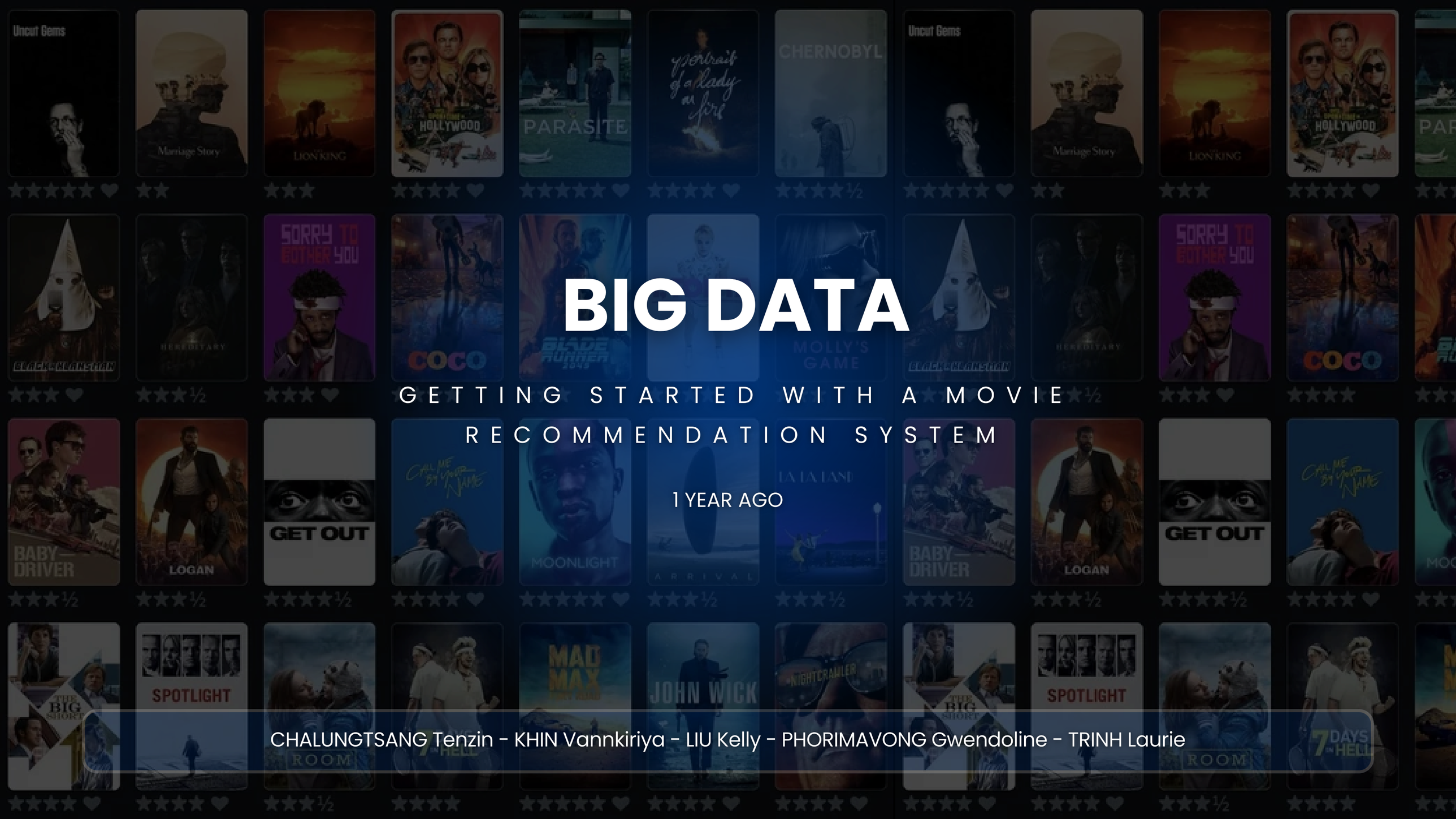


BIG DATA

GETTING STARTED WITH A MOVIE RECOMMENDATION SYSTEM

1 YEAR AGO

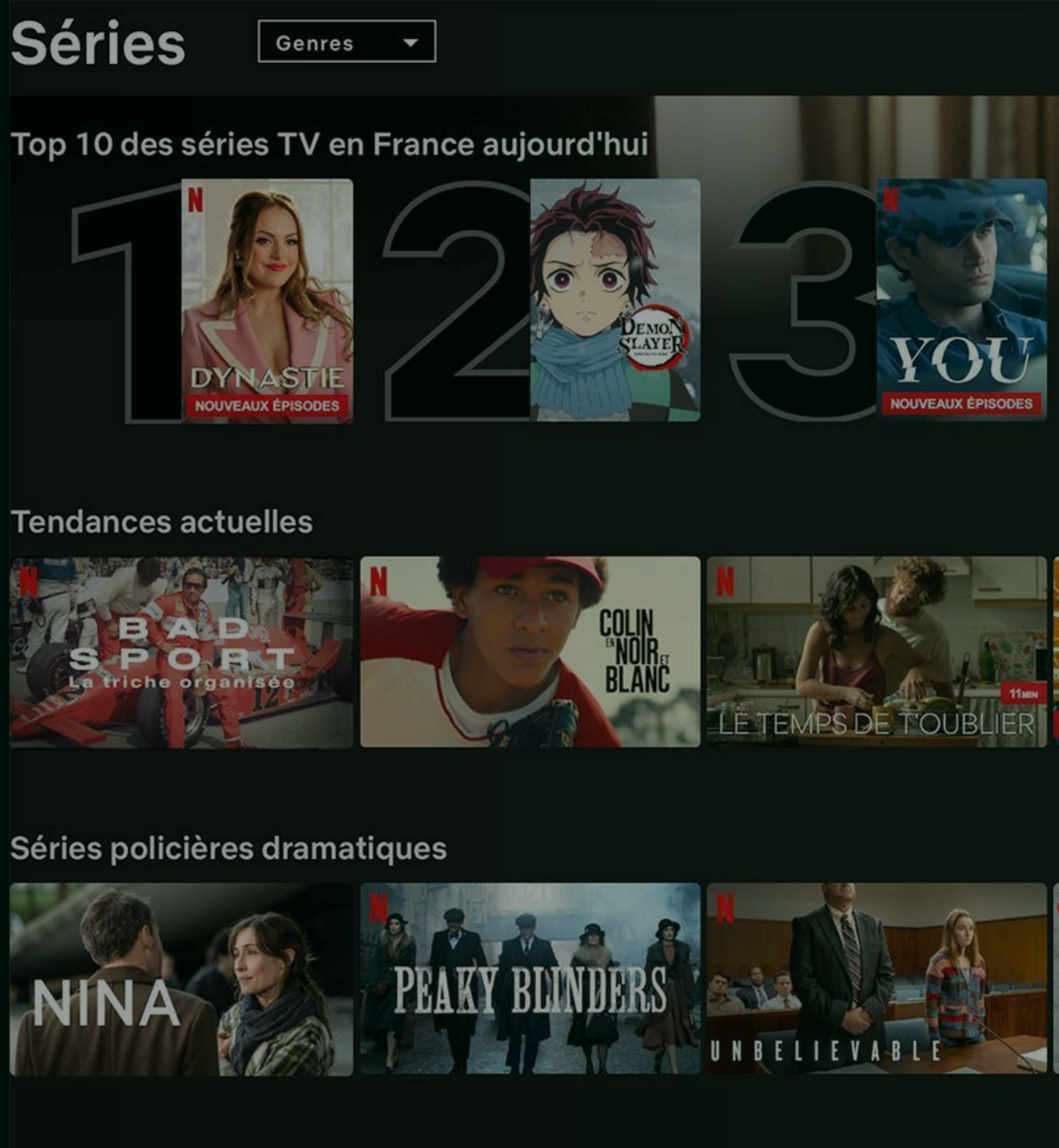
CHALUNGTSANG Tenzin - KHIN Vannkiriya - LIU Kelly - PHORIMAVONG Gwendoline - TRINH Laurie

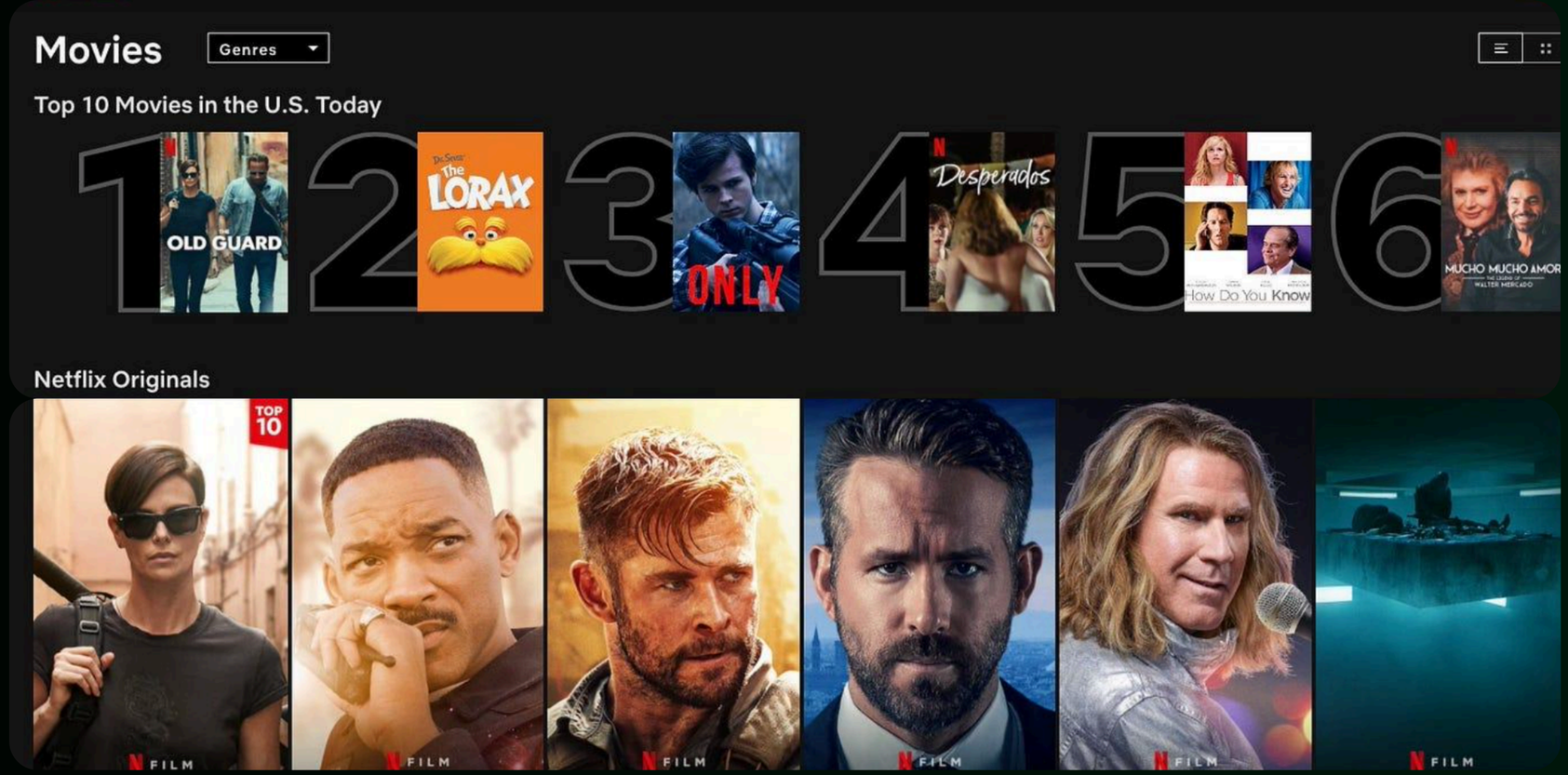
Introduction

The main goal is to suggest similar movies to users by analyzing movie metadata, such as genre, director, actors, and description. This study is based on the Kaggle notebook "Getting Started with a Movie Recommendation System."

To achieve this, we use a dataset movies released up to July 2017, containing detailed information such as cast, production team, plot keywords, budget, revenue, posters, and release dates.

Through this study, we aim to understand how recommendation systems use this information to generate relevant suggestions and apply commonly used techniques in this field.





The Original Notebook

The datasets used in the Kaggle notebook are files containing information about movies, users, and movie ratings. They help train the recommendation model by providing a database from which it can learn to make predictions.

Datasets Used:

TMDB 5000 Movies:

TMDB 5000 Movies: Contains movie details
(title, genre, duration, popularity).

This helps the model understand what types of movies users prefer.

TMDB 5000 Credits:

TMDB 5000 Credits: Includes actor and director information.

This helps connect people involved in the movie to its success and can influence recommendations.

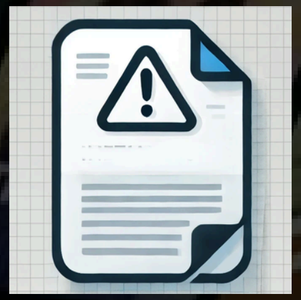
MovieLens Ratings (Small):

MovieLens Ratings (Small): Stores user ratings for movies.

This information is used to understand user preferences and improve movie recommendations.

These datasets help train the model by combining movie details, cast/crew data, and audience ratings to improve recommendations.

Problems Encountered on Google Colab



Initial Issue :

On Kaggle, everything is ready to use.

On Colab, datasets are not recognized automatically, requiring us to manually download and prepare the data from Kaggle.

Access to Files:.

On Kaggle datasets are preloaded and easy to use.

On Colab we used Kaggle API + kaggle.json to download files, then unzipped them.

Access to Surprise Library:

Scikit-surprise not pre-installed on Colab, so we had to install it manually.

File paths differ between platforms, so we adjusted code to handle both environments**.**

Changes in Surprise Library:

The method **evaluate()** was replaced by **cross_validate()** for model validation.

The function **split(n_folds=5)** was removed and replaced by **cross_validate()**.

We adapted the code for Colab by managing file access, installing dependencies, and updating outdated functions.

Models

The notebook contains two machine learning models

Content-Based Filtering Model

Collaborative Filtering Model

Content-Based Filtering Model

Recommends movies similar to those a user has already watched

The content-based filtering model works by recommending movies similar to those a user has already watched, based only on their descriptions. In our model, we analyze elements such as the title, synopsis, and genres of movies.

TF-IDF Vectorizer:

Converts movie descriptions into numerical vectors

Reduces the importance of frequent words, highlights specific words

Cosine Similarity:

Measures similarity between movies (smaller angle = more similar)

Uses linear_kernel for greater efficiency

Collaborative Filtering

Recommends movies based on user ratings
Uses a User-Movie Matrix: each user rates a set of movies
Identifies similar users to recommend new movies
Problem: Many missing ratings (users only rate a few movies)

SVD

SVD (Singular Value Decomposition) decomposes the matrix into smaller matrices

- Captures latent factors (e.g., genre preferences)
- Predicts ratings for unseen movies based on similar users' tastes

Model Validation

Uses cross-validation to test the model's accuracy
RMSE and MAE metrics evaluate predictions
Lower RMSE & MAE = better prediction accuracy

IV. A look at the notebook

Data loading and preparation

To use data from Kaggle:

1. Download it and open it in a format that we can work with using Pandas.

2. Install the Kaggle library, which helps us get files automatically. 3. Add a special file called **kaggle.json**, which acts as a key to connect to Kaggle. This file is stored in a hidden **.kaggle** folder, and we change its settings to keep it secure.

4. Download the dataset as a **ZIP file.** Since ZIP files are compressed, we need to

5. Extract them using **zipfile.ZipFile** before we can read the data.

6. Open the CSV files: **tmdb_5000_credits.csv** and **tmdb_5000_movies.csv**

7. Store these files in two Pandas DataFrames:
   - df1 for the credits file (actors and directors)
   - df2 for the movies file (titles, ratings, descriptions)

Finally, we check that everything is loaded correctly by displaying the first few rows of the data.

```
# 1 Installation de la bibliothèque Kaggle (si non installée)
!pip install kaggle

# 2 Téléchargement de la clé API de Kaggle
from google.colab import files
files.upload()  # L'utilisateur doit importer son fichier kaggle.json

# 3 Configuration de l'authentification
import os
os.makedirs('/root/.kaggle', exist_ok=True)  # Création du dossier .kaggle s'il n'existe pas
!mv kaggle.json /root/.kaggle/  # Déplacement du fichier au bon emplacement
os.chmod('/root/.kaggle/kaggle.json', 600)  # Sécurisation du fichier

# 4 Téléchargement des datasets depuis Kaggle
!kaggle datasets download -d tmdb/tmdb-movie-metadata  # Remplace le chemin de Kaggle

# 5 Décompression du fichier ZIP
import zipfile
with zipfile.ZipFile('tmdb-movie-metadata.zip', 'r') as zip_ref:
    zip_ref.extractall('.')  # Extraction dans le répertoire actuel

# 6 Chargement des fichiers dans Pandas
import pandas as pd
import numpy as np

df1 = pd.read_csv('tmdb_5000_credits.csv')  # Chargement des crédits
df2 = pd.read_csv('tmdb_5000_movies.csv')   # Chargement des films

# Vérification du chargement
print(df1.head())
print(df2.head())
```

```
Downloading tmdb-movie-metadata.zip to /content
  0% 0.00/8.89M [00:00<?, ?B/s]
100% 8.89M/8.89M [00:00<00:00, 108MB/s]
```

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

	budget	genres	homepage	id
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529

To analyze movies along with their actors and directors, we need to combine the two datasets. We do this by merging the two Pandas DataFrames using the id column, which is present in both files and helps match the correct movie information.
The merge() function links the movie details from df2 (titles, ratings, descriptions) with the credits from df1 (actors, directors). After merging, each row in the new DataFrame contains all the information about a movie in one place, making it easier to analyze.

Finally, we display a preview of the merged DataFrame to make sure everything is correct before moving on to the next steps. This helps confirm that the data is well-organized and ready for analysis or creating a model. By merging and preparing the data, we now have a complete dataset with all the important details, like movie titles, ratings, actors, and directors, which we can use in the following stages of the project.

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	...	runtime	spoken_languages	status	tagline	title	vote_average	vote_count	tittle	cast	crew
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Ingenious Film Partners", "id": 289...	...	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released	Enter the World of Pandora.	Avatar	7.2	11800	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt Disney Pictures", "id": 2}, {"...	...	169.0	[{"iso_639_1": "en", "name": "English"}]	Released	At the end of the world, the adventure begins.	Pirates of the Caribbean: At World's End	6.9	4500	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Columbia Pictures", "id": 5}, {"nam...	...	148.0	[{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...	Released	A Plan No One Escapes	Spectre	6.3	4466	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950	[{"name": "Legendary Pictures", "id": 923}, {"...	...	165.0	[{"iso_639_1": "en", "name": "English"}]	Released	The Legend Ends	The Dark Knight Rises	7.6	9106	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	[{"name": "Walt Disney Pictures", "id": 2}]	...	132.0	[{"iso_639_1": "en", "name": "English"}]	Released	Lost in our world, found in another.	John Carter	6.1	2124	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

5 rows × 23 columns

IV. A look at the notebook

Processing Popularity Data

Weighted_rating calculates an adjusted score for each movie.
It takes in a movie (x) and two numbers, **m** and **C**, which represent the minimum number of votes a movie must have to be considered, and the average rating of all movies, respectively.

Inside the function
v : number of votes for a movie.
R : movie's average rating.

The formula is based on **IMDb's** ranking system, where the movie's rating **R** is combined with the overall average **C,** but the number of votes **v** is also considered. If a movie has a lot of votes, its rating matters more. If it has fewer votes, the overall average rating **C** plays a bigger role in adjusting the score.

Once the function is defined, we apply it to the dataset using:
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
This creates a new column called score, which contains the weighted rating of each movie.

Next, we sort the movies by this score:
q_movies = q_movies.sort_values('score', ascending=False)
This provides a ranked list of movies from the most to the least popular.

Finally, we display the top 10 movies based on popularity:
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10).
This allows us to see the highest-rated movies based on their weighted popularity score.

This method helps us recommend movies based on their actual popularity, considering both ratings and the number of votes.

```
[ ]  def weighted_rating(x, m=m, C=C):
         v = x['vote_count']
         R = x['vote_average']
         # Calculation based on the IMDB formula
         return (v/(v+m) * R) + (m/(m+v) * C)
```

```
[ ]  # Define a new feature 'score' and calculate its value with `weighted_rating()`
     q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

```
     #Sort movies based on score calculated above
     q_movies = q_movies.sort_values('score', ascending=False)

     #Print the top 15 movies
     q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)
```

	title	vote_count	vote_average	score
1881	The Shawshank Redemption	8205	8.5	8.059258
662	Fight Club	9413	8.3	7.939256
65	The Dark Knight	12002	8.2	7.920020
3232	Pulp Fiction	8428	8.3	7.904645
96	Inception	13752	8.1	7.863239
3337	The Godfather	5893	8.4	7.851236
95	Interstellar	10867	8.1	7.809479
809	Forrest Gump	7927	8.2	7.803188
329	The Lord of the Rings: The Return of the King	8064	8.1	7.727243
1990	The Empire Strikes Back	5879	8.2	7.697884

Content-Based Recommendation (Similarity)

Recommends movies similar to those a user has already watched

Based on movie descriptions:

Title, synopsis, genres

TF-IDF Vectorizer:

Converts movie descriptions into numerical vectors

Reduces the importance of frequent words, highlights specific words

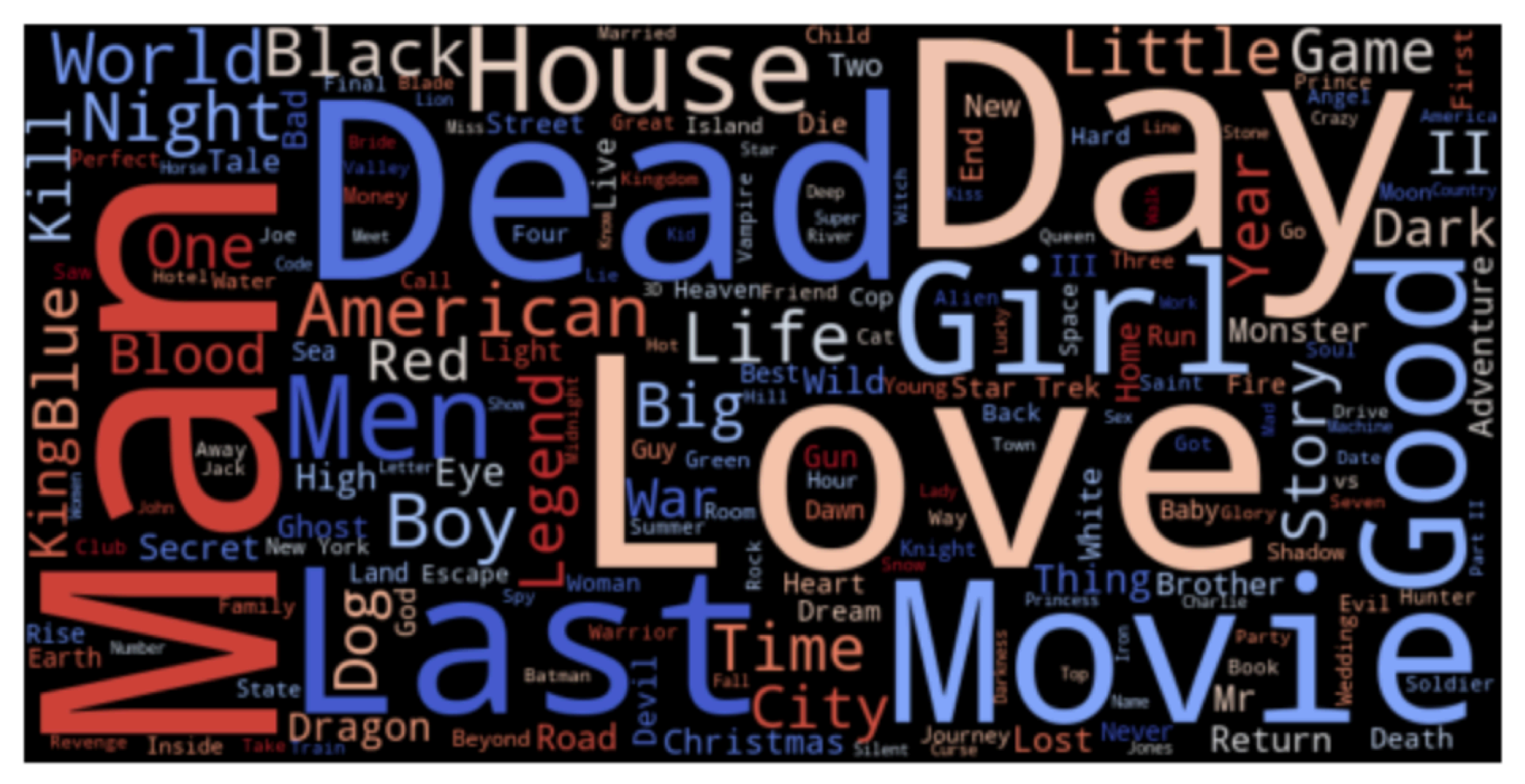
Cosine Similarity:

Measures similarity between movies (smaller angle = more similar)

Uses linear_kernel for greater efficiency

```
[ ]  #Import TfIdfVectorizer from scikit-learn
     from sklearn.feature_extraction.text import TfidfVectorizer

     #Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
     tfidf = TfidfVectorizer(stop_words='english')

     #Replace NaN with an empty string
     df2['overview'] = df2['overview'].fillna('')

     #Construct the required TF-IDF matrix by fitting and transforming the data
     tfidf_matrix = tfidf.fit_transform(df2['overview'])

     #Output the shape of tfidf_matrix
     tfidf_matrix.shape

     (4803, 20978)
```

WordCloud


Man
Dead
Day
Love
Movie
Last
Girl
House
Good
Black
World
Night
Little
Game
American
Men
Life
Big
Boy
Time
City
Blood
Red
One
Kill
Blue
King
Legend
Story
Year
Dark
II
Dragon
Christmas
Return
Death
Lost
Road
Secret
Ghost
War
Thing
Brother
Monster
Adventure
White
High
Eye
Star Trek
Heart
Dream
Dog

Models

Similarity matrix for recommendations

```
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
#Construct a reverse map of indices and movie titles
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```

```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

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

# Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

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

    # Return the top 10 most similar movies
    return df2['title'].iloc[movie_indices]
```

Once the summaries are converted into vectors, and we've done the wordcloud, we calculate the cosine similarity between them.

This measure evaluates how close two movies are by comparing their descriptions as numerical vectors.

The higher the value, the more similar the movies.

```
get_recommendations('The Dark Knight Rises')
```

	title
65	The Dark Knight
299	Batman Forever
428	Batman Returns
1359	Batman
3854	Batman: The Dark Knight Returns, Part 2
119	Batman Begins
2507	Slow Burn
9	Batman v Superman: Dawn of Justice
1181	JFK
210	Batman & Robin

dtype: object

Recommend movies similar to a given movie.

Retrieve the most similar movies based on their similarity score and display the closest matches to the selected film.

Thus, if a user likes a movie, this model will suggest other movies with similar themes, stories, or genres.

Enhanced Recommendations with Actors, Genres, and Directors

DATA CLEANING

Standardizes all names by converting them to lowercase and removing unnecessary spaces

Columns: actors, keywords, directors, and genres

All this information is combined into a single column called "soup."

CountVectorizer to convert this text into a numerical matrix

From this matrix, we compute the cosine similarity between movies, allowing us to measure their similarity based on combined characteristics (actors, director, genre, keywords). The higher the similarity, the more alike the movies are.

```python
# Function to convert all strings to lower case and strip names of spaces
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

```python
# Apply clean_data function to your features.
features = ['cast', 'keywords', 'director', 'genres']

for feature in features:
    df2[feature] = df2[feature].apply(clean_data)
```

```python
def create_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])
df2['soup'] = df2.apply(create_soup, axis=1)
```

```python
# Import CountVectorizer and create the count matrix
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df2['soup'])
```

```python
# Compute the Cosine Similarity matrix based on the count_matrix
from sklearn.metrics.pairwise import cosine_similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

```python
# Reset index of our main DataFrame and construct reverse mapping as before
df2 = df2.reset_index()
indices = pd.Series(df2.index, index=df2['title'])
```

Finally, we use this similarity matrix to recommend movies most similar to a user's preferred film—for instance, suggesting films with shared actors, genres, or directors—resulting in more personalized recommendations that consider a broader range of factors.

```python
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

	title
65	The Dark Knight
119	Batman Begins
4638	Amidst the Devil's Wings
1196	The Prestige
3073	Romeo Is Bleeding
3326	Black November
1503	Takers
1986	Faster
303	Catwoman
747	Gangster Squad

dtype: object

Collaborative Filtering Model

- Uses user ratings to predict preferences
- User-movie matrix:
  - Rows = users, Columns = movies, Cells = ratings
- Challenges: Sparse matrix (most ratings are missing)

The ratings_small.csv file contains user ratings for various movies. We apply Singular Value Decomposition (SVD) to train a model that can predict missing ratings.
The process begins by loading the data using pandas and creating a Dataset with Surprise, a specialized library for recommendation systems. We then use an SVD object to build the model, and the cross_validate method to evaluate its performance using RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error). These metrics indicate the model's prediction accuracy on test data.

CROSS-VALIDATION

- Validates the model's accuracy using cross_validate()
- Splits the data into subsets to test the model
- Metrics: RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error)
- Goal: Check the model's ability to predict ratings for unseen movies

```python
from surprise import SVD
from surprise import Dataset
from surprise import Reader
from surprise.model_selection import cross_validate
from surprise import accuracy # Import the accuracy module


# Load the ratings data
import pandas as pd

reader = Reader()
ratings = pd.read_csv('ratings_small.csv')  # Updated path to the extracted file
# Create the dataset
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
svd = SVD()

# Evaluate the model using cross-validation and print RMSE and MAE
# Perform cross-validation (no need to call data.split)
# For example, to evaluate an SVD algorithm:
results = cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Print the average RMSE and MAE across folds
print(f"Average RMSE: {results['test_rmse'].mean()}")
print(f"Average MAE: {results['test_mae'].mean()}")
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8903  0.9007  0.8981  0.8964  0.8979  0.8967  0.0035
MAE (testset)     0.6846  0.6948  0.6932  0.6888  0.6928  0.6908  0.0037
Fit time          1.56    1.17    0.97    0.95    0.98    1.13    0.23
Test time         0.34    0.07    0.20    0.07    0.07    0.15    0.10
Average RMSE: 0.896672844867779
Average MAE: 0.6908373208183228
```

```python
trainset = data.build_full_trainset()
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f5d7f6c5710>
```

```python
ratings[ratings['userId'] == 1]
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205
5	1	1263	2.0	1260759151
6	1	1287	2.0	1260759187
7	1	1293	2.0	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2.0	1260759131
10	1	1371	2.5	1260759135
11	1	1405	1.0	1260759203
12	1	1953	4.0	1260759191

After performing cross-validation, the model is trained on the entire dataset using fit(), allowing us to make predictions.

User 1 would give to movie the rate of 302.

This process enables us to predict movies a user might enjoy based on the past ratings of other users. Using matrix decomposition via SVD, the model identifies movies with similar rating patterns.

CONCLUSION

	Recommendation methods		
	Popularity-based recommendations	**Content-based recommendations**	**Collaborative filtering (SVD)**
Explanation	Rely on global audience votes.	Identify movies with similar descriptions	Personalize recommendations based on user preferences.
Advantages	Simple and fast but lacks personalization	Useful for users looking for similar movies	The most advanced, as it learns individual user preferences

By combining these approaches, we can create a hybrid recommendation system similar to those used by Netflix or Amazon Prime, improving recommendation accuracy and user experience.

And finally, this project helps us to understand each specific term and how to run a program.

THANK YOU

FOR YOUR ATTENTION