1. **INTRODUCTION**

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 from the input that is in the form of browsing data that 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/). In order to build our recommendation system, we have used the MovieRecommender Dataset.

In our recommender engine we are using content based recommendation. Content Based Recommendation procedure checks for the adores and aversions of the user and creates a User-based Profile. For producing a user profile, we check for the item profiles and their equivalent user rating. The user profile is the combination of sum of the item profiles where combination being the ratings customer or user has evaluated. After profile of the user has been generated, we estimate the resemblance of the user profile with all the items in the database, which is considered using cosine resemblance between the user generated profile and item profile.

A content based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate. Benefits of Content oriented procedure is that other user’s information or data is not essential, and the recommender system can commend new commodities or anything which are not evaluated presently, nevertheless the recommender system will not recommend the items outside the type of items the user has given ratings to.



**FIGURE 2.0 CONTENT BASED RECOMMENDATION**

The concepts of Term Frequency (*TF*) and Inverse Document Frequency (*IDF*) are used in[information retrieval systems](https://www.analyticsvidhya.com/blog/2015/04/information-retrieval-system-explained/) and also content based filtering mechanisms (such as a content based recommender). They are used to determine the relative importance of a document / article / news item / movie etc.

1. **TOOLS USED**

The tools used to implement this recommendation system are:-

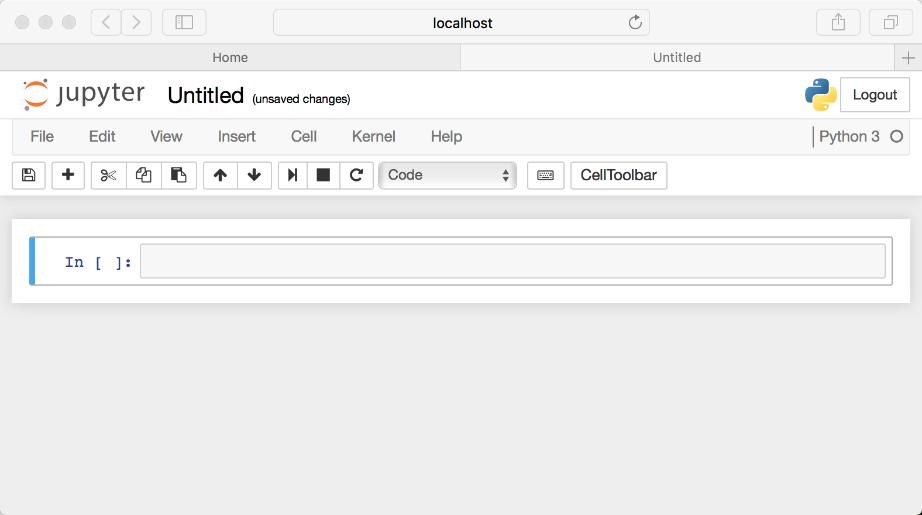
1. Jupyter Notebook

2. Python 3

3. Machine Learning Libraries

**3.1 JUPYTER NOTEOOK**

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Its uses include data cleaning and transformation, numerical simulation, statistical modelling, machine learning and much more. The Jupyter Notebook is a living online notebook, letting faculty and students weave together computational information (code, data, statistics) with narrative, multimedia, and graphs. Faculty can use it to set up interactive textbooks, full of explanations and examples which students can test out right from their browsers. Students can use it to explain their reasoning, show their work, and draw connections between their classwork and the world outside. Scientists, journalists, and researchers can use it to open up their data, share the stories behind their computations, and enable future collaboration and innovation.



**FIGURE 3.0 JUPYTER NOTEBOOK INTERFACE**

**3.2 PYTHON 3**

Python is a general purpose and high level programming language. You can use Python for developing desktop GUI applications, websites and web applications. Also, Python, as a high level programming language, allows you to focus on core functionality of the application by taking care of common programming tasks. it can be used to build just about anything, which will be made easy with the right tools/libraries. Professionally, Python is great for backend web development, data analysis, artificial intelligence, and scientific computing. Develop different applications like web applications, graphic user interface based applications, software development applications, scientific and numeric applications, network programming, games and 3D applications and other business applications.

