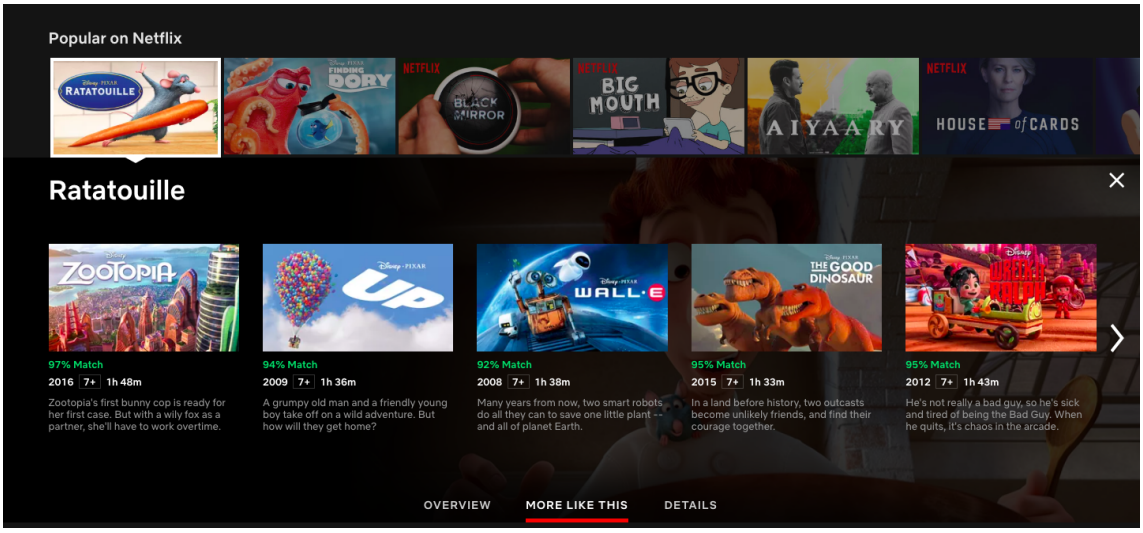
# Lab 7: Hybrid Recommenders

In this final lab, we will discuss recommender systems in the context of practicality and industrial use. Until now, we have learned about various types of recommender, including knowledge, content, and collaborative filtering-based engines. However, when used in practice, each recommender usually suffers from one shortcoming or another.

# Introduction

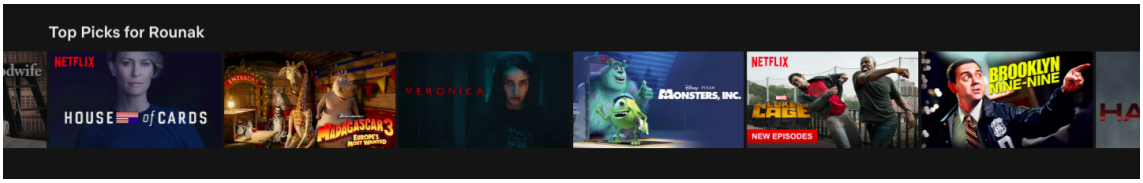
As already mentioned a couple of times, hybrid recommenders are extremely powerful, robust systems that combine various simpler models to give us predictions. There is no single way in which a hybrid model could do this; some hybrids predict using content and collaborative filtering techniques separately to produce results. Some others introduce content-based techniques into collaborative filters and vice versa.

Netflix is a very good example of a hybrid recommender. Netflix employs content-based techniques when it shows you similar movies to a movie you're watching (the [MORE LIKE THIS] section), as shown in the following screenshot:




Here, we can see that while watching *Ratatouille,* Netflix recommends movies to me that are very similar to *Ratatouille*. All the top five recommended movies are all animated and produced by Disney Pixar.

However, animated movies are not the only genre I watch on Netflix. I also like watching drama and comedy. Netflix has a separate row of recommendations for me entitled [Top Picks for Rounak], where it uses collaborative filtering to identify users similar to me and recommend movies that they have liked, but that I haven't watched:




In this way, Netflix employs both content- and collaborative-based techniques separately to produce results that are extremely satisfactory.

# Case study -- Building a hybrid model

In this section, let's build a content-based model that incorporates some collaborative filtering techniques into it.

Imagine that you have built a website like Netflix. Every time a user watches a movie, you want to display a list of recommendations in the side pane (like YouTube). At first glance, a content-based recommender seems appropriate for this task. This is because, if the person is currently watching something they find interesting, they will be more inclined to watch something similar to it.

Let's say our user is watching *The Dark Knight*. Since this is a Batman movie, our content-based recommender is likely to recommend other Batman (or superhero) movies regardless of quality. This may not always lead to the best recommendations. For instance, most people who like *The Dark Knight* do not rate *Batman and Robin* very highly, although they feature the same lead character. Therefore, we will introduce a collaborative filter here that predicts the ratings of the movies recommended by our content-based model and return the top few movies with the highest predictions.

In other words, the workflow of our hybrid model will be as follows:

1. Take in a movie title and user as input
2. Use a content-based model to compute the 25 most similar movies
3. Compute the predicted ratings that the user might give these 25 movies using a collaborative filter
4. Return the top 10 movies with the highest predicted rating

We will be using different datasets for this task. Go ahead and download the datasets from the following links.

**NOTE:** All datasets used in the lab are available in lab environment and GitHub repository:

`https://github.com/fenago/recommendation-systems-python/tree/main/data`

With these files in hand, let's proceed to build our model. The first step is to compute the [cosine_sim] matrix for our movies. In addition, we also need to map every movie to the indices in the [cosine_sim] matrix. We've already learned how to do this in Lab 3, *Building an IMDB Top 250 Clone with Pandas*. Computing this matrix and the mapping, therefore, is left as an exercise for the reader.

Next, let's build a collaborative filtering model. We will use the SVD model from the last lab for this purpose:

```
#Build the SVD based Collaborative filter
from surprise import SVD, Reader, Dataset

reader = Reader()
ratings = pd.read_csv('../data/ratings_small.csv')
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
data.split(n_folds=5)
svd = SVD()
trainset = data.build_full_trainset()
svd.train(trainset)
```

Next, let's load the [movie_ids.cs][v] file into a DataFrame and construct two mappings: one that returns the movie title for a given movie ID, and the other vice versa:

```
#Build title to ID and ID to title mappings
id_map = pd.read_csv('../data/movie_ids.csv')
id_to_title = id_map.set_index('id')
title_to_id = id_map.set_index('title')
```

Now, let's import the metadata for our movies so that our recommender can display useful information, such as the IMDB rating and the year of release. This information can be extracted from the main [movies_metadata.csv ]file, and is again left as an exercise for the reader.

We're finally in a position to build the hybrid recommender function according to the workflow described previously:

```
def hybrid(userId, title):
    #Extract the cosine_sim index of the movie
    idx = cosine_sim_map[cosine_sim_map == title].index[0]

    #Extract the TMDB ID of the movie
    tmdbId = title_to_id.loc[title]['id']

    #Extract the movie ID internally assigned by the dataset
    movie_id = title_to_id.loc[title]['movieId']

    #Extract the similarity scores and their corresponding index for every movie from
the cosine_sim matrix
    sim_scores = list(enumerate(cosine_sim[str(int(idx))]))

    #Sort the (index, score) tuples in decreasing order of similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    #Select the top 25 tuples, excluding the first
    #(as it is the similarity score of the movie with itself)
    sim_scores = sim_scores[1:26]

    #Store the cosine_sim indices of the top 25 movies in a list
    movie_indices = [i[0] for i in sim_scores]

    #Extract the metadata of the aforementioned movies
    movies = smd.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'year',
'id']]

    #Compute the predicted ratings using the SVD filter
    movies['est'] = movies['id'].apply(lambda x: svd.predict(userId,
id_to_title.loc[x]['movieId']).est)

    #Sort the movies in decreasing order of predicted rating
    movies = movies.sort_values('est', ascending=False)

    #Return the top 10 movies as recommendations
    return movies.head(10)
```

Let's put our hybrid model to the test. Let's imagine that users with the IDs 1 and 2 are both watching the movie *Avatar:*

```
hybrid(1, 'Avatar')
```

| | title | vote_count | vote_average | year | id | est |
|---|---|---|---|---|---|---|
| **1011** | The Terminator | 4208.0 | 7.4 | 1984 | 218 | 3.140748 |
| **974** | Aliens | 3282.0 | 7.7 | 1986 | 679 | 3.126947 |
| **8401** | Star Trek Into Darkness | 4479.0 | 7.4 | 2013 | 54138 | 3.079551 |
| **7705** | Alice in Wonderland | 8.0 | 5.4 | 1933 | 25694 | 3.054995 |
| **3060** | Sinbad and the Eye of the Tiger | 39.0 | 6.3 | 1977 | 11940 | 3.028386 |
| **8658** | X-Men: Days of Future Past | 6155.0 | 7.5 | 2014 | 127585 | 2.997411 |
| **2014** | Fantastic Planet | 140.0 | 7.6 | 1973 | 16306 | 2.957614 |
| **522** | Terminator 2: Judgment Day | 4274.0 | 7.7 | 1991 | 280 | 2.914548 |
| **1621** | Darby O'Gill and the Little People | 35.0 | 6.7 | 1959 | 18887 | 2.844940 |
| **1668** | Return from Witch Mountain | 38.0 | 5.6 | 1978 | 14822 | 2.804012 |

```
hybrid(2, 'Avatar')
```

| | title | vote_count | vote_average | year | id | est |
|---|---|---|---|---|---|---|
| **522** | Terminator 2: Judgment Day | 4274.0 | 7.7 | 1991 | 280 | 3.943639 |
| **2834** | Predator | 2129.0 | 7.3 | 1987 | 106 | 3.866272 |
| **8401** | Star Trek Into Darkness | 4479.0 | 7.4 | 2013 | 54138 | 3.858491 |
| **1011** | The Terminator | 4208.0 | 7.4 | 1984 | 218 | 3.856029 |
| **7705** | Alice in Wonderland | 8.0 | 5.4 | 1933 | 25694 | 3.701565 |
| **922** | The Abyss | 822.0 | 7.1 | 1989 | 2756 | 3.676465 |
| **974** | Aliens | 3282.0 | 7.7 | 1986 | 679 | 3.672303 |
| **1621** | Darby O'Gill and the Little People | 35.0 | 6.7 | 1959 | 18887 | 3.628234 |
| **1668** | Return from Witch Mountain | 38.0 | 5.6 | 1978 | 14822 | 3.614118 |
| **2014** | Fantastic Planet | 140.0 | 7.6 | 1973 | 16306 | 3.602051 |

We can see that although both users are currently watching *Avatar,* the recommendations differ in the content as well as the order. This is influenced by the collaborative filter. However, all the movies listed are similar to *Avatar.* This is because of the content-based filtering carried out by the model.

# Summary

With this, we come to the end of this lab, as well as the main part of the course. In this course, we learned the following:

1. We introduced recommender systems, defined the problem, and discussed different types of recommendation engines.
2. We learned data wrangling with pandas, focusing on Series and DataFrames.
3. We built an IMDB Top 250 clone and enhanced it into a knowledge-based recommender using genre, duration, and release year.
4. We explored content-based recommenders using plot lines and metadata, applying vectorizers and cosine similarity.
5. We covered data mining techniques, including clustering (k-means), PCA, supervised learning, and evaluation metrics.
6. We experimented with collaborative filtering models and used the *surprise* library to simplify implementation.
7. We built a hybrid recommender combining collaborative filtering and content-based approaches for personalized recommendations.