**MOVIE RECOMMENDATION SYSTEM**

**UCM008 Machine Learning Project Report**

**END-Semester Evaluation**

**Submitted by:**

**(102108211) GURRIDAY CHAWLA**

**(102108032) UTKARSH GULERIA**

**BE Third Year, CS Minor**

**Submitted to:**

Dr. Anurag tiwari



**Computer Science and Engineering Department**

**TIET, Patiala**

**Nov 2023**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SR.NO.** | **TOPIC** | **PG.NO.** |
| 1. | Introduction | 3 |
| 2. | Problem Definition and Algorithm | 4 |
| 2.1. | Task Definition | 4 |
| 2.2. | Algorithm and Approach | 4 |
| 3. | Experimental Evaluation | 8 |
| 3.1. | Methodology | 8 |
| 3.2. | Results | 13 |
| 4. | Website | 15 |
| 5. | Future scope | 17 |
| 6. | Conclusion | 18 |

**INTRODUCTION**

In the vast realm of entertainment, movies have emerged as a captivating form of storytelling, immersing audiences in worlds of imagination, adventure, and emotion. However, with the ever-expanding universe of cinematic creations, navigating the vast landscape of films can be a daunting task. Movie recommendation systems, powered by machine learning algorithms, have emerged as a valuable tool, alleviating the burden of choice and guiding viewers towards films that align with their preferences and tastes.

At the heart of movie recommendation systems lies the ability to analyze and interpret vast amounts of data, encompassing user preferences, movie attributes, and the intricate relationships between them. These systems employ sophisticated algorithms to extract patterns and correlations, unveiling the hidden connections that shape user enjoyment. By understanding these connections, recommendation systems can predict which movies a particular user is likely to find appealing, transforming the cinematic experience from a haphazard exploration into a personalized journey of discovery.

**PROBLEM DEFINITION AND ALGORITHM**

**TASK DEFINITION**

The task of building a movie recommendation system is to develop an algorithm that can predict which movies a particular user would be likely to enjoy. This is a challenging task because there are many factors that can influence a user's movie preferences, such as their personal taste, past viewing history, and demographic information.

**ALGORITHM AND APPROACH**

**Content-Based Filtering**

Content-based filtering (CBF) is a recommendation technique that suggests items to users based on the similarity of those items to items that the user has already liked or interacted with. It assumes that users have similar preferences for items that share similar characteristics. For instance, if a user has enjoyed watching science fiction movies, the CBF algorithm might recommend other science fiction movies to that user.

CBF algorithms typically work by extracting features from items, such as keywords, genres, directors, actors, and plot descriptions. These features are then used to represent the items in a high-dimensional space, where items with similar features are located closer together in the space. To recommend items to a user, the CBF algorithm calculates the similarity between the user's profile, which represents the user's preferences, and the items in the database. Items with the highest similarity scores are then recommended to the user.

**Collaborative Filtering**

Collaborative filtering (CF) is another recommendation technique that suggests items to users based on the preferences of similar users. It assumes that users with similar tastes will also like similar items. For example, if a user has similar tastes to another user who has liked a particular movie, the CF algorithm might recommend that movie to the first user.

CF algorithms typically work by creating a user-item matrix, which represents the preferences of all users for all items. The entries in the matrix can be binary values indicating whether a user has liked an item or numerical values indicating the degree to which a user likes an item. To recommend items to a user, the CF algorithm finds users who are similar to the target user and then recommends items that those similar users have liked.

**Hybrid recommendation**

Hybrid recommendation systems combine the strengths of both CBF and CF algorithms to provide more accurate and personalized recommendations. CBF algorithms can provide good recommendations for new items or for users with very few interactions, while CF algorithms can provide good recommendations for items that are popular or for users with many interactions.

There are many different ways to combine CBF and CF algorithms. One common approach is to use CBF to generate a list of candidate items and then use CF to rank those items and select the top-ranked items to recommend to the user. Another approach is to use a weighted combination of CBF and CF scores to generate the final recommendations.

**TEXT VECTORIZATION**

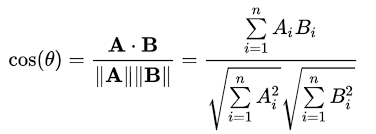
Text vectorization is a crucial step in natural language processing and machine learning projects. It involves converting textual data into numerical vectors, enabling algorithms to analyze and make predictions. Common techniques include bag-of-words and TF-IDF, where words are represented by their frequency or importance in a document. Another powerful approach is word embeddings, like Word2Vec or GloVe, which capture semantic relationships between words. These numerical representations facilitate the application of various machine learning models, allowing them to interpret and learn patterns from text, ultimately enhancing the performance of tasks such as sentiment analysis, document classification, and language translation.

**BAG OF WORDS**

The Bag-of-Words (BoW) model stands as a fundamental and widely used technique in natural language processing (NLP) and machine learning, particularly in text analysis and document classification. The essence of the Bag-of-Words model lies in its simplicity and effectiveness. It treats a document as an unordered set of words, disregarding grammar and word order, focusing solely on the frequency of words within the text. In this approach, a corpus's vocabulary is identified, and each document is represented as a vector, with each dimension corresponding to a unique word in the vocabulary. The values in the vector denote the frequency of each word's occurrence in the document. While BoW discards the sequential structure of words, it successfully transforms textual information into a numerical format that machine learning algorithms can comprehend. Despite its simplicity, the Bag-of-Words model has proven instrumental in tasks such as document classification, sentiment analysis, and information retrieval, making it a foundational concept for understanding and processing textual data in the realm of machine learning.

**COSINE DISTANCE**

Cosine distance, a widely employed metric in the realm of similarity analysis, provides a nuanced measure of similarity between vectors, particularly in the context of text and document comparisons. This distance metric hinges on the geometric interpretation of vectors, assessing the cosine of the angle between them. In the domain of text vectorization, where documents are often represented as numerical vectors, cosine distance emerges as a valuable tool for quantifying the similarity or dissimilarity between these vectors. The formula for cosine distance is expressed as the cosine of the angle θ between two vectors, A and B, and is mathematically represented as:



Here, A.B denotes the dot product of vectors A and B, while ||A|| and ||B|| represent the Euclidean norms of vectors A and B, respectively. The resulting value ranges between -1 (perfect dissimilarity) and 1 (perfect similarity), with 0 indicating orthogonality or no similarity.

Cosine distance's significance extends beyond its mathematical elegance. Its application spans diverse fields, including information retrieval, text clustering, and recommendation systems. In text analysis, for instance, it allows for the comparison of document vectors, enabling the identification of documents that share similar thematic content. The flexibility of cosine distance lies in its ability to mitigate the impact of vector magnitude, focusing solely on the directionality of vectors. This proves especially advantageous when dealing with high-dimensional data, as it minimizes the impact of varying vector lengths.

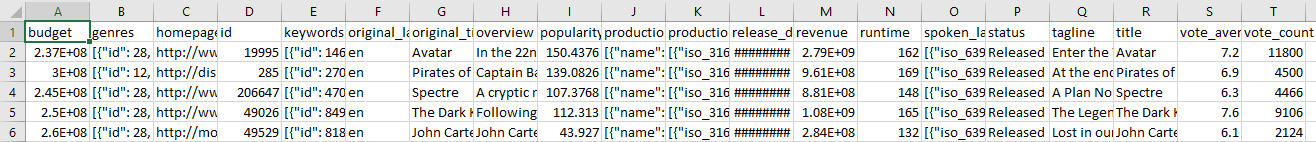
As machine learning and natural language processing applications continue to burgeon, cosine distance remains a stalwart tool, facilitating the extraction of valuable insights from large corpora of text data. Its ability to discern semantic similarity by considering the orientation of vectors contributes to its resilience in diverse analytical scenarios. Thus, the formulaic elegance and practical utility of cosine distance underscore its enduring relevance in the evolving landscape of similarity analysis and text mining.

**EXPERIMENTAL EVALUATION**

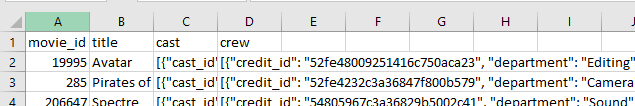
**METHODOLOGY**

**DATASET**: THERE ARE 2 DATA SETS INVOLVED IN THE PROJECT . THE MOVIES DATASET HAS 20 ATTRIBUTES AND THE CREDITS DATASET HAS 4 ATTRIBUTES.

**MOVIE DATASET**



**CREDITS DATASET**



**Links:-**

**1):-** [**https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata/?select=tmdb\_5000\_movies.csv**](https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata/?select=tmdb_5000_movies.csv)

**2):-** **https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset/code**

**STEPS**

IMPORTING REQUIRED LIBRARIES

import numpy as np

import pandas as pd

IMPORTING DATASETS

movies = pd.read\_csv('tmdb\_5000\_movies.csv')

credits = pd.read\_csv('tmdb\_5000\_credits.csv')

MERGING THE TWO DATASETS

movies = movies.merge(credits,on='title')

FILTERING THE TAGS

movies = movies[['movie\_id','title','overview','genres','keywords','cast','crew']]

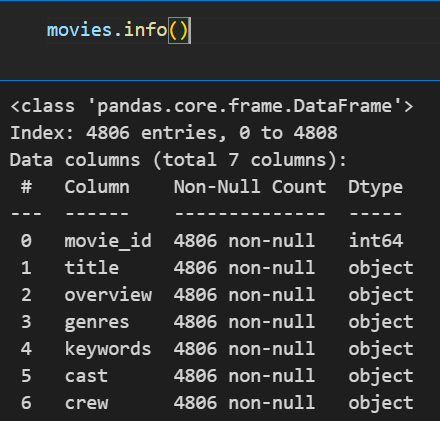


DATA PREPROCESSING AND CLEANING

movies.isnull().sum()

movies.dropna(inplace=True)

movies.duplicated().sum()



CONVERTING FROM STRING TO LIST

def convert(text):

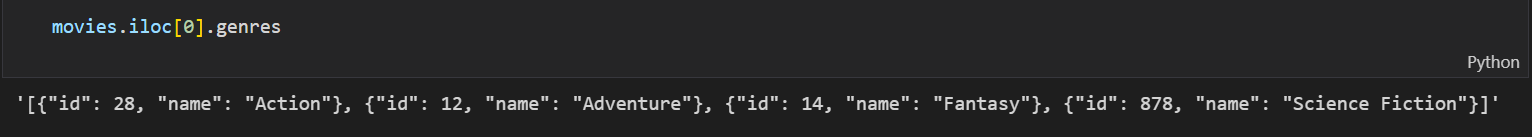
    L = []

    for i in ast.literal\_eval(text):

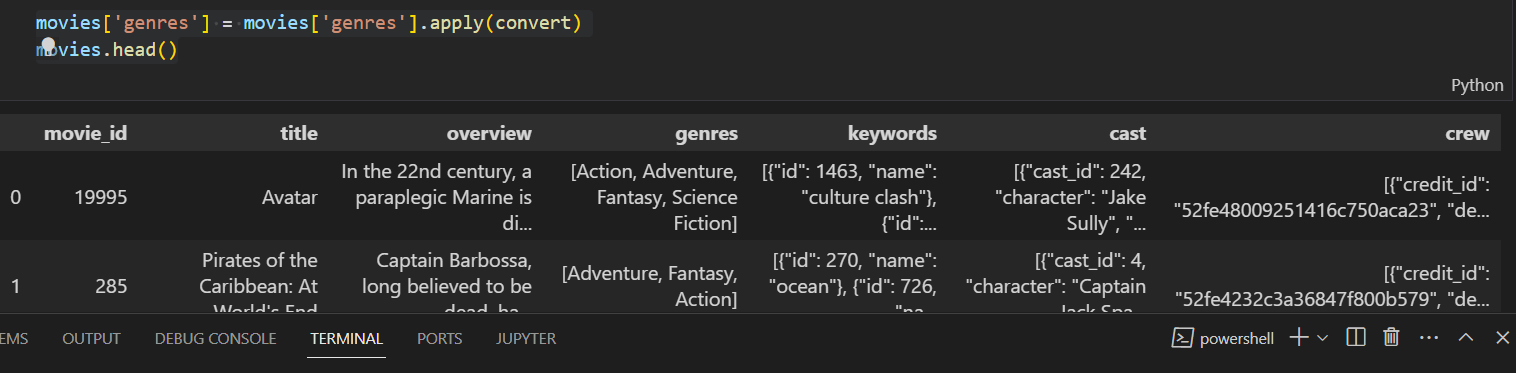
        L.append(i['name'])

    return L

BEFORE



AFTER



FOR CAST

def convert3(text):

    L = []

    counter = 0

    for i in ast.literal\_eval(text):

        if counter < 3:

            L.append(i['name'])

        counter+=1

    return L

movies['cast'] = movies['cast'].apply(convert)

movies.head()

movies['cast'] = movies['cast'].apply(lambda x:x[0:3])

FOR DIRECTOR

def fetch\_director(text):

    L = []

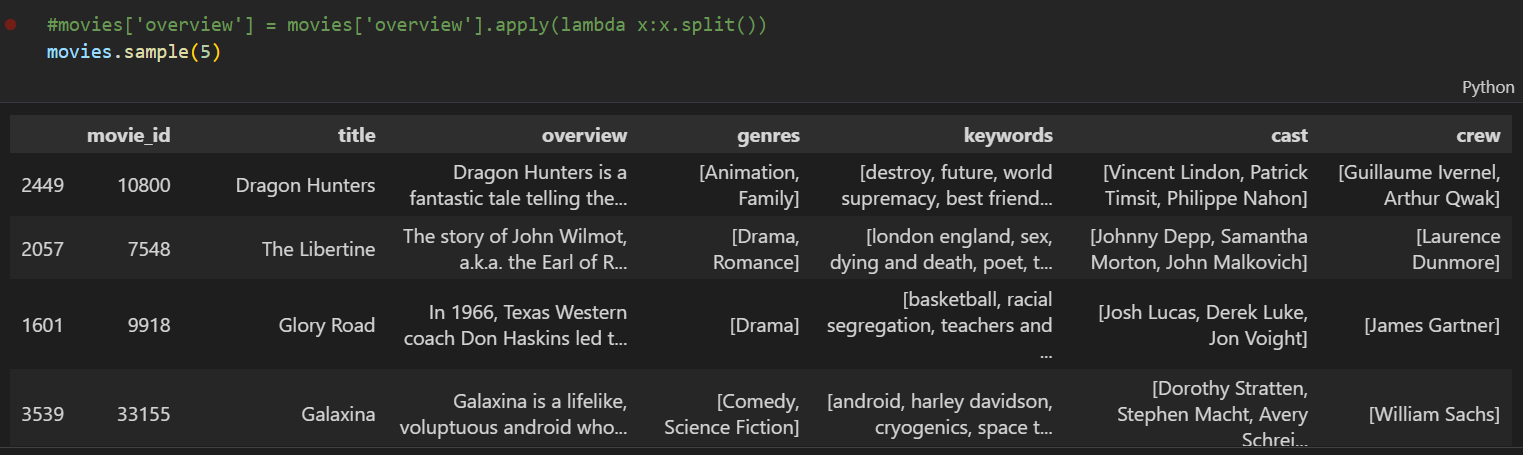
    for i in ast.literal\_eval(text):

        if i['job'] == 'Director':

            L.append(i['name'])

    return L

movies['crew'] = movies['crew'].apply(fetch\_director)



REMOVING SPACE FOR TAGS

def collapse(L):

    L1 = []

    for i in L:

        L1.append(i.replace(" ",""))

    return L1

movies['cast'] = movies['cast'].apply(collapse)

movies['crew'] = movies['crew'].apply(collapse)

movies['genres'] = movies['genres'].apply(collapse)

movies['keywords'] = movies['keywords'].apply(collapse)



CONVERTING OVERVIEW FROM PARAGRAPH TO LIST

movies['overview'] = movies['overview'].apply(lambda x:x.split())

movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] + movies['crew']

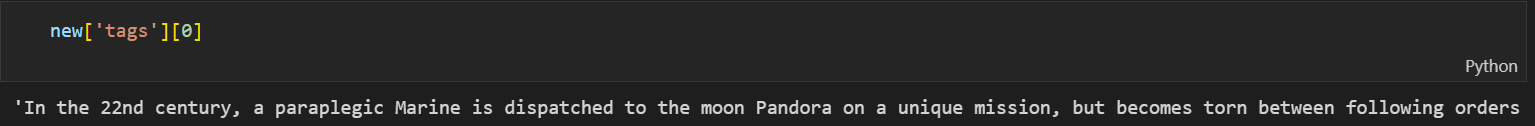
new = movies.drop(columns=['overview','genres','keywords','cast','crew'])

JOINING ELEMENTS IN TAG

new['tags'] = new['tags'].apply(lambda x: " ".join(x))

new.head()





REMOVING STOPWORDS

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(max\_features=5000,stop\_words='english')

CONVERTING TO ARRAY

vector = cv.fit\_transform(new['tags']).toarray()

FROM STEM WORDS

import nltk

from nltk.stem.porter import PorterStemmer

ps=PorterStemmer()

def stem(text):

  y=[]

  for i in text.split():

    ps.stem(i)

    y.append(ps.stem(i))

    return " ".join(y)

ps.stem('collegees')



APPLYING COSINE SIMILARITY

from sklearn.metrics.pairwise import cosine\_similarity

similarity = cosine\_similarity(vector)

sorted(similarity[0],reverse=True)

MAKING FUNCTION TO RECOMMEND A MOVIE

def recommend(movie):

    index = new[new['title'] == movie].index[0]

    distances = sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x: x[1])

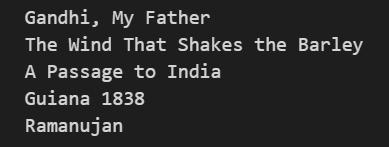
    for i in distances[1:6]:

        print(new.iloc[i[0]].title)

**INPUT**

recommend('Gandhi')

**OUTPUT**



**APP code**

**Some library for app**

import pickle

import streamlit as st

import numpy as np

import pandas as pd

Streamlit is been used by us here to join python generated pkl file to make

our website.

pickle is used to unzip and use the pkl file generated by recommendation system.

def recommend(movie):

    index = movies[movies['title'] == movie].index[0]

    distances = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda x: x[1])

    recommended\_movie\_names = []

    for i in distances[1:6]:

        movie\_id = movies.iloc[i[0]].movie\_id

        recommended\_movie\_names.append(movies.iloc[i[0]].title)

    return recommended\_movie\_names

This is code which works same as the one metioned above here it will recommend the movie based on cosine similarity and distance.

movies = pickle.load(open('movie\_list.pkl','rb'))

similarity = pickle.load(open('similarity.pkl','rb'))

movie\_list = movies['title'].values

selected\_movie = st.selectbox(

    "Type or select a movie from the dropdown",

    movie\_list

)

Here we open and load our dataset generated by the recommendation system and then using a inbuilt selectbox to show movie name.

if st.button('Show Recommendation'):

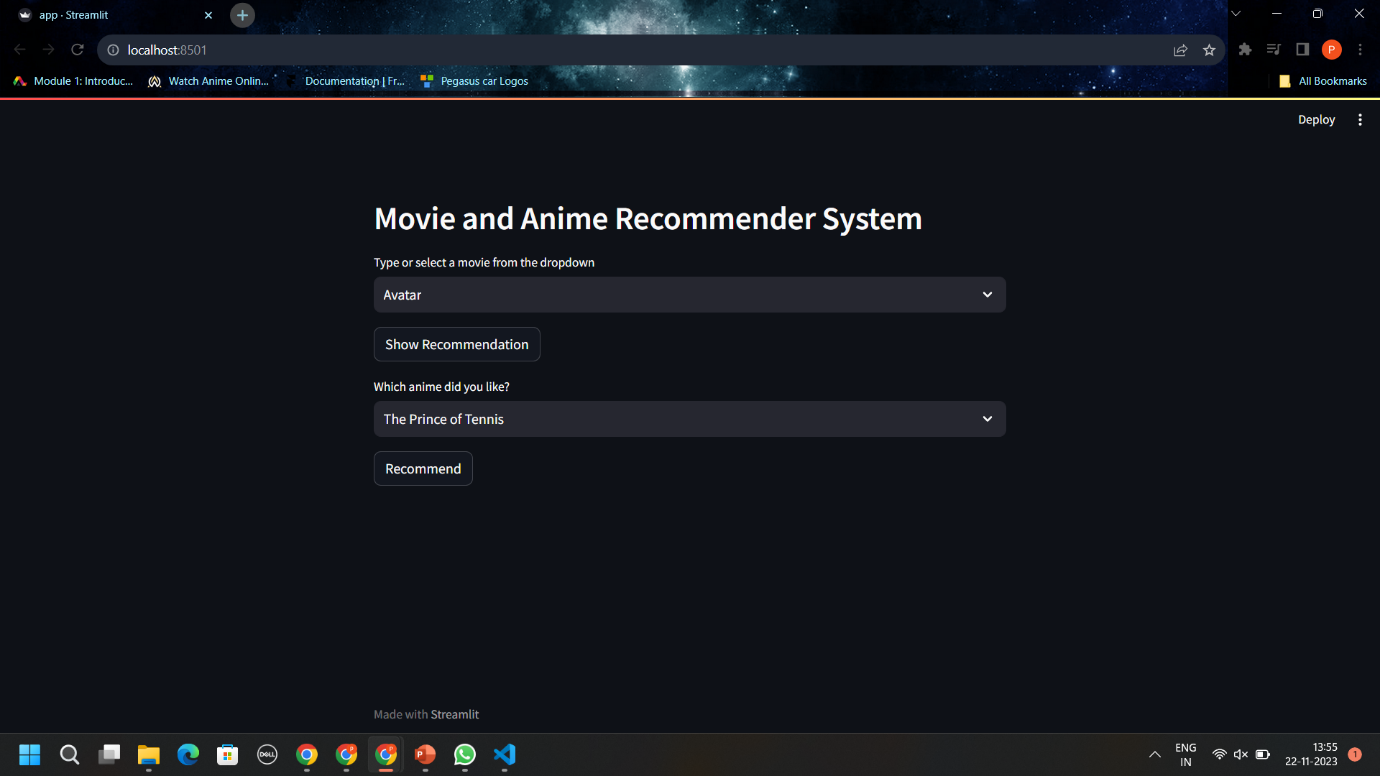
   recommended\_movie\_names= recommend(selected\_movie)

   for i in recommended\_movie\_names:

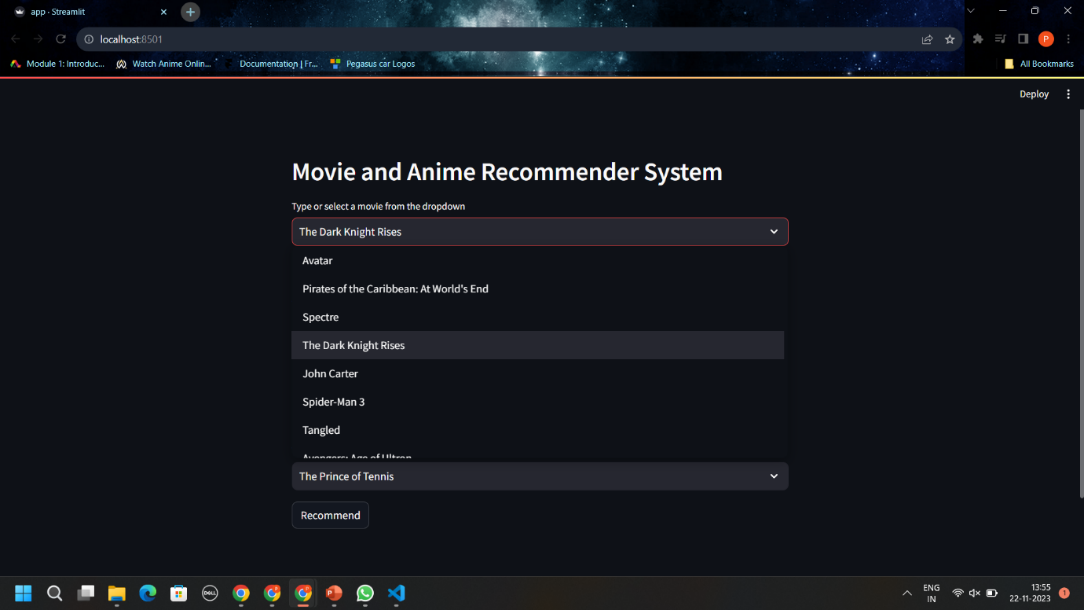
    st.write(i)

Here we have make a button on which when clicked will take the input movie and use the function recommend to find the most related movie.

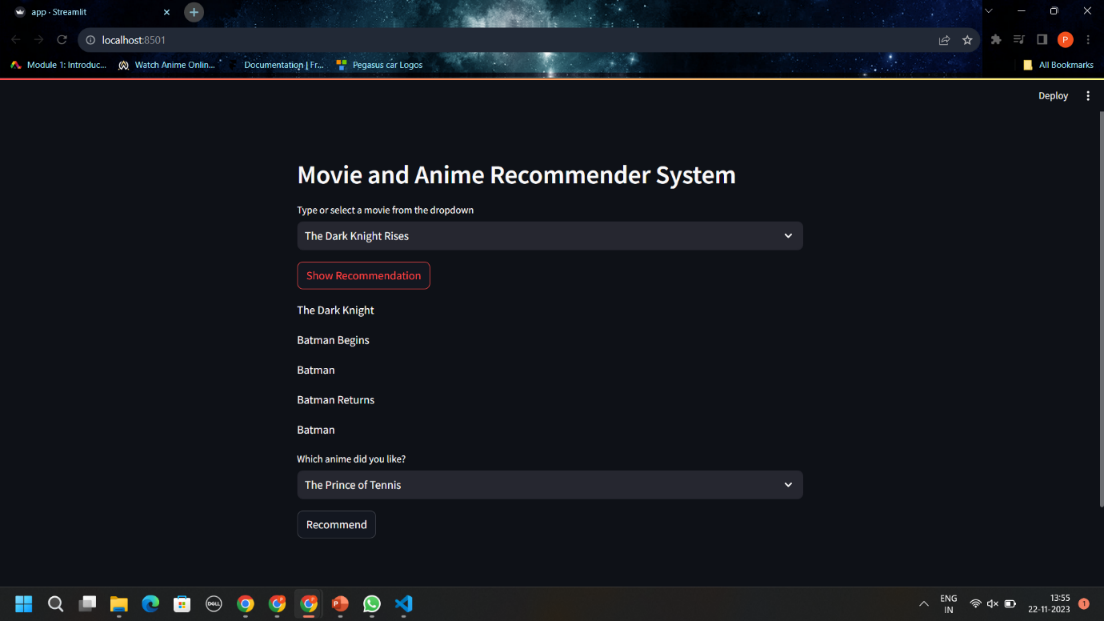
**WEBSITE**

****

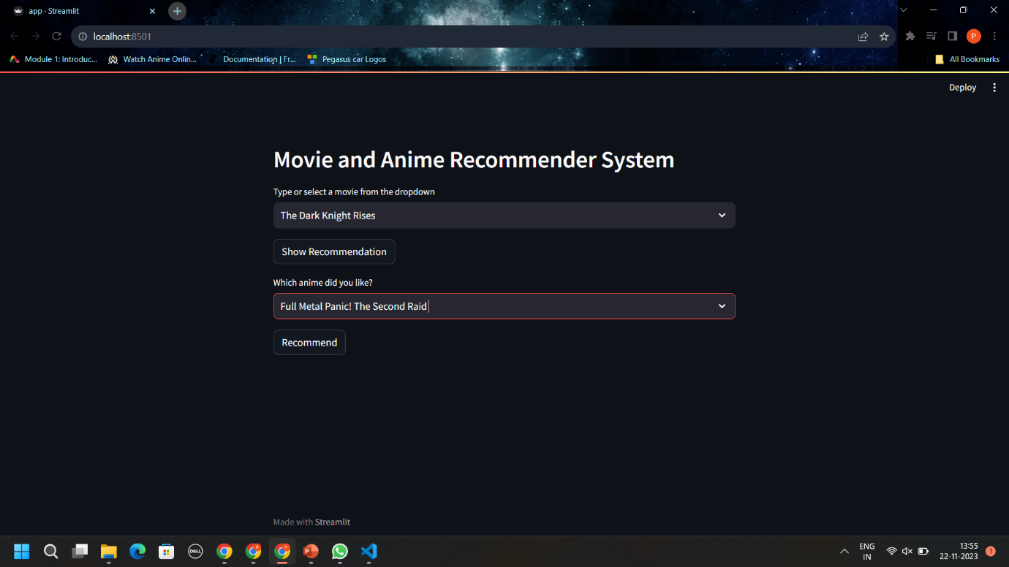
FRONT PAGE ON PC

****

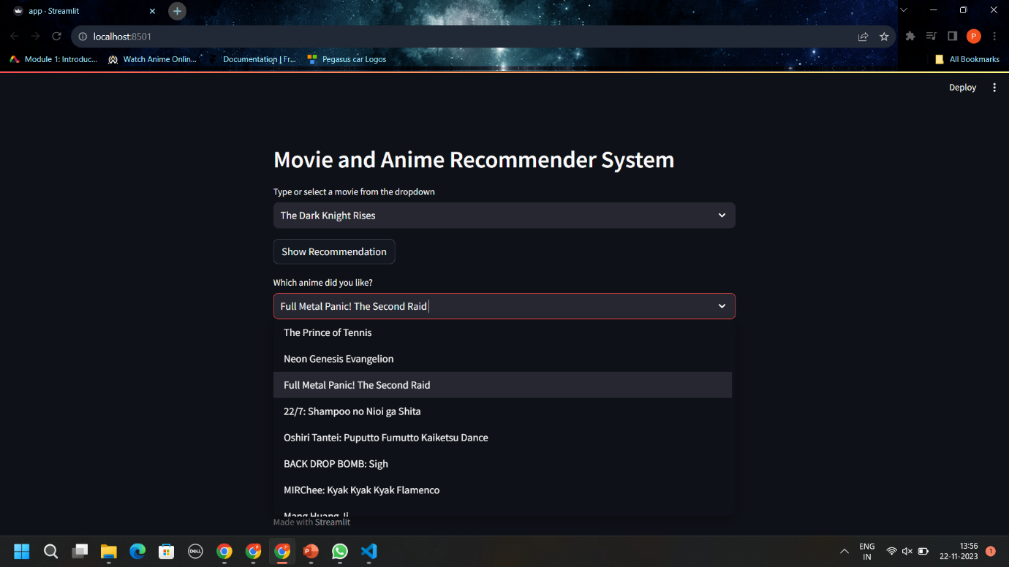
Movie input

****

Movies recommended

****

Anime input

****

Anime recommended

**FUTUTRE SCOPE**

The future of movie recommendation systems is bright, with advancements in artificial intelligence and personalized user experiences. Expect more sophisticated algorithms using deep learning, converging collaborative and content-based filtering for a holistic understanding of user preferences. Integration of contextual information, sentiment analysis, and emerging technologies like VR and AR may redefine the recommendation process. Interactive interfaces and ethical considerations for diversity and privacy will likely shape the evolution of movie recommendation systems, promising a more engaging and inclusive cinematic experience.

**CONCLUSION**

In conclusion, the future of movie recommendation systems is marked by a promising trajectory, propelled by advancements in AI, immersive technologies, and a nuanced understanding of user behavior. As these technologies converge, we anticipate more intelligent, context-aware algorithms providing personalized suggestions with a greater emphasis on user engagement. Striking the right balance between personalization and ethical considerations, such as inclusivity and privacy, will be pivotal. These evolving systems hold the potential to not only enhance the cinematic experience for audiences but also to navigate the challenges of content abundance in an ever-expanding streaming landscape. The future promises not just refined algorithms but a more interactive, diverse, and user-friendly approach to movie recommendations.