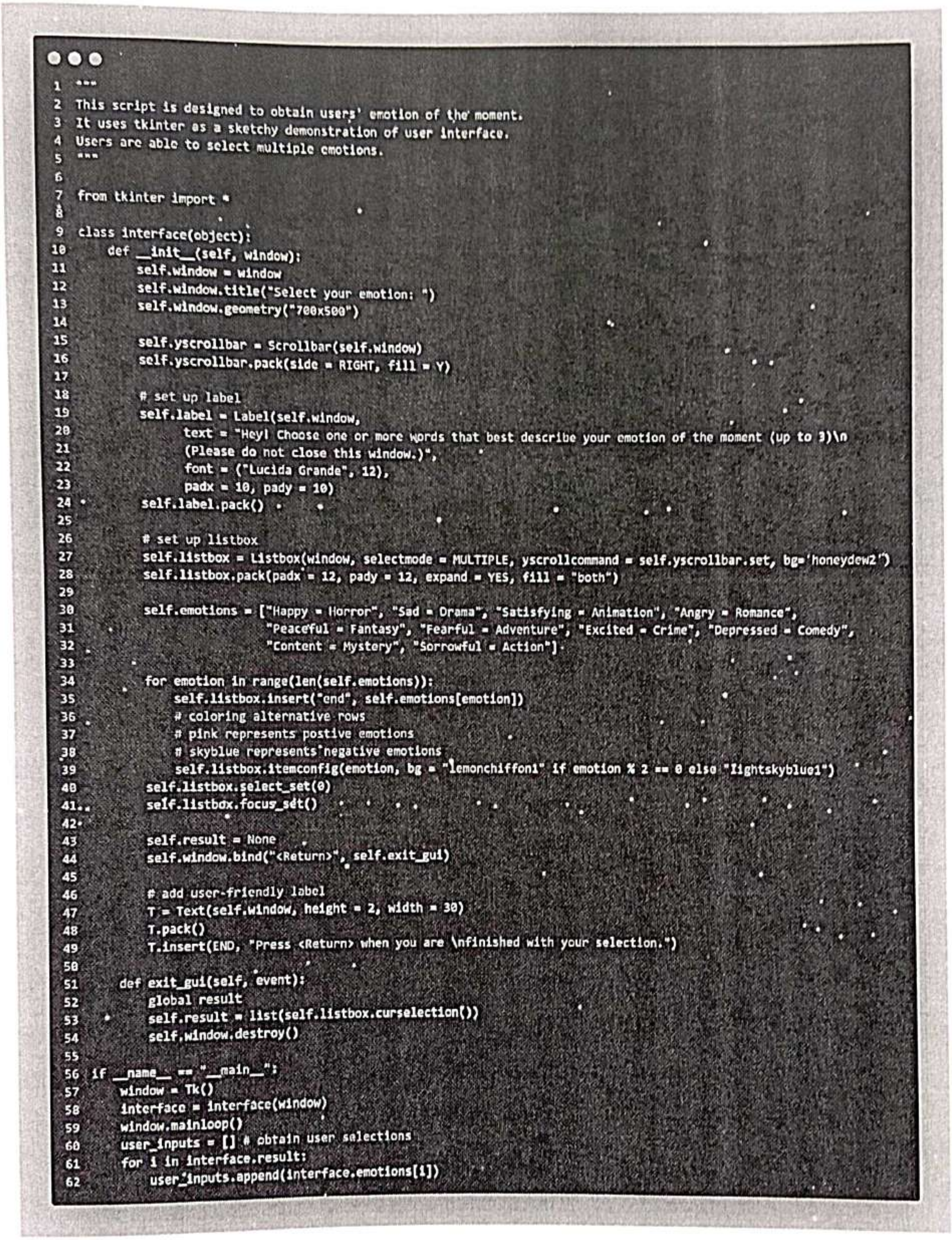
There are 10 categories of emotion the system presented to users to choose from. These are 5 positive emotions ("Happy", "Satisfied", "Peaceful", "Excited", "Content") and 5 negative emotions ("Sad", "Angry", "Fearful","Depressed", "Sorrowful").

The correspondence of every emotion with genre of movies are set up as below:

- Happy – Horror
- Sad – Drama
- Satisfied – Animation
- Angry – Romance
- Peaceful – Fantasy
- Fearful – Adventure
- Excited – Crime
- Depressed – Comedy
- Content – Mystery
- Sorrowful – Action

Based on the inputted emotion, the system is going to be selected from the corresponding genre based on their ratings given by two websites: IMDB and Rotten Tomatoes. The reason why we are collecting movie information from both websites is that we believe the system is able to capture a more full-scaled opinions from movie lovers.

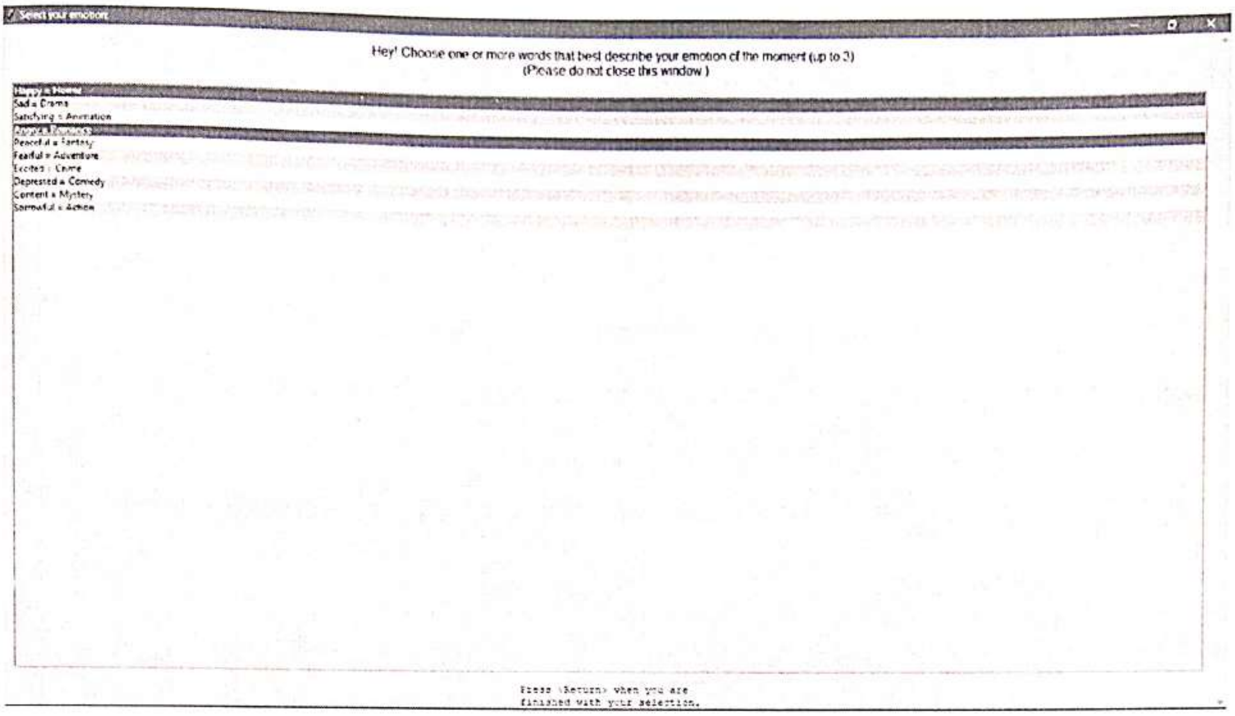
3.3 GUI Code:



```
"""
This script is designed to obtain users' emotion of the moment.
It uses tkinter as a sketchy demonstration of user interface.
Users are able to select multiple emotions.
"""


from tkinter import *

class interface(object):
    def __init__(self, window):
        self.window = window
        self.window.title("Select your emotion: ")
        self.window.geometry("700x500")

        self.yscrollbar = Scrollbar(self.window)
        self.yscrollbar.pack(side = RIGHT, fill = Y)

        # set up label
        self.label = Label(self.window,
                text = "Hey! Choose one or more words that best describe your emotion of the moment (up to 3)\n
                (Please do not close this window.)",
                font = ("Lucida Grande", 12),
                padx = 10, pady = 10)
        self.label.pack()

        # set up listbox
        self.listbox = Listbox(window, selectmode = MULTIPLE, yscrollcommand = self.yscrollbar.set, bg='honeydew2')
        self.listbox.pack(padx = 12, pady = 12, expand = YES, fill = "both")

        self.emotions = ["Happy - Horror", "Sad - Drama", "Satisfying - Animation", "Angry - Romance",
                         "Peaceful - Fantasy", "Fearful - Adventure", "Excited - Crime", "Depressed - Comedy",
                         "Content - Mystery", "Sorrowful - Action"]

        for emotion in range(len(self.emotions)):
            self.listbox.insert("end", self.emotions[emotion])
            # coloring alternative rows
            # pink represents postive emotions
            # skyblue represents negative emotions
            self.listbox.itemconfig(emotion, bg = "lemonchiffon1" if emotion % 2 == 0 else "lightskyblue1")
        self.listbox.select_set(0)
        self.listbox.focus_set()

        self.result = None
        self.window.bind("<Return>", self.exit_gui)

        # add user-friendly label
        T = Text(self.window, height = 2, width = 30)
        T.pack()
        T.insert(END, "Press <Return> when you are \nfinished with your selection.")

    def exit_gui(self, event):
        global result
        self.result = list(self.listbox.curselection())
        self.window.destroy()

if __name__ == "__main__":
    window = Tk()
    interface = interface(window)
    window.mainloop()
    user_inputs = [] # obtain user selections
    for i in interface.result:
        user_inputs.append(interface.emotions[i])
```

