Plot Description Based Recommender

Juan David Serna Valderrama

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

#Import data from the clean file
df = pd.read_csv('../data/metadata_clean.csv')

#Print the head of the cleaned DataFrame
df.head()
```

Out[1]:

	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

In [2]:
```
#Import the original file
orig_df = pd.read_csv('../data/movies_metadata.csv', low_memory=False)

#Add the useful features into the cleaned dataframe
df['overview'], df['id'] = orig_df['overview'], orig_df['id']

df.head()
```

Out[2]:

	title	genres	runtime	vote_average	vote_count	year	overview	id
0	Toy Story	['animation', 'comedy', 'family']	81.0	7.7	5415.0	1995	Led by Woody, Andy's toys live happily in his ...	862
1	Jumanji	['adventure', 'fantasy', 'family']	104.0	6.9	2413.0	1995	When siblings Judy and Peter discover an encha...	8844
2	Grumpier Old Men	['romance', 'comedy']	101.0	6.5	92.0	1995	A family wedding reignites the ancient feud be...	15602
3	Waiting to Exhale	['comedy', 'drama', 'romance']	127.0	6.1	34.0	1995	Cheated on, mistreated and stepped on, the wom...	31357
4	Father of the Bride Part II	['comedy']	106.0	5.7	173.0	1995	Just when George Banks has recovered from his ...	11862

In [3]:
```
#Import TfIdfVectorizer from the scikit-learn library
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stopwords
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string
df['overview'] = df['overview'].fillna('')

#Construct the required TF-IDF matrix by applying the fit_transform method on the overview feature
tfidf_matrix = tfidf.fit_transform(df['overview'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

Out[3]:
```
(45466, 75827)
```

In [4]:
```
# Import linear_kernel to compute the dot product
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [5]:
```
#Construct a reverse mapping of indices and movie titles, and drop duplicate titles, if any
indices = pd.Series(df.index, index=df['title']).drop_duplicates()
```

In [6]:
```
# Function that takes in movie title as input and gives recommendations
def content_recommender(title, cosine_sim=cosine_sim, df=df, indices=indices):
    # Obtain the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    # And convert it into a list of tuples as described above
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the cosine similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies. Ignore the first movie.
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df['title'].iloc[movie_indices]
```

In [7]:
```
#Get recommendations for The Lion King
content_recommender('The Lion King')
```

Out[7]:
```
34682    How the Lion Cub and the Turtle Sang a Song
9353                                 The Lion King 1½
9115                   The Lion King 2: Simba's Pride
42829                                            Prey
25654                                  Fearless Fagan
17041                                    African Cats
27933              Massaï, les guerriers de la pluie
6094                                        Born Free
37409                                      Sour Grape
3203                                 The Waiting Game
Name: title, dtype: object
```

Metadata Based Recommender

In [8]:
```
# Load the keywords and credits files
cred_df = pd.read_csv('../data/credits.csv')
key_df = pd.read_csv('../data/keywords.csv')
```

In [9]:
```
#Print the head of the credit dataframe
cred_df.head()
```

Out[9]:

	cast	crew	id
0	[{'cast_id': 14, 'character': 'Woody (voice)',...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	862
1	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	8844
2	[{'cast_id': 2, 'character': 'Max Goldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...	15602
3	[{'cast_id': 1, 'character': "Savannah 'Vannah...	[{'credit_id': '52fe44779251416c91011acb', 'de...	31357
4	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...	11862

In [10]:
```
#Print the head of the keywords dataframe
key_df.head()
```

Out[10]:

	id	keywords
0	862	[{'id': 931, 'name': 'jealousy'}, {'id': 4290,...
1	8844	[{'id': 10090, 'name': 'board game'}, {'id': 1...
2	15602	[{'id': 1495, 'name': 'fishing'}, {'id': 12392...
3	31357	[{'id': 818, 'name': 'based on novel'}, {'id':...
4	11862	[{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...

In [11]:
```
#Convert the IDs of df into int
df['id'] = df['id'].astype('int')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-d101985d6747> in <module>()
      1 #Convert the IDs of df into int
----> 2 df['id'] = df['id'].astype('int')

/usr/local/lib/python3.6/site-packages/pandas/util/_decorators.py in wrapper(*args, **kwargs)
     89                 else:
     90                     kwargs[new_arg_name] = new_arg_value
---> 91             return func(*args, **kwargs)
     92         return wrapper
     93     return _deprecate_kwarg

/usr/local/lib/python3.6/site-packages/pandas/core/generic.py in astype(self, dtype, copy, errors, **kwargs)
   3408         # else, only a single dtype is given
   3409         new_data = self._data.astype(dtype=dtype, copy=copy, errors=errors,
-> 3410                                      **kwargs)
   3411         return self._constructor(new_data).__finalize__(self)
   3412

/usr/local/lib/python3.6/site-packages/pandas/core/internals.py in astype(self, dtype, **kwargs)
   3222
   3223     def astype(self, dtype, **kwargs):
-> 3224         return self.apply('astype', dtype=dtype, **kwargs)
   3225
   3226     def convert(self, **kwargs):

/usr/local/lib/python3.6/site-packages/pandas/core/internals.py in apply(self, f, axes, filter, do_integrity_check, consolidate, **kwargs)
   3089
   3090             kwargs['mgr'] = self
-> 3091             applied = getattr(b, f)(**kwargs)
   3092             result_blocks = _extend_blocks(applied, result_blocks)
   3093

/usr/local/lib/python3.6/site-packages/pandas/core/internals.py in astype(self, dtype, copy, errors, values, **kwargs)
    469     def astype(self, dtype, copy=False, errors='raise', values=None, **kwargs):
    470         return self._astype(dtype, copy=copy, errors=errors, values=values,
--> 471                             **kwargs)
    472
    473     def _astype(self, dtype, copy=False, errors='raise', values=None,

/usr/local/lib/python3.6/site-packages/pandas/core/internals.py in _astype(self, dtype, copy, errors, values, klass, mgr, raise_on_error, **kwargs)
    519
    520                 # _astype_nansafe works fine with 1-d only
--> 521                 values = astype_nansafe(values.ravel(), dtype, copy=True)
    522                 values = values.reshape(self.shape)
    523

/usr/local/lib/python3.6/site-packages/pandas/core/dtypes/cast.py in astype_nansafe(arr, dtype, copy)
    623     elif arr.dtype == np.object_ and np.issubdtype(dtype.type, np.integer):
    624         # work around NumPy brokenness, #1987
--> 625         return lib.astype_intsafe(arr.ravel(), dtype).reshape(arr.shape)
    626
    627     if dtype.name in ("datetime64", "timedelta64"):

pandas/_libs/lib.pyx in pandas._libs.lib.astype_intsafe (pandas/_libs/lib.c:16264)()

pandas/_libs/src/util.pxd in util.set_value_at_unsafe (pandas/_libs/lib.c:73298)()

ValueError: invalid literal for int() with base 10: '1997-08-20'
```

In [12]:
```
# Function to convert all non-integer IDs to NaN
def clean_ids(x):
    try:
        return int(x)
    except:
        return np.nan
```

In [13]:
```
#Clean the ids of df
df['id'] = df['id'].apply(clean_ids)

#Filter all rows that have a null ID
df = df[df['id'].notnull()]
```

In [14]:
```
# Convert IDs into integer
df['id'] = df['id'].astype('int')
key_df['id'] = key_df['id'].astype('int')
cred_df['id'] = cred_df['id'].astype('int')

# Merge keywords and credits into your main metadata dataframe
df = df.merge(cred_df, on='id')
df = df.merge(key_df, on='id')

#Display the head of df
df.head()
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

Out[14]:

	title	genres	runtime	vote_average	vote_count	year	overview	id	cast	crew	keywords
0	Toy Story	['animation', 'comedy', 'family']	81.0	7.7	5415.0	1995	Led by Woody, Andy's toys live happily in his ...	862	[{'cast_id': 14, 'character': 'Woody (voice)',...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	[{'id': 931, 'name': 'jealousy'}, {'id': 4290,...
1	Jumanji	['adventure', 'fantasy', 'family']	104.0	6.9	2413.0	1995	When siblings Judy and Peter discover an encha...	8844	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	[{'id': 10090, 'name': 'board game'}, {'id': 1...
2	Grumpier Old Men	['romance', 'comedy']	101.0	6.5	92.0	1995	A family wedding reignites the ancient feud be...	15602	[{'cast_id': 2, 'character': 'Max Goldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...	[{'id': 1495, 'name': 'fishing'}, {'id': 12392...
3	Waiting to Exhale	['comedy', 'drama', 'romance']	127.0	6.1	34.0	1995	Cheated on, mistreated and stepped on, the wom...	31357	[{'cast_id': 1, 'character': "Savannah 'Vannah...	[{'credit_id': '52fe44779251416c91011acb', 'de...	[{'id': 818, 'name': 'based on novel'}, {'id':...
4	Father of the Bride Part II	['comedy']	106.0	5.7	173.0	1995	Just when George Banks has recovered from his ...	11862	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...	[{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...

In [15]:
```
# Convert the stringified objects into the native python objects
from ast import literal_eval

features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    df[feature] = df[feature].apply(literal_eval)
```

In [16]:
```
#Print the first cast member of the first movie in df
df.iloc[0]['crew'][0]
```

Out[16]:
```
{'credit_id': '52fe4284c3a36847f8024f49',
 'department': 'Directing',
 'gender': 2,
 'id': 7879,
 'job': 'Director',
 'name': 'John Lasseter',
 'profile_path': '/7EdqiNbr4FRjIhKHyPPdFfEEEFG.jpg'}
```

In [17]:
```
# Extract the director's name. If director is not listed, return NaN
def get_director(x):
    for crew_member in x:
        if crew_member['job'] == 'Director':
            return crew_member['name']
    return np.nan
```

In [18]:
```
#Define the new director feature
df['director'] = df['crew'].apply(get_director)

#Print the directors of the first five movies
df['director'].head()
```

Out[18]:
```
0      John Lasseter
1       Joe Johnston
2      Howard Deutch
3    Forest Whitaker
4      Charles Shyer
Name: director, dtype: object
```

In [19]:
```
# Returns the list top 3 elements or entire list; whichever is more.
def generate_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        #Check if more than 3 elements exist. If yes, return only first three. If no, return entire list.
        if len(names) > 3:
            names = names[:3]
        return names

    #Return empty list in case of missing/malformed data
    return []
```

In [20]:
```
#Apply the generate_list function to cast and keywords
df['cast'] = df['cast'].apply(generate_list)
df['keywords'] = df['keywords'].apply(generate_list)
```

In [21]:
```
#Only consider a maximum of 3 genres
df['genres'] = df['genres'].apply(lambda x: x[:3])
```

In [22]:
```
# Print the new features of the first 5 movies along with title
df[['title', 'cast', 'director', 'keywords', 'genres']].head()
```

Out[22]:

	title	cast	director	keywords	genres
0	Toy Story	[Tom Hanks, Tim Allen, Don Rickles]	John Lasseter	[jealousy, toy, boy]	[animation, comedy, family]
1	Jumanji	[Robin Williams, Jonathan Hyde, Kirsten Dunst]	Joe Johnston	[board game, disappearance, based on children'...	[adventure, fantasy, family]
2	Grumpier Old Men	[Walter Matthau, Jack Lemmon, Ann-Margret]	Howard Deutch	[fishing, best friend, duringcreditsstinger]	[romance, comedy]
3	Waiting to Exhale	[Whitney Houston, Angela Bassett, Loretta Devine]	Forest Whitaker	[based on novel, interracial relationship, sin...	[comedy, drama, romance]
4	Father of the Bride Part II	[Steve Martin, Diane Keaton, Martin Short]	Charles Shyer	[baby, midlife crisis, confidence]	[comedy]

In [23]:
```
# Function to sanitize data to prevent ambiguity. It removes spaces and converts to lowercase
def sanitize(x):
    if isinstance(x, list):
        #Strip spaces and convert to lowercase
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

In [24]:
```
#Apply the generate_list function to cast, keywords, director and genres
for feature in ['cast', 'director', 'genres', 'keywords']:
    df[feature] = df[feature].apply(sanitize)
```

In [25]:
```
#Function that creates a soup out of the desired metadata
def create_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])
```

In [26]:
```
# Create the new soup feature
df['soup'] = df.apply(create_soup, axis=1)
```

In [27]:
```
#Display the soup of the first movie
df.iloc[0]['soup']
```

Out[27]:
```
'jealousy toy boy tomhanks timallen donrickles johnlasseter animation comedy family'
```

In [28]:
```
# Import CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer

#Define a new CountVectorizer object and create vectors for the soup
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df['soup'])
```

In [29]:
```
#Import cosine_similarity function
from sklearn.metrics.pairwise import cosine_similarity

#Compute the cosine similarity score (equivalent to dot product for tf-idf vectors)
cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

In [34]:
```
# Reset index of your df and construct reverse mapping again
df = df.reset_index()
indices2 = pd.Series(df.index, index=df['title'])
```

In [37]:
```
content_recommender('The Lion King', cosine_sim2, df, indices2)
```

Out[37]:
```
29607                                      Cheburashka
40904                  VeggieTales: Josh and the Big Wall
40913    VeggieTales: Minnesota Cuke and the Search for...
27768                                 The Little Matchgirl
15209             Spiderman: The Ultimate Villain Showdown
16613                           Cirque du Soleil: Varekai
24654                                 The Seventh Brother
29198                                      Superstar Goofy
30244                                              My Love
31179               Pokémon: Arceus and the Jewel of Life
Name: title, dtype: object
```

In [ ]: