**Database Details**

The movie database for this project was taken from [**here**](https://github.com/Praful2000/YoutubeLectures/tree/master/Movie%20KNN)

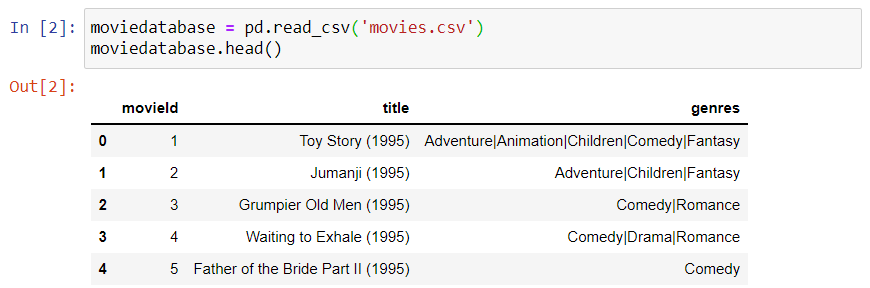
The movie data is split into two separate files: **movies.csv** and **ratings.csv**. The former contains details about the movie under the headings:

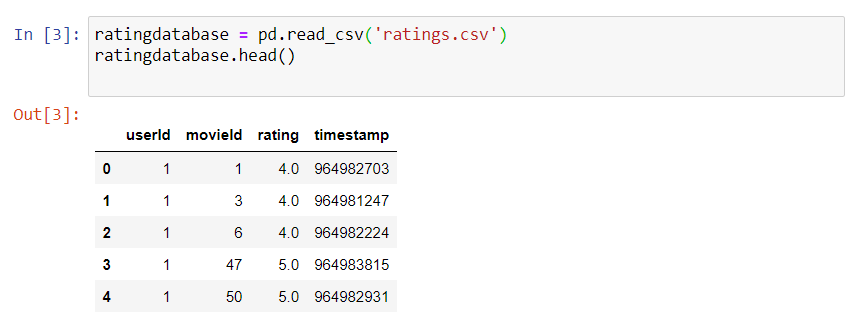
* **movieId** : Numeric ID of the selected movie in the database
* **title** : The name of the movie along with the year it was created in
* **genre** : One or multiple genres the selected movie belongs to

The latter, **ratings.csv** contains details about multiple users and how they have rated the movies under the following headings:

* **userId** : The numeric ID of the users who have rated the movies present in the database
* **movieId** : The name of the movie along with the year it was created in
* **rating** : Ratings assigned to the movies by the users ranging from **0.0** to **5.0**
* **timestamp** : Showcases the length (or timestamp) of the given movie

The initial structure of **movies.csv** and **ratings.csv** is shown in the images below:





In total with both databases included the net data includes **9,472 movies** and **100,836 user ratings** which are all used to compute the recommendations for a selected movie.

**Help Document**

The following code is written completely in native **python** with the help of a few additional python libraries. Here are all the dependencies and libraries used in the program file:

* **pandas** : Used for database computation and cleaning up given databases. The documentation for the library can be found [**here**](https://pandas.pydata.org/docs/user_guide/index.html#user-guide).
* **scipy** : Used to convert the sparse matric to a non-sparse matrix. The documentation for the library can be found [**here**](https://docs.scipy.org/doc/).
* **fuzzywuzzy** : AI python library used to aid search processes in the main recommender functions. The documentation for the library can be found [**here**](https://pypi.org/project/fuzzywuzzy/#description).
* **sklearn** : python library used to implement the KNN algorithm. The documentation for the library can be found [**here**](https://scikit-learn.org/stable/).

The entire code is written in just one executable python file, namely **recommendersystem.py**. The file takes no inputs, but outputs a list of 20 recommended movies based on the movie initially provided to the algorithm. The execution steps are as follows:

python recommendersystem.py

(make sure that all files are in the same directory)

**Methods and Techniques**

The program **recommendersystem.py** uses the KNN (K Nearest Neighbors) algorithm to find out the similarity between two users and accordingly recommends a list of 20 movies based on the initial movie provided. The code uses the following methods and techniques to achieve the goal:

moviedatabase = pd.read\_csv(‘movies.csv’, usecols = [‘movieId’, ‘title’]) ratingdatabase = pd.read\_csv(‘ratings.csv’, usecols = [‘userId’, ‘movieId’, ‘rating’])

This section of code uses **pandas** to read through the .csv files and extract the important information from **movies.csv** and **ratings.csv** to clean up the data and reduce it to a usable form, in our case, **moviedatabase** and **ratingdatabase**.

featurematrix = ratingdatabase.pivot(index = ‘movieId’, columns = ‘userId’).fillna(0) sparsefeaturematrix = csr\_matrix(featurematrix.values)

The following section of the code makes use of **sklearn** to generate a **feature matrix** from the cleaned up databases with **userID** and **movieID** as rows and columns. the generated matrix is stored in the variable **featurematrix**. Aforementioned matrix is a sparse matrix, hence after cleaning it up and getting rid of all null values it is stored in the variable **sparsefeaturematrix**.

modelcosine = NearestNeighbors(metric = ‘cosine’, algorithm = ‘brute’, n\_neighbors = 25) modelcosine.fit(sparsefeaturematrix)

def cosinemovierecommender(moviename, data, number):

index = process.extractOne(moviename, moviedatabase[‘title’])[2] print(“Preferred movie:”, moviedatabase[‘title’][index], “Index:” , index)

print(“Generating recommendation list…”) distance, indices = modelcosine.kneighbors(data[index], n\_neighbors = number)

for i in indices:

print(moviedatabase[‘title’][i].where(i != index)) #print(distance)

cosinemovierecommender(‘shawshank redemption’, sparsefeaturematrix, 20)

The following section of code makes use of **scipy** and **fuzzywuzzy** to generate a model for the core computation. **modelcosine** generates a data model which when plugged in the function **cosinemovierecommender** outputs a list of movies derived from the algorithm which are the closest to the provided movie (in this case, **Shawshank Redemption**).

For computational purposes, the algorithm uses three different metrics to compute similarity, namely **cosine similarity**, **manhattan distance** and **euclidean distance**. The functions **manhattanmovierecommender** and **euclideanmovierecommender** use the other two metrics to calculate similarity using said metrics.

**Results**

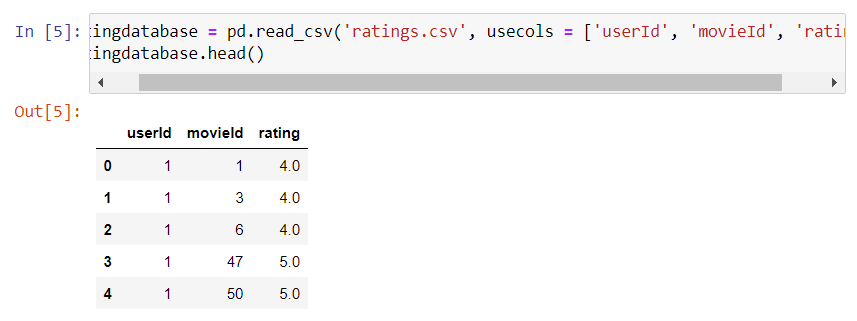
The even though the program only outputs a list of desired movies, it does a few things to aid the final outcome.

moviedatabase = pd.read\_csv(‘movies.csv’, usecols = [‘movieId’, ‘title’]) moviedatabase.head()

ratingdatabase = pd.read\_csv(‘ratings.csv’, usecols = [‘userId’, ‘movieId’, ‘rating’]) ratingdatabase.head()

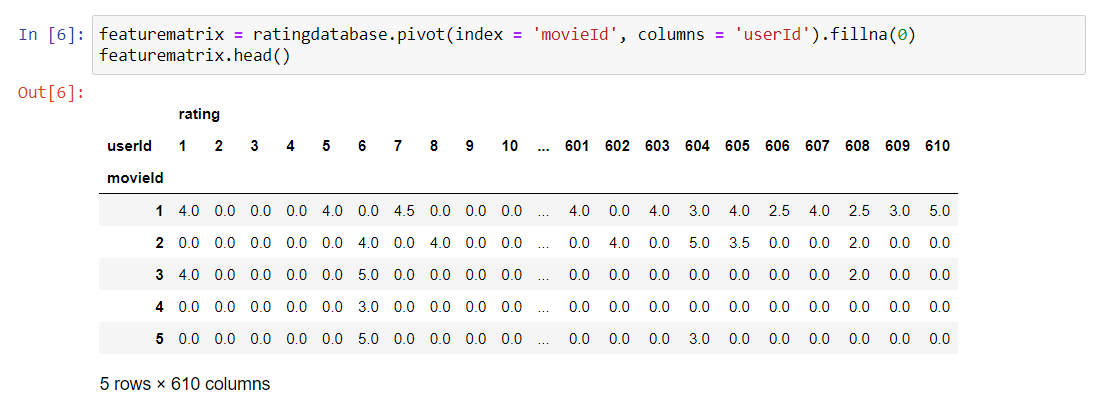
These two command statements clean and reduce the initial datasets into a database finally used in the calculations.





featurematrix = ratingdatabase.pivot(index = ‘movieId’, columns = ‘userId’).fillna(0) featurematrix.head()

This command makes a matrix using the data in **moviedatabase** and **ratingdatabase**.



The final execution of the command results in a list of recommended movies based on the input movie:

