DATA SCIENCE PROJECT

Building a Movie/Show Recommendation System for Hutu TV

Student: Phan Thuy Anh

Class: Business Analytics 62

introduction

objective

Create a recommendation system tailored to the specific content offerings of Hutu TV.

dataset overview

The dataset is available in Kaggle (2022): Hulu TV Shows and Movies.

look through the dataset...

data dictionary

attributes	descrition
id	Unique identifier for each movie/show
title	The title of the movie or show.
type	Whether the entry is a movie or a show.
description	A brief summary or synopsis of the movie or show's plot or content.
release_year	The year the movie or show was released.
age_certification	The age group for which the content is deemed appropriate.
runtime	The duration of the movie or show in minutes.
genres	The categories or genres that the movie or show belongs to.
production_countries	The countries where the movie or show was produced.
seasons	The number of seasons available.
imdb_id	The unique identifier for the movie or show on IMDb.
imdb_score	The rating score given to the movie or show on IMDb.
imdb_votes	The number of votes/ratings received on IMDb.
tmdb_popularity	A popularity score for the movie or show on TMDb.
tmdb_score	The rating score given to the movie or show on TMDb.

look through the dataset...

shape

```
#Explore data
df.shape
```

```
(2398, 15)
```

information

```
#Explore data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2398 entries, 0 to 2397
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    2398 non-null   object 
 1   title                 2398 non-null   object 
 2   type                  2398 non-null   object 
 3   description           2385 non-null   object 
 4   release_year          2398 non-null   int64  
 5   age_certification     1713 non-null   object 
 6   runtime               2398 non-null   int64  
 7   genres                2398 non-null   object 
 8   production_countries  2398 non-null   object 
 9   seasons               1330 non-null   float64
 10  imdb_id               2263 non-null   object 
 11  imdb_score            2232 non-null   float64
 12  imdb_votes            2231 non-null   float64
 13  tmdb_popularity       2348 non-null   float64
 14  tmdb_score            2238 non-null   float64
dtypes: float64(5), int64(2), object(8)
memory usage: 281.1+ KB
```

DATA PREPROCESSING

import libraries

```
#Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics.pairwise import linear_kernel
import tensorflow as tf
from tensorflow.keras.models import Model
from keras.models import Sequential
from tensorflow.keras.layers import Input, Embedding,
Flatten, Dense, Concatenate
```

load data

```
#Load data
df = pd.read_csv("hulutv.csv")
```

clean data

handling missing values

```
def missing_values_table(df):
        mis_val = df.isnull().sum()
        mis_val_percent = 100*df.isnull().sum() / len(df)
        mis_val_table = pd.concat([mis_val,
mis_val_percent], axis=1)
        mis_val_table_ren_columns =
mis_val_table.rename(columns = {0:'Missing Values' ,
1:'% of Total Values'})
          return mis_val_table_ren_columns.round(1)
missing_values_table(df)
```

	Missing Values	% of Total Values
id	0	0.0
title	0	0.0
type	0	0.0
description	13	0.5
release_year	0	0.0
age_certification	685	28.6
runtime	0	0.0
genres	0	0.0
production_countries	0	0.0
seasons	1068	44.5
imdb_id	135	5.6
imdb_score	166	6.9
imdb_votes	167	7.0
tmdb_popularity	50	2.1
tmdb_score	160	6.7

clean data

drop null values in attributes that have small proportions

```
#Drop the null values
null = ['description','imdb_score', 'imdb_score', 'imdb_votes', 'tmdb_popularity', 'tmdb_score']
df.dropna(subset=null , inplace=True)
```

‘season’: replace the null values with value “0”

```
#Deal with "season"
df['seasons'].fillna(0, inplace=True)
```

note: only type “movie” has null value in ‘season’

‘age certification’: drop out

```
#Deal with "age certification"
df.drop(['age_certification'], axis=1,inplace=True)
```

clean data

check missing value again

```
#Check missing value again
df.isnull().sum()
```

```
id                      0
title                   0
type                    0
description             0
release_year            0
runtime                 0
genres                  0
production_countries    0
seasons                 0
imdb_id                 0
imdb_score              0
imdb_votes              0
tmdb_popularity         0
tmdb_score              0
dtype: int64
```

clean data

check duplicates

```
#Check duplicates
df.duplicated().sum()
```

```
0
```

clean data

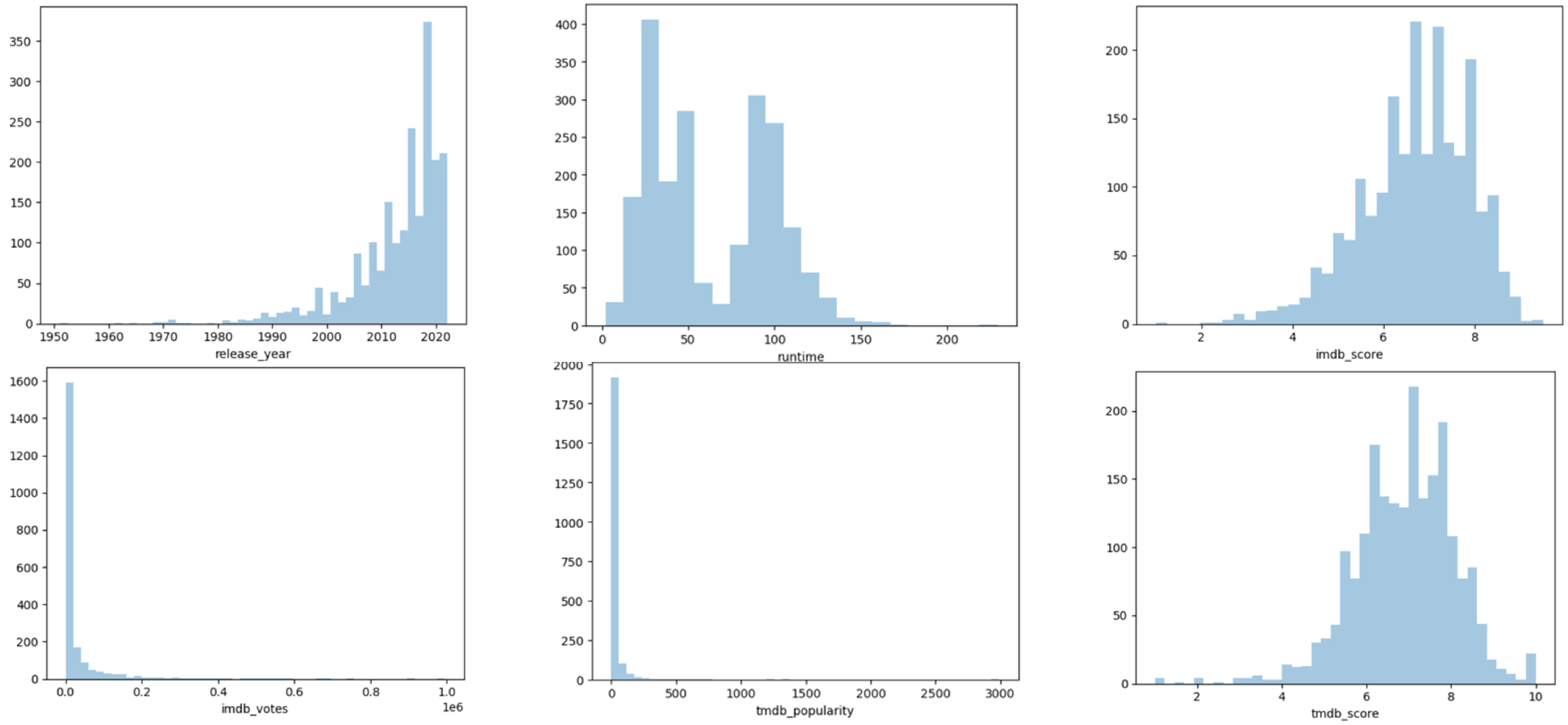
encoding the categorical data

```
#Encoding the neccessary categorical variables
encoder = LabelEncoder()
df["title_encoded"] = encoder.fit_transform(df["title"])
df["type_encoded"] = encoder.fit_transform(df["type"])
```

EXPLORATORY DATA ANALYSIS

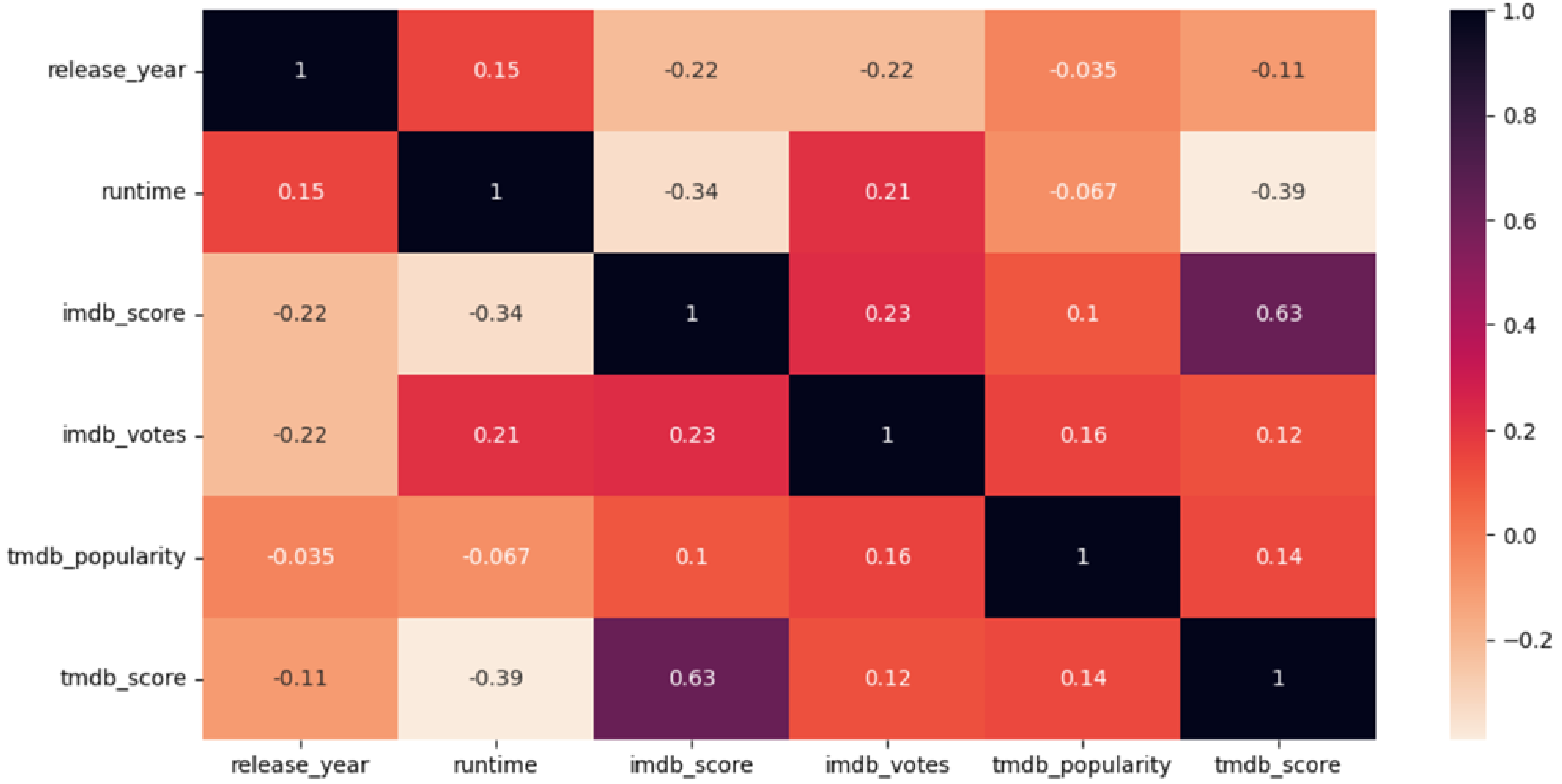
distribution of attributes

```
# Distribution of attributes
features = ['release_year','runtime', 'imdb_score','imdb_votes', 'tmdb_popularity','tmdb_score']
for feat in features:
    plt.figure()
    sns.distplot(df[feat], kde = False)
```



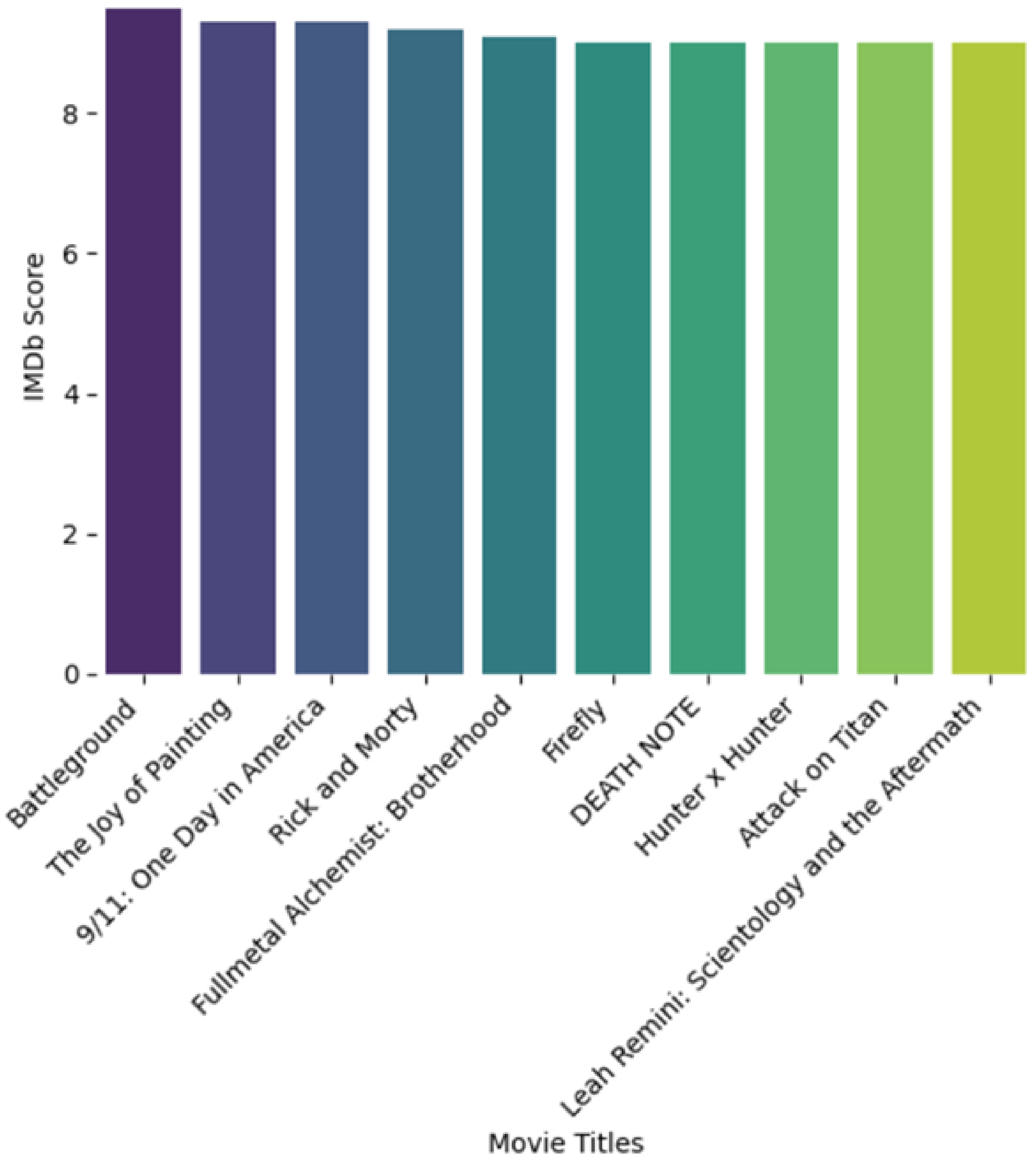
correlation between numeric attributes

```
# Correlation between numeric attributes
plt.figure(figsize = (12, 6))
sns.heatmap(df[features].corr(), annot = True, cmap = 'rocket_r')
```

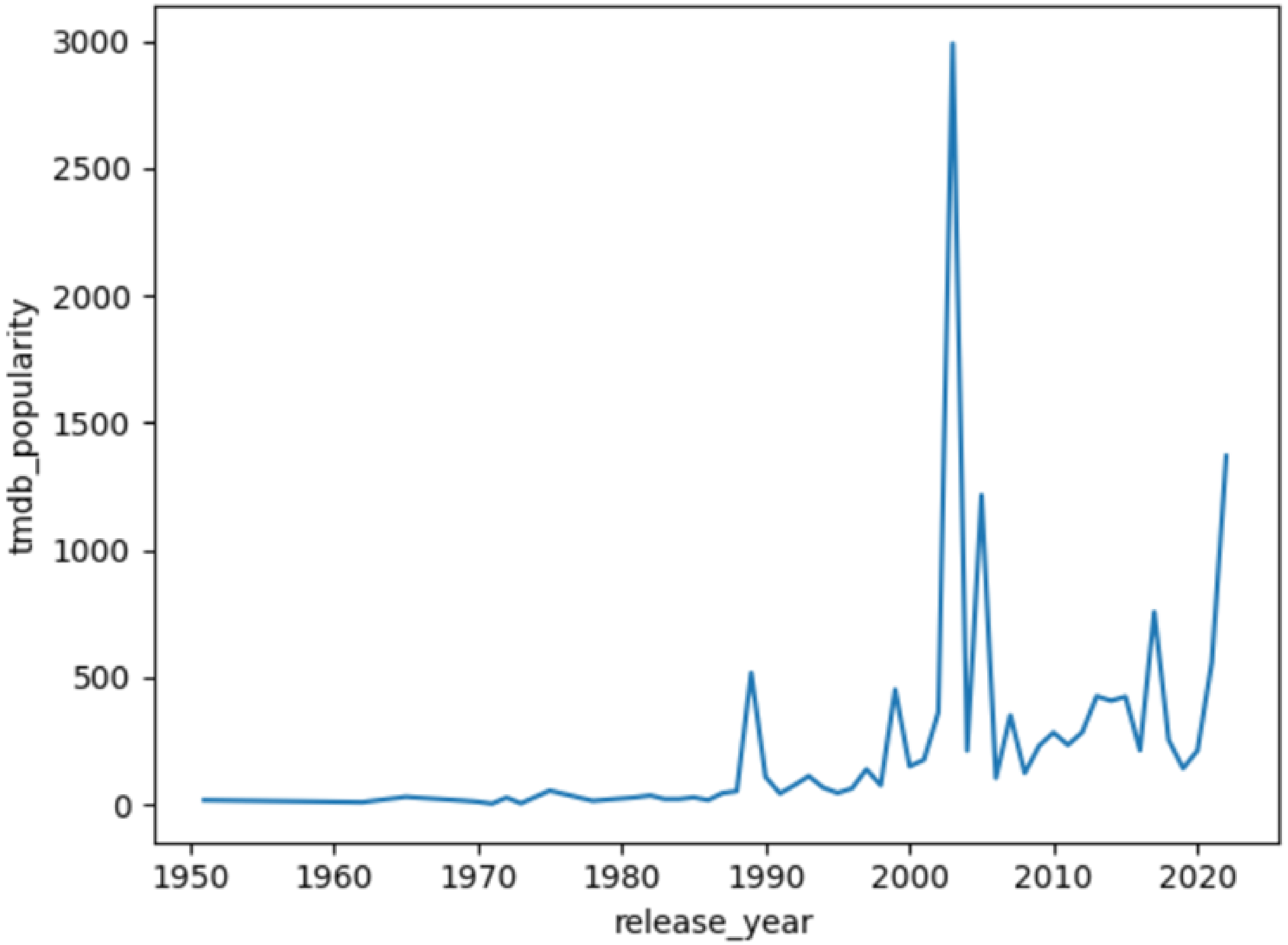


There are no variables that have extremely strong correlation.
The highest is belong to positive relationship of tmdb_score and imdb_score (0.63).

top 10 movies/shows have highest IMDB score



maximum popularity by release year



BUILDING RECOMMENDATION SYSTEM FOR HULU

about recommendation system...

type

Content-Based

Focus on attributes of items (songs) and recommends based on similarity

Collaborative Filtering

Focus on wisdom of the crowd, recommends based on other users

content-based recommendation system

cosine similarity

$$sim(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

How cosine similarity works:
Consider: vector: A and B
Ai and Bi: representing features in each movie or show.

How the cosine similarity means: A higher cosine similarity value indicates a greater resemblance, which translates to a higher position on the recommendation list.

building content-based recommendation system

Term Frequency-Inverse Document Frequency (**TF-IDF**)**:** count occurrences of words in 'description' and weighing the importance of words to calculate a score.

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

#Fit and transform the data
tfidf_matrix = tfidf.fit_transform(df['description'])
tfidf_matrix.shape
```

```
(2106, 14045)
```

building content-based recommendation system

calculating the cosine similarity between all pairs of items

```
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

a mapping between titles and dataset indices

```
indices = pd.Series(df.index, index=df['title']).drop_duplicates()
```

building content-based recommendation system

function: use cosine similarities matrix to streamline the recommendation

```
def get_recommendations(title, cosine_sim=cosine_sim):
    idx = indices[title]

    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]

    movie_indices = [i[0] for i in sim_scores]
    similarity_scores = [i[1] for i in sim_scores]

    results = pd.DataFrame({'title': df['title'].iloc[movie_indices], 'cosine_similarity':
similarity_scores})

    return results
```

testing content-based recommendation system

```
#Test content-based recommendation system
get_recommendations('Battleground')
```

	title	cosine_similarity
2172	The Wonder Years	0.107581
2299	kid 90	0.100477
456	My Wife and Kids	0.097814
1166	Pickle & Peanut	0.092072
859	The Killer Speaks	0.082975
1711	Rent-A-Pal	0.075277
1931	mixed-ish	0.074442
1741	Bless This Mess	0.074233
1437	OK K.O.! Let's Be Heroes	0.070904
1025	Fresh Off the Boat	0.070690

The cosine similarity scores are small (often below 0,1) => Not a effective recommendation approach

collaborative filtering recommendation system

SVD: user-item matrix

user

item

diagonal matrix

How SVD works:
Reduction of dimensionality allows the system to focus on the most significant characteristics while discarding less relevant ones.

How SVD be applied: SVD-based recommendation system provides personalized suggestions by utilizing latent features from user-item interactions.

building collaborative filtering recommendation system

create user-item matrix

```
num_users = 1
num_items = len(df)
user_item_matrix = np.zeros((num_users, num_items))

#Assume all items have the same ratings for this user
ratings = np.ones(num_items)

user_item_matrix[0, :] = ratings
```

building collaborative filtering recommendation system

compute similarities between items using embedding and post-trainning

```
#Define the SVD model
embedding_dim = 10
SVDmodel = tf.keras.Sequential([
  tf.keras.layers.Embedding(num_items, embedding_dim, input_length=num_items),
  tf.keras.layers.Reshape((embedding_dim, num_items)),
  tf.keras.layers.Lambda(lambda x: tf.reduce_mean(x, axis=1))
])

#Compile the model
SVDmodel.compile(optimizer='adam', loss='mse')

#Train the model
SVDmodel.fit(user_item_matrix, user_item_matrix, epochs=10)

#Use the learned embeddings to make recommendations
user_embedding = SVDmodel.layers[0].get_weights()[0][0]
similarities = np.dot(model.layers[0].get_weights()[0], user_embedding)

#Combine recommendations with their similarity scores
recommendations = [(df['title'].iloc[i], similarities[i]) for i in range(num_items)]
```

testing collaborative filtering recommendation system

```
#Sort recommendations by similarity score
recommendations.sort(key=lambda x: x[1], reverse=True)

#Create a DataFrame to store the results
SVD_recommenddations = pd.DataFrame({'title': [rec[0] for rec in recommendations], 'similarities': [rec[1] for rec in recommendations]})

#Return the result
SVD_recommendations
```

	title	similarities
0	Astra Lost in Space	0.006830
1	Gargantia on the Verdurous Planet	0.006794
2	Sister of the Groom	0.006649
3	Sex Ed	0.006638
4	Terriers	0.006417
...	...	...
2101	Bleak House	-0.007904
2102	Better Things	-0.008247
2103	Harmony From The Heart	-0.008346
2104	Knowing	-0.008668
2105	Ghosts of Mars	-0.009116

The similarity scores are also small. This could be due to insufficient data for accurate recommendations.

evaluate the SVD recommendation system model

```
#Calculate evaluation metric
actual_ratings = df['imdb_score'].values
SVDpredicted_ratings = np.array([rec[1] for rec in recommendations])
rmse_svd = np.sqrt(mean_squared_error(actual_ratings, SVDpredicted_ratings))
print(f'Root Mean Squared Error (RMSE) of SVD model: {rmse_svd}')
```

```
Root Mean Squared Error (RMSE) of SVD model: 6.805309918935054
```

THANK YOU FOR LISTENING

Reference: Victor Soeiro (2022). Hulu TV Shows and Movies. Kaggle.
https://www.kaggle.com/datasets/victorsoeiro/hulu-tv-shows-and-movies