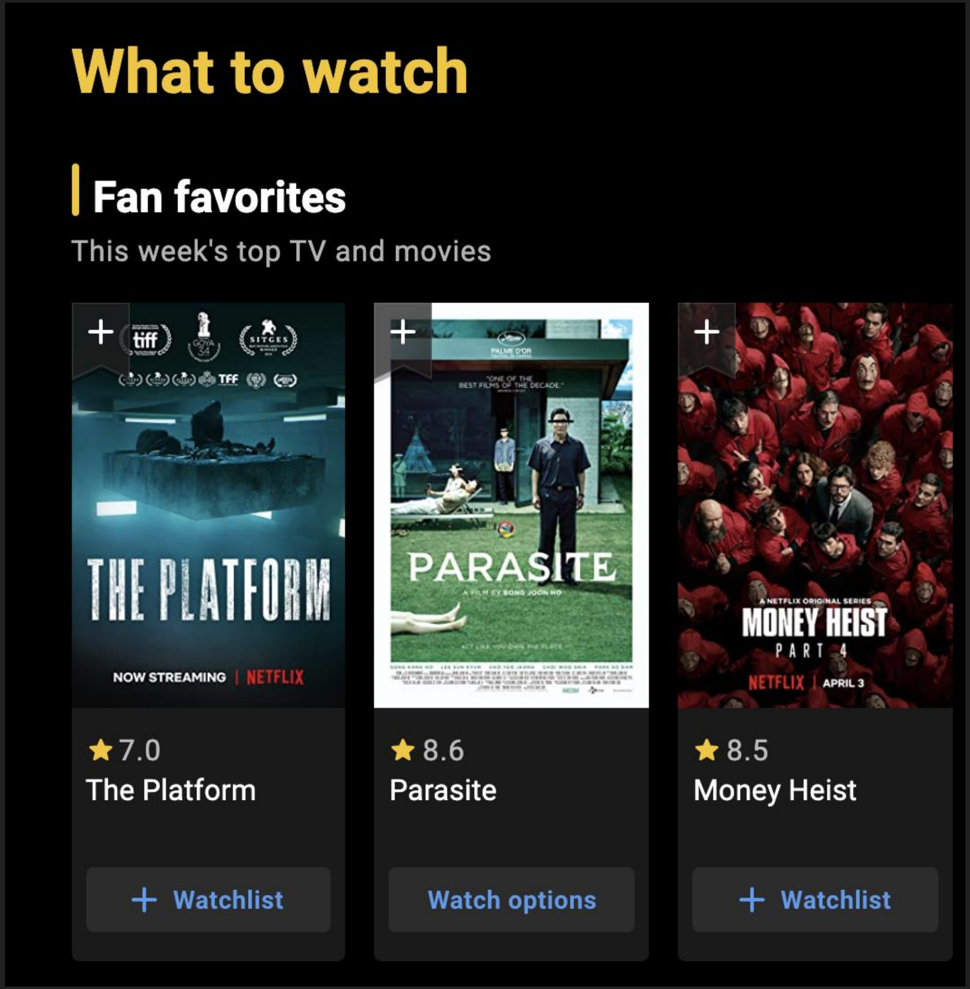
Movie Recommendation System

Marissa F. , Vishal N., Alex Q., Neha T., Abbie A.

Data Description & System Objective

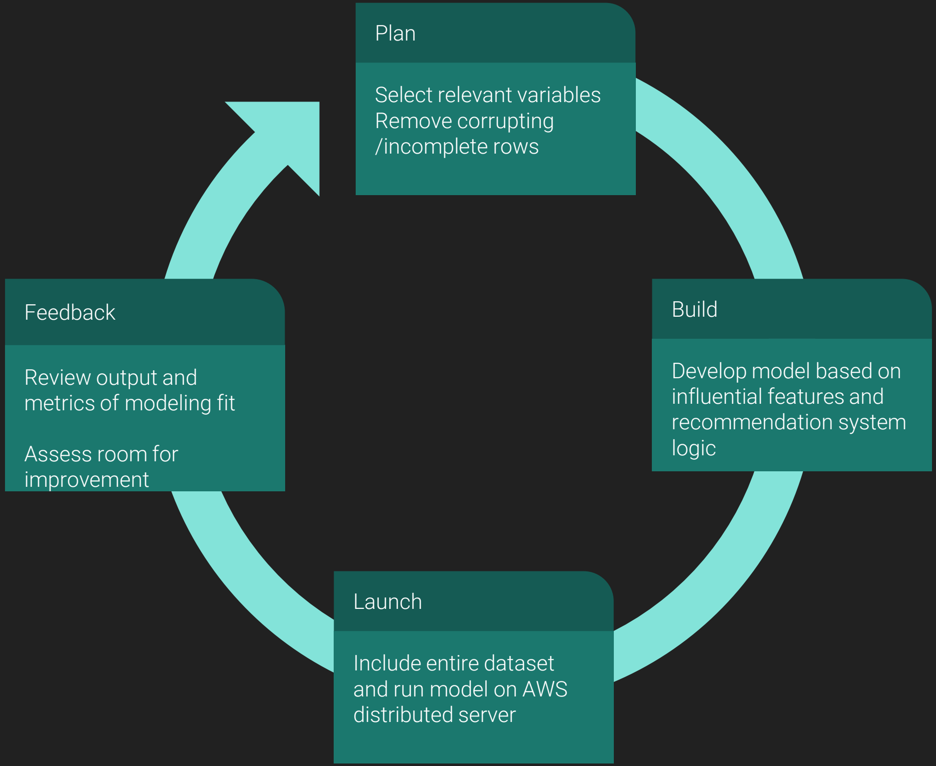
- Dataset: export from IMDb
  - Distributed into tables on movie data and their ratings
  - ~ 20M rows of data
  - Films from 1894 - 2019
- Project objective: ingest large dataset using AWS and provide end users a recommendation
  - Intake selected preferences
  - Recommendation driven by ratings and machine learning.



Requirements

Business Requirements and Attributes				
Business Requirement ID	**Functional/Technical**	**Business Requirement**	**Rationale**	**Source**
1	Technical	Collect IMBD movie data.	This data will be used for predictive analysis.	Kaggle
2	Technical	Merge applicable tables to build single datasource.	This will allow for comprehensive analysis of all data.	Python
3	Functional	Populate single dataset to AWS	Store data in secure and faster environment	AWS
4	Technical	Wrangle data to remove duplicate entries, keeping only original langauge and title.	Remove duplicates that may weaken analysis	Python
5	Functional	Remove incomplete data , data with low number of votes for ratings (dependent variable), and more recent years only (as determined by quality of earlier movie data).	Ensure high quality data is used	Python
6	Functional	Perform basic statistical analysis to understand data.	Understand linkages and relationships between variables.	Python
7	Functional	Use data to make predictive recommendations on movies.	Utilize findings to recommend movies based on various preferences.	Python
8	Technical	Test model	Ensure model works	Python
9	Functional	Deploy model using AWS	Share with others!	AWS
10	Technical	Ensure application should shows horizontal scale	Fast computing speed	AWS

Program Development



Data Preparation

- Combine 4 tables, selecting relevant variables to include in models
- Convert csv into Pyspark dataframe
- Remove redundant entries
  - Redundancies caused by movies translated into various languages
- Delete entries without ratings
  - This is a key variable for modeling
- Export cleaned data into parquet files for consumption

Filtering data in PySpark for original titles, and removing rows with NA, filtering for movies with less than 250 votes (numVotes)

```
import pandas as pd
import numpy as np
```

```
from pyspark.sql.functions import lit, when, col, regexp_extract
```

```
movies_df_1 = movies_df.filter(movies_df["isOriginalTitle"] == '1')
movies_df_1.show()
```

```
+---------+--------+--------------------+------+--------+--------+----------+---------------+---------+---------+--------
|  titleId|ordering|               title|region|language|   types|attributes|isOriginalTitle|   tconst|titleType|
+---------+--------+--------------------+------+--------+--------+----------+---------------+---------+---------+--------
|tt0000001|       5|          Carmencita|    \N|      \N|original|        \N|              1|tt0000001|    short|
|tt0000002|       1|Le clown et ses c...|    \N|      \N|original|        \N|              1|tt0000002|    short|Le clown
|tt0000003|       4|      Pauvre Pierrot|    \N|      \N|original|        \N|              1|tt0000003|    short|      Pa
|tt0000004|       1|         Un bon bock|    \N|      \N|original|        \N|              1|tt0000004|    short|
|tt0000005|       8|     Blacksmith Scene|    \N|      \N|original|        \N|              1|tt0000005|    short|    Blac
|tt0000006|       5|   Chinese Opium Den|    \N|      \N|original|        \N|              1|tt0000006|    short|   Chine
|tt0000007|       2|Corbett and Court...|    \N|      \N|original|        \N|              1|tt0000007|    short|Corbett
|tt0000008|       4|Edison Kinetoscop...|    \N|      \N|original|        \N|              1|tt0000008|    short|Edison K
|tt0000009|       1|          Miss Jerry|    \N|      \N|original|        \N|              1|tt0000009|    movie|
|tt0000010|      11|La sortie de l'us...|    \N|      \N|original|        \N|              1|tt0000010|    short| Exiting
```

