# 

## DLITHE INTERNSHIP RESEARCH PROJECT

### PROJECT ON

### MOVIE RECOMMENDATION SYSTEM

**FROM**

**H DEEKSHA SHANBHAG**

**ANAGHA N**

**JESSICA DSOUZA**

**ABSTRACT:**

A recommendation engine filters the data using different algorithms and recommends the most relevant items to users. It first captures the past behavior of a customer and based on that, recommends products which the users might be likely to buy. If a completely new user visits an e-commerce site, that site will not have any past history of that user. So how does the site go about recommending products to the user in such a scenario? One possible solution could be to recommend the best selling products, i.e. the products which are high in demand. Another possible solution could be to recommend the products which would bring the maximum profit to the business. Approaches used for our recommender systems is collaborative filtering, where we try to group similar users together and use information about the group to make recommendations to the user.

**TABLE OF CONTENTS:**

|  |  |  |
| --- | --- | --- |
| S.no | Contents | P.no |
| 1. | Introduction | 1 |
| 2. | Technology used | 2 |
| 3. | Theory | 3 |
| 4. | Code | 4-8 |
| 5. | Screenshots | 9-11 |
| 6. | Conclusion | 12 |

**INTRODUCTION:**

## Movie Recommendation System Project using ML

The main goal of this machine learning project is to build a recommendation engine that recommends movies to users. This project is designed to help us understand the functioning of how a recommendation system works. We will be developing an Item Based Collaborative Filter.

A recommendation system provides suggestions to the users through a filtering process that is based on user preferences and browsing history. The information about the user is taken as an input. The information is taken from the input that is in the form of browsing data. This information reflects the prior usage of the product as well as the assigned ratings. A recommendation system is a platform that provides its users with various contents based on their preferences and likings. A recommendation system takes the information about the user as an input. The recommendation system is an implementation of the [**machine learning algorithms**](https://data-flair.training/blogs/machine-learning-algorithms/).

A recommendation system also finds a similarity between the different products. For example,netflix and amazon prime, provides you with the recommendations of the movies which is similar to the ones that we have watched before. There are two types of recommendation systems – Content-Based Recommendation System and Collaborative Filtering Recommendation. this project of recommendation system is a collaborative filtering recommendation system. It is also an ITEM based collaborative recommendation system(**Item**-**based collaborative filtering** is a model-based algorithm for making recommendations.)