**3.3 MACHINE LEARNING LIBRARIES**

Essential libraries for Machine Learning in Python. Libraries are sets of routines and functions that are written in a given language. A robust set of libraries can make it easier for developers to perform complex tasks without rewriting many lines of code. Machine learning is largely based upon mathematics. In our recommender we have used the following libraries:-

**pandas, numpy and sklearn.feature\_extraction.text**

**pandas:-** In computer programming, **pandas** is a software **library** written for the **Python** programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

**numpy:- NumPy** is an open source numerical **Python library**. **NumPy** contains a multi-dimentional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays such as trigonometric, statistical and algebraic routines.

**sklearn.feature\_extraction.text:-** The sklearn.feature\_extraction module can be used to extract features in a format supported by machine learning algorithms from datasets consisting of formats such as **text** and image.

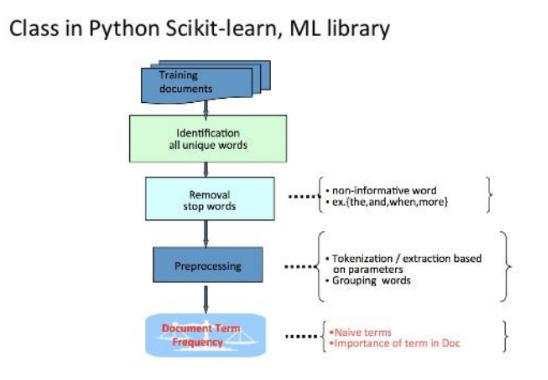
1. **IMPLEMENTATION**

We have used two types of vectorizers in our model. Firstly, vectorization means converting text into vectors and in machine learning vectors are nothing but one dimensional arrays. The two vectorization techniques we used are Count vectorization and TF-IDF vectorization. After the process of vectorization is done, Cosine similarity and Jaccard similarity techniques are applied and the text similarity is known.

**4.1 COUNT VECTORIZATION**

The [CountVectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. You can use it as follows: Create an instance of the CountVectorizer class.

* Call the fit() function in order to learn a vocabulary from one or more documents.
* Call the transform() function on one or more documents as needed to encode each as a vector.



**FIGURE 4.1.0 COUNT VECTORIZATION**

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. Because these vectors will contain a lot of zeros, we call them sparse. Python provides an efficient way of handling sparse vectors in the [scipy.sparse](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html) package. The vectors returned from a call to transform() will be sparse vectors, and you can transform them back to numpy arrays to look and better understand what is going on by calling the toarray() function.

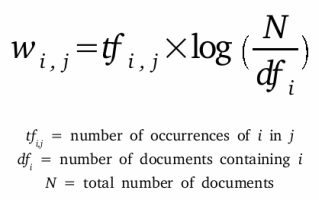
**4.2 TFIDF VECTORIZATION**

One issue with simple counts is that some words like “the” will appear many times and their large counts will not be very meaningful in the encoded vectors. An alternative is to calculate word frequencies, and by far the most popular method is called [TF-IDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf). This is an acronym than stands for “Term Frequency – Inverse Document” Frequency which are the components of the resulting scores assigned to each word.

* Term Frequency: This summarizes how often a given word appears within a document.
* Inverse Document Frequency: This downscales words that appear a lot across documents.

Without going into the math, TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.

The [TfidfVectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) will tokenize documents, learn the vocabulary, inverse document frequency weightings, and allow you to encode new documents. Alternately, if you already have a learned CountVectorizer, you can use it with a [TfidfTransformer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html) to just calculate the inverse document frequencies and start encoding documents. The same create, fit, and transform process is used as with the CountVectorizer.

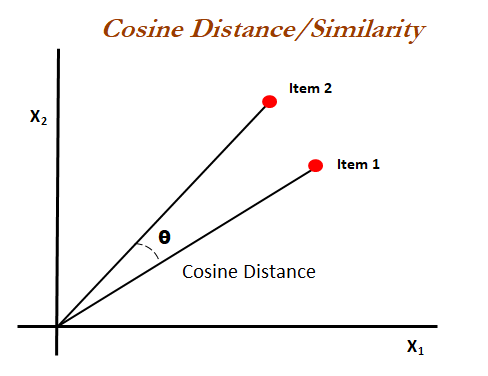


**FIGURE 4.2.0 FORMULA FOR WORD COUNT IN TFIDF VECTORIZER**

**4.3 COSINE SIMILARITY**

A commonly used approach to match similar documents is based on counting the maximum number of common words between the documents. Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors I am talking about are arrays containing the word counts of two documents. When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead.

The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size (like, the word ‘movie’ appeared 50 times in one document and 10 times in another) they could still have a smaller angle between them. Smaller the angle, higher the similarity.



**FIGURE 4.3.0 COSINE SIMILARITY**

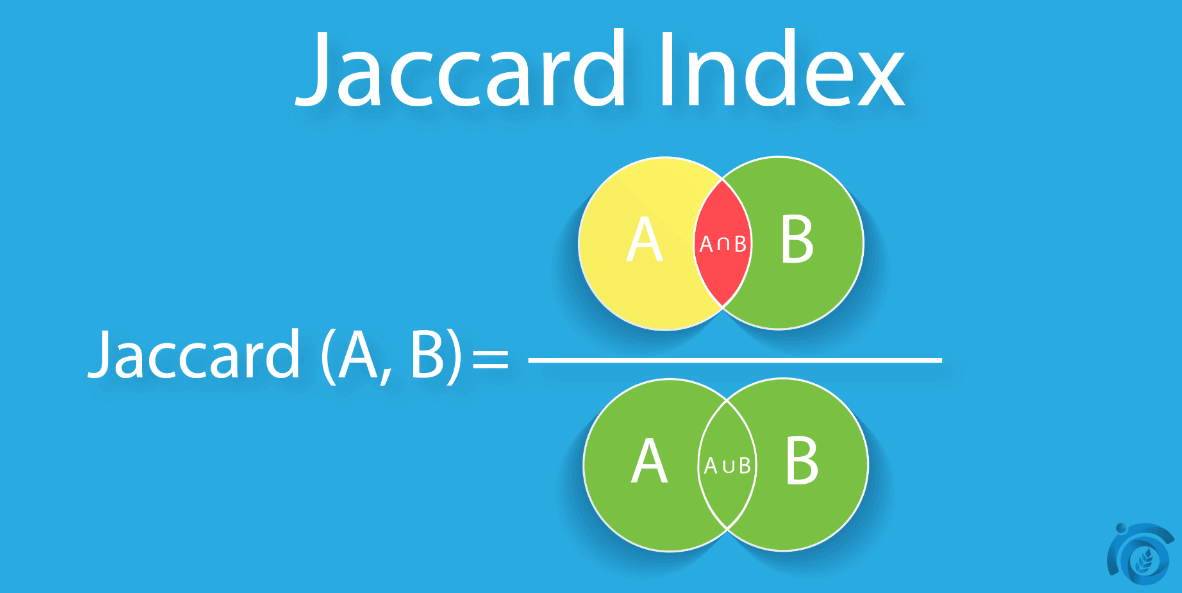
**4.4 JACCARD SIMILARITY**

Jaccard similarity and cosine similarity are two very common measurements while comparing item similarities and today, Similarity measures are used in various ways, examples include in plagiarism, asking a similar question that has been asked before on Quora, collaborative filtering in recommendation systems, etc.

The Jaccard similarity index compares members for two sets to see which members are shared and which are distinct. It’s a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. Although it’s easy to interpret, it is extremely sensitive to small samples sizes and may give erroneous results, especially with very small samples or data sets with missing observations.

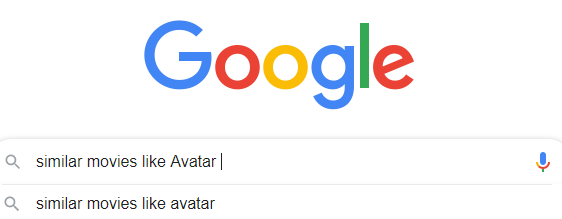
The formula to find the Index is: Jaccard Index = (the number in both sets) / (the number in either set) \* 100.

**The same formula in notation is: J(X,Y) = |X∩Y| / |X∪Y|**



**FIGURE 4.4.0 JACCARD COEFFICIENT**

**5. INPUT AND DATASET**

User can give a movie name of his or her choice. And applying the vectorization techniques and similarity technique the output is generated and displayed immediately.

**FIGURE 5.0 SAMPLE INPUT**

**DATASET**

* Our dataset movie recommender.csv has all the movies from Netflix till 2018.
* It nearly has 4800 records and is correctly defined.
* There are fields like Movie title, genre, director, cast, crew, overview, keywords, rating average etc.
* In the dataset the genres column has entries like action, fiction, fantasy, comedy.
* In the dataset the keywords column has entries regarding the important aspects present in the movie.
* In the dataset the language column has entries like en for English, fr for French.
* In the dataset the title entries column has the names of the movies.
* In the dataset the tagline column has taglines of the movies for example Avatar has the tagline of ‘Enter the World of Pandora’.
* In the dataset the cast column has the names of the lead actors in the movie.
* In the dataset the director column has the name of the director of the movie.



**FIGURE 5.1 MOVIE RECOMMENDER DATASET**

1. **CODE**

So, coming to the coding part it starts with downloading the technical requirements and the dataset. Next the flow is as follows.

**Step1**

* import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

df = pd.read\_csv("movie\_dataset.csv")

After downloading the dataset, we need to import all the required libraries and then read the csv file using read\_csv() method

**Step 2**

* features = ['keywords','cast','genres','director']

If you visualize the dataset, you will see that it has many extra info about a movie. We don’t need all of them. So, we choose keywords, cast, genres and director, overview column to use as our feature set.

**Step 3**

* def combine\_features(row):

return row['keywords']+" "+row['cast']+" "+row['genres']+" +row['director']+row[‘overview’]

Next task is to create a function for combining the values of these columns into a single string.

**Step 4**

* for feature in features:

df[feature] = df[feature].fillna('')

df["combined\_features"] = df.apply(combine\_features,axis=1)

we need to call this function over each row of our dataframe. But, before doing that, we need to clean and preprocess the data for our use. We will fill all the NaN values with blank string in the dataframe.

**Step 5**

* cv = CountVectorizer()

count\_matrix = cv.fit\_transform(df["combined\_features"])

we can now feed these strings to a CountVectorizer() object for getting the count matrix.

* tfidf = TfidfVectorizer()

tfidf\_matrix = tfidf.fit\_transform(df["combined\_features"])

we can now feed these strings to a tfidfVectorizer() object for getting the count matrix.

**Step 6**

* cosine\_sim = cosine\_similarity(count\_matrix)
* tfidf\_sim = cosine\_similarity(tfidf\_matrix)

At this point, 60% work is done. Now, we need to obtain the cosine similarity matrix from the count matrix obtained from both vector matrices.

**Step 7**

* def get\_title\_from\_index(index):

return df[df.index == index]["title"].values[0]

def get\_index\_from\_title(title):

return df[df.title == title]["index"].values[0]

Now, we will define two helper functions to get movie title from movie index and vice-versa.

**Step 8**

* movie\_user\_likes = input()

movie\_index = get\_index\_from\_title(movie\_user\_likes)

similar\_movies = list(enumerate(cosine\_sim[movie\_index]))

tfidf\_similar\_movies = list(enumerate(tfidf\_sim[movie\_index]))

sorted\_similar\_movies = sorted(similar\_movies,key=lambda x:x[1],reverse=True)[1:6]

sort\_by\_average\_vote = sorted(sorted\_similar\_movies,key=lambda x:df["vote\_average"][x[0]],reverse=True)

sorted\_tfidf\_similar\_movies = sorted(tfidf\_similar\_movies,key=lambda x:x[1],reverse=True)[1:6]

sort\_by\_average\_vote\_tfidf = sorted(sorted\_tfidf\_similar\_movies,key=lambda x:df["vote\_average"][x[0]],reverse=True)

Next step is to get the title of the movie that the user currently likes. Then we will find the index of that movie. After that, we will access the row corresponding to this movie in the similarity matrix. Thus, we will get the similarity scores of all other movies from the current movie. Then we will enumerate through all the similarity scores of that movie to make a tuple of movie index and similarity score. Now comes the most vital point. We will sort the list similar\_movies according to similarity scores in descending order. Since the most similar movie to a given movie will be itself, we will discard the first element after sorting the movies.

**Step 9**

* i=0

print("Top 5 similar movies to "+movie\_user\_likes+" are:\n")

for element in sort\_by\_average\_vote:

print(get\_title\_from\_index(element[0]))

i=i+1

if i>5:

break

* i=0

print("Top 5 similar movies to "+movie\_user\_likes+" are:\n")

for element in sort\_by\_average\_vote\_tfidf:

print(get\_title\_from\_index(element[0]))

i=i+1

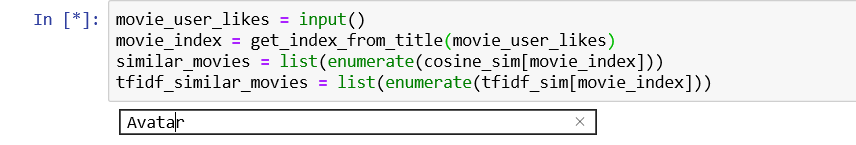
if i>5:

break

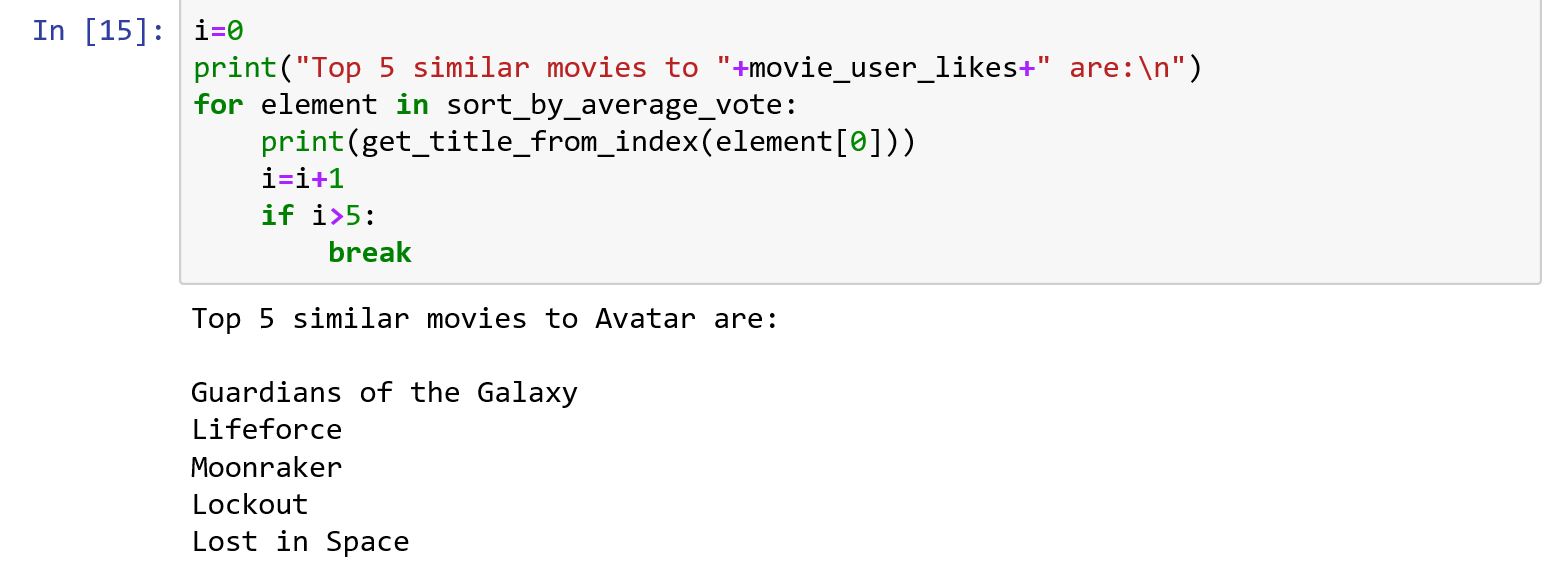
Now, we will run a loop to print first 5 entries from sorted\_similar\_movies list.

1. **OUTPUT**

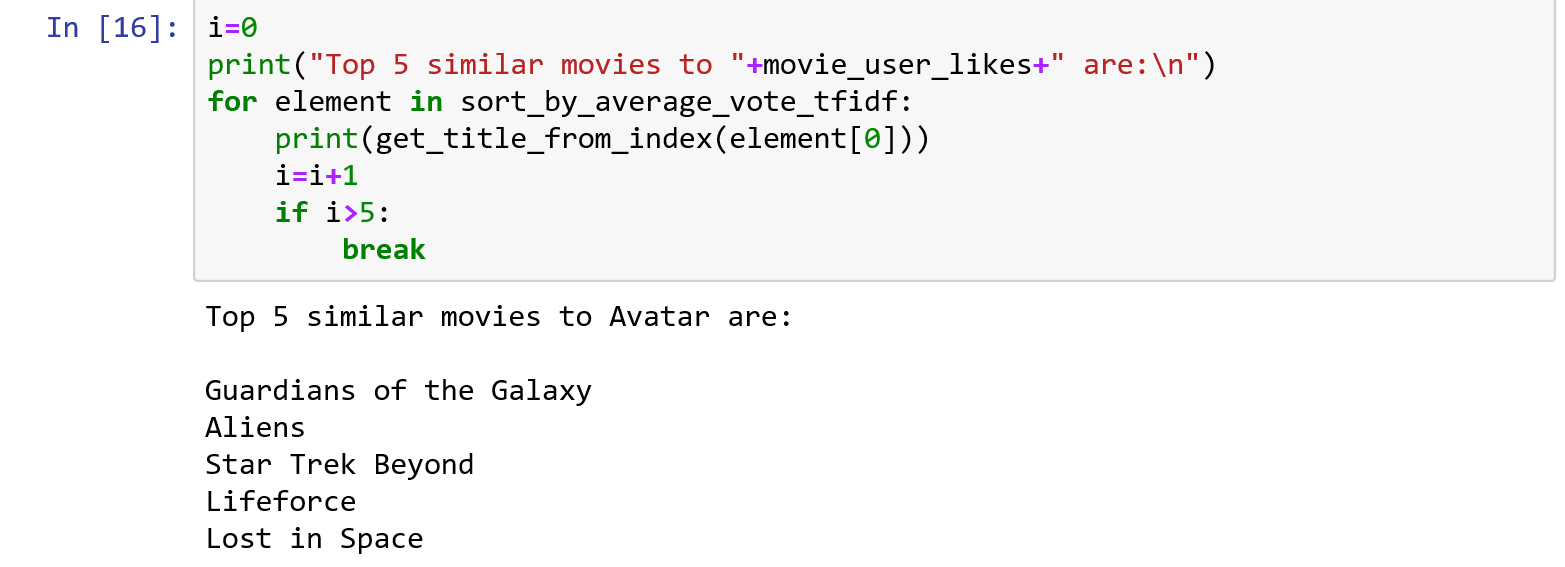
The input looks like the figure 7.0

**FIGURE 7.0 INPUT**

The result comes down as a list for both Count Vectorizer and TFIDF Vectorizer these are shown in figure 7.1 and 7.2.



**FIGURE 7.1 COUNT VECTORIZER**

****

**FIGURE 7.2 TFIDF VECTORIZER**

1. **CONCLUSION AND FUTURE SCOPE**

In this project, by the end we learnt how to build a recommender model that gives the top 5 movies in the desired category and the types of movies you watch. We have used some machine learning libraries to get the desired output. Content-based algorithm & data mining because of which the user will not only be recommended movies but this scheme also delivers the user with additionally advanced and sophisticated endorsements as movies which have a poor rating score in any of the Movie features produced based on data mining will be refined out during the significant allocation platform of the expected three way hybrid movie recommendation system. The future scope of this project is that, as of now we are only doing the recommendations for the movies in Netflix and wanted to extend this feature for all the movies on all platforms and also to develop a live website and integrate the model with it.

**REFERENCES**

[1] <https://www.irjet.net/archives/V5/i2/IRJET-V5I270.pdf>

[2] <https://www.machinelearningplus.com/nlp/cosine-similarity/>

[3] <https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/>

[4][https://techinpink.com/2017/08/04/implementing-similarity-measures-cosine- similarity](https://techinpink.com/2017/08/04/implementing-similarity-measures-cosine-%20%20%20%20%20%20%20%20similarity)

[5]<https://towardsdatascience.com/natural-language-processing-count-vectorization-with-scikit-learn-e7804269bb5e>