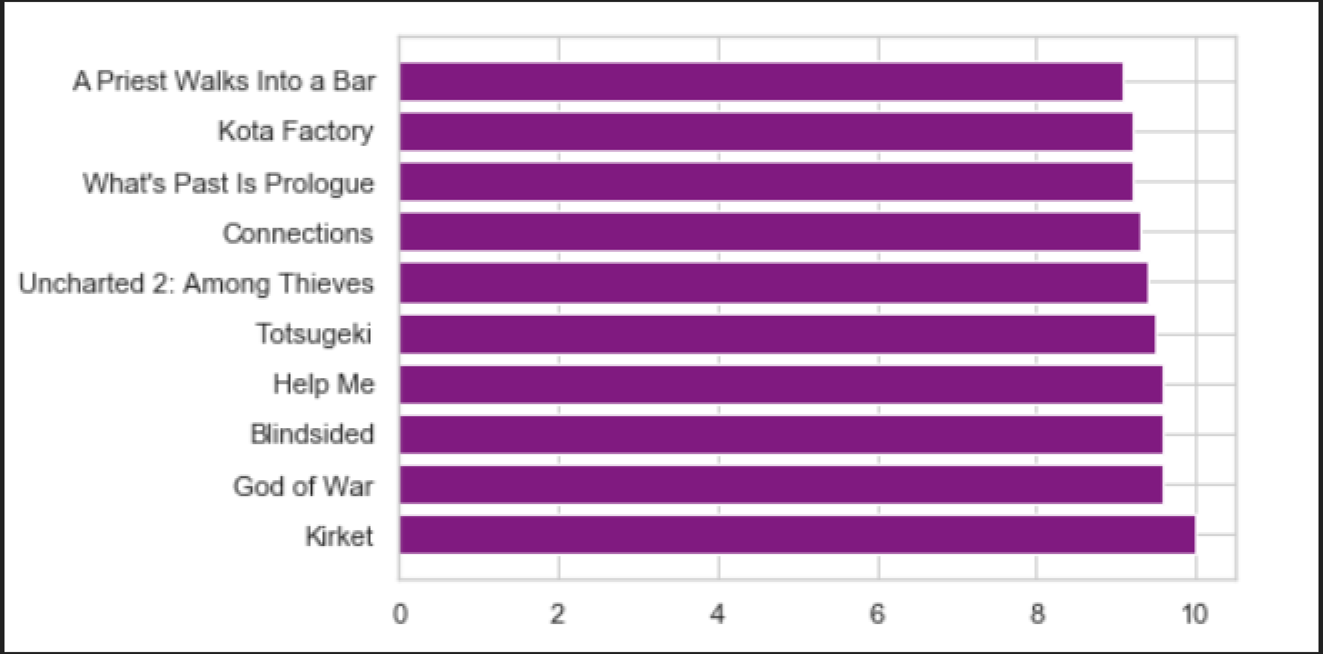
Exporting code in PySpark

1. Parquet export code
2. CSV expport code

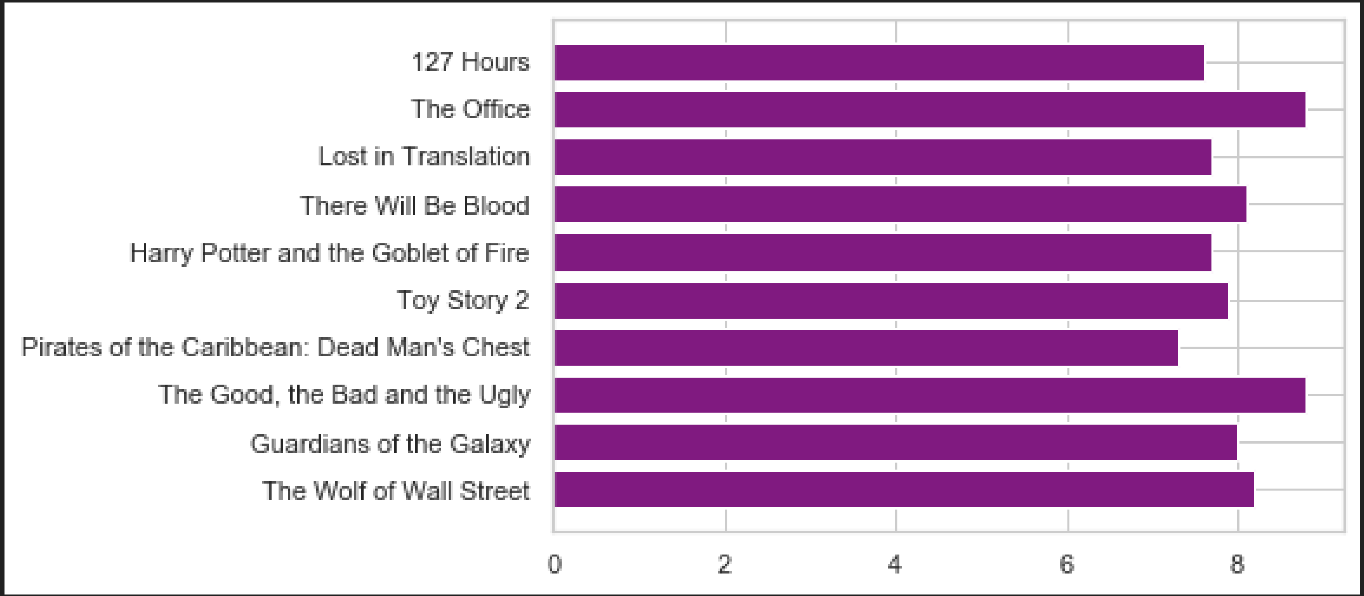
```
#partitioned file
movies_df_2.write.parquet("drive/My Drive/Unstructured Final Project/MoviesDatasetParquet")
#csv
movies_df_2.coalesce(1).write.mode("overwrite")\
.format("com.databricks.spark.csv")\
.option("header", "true")\
.option("sep", "|")\
.save('drive/My Drive/Unstructured Final Project/my_output_csv')
```

