Contents

1. Introduction

2. Problem definition and algorithm

2.1 Task definition

2.2 data set description

2.3 Algorithm definition

3. Experimental set up and Implementation

3.1 components to be installed

3.2 methodology

3.2.1 Content based filtering

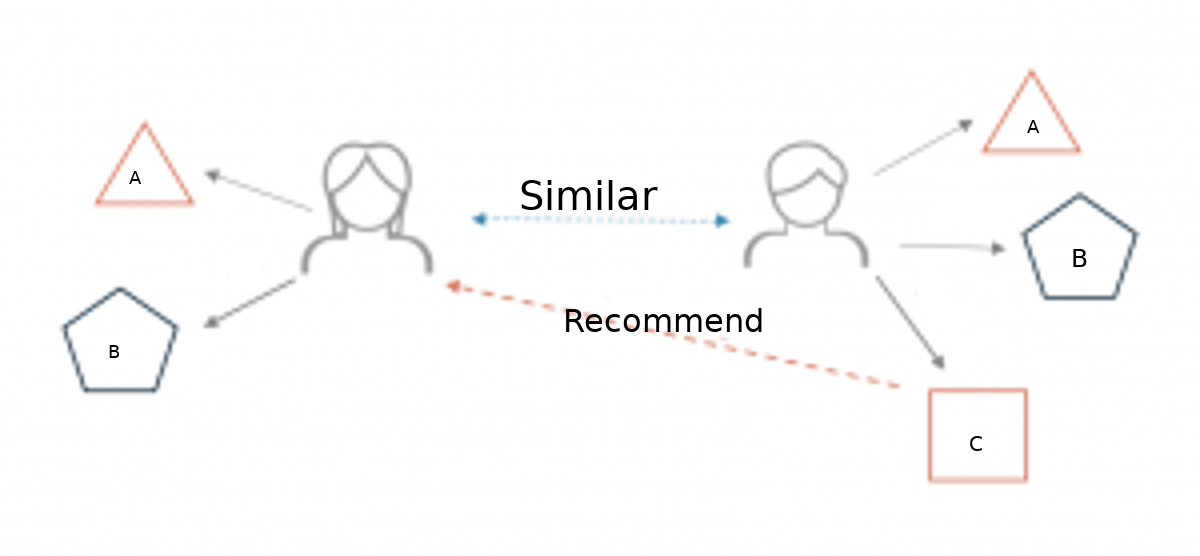
3.2.2 Collaborative filtering

4. Conclusion and Future work

**CHAPTER 1**

**Introduction**

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you. They are also used by Music streaming applications such as Spotify and Deezer to recommend music that you might like.



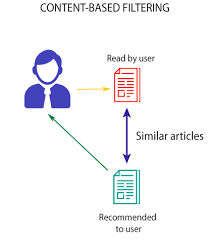
**Types of recommendation system**

There are two methods to construct a recommender system :

1. Content based recommendation
2. Collaborative filtering based recommendation

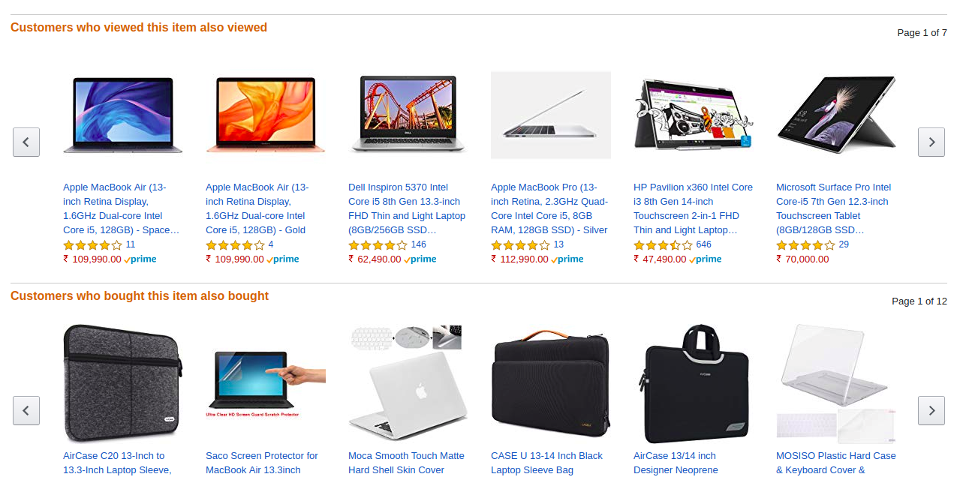
**Content based recommendation**

This type of recommendation systems, takes in a movie that a user currently likes as input. Then it analyzes the contents (storyline, genre, cast, director etc.) of the movie to find out other movies which have similar content. Then it ranks similar movies according to their similarity scores and recommends the most relevant movies to the user.  Such a recommendation would be for instance recommending Infinity War that featured Vin Disiel because someone watched and liked The Fate of the Furious. Similarly you can get music recommendations from certain artists because you liked their music. Content based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

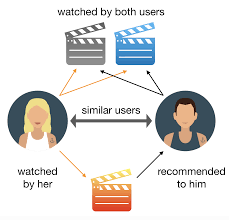


**Collaborative filtering based recommendation**

This algorithm at first tries to find similar users based on their activities and preferences (for example, both the users watch same type of movies or movies directed by the same director). Now, between these users(say, A and B) if user A has seen a movie that user B has not seen yet, then that movie gets recommended to user B and vice-versa. In other words, the recommendations get filtered based on the collaboration between similar user’s preferences (thus, the name “Collaborative Filtering”). One typical application of this algorithm can be seen in the Amazon e-commerce platform, where you get to see the “Customers who viewed this item also viewed” and “Customers who bought this item also bought” list.



In collaborative filtering the behavior of a group of users is used to make recommendations to other users. Recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie.

****

There are two types of collaborative models

* **User-based collaborative filtering**:

In this model products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example if Derrick and Dennis like the same movies and a new movie comes out that Derick likes,then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.

* **Item-based collaborative filtering**

These systems identify similar items based on users’ previous ratings. For example if users A,B and C gave a 5 star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A,B and C.

**CHAPTER 2**

**Task definition and Algorithm**

**2.1Task Definition**

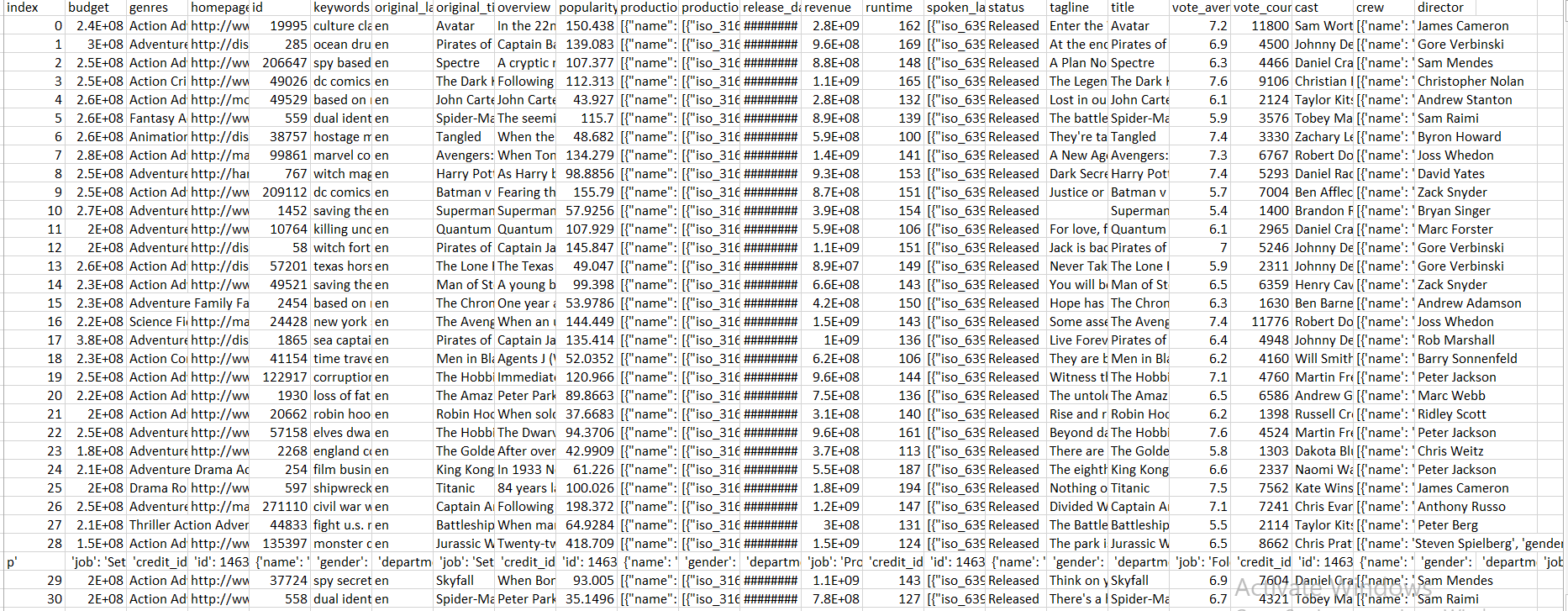
The focus of this project is to develop a movie recommendation system by using

* + Content based filtering
  + Collaborative filtering

**2.2 Dataset Description:**

For content based filtering I used the following dataset. This dataset has following attributes.

* Index
* Budget
* Genres
* Home page
* Id
* Keywords
* Original\_language
* Original\_title
* Overview
* Popularity
* production\_companies
* production\_countries
* release\_date
* revenue
* runtime
* spoken\_languages
* status
* tagline
* title
* vote\_average
* vote\_count
* cast
* crew
* director



Movie dataset for content based filtering

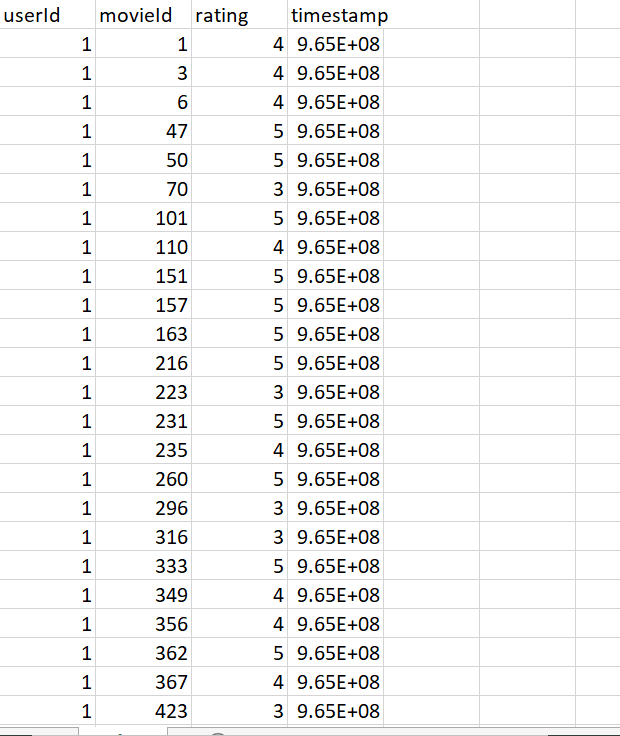
I the above datasets several attributes are available. But For content based filtering we used the following attributes.

* Keywords
* Cast
* Genres
* director

For colaberative filtering I used the following data set



Movie dataset for collaborative filtering



Rating dataset for collaborative filtering

**2.3 Algorithm Definition**

The job of this project is to implement content based filtering and collaborative filtering.

For content based filtering cosine similarity is used.And for collaborative filtering Pearson correlation is used.

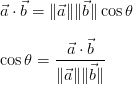
**Cosine similarity**

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

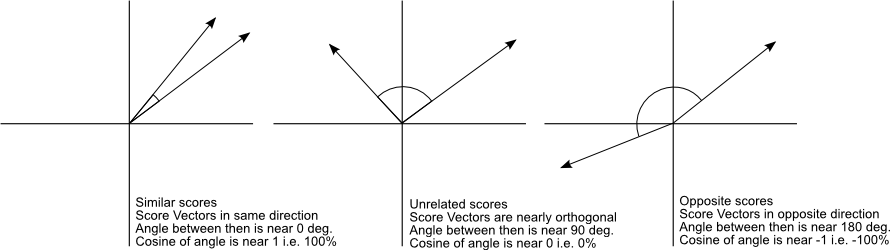
A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a term-frequency vector. For example, in Table 2.5, we see that Document1 contains five instances of the word team, while hockey occurs three times. The word coach is absent from the entire document, as indicated by a count value of 0. Such data can be highly asymmetric.

Cosine similarity is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let *x* and *y* be two vectors for comparison. Using the cosine measure as a similarity function, we have

The cosine similarity between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle between them. This metric is a measurement of orientation and not magnitude, it can be seen as a comparison between documents on a normalized space because we’re not taking into the consideration only the magnitude of each word count (tf-idf) of each document, but the angle between the documents. What we have to do to build the cosine similarity equation is to solve the equation of the dot product for the \cos{\theta}:



And that is it, this is the cosine similarity formula. Cosine Similarity will generate a metric that says how related are two documents by looking at the angle instead of magnitude, like in the examples below:



Even if we had a vector pointing to a point far from another vector, they still could have an small angle and that is the central point on the use of Cosine Similarity, the measurement tends to ignore the higher term count on documents. Suppose we have a document with the word “sky” appearing 200 times and another document with the word “sky” appearing 50, the Euclidean distance between them will be higher but the angle will still be small because they are pointing to the same direction, which is what matters when we are comparing documents.

Let’s explain through an example

Here our recommendation engine will be content based. So, we need to find similar movies to a given movie and then recommend those similar movies to the user. The logic is pretty straightforward.

But, wait…. How can we find out which movies are similar to the given movie in the first place? How can we find out how much similar(or dissimilar) two movies are?

Let us start with something simple and easy to understand.

Suppose, you are given the following two texts:

Text A: London Paris London

Text B: Paris Paris London

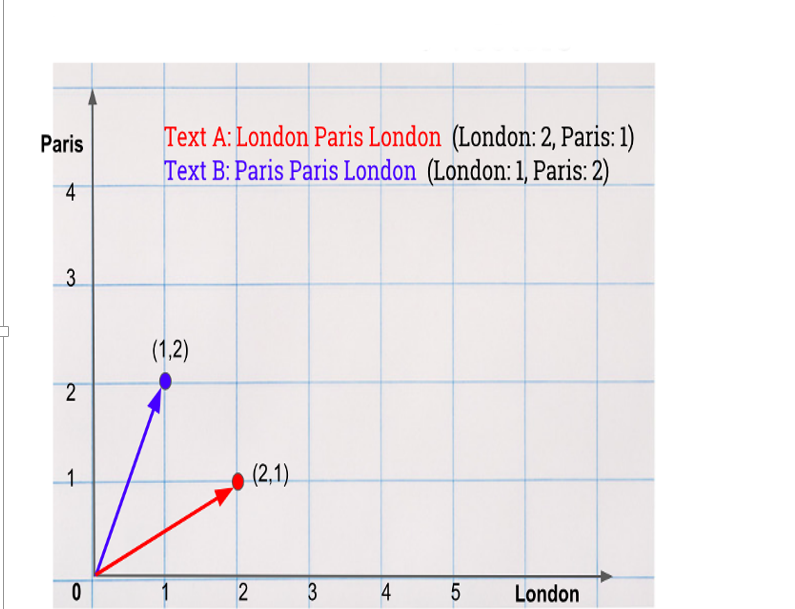
Now let’s find the similarity between Text A and Text B .

Let’s analyze these texts….

1. Text A: Contains the word “London” 2 times and the word “Paris” 1 time.
2. Text B: Contains the word “London” 1 time and the word “Paris” 2 times.

Now, what will happen if we try to represent these two texts in a 2D plane (with “London” in X axis and “Paris” in Y axis)? Let’s try to do this.

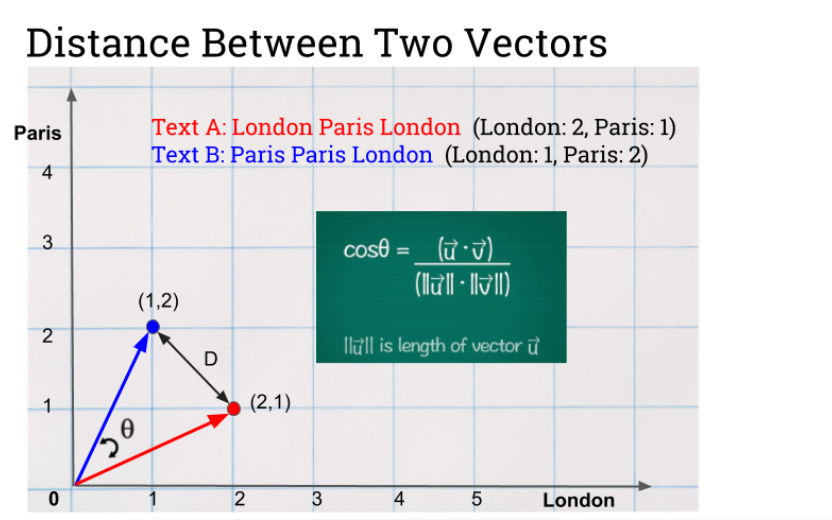
It will look like this-



Here, the red vector represents “Text A” and the blue vector represents “Text B”.

Now we have graphically represented these two texts. So, now can we find out the similarity between these two texts?

These two texts are represented as vectors. Right? So, we can say that two vectors are similar if the distance between them is small. By distance, we mean the angular distance between two vectors, which is represented by θ (theta). By thinking further from the machine learning perspective, we can understand that the value of **cos θ** makes more sense to us rather than the value of θ (theta) because, the cosine(or “cos”) function will map the value of θ in the first quadrant between 0 to 1 (Remember? cos 90° = 0 and cos 0° = 1 ).

****

**We can implement this by using a python program**

Let’s take the following 2 texts

**text = [“London Paris London”,”Paris Paris London”]**

Now, we need to find a way to represent these texts as vectors. The [CountVectorizer()](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html" \t "_blank) class from sklearn.feature\_extraction.text library can do this for us. We need to import this library before we can create a new CountVectorizer() object.

**from sklearn.feature\_extraction.text import CountVectorizer  
cv = CountVectorizer()  
count\_matrix = cv.fit\_transform(text)**

count\_matrix gives us a sparse matrix. To make it in human readable form, we need to apply toarrray() method over it. And before printing out this count\_matrix, let us first print out the feature list(or, word list), which have been fed to our CountVectorizer() object.

**print(cv.get\_feature\_names())  
print(count\_matrix.toarray())**

The output of the above code will look like this-

**[‘london’, ‘paris’]  
[[2 1]  
[1 2] ]**

This indicates that the word ‘london’ occurs 2 times in A and 1 time in B. Similarly, the word ‘paris’ occurs 1 time in A and 2 times in B.

Now, we need to find cosine(or “cos”) similarity between these vectors to find out how similar they are from each other. We can calculate this using [cosine\_similarity()](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html" \t "_blank) function from sklearn.metrics.pairwise library.

**from sklearn.metrics.pairwise import cosine\_similarity  
similarity\_scores = cosine\_similarity(count\_matrix)  
print(similarity\_scores)**

The above code will output a similarity matrix, which looks like this-

**[[1. 0.8]  
[0.8 1. ]]**

We can interpret this output like this-

1. Each row of the similarity matrix indicates each sentence of our input. So, row 0 = Text A and row 1 = Text B.
2. The same thing applies for columns. To get a better understanding over this, we can say that the output given above is same as the following:

**Text A: Text B:  
Text A: [[1. 0.8]**

**Text B: [0.8 1.]]**

Interpreting this, says that Text A is similar to Text A(itself) by 100%(position [0,0]) and Text A is similar to Text B by 80%(position [0,1]). And by looking at the kind of output it is giving, we can easily say that this is always going to output a symmetric matrix. Because, if Text A is similar to Text B by 80% then, Text B is also going to be similar to Text A by 80%.

**Pearson Correlation**

Correlation is a bi-variate analysis that measures the strength of association between two variables and the direction of the relationship. In terms of the strength of relationship, the value of the correlation coefficient varies between +1 and -1. A value of ± 1 indicates a perfect degree of association between the two variables. As the correlation coefficient value goes towards 0, the relationship between the two variables will be weaker. The direction of the relationship is indicated by the sign of the coefficient; a + sign indicates a positive relationship and a - sign indicates a negative relationship.

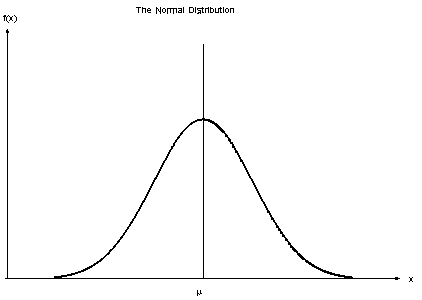
Usually, in statistics, we measure four types of correlations:

* Pearson correlation
* Kendall rank correlation
* Spearman correlation
* Point-Biserial correlation.

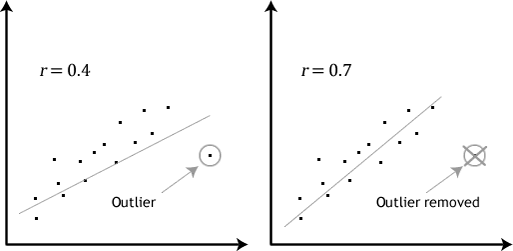
As the title suggests, we’ll only cover Pearson correlation coefficient. I’ll keep this short but very informative so you can go ahead and do this on your own. Pearson correlation coefficient is a measure of the strength of a linear association between two variables — denoted by r. You’ll come across Pearson r correlation

## **Assumptions**

1. For the Pearson r correlation, both variables should be **normally distributed**. i.e the normal distribution describes how the values of a variable are distributed. This is sometimes called the ‘Bell Curve’ or the ‘Gaussian Curve’. A simple way to do this is to determine the normality of each variable separately using the Shapiro-Wilk Test.



There should be **no significant outliers**. We all know what outliers are but we don’t know the effect of outliers on Pearson’s correlation coefficient, r. Pearson’s correlation coefficient, r, is very sensitive to outliers, which can have a very large effect on the line of best fit and the Pearson correlation coefficient. This means — including outliers in your analysis can lead to misleading results.



Each variable should be **continuous**i.e. interval or ratios for example weight, time, height, age etc. If one or both of the variables are ordinal in measurement, then a Spearman correlation could be conducted instead.

1. The two variables have a **linear relationship**. Scatter plots will help you tell whether the variables have a linear relationship. If the data points have a straight line (and not a curve), then the data satisfies the linearity assumption. If the data you have is not linearly related you might have to run a non-parametric .



The observations are **paired observations.**That is, for every observation of the independent variable, there must be a corresponding observation of the dependent variable. For example if you’re calculating the correlation between age and weight. If there are 12 observations of weight, you should have 12 observations of age.**i.e. no blanks.**

6. **Homoscedascity**. I’ve saved best for last. The hard is hard to pronounce but the concept is simple. Homoscedascity simply refers to ‘**equal variances**’. A scatter-plot makes it easy to check for this. If the points lie equally on both sides of the line of best fit, then the data is homoscedastic. As a bonus — the opposite of homoscedascity is heteroscedascity which refers to refers to the circumstance in which the variability of a variable is unequal across the range of values of a second variable that predicts it.

Python is an amazing language for data analytics, primarily because of the fantastic ecosystem of data-centric Python packages. **Pandas**is one of those packages and makes importing and analyzing data much easier.

Pandas**dataframe.corr()** is used to find the pairwise correlation of all columns in a dataframe.

*Any na values are automatically excluded.*

*Any non-numeric data type column in the dataframe will be ignored.*

dataframe.corr parameters: **dataframe.corr(method='',min\_periods=1)**

method: {‘pearson’, ‘kendall’, ‘spearman’} or callable

* pearson: standard correlation coefficient
* kendall: Kendall Tau correlation coefficient
* spearman: Spearman rank correlation

min\_periods : int, optional

* Minimum number of observations required per pair of columns to have a valid result. This is currently only available for pearson and spearman correlation

# **An Example**

First we import the packages that we need:

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sb

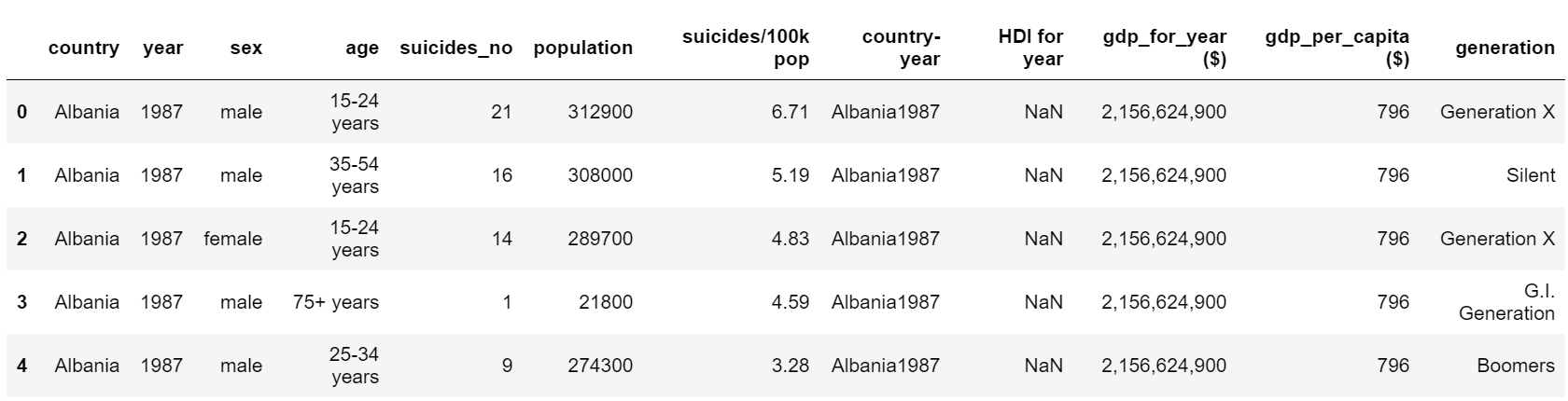
Now we need some data. For this, I’ve chosen a simple but interesting data set that I came across on Kaggle.

Let’s read the data and put it in a dataframe.

SuicideRate = pd.read\_csv("suicide-rates-overview-1985-to-2016.csv")

If you want a glance of the data on Python and have a look at what data is in there.

SuicideRate.head()



# **Calculating the Pearson Corellation**

We’ll use method = ‘pearson’ for the dataframe.corr since we want to calculate the pearson coefficient of correlation. Then we’ll print it out!

pearsoncorr = SuicideRate.corr(method='pearson')pearsoncorr



**CHAPTER 3**

**Experimental set up and evaluation**

**3.1 COMPONENTS TO BE INSTALLED**

* Python
* Numpy
* pandas
* Sklearn

**3.1.1 PYTHON**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

1. Python is Interpreted − Python is processed at runtime by the interpreter. We do not need to compile wer program before executing it. This is similar to PERL and PHP.

2. Python is Interactive − We can actually sit at a Python prompt and interact with the interpreter directly to write wer programs.

3. Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

4. Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

**3.1.3. PANDAS**

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.

Prior to Pandas, Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.

Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

**3.1.4 SKLEARN**

Scikit-learn is probably the most useful library for machine learning in Python. It is on NumP and matplotlib, this library contains a lot of effiecient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

# **CountVectorizer**

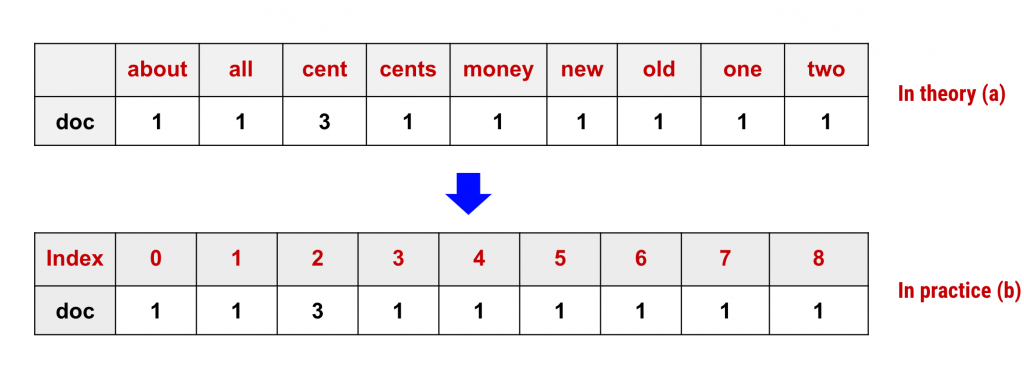
Scikit-learn’s [CountVectorizer](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) is used to transform a corpora of text to a vector of term / token counts. It also provides the capability to preprocess your text data prior to generating the vector representation making it a highly flexible feature representation module for text.

## **Example of How CountVectorizer Works**

To show you an example of how CountVectorizer works, let’s take the book title below (for context: this is part of a [book series that kids love](https://amzn.to/366dOj9)) :

**doc=["One Cent, Two Cents, Old Cent, New Cent: All About Money"]**

This text is transformed to a sparse matrix as shown in Figure (b) below:



CountVectorizer sparse matrix representation of words. (a) is how you visually think about it. (b) is how it is really represented in practice.

Notice that here we have **9 unique words**. So 9 columns. Each column in the matrix represents a unique word in the vocabulary, while each row represents the document in our dataset. In this case, we only have one book title (i.e. the document), and therefore we have only 1 row. The values in each cell are the word counts. Note that with this representation, counts of some words could be 0 if the word did not appear in the corresponding document.

While visually it’s easy to think of a word matrix representation as Figure (a), in reality, these words are transformed to numbers and these numbers represent positional index in the sparse matrix as seen in Figure (b).

With CountVectorizer we are converting raw text to a numerical vector representation of words and [n-grams](https://kavita-ganesan.com/what-are-n-grams/). This makes it easy to directly use this representation as features (signals) in Machine Learning tasks such as for [text classification](https://kavita-ganesan.com/news-classifier-with-logistic-regression-in-python/) and clustering.

Note that these algorithms only understand the concept of numerical features irrespective of its underlying type (text, image pixels, numbers, categories and etc.) allowing us to perform complex machine learning tasks on different types of data.

.

**3.2 Methodology**

**3.2.1 Content based Recommendation**

**Flow diagram**

MovieData.csv

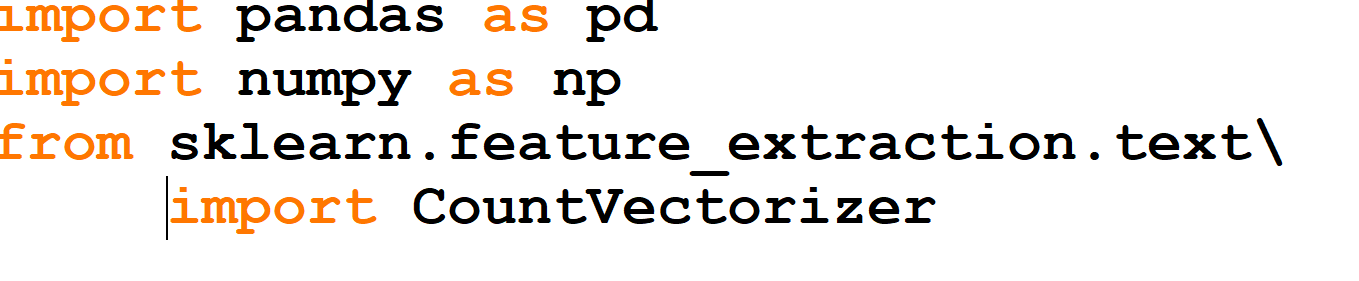
Tokenization

Preprocessing Text

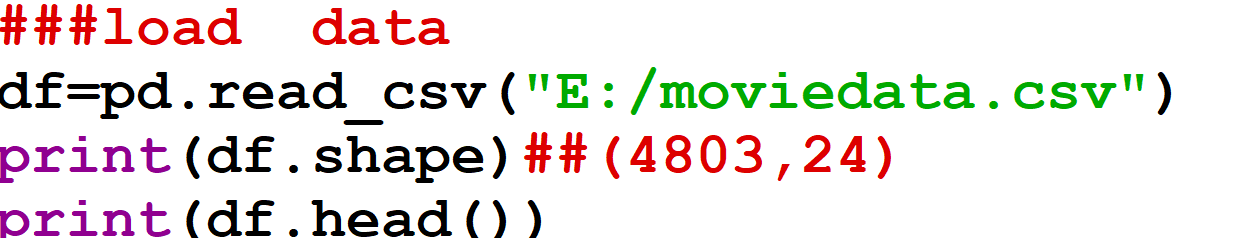
Cosine similarity

Recommended Output

**Step 1 import packages**

****

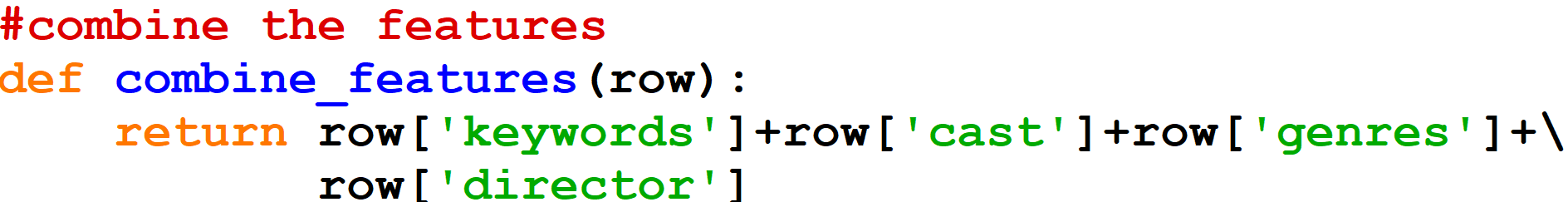
**Step 2 load movie data**

****

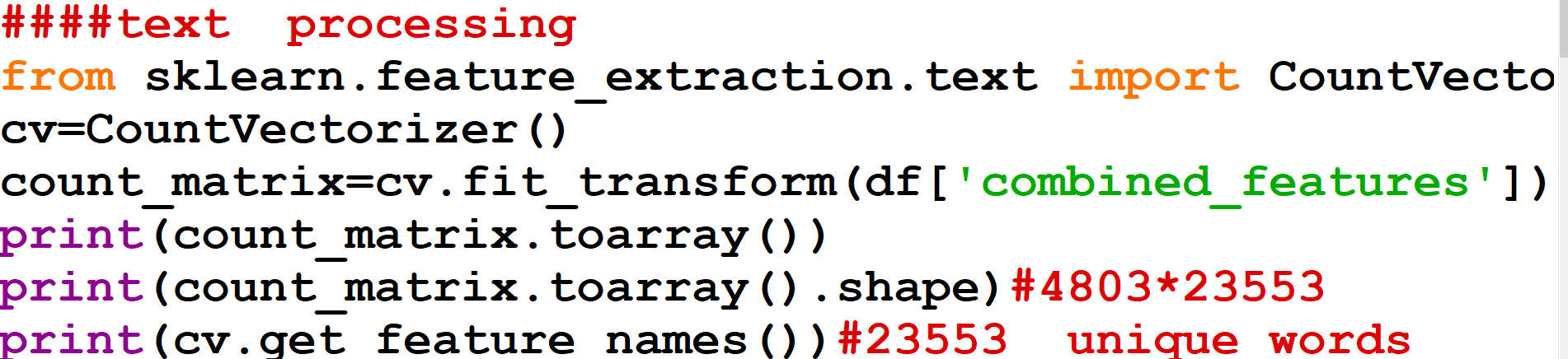
**Step 3 select features**

****

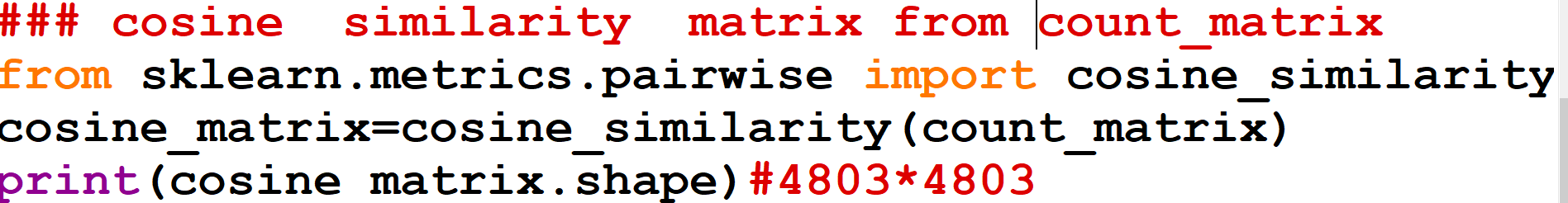
**Step 4 Combine features**

****

**Step 5 text processing by count vectorizer**

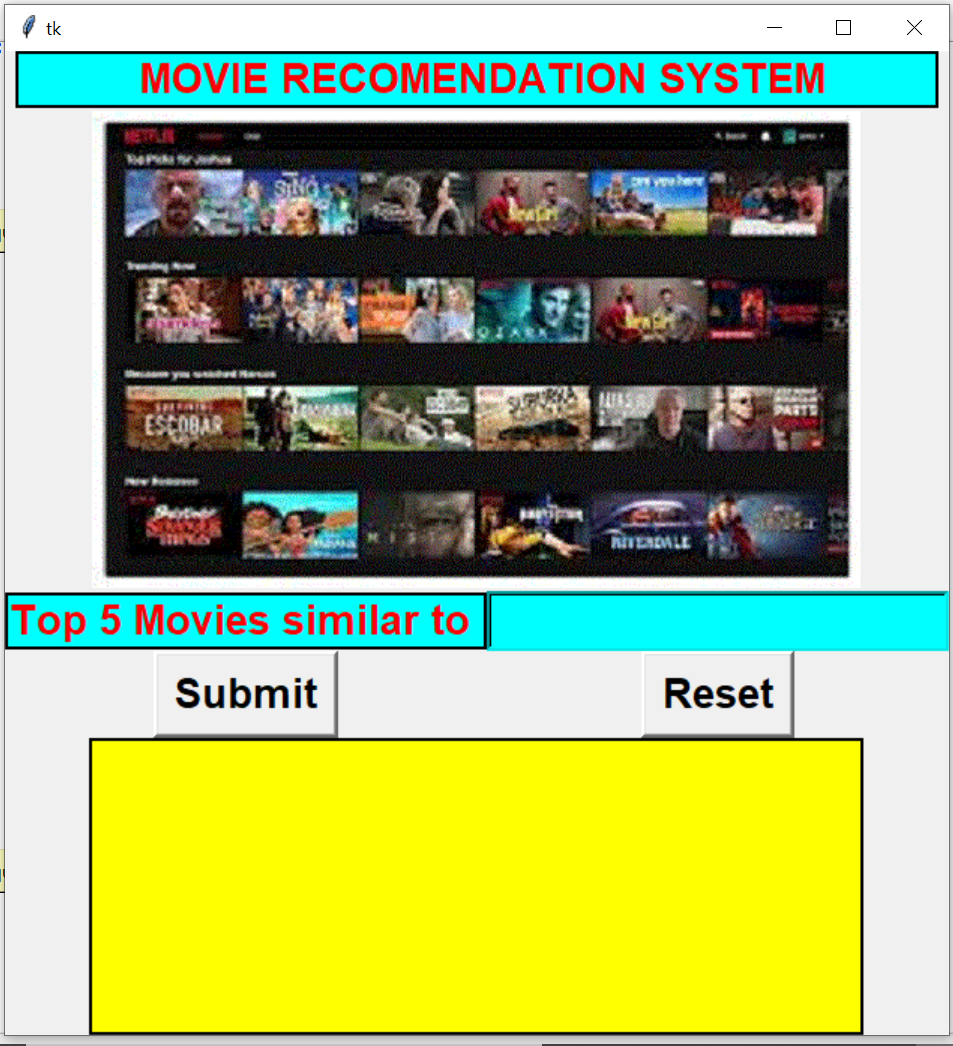
****

**Step 6 Make recommendation by cosine similarity**

****

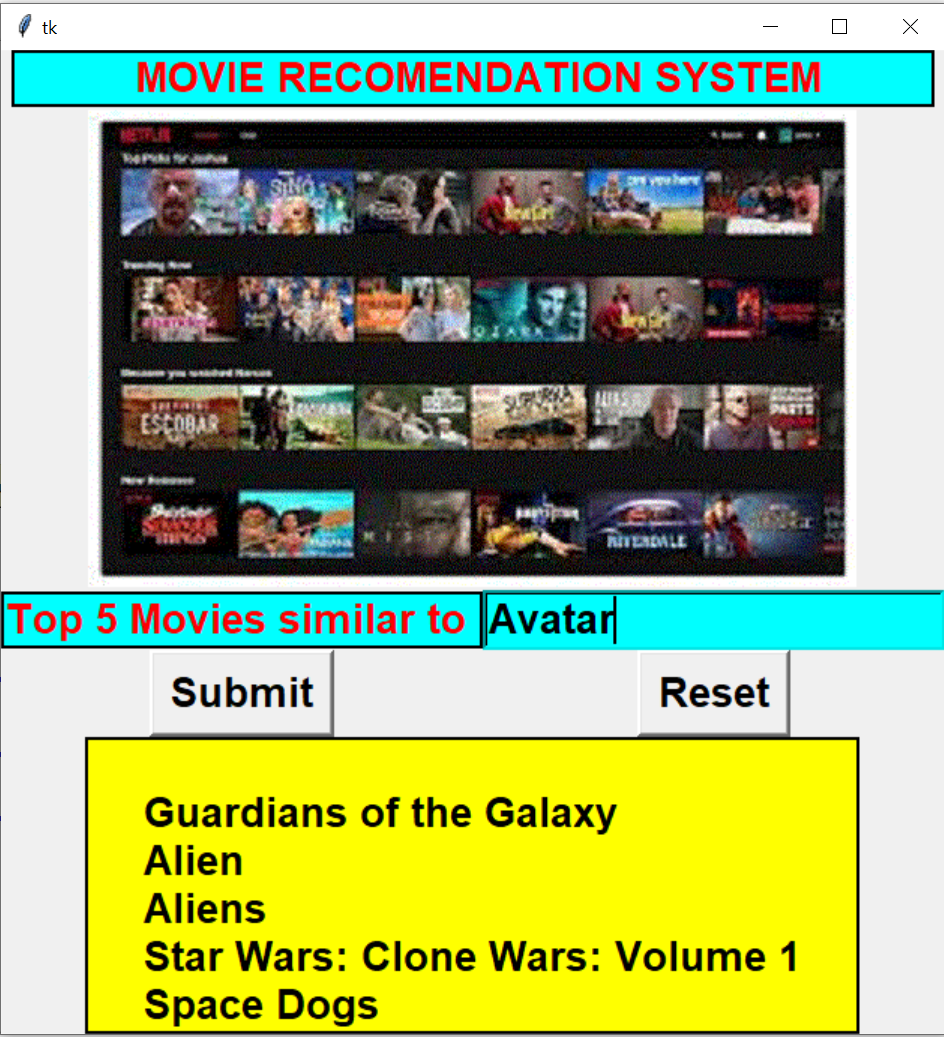
**Graphical User Interface**

**Before Recomendation**

****

**Before Recomendation**

**After Recommendation**

****

**After Recommendation**

**3.2.2 Collaborative Recommendation**

**Flow diagram**

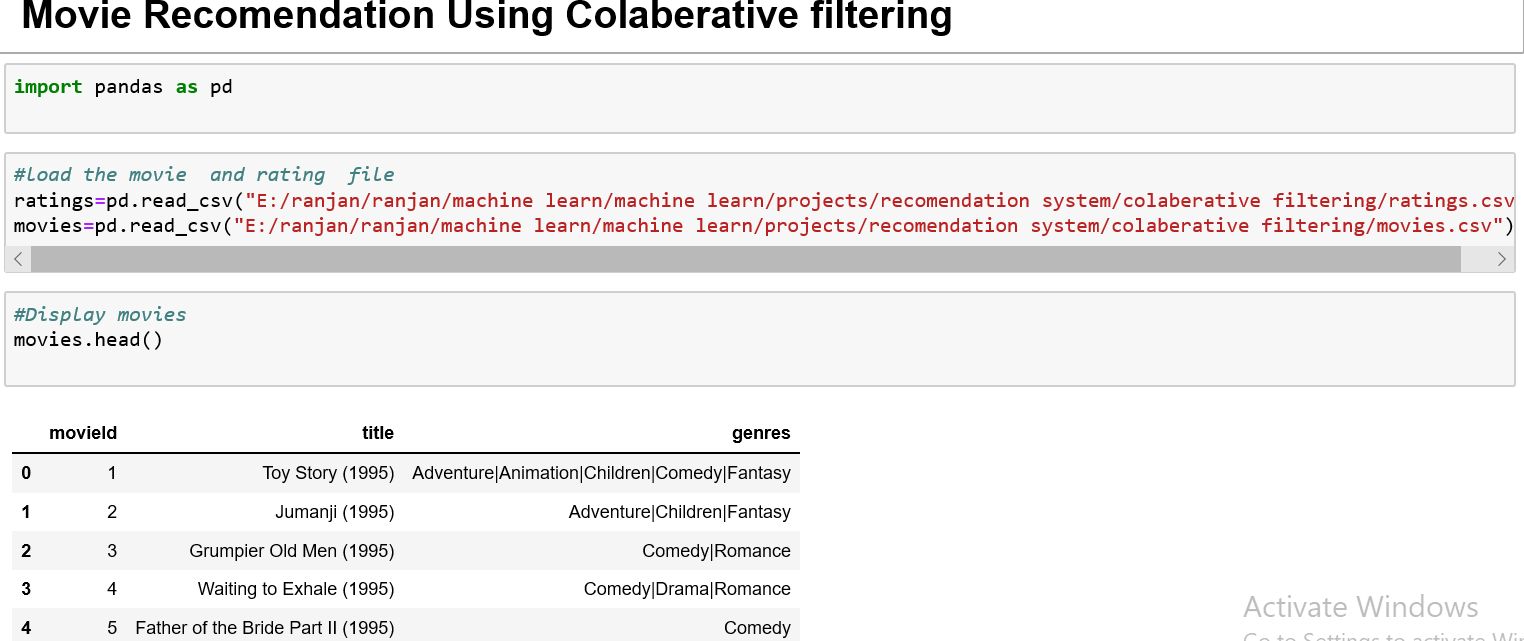
MovieData.csv

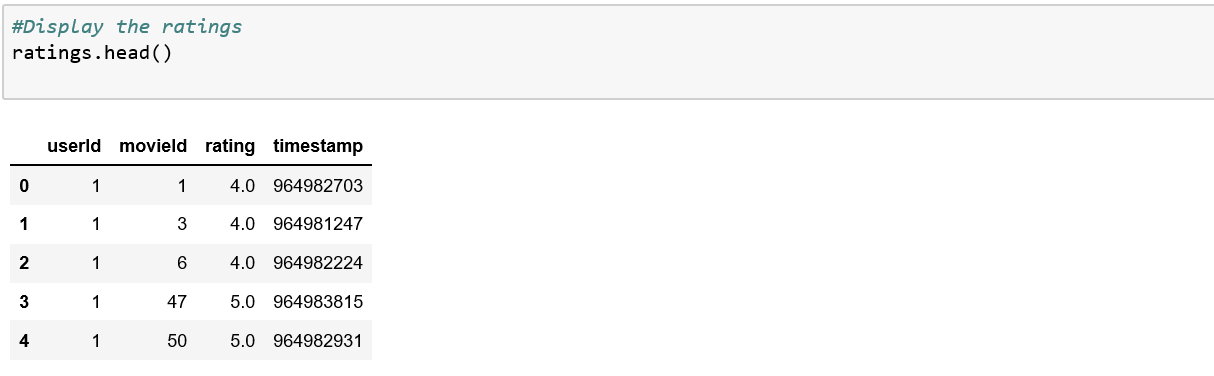
Rating.csv

Pearson Correlation

Recommended Output

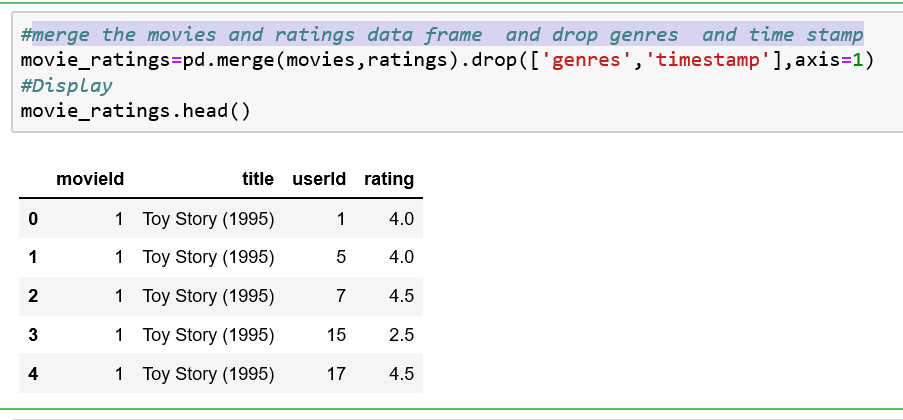
**Step1 import packages and load movie and rating data**

