1. **TMDB Movie 5000 Dataset and The Movie Dataset**
   1. **Introduction**

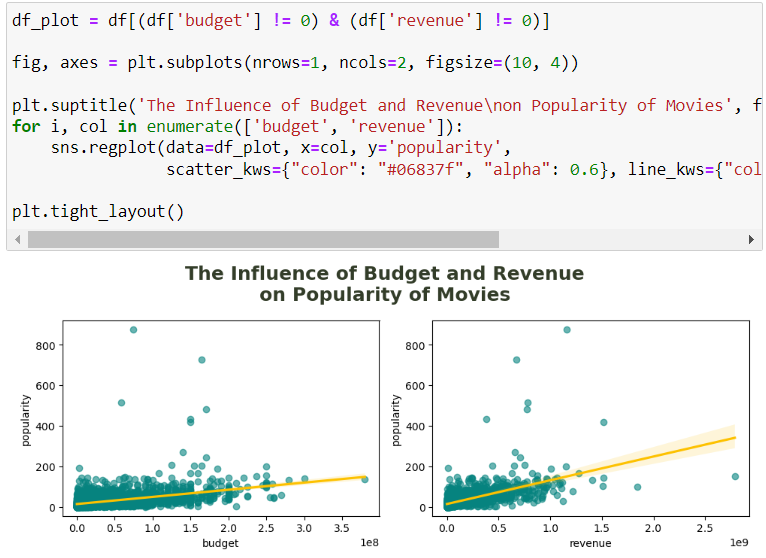
There are three datasets in this recommendation exercise. The first dataset contain four columns, movie id, title, cast and crew. The movie id is in integer while the rest of the columns are in object form. The second dataset comprises twenty columns, seven of which are numeric and the remaining sixteen are object-related. The budget, id, popularity, income, runtime, vote average, and vote count are the seven columns that are numerical.

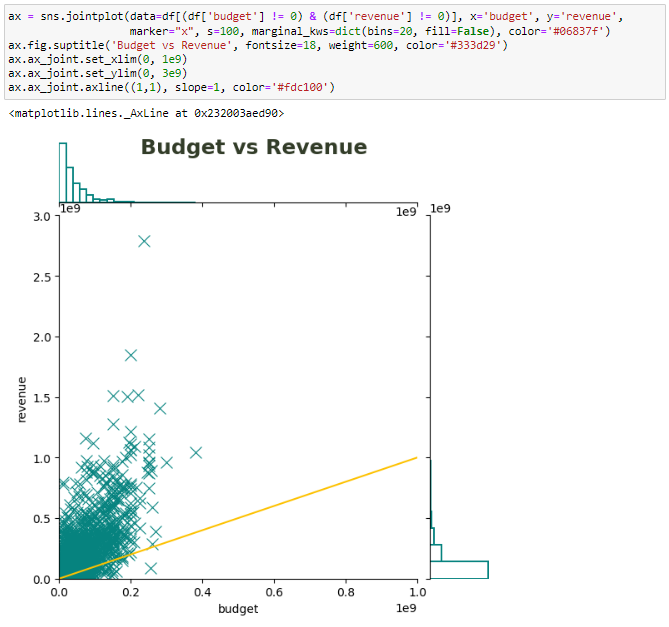
Obtaining the datasets for this activity required downloading them from

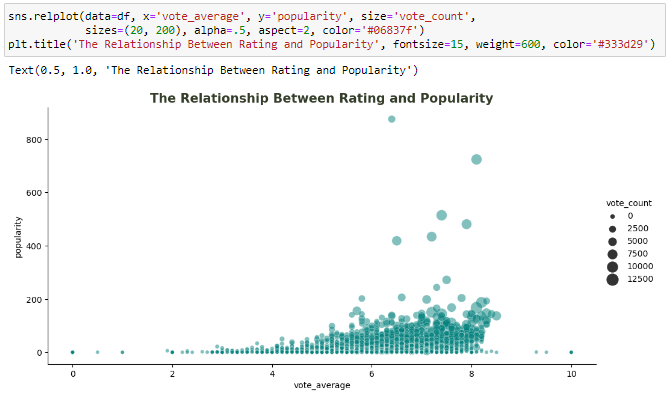
* <https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata/code>
  + The first dataset (tmdb\_5000\_credits.csv) has the attributes listed below:
* A special identification for each movie is its movie\_id.
* cast - A list of the principal and supporting actors.
* The names of the director, editor, composer, writer, etc. are included under crew.
* The tmdb\_5000\_movies.csv second dataset contains the following characteristics:The sum of money used to produce the film.
* genre - The type of film it is, such as action, comedy, or thriller.
* homepage - A link to the movie's homepage.
* id is actually the same as the movie\_id from the first dataset.
* keywords - The words or phrases that describe the film.
* The original language in which the film was produced is original\_language.
* Original title: The name of the film before it was translated or adapted.
* Overview: A succinct synopsis of the film.
* popularity is a numerical indicator of a movie's level of
* popularity.production\_companies - The film's production company.
* The nation in which it was produced is indicated by the field production\_countries.
* release\_date: The day it was made available.
* Revenue is the amount of money the movie made globally.
* Runtime is the number of minutes the film lasts.
* status: "Released" or "Rumoured".
* tagline - The tagline of the film.
* title - The film's title.
* vote\_average shows the film's typical ratings.
* vote\_count is the total number of votes cast.
  + The third dataset (ratings\_small.csv)
    - Contains 100 ratings from 700 users on 9000 movies. It is a subset of the ratings available in the The Movie dataset.
    - <https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset>
  1. **Preprocessing of the data**

The “genres” and “keywords” column of the dataset consist of a list of items which will be noisy for the model later. The keyword are extracted to become only string of words. The “cast” column is also processed to contain only the top three main character while only director name is extracted from the “crew” column.

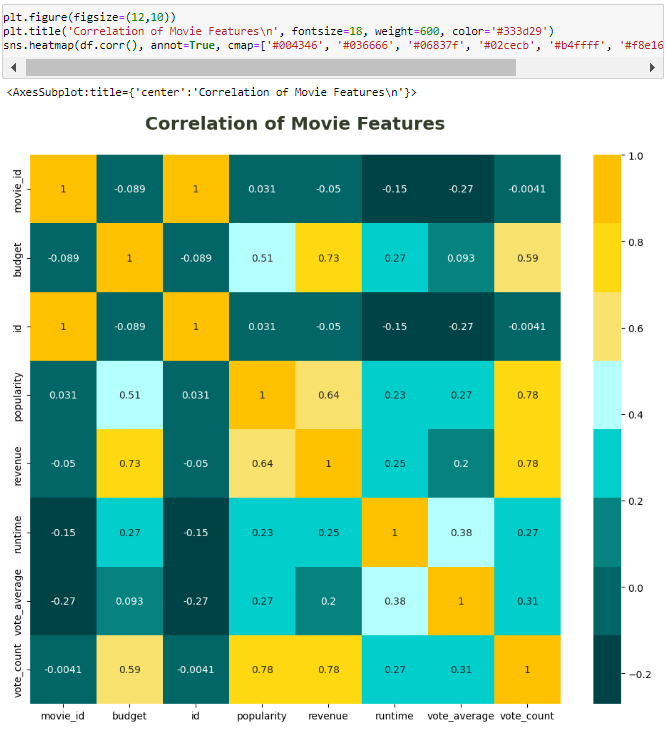
* 1. **Exploratory Data Analysis**

From the graph, we can see that budget and revenue just slightly influence the popularity of the movies.

The movies that are on top of the yellow line indicates that these are the movies that made a profit.

****

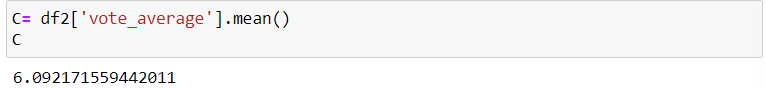
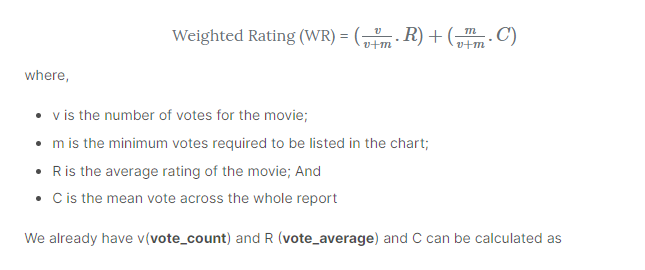
The insights from the relationship between rating and popularity is that movies that either got rating 0 and 10 are because of small number of voters and as the vote increases, the rating is most likely around 5 to 8.5. It also indicates that the more vote count, the more popular the movie is.

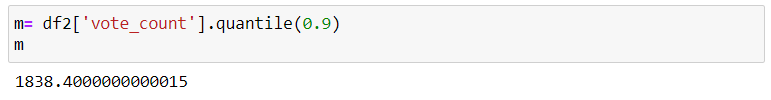
****

The correlation heatmap indicates that votes count, budget and popularity are the 3 dominant features as they showed high correlation with revenue.

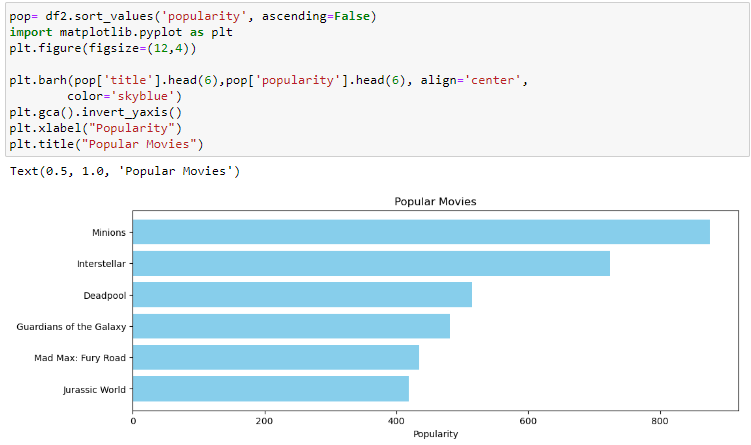
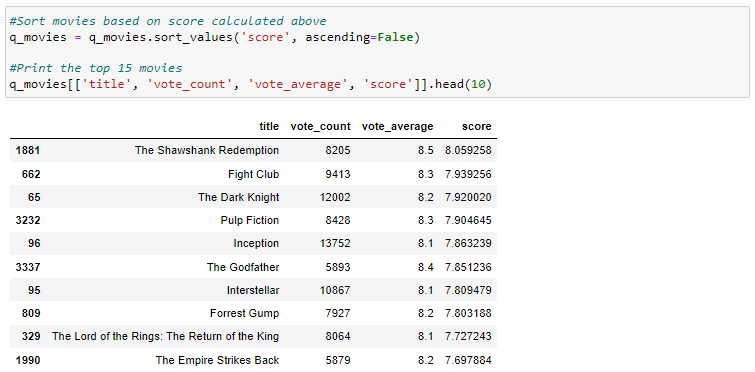
* 1. **Recommender System**
     1. **Demographic Filtering**

Before we begin, we need a metric to score or rate movies. We then calculate the scores for each movie, sort the scores, and then recommend the highest rated movie to the consumers.



On a scale of 1 to 10, the average rating for all the films is approximately 6.The next step is to choose a suitable value for m, the minimum number of votes necessary to appear in the chart. Our cut-off will be the 90th percentile.

Let's sort the DataFrame according to the score feature and output the names, total votes cast, the average number of votes cast, and the weighted rating or score of the top 10. films.

  
It is important to note that demographic recommenders offer broad recommendations of movies for all users and do not cater to the specific preferences and interests of an individual user. Therefore, for a more tailored approach, content-based filtering is a more advanced system to consider.

* + 1. **Content Based Filtering**

Filtering based on Content The movie's content (summary, cast, crew, keyword, tagline, etc.) is used in this recommender system to determine whether it is similar to other films. The most likely films to be comparable are then suggested.

