## Movie Recommendation System

### A Minor Project report for the evaluation and partial fulfilment of the requirement for the award of the degree

**INTEGRATED DUAL DEGREE B. TECH. (IT)**



Submitted By -

|  |  |
| --- | --- |
| **Names** | **Roll Number** |
| Aditya Aggarwal | (18/BIT/004) |
| Mohammad Anas | (18/BIT/032) |
| Mohit Gautam | (18/BIT/034) |
| Prasoon Srivastava | (18/BIT/041) |

UNDER THE SUPERVISION OF

Dr. AARTI GAUTAM DINKER

UNIVERSITY SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY GAUTAM BUDDHA UNIVERSITY

GREATER NOIDA –201312, GAUTAM BUDDHA NAGAR UTTAR PRADESH INDIA

11th Dec. 2021

Acknowledgment

We would like to express our special thanks to our Mentor Mr.Ankush Kumar Singla as well as our HOD Dr. Neeta Singh who gave us the opportunity to do this wonderful project on the topic “Movie Recommendation System” which also helped us in doing a lot of Research and also came to know about so many new things

Secondly we would also like to thank to all friends and teachers who helped us a lot in finalizing this project within the time frame.

By:- Team Members

Index

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Acknowledgment* | | | | | *....................................................................................................* | | | | | *1* |
| *Index* | *............................................................................................................................* | | | | | | | | | *2* |
| About the Project | | | | | ................................................................................................. | | | | | 3 |
| What are Recommendation systems? | | | | | | | | ......................................................... | | 4 |
| Use case of Recommendation systems | | | | | | | | ......................................................... | | 5 |
| What can be recommended? | | | | | | | ........................................................................... | | | 6 |
| Various types of recommendation systems | | | | | | | | | .............................................. | 8 |
| Content-based Filtering | | | | | | ................................................................................... | | | | 9 |
| Our Project | | | ........................................................................................................... | | | | | | | 11 |
| Future Direction | | | | ................................................................................................... | | | | | | 36 |
| Reference | | ................................................................................................................ | | | | | | | | 37 |

About the Project

The objective of this mini project is to let the students apply their programming knowledge into a real-world situation/problem and expose the students to how programming skills helps in developing a good engineer.

In this project, we are developing a Movie Recommendation System that will inquire to a movie database and recommend user some movies based on which movie the user has entered.

Thank you

What are Recommendation systems?

Recommender System is a system that seeks to predict or filter preferences according to the user’s choices. Recommender systems are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general.

It is among the most popular applications of data science. They are used to predict the Rating or Preference that a user would give to an item.

Almost every major company has applied them in some form or the other: **Amazon** uses it to suggest products to customers, **YouTube** uses it to decide which video to play next on auto play, and **Facebook** uses it to recommend pages to like and people to follow.

Use case of Recommendation systems

The aim of recommendation systems is just to recommend items to the customers for the purpose of upselling and cross-selling, and ultimately maximize profit. Another objective of the recommendation system is to achieve customer loyalty by providing relevant content and maximising the time spent by a user on your website or channel. This also helps in increasing customer engagement.

The key objectives of a recommendation system are

* They help the user find items of their interest
* Helps the item provider to deliver their items to the right user
  + To identify the most relevant products for each user
  + Showcase personalised content to each user
  + Suggest top offers and discounts to the right user
* Websites can improve user-engagement
* It increases revenues for business through increased consumption

What can be recommended?

Some of the things that can be recommended using a recommendation system are;

* Advertising Messages
* Movies
* Books
* Music Tracks
* News Articles
* Restaurants
* Courses in e-learning
* Jobs
* TV Programs
* Clothes
* Supermarket Goods, etc.

**Real-World examples**

Here are some of the examples of the pioneers in creating algorithms for recommendation systems and using them to serve their customers better in a personalized manner. These are

* **GroupLens:**
  + Helped in developing initial recommender systems by pioneering collaborative filtering model
  + It also provided many data-sets to train models including MovieLens and BookLens
* **Amazon:**
  + Implemented commercial recommender systems
  + They also implemented a lot of computational improvements
* **Netflix Prize:**
  + Pioneered Latent Factor/ Matrix Factorization models
* **Google-Youtube:**
  + Hybrid Recommendation Systems
  + Deep Learning based systems
  + Social Network Recommendations

Various types of recommendation systems

A recommender system produces a list of recommendations in any of the two ways;

* Collaborative filtering: Collaborative filtering approaches build a model from the user’s past behavior (i.e. items purchased or searched by the user) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that users may have an interest in.
* Content-based filtering: Content-based filtering approaches uses a series of discrete characteristics of an item in order to recommend additional items with similar properties. Content-based filtering methods are totally based on a description of the item and a profile of the user’s preferences. It recommends items based on the user’s past preferences.

Content-Based Filtering

A Content-Based Recommender works by the data that we take from the user, either explicitly (rating) or implicitly (clicking on a link). By the data we create a user profile, which is then used to suggest to the user, as the user provides more input or take more actions on the recommendation, the engine becomes more accurate.

**Components of Content-Based Filtering**

* User Profile:

In the User Profile, we create vectors that describe the user’s preference. In the creation of a user profile, we use the utility matrix which describes the relationship between user and item. With this information, the best estimate we can make regarding which item user likes, is some aggregation of the profiles of those items.

* Item Profile:

In Content-Based Recommender, we must build a profile for each item, which will represent the important characteristics of that item.

For example, if we make a movie as an item then its actors, director, release year and genre are the most significant features of the movie. We can also add its rating from the IMDB (Internet Movie Database) in the Item Profile.

* Utility Matrix:

Utility Matrix signifies the user’s preference with certain items. In the data gathered from the user, we have to find some relation between the items which are liked by the user and those which are disliked, for this purpose we use the utility matrix. In it we assign a particular value to each user-item pair, this value is known as the degree of preference. Then we draw a matrix of a user with the respective items to identify their preference relationship.

**Approaches for recommending Items to User Based on Content:**

* Method 1:

We can use the cosine distance between the vectors of the item and the user to determine its preference to the user. For explaining this, let us consider an example:

We observe that the vector for a user will have a positive number for actors that tend to appear in movies the user likes and negative numbers for actors user doesn’t like, Consider a movie with actors which user likes and only a few actors which user doesn’t like, then the cosine angle between the user’s and movie’s vectors will be a large positive fraction. Thus, the angle will be close to 0, therefore a small cosine distance between the vectors.

It represents that the user tends to like the movie, if the cosine distance is large, then we tend to avoid the item from the recommendation.

* Method 2:

We can use a classification approach in the recommendation systems too, like we can use the Decision Tree for finding out whether a user wants to watch a movie or not, like at each level we can apply a certain condition to refine our recommendation.

Our Project

In this project we have used **Method 1 Content-Based Filtering Recommendation System** to develop our Movie Recommendation System.

The method 1 content-based filtering recommendation system eliminates the need for knowing other factors like user browsing history, user preferences, and other factors. Hence, the factors considered are the star ratings, the star cast of the movie, genre, directors, etc. to generate a scalable recommendation system. This increases the chances of user engagement as compared to when there was no recommendation system.

**Code**

**Program**

import numpy as np

import pandas as pd

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

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

movies.head(1)

credits.head(1)

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

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

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

movies.head(1)

movies.info()

# genres

# id

# keywords

# overview

# popularity (not for our use)

# release\_date (not for our use)

# title

# cast

# crew

​

movies = movies[['id','title','overview','genres','keywords','cast','crew']]

# new movies data of 7 column

movies.head()

#### New data frame only 3 column ####

id | title| tags

tags will come after merging all 5 column

then perform data preprocessing in tags(remove duplicate, only top 3 cast, in crew only director )

# missing data finding

movies.isnull().sum()

# drop or delete those movies which have no overview

movies.dropna()

movies.dropna(inplace=True)

movies.isnull().sum() #now no null values in our data

# Check duplicate data

movies.duplicated().sum()

# o/p says no duplicates in data

# data of genres column

movies.iloc[0].genres

# o/p says it is in list of dictionary (wiered)

# from '[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]'

# we want ["Action","Adventure","Fantasy","Science Fiction"]

import ast

# create helper function named convert

def convert(obj):

L=[]

for i in ast.literal\_eval(obj): # loop works in upper dictionaries like {A:345,B:324,C:"dgdg"} then so on

L.append(i['name']) # in one dictionary we only take only "name" not "id"

return L

convert('[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]')

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

movies.head()

# o/p shows complexity of genres column is now solved

# Now apply same thing for for keywords column

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

movies.head()

# o/p shows complexity of keywords column is now solved

movies['cast'][0]

# Now apply same thing for cast column but we only go in 3 dictionaries(bcz we only take 3 main actors)

​

# create helper function named convert

def convert3(obj):

L=[]

counter = 0

for i in ast.literal\_eval(obj): # loop works in upper dictionaries like {A:345,B:324,C:"dgdg"} then so on

if counter != 3:

L.append(i['name']) # in one dictionary we only take only "name" not "id"

counter+=1

else:

break

return L

# cast is string of list and we need in integer i.e. why it give error in movies['cast'].apply(convert3)

# without ast function

movies['cast'] = movies['cast'].apply(convert3)

movies.head()

# o/p shows complexity of cast column is now solved

movies['crew'][0] # we need only director name

# Now apply same thing for crew column but we only go in where job is director and its name in output

​

# create helper function named convert movies['cast'].apply(convert3)

def fetch\_director(obj):

L=[]

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

if i['job'] == 'Director':

L.append(i['name'])

break

return L

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

movies.head()

movies['overview'][0]

# o/p shows it is string

# convert string to list for easy in concatenation

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

movies.head()

# Removing space b/w one tags like Sam Worthington to SamWorthington

movies['genres'] = movies['genres'].apply(lambda x:[i.replace(" ","") for i in x])

movies['keywords'] = movies['keywords'].apply(lambda x:[i.replace(" ","") for i in x])

movies['cast'] = movies['cast'].apply(lambda x:[i.replace(" ","") for i in x])

movies['crew'] = movies['crew'].apply(lambda x:[i.replace(" ","") for i in x])

movies.head()

# Make one column Tags by concatenation of genres,keywords,cast and crew

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

​

# Make new table

new\_df = movies[['id','title','tags']]

new\_df

# now convert tags list datatypes to string

new\_df['tags'] = new\_df['tags'].apply(lambda x:" ".join(x)) # join at space

new\_df

new\_df['tags'][0]

new\_df['tags'][1]

# convert it into lower case

new\_df['tags'] = new\_df['tags'].apply(lambda x:x.lower())

new\_df.head()

new\_df['tags'][1]

# come here after the text vectorization for apply stamming \*\*\*(^)\*\*\*

!pip install nltk

import nltk

from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

# make helper function

# first convert every list to string then stem every word

​

​

def stem(text):

y= []

for i in text.split():

y.append(ps.stem(i))

return " ".join(y)

# Now apply sremmering for table new\_df's column named tags

new\_df['tags'] = new\_df['tags'].apply(stem)

Now apply Text vectorization and doesn't consider stop words

# Removing the stop words

from sklearn.feature\_extraction.text import CountVectorizer

# make object cv

# (max\_features = 5000 ) means Total number of common tags taken

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

# list of common tags or words in corpus

​

# One problem is that many same word is also come (accept,accepts,accepted)

# Solution : Apply stamming ( accept,accept,accepts)

# apply it to whole tags column big data \*\*\*(^)\*\*\*

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

# o/p shows all movies are convert into vectors

​

cv.fit\_transform(new\_df['tags']).toarray().shape

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

vectors[0]

cv.get\_feature\_names()

len(cv.get\_feature\_names())

# 5000 common words which have frequency

# Apply stammer function

ps.stem('loved')

ps.stem('walking')

ps.stem("captain barbossa, long believed to be dead, has come back to life and is headed to the edge of the earth with will turner and elizabeth swann. but nothing is quite as it seems. adventure fantasy action ocean drugabuse exoticisland eastindiatradingcompany loveofone'slife traitor shipwreck strongwoman ship alliance calypso afterlife fighter pirate swashbuckler aftercreditsstinger johnnydepp orlandobloom keiraknightley goreverbinski")

from sklearn.metrics.pairwise import cosine\_similarity #simalirity is opposite of distance and in b/w 0-1

cosine\_similarity(vectors)

# o/p shows diagonal is always 1

# means similarity with itself

cosine\_similarity(vectors).shape

# distance of 1'st vector with other 4806 vectors

# distance of 2'nd vector with other 4806 vectors

# distance of 3'rd vector with other 4806 vectors

# .

# .

# distance of 4806'th vector with other 4806 vectors

similarity = cosine\_similarity(vectors)

similarity[0]

# o/p shows distance of 1'st vector with other 4806 vectors

sorted(similarity[0],reverse=True)

# we need index also, this with this

similarity = cosine\_similarity(vectors)

​

list(enumerate(similarity[0]))

# perform sorting on the basis of similarity not indexes

sorted(list(enumerate(similarity[0])),reverse=True,key=lambda x:x[1])

sorted(list(enumerate(similarity[0])),reverse=True,key=lambda x:x[1])[1:6]

# function, i/p -> any movie

# o/p -> recommend any near similar 5 movies

# sol. or algo -> 1) from movie find its index position number(called movie\_index) from the table

# 2) find its distance or similary with other 4806 movies

# 3) sort in descending order(perform above 3 rows) and fetch top 5 similar movies(first index then movie)

def recommend(movie):

movie\_index = new\_df[new\_df['title'] == movie].index[0]

distances = similarity[movie\_index]

movies\_list = sorted(list(enumerate(distances)),reverse=True, key=lambda x: x[1])[1:6]

for i in movies\_list:

print(new\_df.iloc[i[0]].title)

recommend("Falcon Rising")

# print(i[0])

# new\_df.iloc[1216].title

**DEVELOPMENT**

import pickle

# first we convert it into dictionary then send it into pycharm project

new\_df.to\_dict()

pickle.dump(new\_df.to\_dict(),open('movie\_dict.pkl','wb')) # for opening new file

pickle.dump(similarity,open('similarity.pkl','wb'))

**Development in pycharm**

import pickle

import streamlit as st

import requests

def fetch\_poster(movie\_id):

url = "https://api.themoviedb.org/3/movie/{}?api\_key=8265bd1679663a7ea12ac168da84d2e8&language=en-US".format(movie\_id)

data = requests.get(url)

data = data.json()

poster\_path = data['poster\_path']

full\_path = "https://image.tmdb.org/t/p/w500/" + poster\_path

return full\_path

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 = []

recommended\_movie\_posters = []

for i in distances[1:6]:

# fetch the movie poster

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

recommended\_movie\_posters.append(fetch\_poster(movie\_id))

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

return recommended\_movie\_names,recommended\_movie\_posters

st.header('Movie Recommender System')

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

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

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

selected\_movie = st.selectbox(

"Type or select a movie from the dropdown",

movie\_list

)

if st.button('Show Recommendation'):

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

col1, col2, col3, col4, col5 = st.beta\_columns(5)

with col1:

st.text(recommended\_movie\_names[0])

st.image(recommended\_movie\_posters[0])

with col2:

st.text(recommended\_movie\_names[1])

st.image(recommended\_movie\_posters[1])

with col3:

st.text(recommended\_movie\_names[2])

st.image(recommended\_movie\_posters[2])

with col4:

st.text(recommended\_movie\_names[3])

