**Project: Machine learning**

**Movie recommender system**

**Contents of the project**

|  |  |
| --- | --- |
| **1.** | **What is a Recommender System?** |
| **2.** | **Types of Recommender Systems** |
| **3.** | **Project Flow** |
| **4.** | **About the data set and python packages** |
| **5.** | **Pre-Processing** |
| **6.** | **Model Building** |
| **7.** | **Creating a website** |
| **8.** | **Deployment** |

**Movie recommender system: An end-end project**

* Data set will be chosen
* Building a model on it
* Convert it into a website
* deploy it on the server

## What is a Recommender system?

When we go out shopping, what is the flow? Let’s say we are shopping for clothes. Let’s say we wish to buy a pair of jeans; we pick one style and ask the shopkeeper to show us styles similar to what we like. In an offline setting, we can call this a recommender system. The salesman knows the product, and he is recommending a particular product. The machine is a salesman in the online environment who recommends the product according to what we browsed.

Recommender systems are used in various fields nowadays; it is used to suggest a song or a product on an e-commerce website; YouTube, FB, Instagram, and Netflix have recommender system too.

## Types of Recommender Systems

Two types: -

1. Content-based
2. Collaborative filter-based

**Content-Based: -**

Recommends based on similarity of the content. If you watch cricket videos on YouTube, you will be recommended only more cricket matches, not songs.

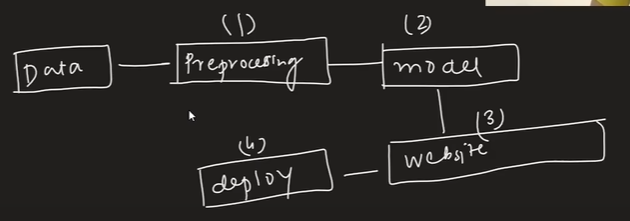
**Collaborative filter-based**

Content is filtered based on the user’s interest.Let’s say P1 and P2 have a similar liking of movies. They might be giving an equal rating to the film. So now, when P1 watches a new movie, and since we know that P1 and P2 have the same interests, the new film would also be recommended to person P2. This happens in reels as well. My friend and I like similar reels; the reels my friend watches; will also be recommended to me.

**We will be making a content-based recommender system.**

## Project Flow

1. pre-processing of data
2. building a Machine learning model
3. convert the model into a website
4. deploy the website



## About the data sets and Packages

TMDB 5000 Movie Dataset; Data is taken from Kaggle.com, which has data on 5000 movies.

* Metadata on ~5,000 movies from TMDb
* Movie csv, credits data containing columns like budget, homepage, original language, release date, cast, directors, genre id, etc

Imports

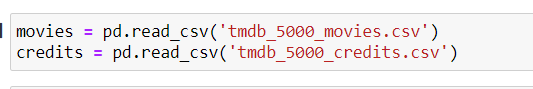
Pandas-

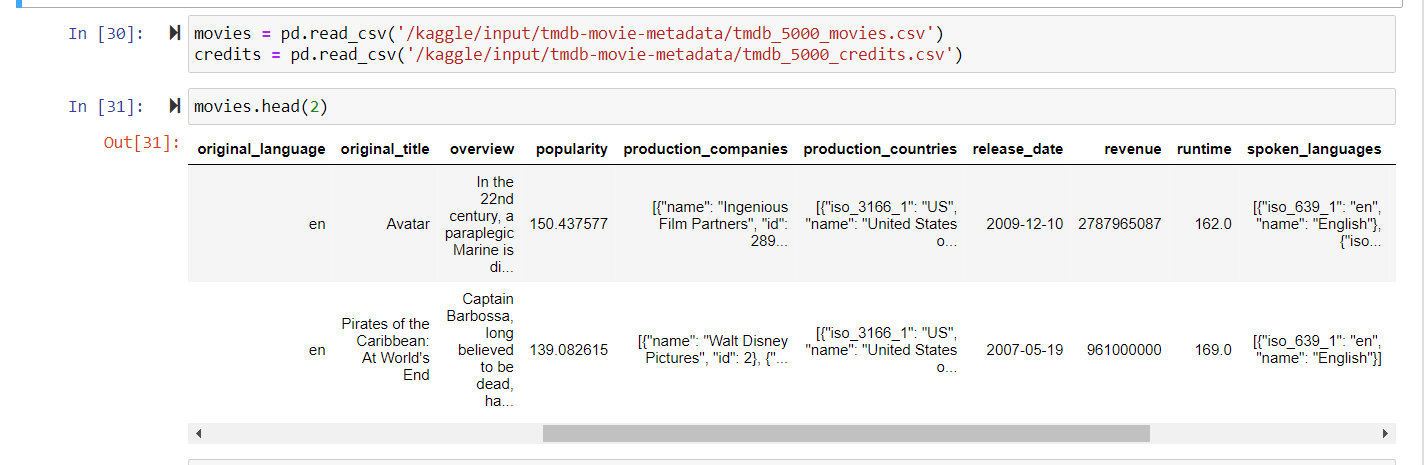
Numpy -

## Data and it’s pre-processing

### Steps Followed: -

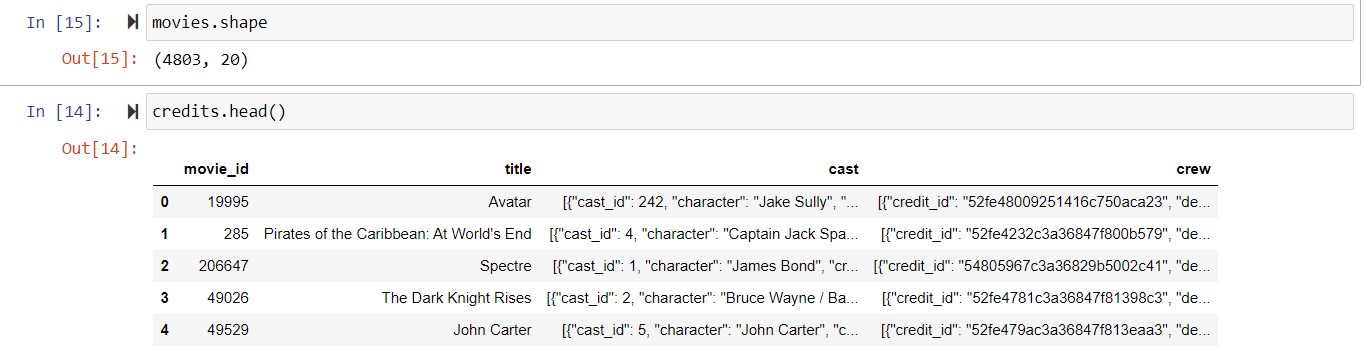
Importing the data and looking at it





We had two datasets –

1. one has data about the movie data. E.g., release date, run time, languages, original languages, etc.
2. Second, having movie id, cast, and crew.



**We are going to merge both datasets into one single data set.**

**Dataframe1.merge(dataframe2 , on =”on which column”) and update it in movies frame only**



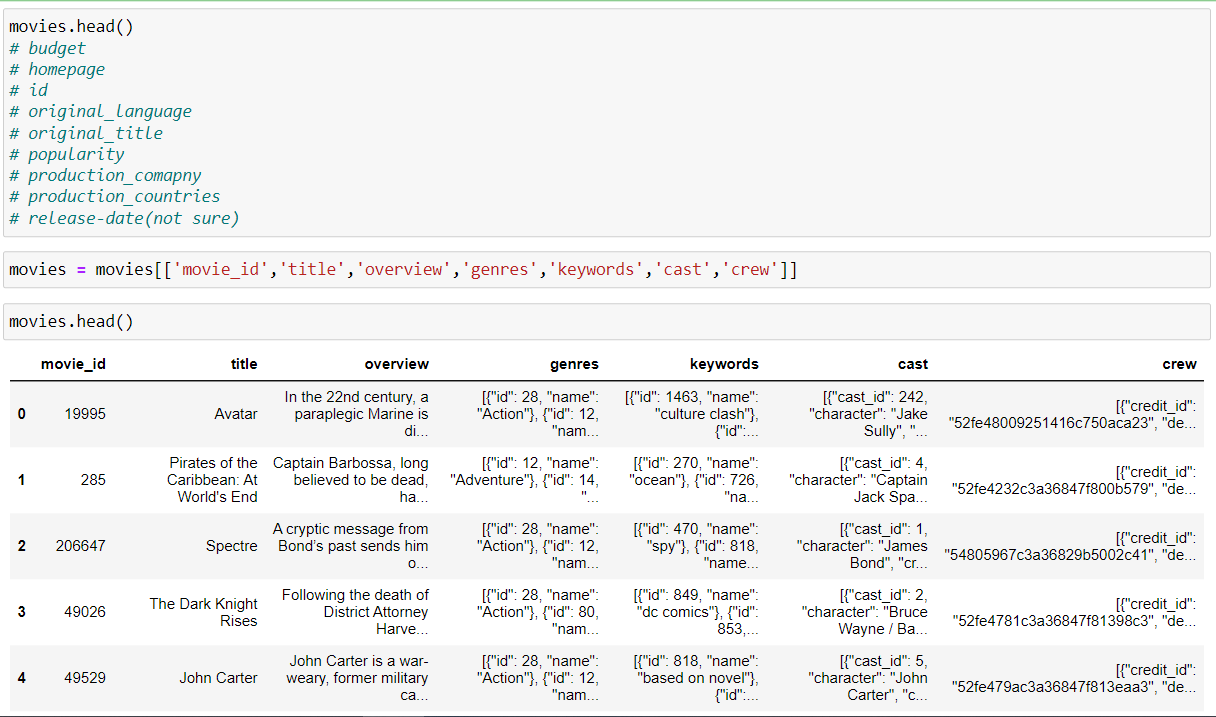
The basic aim is to create a tag for a movie. Now we must ask ourselves whether a particular column will help us create a tag for a film. If not, we are going to remove those columns.

Operations on the data - Removing unwanted columns

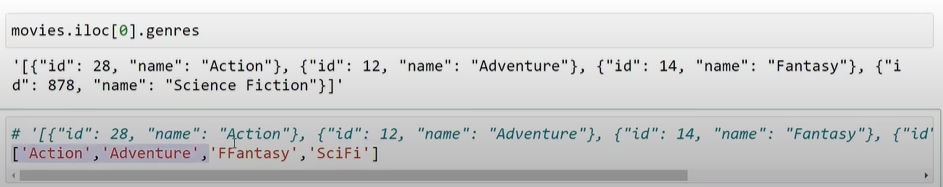
Columns kept

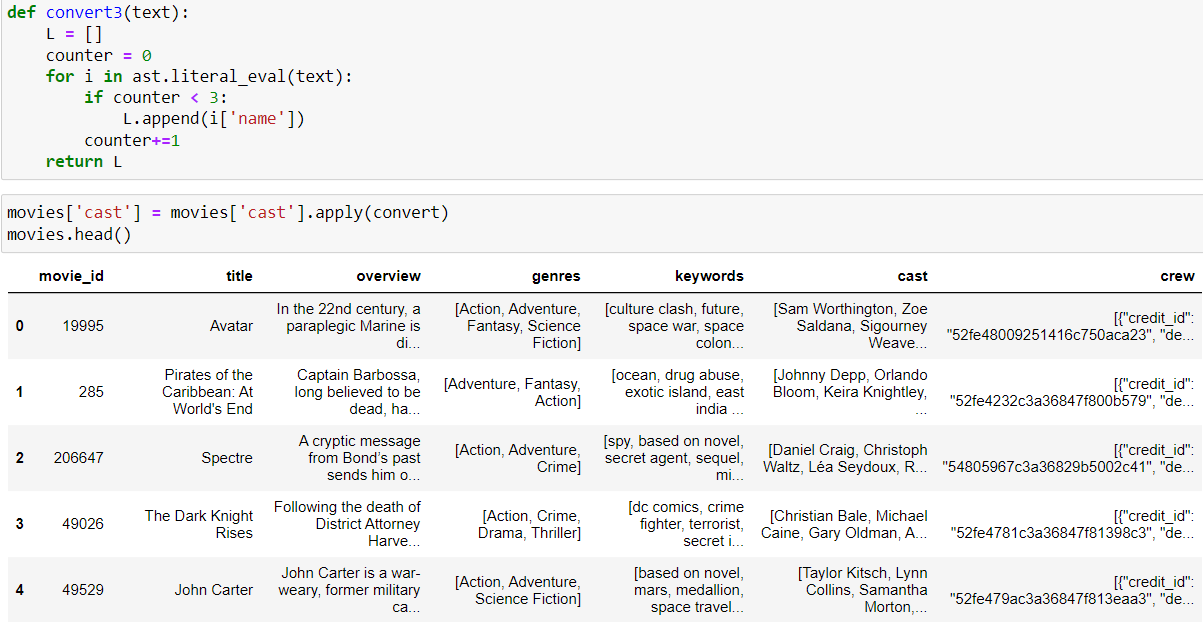
* Genre – Similar genre movies might be recommended
* Id – Posters of the movies would be fetched via the id
* Keywords – if keywords of 2 movies are similar, then the user might like them
* Overview – If the summary of a movie is similar

**Refer to the image below.**

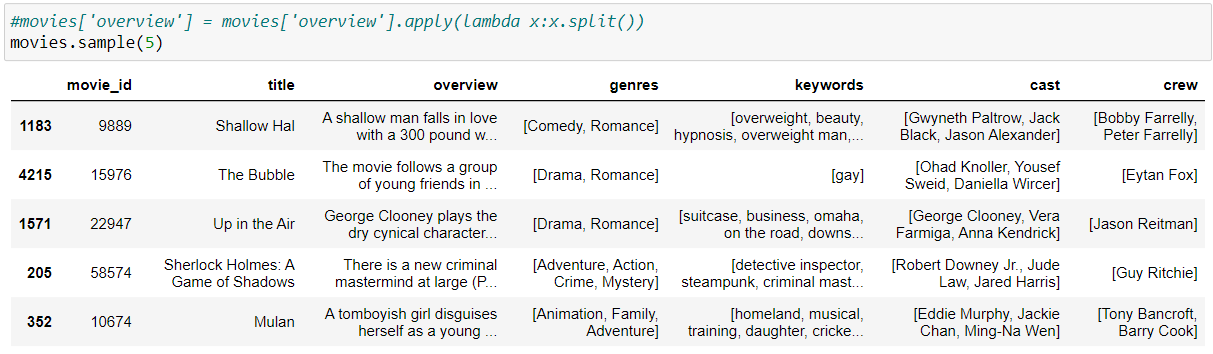


### Processing of the Data

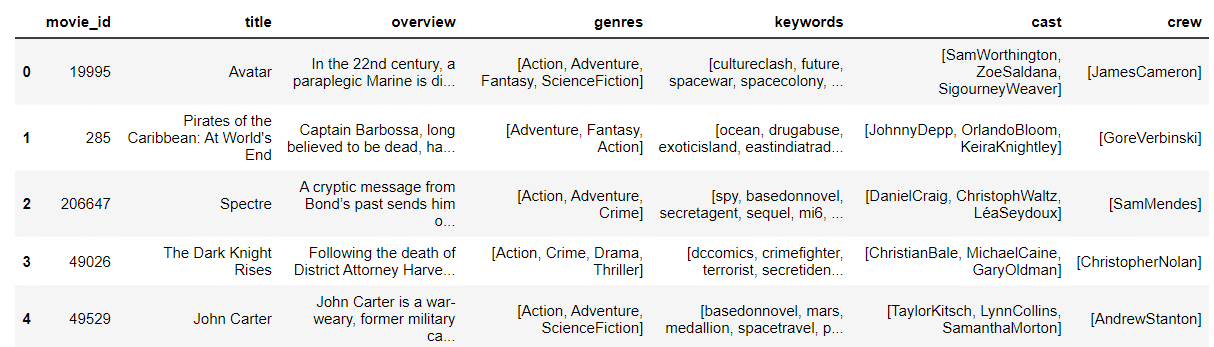
* We removed the missing data. Some movies did not have overview🡪 **dropna(inplace=True)**
* We formatted the data so that strings were converted into a list. And uniformity is maintained in all columns. 
* We removed the space between the first and last names of people 🡪 to create one tag of a person rather than two tags (first name and last name). This concept is applied to all things 🡪 in names of directors, cast, and genres (Science Fiction 🡪 ScienceFiction).
* Using a function, we pick up three prominent cast members of the film and leave out the rest of the info in the form of a dictionary.



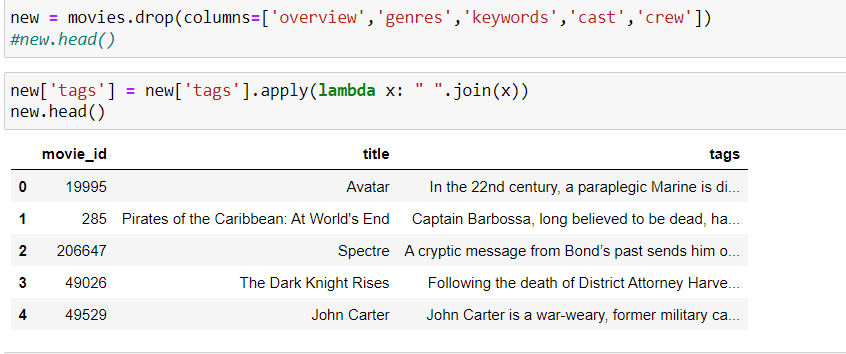
Similarly, we must fetch director names from the crew members column.



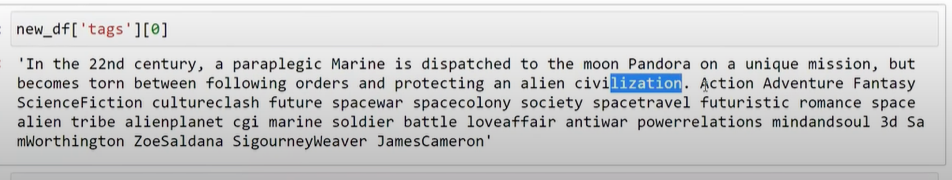
**Every two-letter word is converted to a one-letter word to save the recommender system from getting confused**. E.g. SamWorginton vs SamMendis



Lastly, we will merge overview, genres, keywords, cast, and crew into one column called tags.



A complete merged string of the tags looks like this –



## Model Building

After cleaning and setting data in an appropriate format, we must start finding similarities in texts to proceed with our recommender system.

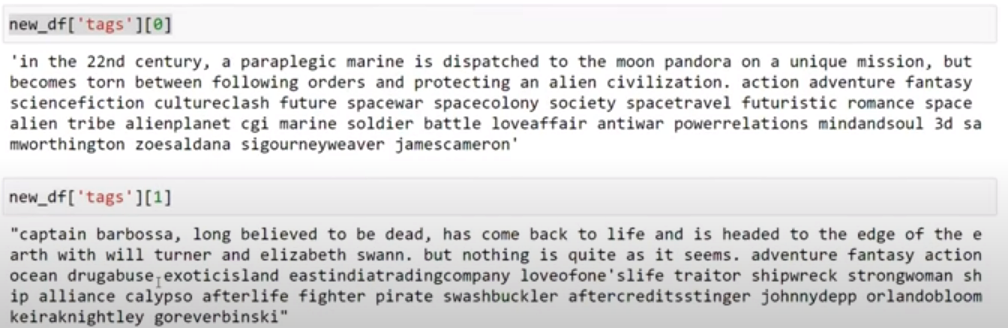
### Text Vectorization:

Scenario:

We have created tags for movies. **The problem Statement is to create a website where the user inputs the film’s name. And we must show the most similar five films to the user.**

Now, out of 5000, how to know which movie is most similar? We need to find this out based on tags.