**Plot Description Based Filtering**

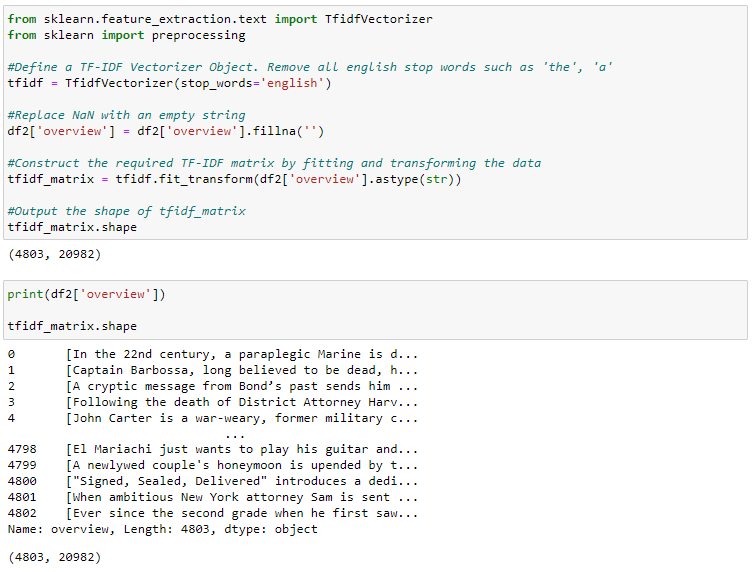
Using the plot summaries of all the movie pairs, our recommendation engine will determine the degree of similarity between each pair of films and base its suggestions on that degree. The overview feature of our dataset served as the source for the plot descriptions.

As part of the text-processing procedure, we must transform the word vector of each overview, therefore we will generate Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview.

The term "term frequency" describes how frequently a word appears in a certain document in relation to all the other terms there. It is determined by dividing the term's frequency in the text by the total number of words in the text.

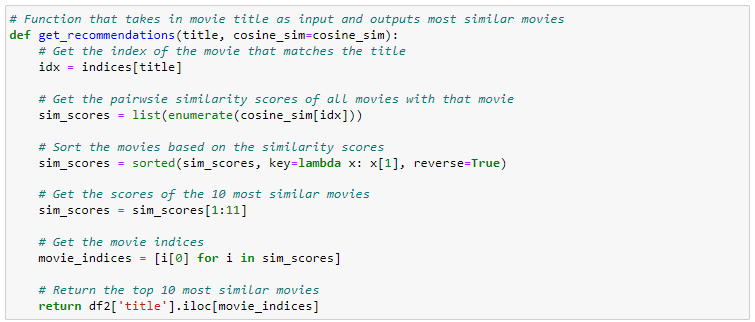
The relative count of documents that contain the term, on the other hand, is known as inverse document frequency (IDF), which is determined by dividing the total number of documents by the proportion of documents that contain the phrase.

TF \* IDF, where TF is the term frequency and IDF is the inverse document frequency, is the formula used to calculate a word's overall significance to the documents in which it appears.

Each row represents a movie, and each column represents a word from the lexicon of all the words found in at least one document. This is done to lessen the weight given to words that recur frequently in plot summaries, lessening their influence on the computation of the final similarity score.

In summary, the code calculates TF-IDF values for the 'overview' text data in **df2** and generates a matrix representation that can be used for further analysis or modeling tasks.

For this project, we will use the cosine similarity score to determine the similarity between two movies. The reason being that it is a magnitude-independent metric and is computationally efficient. In mathematical terms, cosine similarity is defined as follows:



The function takes a movie title and a cosine similarity matrix as inputs. It retrieves the index of the movie that matches the provided title in the similarity matrix. The function calculates the similarity scores between the selected movie and all other movies in the matrix.

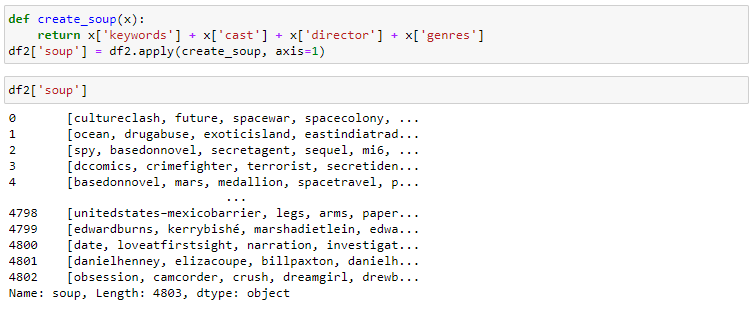
The similarity scores are sorted in descending order, ranking the most similar movies at the top. The function selects the top 10 most similar movies by excluding the first entry (which represents the provided movie itself) and extracting the movie indices. Finally, it returns the titles of the top 10 most similar movies from the dataset.

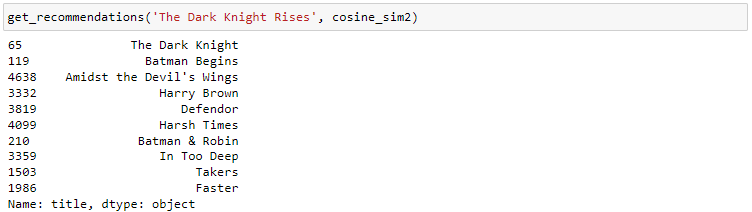
Since we have already used the TF-IDF vectorizer, the cosine similarity score can be obtained by calculating the dot product. Thus, we will use the linear\_kernel() function from the sklearn library as it is faster than cosine\_similarities().

* + 1. **Credits, Genres and Keywords Based Filtering**

We will design a recommender using the following metadata: the director, the associated genres, the top 3 actors, and the storyline keywords from the film.

We must identify the three essential actors, the director, and the keywords related to that movie from the cast, crew, and cast features. We need to transform our data from its current state as "stringified" lists into a structure that is secure and practical.



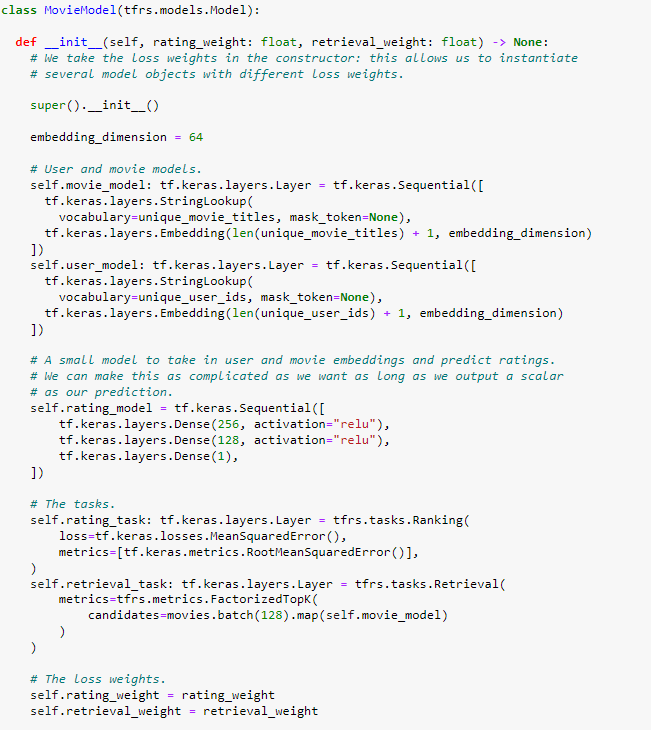
The next step is to replace TF-IDF with CountVectorizer(). This is so that if an actor or director appears in multiple films, their presence won't be downplayed. The updated cosine\_sim2 matrix can now be passed as your second input to our get\_recommendation() function so that we can reuse it.

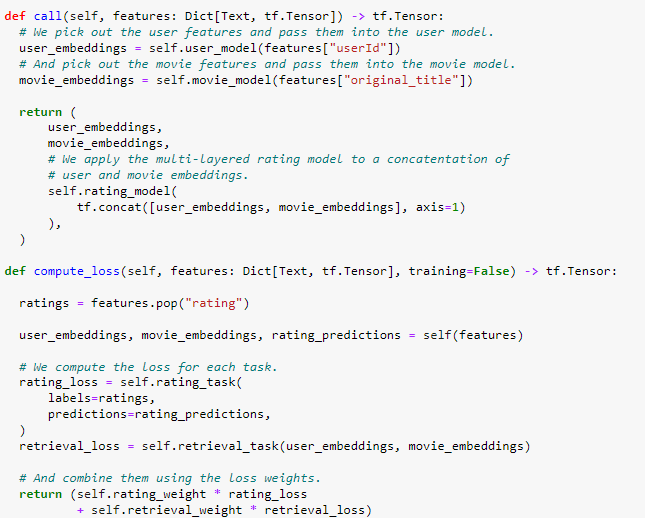
* 1. **Deep Learning**

The usage of deep learning for a hybrid recommender system is then investigated.

TensorFlow Recommenders (TFRS) makes it possible to build recommender systems using a multi-objective methodology. Using past data, the method enables the model to forecast which movies the user should watch. As Keras has a low learning curve and the ability to build complicated models, it is the foundation around which TFRS is based.

The code converts the 'userId' column to string data type, creates TensorFlow datasets for the ratings and movies data, and performs mapping operations to transform and extract the required columns from the datasets.



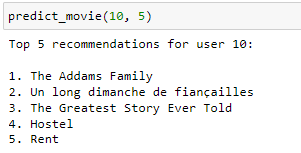


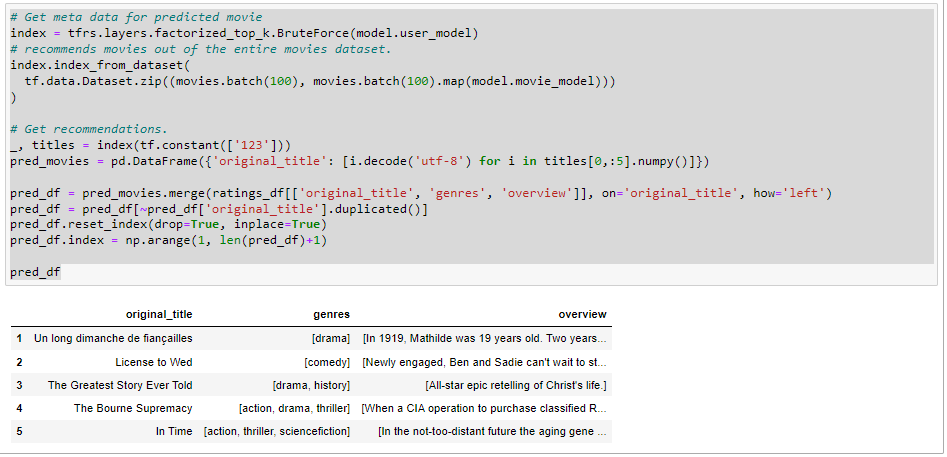
The code defines a movie recommendation model using TensorFlow Recommenders (TFRS). The model has two main components: the movie model and the user model.

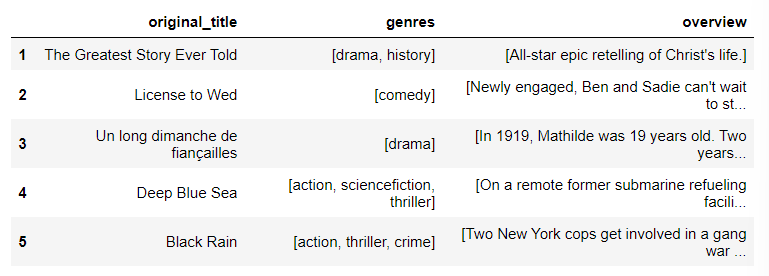
The movie model and user model convert categorical input (movie titles and user IDs) into dense vectors using embedding layers. These vectors represent the characteristics of movies and users.

The rating model is a small neural network that takes concatenated user and movie embeddings as input and predicts ratings. It consists of three dense layers with ReLU activation functions.

Overall, the model learns from user and movie features to make movie recommendations by predicting ratings and optimizing for both rating accuracy and item retrieval quality.







The code utilizes the **index** to generate movie recommendations for a specific user. It then merges the recommended movies with the original dataset to retrieve additional metadata, resulting in a DataFrame (**pred\_df**) that includes the movie titles, genres, and overviews of the predicted movies.

The genres "drama" appeared the most in the history of user "123" and it is not a suprise that two of the recommended movies are "drama". While no "animation" is recommended by the algorithm.

* 1. **Conclusion**

The TensorFlow Recommenders performs well in the sense that it can go through all the data available and would performs better if the dataset become larger. The results shown by the recommendation are impressive as it recommended movies that are very similar in genre and storyline and uses the available information well.

On the other hand, Credits, Genre and Keyword Based recommender and Plot Description Based Recommender works reasonably well where their recommendation is pretty on point. The only negative point I can get from both recommender system is that they don’t use the other available information efficiently and will not be able to scale if the dataset become large.