<u>Evacuative summary</u>

In this assignment, I use Python and Python libraries with Jupyter Notebooks to solve the two movie database problems: ‘Top Five’ and ‘Genre Search’. To solve ‘Top Five’, I preprocessed the movie database to calculate totals and averages, then plugged these values into a Bayesian estimate formula to calculate popularities for movies. Datasets were filtered by a desired genre, and the top 5 movies of that category were extracted from a descending order dataframe.

To solve ‘Genre Search’, I utilized a machine learning model for multi-label binary variance using logistic regression. I extracted keywords to create a string and extracted genres to create a list of categories. This creates a sentiment/opinion analysis data mining scenario. I trained the models on this processed data, then measured its performance on test data. A high score is indicative of a strong relationship between the keywords and genres variables.

The resulting score for ‘Genre Search’ was ~0.3777. This is the result from a threshold of 0.5 confidence for the logistic regression. By lowering the threshold to 0.25 we see a jump in performance to ~0.5828. This score is substantial enough to conclude that there is a relationship between keywords and genres, considering the large number of unique terms to be connected.

Weka is another useful tool for data mining and text analysis that provides a visually intuitive experience. Its tools could also be used for this assessment to great effect. If a genre predictor machine learning model were desired, ‘Genre Search’ could be expanded upon by applying various other models such as Support Vector Machines to compare performance.

Task 1:

-Any assumptions you are making about the scenario task;

I am using Jupyter Notebooks to run python scripts in a presentable format. I assume that python and the libraries pandas, numpy, and sklearn contain methods and tools that are tested and accurate. I assume that documentation and support for these tools and methods are up to date and reflective of their actual use.

For both problems, I make the assumption that the datasets contain accurate information with the exception of NaN values. Entries with NaN values are dropped for consistency and accuracy. Entries with NaN values are considered irrelevant to the task. I assume that the feature 'tmdbId' in the 'links.csv' database corresponds with 'id' in 'metadata.csv' and the feature 'movieId' in 'links.csv' connects with 'movieId' in 'ratings.csv'.

Although 'ratings_small.csv' and 'links_small.csv' are assumed to be smaller samples of their regular-sized csv counterparts, for testing I used the parameter 'nrows=3000' for the '.read_csv()' function while loading the csv files as I worked. I did not rely on the 'small' csv files in order to be consistent with test data collected from 'metadata.csv'.

Finally, I assume that individual votes in 'metadata.csv' carry the same weight as user ratings in 'ratings.csv'. When counting totals and averages, ratings are converted into a score out of 10 from a score out of 5 in order to merge these bodies. I don't factor in potential biases or tendencies for scores in a 5 point vs 10 point system. I also assume that users aren't entering multiple ratings for the same movie and that doing so wouldn't affect the weight of each rating submitted.

-Any pre-processing you would undertake to make the data fit for purpose;

I begin by importing the appropriate libraries. Then, csv files containing data are loaded into pandas dataframes. Unused features such as ‘timestamp’ and ‘userId’ are dropped before performing operations to save time and memory. Rows with na values are dropped. Columns such as ‘tmdbId’ in the links dataframe are renamed for merge operations with relevant dataframes. Id features in links and metadata are converted from string to int.

Means and counts for votes are calculated for the ratings dataframe using the ‘.groupby()’ method. After doing so, indexes are reset to clean up and make dataframes workable. Ratings in the ratings dataframe are multiplied by 2 in order to match 5 point ratings with 10 point votes from metadata. Any duplicate values up to this point are dropped.

Ratings, links, and metadata are merged into a comprehensive movies dataframe. It is at this point where rows are filtered by the desired genre for the Top 5 problem. Na values are once again dropped. Vote averages and rating averages are combined and calculated while counts are combined. An accumulative score is calculated while finding converged averages for use in calculating a mean vote across the whole report. The dataframes are now ready for finding popularity scores.

For the Genre Search problem, I begin by importing testing and model libraries from sklearn. A train test set splitter, a vectorizer for parameterizing text, a one hot encoder, a logistic regression model, a binary relevance model for multi-label data sets, and a performance measurement tool, f1_score, will be used.

Data from the previous problem are used to save memory and time. Vote average and vote counts are dropped from the preprocessed metadata dataframe. Na values are dropped from keywords and ‘id’ is cast to type int. Genres are extracted from metadata and put into a column of lists. Keywords are extracted from keywords and put into a string simulating a document for document search algorithms. Both the original genres and keywords columns are dictionaries in string format so the values must be converted into a dictionary from which values can be iterated over and extracted. The columns from which data was extracted are dropped. Finally, keywords and metadata are merged on ‘id’.

-Which data mining techniques you would apply and why;

To find the top 5 movies of a particular genre, I applied a Bayesian estimate formula to determine the rating/popularity of movies within the specified genre. For the genre search problem, I use an algorithm similar to opinion mining and sentiment analysis for finding correlations between genres and keywords. Opinion or sentiment can be substituted with ‘genre’ and the collection of keywords can be treated as a string. The beauty of using this technique on this particular set of data is that the keywords are already preprocessed, cleaned text. Stopwords aren’t or are rarely present as these collections of words aren’t sentences.

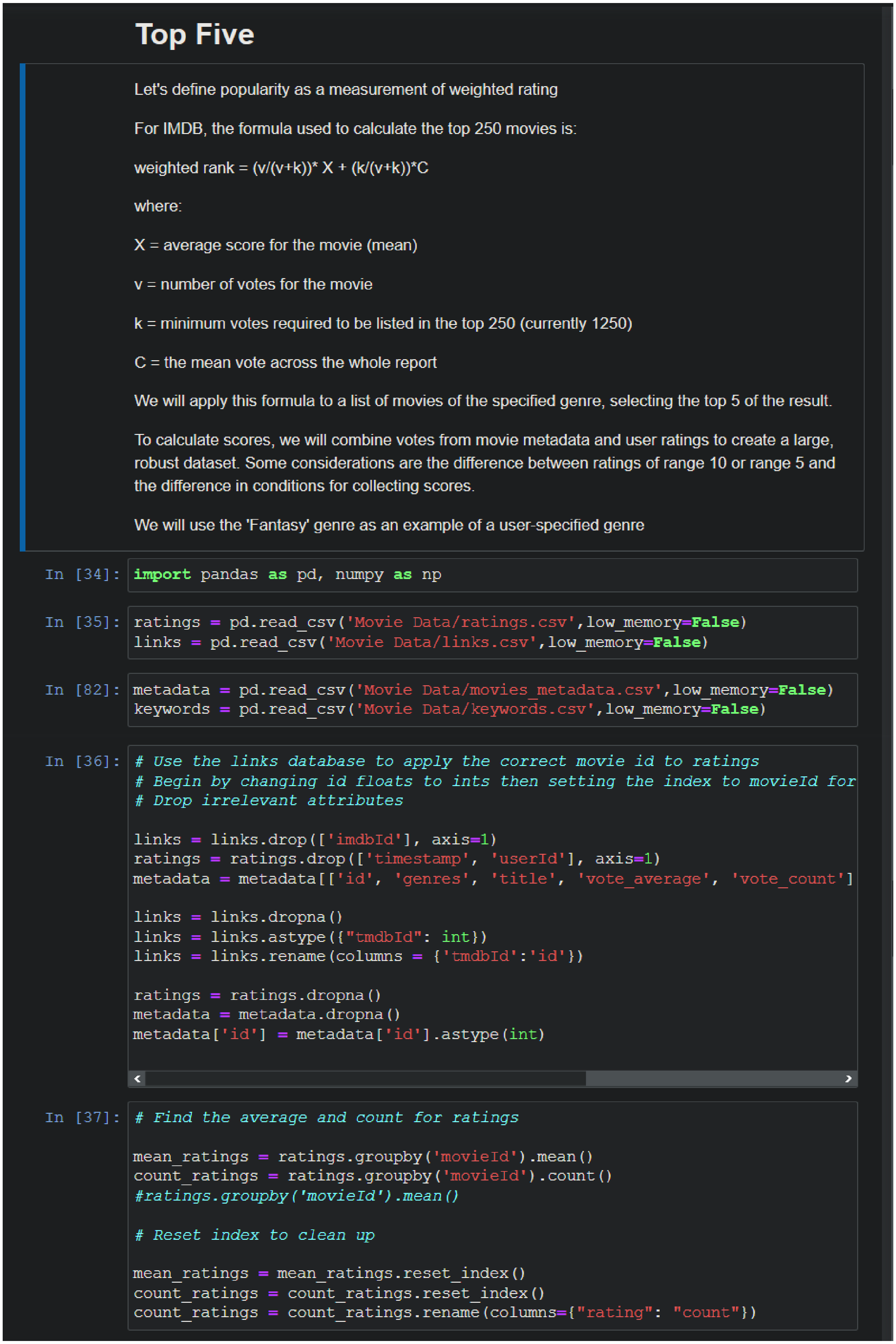
The problem for Genre Search is to ascertain if there is a strong relationship between keywords and genres. If a machine learning model can be trained to predict a movie’s genres based on a corpus of keywords, then that means there is a strong relationship. To this end, I trained a machine learning model with movie data containing keywords and genres, then measured its accuracy to see if a relationship could be established.

I use one-hot encoding to create trainable data from the collection of terms. This is a multi-label classification problem which we solve with Binary Relevance (Zhang). Each label or genre is treated as a binary learning task for each input or movie. Multiple logistic regression models are trained with each movie genre using keywords as input.

-An evaluation of the techniques applied in terms of the accuracy of their results.

After fitting, training, and validating the model, a prediction performance test produces an f1 score of ~0.3777 accuracy. This is the result from a threshold of 0.5 confidence for logistic regression. By lowering the threshold to 0.25 we see a jump in performance to ~0.5828. Prediction with this level of accuracy establishes that there is a relationship between the two variables.

Binary relevance multi-label learning is a straightforward and practical approach to machine learning that produces the results needed for this problem. This simple adaptation of sentiment analysis works well with a text set that has been properly preprocessed to only contain relevant keywords.



Top Five

Let's define popularity as a measurement of weighted rating

For IMDB, the formula used to calculate the top 250 movies is:

weighted rank = (v/(v+k))* X + (k/(v+k))*C

where:

X = average score for the movie (mean)

v = number of votes for the movie

k = minimum votes required to be listed in the top 250 (currently 1250)

C = the mean vote across the whole report

We will apply this formula to a list of movies of the specified genre, selecting the top 5 of the result.

To calculate scores, we will combine votes from movie metadata and user ratings to create a large, robust dataset. Some considerations are the difference between ratings of range 10 or range 5 and the difference in conditions for collecting scores.

We will use the 'Fantasy' genre as an example of a user-specified genre

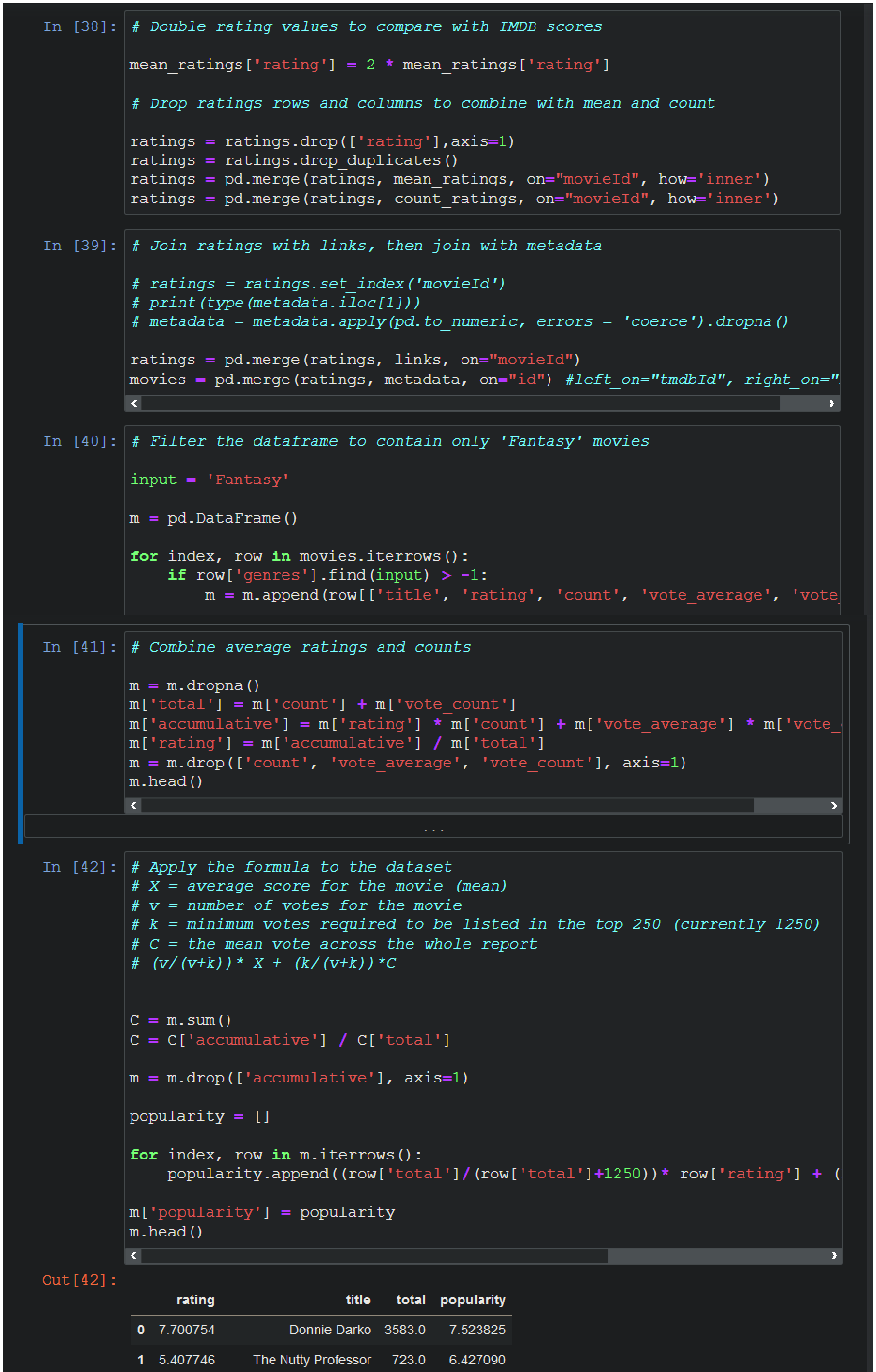
In [34]:
```
import pandas as pd, numpy as np
```

In [35]:
```
ratings = pd.read_csv('Movie Data/ratings.csv',low_memory=False)
links = pd.read_csv('Movie Data/links.csv',low_memory=False)
```

In [82]:
```
metadata = pd.read_csv('Movie Data/movies_metadata.csv',low_memory=False)
keywords = pd.read_csv('Movie Data/keywords.csv',low_memory=False)
```

In [36]:
```
# Use the links database to apply the correct movie id to ratings
# Begin by changing id floats to ints then setting the index to movieId for
# Drop irrelevant attributes

links = links.drop(['imdbId'], axis=1)
ratings = ratings.drop(['timestamp', 'userId'], axis=1)
metadata = metadata[['id', 'genres', 'title', 'vote_average', 'vote_count']

links = links.dropna()
links = links.astype({"tmdbId": int})
links = links.rename(columns = {'tmdbId':'id'})

ratings = ratings.dropna()
metadata = metadata.dropna()
metadata['id'] = metadata['id'].astype(int)
```

In [37]:
```
# Find the average and count for ratings

mean_ratings = ratings.groupby('movieId').mean()
count_ratings = ratings.groupby('movieId').count()
#ratings.groupby('movieId').mean()

# Reset index to clean up

mean_ratings = mean_ratings.reset_index()
count_ratings = count_ratings.reset_index()
count_ratings = count_ratings.rename(columns={"rating": "count"})
```



```
In [38]: # Double rating values to compare with IMDB scores

         mean_ratings['rating'] = 2 * mean_ratings['rating']

         # Drop ratings rows and columns to combine with mean and count

         ratings = ratings.drop(['rating'],axis=1)
         ratings = ratings.drop_duplicates()
         ratings = pd.merge(ratings, mean_ratings, on="movieId", how='inner')
         ratings = pd.merge(ratings, count_ratings, on="movieId", how='inner')
```

```
In [39]: # Join ratings with links, then join with metadata

         # ratings = ratings.set_index('movieId')
         # print(type(metadata.iloc[1]))
         # metadata = metadata.apply(pd.to_numeric, errors = 'coerce').dropna()

         ratings = pd.merge(ratings, links, on="movieId")
         movies = pd.merge(ratings, metadata, on="id") #left_on="tmdbId", right_on="
```

```
In [40]: # Filter the dataframe to contain only 'Fantasy' movies

         input = 'Fantasy'

         m = pd.DataFrame()

         for index, row in movies.iterrows():
             if row['genres'].find(input) > -1:
                 m = m.append(row[['title', 'rating', 'count', 'vote_average', 'vote_
```

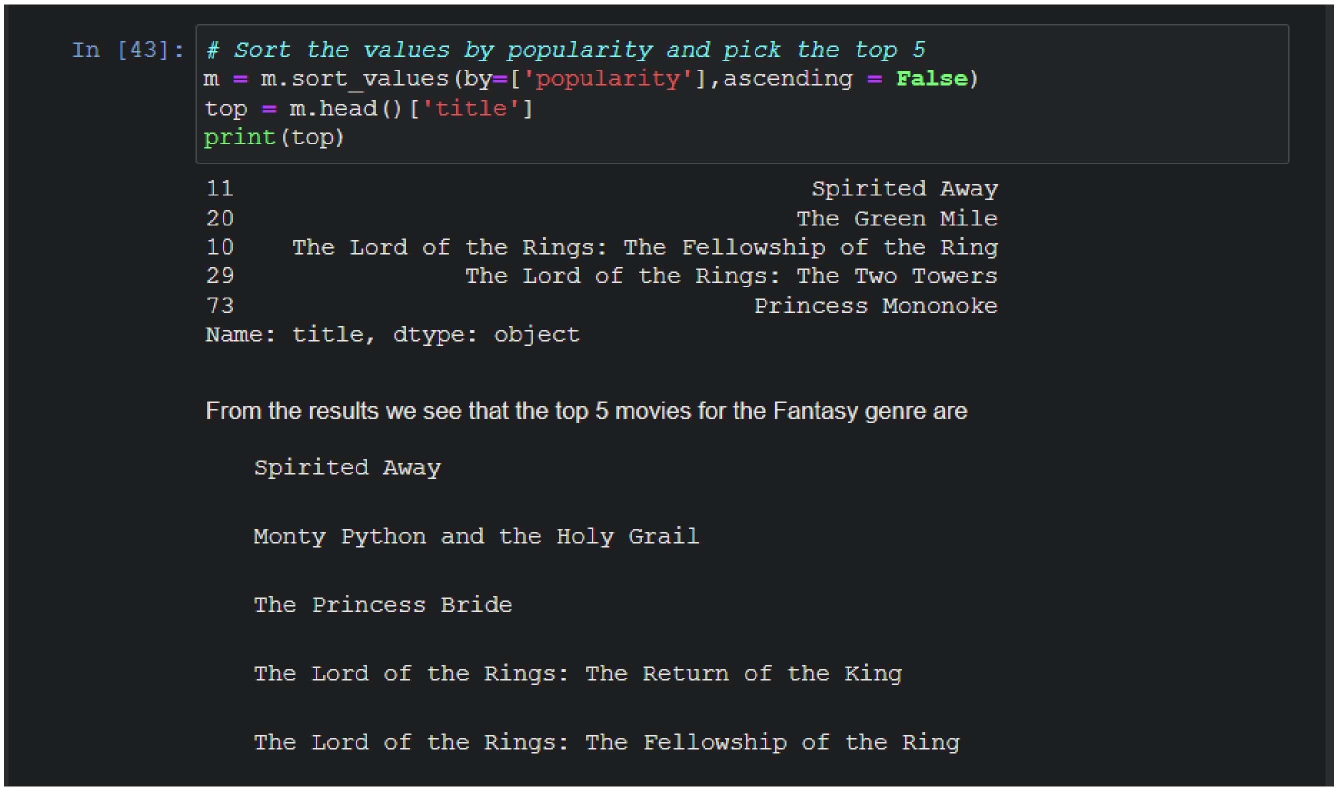
```
In [41]: # Combine average ratings and counts

         m = m.dropna()
         m['total'] = m['count'] + m['vote_count']
         m['accumulative'] = m['rating'] * m['count'] + m['vote_average'] * m['vote_
         m['rating'] = m['accumulative'] / m['total']
         m = m.drop(['count', 'vote_average', 'vote_count'], axis=1)
         m.head()
```

...

```
In [42]: # Apply the formula to the dataset
         # X = average score for the movie (mean)
         # v = number of votes for the movie
         # k = minimum votes required to be listed in the top 250 (currently 1250)
         # C = the mean vote across the whole report
         # (v/(v+k))* X + (k/(v+k))*C


         C = m.sum()
         C = C['accumulative'] / C['total']

         m = m.drop(['accumulative'], axis=1)

         popularity = []

         for index, row in m.iterrows():
             popularity.append((row['total']/(row['total']+1250))* row['rating'] + (

         m['popularity'] = popularity
         m.head()
```

Out[42]:

	rating	title	total	popularity
0	7.700754	Donnie Darko	3583.0	7.523825
1	5.407746	The Nutty Professor	723.0	6.427090



```
In [43]: # Sort the values by popularity and pick the top 5
         m = m.sort_values(by=['popularity'],ascending = False)
         top = m.head()['title']
         print(top)
```

```
11                                       Spirited Away
20                                      The Green Mile
10    The Lord of the Rings: The Fellowship of the Ring
29               The Lord of the Rings: The Two Towers
73                                   Princess Mononoke
Name: title, dtype: object
```

From the results we see that the top 5 movies for the Fantasy genre are

```
Spirited Away

Monty Python and the Holy Grail

The Princess Bride

The Lord of the Rings: The Return of the King

The Lord of the Rings: The Fellowship of the Ring
```

Genre Search:

Ascertain if there is a strong relationship between the keywords used to describe a movie, and the genre/s the movie is associated with. You may want to consider this in the context of a search engine and which terms are more ‘specific’ to a genre.

We will create a string of keywords similar to a document and build a logistic regression model to predict genres. We will measure performance to determine whether there is indeed a relationship between keywords and genres

In [83]:
```python
from sklearn.model_selection import train_test_split
# Text frequency inverse document frequency
from sklearn.feature_extraction.text import TfidfVectorizer
# One-hot encoder
from sklearn.preprocessing import MultiLabelBinarizer
# Logistic Regression machine learning model
from sklearn.linear_model import LogisticRegression
# Binary relevance for learning from multi-label examples
from sklearn.multiclass import OneVsRestClassifier
# Performance measurement
from sklearn.metrics import f1_score
```

In [84]:
```python
# Clean dataframes
keywords = keywords.dropna()
metadata = metadata.drop(['vote_average', 'vote_count'], axis=1)

# Convert ids for keywords to int
keywords['id'] = keywords['id'].astype('int')
```

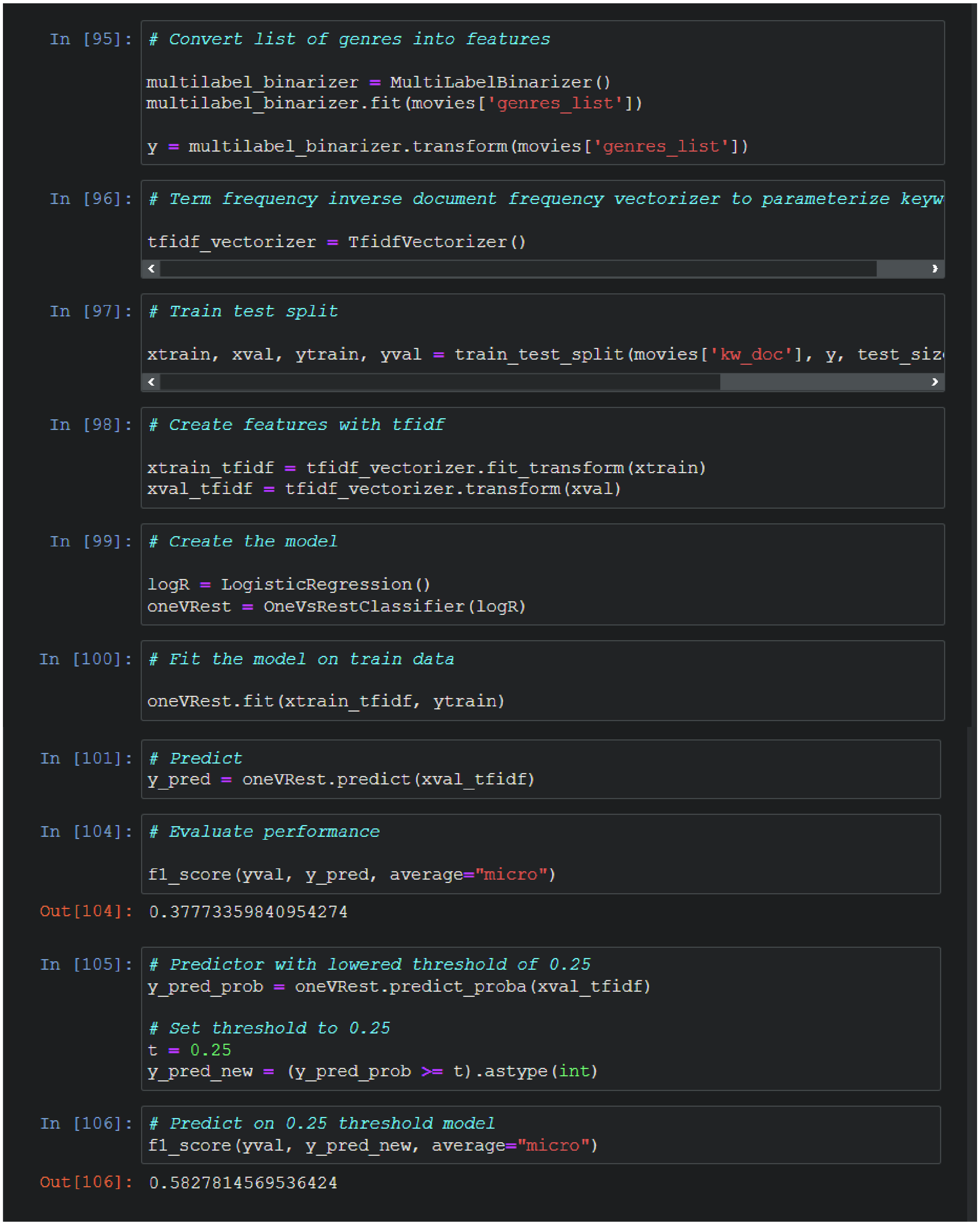
In [88]:
```python
# Extract genres from metadata

genres = []

# extract genres
for i in metadata['genres']:
  genres.append([sub['name'] for sub in eval(i)])

metadata['genres_list'] = genres
metadata = metadata.drop(['genres'], axis=1)
```

In [89]:
```python
# Extract keywords from keywords and append into a string

kw = []

# extract keywords
for i in keywords['keywords']:
  kw.append(' '.join([sub['name'] for sub in eval(i)]))

keywords['kw_doc'] = kw
keywords = keywords.drop(['keywords'], axis=1)
```

In [94]:
```python
# Join the keywords and metadata dataframes

movies = pd.merge(metadata, keywords, on="id", how='inner')
movies.head()
```

Out[94]:

	id	genres_list	kw_doc
0	710	[Adventure, Action, Thriller]	cuba falsely accused secret identity computer ...
1	10634	[Comedy]	rap music parent child relationship rapper job
2	755	[Horror, Action, Thriller, Crime]	dancing brother brother relationship sexual ob...
3	5894	[Comedy]	smoking corner shop cigarette tobacco cigar in...
4	9070	[Action, Adventure, Science Fiction, Family, F...	based on tv series tokusatsu superhero team et...



```
In [95]: # Convert list of genres into features

         multilabel_binarizer = MultiLabelBinarizer()
         multilabel_binarizer.fit(movies['genres_list'])

         y = multilabel_binarizer.transform(movies['genres_list'])

In [96]: # Term frequency inverse document frequency vectorizer to parameterize keyw

         tfidf_vectorizer = TfidfVectorizer()

In [97]: # Train test split

         xtrain, xval, ytrain, yval = train_test_split(movies['kw_doc'], y, test_siz

In [98]: # Create features with tfidf

         xtrain_tfidf = tfidf_vectorizer.fit_transform(xtrain)
         xval_tfidf = tfidf_vectorizer.transform(xval)

In [99]: # Create the model

         logR = LogisticRegression()
         oneVRest = OneVsRestClassifier(logR)

In [100]: # Fit the model on train data

          oneVRest.fit(xtrain_tfidf, ytrain)

In [101]: # Predict
          y_pred = oneVRest.predict(xval_tfidf)

In [104]: # Evaluate performance

          f1_score(yval, y_pred, average="micro")

Out[104]: 0.37773359840954274

In [105]: # Predictor with lowered threshold of 0.25
          y_pred_prob = oneVRest.predict_proba(xval_tfidf)

          # Set threshold to 0.25
          t = 0.25
          y_pred_new = (y_pred_prob >= t).astype(int)

In [106]: # Predict on 0.25 threshold model
          f1_score(yval, y_pred_new, average="micro")

Out[106]: 0.5827814569536424
```

Task 2:

I used Python in Jupyter Notebook. I use pandas, numpy, and scikitlearn libraries for data preprocessing and machine learning. The tools provided were effective for solving the given tasks. Preprocessing tasks such as dropping missing values and unneeded features are made simple with methods such as '.dropna()'. Type conversion such as from string to dictionary was also efficient by using python functions.

Operations on multiple values and instances with dataframes are also straightforward. Python takes operations and applies them to dataframes in the desired fashion. Errors can be diagnosed through searching for online solutions at forums or in official documentation. Jupyter notebooks and python provide presentable error reports that narrow down the cause well enough to quickly find mistakes in code.

Transforming data sets of large amounts of text into a multi-class dataframe for machine learning is made simple with functions for scikitlearn. Splitting a dataset for training, validation, and testing is also straightforward with the train_test_split function. Machine learning models are included as well and can be applied simply. Even a prediction score measuring tool, f1_score(), is included in scikitlearn.

Weka similarly has useful tools for preprocessing data packaged in a nice GUI. Data is visualized with its properties which makes working on data less code-intensive. Weka includes classifiers that could be used for genre prediction such as Naive Bayes and Support Vector Machines. These different models can then be compared side-by-side for simple analysis. Weka is visually comprehensive and useful for working with multiple data sets and comparing/contrasting models. Jupyter's notebook format provides a step-by-step representation that I prefer to understand my work.

Task 3:

Movie recommendation systems tend to trade off between content profiling and collaborative filtering (Sallam). Content profiling pulls recommendations based on an individual user's history. If a user watches many fantasy movies, the recommenders will find movies that match those parameters. A failing of this approach is that no other movies would be recommended unless the profile contains values for the attributes of that movie. This problem is referred to as the "serendipitous problem." Collaborative filtering takes data from other users who have liked similar movies. New users generally have few data points to draw from, resulting in no ways to generate a recommendation. This is referred to as the "cold start problem" (Park). Solutions generally involve hybrid approaches utilizing both. Such algorithms have been optimized to scale effectively for large data sets. I agree that utilizing both approaches would be optimal based on these descriptions. Balancing these systems is a challenge that involves constant testing and must be adapted to fit the data being used

These values are additionally related to coverage, the measure of the domain of items over which the system can make recommendations (Ge). By increasing the coverage, we can increase prediction accuracy and have recommendations be more or less the same. This is inversely related to "serendipity" or the measure of novelty. More serendipity could also confuse users with recommendations unrelated to their interests. 2 solutions are provided in the paper written by Ge, Mouzhi, et all to address this trade off. 1 is to provide explanations as to why an item is recommended with this increased coverage and 2 is to arrange recommendation lists such that serendipitous items appear throughout a list of accurate recommendations to pique interest. I believe that such a recommender provides an enjoyable user experience, but such a subjective measurement would require thorough testing. More UX or psychology experiments should be conducted to ascertain the objectivity of these solutions.

Citations:

Ge, Mouzhi, Carla Delgado-Battenfeld, and Dietmar Jannach. "Beyond accuracy: evaluating recommender systems by coverage and serendipity." *Proceedings of the fourth ACM conference on Recommender systems*. 2010.

Park, Seung-Taek, and Wei Chu. "Pairwise preference regression for cold-start recommendation." *Proceedings of the third ACM conference on Recommender systems*. 2009.

Mrzic, Erol, and Tarik Zaimovic. "Data Science Methods and Machine Learning Algorithm Implementations for Customized Pratical Usage."

Sallam, Rouhia M., Mahmoud Hussein, and Hamdy M. Mousa. "An Enhanced Collaborative Filtering-based Approach for Recommender Systems." *International Journal of Computer Applications* 975: 8887.

Zhang, Min-Ling, et al. "Binary relevance for multi-label learning: an overview." *Frontiers of Computer Science* 12.2 (2018): 191-202.