Fig 3.3.1 GUI Code

3.4 GUI Output:



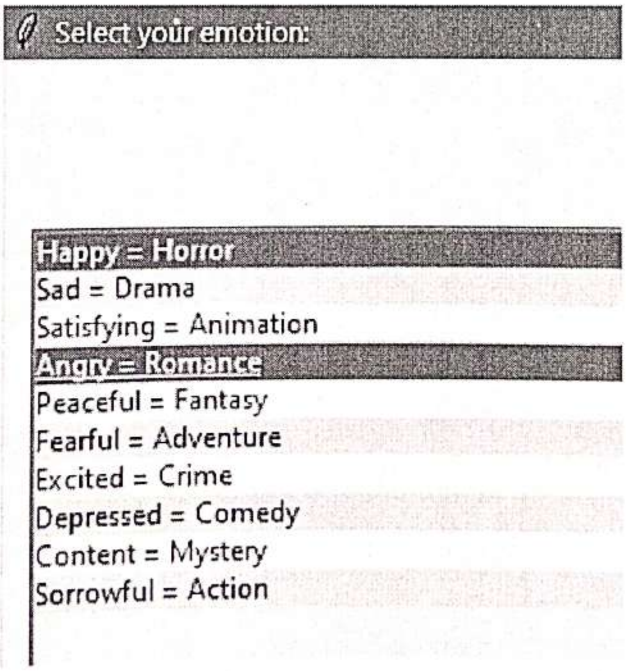
Select your emotion:

Hey! Choose one or more words that best describe your emotion of the moment (up to 3)
(Please do not close this window)

Happy = Horror
Sad = Drama
Satisfying = Animation
Angry = Romance
Peaceful = Fantasy
Fearful = Adventure
Excited = Crime
Depressed = Comedy
Content = Mystery
Sorrowful = Action

Press <Return> when you are
finished with your selection.

Fig 3.4.1 GUI Output (1)



Select your emotion:

Happy = Horror
Sad = Drama
Satisfying = Animation
Angry = Romance
Peaceful = Fantasy
Fearful = Adventure
Excited = Crime
Depressed = Comedy
Content = Mystery
Sorrowful = Action

Fig 3.4.2 GUI Output (2)

3.5 Crawling and Scraping:

A Web crawler, sometimes called a spider or spider-bot and often shortened to crawler, is an Internet bot that systematically browses the World Wide Web and that is typically operated by search engines for the purpose of Web indexing (*web spidering*).

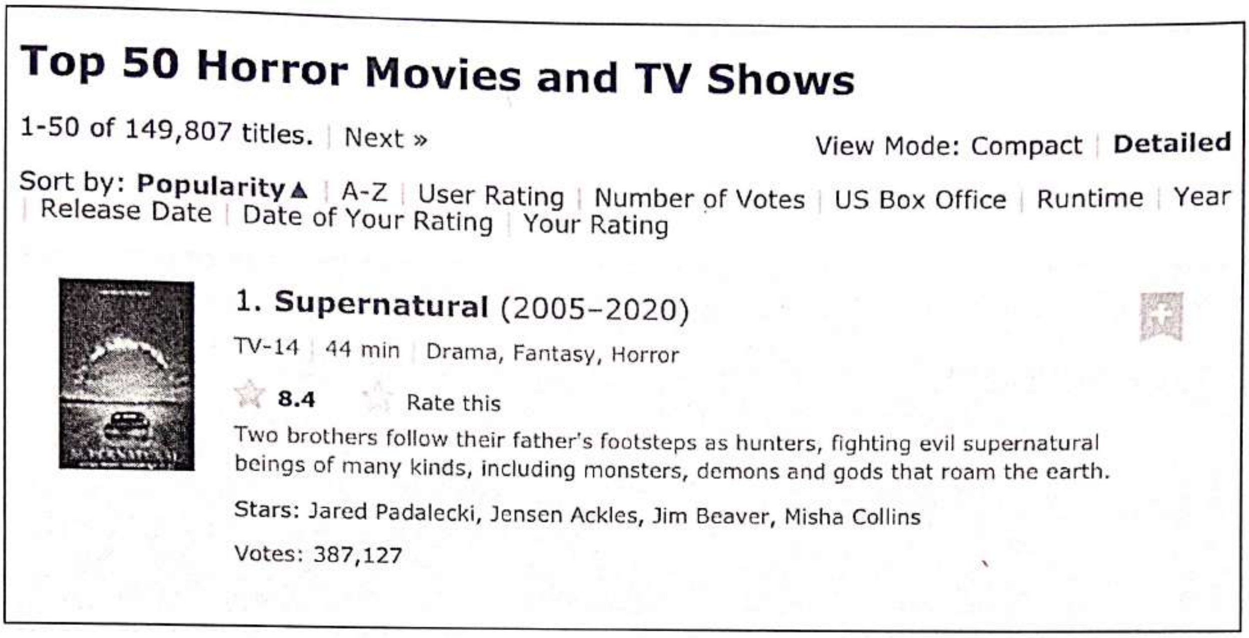
It can also be understood as a computer program that automatically and systematically searches web pages for certain keywords Each search engine has its own proprietary computation (called an "algorithm") that ranks websites for each keyword or combination of keywords.

Web scraping is about extracting the data from one or more websites. While crawling is about finding or discovering URLs or links on the web. Usually, in web data extraction projects, you need to combine crawling and scraping.

Because we intend to scrape two websites with different web structure, we developed one IMDB crawler and another RT crawler to extract movie information. Check out scraper.py for more details.

As you can see, comparing to IMDB, Rotten Tomatoes includes the majority ofmovie information in each movie profile link. Our crawler had to look up each link to capture hidden information, such as movie length, maturity grading, cast, etc. The user or rating matrix is very sparse. It is very hard to find users that have rated the same items because most of the user does not rate the items. So it becomes hard to find set of users who rate the items. Therefore, it is unavoidable that the program takes more time to scrape RT pages.

Here are two example movie pages of IMDB and Rotten Tomatoes:



Top 50 Horror Movies and TV Shows

1-50 of 149,807 titles. | Next »

View Mode: Compact | **Detailed**

Sort by: **Popularity▲** | A-Z | User Rating | Number of Votes | US Box Office | Runtime | Year | Release Date | Date of Your Rating | Your Rating

1. **Supernatural** (2005–2020)

TV-14 | 44 min | Drama, Fantasy, Horror

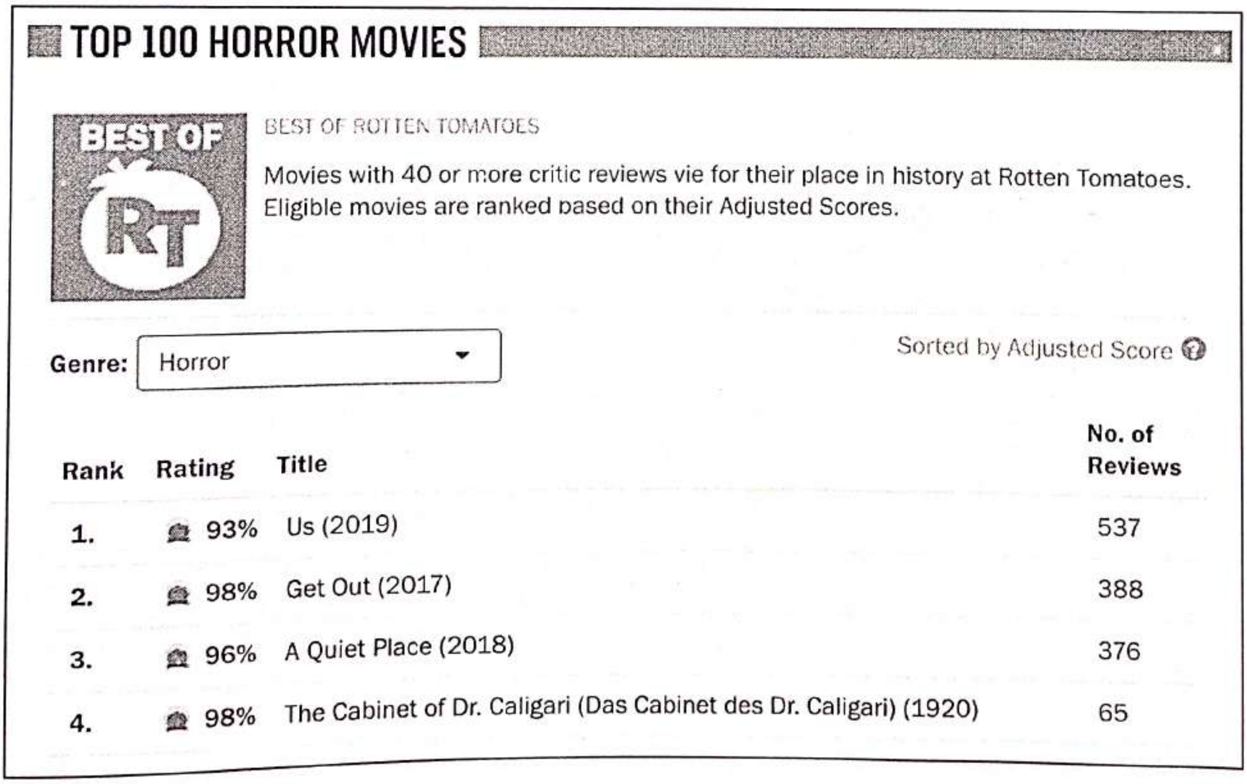
☆ **8.4** ☆ Rate this

Two brothers follow their father's footsteps as hunters, fighting evil supernatural beings of many kinds, including monsters, demons and gods that roam the earth.

Stars: Jared Padalecki, Jensen Ackles, Jim Beaver, Misha Collins

Votes: 387,127

Fig 3.5.1 IMDB Review



TOP 100 HORROR MOVIES

BEST OF ROTTEN TOMATOES

Movies with 40 or more critic reviews vie for their place in history at Rotten Tomatoes. Eligible movies are ranked based on their Adjusted Scores.

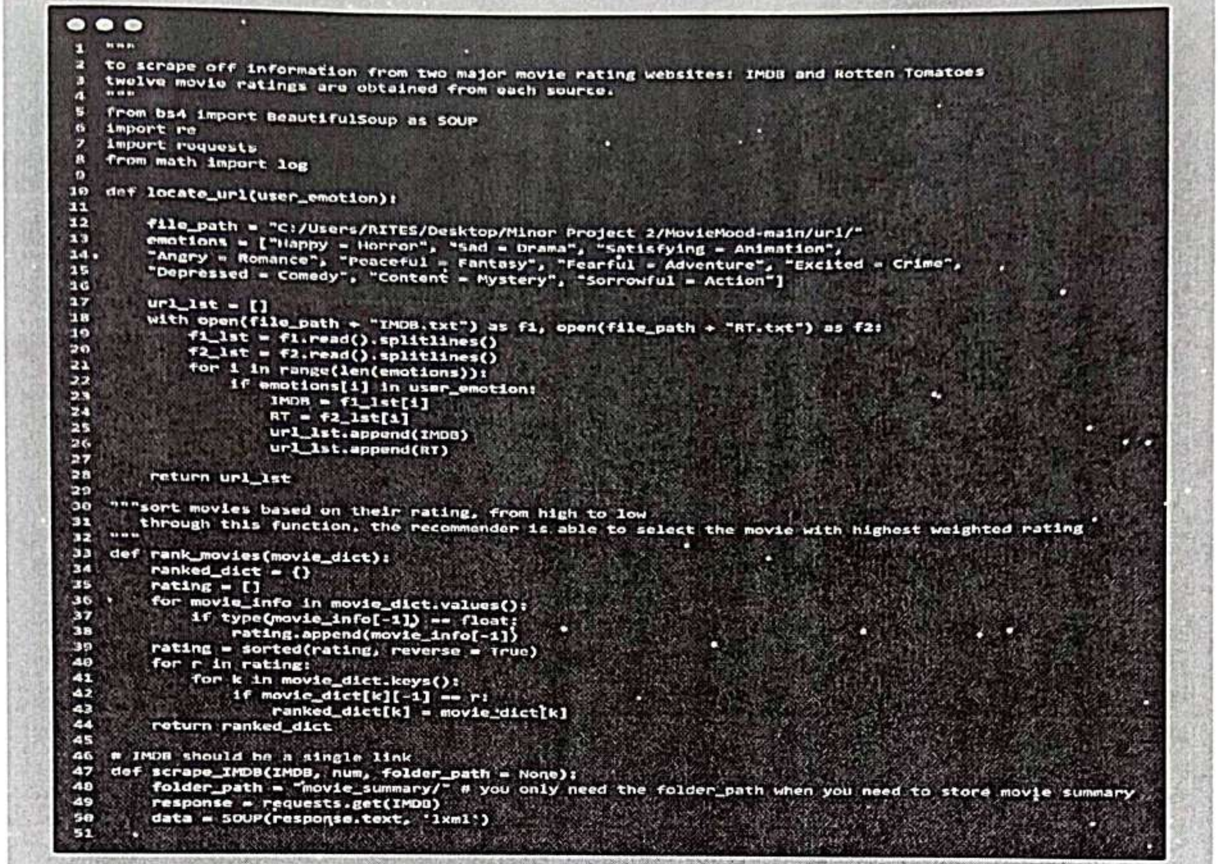
Genre: Horror ▾

Sorted by Adjusted Score ?

Rank	Rating	Title	No. of Reviews
1.	93%	Us (2019)	537
2.	98%	Get Out (2017)	388
3.	96%	A Quiet Place (2018)	376
4.	98%	The Cabinet of Dr. Caligari (Das Cabinet des Dr. Caligari) (1920)	65

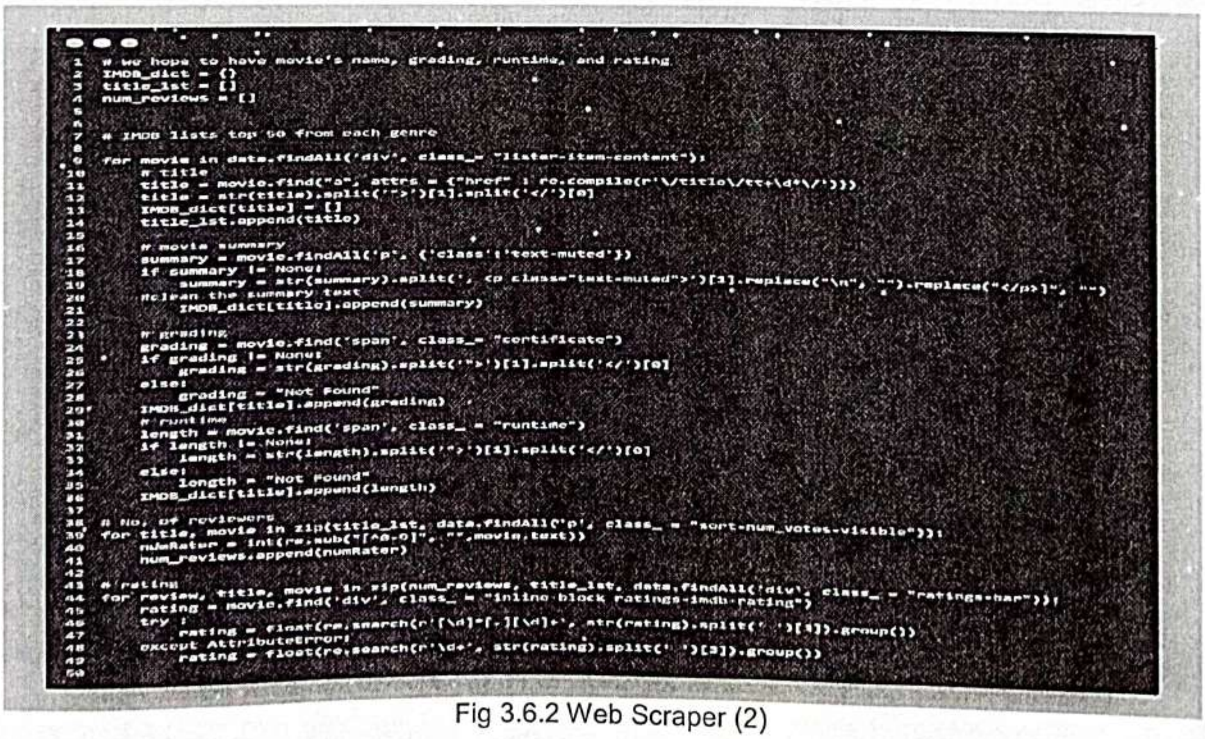
Fig 3.5.2 Rotten Tomatoes

3.6 Web Scraper Code:



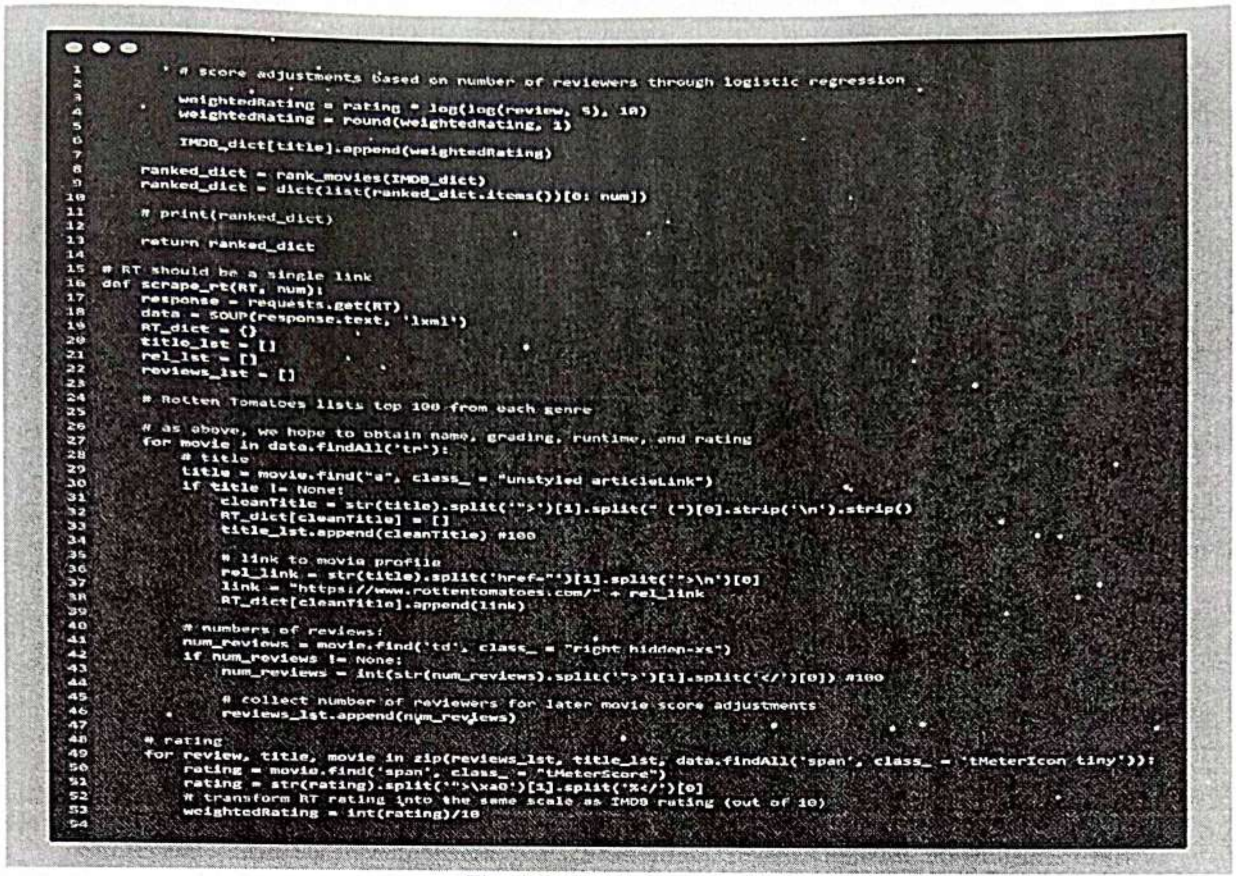
```python
"""
to scrape off information from two major movie rating websites: IMDB and Rotten Tomatoes
twelve movie ratings are obtained from each source.
"""
from bs4 import BeautifulSoup as SOUP
import re
import requests
from math import log

def locate_url(user_emotion):

    file_path = "C:/Users/RITES/Desktop/Minor Project 2/MovieMood-main/url/"
    emotions = ["Happy = Horror", "Sad = Drama", "Satisfying = Animation",
    "Angry = Romance", "Peaceful = Fantasy", "Fearful = Adventure", "Excited = Crime",
    "Depressed = Comedy", "Content = Mystery", "Sorrowful = Action"]

    url_lst = []
    with open(file_path + "IMDB.txt") as f1, open(file_path + "RT.txt") as f2:
        f1_lst = f1.read().splitlines()
        f2_lst = f2.read().splitlines()
        for i in range(len(emotions)):
            if emotions[i] in user_emotion:
                IMDB = f1_lst[i]
                RT = f2_lst[i]
                url_lst.append(IMDB)
                url_lst.append(RT)

    return url_lst

"""sort movies based on their rating, from high to low
    through this function, the recommender is able to select the movie with highest weighted rating
"""
def rank_movies(movie_dict):
    ranked_dict = {}
    rating = []
    for movie_info in movie_dict.values():
        if type(movie_info[-1]) == float:
            rating.append(movie_info[-1])
    rating = sorted(rating, reverse = True)
    for r in rating:
        for k in movie_dict.keys():
            if movie_dict[k][-1] == r:
                ranked_dict[k] = movie_dict[k]
    return ranked_dict

# IMDB should be a single link
def scrape_IMDB(IMDB, num, folder_path = None):
    folder_path = "movie_summary/" # you only need the folder_path when you need to store movie summary
    response = requests.get(IMDB)
    data = SOUP(response.text, 'lxml')

```

Fig 3.6.1 Web Scraper (1)



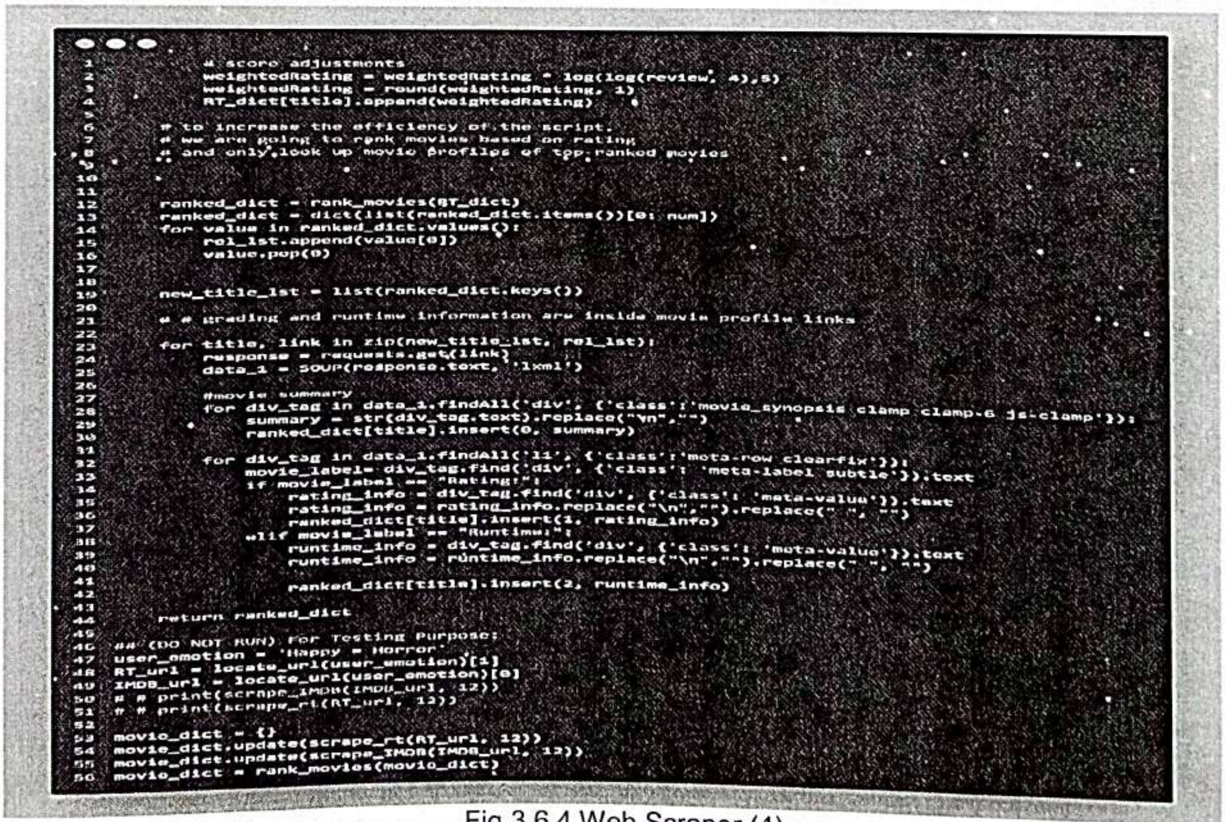
```python
# we hope to have movie's name, grading, runtime, and rating
IMDB_dict = {}
title_lst = []
num_reviews = []


# IMDB lists top 50 from each genre

for movie in data.findAll('div', class_= "lister-item-content"):
    # title
    title = movie.find("a", attrs = {"href" : re.compile(r'\/title\/tt+\d*\/')})
    title = str(title).split('">')[1].split('</')[0]
    IMDB_dict[title] = []
    title_lst.append(title)

    # movie summary
    summary = movie.findAll('p', {'class':'text-muted'})
    if summary != None:
        summary = str(summary).split(', <p class="text-muted">')[1].replace("\n", "").replace("</p>]", "")
    #clean the summary text
        IMDB_dict[title].append(summary)

    # grading
    grading = movie.find('span', class_= "certificate")
    if grading != None:
        grading = str(grading).split('">')[1].split('</')[0]
    else:
        grading = "Not Found"
    IMDB_dict[title].append(grading)
    # runtime
    length = movie.find('span', class_ = "runtime")
    if length != None:
        length = str(length).split('">')[1].split('</')[0]
    else:
        length = "Not Found"
    IMDB_dict[title].append(length)

# No. of reviewers
for title, movie in zip(title_lst, data.findAll('p', class_ = "sort-num_votes-visible")):
    numRater = int(re.sub("[^0-9]", "",movie.text))
    num_reviews.append(numRater)

# rating
for review, title, movie in zip(num_reviews, title_lst, data.findAll('div', class_ = "ratings-bar")):
    rating = movie.find('div', class_ = "inline-block ratings-imdb-rating")
    try :
        rating = float(re.search(r'[\d]*[.][\d]+', str(rating).split(' ')[3]).group())
    except AttributeError:
        rating = float(re.search(r'\d+', str(rating).split(' ')[3]).group())

```

Fig 3.6.2 Web Scraper (2)



```
        # score adjustments based on number of reviewers through logistic regression

        weightedRating = rating * log(log(review, 5), 10)
        weightedRating = round(weightedRating, 1)

        IMDB_dict[title].append(weightedRating)

    ranked_dict = rank_movies(IMDB_dict)
    ranked_dict = dict(list(ranked_dict.items())[0: num])

    # print(ranked_dict)

    return ranked_dict

# RT should be a single link
def scrape_rt(RT, num):
    response = requests.get(RT)
    data = SOUP(response.text, 'lxml')
    RT_dict = {}
    title_lst = []
    rel_lst = []
    reviews_lst = []

    # Rotten Tomatoes lists top 100 from each genre

    # as above, we hope to obtain name, grading, runtime, and rating
    for movie in data.findAll('tr'):
        # title
        title = movie.find("a", class_ = "unstyled articleLink")
        if title != None:
            cleanTitle = str(title).split('">')[1].split(" (")[0].strip('\n').strip()
            RT_dict[cleanTitle] = []
            title_lst.append(cleanTitle) #100

            # link to movie profile
            rel_link = str(title).split('href="')[1].split('">\n')[0]
            link = "https://www.rottentomatoes.com/" + rel_link
            RT_dict[cleanTitle].append(link)

        # numbers of reviews:
        num_reviews = movie.find('td', class_ = "right hidden-xs")
        if num_reviews != None:
            num_reviews = int(str(num_reviews).split('">')[1].split('</')[0]) #100

            # collect number of reviewers for later movie score adjustments
            reviews_lst.append(num_reviews)

    # rating
    for review, title, movie in zip(reviews_lst, title_lst, data.findAll('span', class_ = 'tMeterIcon tiny')):
        rating = movie.find('span', class_ = "tMeterScore")
        rating = str(rating).split('">\xa0')[1].split('%</')[0]
        # transform RT rating into the same scale as IMDB rating (out of 10)
        weightedRating = int(rating)/10
```

Fig 3.6.3 Web Scraper (3)



```
        # score adjustments
        weightedRating = weightedRating * log(log(review, 4),5)
        weightedRating = round(weightedRating, 1)
        RT_dict[title].append(weightedRating)

    # to increase the efficiency of the script,
    # we are going to rank movies based on rating
    # and only look up movie profiles of top ranked movies


    ranked_dict = rank_movies(RT_dict)
    ranked_dict = dict(list(ranked_dict.items())[0: num])
    for value in ranked_dict.values():
        rel_lst.append(value[0])
        value.pop(0)


    new_title_lst = list(ranked_dict.keys())

    # # grading and runtime information are inside movie profile links

    for title, link in zip(new_title_lst, rel_lst):
        response = requests.get(link)
        data_1 = SOUP(response.text, 'lxml')

        #movie summary
        for div_tag in data_1.findAll('div', {'class':'movie_synopsis clamp clamp-6 js-clamp'}):
            summary = str(div_tag.text).replace("\n","")
            ranked_dict[title].insert(0, summary)

        for div_tag in data_1.findAll('li', {'class':'meta-row clearfix'}):
            movie_label= div_tag.find('div', {'class': 'meta-label subtle'}).text
            if movie_label == "Rating:":
                rating_info = div_tag.find('div', {'class': 'meta-value'}).text
                rating_info = rating_info.replace("\n","").replace(" ", "")
                ranked_dict[title].insert(1, rating_info)
            elif movie_label == "Runtime:":
                runtime_info = div_tag.find('div', {'class': 'meta-value'}).text
                runtime_info = runtime_info.replace("\n","").replace(" ", "")

                ranked_dict[title].insert(2, runtime_info)

    return ranked_dict

## (DO NOT RUN) For Testing Purpose:
user_emotion = 'Happy = Horror'
RT_url = locate_url(user_emotion)[1]
IMDB_url = locate_url(user_emotion)[0]
# # print(scrape_IMDB(IMDB_url, 12))
# # print(scrape_rt(RT_url, 12))

movie_dict = {}
movie_dict.update(scrape_rt(RT_url, 12))
movie_dict.update(scrape_IMDB(IMDB_url, 12))
movie_dict = rank_movies(movie_dict)
```

Fig 3.6.4 Web Scraper (4)

3.7 Crawling and Scraping:

We would pull user rating scores from both IMDb and Rotten Tomatoes. Due to the different rating scales used by IMDb and Rotten Tomatoes, we would first convert both scores to a 10-point scale for the case of comparison. We would also take the number of ratings into consideration, as larger number of ratings tends to make the overall rating more credible. Therefore, we would runlogistic regression function on the number of ratings, and add it as an additional weightage to the final movie score.

3.7.1 Logistic Regression Function:

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).

3.8 Current Movie Information:

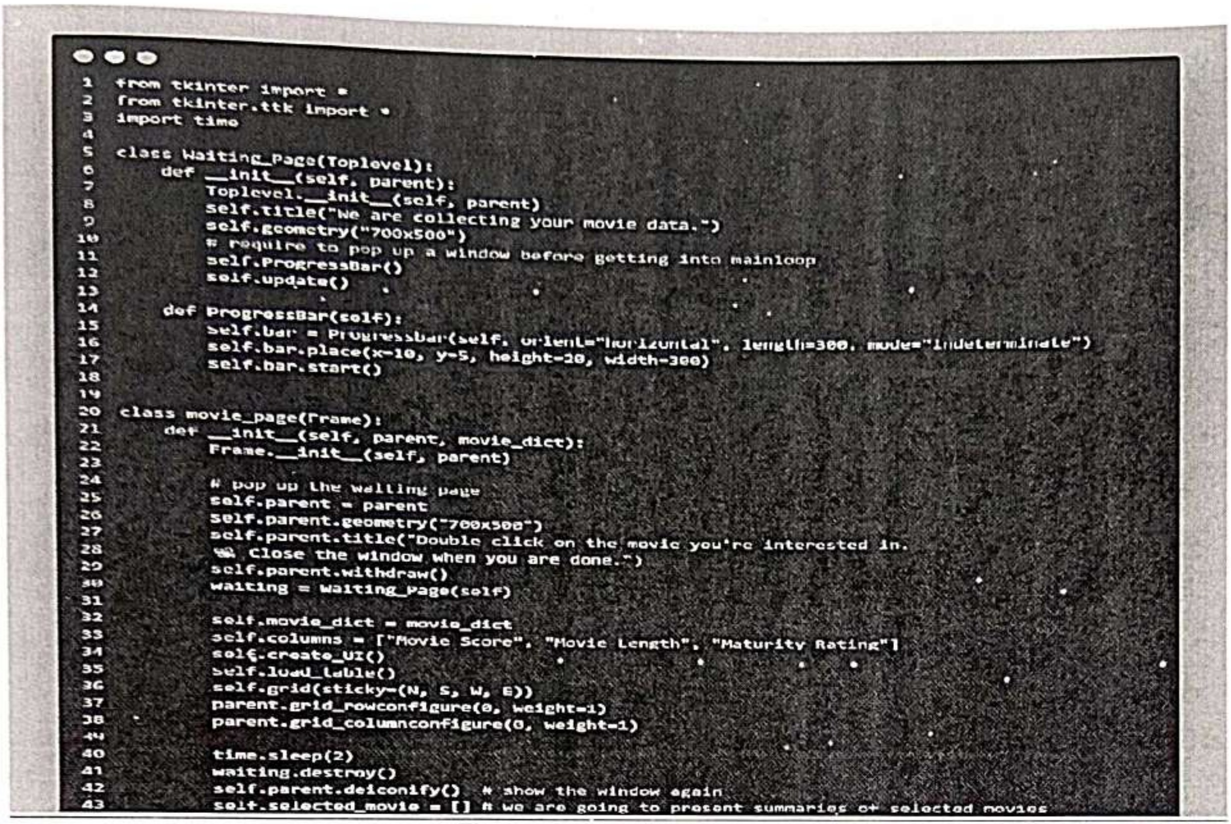
After users indicate their moods, the program is going to look up the corresponding link to the movie page and present movie information asTreeview, which is a module included by the tkinter library displaying ahierarchical collection of items.

Here is an example output of the list of recommended movies:

Note: Not every movie has all information listed. If the crawler cannot find relevant information, it will automatically fill the space with "Not Found".

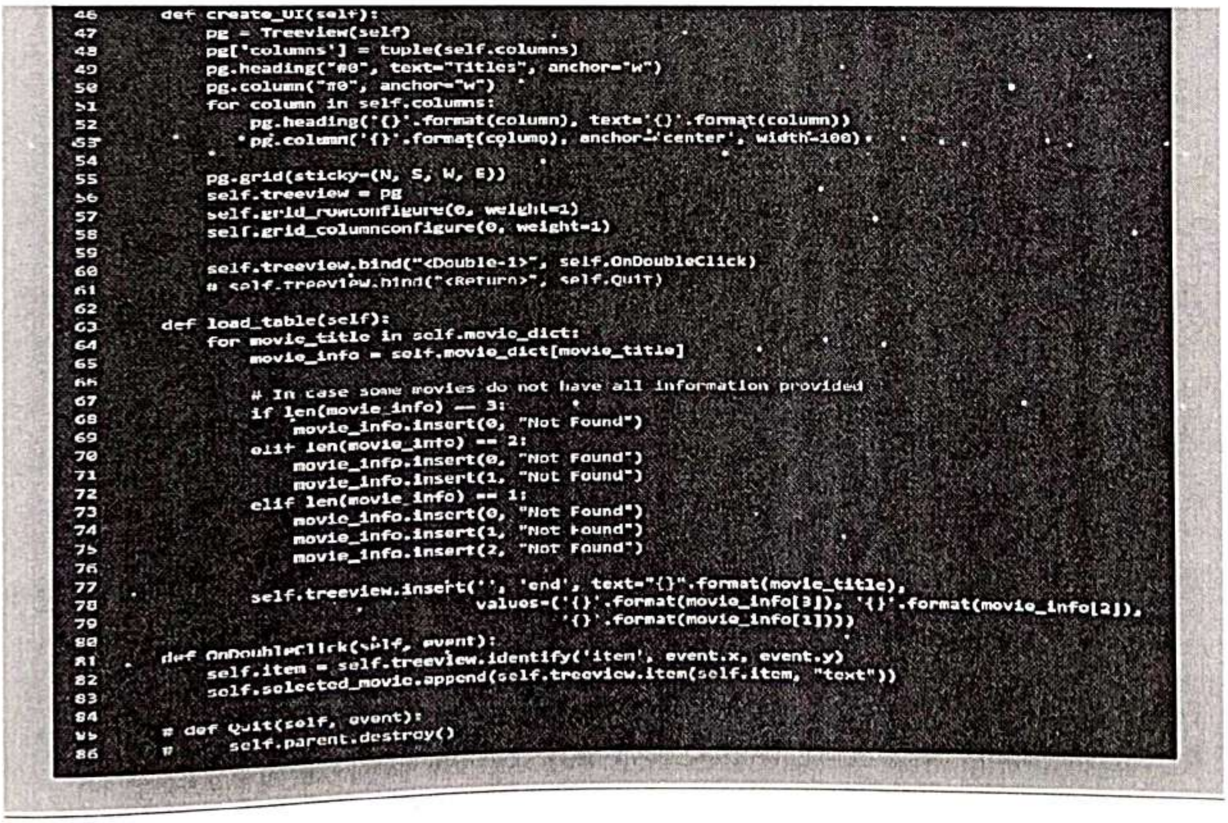
Otherwise movies with all details have their titles, rating, movie length and maturity rating also.

3.9 Tree View Source Code:



```python
from tkinter import *
from tkinter.ttk import *
import time

class Waiting_Page(Toplevel):
    def __init__(self, parent):
        Toplevel.__init__(self, parent)
        self.title("We are collecting your movie data.")
        self.geometry("700x500")
        # require to pop up a window before getting into mainloop
        self.ProgressBar()
        self.update()

    def ProgressBar(self):
        self.bar = Progressbar(self, orient="horizontal", length=300, mode="indeterminate")
        self.bar.place(x=10, y=5, height=20, width=300)
        self.bar.start()


class movie_page(Frame):
    def __init__(self, parent, movie_dict):
        Frame.__init__(self, parent)

        # pop up the waiting page
        self.parent = parent
        self.parent.geometry("700x500")
        self.parent.title("Double click on the movie you're interested in.
        Close the window when you are done.")
        self.parent.withdraw()
        waiting = Waiting_Page(self)

        self.movie_dict = movie_dict
        self.columns = ["Movie Score", "Movie Length", "Maturity Rating"]
        self.create_UI()
        self.load_table()
        self.grid(sticky=(N, S, W, E))
        parent.grid_rowconfigure(0, weight=1)
        parent.grid_columnconfigure(0, weight=1)

        time.sleep(2)
        waiting.destroy()
        self.parent.deiconify()  # show the window again
        self.selected_movie = [] # we are going to present summaries of selected movies
```

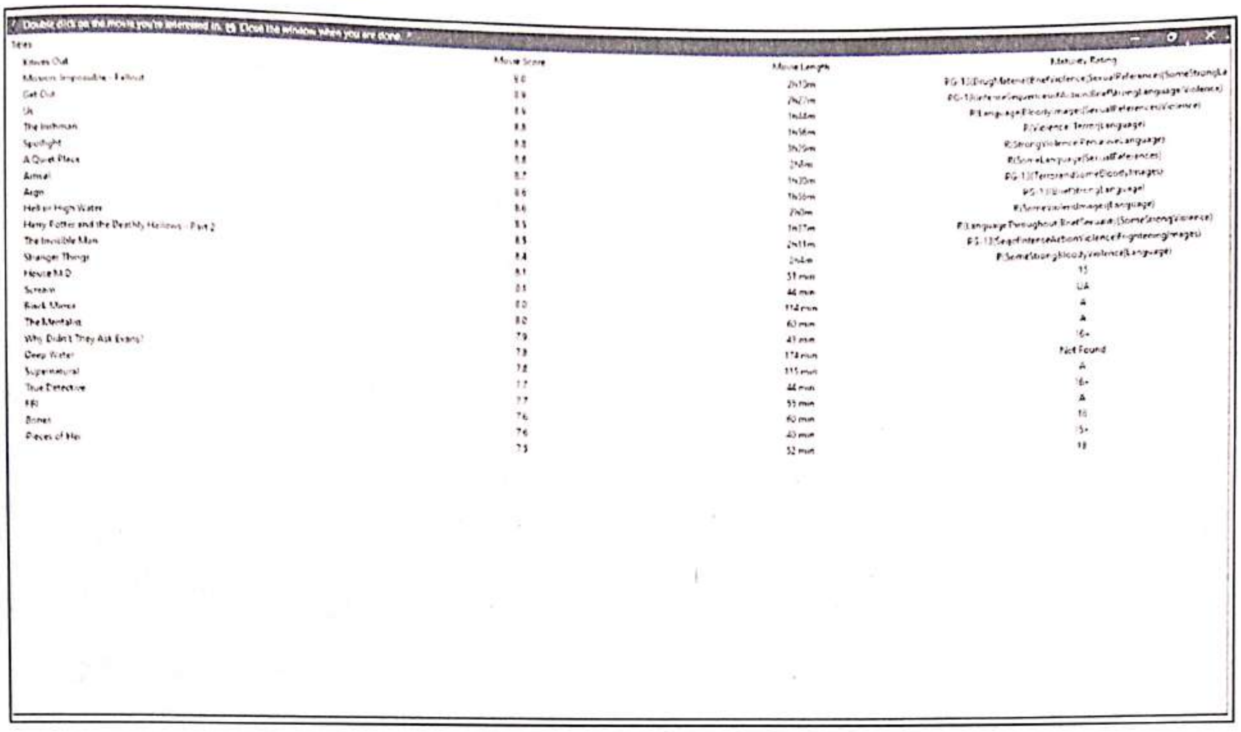
Fig 3.9.1 Tree View Code (1)



```python
    def create_UI(self):
        pg = Treeview(self)
        pg['columns'] = tuple(self.columns)
        pg.heading("#0", text="Titles", anchor="w")
        pg.column("#0", anchor="w")
        for column in self.columns:
            pg.heading('{}'.format(column), text='{}'.format(column))
            pg.column('{}'.format(column), anchor='center', width=100)

        pg.grid(sticky=(N, S, W, E))
        self.treeview = pg
        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=1)

        self.treeview.bind("<Double-1>", self.OnDoubleClick)
        # self.treeview.bind("<Return>", self.QuiT)

    def load_table(self):
        for movie_title in self.movie_dict:
            movie_info = self.movie_dict[movie_title]

            # In case some movies do not have all information provided
            if len(movie_info) == 3:
                movie_info.insert(0, "Not Found")
            elif len(movie_info) == 2:
                movie_info.insert(0, "Not Found")
                movie_info.insert(1, "Not Found")
            elif len(movie_info) == 1:
                movie_info.insert(0, "Not Found")
                movie_info.insert(1, "Not Found")
                movie_info.insert(2, "Not Found")

            self.treeview.insert('', 'end', text="{}".format(movie_title),
                                 values=('{}'.format(movie_info[3]), '{}'.format(movie_info[2]),
                                         '{}'.format(movie_info[1])))

    def OnDoubleClick(self, event):
        self.item = self.treeview.identify('item', event.x, event.y)
        self.selected_movie.append(self.treeview.item(self.item, "text"))

    # def Quit(self, event):
    #     self.parent.destroy()
```

Fig 3.9.2 Tree View Code (2)

3.10 Tree View Output:



Double click on the movie you're interested in. 😀 Close the window when you are done.

Titles	Movie Score	Movie Length	Maturity Rating
Knives Out	9.0	2h10m	PG-13(DrugMaterial)(BriefViolence)(SexualReferences)(SomeStrongLa
Mission Impossible - Fallout	8.9	2h27m	PG-13(IntenseSequencesofAction)(BriefStrongLanguage)(Violence)
Get Out	8.9	1h44m	R(Language)(BloodyImages)(SexualReferences)(Violence)
Us	8.8	1h56m	R(Violence)(Terror)(Language)
The Irishman	8.8	3h29m	R(StrongViolence)(PervasiveLanguage)
Spotlight	8.8	2h8m	R(SomeLanguage)(SexualReferences)
A Quiet Place	8.7	1h30m	PG-13(TerrorandSomeBloodyImages)
Animal	8.6	1h56m	PG-13(BriefStrongLanguage)
Argo	8.6	2h0m	R(SomeViolentImages)(Language)
Hell or High Water	8.5	1h42m	R(LanguageThroughout)(BriefSexuality)(SomeStrongViolence)
Harry Potter and the Deathly Hallows - Part 2	8.5	2h11m	PG-13(SequelIntenseActionViolence)(FrighteningImages)
The Invisible Man	8.4	2h4m	R(SomeStrongBloodyViolence)(Language)
Stranger Things	8.1	51 min	15
House M.D	8.1	44 min	UA
Scream	8.0	114 min	A
Black Mirror	8.0	60 min	A
The Mentalist	7.9	43 min	16+
Why Didn't They Ask Evans?	7.9	174 min	Not Found
Deep Water	7.8	115 min	A
Supernatural	7.7	44 min	16+
True Detective	7.7	55 min	A
FBI	7.6	60 min	16
Bones	7.6	40 min	15+
Pieces of Her	7.5	52 min	18

Fig 3.10.1 Tree View Output

3.11 Summary & Recommendation:

After users chose their favorite movie from the list, we would run a CosineSimilarity analysis to recommend 3 similar movies based on the summary.

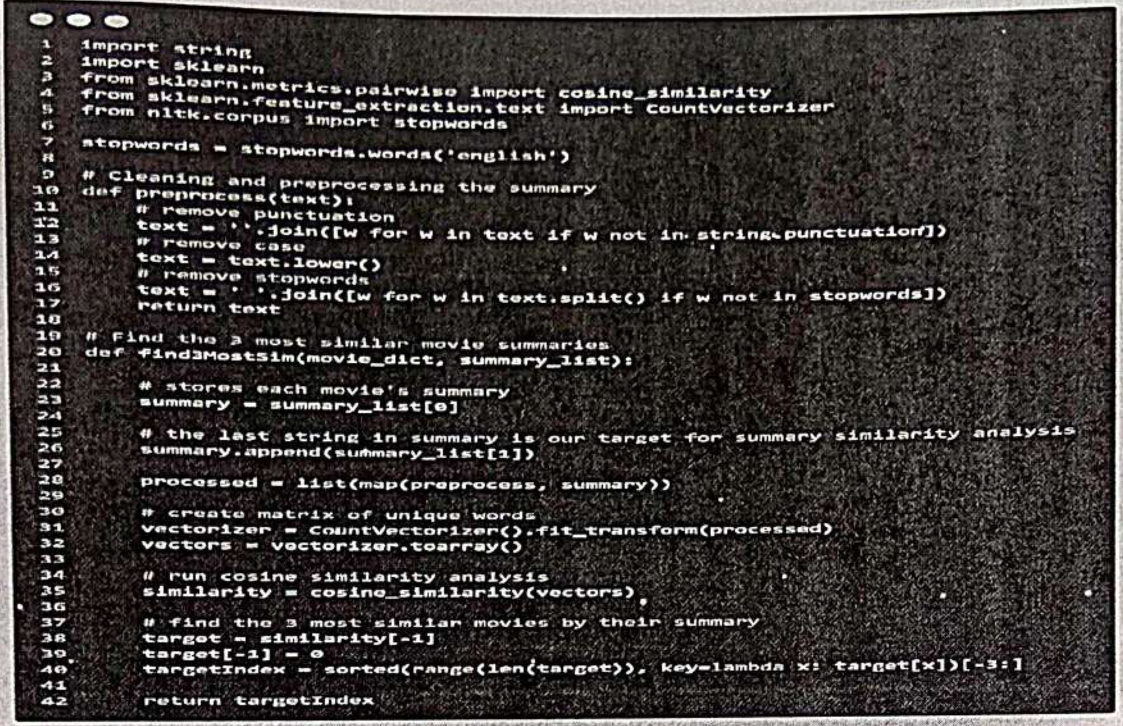
3.11.1 Cosine Similarity:

Cosine similarity is a **metric used to measure how similar two items are**. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The output value ranges from 0–1. 0 means no similarity, whereas 1 means that both the items are 100% similar.

We will import the two important libraries for data analysis and manipulation; **pandas** and **numpy**.

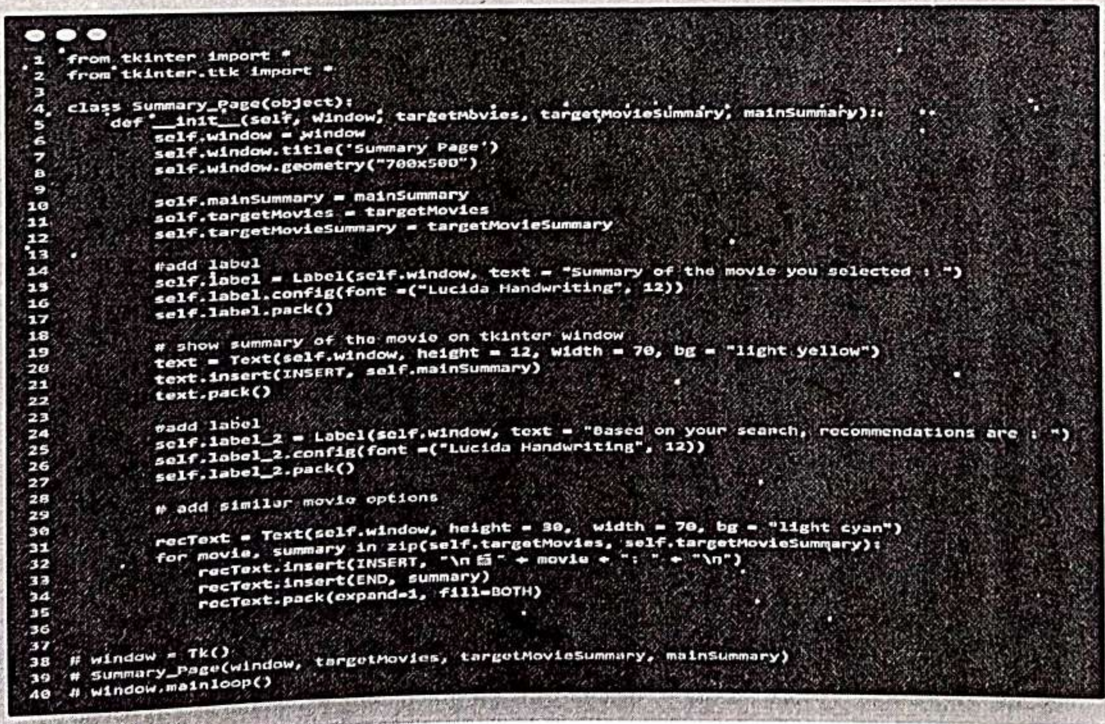
We will also import Scikit-learn's **CountVectorizer,** used to convert a collectionof text documents to a vector of term/token counts.

3.11.2 Summary Source Code:



```python
import string
import sklearn
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords

stopwords = stopwords.words('english')

# Cleaning and preprocessing the summary
def preprocess(text):
    # remove punctuation
    text = ''.join([w for w in text if w not in string.punctuation])
    # remove case
    text = text.lower()
    # remove stopwords
    text = ' '.join([w for w in text.split() if w not in stopwords])
    return text

# Find the 3 most similar movie summaries
def find3MostSim(movie_dict, summary_list):

    # stores each movie's summary
    summary = summary_list[0]

    # the last string in summary is our target for summary similarity analysis
    summary.append(summary_list[1])

    processed = list(map(preprocess, summary))

    # create matrix of unique words
    vectorizer = CountVectorizer().fit_transform(processed)
    vectors = vectorizer.toarray()

    # run cosine similarity analysis
    similarity = cosine_similarity(vectors)

    # find the 3 most similar movies by their summary
    target = similarity[-1]
    target[-1] = 0
    targetIndex = sorted(range(len(target)), key=lambda x: target[x])[-3:]

    return targetIndex
```

Fig 3.11.2.1 Summary Code (1)

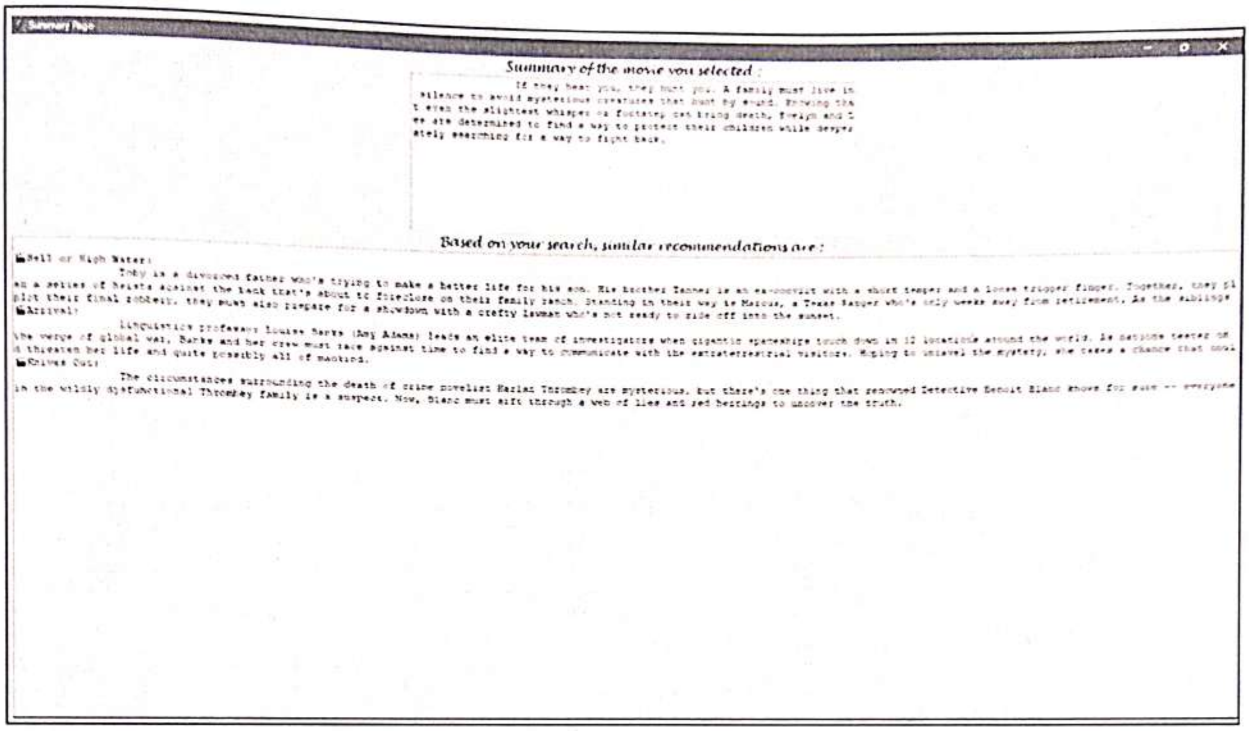


```python
from tkinter import *
from tkinter.ttk import *

class Summary_Page(object):
    def __init__(self, window, targetMovies, targetMovieSummary, mainSummary):
        self.window = window
        self.window.title('Summary Page')
        self.window.geometry("700x500")

        self.mainSummary = mainSummary
        self.targetMovies = targetMovies
        self.targetMovieSummary = targetMovieSummary

        #add label
        self.label = Label(self.window, text = "Summary of the movie you selected : ")
        self.label.config(font =("Lucida Handwriting", 12))
        self.label.pack()

        # show summary of the movie on tkinter window
        text = Text(self.window, height = 12, width = 70, bg = "light yellow")
        text.insert(INSERT, self.mainSummary)
        text.pack()

        #add label
        self.label_2 = Label(self.window, text = "Based on your search, recommendations are : ")
        self.label_2.config(font =("Lucida Handwriting", 12))
        self.label_2.pack()

        # add similar movie options

        recText = Text(self.window, height = 30,  width = 70, bg = "light cyan")
        for movie, summary in zip(self.targetMovies, self.targetMovieSummary):
            recText.insert(INSERT, "\n 🎬 " + movie + ": " + "\n")
            recText.insert(END, summary)
            recText.pack(expand=1, fill=BOTH)



# window = Tk()
# Summary_Page(window, targetMovies, targetMovieSummary, mainSummary)
# window.mainloop()
```

Fig 3.11.2.1 Summary Code (2)

3.11.3 Summary Code Output:

Here is an example of movies similar to A Quiet Place:



Summary of the movie you selected :

```
              If they hear you, they hunt you. A family must live in
 silence to avoid mysterious creatures that hunt by sound. Knowing tha
t even the slightest whisper or footstep can bring death, Evelyn and L
ee are determined to find a way to protect their children while desper
ately searching for a way to fight back.
```

Based on your search, similar recommendations are :

```
Hell or High Water:
                Toby is a divorced father who's trying to make a better life for his son. His brother Tanner is an ex-convict with a short temper and a loose trigger finger. Together, they pl
an a series of heists against the bank that's about to foreclose on their family ranch. Standing in their way is Marcus, a Texas Ranger who's only weeks away from retirement. As the siblings
plot their final robbery, they must also prepare for a showdown with a crafty lawman who's not ready to ride off into the sunset.
Arrival:
                Linguistics professor Louise Banks (Amy Adams) leads an elite team of investigators when gigantic spaceships touch down in 12 locations around the world. As nations teeter on
the verge of global war, Banks and her crew must race against time to find a way to communicate with the extraterrestrial visitors. Hoping to unravel the mystery, she takes a chance that coul
d threaten her life and quite possibly all of mankind.
Knives Out:
                The circumstances surrounding the death of crime novelist Harlan Thrombey are mysterious, but there's one thing that renowned Detective Benoit Blanc knows for sure -- everyone
in the wildly dysfunctional Thrombey family is a suspect. Now, Blanc must sift through a web of lies and red herrings to uncover the truth.
```

Fig 3.11.3.1 Summary Output

3.12 Environment Setup:

Please check out requirements.txt for information.