Data Exploration & Analytics

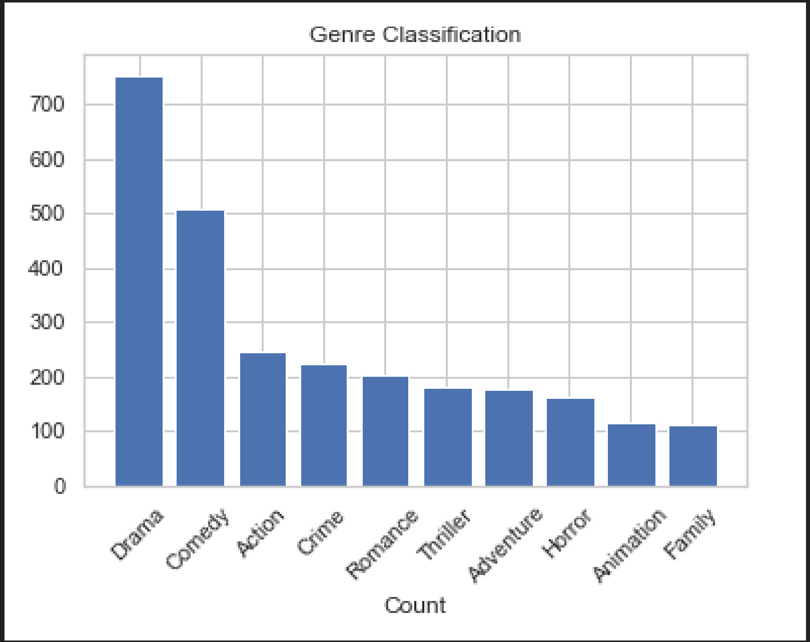
Highest Rated Content on IMDb



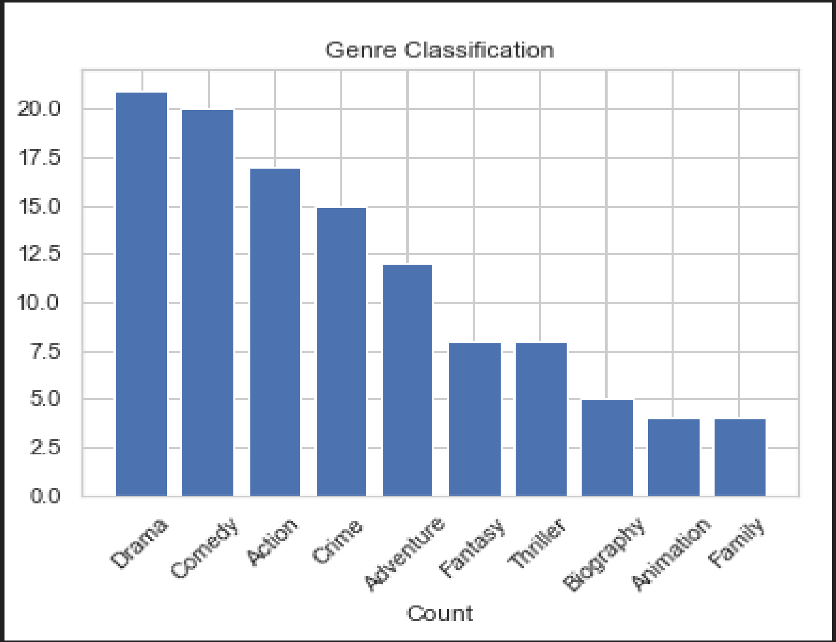
Most Popular Content on IMDb



Available Content



Popular Content



Machine Learning in PySpark to provide a recommendation for successful movies

Predictor Variables:

1. Title Type: TV Series, movie, short film.
2. Number of Votes
3. Year
4. Movie Runtime
5. Genre

label	features	titleType	isAdult	startYear	runtimeMinutes	genres	numVotes	Success
0.0	(1330,[45,1188,13...	short	0.0	1894	1	Documentary,Short	1550.0	0
0.0	(1330,[221,1188,1...	short	0.0	1892	4	Animation,Comedy,...	1207.0	0
0.0	(1330,[22,1188,13...	short	0.0	1893	1	Comedy,Short	1934.0	0
0.0	(1330,[616,1188,1...	short	0.0	1894	1	Short,Sport	615.0	0
0.0	(1330,[45,1188,13...	short	0.0	1894	1	Documentary,Short	1667.0	0
0.0	(1330,[45,1188,13...	short	0.0	1895	1	Documentary,Short	5545.0	0
1.0	(1330,[785,1188,1...	short	0.0	1896	1	Action,Documentar...	9435.0	1
0.0	(1330,[45,1188,13...	short	0.0	1895	1	Documentary,Short	1447.0	0
1.0	(1330,[22,1188,13...	short	0.0	1895	1	Comedy,Short	4111.0	1
0.0	(1330,[101,1188,1...	short	0.0	1894	2	Animation,Short	741.0	0

only showing top 10 rows

Output:

Predicts whether a movie can be classified as a success or not.

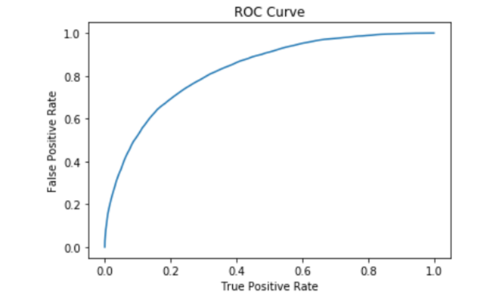
Approach:

1. Set a threshold of an average rating >= 6.4 to determine whether the movie is successful.
2. Encode your feature variables (strings/categorical).
3. Transform your input variables into a vector of features that your model can work with.
4. Fit your model on the training sets and predict it on the test sets.

