****

**(Project Final Report)**

**Group Members: -**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Name** | **Roll No.** |
| **1** | **Usama Asif** | **191370237** |
| **2** | **Taha** | **191370164** |

**Project title: -**

Movie Recommendation System

**Submitted to: -**

Sir Qamar Askari

**Subject: -**

Machine Learning

**Movie Recommendation System:**

A movie recommendation system, also known as a movie recommender system, uses machine learning (ML) to predict or filter users’ film preferences based on their prior decisions and actions. It is an advanced filtration system that anticipates the consumer in question’s potential like selection for a domain-specific item a movie.

**Working of Movie Recommendation System:**

A movie recommendation system fundamentals idea is pretty straightforward. Every recommender system primarily consists of two components **User and Movie Items.** Users receive more predictions from the system and the actual movies are the products. **Filtering and predicting** only the movies that a matching user is most likely to wish to see is the main objective of a movie recommendation system. The user information from the system database is used by the ML algorithm for these recommendation systems based on information from the past. This data is used to forecast the user in question’s behavior in the future. Data should be handled by experts because it is so crucial to ML projects including the movie recommendation system.

**Filtration Strategies for Movie Recommendation System:**

To assist in finding the most relevant films movie recommendation system employ a variety of filtration techniques and algorithm. The **content-based filtering and collaborative filtering** system subcategories of the ML algorithm used for the movie recommendations are the most well-liked ones.

**Content-Based Filtering:**

A method of filtering movies in a movie recommendation system that makes advantage of the item data. This information which is taken from just one user is quite important in this case. This technique uses an ML algorithm to suggest movies that are comparable to the users’ past choices. Therefore the information about the prior movie choices and likes of just one person is used to generate similarity in content-based filtering.

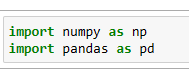
**Collaborative Filtering:**

These filtering techniques are based on the interaction between the relevant person and the other user. For the best outcomes the system contrast and compare these behaviors it combines the film choices and user patterns of several people. Thereare two types of collaborative filtering algorithms. The first one is **Collaborative Filtering Based on Users** the goal is to find patterns in target users and other database users like movie preferences the second one is **Collaborative Item Based Filtering** the fundamental idea behind this is to find comparable products like movies that target users rate or interact with so after seeing the filtering strategies for the movie recommendation system.

**Project Flow:**



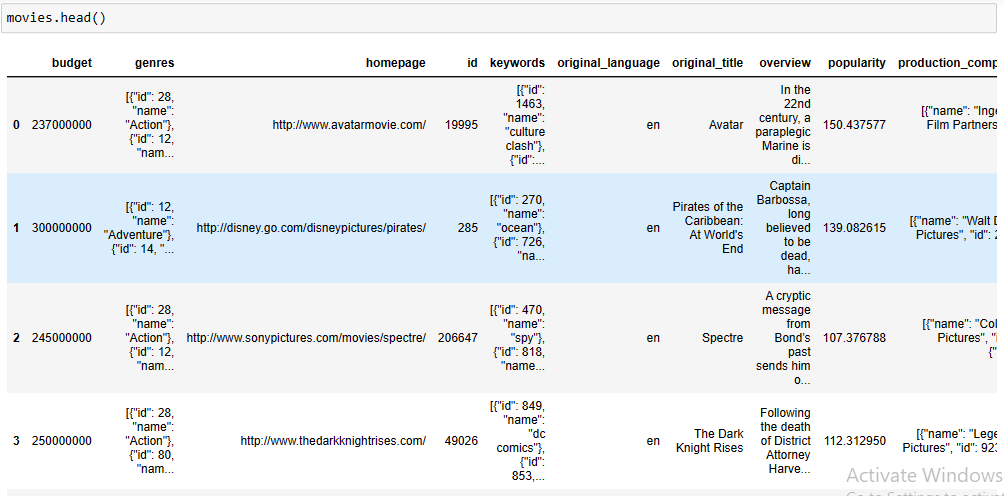
**Report:**



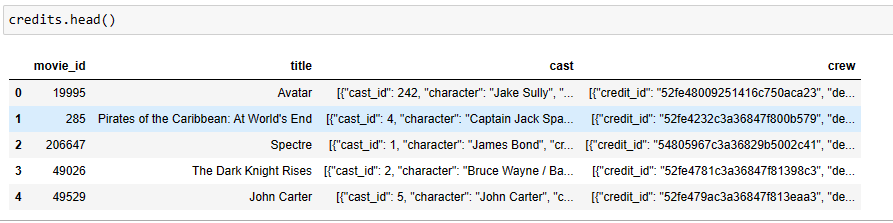
* First of all import two libraries Numpy and Pandas.



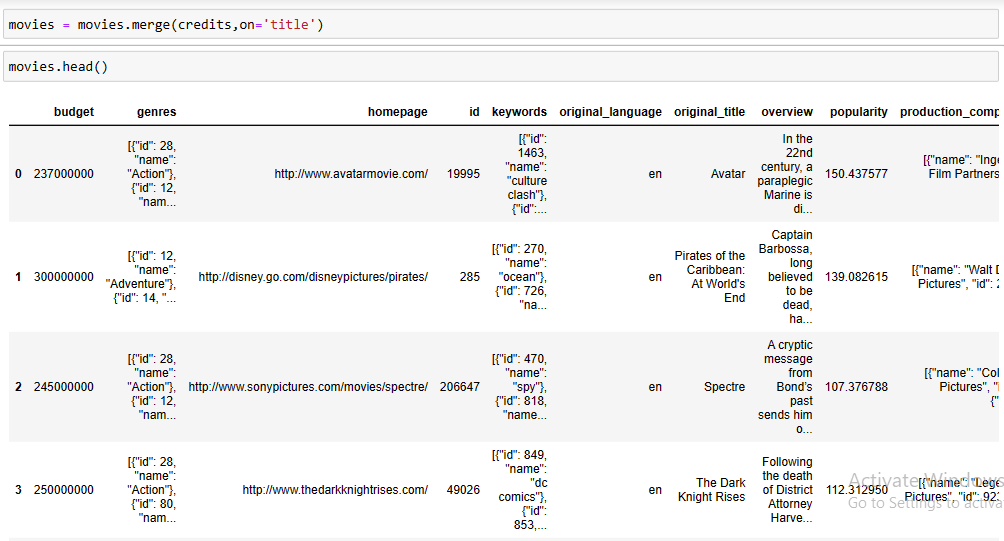
* Then import the datasets of movies and credits.
* In the **movies** dataset, we have so many columns like “budget, genres, homepage, id, etc.” and many more that the single movie could have the attributes.
* In the **credit** dataset, we have 4 columns like “movie\_id, title, cast, and crew”. Cast attribute means actors in the movie and crew means those people which are working behind the scene like the director, editor, music director, etc.
* In short, we have data from 5 thousand movies in this dataset.