****

****

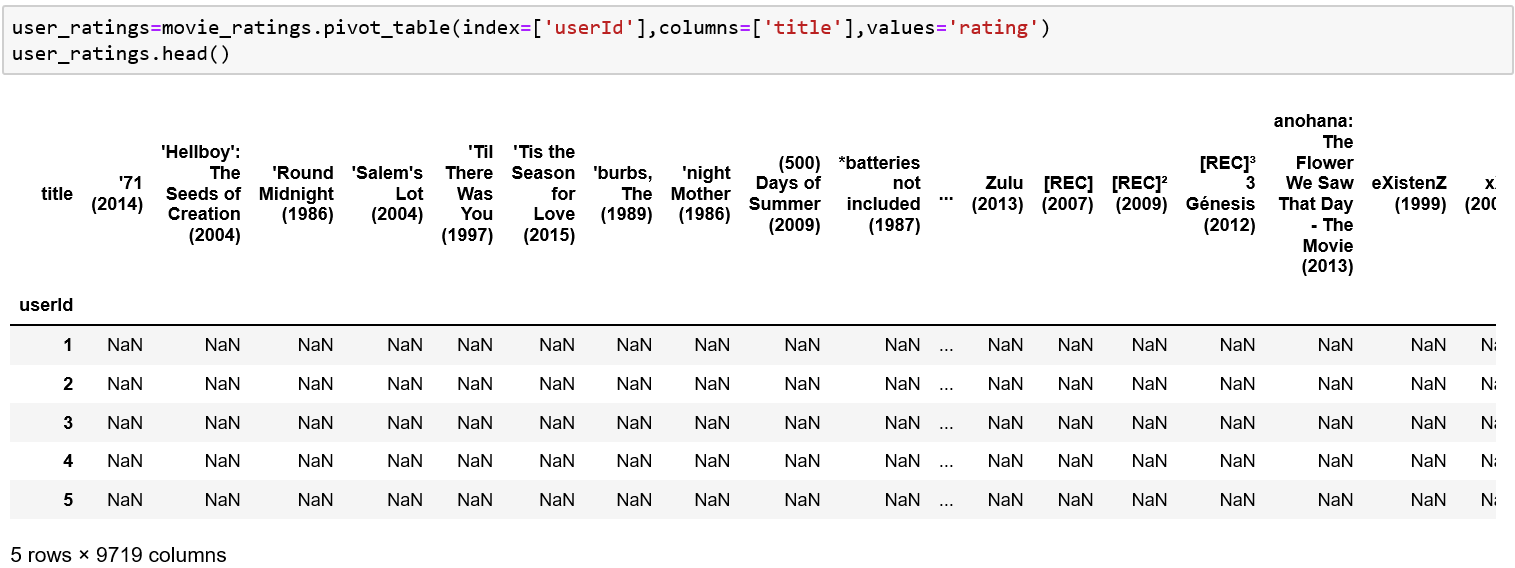
**Step 2**

**merge the movies and ratings data frame and drop genres and time stamp**

****

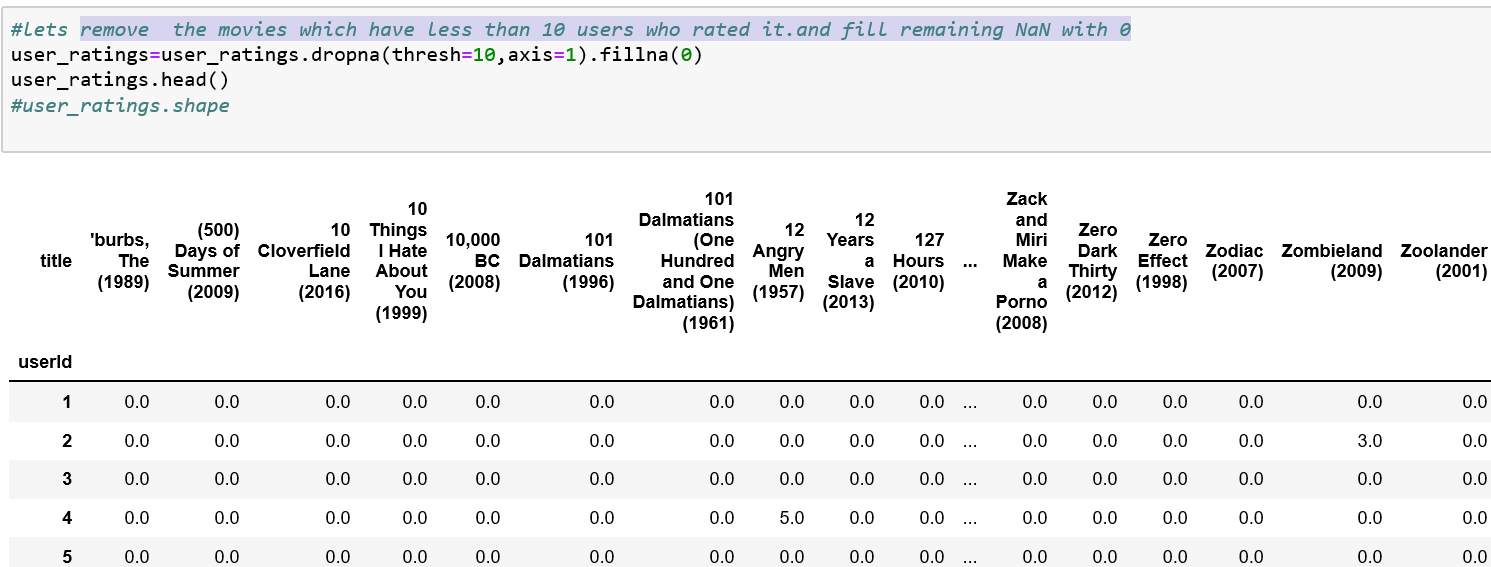
**Step3**

**Find the pivot table**

****

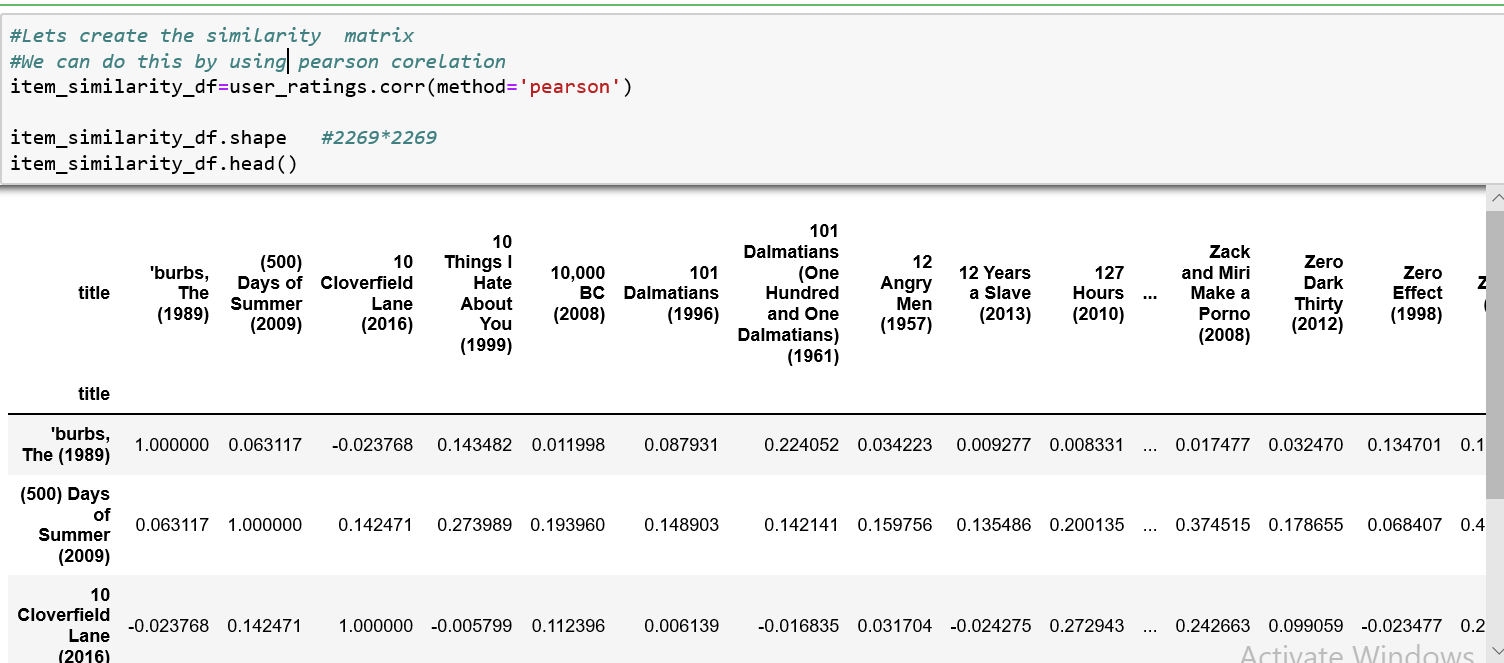
**Step 5**

Remove the movies which have less than 10 users who rated it.and fill remaining NaN with 0 .

****

**Step 6**

**Create the similarity matrix by using Pearson correlation**

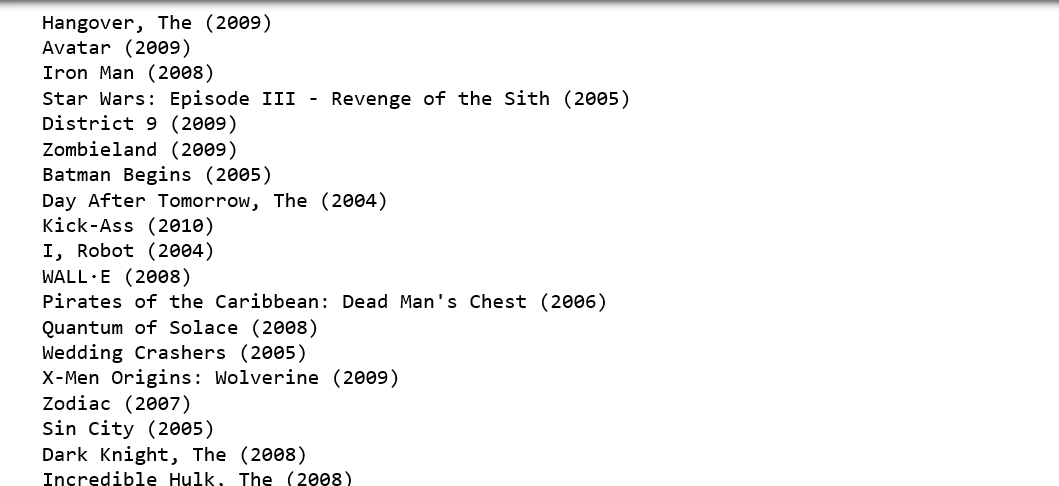
****

**Step 7**

**Make prediction**

****

**Output**

****

These are the recommended movies.

**CHAPTER 4**

**Conclusion and Future work**

In this project content based filtering and collaborative filtering methods are used for recommendation of movie. In contentment based filtering cosine similarity concept is used. And in collaborative filtering Pearson correlation concept is used for recommending movie. Both the concepts works fine. In future we can work on matrix factorization and hybrid recommender using clustering for better performance. Our approach can be further extended to other domains to recommend songs, news, venue, books, tourism and e-commerce etc.