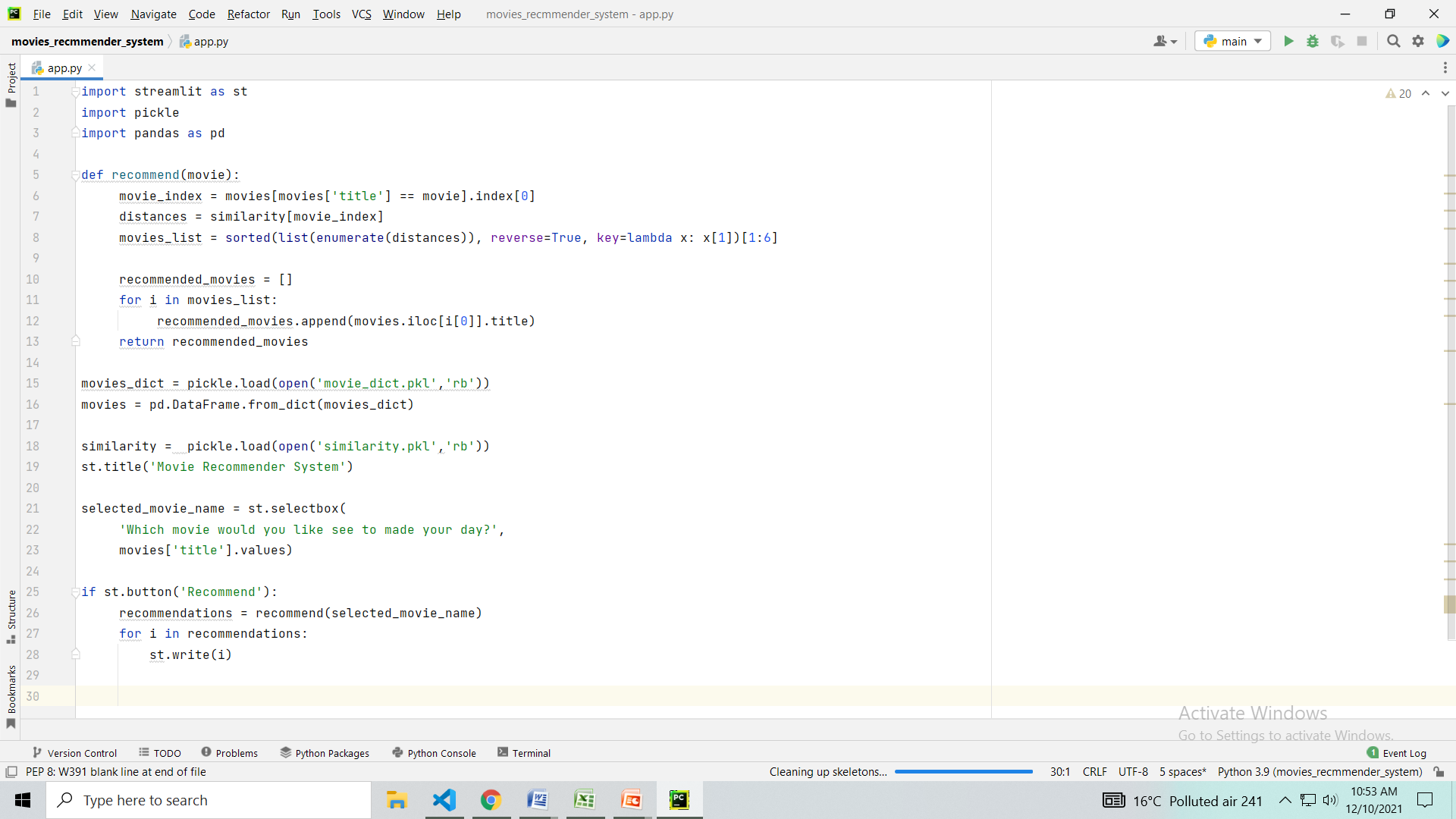
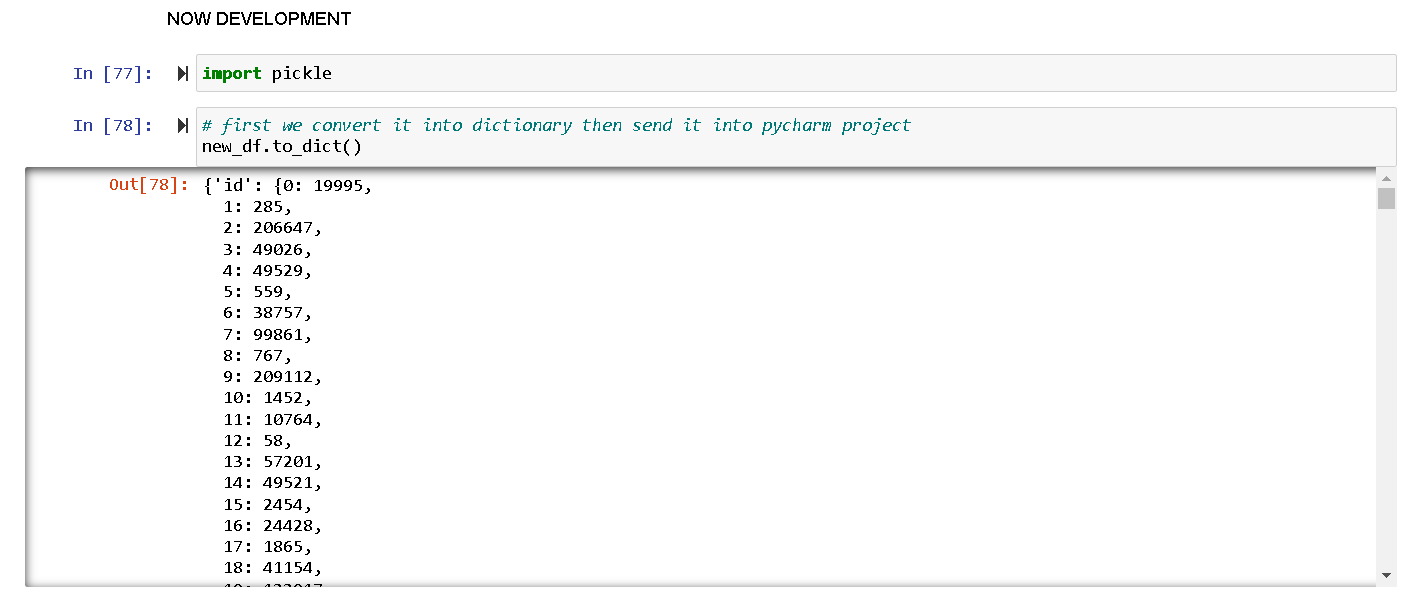
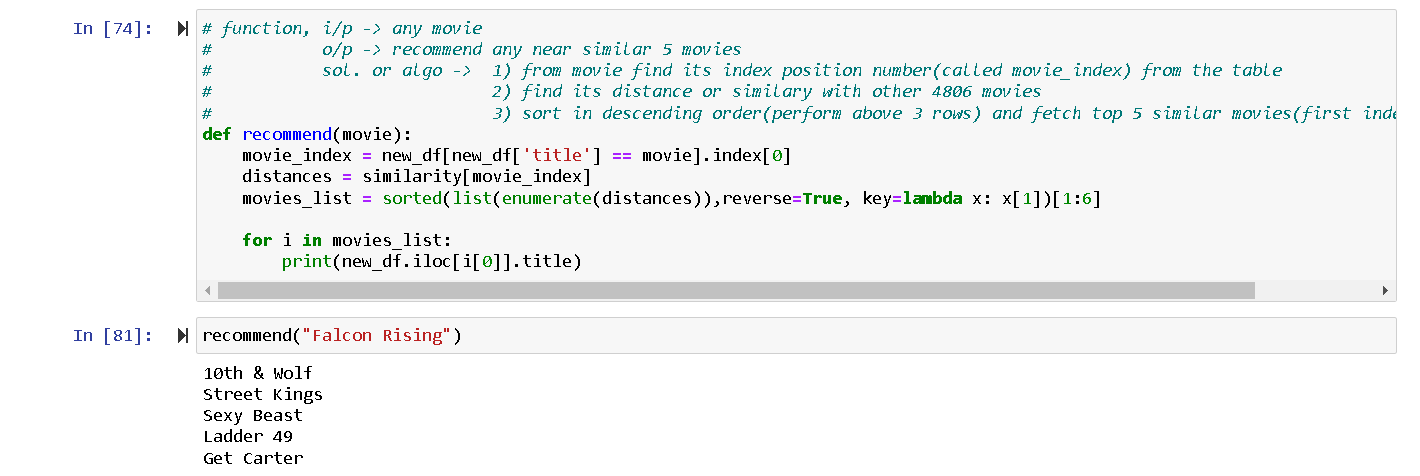
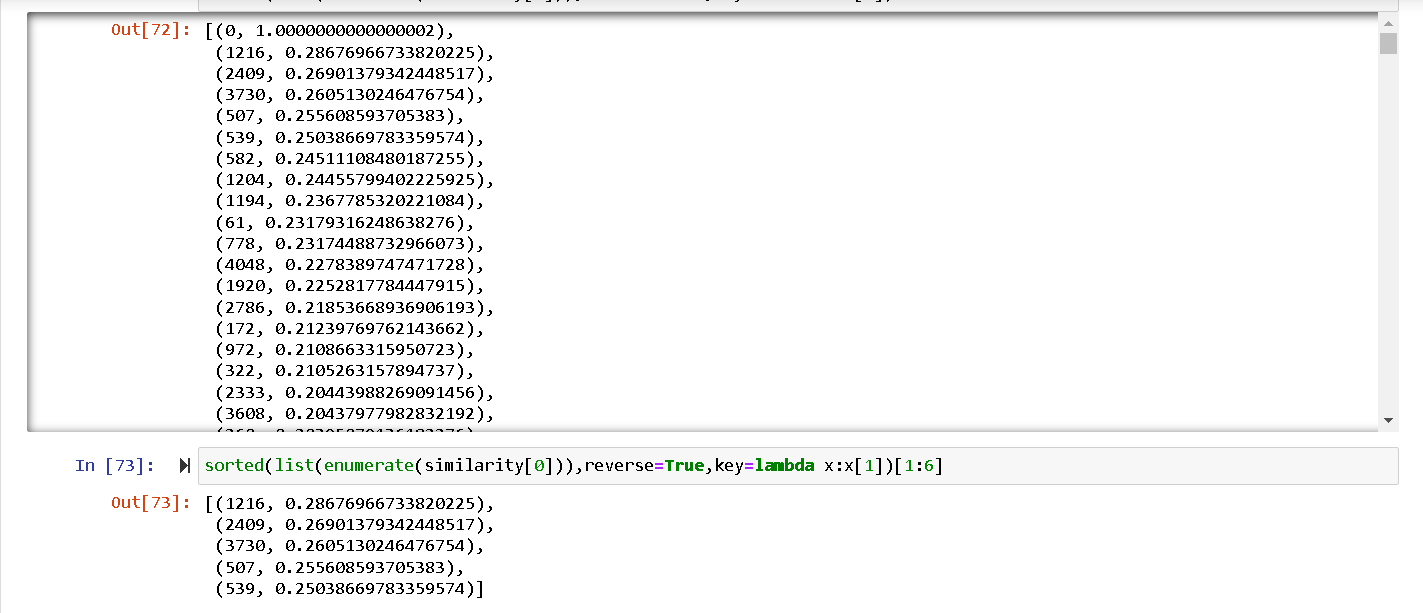
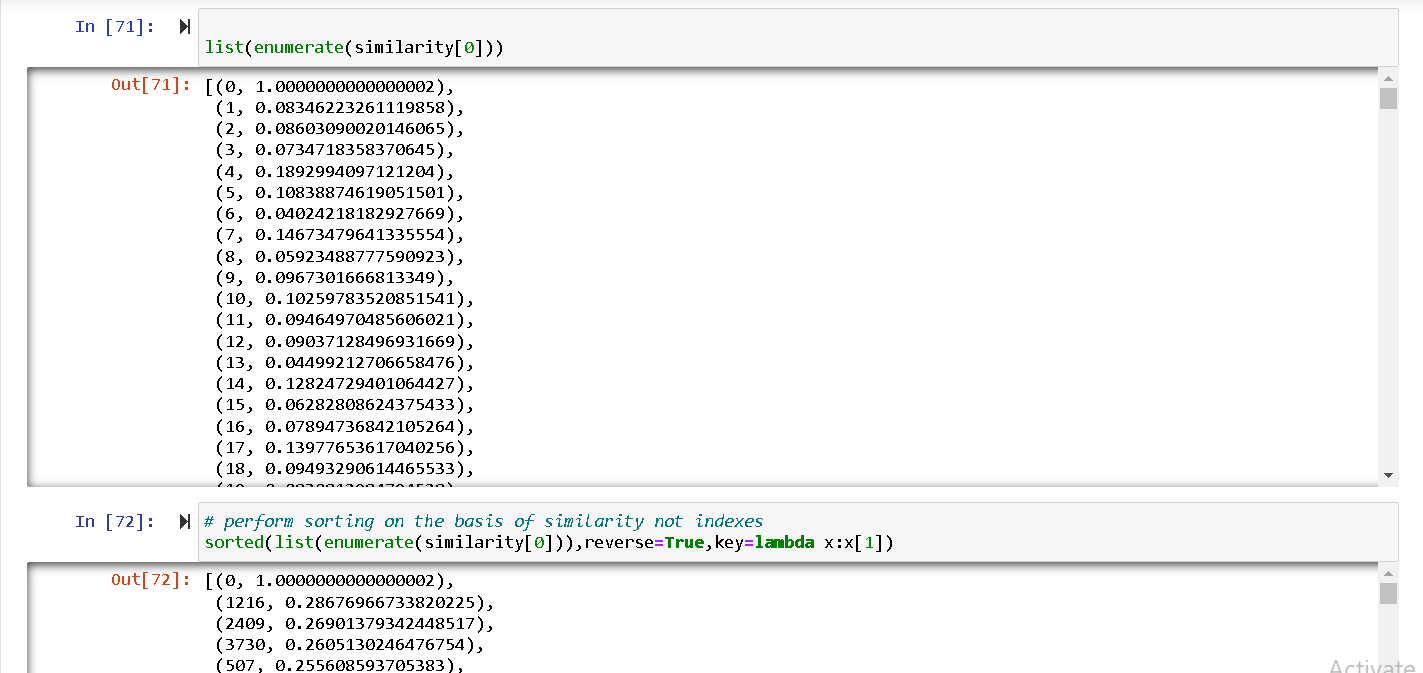
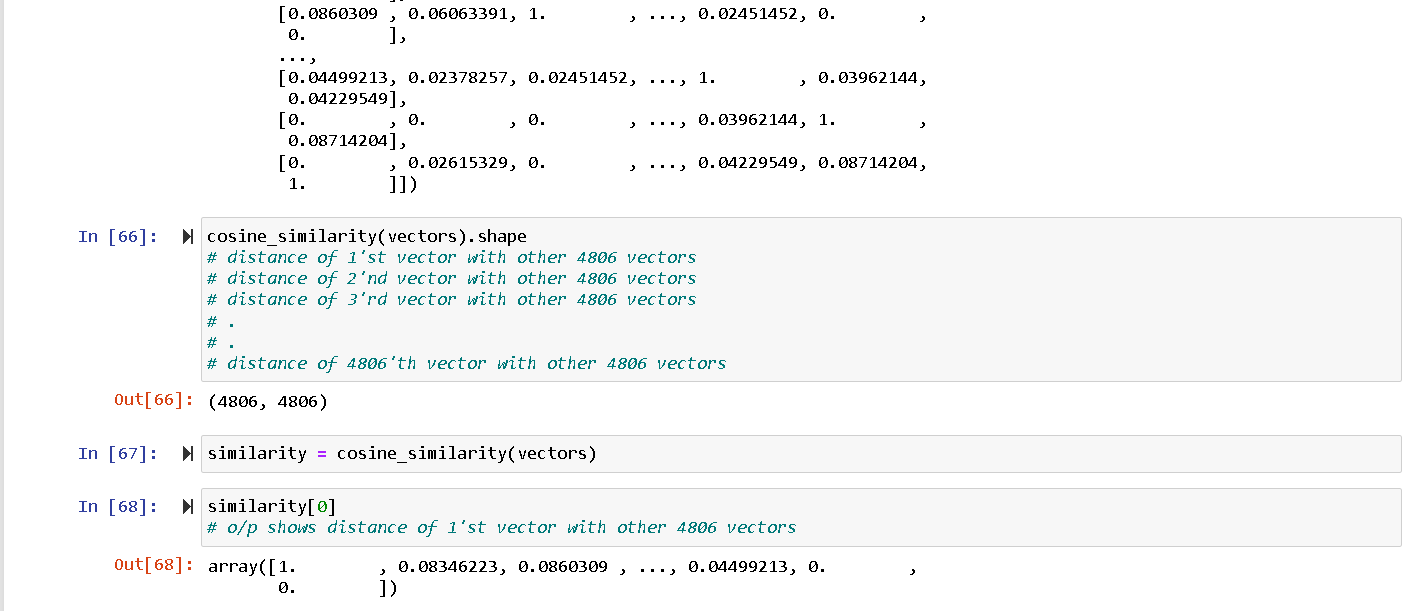
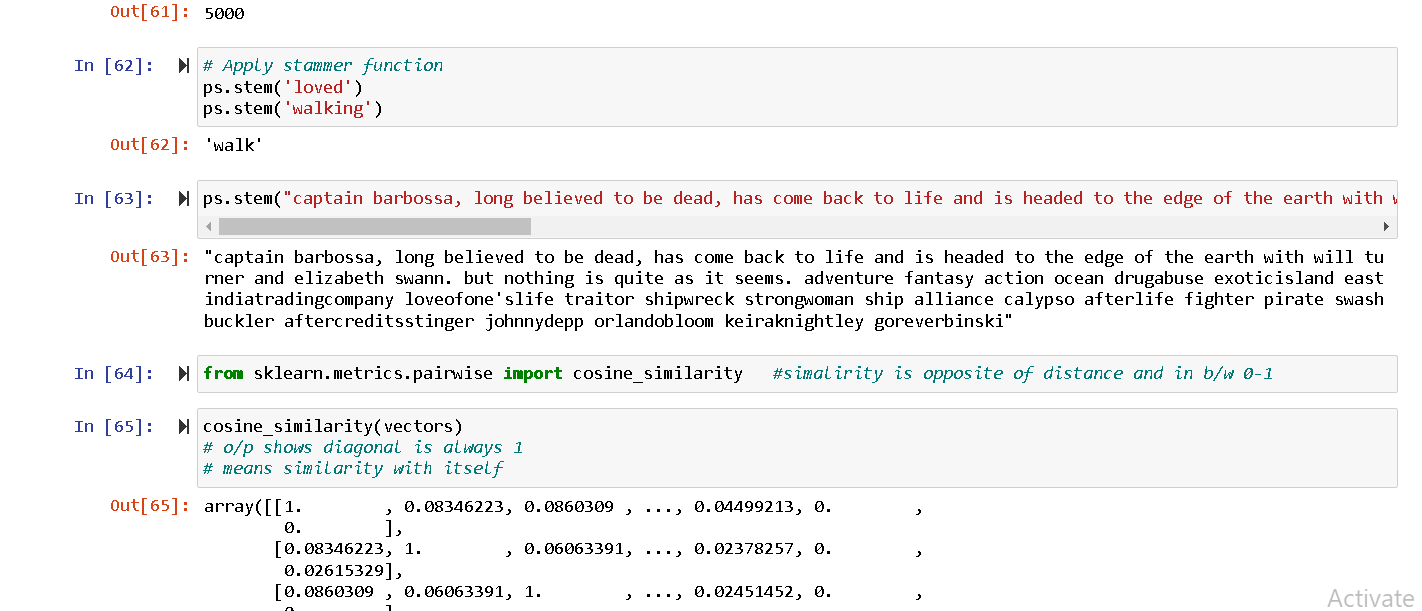
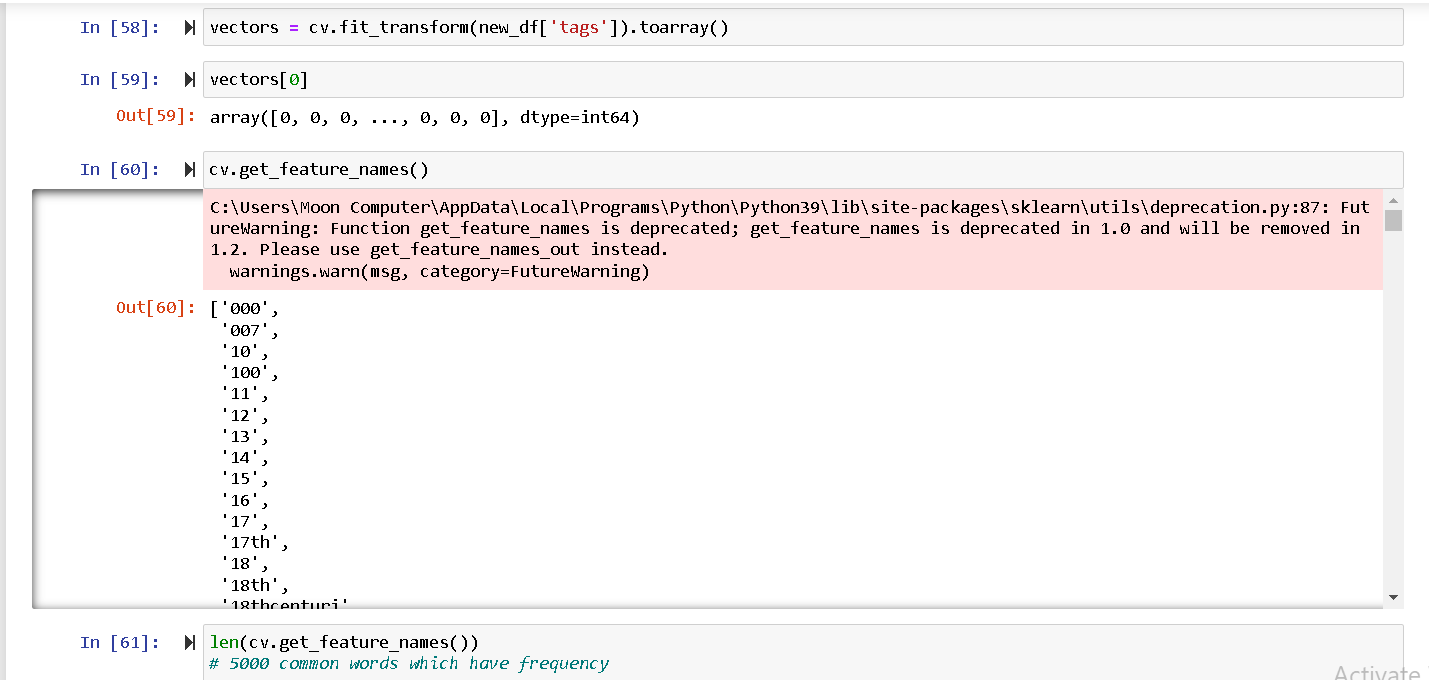
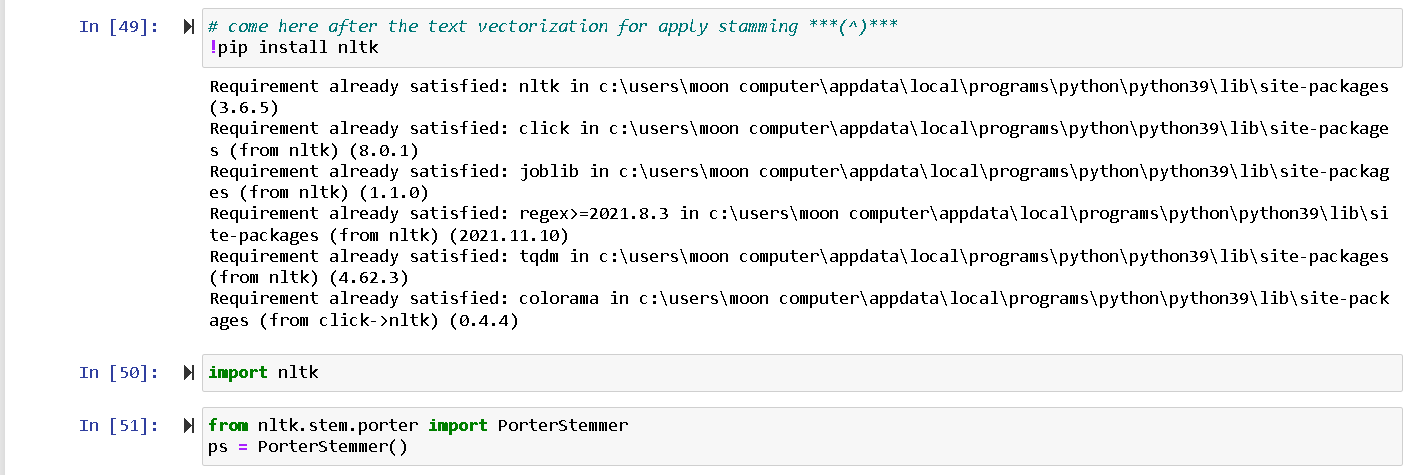
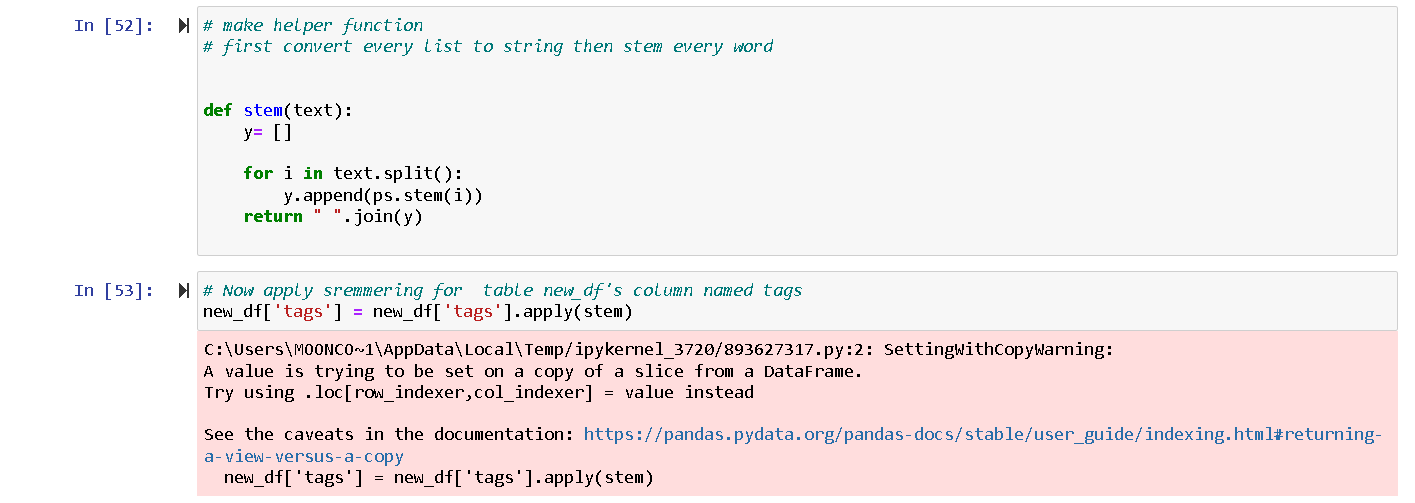
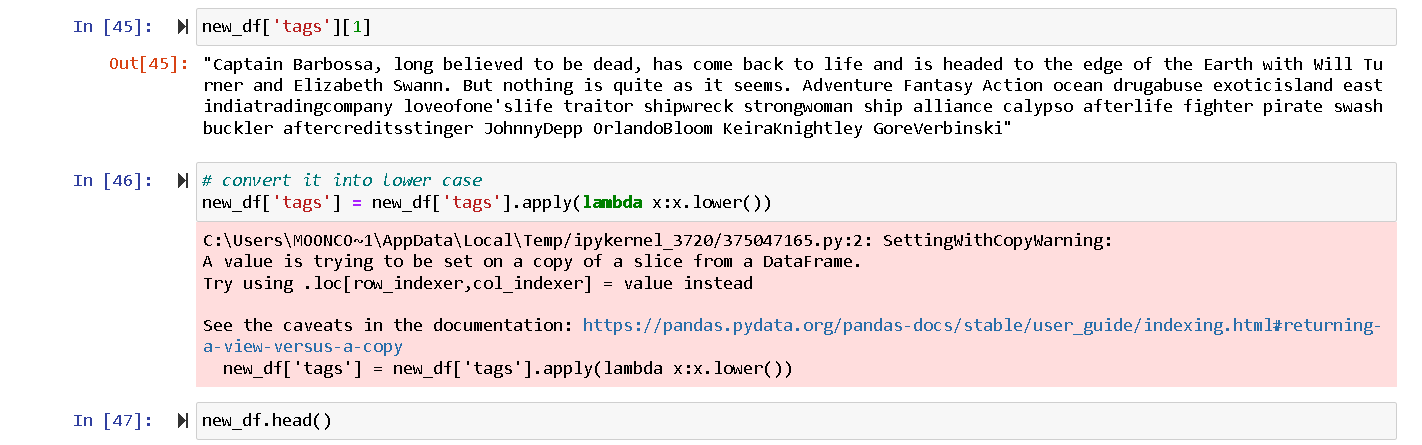
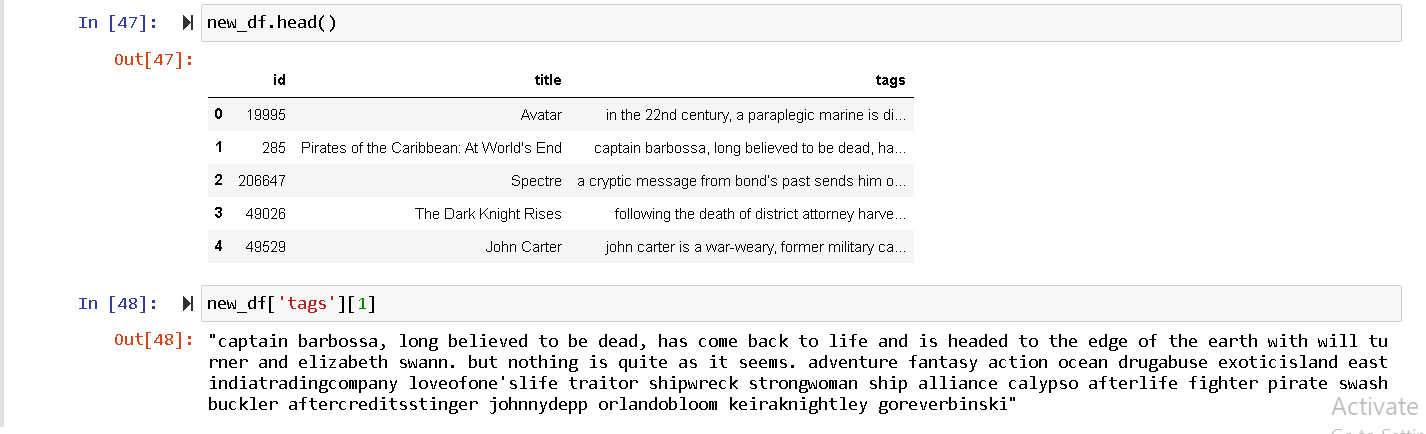
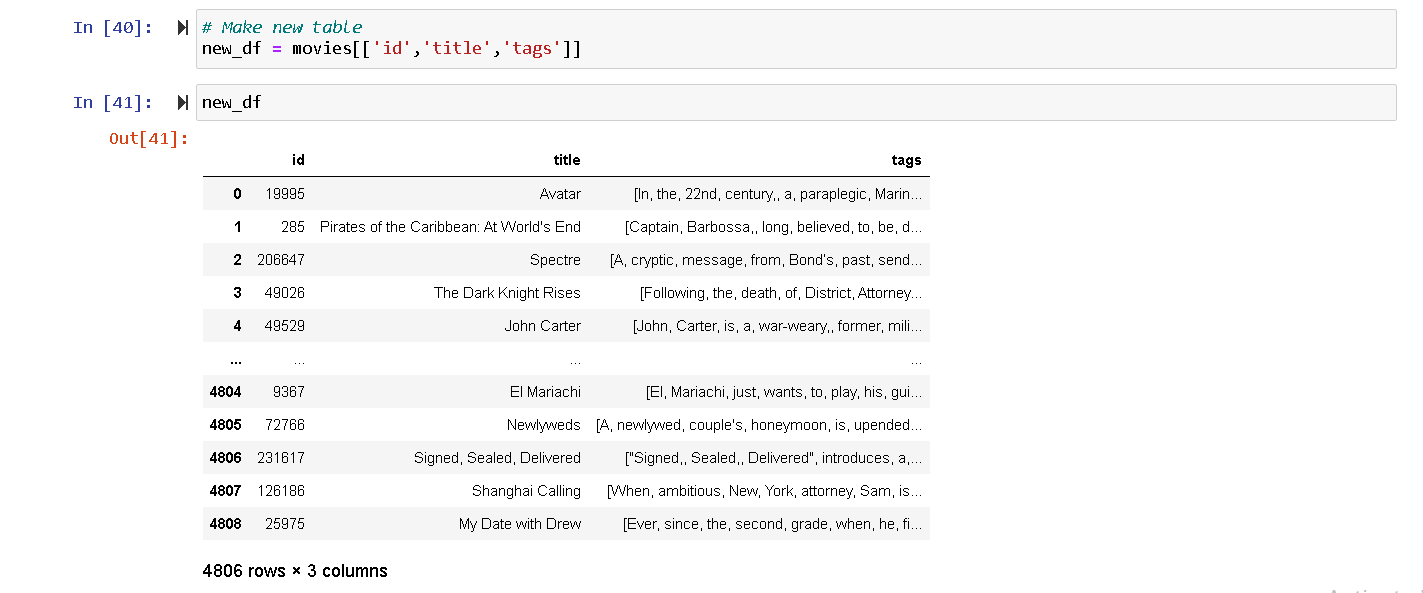