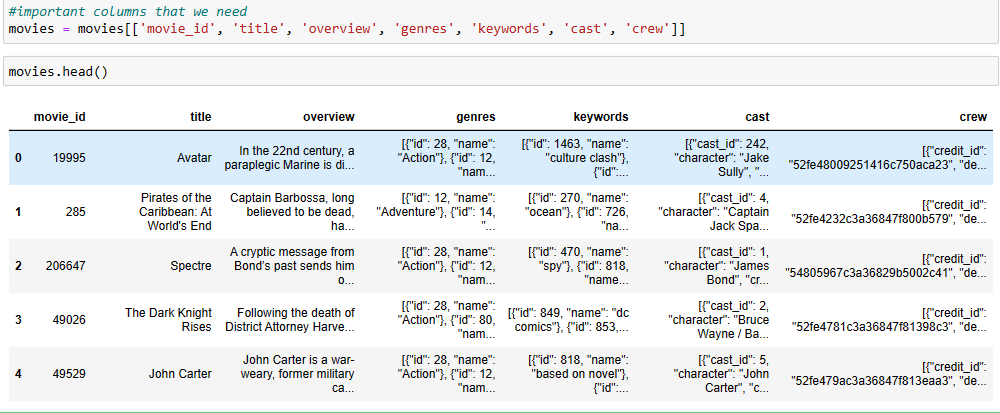
* Print movie dataset



* Print credit dataset.



* We have two separate data frames and both have data from the same movie. First of all, we have to do is to merge these data frames to make it easy.
* We can merge these two data frames on the basis of movie\_id and title and we merge these data frames on the basis of title as you can see in the above image.



* We have a very large dataset. In the dataset, there are many extra columns that we don’t need for our recommender system. So we only take those columns that are important for our system.
* These columns are **“movie-id, title, overview, genres, keywords, cast, and crew”**.

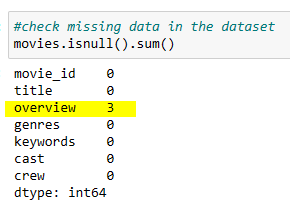
**Preprocessing:**

Now after extracting the columns we have a new dataset. From this data frame, we have to make a new data frame that consists of only three columns like **“movie-id, title, and tags”**.

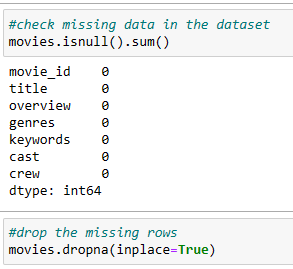
Now we make the **tags** column by merging the existing columns that are **“overview, genres, keywords, cast, and crew”**.

Before doing this we have to perform preprocessing on data as fellow;

**Missing Data:**

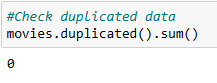


* First of all, we check the missing data.
* You can see in the **“overview”** column we have three rows that is empty. It means we don’t know the overview of three movies in the dataset.



* So we fix the missing data by running this code and removing the missing rows in the dataset.
* As you can see in the above image all the rows have 0 missing data.

**Duplicate Data:**

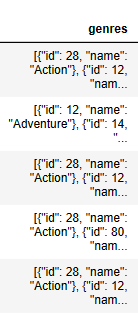
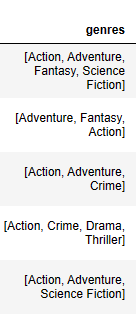


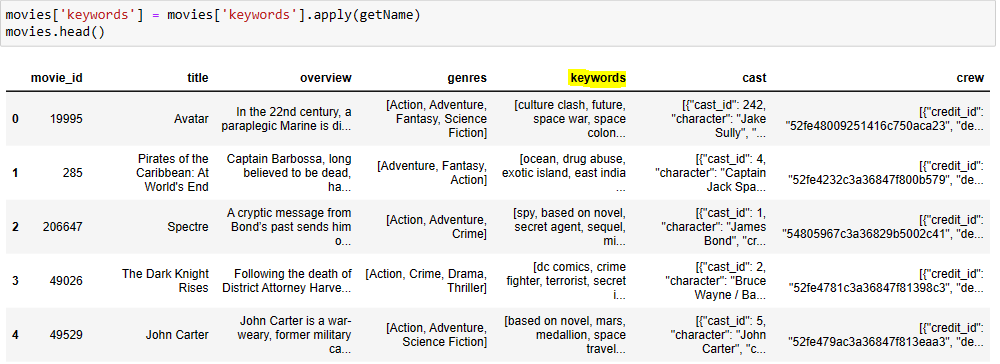
* Now we check the duplicate data in the dataset. Fortunately, there is no duplicate data.
* If there are some duplicate data then we can drop them by using the **drop\_duplicate ()** method.



* Now we extract only genres name in the genres column. There are many of data in a weird form that is not helpful for us that’s why we extract only genres name.
* To do so first of all we import the module **ast** and ast has a function called literal\_eval ().
* Literal\_eval () is used to the convert string of the list into the list.
* Then we declare the list and run a loop and then we take the value of the name from every dictionary and then append that value to the list.
* In the end, we return to the list of genres.
* By doing this, for every movie, we are getting the list of all the genres of that particular movie

**Comparison:**

** Before Preprocessing After Preprocessing**

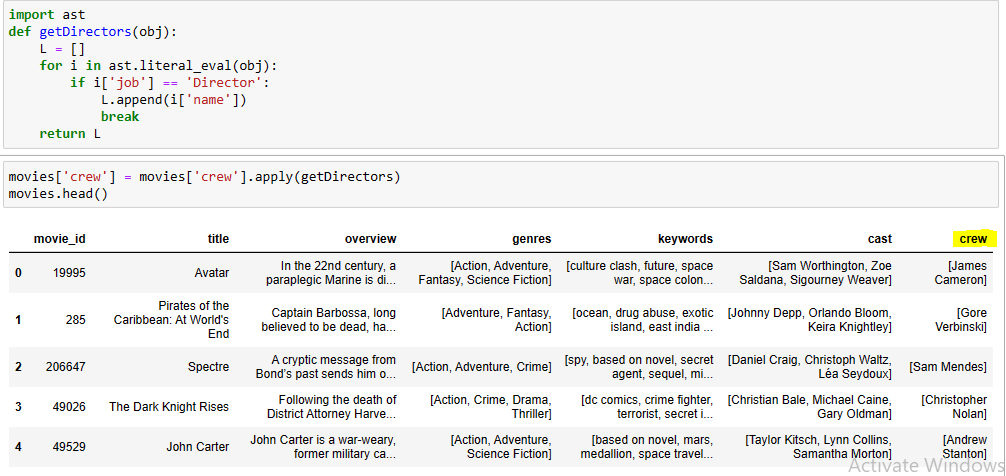


* Apply the same function on the **keywords** column as performed on **genres**.
* Then we got the keywords of all movies as you can see in the keywords column on the above image.



\

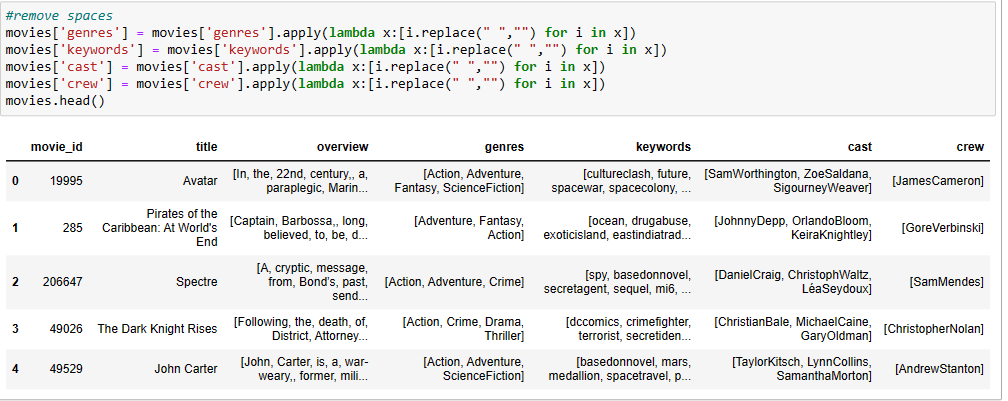
* Now we want to get the name of the first top three characters in the **cast** column.
* To do so we use the same previous function with minor changes as you can see in the above code.
* Declare and initialize the counter variable with 0
* And in the **“if”** condition we set the counter value to less than 3.
* By doing this we got the top 3 characters’ names for all the movies as you can see in the cast column in the above image.



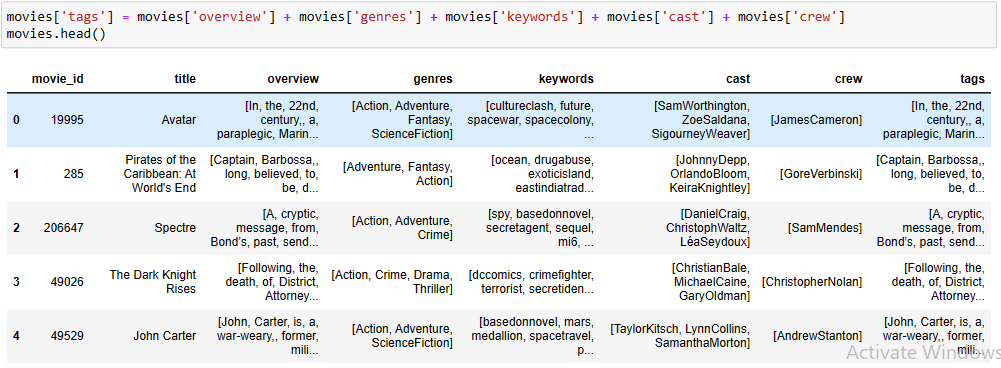
* In the **crew** column wejust only need the director’s name which has a job value equal to **‘Director’**
* To do so in the **“if”** condition we write the condition that the job value of ith is equal to **‘Director’**.
* Then break the loop.
* By doing this we got the name of directors that has the job of directors.



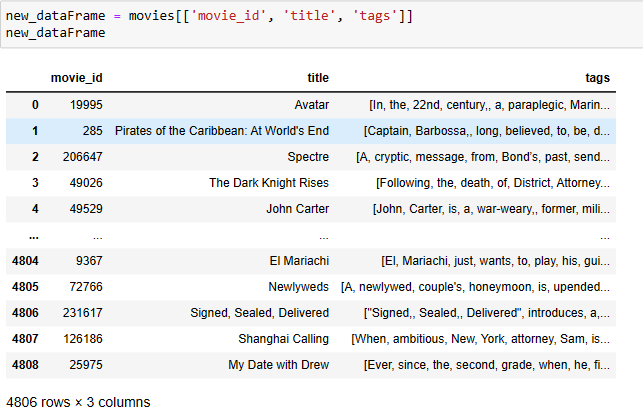
* The **overview** column is a String. We convert it into a String so that we concatenate this list with other lists.
* To do so we use a **lambda** function that converts the String into the list.



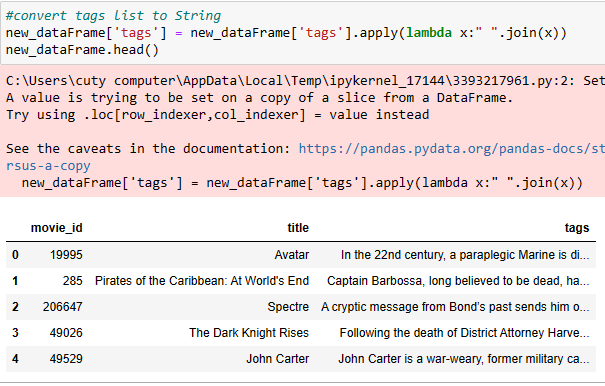
* Remove spaces from the **“genres, keywords, cast, and crew”** column.
* It is very important for our recommender system.



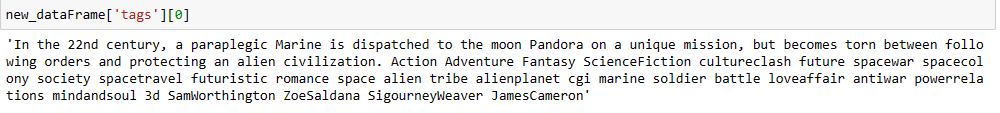
* Now we make a new column called tags that is a concatenation of **“overview, genres, keyword, cast, and crew”** as we were talking about at the start of preprocessing.



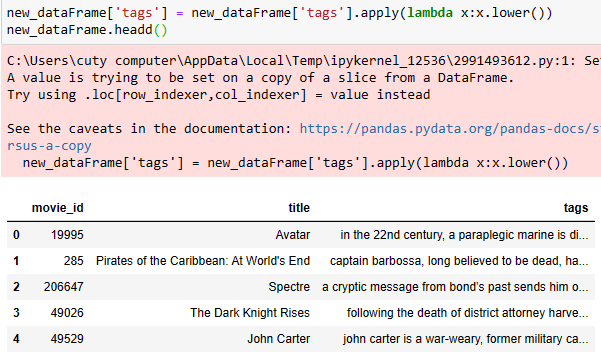
* And we remove these 5 columns from the dataset and make a new data frame with **“movie\_id, title, and tags”.**
* Therefore **tags** column is a combination of **“overview, genres, keywords, cast, and crew”**.



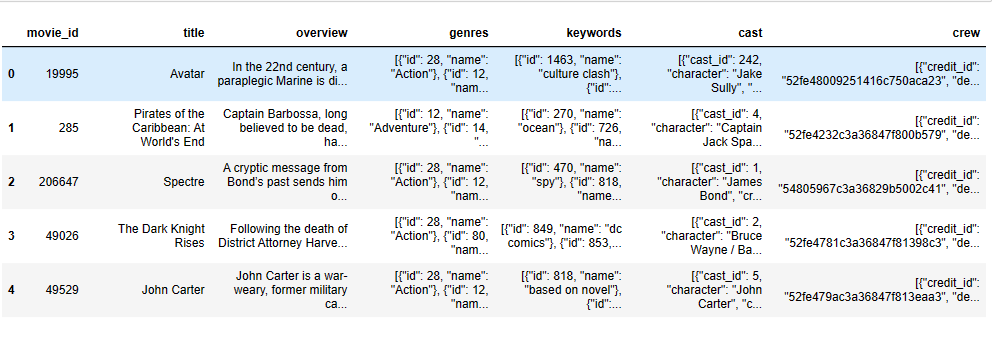
* Convert the list into a String of **tags** column.

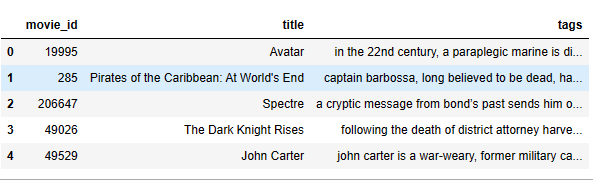


* Now you can see the tags of 0 index. It is completely converted into a paragraph that consists of an **overview, genres, keywords, cast, and crew**.



* And we suggest converting it into lowercase which is part of preprocessing.
* Now you can see in the above image **tags** column converted into lowercase.

**Before Preprocessing:**

**After Preprocessing:**

* Now you can see the difference between before and after preprocessing.

**Vectorization**

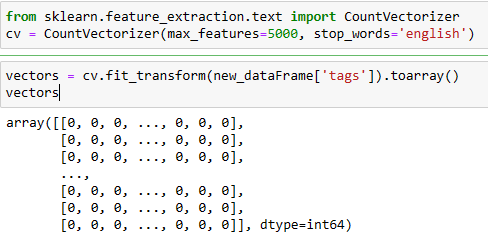
After preprocessing now we are doing text vectorization.The user tells us the name of 1 movie and we recommend 5 movies related to that movie. We recommend the movie based on tags. And tags tell us the similarity between the movies. And we find the similarity based on the tags. And the tags column has textual data. To find similarities between textual data we need vectorization. So we have to do is to convert this whole text into different vectors. We convert the tag of each movie into a vector. After converting each movie will become a vector. Suppose we have 5 thousand movies so in the whole space there will be 5 thousand vectors. And then we take the 5 nearest vectors for the recommendation and show them to the user.

Now our task is to convert the text into vectors. And this process is called text vectorization.   
We use the **bag of words technique** for text vectorization.

**Bag of words technique:**

In the bag of word technique, we concatenate all tags of all the movies.  
In tags, we have thousands of words and we combine them. After concatenating we have a huge text. Let’s say from the huge text we need 5 thousand most common words. For taking 5 thousand common words we check the huge text and calculate the frequency of each word.  
And then we extract the top 5 thousand words of the highest frequency. After that, we take the first word from 5 thousand words and check in the tags of all the movies how many times that word is repeated in the tags of the movie. And then take the 2nd word and check in the tags of all the movies again how many times that word is repeated in the tags of the movie and so on for the 5 thousand words. After completing that table each row will convert into a vector in the 5 thousand-dimensional space. And then the user enters the name of the particular movie that he wants to watch. After that our system takes the 5 nearest vectors from the space and recommends them to the user.

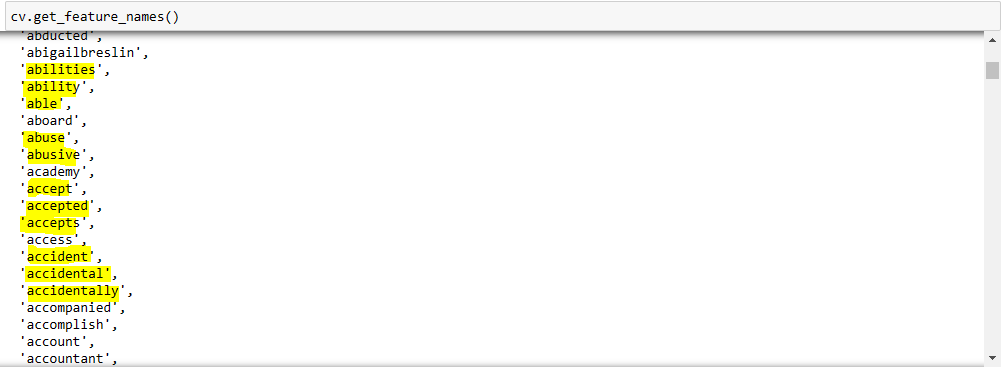
When we perform vectorization we consider one important thing that we do not consider **stop words** e.g. are, and, of, to, is, from, etc.



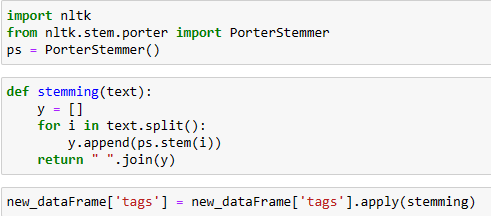
* We use sklearn for vectorization. In sklearn, there is a class called CountVectorizer used for vectorization. We use it for vectorization.
* First of all, we import the CountVectorizer from sklearn.
* Then we made an object of CountVectorizer named CV.
* And inside this class we pass **max\_features** = 5 thousand this is the number of words and **stop\_words** of the English language.
* Then we use the CV object by writing cv.fit\_transform () and pass the tags in fit\_transform () and explicitly convert into a numpy array because by default CountVectorizer returns the object this is a sparse matrix that’s why we convert into a numpy array and this whole code is equal to the vectors.
* Now you can see all the movies are in vector form.



* So you can see 5 thousand frequently used words by writing this function.



* Now there is one problem multiple different words with the same meaning are included in the array.
* So we have to fix it.



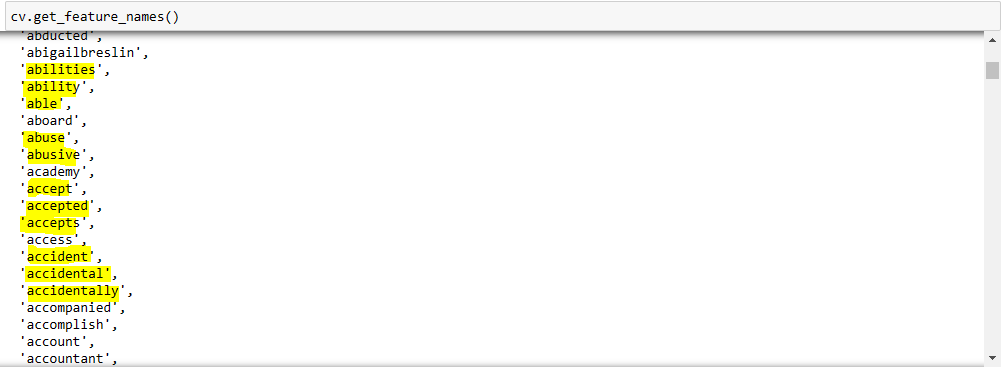
* We can fix it by using stemming. **Stemming** is a technique that returns the first form of all English words.
* Then we have to apply stemming on the tags.
* To do so, we import nltk, and then from nltk, we import portal stemmer and then made an object of portal stemmer.
* And then we made a helper function named stemming and pass with text and then declare a list and then run a loop by splitting the Strung to list and then stem all the words.
* And in the end, we return the list by converting the String into the list again.
* And then we apply stemming on tags.



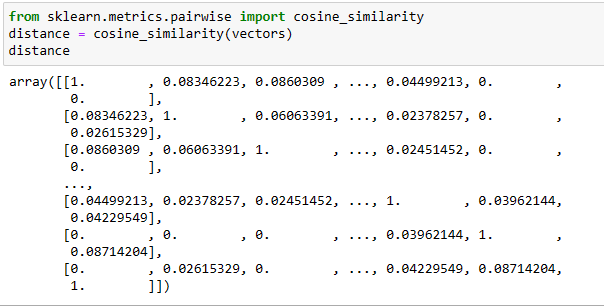
* Now you can there are no different words with the same meaning.

**Comparison:**

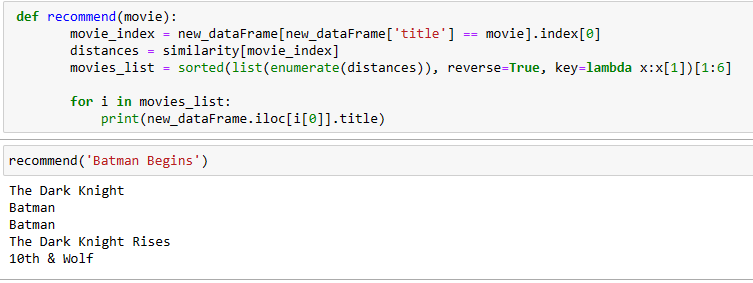
**Before Stemming:**



**After Stemming:**



* Now we calculate the distance from each movie by using cosine similarity as you can see in the above image.

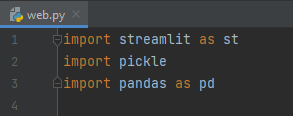


* Now we made a recommend function. Working of this function is if we pass one movie then this function returns the name of five similar movies with that movie.
* First of all, we find the index of the movie from the dataset.
* Then through an index, we go inside the similarity matrix and fetch that particular movie of that index. Then we have a vector array of distances of the particular movie from all the movies.
* Then we sort the distances in descending order. By doing this most similar movie comes first. And then we fetch the top 5 movies and recommend those movies to the user.
* We used enumerate function in sorting for the index position and convert that function into the list. By doing this the list will convert into the list of tuples and index positions will be fixed.
* Then in the end we will print the list of 5 movies.

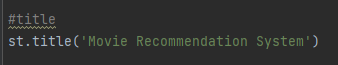
**Website:**

Our model is ready. The next step is to convert this entire thing into a website. We **PyCharm** for the working of the website. It’s an IDE for Python development.

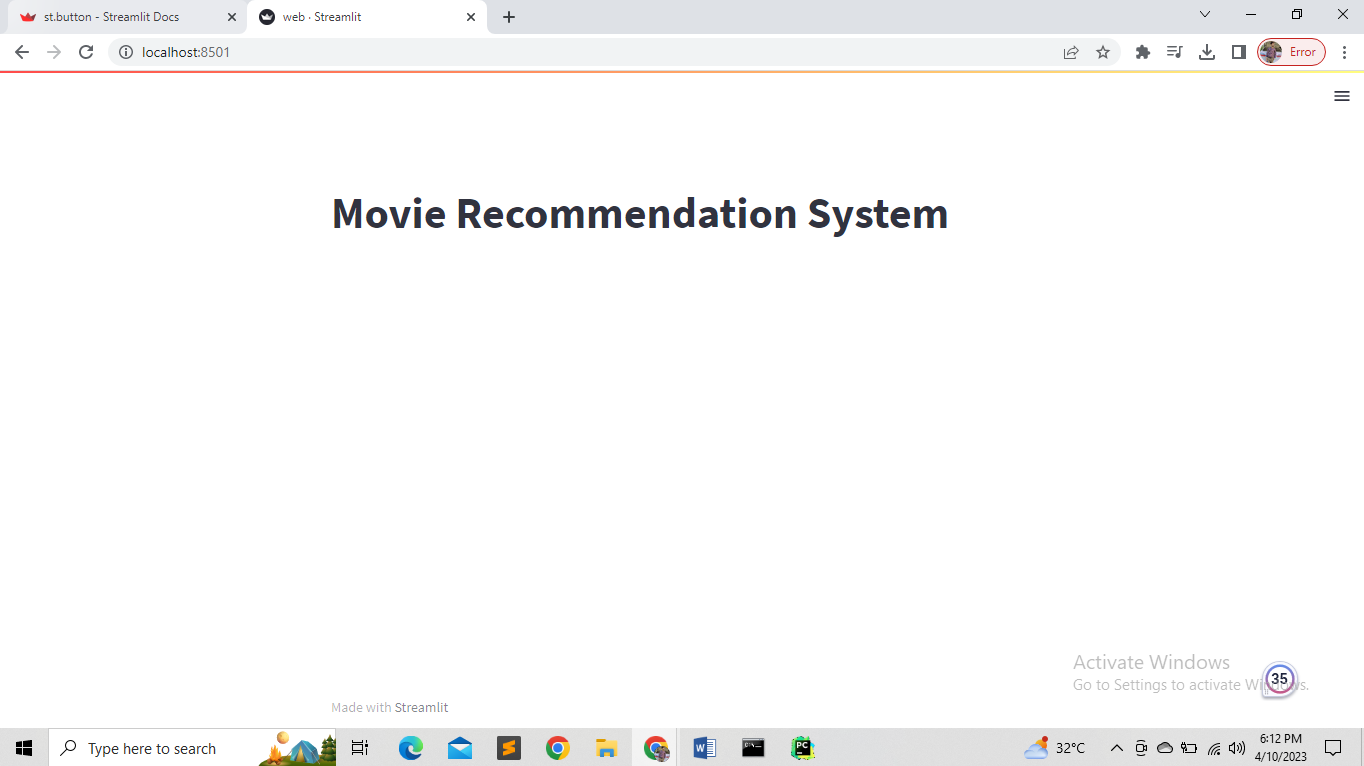
* First of all, we are creating a new virtual environment project in PyCharm.
* And in the project, we create a new file called **web.py**



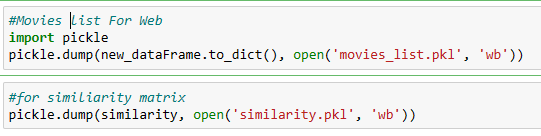
* First of all importing the libraries



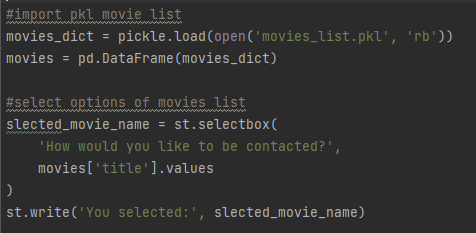
* Then we write the title of our website and run this code.



