In [1]:
```
# Author: Surena Nokham
# Week 10: Recommender Systems
# Date: 2/18/2023
```

# Week 10 - Recommender Systems

Using the small MovieLens data set, create a recommender system that allows users to input a movie they like (in the data set) and recommends ten other movies for them to watch. In your write-up, clearly explain the recommender system process and all steps performed. If you are using a method found online, be sure to reference the source. You can use R or Python to complete this assignment. Submit your code and output to the submission link. Make sure to add comments to all of your code and to document your steps, process, and analysis.

**Recommender System Process: TF-IDF-based Movie Recommendations**

In [2]:
```
# Load necessary libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

- Load necessary libraries: The initial step involves importing essential libraries, including pandas for data manipulation and TfidfVectorizer along with linear_kernel from sklearn for implementing the TF-IDF vectorization and cosine similarity calculations.

In [3]:
```
# Load the dataset
movies_df = pd.read_csv('movies.csv')
```

- Load the dataset: The movie dataset is loaded into a Pandas DataFrame using the pd.read_csv function.

In [4]:
```
# Initialize TF-IDF Vectorizer
tfidf = TfidfVectorizer(stop_words='english')
```

- Initialize TF-IDF Vectorizer: A TF-IDF Vectorizer is set up with English stop words to convert the textual movie genres into a TF-IDF matrix representation.

In [5]:
```
# Replace NaN values with empty strings
movies_df['genres'] = movies_df['genres'].fillna('')
```

- Replace NaN values with empty strings: Any missing values in the 'genres' column of the dataset are replaced with empty strings to ensure seamless processing.

In [6]:
```
# Build the TF-IDF matrix
tfidf_matrix = tfidf.fit_transform(movies_df['genres'])
```

- Build the TF-IDF matrix: The TF-IDF matrix is constructed by fitting and transforming the movie genres using the initialized TF-IDF Vectorizer.

In [7]:
```
# Calculate the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

- Calculate the cosine similarity matrix: Using the linear kernel function, the cosine similarity matrix is computed based on the TF-IDF matrix.

In [8]:
```
# Create movie recommendation function
def recommend_movies(movie_title, cosine_sim=cosine_sim):
    # Get index of movie that matches the title
    idx = movies_df.loc[movies_df['title'].str.lower() == movie_title.

    # Compute pairwise movie similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort movies based on scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Retrieve scores for the top 10 similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Provide the top 10 similar movies
    return movies_df['title'].iloc[movie_indices]
```

- **Create a movie recommendation function: A function named recommend_movies is defined to take a movie title as input and return the top 10 recommended movies based on cosine similarity scores.**

  1. Input Matching: The function takes two parameters: movie_title (the input movie for which recommendations are sought) and cosine_sim (the precomputed cosine similarity matrix).
  2. Movie Index Retrieval: It starts by finding the index of the movie in the dataset that matches the input title. The .str.lower() ensures case-insensitive matching, and .index[0] retrieves the index of the first matching movie.
  3. Pairwise Similarity Computation: It then computes the pairwise movie similarity scores for the input movie by extracting the corresponding row from the cosine similarity matrix.
  4. Sorting by Scores: The movie indices and their similarity scores are stored as tuples in a list, and the list is sorted based on similarity scores in descending order.
  5. Top 10 Similar Movies Selection: The function then selects the top 10 similar movies by slicing the sorted list, excluding the input movie itself (hence starting from index 1).
  6. Index Extraction: It extracts the indices of the top 10 similar movies from the sorted list.
  7. Recommendation Output: Finally, it returns the titles of the top 10 similar movies based on the obtained indices.

In [9]:
```
# Test recommender system
input_movie = "Stardust (2007)"
recommended_movies = recommend_movies(input_movie)
print("Movies recommended for you based on", input_movie, ":")
recommended_movies
```

```
Movies recommended for you based on Stardust (2007) :
```

Out[9]:
```
24239                        Santos (2008)
51220           The Carmilla Movie (2017)
54623    Legend of the Naga Pearls (2017)
10830                         Click (2006)
2053                         Legend (1985)
3382                       Ladyhawke (1985)
10522                           She (1935)
13667      Secret of Moonacre, The (2008)
22602             Thief of Damascus (1952)
44163                 He's a Dragon (2015)
Name: title, dtype: object
```

- Test the recommender system: Finally, a test is conducted by providing a sample movie title ("Stardust (2007)") to the recommendation function. The results are printed to the console.
- This series of steps outlines the process of creating a movie recommender system using TF-IDF vectorization and cosine similarity calculations based on movie genres.