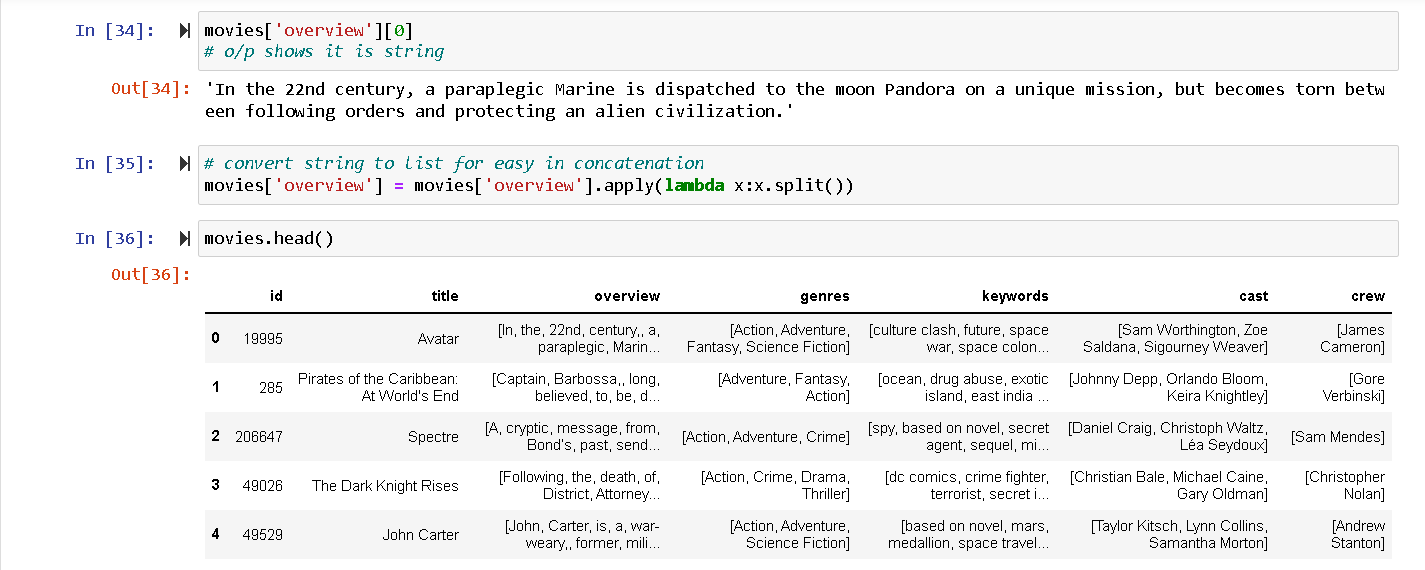
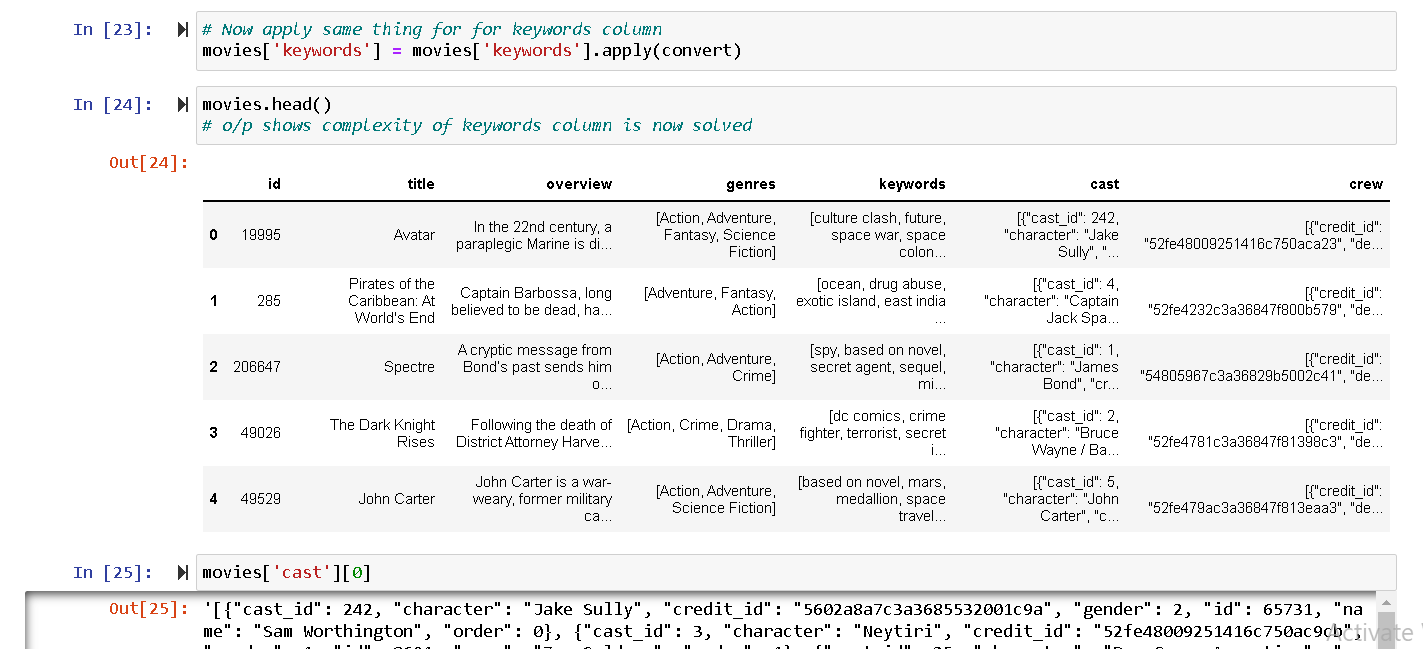
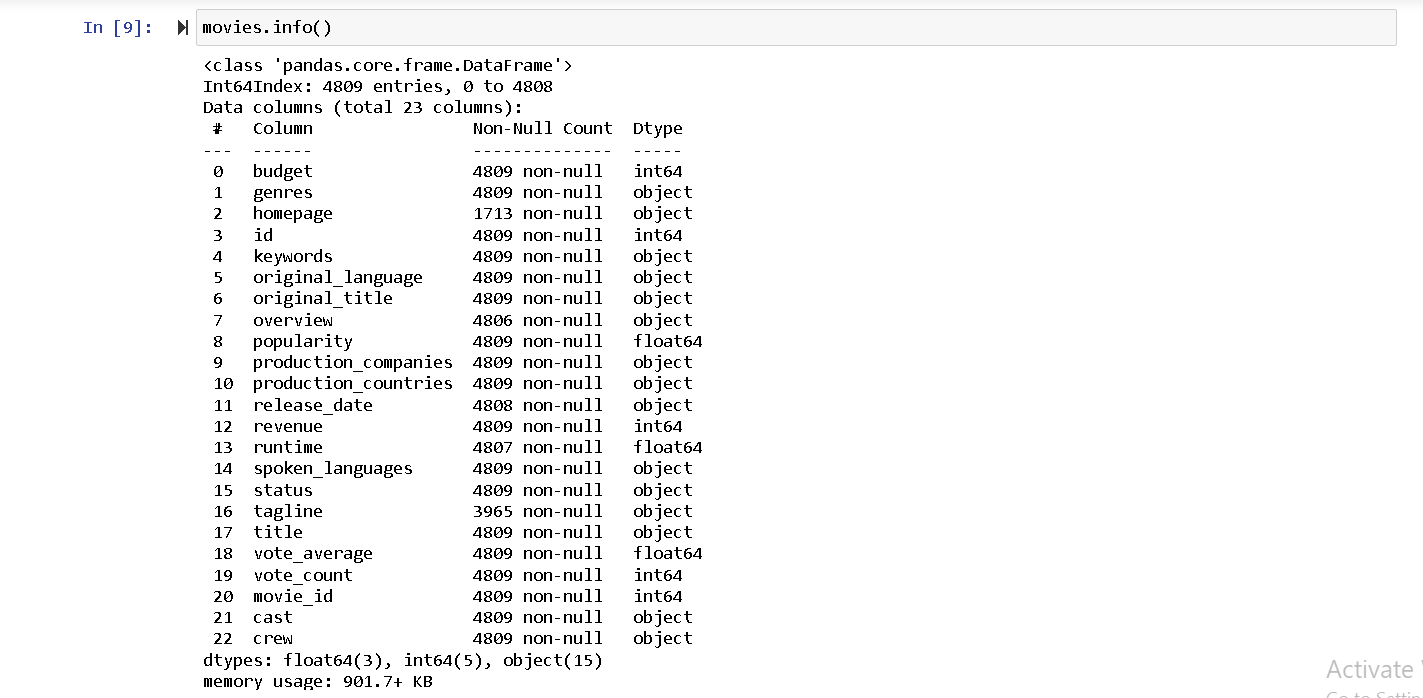
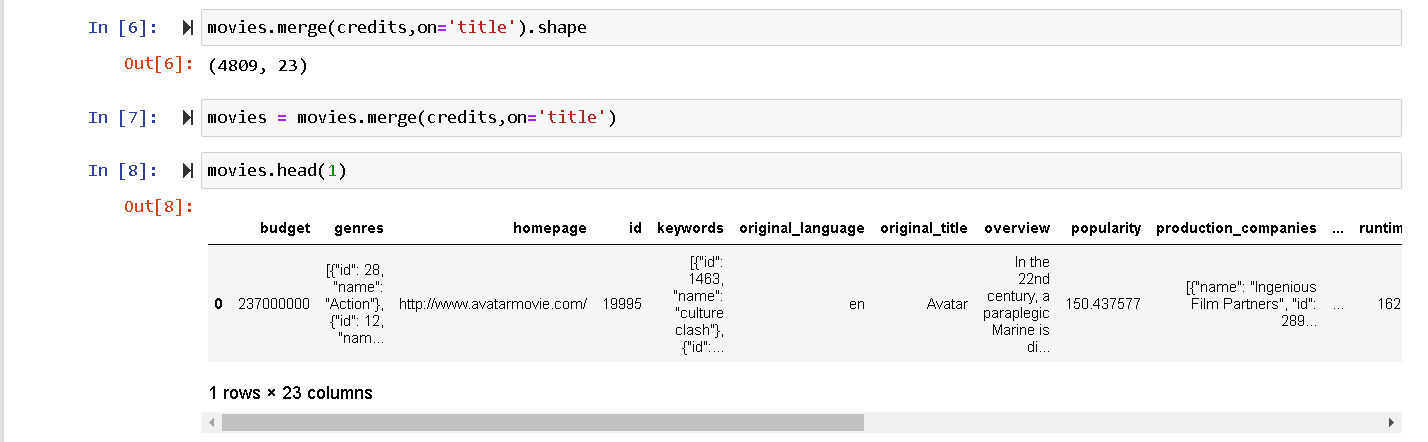
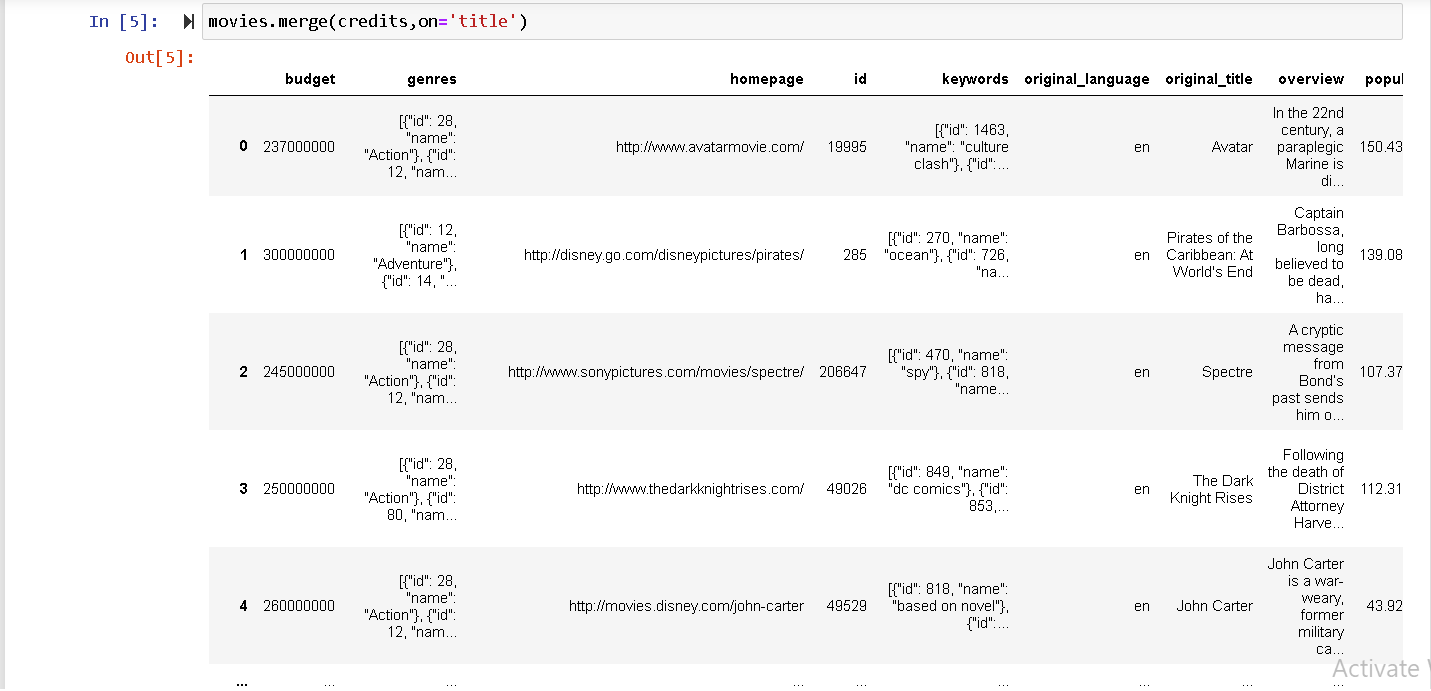
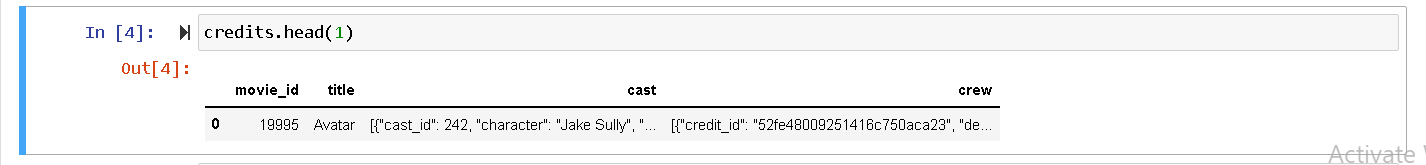
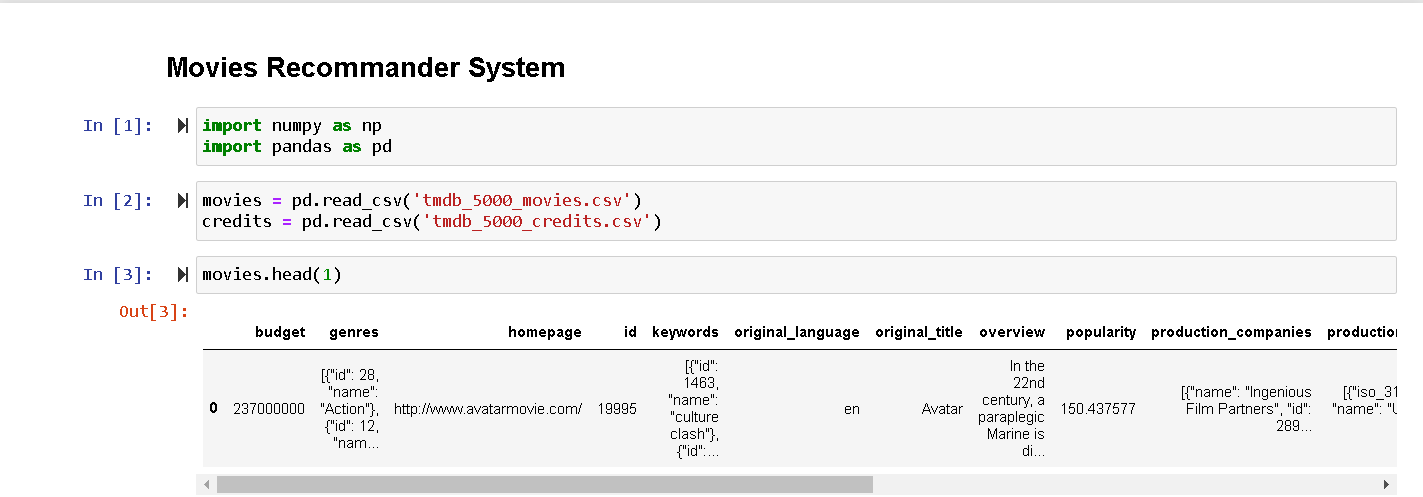
st.image(recommended\_movie\_posters[3])

with col5:

st.text(recommended\_movie\_names[4])

st.image(recommended\_movie\_posters[4])

**Screenshots**



Future Direction

There are various things that can be done in order to improve our project some of which that came to us as possible directions to work on are;

* Hybrid development
* Improve the site inter face
* Applying high efficiency security
* Improving the database and adding some more attributes like sites and trailers links, etc.
* Increase the number of recommendations and why is it recommended.

Reference

1. <https://www.geeksforgeeks.org>
2. <https://www.mygreatlearning.com/blog/masterclass-on-movie-recommendation-system/>
3. <https://www.kaggle.com/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv>