In [1]:
```
cd
```

```
C:\Users\WELCOME
```

In [2]:
```
import pandas as pd
```

In [3]:
```
movies = pd.read_csv("movies.csv")
```

In [4]:
```
movies
```

Out[4]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
62418	209157	We (2018)	Drama
62419	209159	Window of the Soul (2001)	Documentary
62420	209163	Bad Poems (2018)	Comedy|Drama
62421	209169	A Girl Thing (2001)	(no genres listed)
62422	209171	Women of Devil's Island (1962)	Action|Adventure|Drama

62423 rows × 3 columns

In [5]:
```
import re
```

In [6]:
```
def clean_title(title):
    return re.sub("[^a-zA-Z0-9 ]","", title)
```

In [7]:
```
movies["clean_title"] = movies["title"].apply(clean_title)
```

In [8]:
```
movies
```

Out[8]:

	movieId	title	genres	clean_title
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 1995
1	2	Jumanji (1995)	Adventure|Children|Fantasy	Jumanji 1995
2	3	Grumpier Old Men (1995)	Comedy|Romance	Grumpier Old Men 1995
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	Waiting to Exhale 1995
4	5	Father of the Bride Part II (1995)	Comedy	Father of the Bride Part II 1995
...	...	...	...	...
62418	209157	We (2018)	Drama	We 2018
62419	209159	Window of the Soul (2001)	Documentary	Window of the Soul 2001
62420	209163	Bad Poems (2018)	Comedy|Drama	Bad Poems 2018
62421	209169	A Girl Thing (2001)	(no genres listed)	A Girl Thing 2001
62422	209171	Women of Devil's Island (1962)	Action|Adventure|Drama	Women of Devils Island 1962

62423 rows × 4 columns

In [9]:
```
# Creating a TF-IDF (Term Frequency - Inverse Document Frequency) matrix
```

In [10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [11]:
```
vectorizer = TfidfVectorizer(ngram_range = (1,2))
tfidf = vectorizer.fit_transform(movies["clean_title"])
```

In [12]:
```
# Creating a search function
```

In [13]:
```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

def search(title):
    title = clean_title(title)
    query_vec = vectorizer.transform([title])
    similarity = cosine_similarity(query_vec, tfidf).flatten()
    indices = np.argpartition(similarity, -5)[-5:]
    results = movies.iloc[indices].iloc[::-1]
    return results
```

In [14]:
```
search("Toy story")
```

Out[14]:

	movieId	title	genres	clean_title
3021	3114	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 2 1999
14813	78499	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX	Toy Story 3 2010
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 1995
59767	201588	Toy Story 4 (2019)	Adventure|Animation|Children|Comedy	Toy Story 4 2019
20497	106022	Toy Story of Terror (2013)	Animation|Children|Comedy	Toy Story of Terror 2013

In [15]:
```
# Building a interactive search box using jupyter
```

In [16]:
```
import ipywidgets as widgets
from IPython.display import display

movie_input = widgets.Text(
    value = "Toy Story",
    description = "Movie Title : ",
    disabled = False
)

movie_list = widgets.Output()

def on_type(data):
    with movie_list:
        movie_list.clear_output()
        title = data["new"]
        if len(title) > 5:
            display(search(title))

movie_input.observe(on_type, names = 'value')
display(movie_input, movie_list)
```

```
Text(value='Toy Story', description='Movie Title : ')
Output()
```

In [17]:
```
# Reading in movie ratings data
ratings = pd.read_csv("ratings.csv")
```

In [18]:
```
ratings
```

Out[18]:

	userId	movieId	rating	timestamp
0	1	296	5.0	1147880044
1	1	306	3.5	1147868817
2	1	307	5.0	1147868828
3	1	665	5.0	1147878820
4	1	899	3.5	1147868510
...	...	...	...	...
25000090	162541	50872	4.5	1240953372
25000091	162541	55768	2.5	1240951998
25000092	162541	56176	2.0	1240950697
25000093	162541	58559	4.0	1240953434
25000094	162541	63876	5.0	1240952515

25000095 rows × 4 columns

In [19]:
```
ratings.dtypes
```

Out[19]:
```
userId         int64
movieId        int64
rating       float64
timestamp      int64
dtype: object
```

In [20]:
```
# Finding users who liked the same movie
```

In [21]:
```
movie_id = 1
```

In [22]:
```
similar_users = ratings[(ratings["movieId"] == movie_id) & (ratings["rating"] > 4)]["u
```

In [23]:
```
similar_users
```

Out[23]:
```
array([    36,     75,     86, ..., 162527, 162530, 162533], dtype=int64)
```

In [24]:
```
similar_user_recs = ratings[(ratings["userId"].isin(similar_users)) & (ratings["rating
```

In [25]:
```
similar_user_recs
```

Out[25]:
```
5101           1
5105          34
5111         110
5114         150
5127         260
            ...
24998854   60069
24998861   67997
24998876   78499
24998884   81591
24998888   88129
Name: movieId, Length: 1358326, dtype: int64
```

In [26]:
```
similar_user_recs = similar_user_recs.value_counts() / len(similar_users)

similar_user_recs = similar_user_recs[similar_user_recs > .1]
```

In [27]:
```
similar_user_recs
```

Out[27]:
```
1        1.000000
318      0.445607
260      0.403770
356      0.370215
296      0.367295
           ...
953      0.103053
551      0.101195
1222     0.100876
745      0.100345
48780    0.100186
Name: movieId, Length: 113, dtype: float64
```

In [28]:
```
# How much all users like the movie
```

In [29]:
```
all_users = ratings[(ratings["movieId"]).isin(similar_user_recs.index) & (ratings["rat
```

In [30]:
```
all_user_recs = all_users["movieId"].value_counts() / (len(all_users["userId"].unique(
```

In [31]:
```
rec_percentages = pd.concat([similar_user_recs, all_user_recs], axis = 1)
```

In [32]:
```
rec_percentages.columns = ["similar", "all"]
```

In [33]:
```
rec_percentages
```

Out[33]:

	similar	all
1	1.000000	0.124728
318	0.445607	0.342220
260	0.403770	0.222207
356	0.370215	0.235266
296	0.367295	0.284674
...	...	...
953	0.103053	0.045792
551	0.101195	0.040918
1222	0.100876	0.066877
745	0.100345	0.037031
48780	0.100186	0.068314

113 rows × 2 columns

In [34]:
```
rec_percentages["score"] = rec_percentages["similar"] / rec_percentages["all"]
```

In [35]:
```
rec_percentages = rec_percentages.sort_values("score", ascending = False)
```

In [36]:
```
rec_percentages.head(10).merge(movies, left_index = True, right_on = "movieId")
```

Out[36]:

	similar	all	score	movieId	title	ge
0	1.000000	0.124728	8.017414	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Far
3021	0.280648	0.053706	5.225654	3114	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Far
2264	0.110539	0.025091	4.405452	2355	Bug's Life, A (1998)	Adventure|Animation|Children|Con
14813	0.152960	0.035131	4.354038	78499	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|II
4780	0.235147	0.070811	3.320783	4886	Monsters, Inc. (2001)	Adventure|Animation|Children|Comedy|Far
580	0.216618	0.067513	3.208539	588	Aladdin (1992)	Adventure|Animation|Children|Comedy|Mu
6258	0.228139	0.072268	3.156862	6377	Finding Nemo (2003)	Adventure|Animation|Children|Con
587	0.179400	0.059977	2.991150	595	Beauty and the Beast (1991)	Animation|Children|Fantasy|Musical|Romance|II
8246	0.203504	0.068453	2.972889	8961	Incredibles, The (2004)	Action|Adventure|Animation|Children|Con
359	0.253411	0.085764	2.954762	364	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical|II

In [37]:
```
# Building a recommendation function
```

In [38]:
```
def find_similar_movie(movie_id):
    similar_users = ratings[(ratings["movieId"] == movie_id) & (ratings["rating"] > 4)
    similar_user_recs = ratings[(ratings["userId"].isin(similar_users)) & (ratings["ra

    similar_user_recs = similar_user_recs.value_counts() / len(similar_users)
    similar_user_recs = similar_user_recs[similar_user_recs > .1]

    all_users = ratings[(ratings["movieId"]).isin(similar_user_recs.index) & (ratings[
    all_users = ratings[(ratings["movieId"]).isin(similar_user_recs.index) & (ratings[

    rec_percentages = pd.concat([similar_user_recs, all_user_recs], axis = 1)
    rec_percentages.columns = ["similar", "all"]
    rec_percentages["score"] = rec_percentages["similar"] / rec_percentages["all"]
    rec_percentages = rec_percentages.sort_values("score", ascending = False)

    return rec_percentages.head(10).merge(movies, left_index = True, right_on = "movie
```

In [39]:
```
# Creating an interactive recommendation widget
```

In [40]:
```
movie_name_input = widgets.Text(
    value = "Toy story",
    description = "Movie title : ",
```

```
    disabled = False
)

recommendation_list = widgets.Output()

def on_type(data):
    with recommendation_list:
        recommendation_list.clear_output()
        title = data["new"]
        if len(title) > 5:
            results = search(title)
            movie_id = results.iloc[0]["movieId"]
            display(find_similar_movie(movie_id))

movie_name_input.observe(on_type, names = "value")
display(movie_name_input, recommendation_list)
```

```
Text(value='Toy story', description='Movie title : ')
Output()
```

In [ ]: