The knowledge-based recommender

Juan David Serna Valderrama

We are going to go ahead and build a knowledge-based recommender on top of our IMDB Top 250 clone.

Tasks:

1. Ask the user for the genres of movies he/she is looking for

2. Ask the user for the duration

3. Ask the user for the timeline of the movies recommended

4. Using the information collected, recommend movies to the user that have a high weighted rating (according to the IMDB formula) and that satisfy the preceding conditions

In [16]:
```
import pandas as pd
import numpy as np

df = pd.read_csv("C:/Users/juand/OneDrive/Escritorio/Recommendation Systems with Python/Data/movies_metadata.csv")

df.columns
```

```
c:\users\juand\appdata\local\programs\python\python38\lib\site-packages\IPython\core\interactiveshell.py:3145: DtypeWarning: Columns (10) have mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[16]:
```
Index(['adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id',
       'imdb_id', 'original_language', 'original_title', 'overview',
       'popularity', 'poster_path', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'video',
       'vote_average', 'vote_count'],
      dtype='object')
```

Only keep those features that we require

In [17]:
```
df = df[['title','genres', 'release_date', 'runtime', 'vote_average', 'vote_count']]

df.head()
```

Out[17]:

	title	genres	release_date	runtime	vote_average	vote_count
0	Toy Story	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	1995-10-30	81.0	7.7	5415.0
1	Jumanji	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	1995-12-15	104.0	6.9	2413.0
2	Grumpier Old Men	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	1995-12-22	101.0	6.5	92.0
3	Waiting to Exhale	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	1995-12-22	127.0	6.1	34.0
4	Father of the Bride Part II	[{'id': 35, 'name': 'Comedy'}]	1995-02-10	106.0	5.7	173.0

Convert release_date into pandas datetime format

In [18]:
```
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

#Extract year from the datetime
df['year'] = df['release_date'].apply(lambda x: str(x).split('-')[0] if x != np.nan else np.nan)
```

Helper function to convert NaT to 0 and all other years to integers.

In [19]:
```
def convert_int(x):
    try:
        return int(x)
    except:
        return 0

#Apply convert_int to the year feature
df['year'] = df['year'].apply(convert_int)
```

Drop the release_date column

In [20]:
```
df = df.drop('release_date', axis=1)

df.head()
```

Out[20]:

	title	genres	runtime	vote_average	vote_count	year
0	Toy Story	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	81.0	7.7	5415.0	1995
1	Jumanji	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	104.0	6.9	2413.0	1995
2	Grumpier Old Men	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	101.0	6.5	92.0	1995
3	Waiting to Exhale	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	127.0	6.1	34.0	1995
4	Father of the Bride Part II	[{'id': 35, 'name': 'Comedy'}]	106.0	5.7	173.0	1995

Genres

Print genres of the first movie

In [21]:
```
df.iloc[0]['genres']
```

Out[21]:
```
"[{'id': 16, 'name': 'Animation'}, {'id': 35, 'name': 'Comedy'}, {'id': 10751, 'name': 'Family'}]"
```

Import the literal_eval function from ast

In [22]:
```
from ast import literal_eval

#Define a stringified list and output its type
a = "[1,2,3]"
print(type(a))

#Apply literal_eval and output type
b = literal_eval(a)
print(type(b))
```

```
<class 'str'>
<class 'list'>
```

Convert all NaN into stringified empty lists

In [23]:
```
df['genres'] = df['genres'].fillna('[]')

#Apply literal_eval to convert stringified empty lists to the list object
df['genres'] = df['genres'].apply(literal_eval)

#Convert list of dictionaries to a list of strings
df['genres'] = df['genres'].apply(lambda x: [i['name'].lower() for i in x] if isinstance(x, list) else [])
#https://stackoverflow.com/questions/60042516/how-do-i-figure-out-what-this-code-is-doing

df.head()
```

Out[23]:

	title	genres	runtime	vote_average	vote_count	year
0	Toy Story	[animation, comedy, family]	81.0	7.7	5415.0	1995
1	Jumanji	[adventure, fantasy, family]	104.0	6.9	2413.0	1995
2	Grumpier Old Men	[romance, comedy]	101.0	6.5	92.0	1995
3	Waiting to Exhale	[comedy, drama, romance]	127.0	6.1	34.0	1995
4	Father of the Bride Part II	[comedy]	106.0	5.7	173.0	1995

Create a new feature by exploding genres

In [24]:
```
s = df.apply(lambda x: pd.Series(x['genres']), axis=1).stack().reset_index(level=1, drop=True)
#https://stackoverflow.com/questions/41472234/how-to-stack-the-wthin-in-a-pandas-dataframe-carrying-out-its-reference

#Name the new feature as 'genre'
s.name = 'genre'

#Create a new dataframe gen_df which by dropping the old 'genres' feature and adding the new 'genre'.
gen_df = df.drop('genres', axis=1).join(s)

gen_df.head()
```

```
<ipython-input-24-cedad3434e83>:1: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.
  s = df.apply(lambda x: pd.Series(x['genres']), axis=1).stack().reset_index(level=1, drop=True)
```

Out[24]:

	title	runtime	vote_average	vote_count	year	genre
0	Toy Story	81.0	7.7	5415.0	1995	animation
0	Toy Story	81.0	7.7	5415.0	1995	comedy
0	Toy Story	81.0	7.7	5415.0	1995	family
1	Jumanji	104.0	6.9	2413.0	1995	adventure
1	Jumanji	104.0	6.9	2413.0	1995	fantasy

The build_chart function

1. Get user input on their preferences

2. Extract all movies that match the conditions set by the user

3. Calculate the values of m and C for only these movies and proceed to build the chart as in the previous section

In [25]:
```
def build_chart(gen_df, percentile=0.8):
    #Ask for preferred genres
    print("Input preferred genre")
    genre = input()

    #Ask for lower limit of duration
    print("Input shortest duration")
    low_time = int(input())

    #Ask for upper limit of duration
    print("Input longest duration")
    high_time = int(input())

    #Ask for lower limit of timeline
    print("Input earliest year")
    low_year = int(input())

    #Ask for upper limit of timeline
    print("Input latest year")
    high_year = int(input())

    #Define a new movies variable to store the preferred movies. Copy the contents of gen_df to movies
    movies = gen_df.copy()

    #Filter based on the condition
    movies = movies[(movies['genre'] == genre) &
                    (movies['runtime'] >= low_time) &
                    (movies['runtime'] <= high_time) &
                    (movies['year'] >= low_year) &
                    (movies['year'] <= high_year)]

    #Compute the values of C and m for the filtered movies
    C = movies['vote_average'].mean()
    m = movies['vote_count'].quantile(percentile)

    #Only consider movies that have higher than m votes. Save this in a new dataframe q_movies
    q_movies = movies.copy().loc[movies['vote_count'] >= m]

    #Calculate score using the IMDB formula
    q_movies['score'] = q_movies.apply(lambda x: (x['vote_count']/(x['vote_count']+m) * x['vote_average'])
                                       + (m/(m+x['vote_count']) * C)
                                       ,axis=1)

    #Sort movies in descending order of their scores
    q_movies = q_movies.sort_values('score', ascending=False)

    return q_movies
```

Generate the chart for top movies and display top 5.

In [28]:
```
build_chart(gen_df).head()
```

```
Input preferred genre
animation
Input shortest duration
80
Input longest duration
120
Input earliest year
1990
Input latest year
2000
```

Out[28]:

	title	runtime	vote_average	vote_count	year	genre	score
359	The Lion King	89.0	8.0	5520.0	1994	animation	7.806465
0	Toy Story	81.0	7.7	5415.0	1995	animation	7.536451
588	Beauty and the Beast	84.0	7.5	3029.0	1991	animation	7.267660
1798	Mulan	88.0	7.6	2089.0	1998	animation	7.264225
581	Aladdin	90.0	7.4	3495.0	1992	animation	7.209834

In [ ]:
```
```