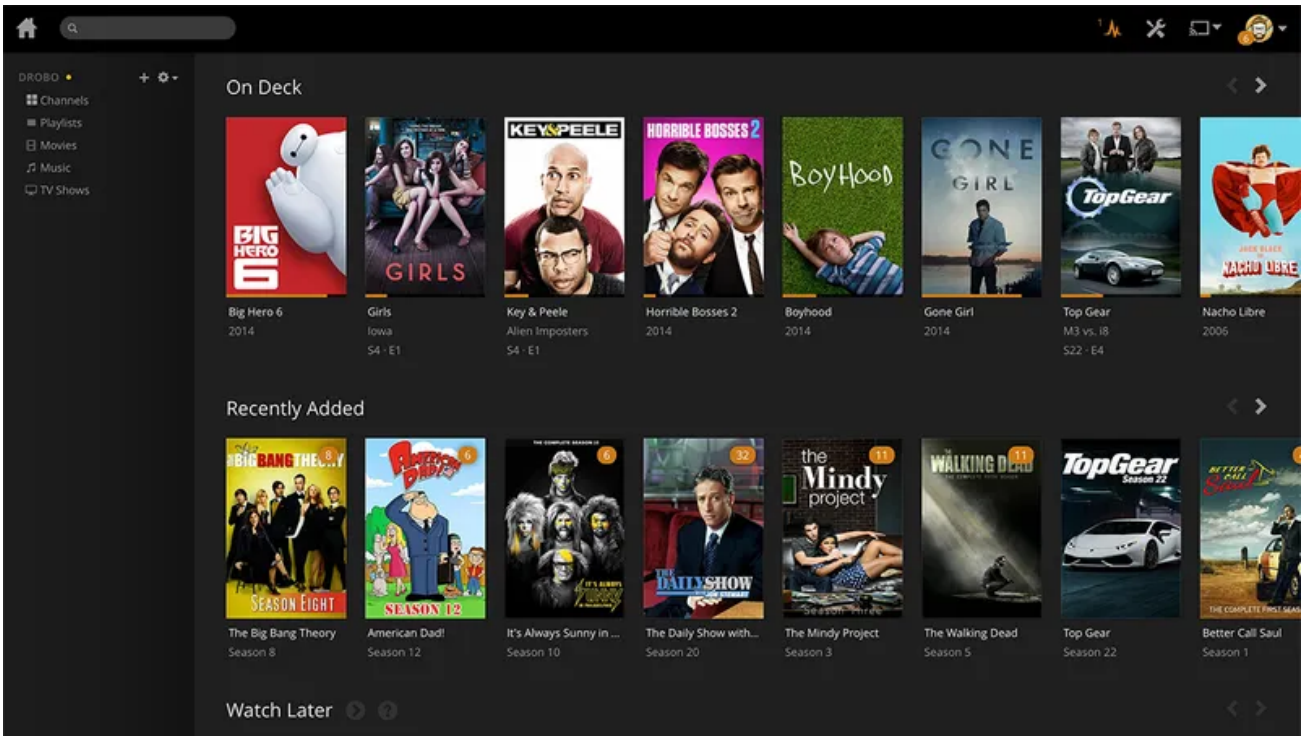
The Age of Recommender Systems

- The rapid growth of data collection has led to a new era of information.
- Data is being used to create more efficient systems and this is where Recommendation Systems come into play.
- Recommendation Systems are a type of **information filtering systems** as they improve the quality of search results and provides items that are more relevant to the search item or are realted to the search history of the user.

- They are used to predict the **rating** or **preference** that a user would give to an item.
- Almost every major tech company has applied them in some form or the other: Amazon uses it to suggest products to customers, YouTube uses it to decide which video to play next on autoplay, and Facebook uses it to recommend pages to like and people to follow.
- Moreover, companies like Netflix and Spotify depend highly on the effectiveness of their recommendation engines for their business and sucees.



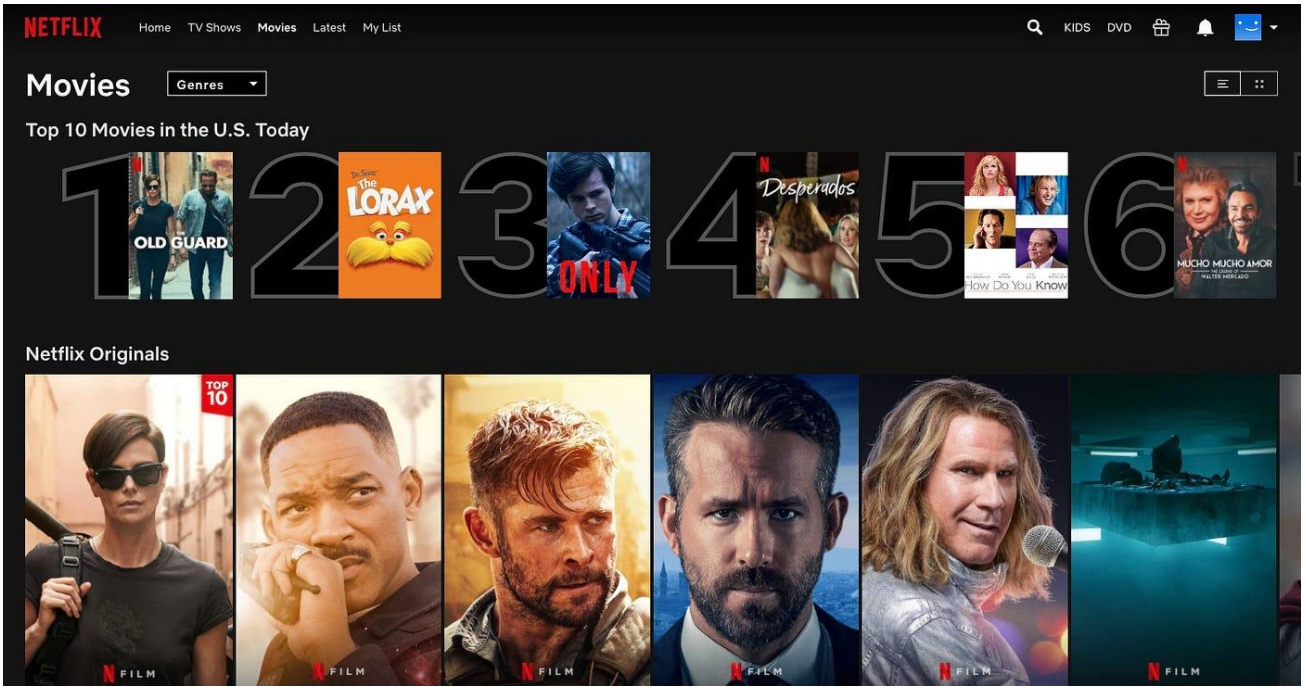
- In this kernel we'll be building a baseline Movie Recommendation System using TMDB 5000 Movie Dataset. For novices like me this kernel will pretty much serve as a foundation in recommendation systems and will provide you with something to start with.

So let's go!

There are basically three types of recommender systems:-

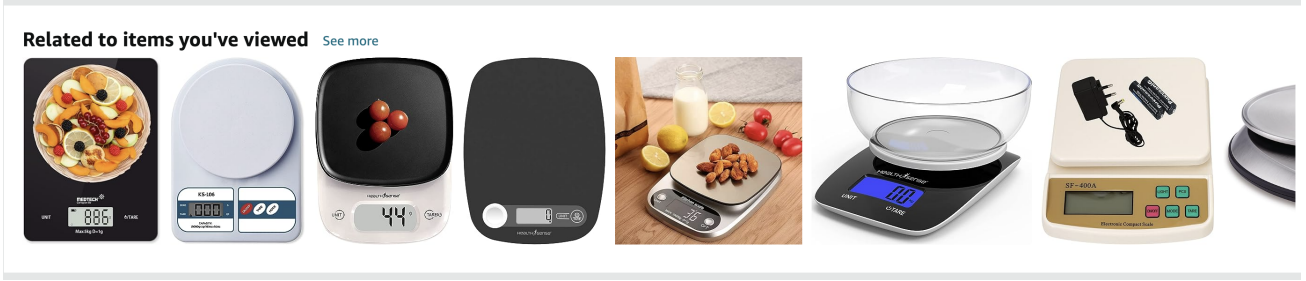
- **Demographic Filtering**-
  - They offer generalized recommendations to every user, based on movie popularity and/or genre.
  - The System recommends the same movies to users with similar demographic features.
  - Since each user is different, this approach is considered to be too simple.

- The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience.

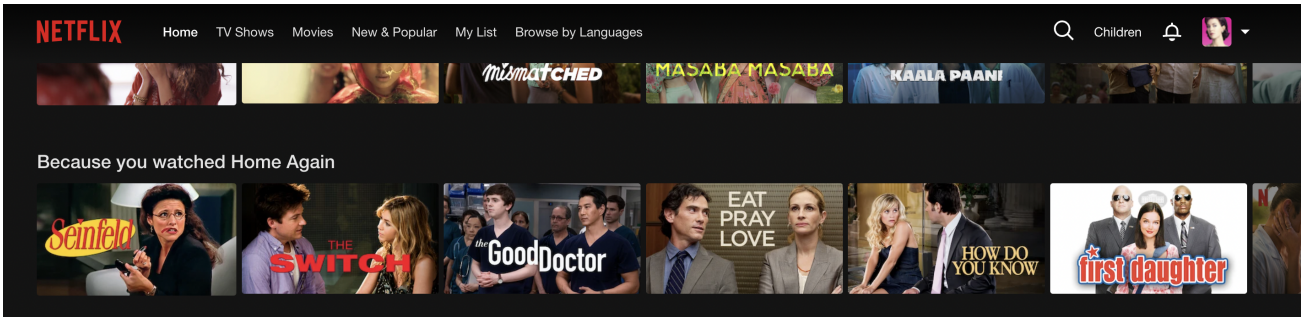


- **Content Based Filtering**-
  - They suggest similar items based on a particular item.
  - This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations.
  - The general idea behind these recommender systems is that if a person liked a particular item, he or she will also like an item that is similar to it.

Amazon



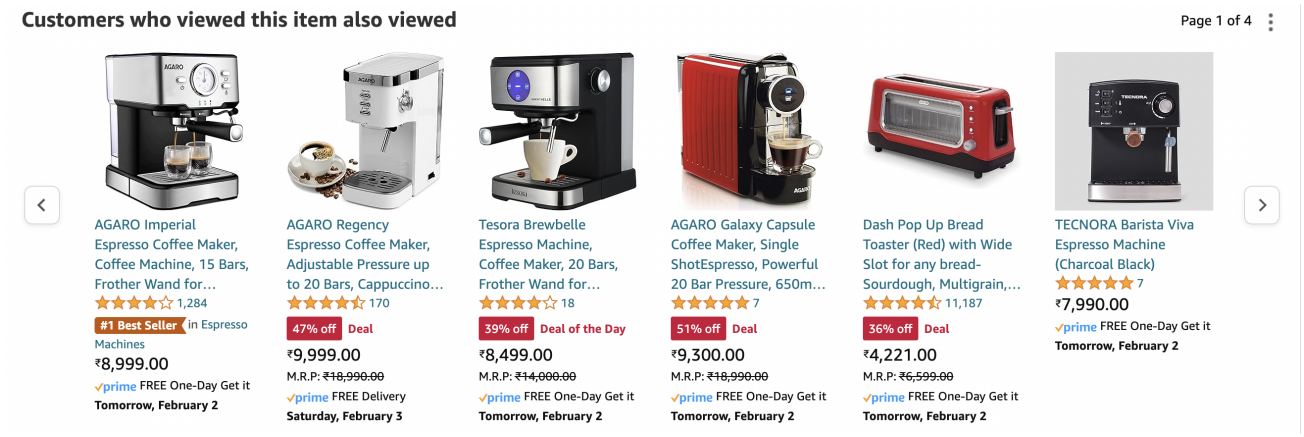
Netflix



- **Collaborative Filtering**-
  - This system matches persons with similar interests and provides recommendations based on this matching.

- Collaborative filters do not require item metadata like its content-based counterparts.

Amazon



Let's load the data now.

In [1]:
```
import pandas as pd


df1 = pd.read_csv("tmdb_5000_credits.csv")
df2 = pd.read_csv("tmdb_5000_movies.csv")
```

The first dataset contains the following features:-

- movie_id - A unique identifier for each movie.
- cast - The name of lead and supporting actors.
- crew - The name of Director, Editor, Composer, Writer etc.

The second dataset has the following features:-

- budget - The budget in which the movie was made.
- genre - The genre of the movie, Action, Comedy, Thriller etc.
- homepage - A link to the homepage of the movie.
- id - This is infact the movie_id as in the first dataset.
- keywords - The keywords or tags related to the movie.
- original_language - The language in which the movie was made.
- original_title - The title of the movie before translation or adaptation.
- overview - A brief description of the movie.
- popularity - A numeric quantity specifying the movie popularity.
- production_companies - The production house of the movie.
- production_countries - The country in which it was produced.
- release_date - The date on which it was released.
- revenue - The worldwide revenue generated by the movie.
- runtime - The running time of the movie in minutes.
- status - "Released" or "Rumored".
- tagline - Movie's tagline.
- title - Title of the movie.
- vote_average - average ratings the movie recieved.
- vote_count - the count of votes recieved.

Let's join the two dataset on the 'id' column

In [ ]:
```
df1.info()
```

In [ ]:
```
df2.info()
```

In [ ]:
```
print(df1.columns)
df1.columns = ["id", "title", "cast", "crew"]  # rename movie_id to id
df2 = df2.merge(df1[["id", "cast", "crew"]], on="id")
```

In [ ]:
```
df2.info()
```

Just a peak at our data.

In [ ]:
```
df2.head(5)
```

Demographic Filtering -

Before getting started with this -

- we need a metric to score or rate a movie
- Calculate the score for every movie
- Sort the scores and recommend the best rated movie to the users.

We can use the average ratings of the movie as the score but using this won't be fair enough since a movie with 8.9 average rating and only 3 votes cannot be considered better than the movie with 7.8 as as average rating but 40 votes. So, I'll be using IMDB's weighted rating (wr) which is given as :-

$$WR = \left(\frac{v}{v+m} \cdot R\right) + \left(\frac{m}{v+m} \cdot C\right)$$

where,

- v is the number of votes for the movie;
- m is the minimum votes required to be listed in the chart;
- R is the average rating of the movie;
- C is the mean vote across the whole report

We already have v (**vote_count**) and R (**vote_average**) and C can be calculated as

In [ ]:
```
C = df2["vote_average"].mean()
print(f"Average vote: {C.round(2)}")
```

So, the mean rating for all the movies is approx 6 on a scale of 10. The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use 90th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 90% of the movies in the list.

In [ ]:
```
# the quantile of the vote count
# is the minimum number of votes required to be in the chart
# calculated as the 90th percentile of all vote counts in the data
# which means only 10% of the movies have more votes than this
m = df2["vote_count"].quantile(0.9)
print(f"Quantile: {int(m)}")
```

Now, we can filter out the movies that qualify for the chart

In [ ]:
```
q_movies = df2.copy().loc[df2["vote_count"] >= m]
q_movies.shape
```

We see that there are 481 movies which qualify to be in this list. Now, we need to calculate our metric for each qualified movie. To do this, we will define a function, **weighted_rating()** and define a new feature **score**, of which we'll calculate the value by applying this function to our DataFrame of qualified movies:

In [11]:
```
def weighted_rating(df, m=m, C=C):
    v = df["vote_count"]
    R = df["vote_average"]
    # Calculation based on the IMDB formula
    return (v / (v + m) * R) + (m / (m + v) * C)
```

In [12]:
```
# Define a new feature 'score' and calculate its value with `weighted_rating()`
# apply function to each row (column-wise)
q_movies["score"] = q_movies.apply(weighted_rating, axis=1)
```

Finally, let's sort the DataFrame based on the score feature and output the title, vote count, vote average and weighted rating or score of the top 10 movies.

In [ ]:
```
# 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(15)
```

Hurray! We have made our first (though very basic) recommender. Under the **Trending Now** tab of these systems we find movies that are very popular and they can just be obtained by sorting the dataset by the popularity column.

In [ ]:
```
import matplotlib.pyplot as plt

pop = df2.sort_values("popularity", ascending=False)

fig = plt.figure(figsize=(12, 4))

plt.barh(
    pop["title"].head(6), pop["popularity"].head(6), align="center", color="skyblue"
)
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")
plt.show()
plt.close(fig)
```

In [ ]:
```
q_movies._get_numeric_data().corr().style.background_gradient(cmap="coolwarm")
```

Now something to keep in mind is that these demographic recommender provide a general chart of recommended movies to all the users. They are not sensitive to the interests and tastes of a particular user.

To read the full article click https://www.kaggle.com/code/ibtesama/getting-started-with-a-movie-recommendation-system