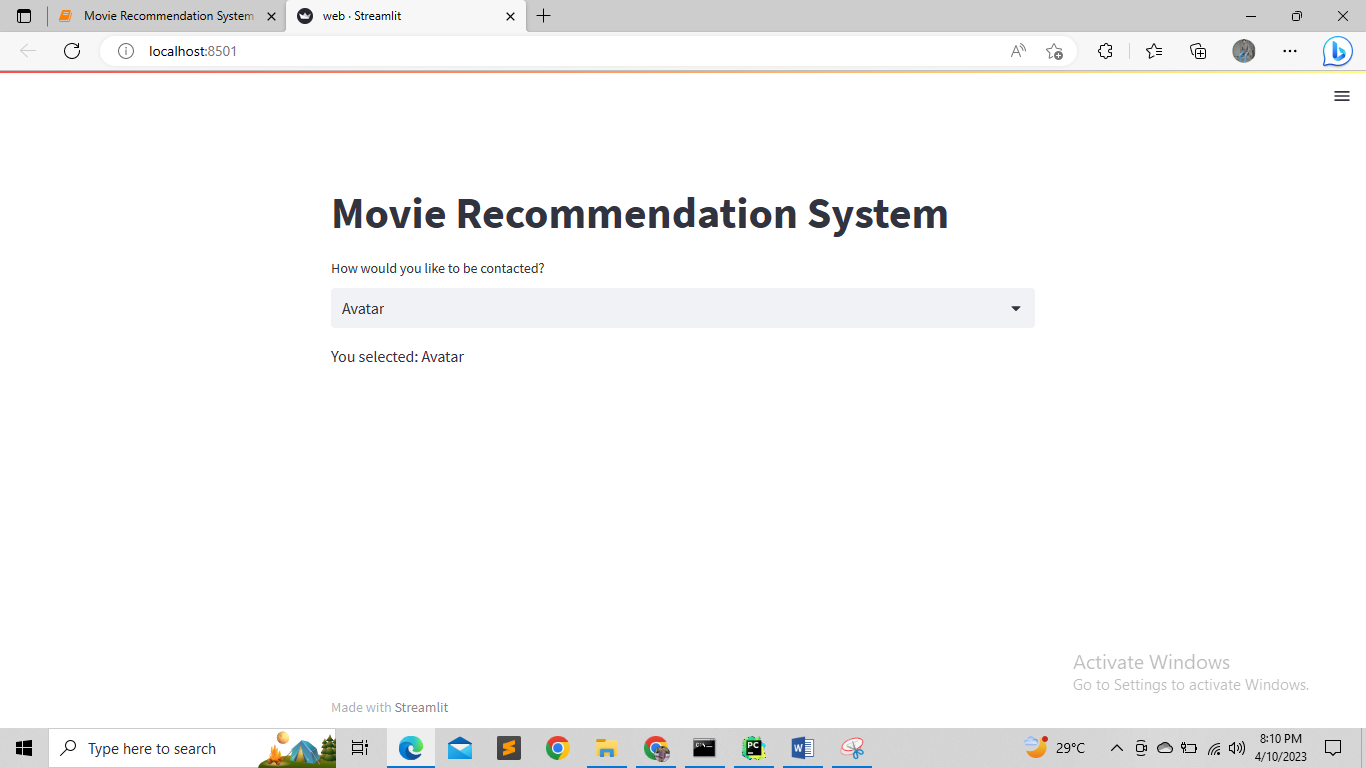
* You can see it’s the web page of our website with the title.

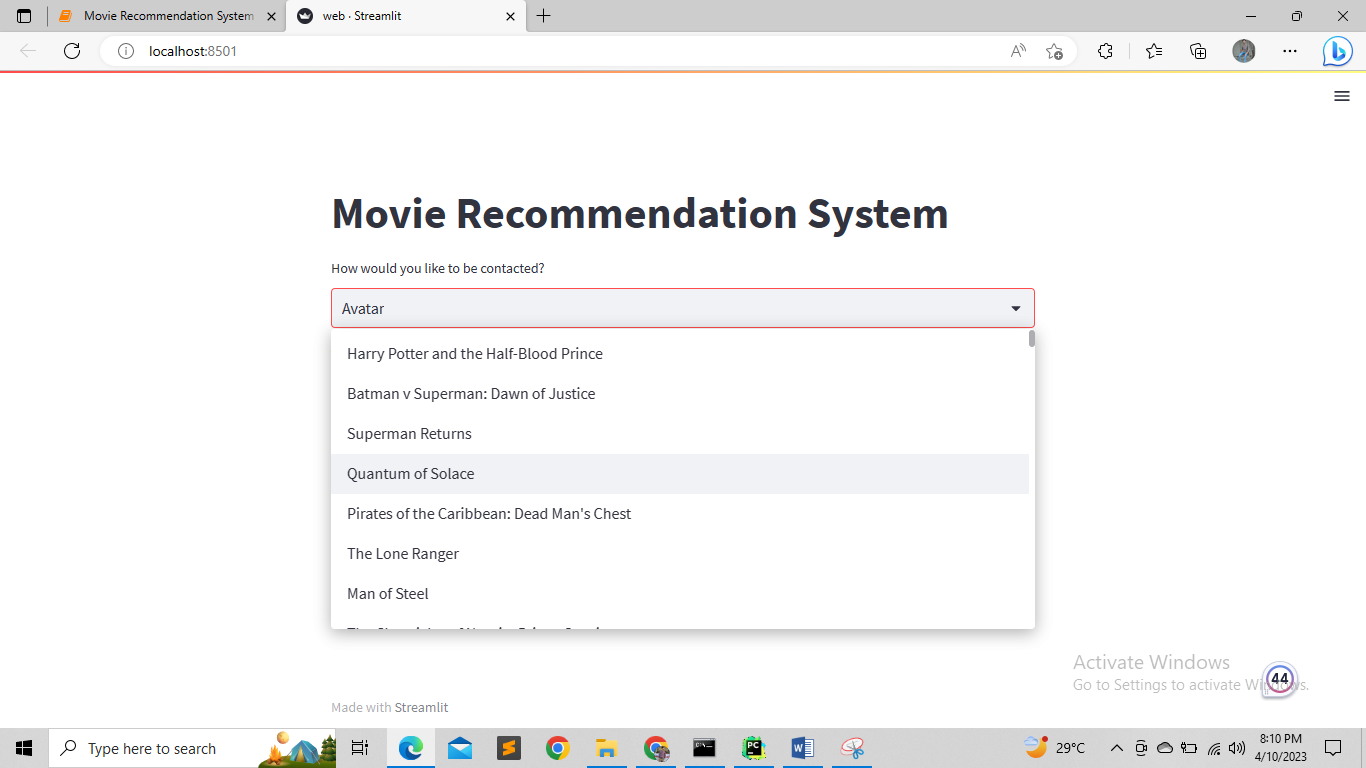


* Now we want to send a list of movies and a list of an array of similarity matrices to the website code.
* For doing this we used the **pickle library**.
* This file generates **.pkl** files that can be used in the PyCharm of our website.
* So first of all, we import the **pickle library.**
* Then we generate the **pkl** files in binary form for the website of movies list and similarity matrix.

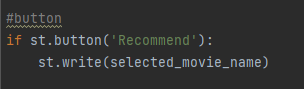


* So for the drop-down options of movies on our web page first of all we imported the pkl files of the list of movies.
* Then make a new data frame using pandas by passing pkl file.
* Then in the drop-down menu code, we extract the title of the movie from the data frame.

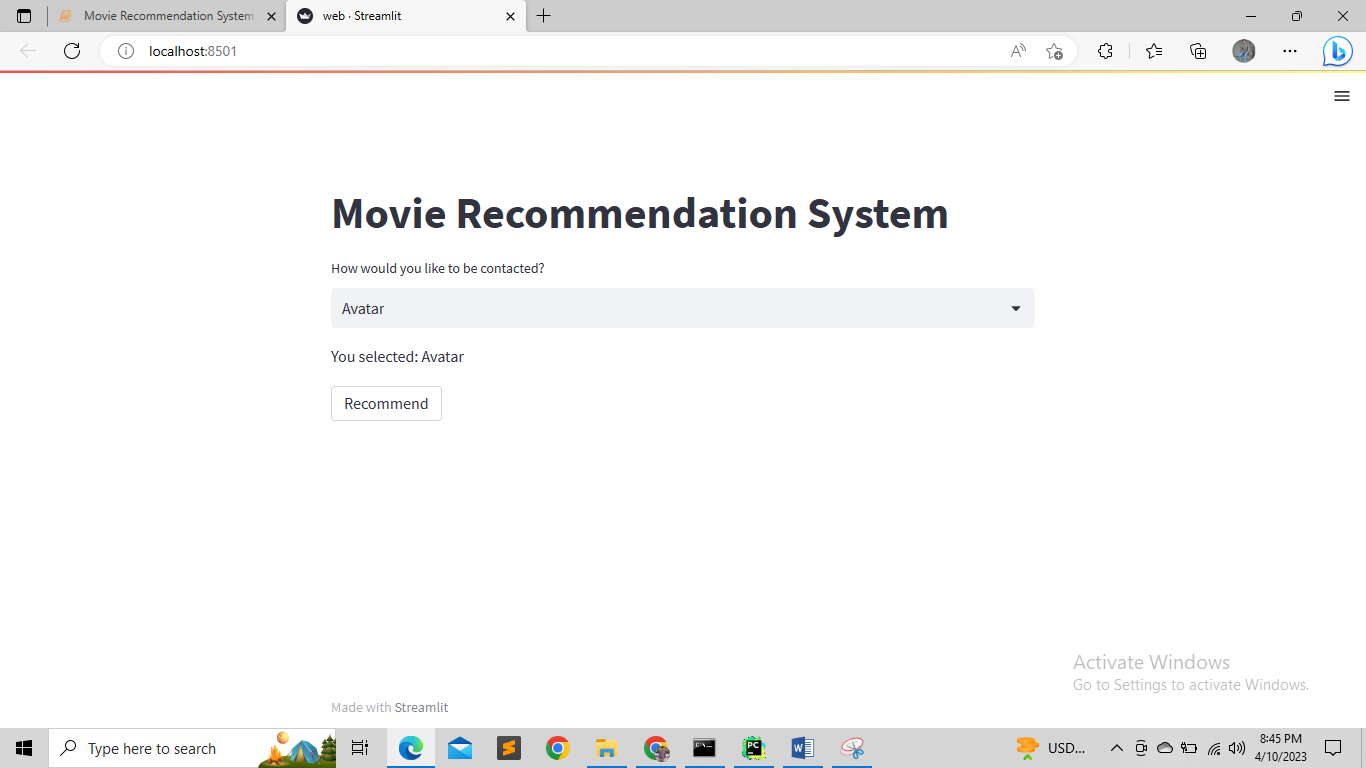


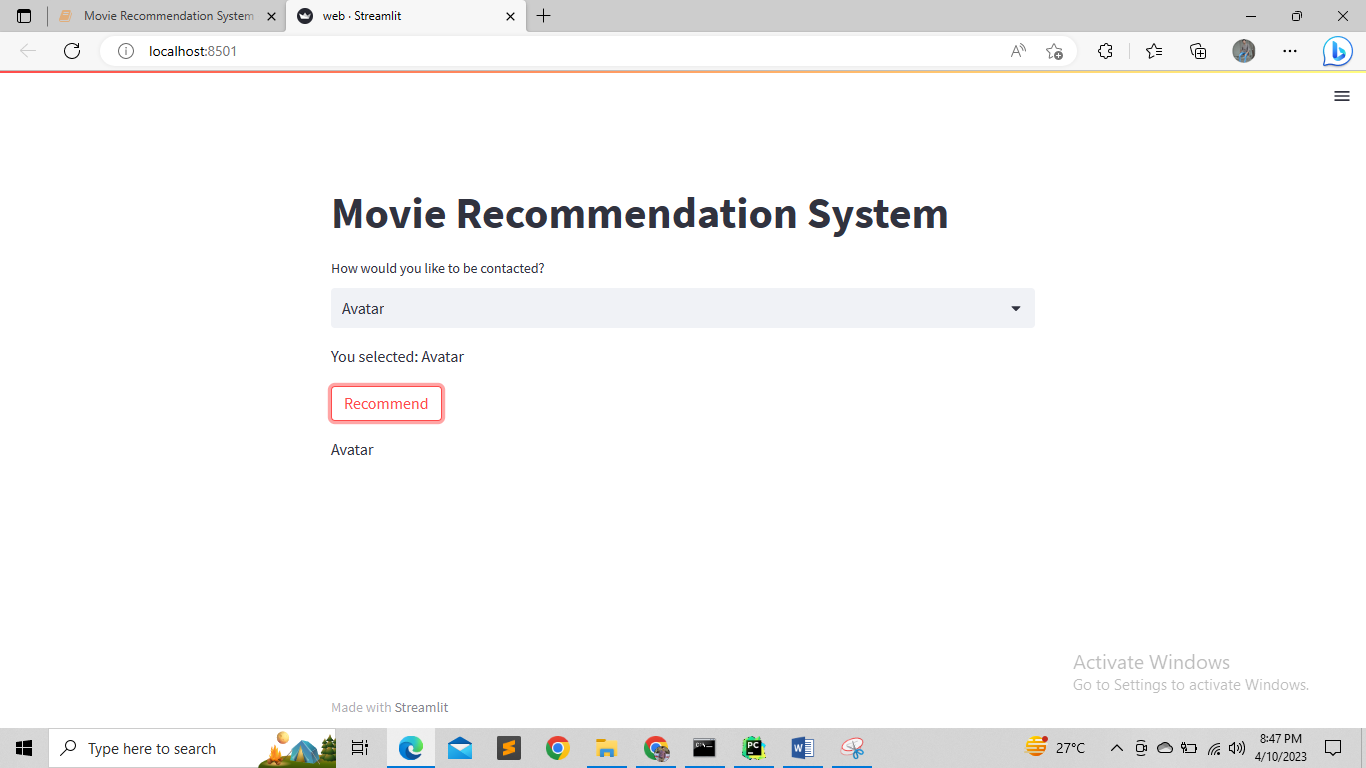


* You can see we have option for all the movies to select.



* Then we made a button by writing this code
* By clicking on recommend button we show the selected movie name.

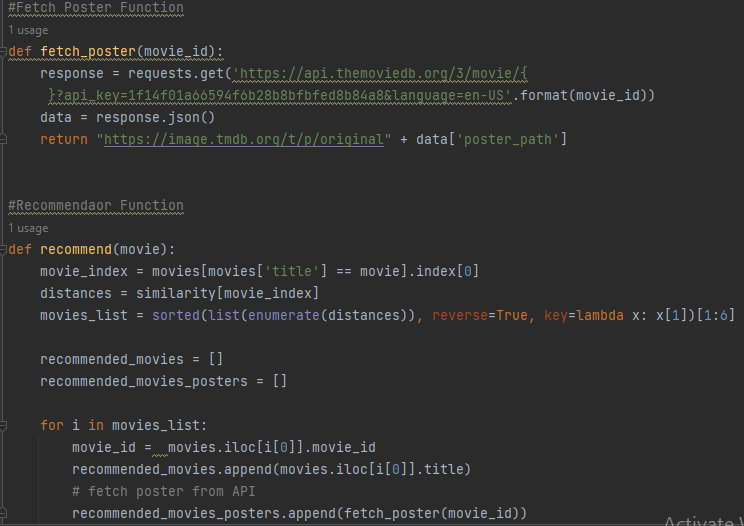




* You can see the recommend button and the working of recommend button.