**TECHNOLOGY USED:**

PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

**JUPYTER NOTEBOOK**

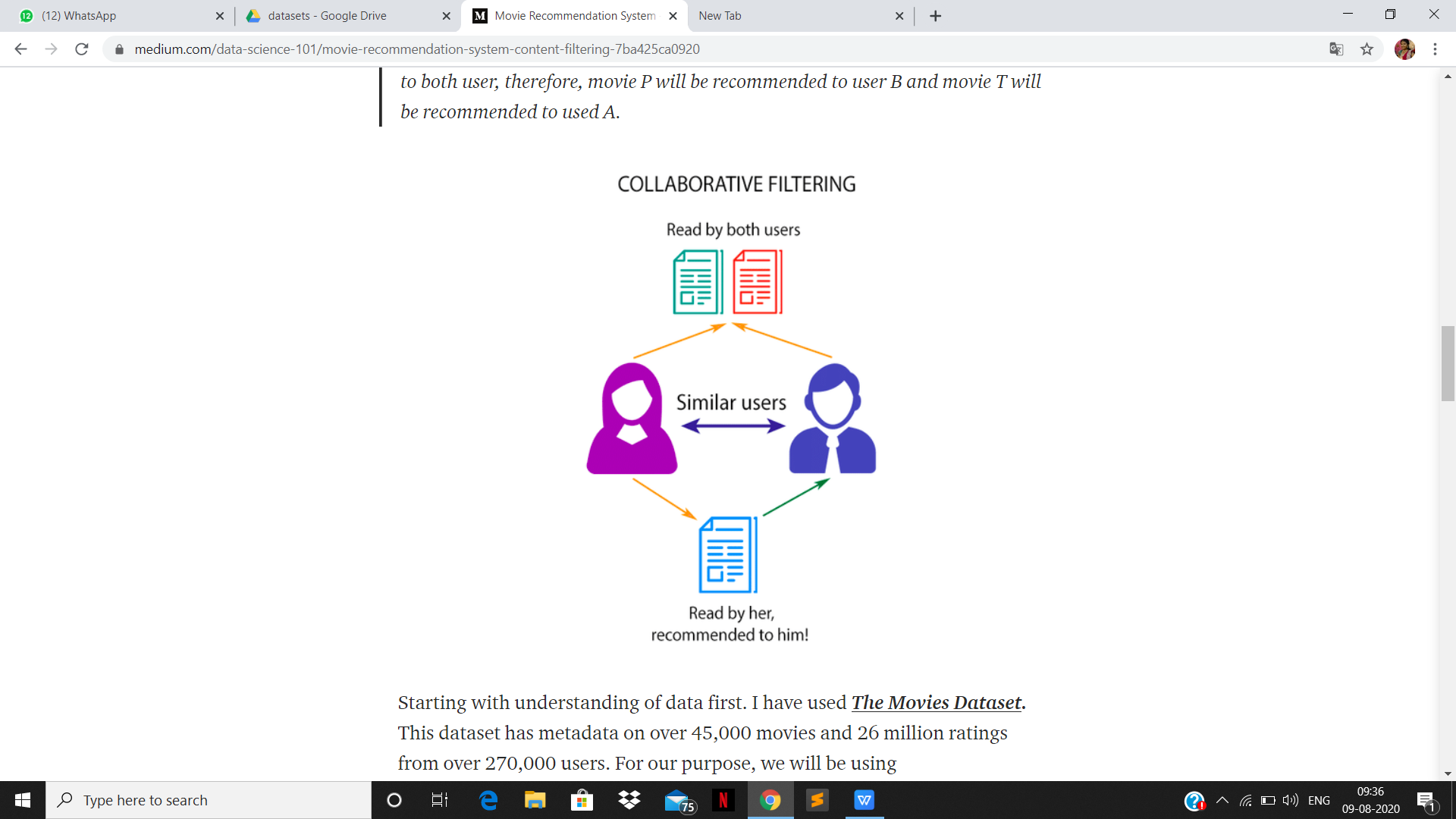
The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

Uses include:

data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

**THEORY:**

Collaborative Filtering involves suggesting movies to the users that are based on collecting preferences from many other users. For example, if a user A likes to watch action films and so does user B, then the movies that the user B will watch in the future will be recommended to A and vice-versa. Therefore, recommending movies is dependent on creating a relationship of similarity between the two users.



**CODE:**

main.py

import numpy as np

import pandas as pd

from flask import Flask, render\_template, request

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import json

import bs4 as bs

import urllib.request

import pickle

import requests

What **pickle** does is that it “serializes” the object first before writing it to **file**. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

(nlp\_model.pkl and transform.pkl)

# load the nlp model and tfidf vectorizer from disk

filename = 'nlp\_model.pkl'

clf = pickle.load(open(filename, 'rb'))

vectorizer = pickle.load(open('tranform.pkl','rb'))

A **Dataset** is the basic data container in PyMVPA. It serves as the primary form of data storage, but also as a common container for results returned by most algorithms. In this tutorial part we will take a look at what a **dataset** consists of, and how it works.

Here the dataset contains information about the movies.(main\_data.csv)

def create\_similarity():

data = pd.read\_csv('datasets/main\_data.csv')

# creating a count matrix

cv = CountVectorizer()

count\_matrix = cv.fit\_transform(data['comb'])

# creating a similarity score matrix

similarity = cosine\_similarity(count\_matrix)

return data,similarity

def rcmd(m):

m = m.lower()

try:

data.head()

similarity.shape

except:

data, similarity = create\_similarity()

if m not in data['movie\_title'].unique():

return('Sorry! The movie you requested is not in our database. Please check the spelling or try with some other movies')

else:

i = data.loc[data['movie\_title']==m].index[0]

lst = list(enumerate(similarity[i]))

lst = sorted(lst, key = lambda x:x[1] ,reverse=True)

lst = lst[1:11] # excluding first item since it is the requested movie itself

l = []

for i in range(len(lst)):

a = lst[i][0]

l.append(data['movie\_title'][a])

return l

# converting list of string to list (eg. "["abc","def"]" to ["abc","def"])

def convert\_to\_list(my\_list):

my\_list = my\_list.split('","')

my\_list[0] = my\_list[0].replace('["','')

my\_list[-1] = my\_list[-1].replace('"]','')

return my\_list

def get\_suggestions():

data = pd.read\_csv('datasets/main\_data.csv')

return list(data['movie\_title'].str.capitalize())

**Flask** is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.

app = Flask(\_\_name\_\_)

@app.route("/")

@app.route("/home")

def home():

suggestions = get\_suggestions()

return render\_template('home.html',suggestions=suggestions)

@app.route("/similarity",methods=["POST"])

def similarity():

movie = request.form['name']

rc = rcmd(movie)

if type(rc)==type('string'):

return rc

else:

m\_str="---".join(rc)

return m\_str

Here def recommand() is the function used for the recommendation.

The data is collected from the dataset and then all relatale movies will be suggested.

@app.route("/recommend",methods=["POST"])

def recommend():

# getting data from AJAX request

title = request.form['title']

cast\_ids = request.form['cast\_ids']

cast\_names = request.form['cast\_names']

cast\_chars = request.form['cast\_chars']

cast\_bdays = request.form['cast\_bdays']

cast\_bios = request.form['cast\_bios']

cast\_places = request.form['cast\_places']

cast\_profiles = request.form['cast\_profiles']

imdb\_id = request.form['imdb\_id']

poster = request.form['poster']

genres = request.form['genres']

overview = request.form['overview']

vote\_average = request.form['rating']

vote\_count = request.form['vote\_count']

release\_date = request.form['release\_date']

runtime = request.form['runtime']

status = request.form['status']

rec\_movies = request.form['rec\_movies']

rec\_posters = request.form['rec\_posters']

# get movie suggestions for auto complete

suggestions = get\_suggestions()

# call the convert\_to\_list function for every string that needs to be converted to list

rec\_movies = convert\_to\_list(rec\_movies)

rec\_posters = convert\_to\_list(rec\_posters)

cast\_names = convert\_to\_list(cast\_names)

cast\_chars = convert\_to\_list(cast\_chars)

cast\_profiles = convert\_to\_list(cast\_profiles)

cast\_bdays = convert\_to\_list(cast\_bdays)

cast\_bios = convert\_to\_list(cast\_bios)

cast\_places = convert\_to\_list(cast\_places)

# convert string to list (eg. "[1,2,3]" to [1,2,3])

cast\_ids = cast\_ids.split(',')

cast\_ids[0] = cast\_ids[0].replace("[","")

cast\_ids[-1] = cast\_ids[-1].replace("]","")

# rendering the string to python string

for i in range(len(cast\_bios)):

cast\_bios[i] = cast\_bios[i].replace(r'\n', '\n').replace(r'\"','\"')

combining multiple lists as a dictionary which can be passed to the html file so that it can be processed easily and the order of information will be preserved

movie\_cards = {rec\_posters[i]: rec\_movies[i] for i in range(len(rec\_posters))}

casts = {cast\_names[i]:[cast\_ids[i], cast\_chars[i], cast\_profiles[i]] for i in range(len(cast\_profiles))}

cast\_details = {cast\_names[i]:[cast\_ids[i], cast\_profiles[i], cast\_bdays[i], cast\_places[i], cast\_bios[i]] for i in range(len(cast\_places))}

# web scraping to get user reviews from IMDB site

sauce = urllib.request.urlopen('https://www.imdb.com/title/{}/reviews?ref\_=tt\_ov\_rt'.format(imdb\_id)).read()

soup = bs.BeautifulSoup(sauce,'lxml')

soup\_result = soup.find\_all("div",{"class":"text show-more\_\_control"})

reviews\_list = [] # list of reviews

reviews\_status = [] # list of comments (good or bad)

for reviews in soup\_result:

if reviews.string:

reviews\_list.append(reviews.string)

# passing the review to our model

movie\_review\_list = np.array([reviews.string])

movie\_vector = vectorizer.transform(movie\_review\_list)

pred = clf.predict(movie\_vector)

reviews\_status.append('Good' if pred else 'Bad')

# combining reviews and comments into a dictionary

movie\_reviews = {reviews\_list[i]: reviews\_status[i] for i in range(len(reviews\_list))}

In this snippet,all the data is passed to the html file.

return render\_template('recommend.html',title=title,poster=poster,overview=overview,vote\_average=vote\_average,

vote\_count=vote\_count,release\_date=release\_date,runtime=runtime,status=status,genres=genres,

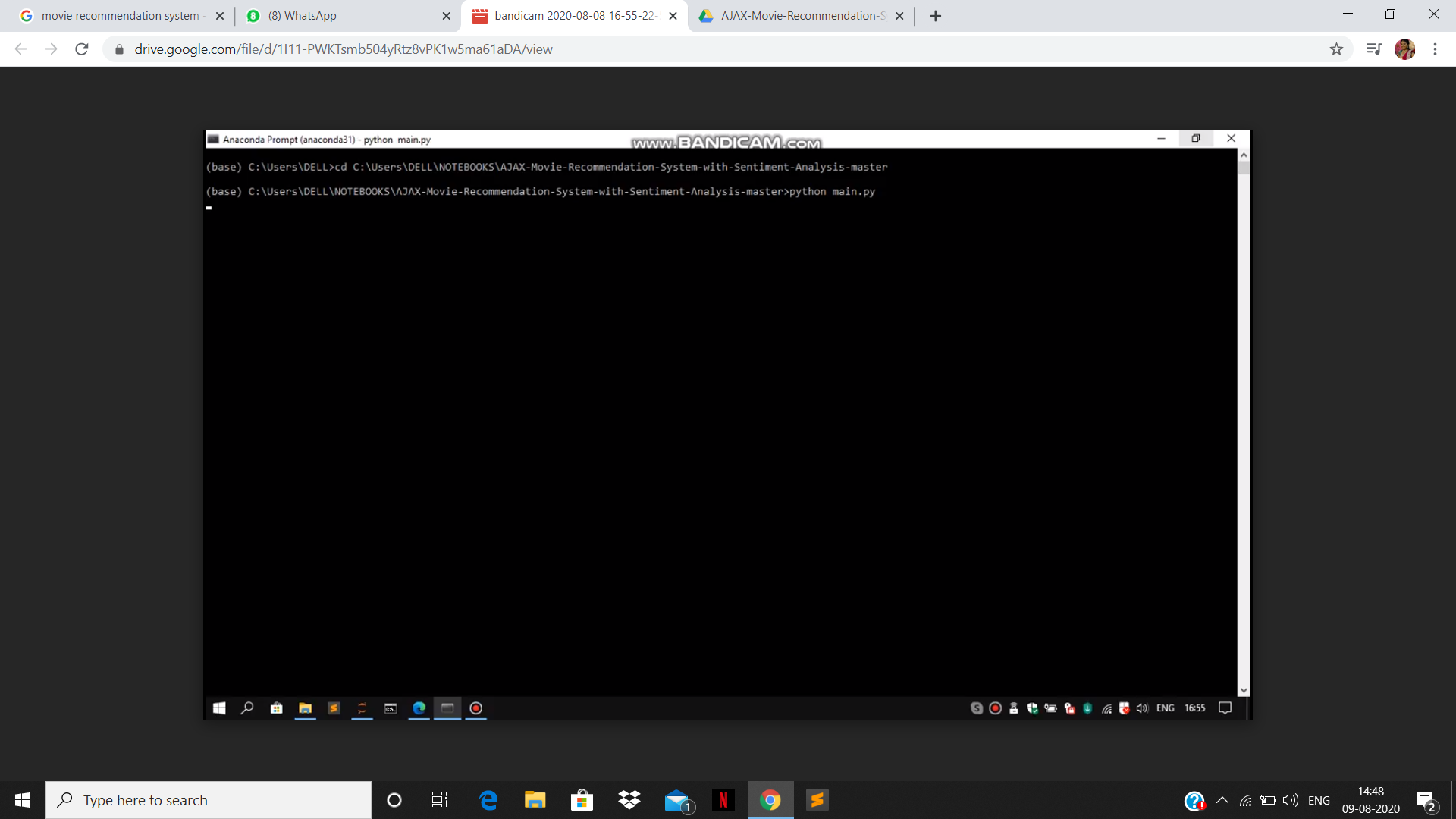
movie\_cards=movie\_cards,reviews=movie\_reviews,casts=casts,cast\_details=cast\_details)

if \_\_name\_\_ == '\_\_main\_\_':

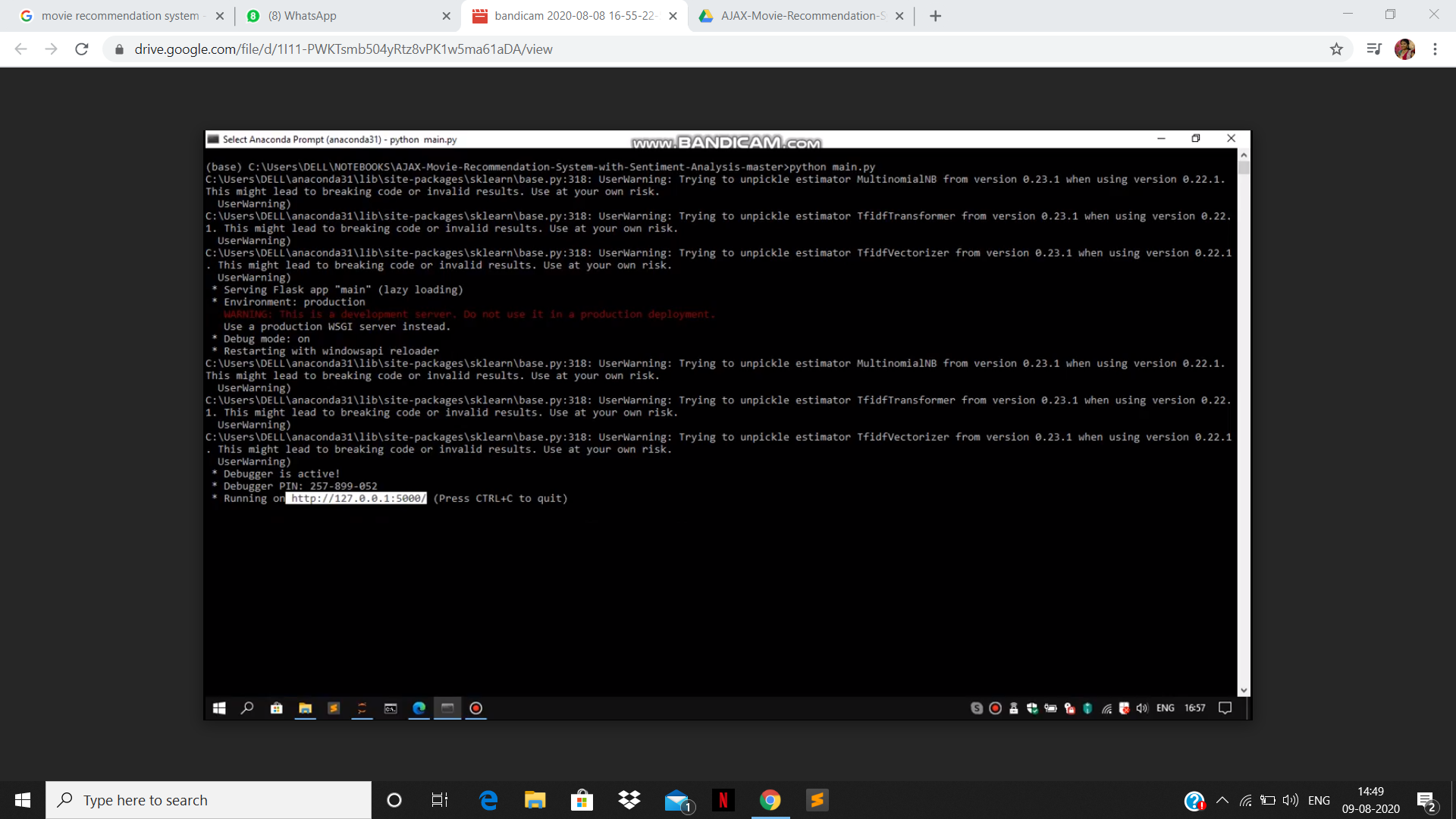
app.run(debug=True)

**SCREENSHOTS:**

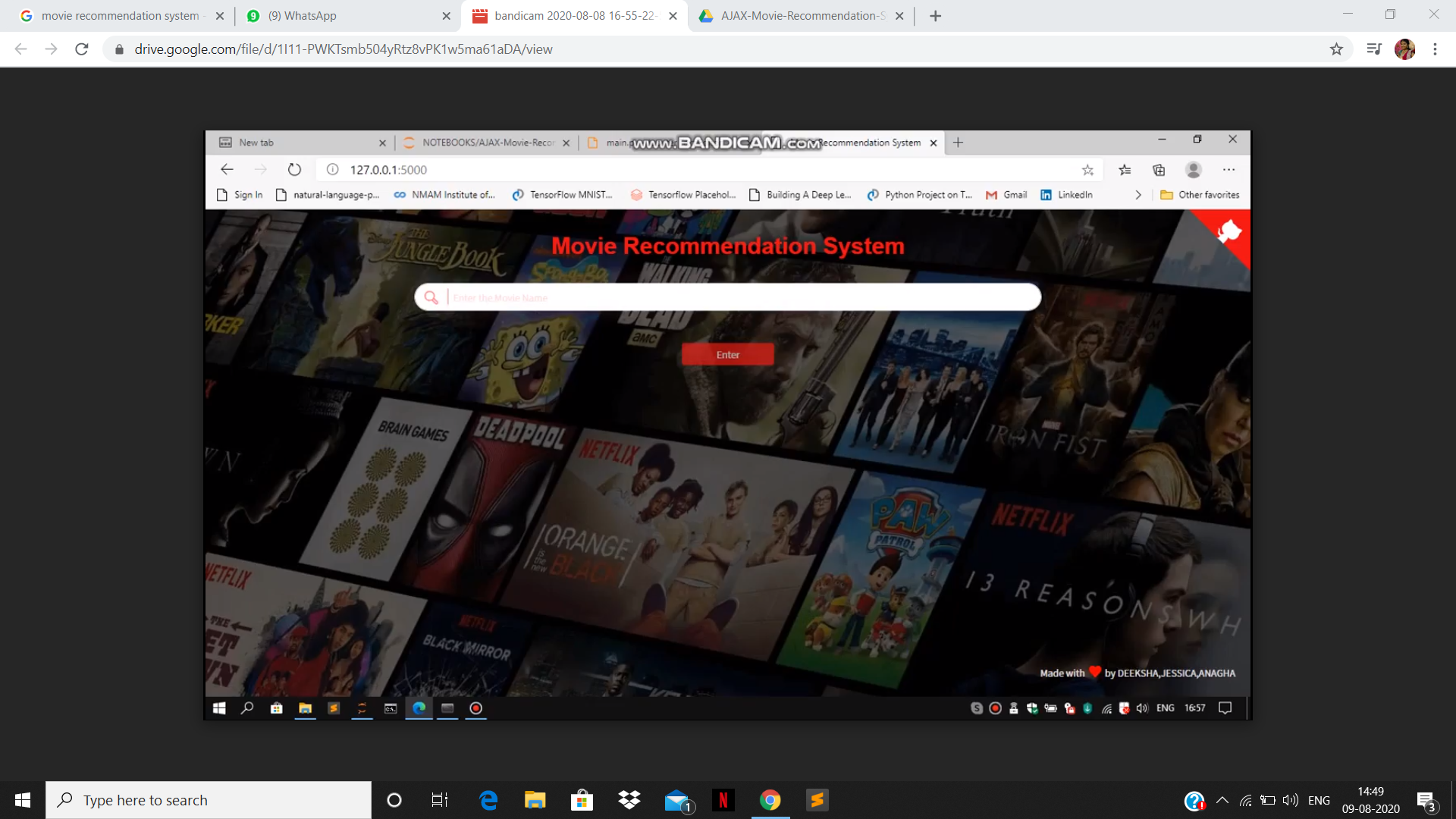
1. A)In the command prompt, the main.py code is running.



1. B)



2)



**CONCLUSION:**

Recommender systems open new opportunities of retrieving personalized information on the Internet. It also helps to alleviate the problem of information overload which is a very common phenomenon with information retrieval systems and enables users to have access to products and services which are not readily available to users on the system. We come up with a strategy that focuses on dealing with user’s personal interests and based on his previous reviews, movies are recommended to users. This strategy helps in improving accuracy of the recommendations. A personal profile is created for each user, where each user has access to his own history, his likes, ratings, comments, password modification processes. It also helps in collecting authentic data with improved accuracy and makes the system more responsive.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*