Observe the tags of movie one and movie 2. Now how much similar are they? **This is dependent on how similar their text is. We need to calculate the similarity score between the two.**



**We will convert every tag /text of the movie to a vector. Every film is a vector. We will find the closest vectors to recommend a movie.**

**Concept:**

Converting text to vectors is called **Text Vectorization.**

**The technique used** is - **Bag of words.**

**Bag of words**: Since we have multiple tags of a movie, we will concatenate all the tags; after concatenation, we get a long string with many words. We will take out the 5000 most common words from that large text.

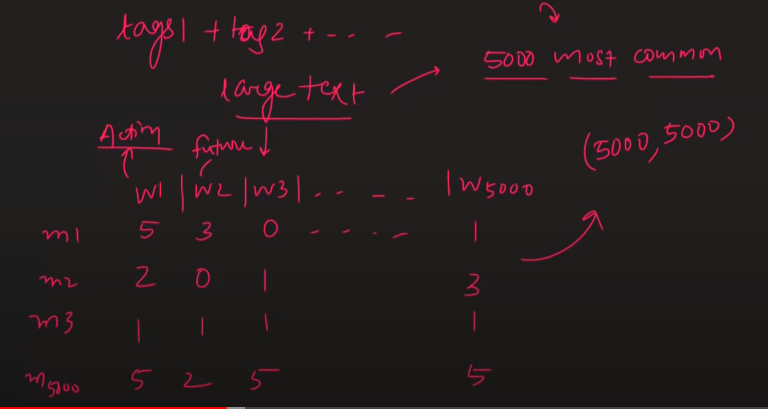
We will pick out the top 5000 words of the highest frequency.

M1: W1 W2 W3

M2 3 5 6

M3 2 4 8

This acts like a vector of the Wn dimension.



From the Picture: We picked a word action and saw that it comes five times, then the word future; it comes three times, and similarly, data will be created for 5000 movies with 5000 common words.

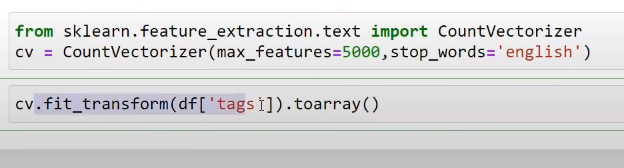
Every movie is a vector in a 5000 dimension space.

We started with fewer words: 2000, and 5000, so that with less dimensionality, we handled our data properly.

**Stop words are removed. Stop words: - are, an the, they. They have no contribution to the meaning of a sentence.**

**Note: We must manually concatenate the strings, pick up the most common words, and vectorize ourselves. We are using a package sklearn class count vectorizer.**

Convert a collection of text documents to a matrix of token counts. This implementation produces a sparse representation of the counts using scipy.sparse.csr\_matrix.

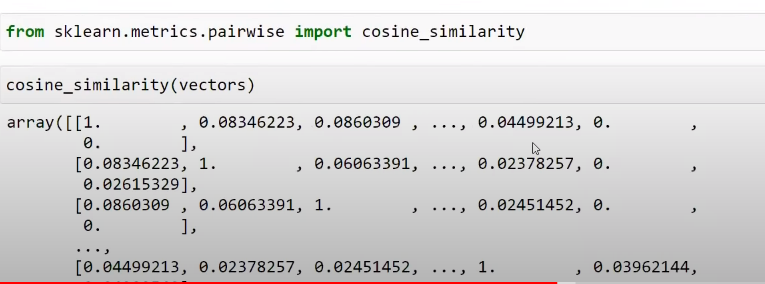


**We will use the library count vector, and perform stemming on the tags, so that actor, actors, and acting become one common word called an actor. After stemming, terms would not repeat too.**

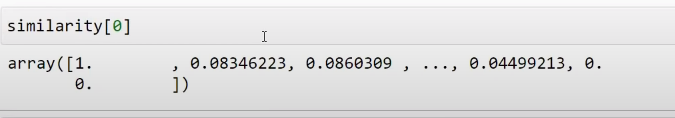
Once we are done creating vectors, we calculate the distance between every movie—the more the distance between two vectors, the less the similarity.

The distance method would be **the cosine distance method rather than the Euclidean distance:** the smaller the angle, the more the similarity. Euclidean distance is not reliable in higher dimensions. The more data dimension (in our case, 5000D), the less accurate the Euclidean distance.

We are finding distance between each movie with another movie. We import a function from sklearn, and cosine similarity finds the cosine distance of each movie from all other films.



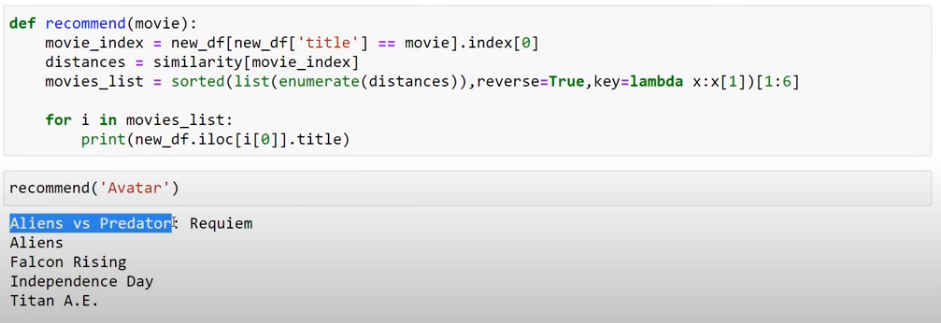
The below picture represents the similarity of each movie with other movies.



The similarity between movie one itself is one and movie two is 0.08, and so on.

**Next Step:** We generated a function in which, if we input one movie, that function would return five similar movies to the input given.

We need to fetch the index of the input movie. We will take out the distance of our input movie from the index from all other films. We will sort the distance and output the first 5 movies from the sorted distance list.



We sorted the similarity count array using enumerate func. Enumerate helped us keep track of movie numbers and their similarity with other movies.

If a user enters a valid movie title, the function prints out the movie’s name most relevant to the input movie.

Once that is done, we do the litmus test to see if the model works. After that, we start building our website.

## Website creation

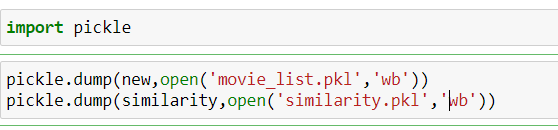
Now we will be using PyCharm for website creation.

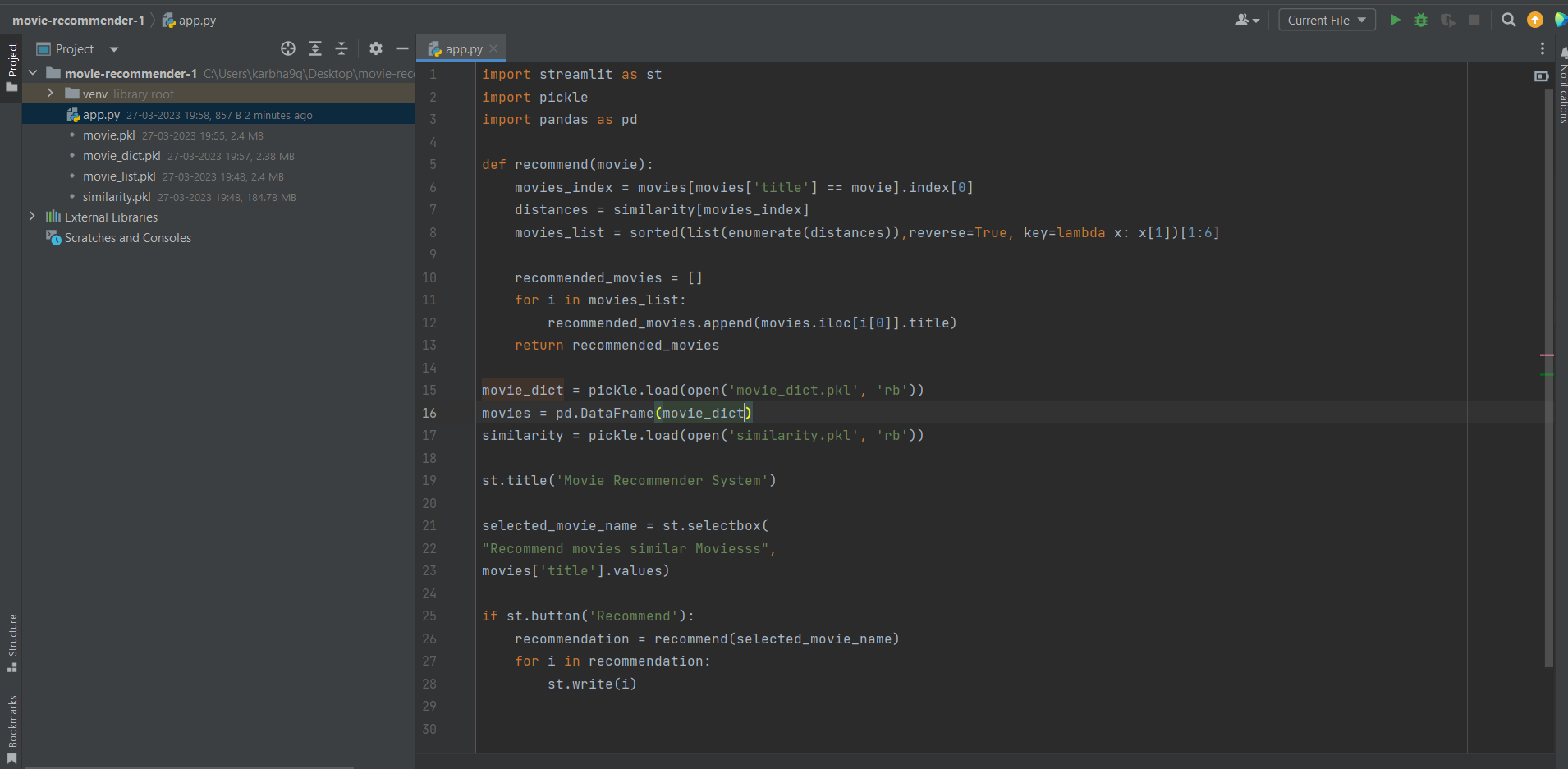
A new project is created in which a python file named “APP” is made.

Flask could also be used, but we are using streamlit, a library.

Streamlit – import, demo run

Now, we want to provide a list of movies to the user. For that, we will be using a library called pickle.





**Now we require the poster of the movie. Movie\_id will help with posters.**

**TM Db API site, we will have many APIs; there is an API in which, if you have any of the movies, you can fetch all the details of a film.**

**The api is called get details GET**/movie/{movie\_id}

https://api.themoviedb.org/3/movie/{movie\_id}?api\_key=<<api\_key>>&language=en-US

<https://api.themoviedb.org/3/movie/67?api_key=884b33da0c5a6ca0e0591fedd664e393&language=en-US>

We will now create a new function , in which we will provide the movie id and that will hit the api , whenever we hit an api , we need a library called request