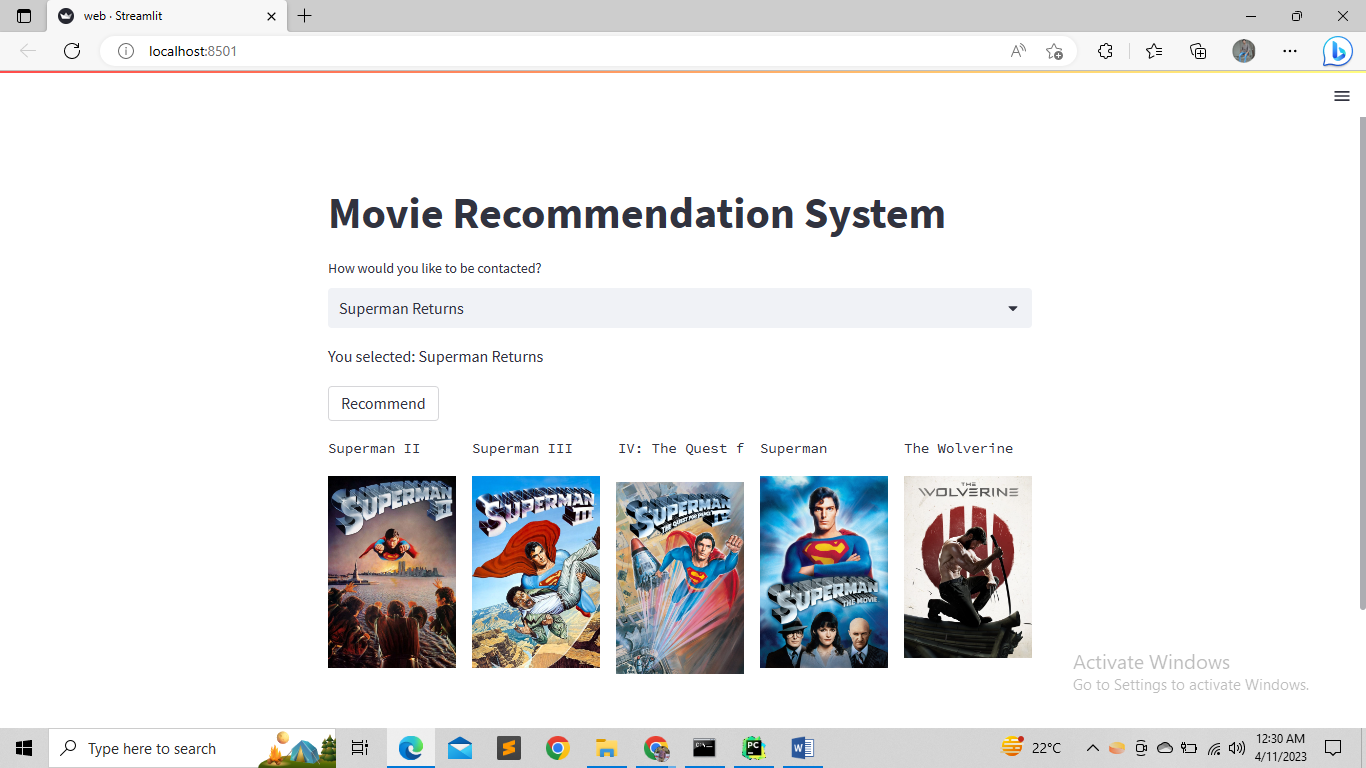


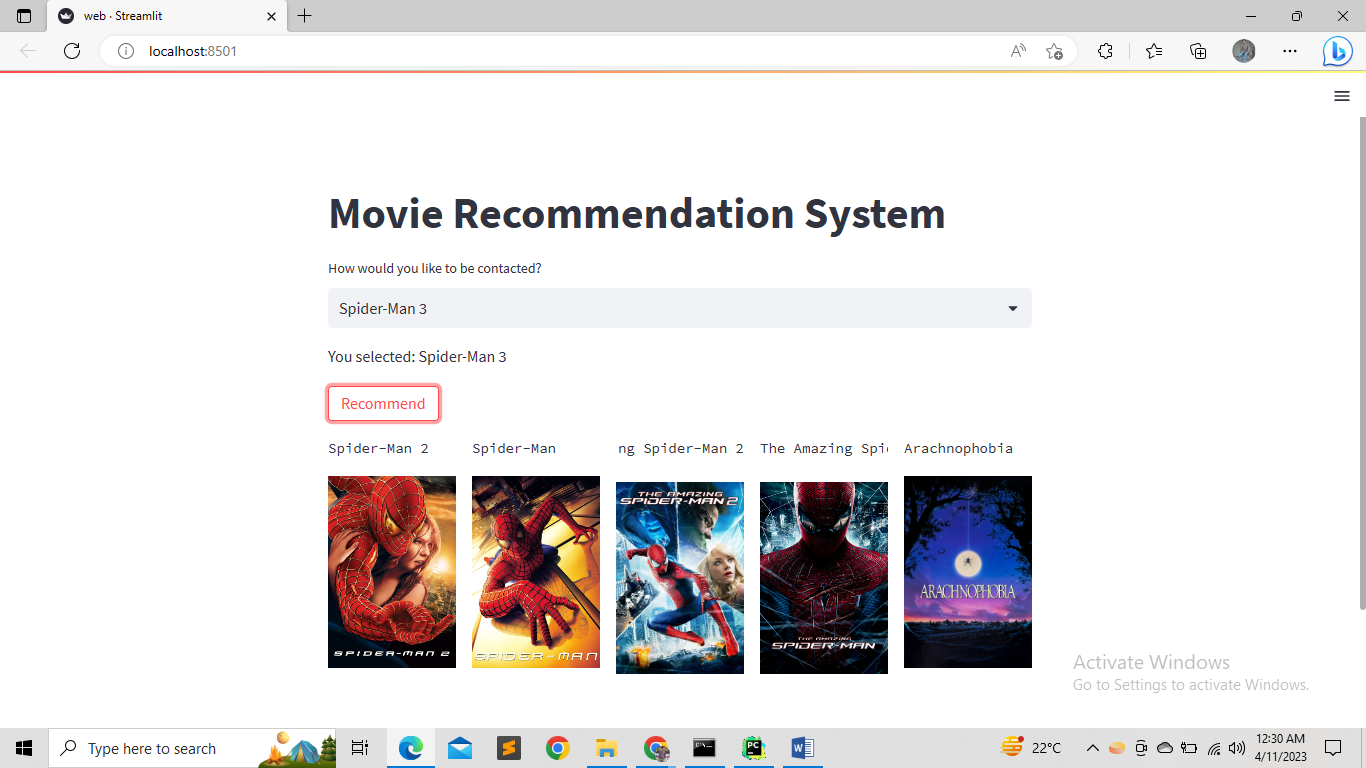
* First of all, we import the similarity matrix pkl file.
* And then write the recommendation function that is quite similar to the previous project code
* This function returns the list of five movies similar to the selected movie.
* Then in the button code, we run a loop for printing the 5 recommended movies on the screen.





* Now we print the posters with the movie name.
* We can do this by using **TMDB API**.
* First of all, we write a function that fetches the poster from the API.
* We import the library called **requests.** It is used to hit the API.
* Then we will get the response from the API and convert that response into **JSON**.
* And then this returns the poster path.
* Then in the recommendation function, we make an array of posters.
* And then through the loop, we append all posters of movies in the array.
* Then in the end this function returns the posters of movies.
* Then in the button code, we display the name and poster of 5 recommended movies.





* So yeah you can see this is our complete website.