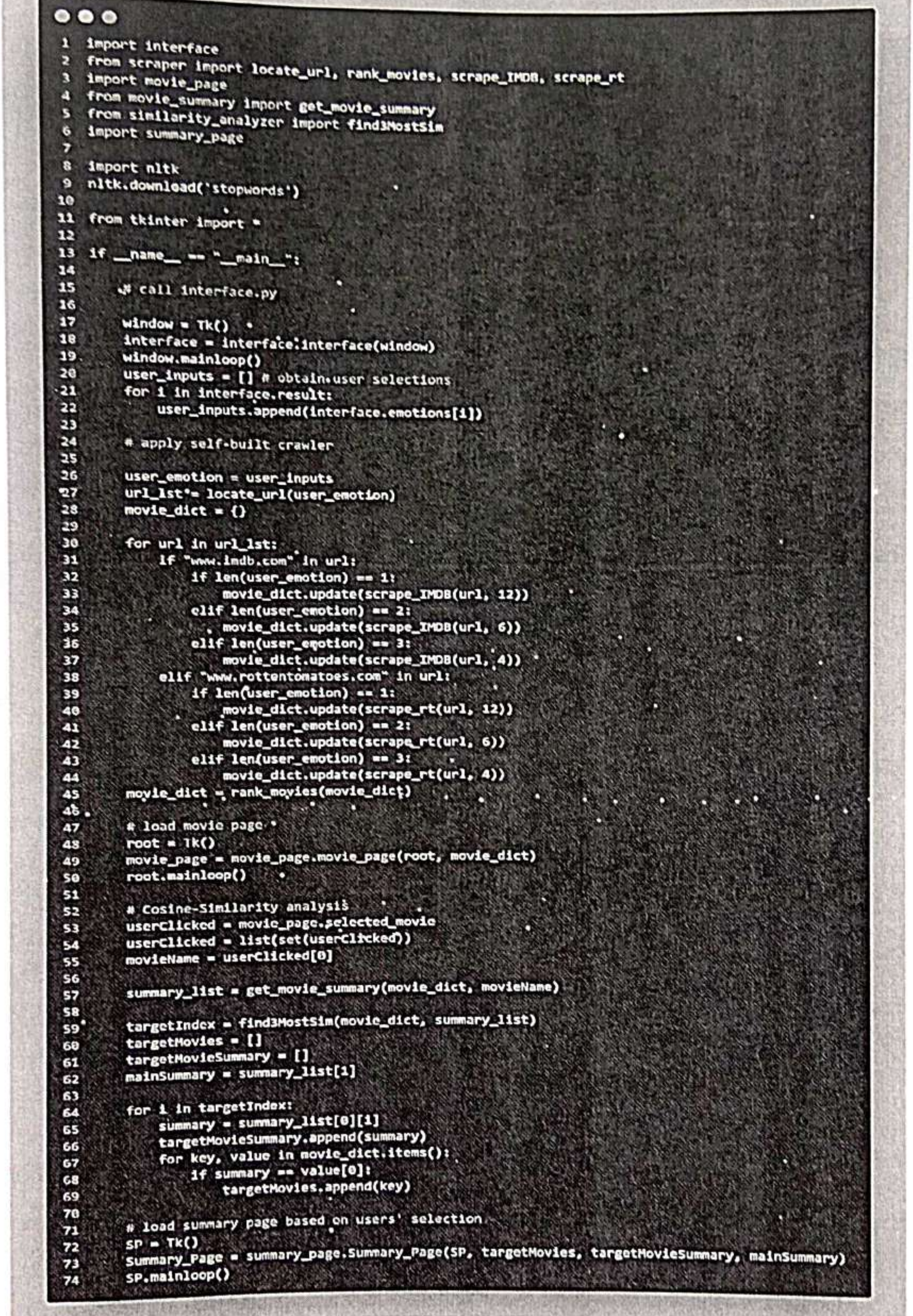
You can install all packages at once using $ pip install -r requirements.txt. Please use Python 3. Otherwise you will need to import tkinter.ttk separately because it is not a submodule of tkinter in Python2.

3.12 How to use program:

After making sure you have all packages installed, activate the program through main.py. The program will start running immediately.

The scraping process may take up to 30 seconds. Please do not close the tkinter window when the program is running.

CHAPTER – 4 : MAIN SCRIPT CODE



```
import interface
from scraper import locate_url, rank_movies, scrape_IMDB, scrape_rt
import movie_page
from movie_summary import get_movie_summary
from similarity_analyzer import find3MostSim
import summary_page

import nltk
nltk.download('stopwords')

from tkinter import *

if __name__ == "__main__":

    # call interface.py

    window = Tk()
    interface = interface.interface(window)
    window.mainloop()
    user_inputs = [] # obtain user selections
    for i in interface.result:
        user_inputs.append(interface.emotions[i])

    # apply self-built crawler

    user_emotion = user_inputs
    url_lst = locate_url(user_emotion)
    movie_dict = {}

    for url in url_lst:
        if "www.imdb.com" in url:
            if len(user_emotion) == 1:
                movie_dict.update(scrape_IMDB(url, 12))
            elif len(user_emotion) == 2:
                movie_dict.update(scrape_IMDB(url, 6))
            elif len(user_emotion) == 3:
                movie_dict.update(scrape_IMDB(url, 4))
        elif "www.rottentomatoes.com" in url:
            if len(user_emotion) == 1:
                movie_dict.update(scrape_rt(url, 12))
            elif len(user_emotion) == 2:
                movie_dict.update(scrape_rt(url, 6))
            elif len(user_emotion) == 3:
                movie_dict.update(scrape_rt(url, 4))
    movie_dict = rank_movies(movie_dict)

    # load movie page
    root = Tk()
    movie_page = movie_page.movie_page(root, movie_dict)
    root.mainloop()

    # Cosine-Similarity analysis
    userClicked = movie_page.selected_movie
    userClicked = list(set(userClicked))
    movieName = userClicked[0]

    summary_list = get_movie_summary(movie_dict, movieName)

    targetIndex = find3MostSim(movie_dict, summary_list)
    targetMovies = []
    targetMovieSummary = []
    mainSummary = summary_list[1]

    for i in targetIndex:
        summary = summary_list[0][i]
        targetMovieSummary.append(summary)
        for key, value in movie_dict.items():
            if summary == value[0]:
                targetMovies.append(key)

    # load summary page based on users' selection
    SP = Tk()
    Summary_Page = summary_page.Summary_Page(SP, targetMovies, targetMovieSummary, mainSummary)
    SP.mainloop()
```

Fig 4.1 Main Script Code

CHAPTER – 5 : FINAL ANALYSIS

5.1 **Result**: Successfully created an emotion-based movie recommendation project using Python and its libraries.

5.2 **Application**: E- MRS is used to provide adapted and personalized suggestions to users using a combination of collaborative filtering and content-based techniques.

5.3 **Problem faced:** Web crawling and Web scraping was a tough work to learn and use them in project to collect data and URL of the websites. There are various challenges faced by Recommendation System. These challenges are Cold Start problem, Data Sparsity, Scalability. Cold Start Problem: It needs enough users in the system to find a match.

5.4 **Limitations**:

- Can't see users' behavior change.
- Lack of Data Analysis capability.
- Complex Execution Process.
- Too many choices for users.
- Takes more time to execute.

CHAPTER – 6 : CONCLUSION

6.1 Conclusion:

Emotion based Movie Recommendation System formed is helpful for user to get movies recommendations based according to the users' emotion selected by them.

Requires a better internet connection and web scraping method which takes lesser time to fetch the movies from IMDB and Rotten Tomatoes.

6.2 Future Work:

Can include http link to movies available on YouTube or other platforms for user to directly watch rather than searching it by themselves.

We can use face recognition system also to detect emotion of the user by web cam and higher use of Machine Learning instead of giving emotion input by GUI.

REFERENCES

1. The Hundred-Page Machine Learning Book. Author – Andriy Burkov. Latest Edition – First. Publisher – Andriy Burkov.

2. 4. Machine Learning By Tom M Mitchell. Author – Tom M. Mitchell. Latest Edition – First. Publisher – McGraw Hill Education.

3. Think Python: How to Think Like a Computer Scientist, 2nd edition Allen B. Downey (O'Reilly, 2015)

4. Python Crash Course Eric Matthes (No Starch Press, 2016)