Model Evaluation & Performance

titleType	numVotes	runTimeMinutes	startYear	genres	Success	prediction	probability
movie	457.0	136	2010	Drama	0	0.0	[0.67225970212625...
movie	323.0	88	2002	Drama	0	0.0	[0.74367873307500...
movie	430.0	79	2001	Drama	0	0.0	[0.72900975776502...
movie	2855.0	93	2001	Drama	0	0.0	[0.70419785400695...
movie	2577.0	97	2001	Drama	0	0.0	[0.69970649230874...
movie	337.0	108	2001	Drama	0	0.0	[0.69153701751024...
movie	888.0	116	2001	Drama	0	0.0	[0.67896847572049...
movie	493.0	89	2000	Drama	0	0.0	[0.71222581254970...
movie	2176.0	91	2000	Drama	0	0.0	[0.70479379287038...
movie	748.0	92	2000	Drama	0	0.0	[0.70758704334167...
movie	258.0	97	2000	Drama	0	0.0	[0.70241694037393...
movie	887.0	98	2000	Drama	0	0.0	[0.69926183413203...
movie	2048.0	98	2000	Drama	0	0.0	[0.69586695669028...
movie	269.0	99	2000	Drama	0	0.0	[0.69973173436651...
movie	2857.0	103	2000	Drama	0	0.0	[0.68671986477920...
movie	541.0	105	2000	Drama	0	0.0	[0.69088690434452...
movie	300.0	108	2000	Drama	0	0.0	[0.68753460196145...
movie	1014.0	115	2000	Drama	0	0.0	[0.67577226830311...
movie	307.0	139	2000	Drama	0	0.0	[0.64388864504795...
movie	302.0	55	1999	Drama	0	0.0	[0.77131689963880...
movie	1245.0	84	1999	Drama	0	0.0	[0.73479982404407...
movie	712.0	89	1999	Drama	0	0.0	[0.73004469720506...
movie	275.0	90	1999	Drama	0	0.0	[0.72999083882833...
movie	564.0	90	1999	Drama	0	0.0	[0.72920057599836...
movie	831.0	90	1999	Drama	0	0.0	[0.72846918075243...
movie	289.0	92	1999	Drama	0	0.0	[0.72744965878850...

True/Predicted	Positive	Negative	Total
Positive	6,341	3,207	16,451
Negative	2,594	10,110	5,801
Total	**8,935**	**13,317**	**22,252**



Training Data: Area Under ROC: 0.8303336131547687

Metrics	
Sensitivity/Recall	66.4%
Specificity	79.6%
Precision	71.0%
Accuracy	73.9%

- Preferred model based on results: Logistic Regression from MLib in PySpark
- The outcome probabilities are shown for the combination of features in our data.
- The model has scope for improvement, either through the addition of more relevant predictor variables and additional data.

Recommendation System

- Based on weighted ratings
  - (v/(v+m) * R) + (m/(m+v) * C)
  - v = number of votes for the movie
  - m = minimum number of votes to be considered (in Python, in the 10th percentile. In PySpark, over 100 votes)
  - R = Average Rating of the movie
  - C = Average Rating across all movies
- Three user inputs
  - Genre of choice
  - Type (i.e. movie, TV show, videogame)
  - Age

Built in Python with a For Loop and Function

```
for i in range(0,len(movies)) :
  v = movies.numVotes[i]
  m = np.percentile(movies.averageRating, q = .10)
  C = movies['averageRating'].mean()
  R = movies.averageRating[i]
  movies.WeightedRating[i] = (v/(v+m) * R) + (m/(m+v) * C)
```

```
def recommend(genrechoice, typechoice, agechoice = 0):
  PossMovie = movies[movies['genres'].str.contains(genrechoice)]
  PossMovie = PossMovie[PossMovie['titleType'].str.contains(typechoice)]
  if (agechoice == "Old" or agechoice == 'old' or agechoice == "OLD"):
    PossMovie = PossMovie[PossMovie.startYear <= 2000]
  if (agechoice == "New" or agechoice == 'new' or agechoice == "NEW"):
    PossMovie = PossMovie[PossMovie.startYear > 2000]
  else:
    PossMovie = PossMovie
  Ordered = PossMovie.sort_values(by="WeightedRating", ascending = False)
  Ordered = Ordered[['title', 'genres', 'titleType', 'startYear', 'WeightedRating']]
  Top5 = Ordered[:5]
  return Top5
```

```python
recommend('Comedy', 'movie', 'New')
```

	title	genres	titleType	startYear	WeightedRating
45122	Shibu	Comedy	movie	2019	9.391544
59808	CM101MMXI Fundamentals	Comedy,Documentary	movie	2013	9.199900
67251	The Weight of Chains 2	Comedy,Documentary,History	movie	2014	8.996412
75873	10 Days Before the Wedding	Comedy,Drama,Musical	movie	2018	8.890196
34554	Anbe Sivam	Adventure,Comedy,Drama	movie	2003	8.799735

Built in PySpark with a UDF and spark.sql

```
def weighted(v, m, C, R) :
  return (v/(v+m) * R) + (m/(m+v) * C)
```

```
weighted_udf = udf(weighted, FloatType())
```

```
WeightedRating = weighted_udf(movies.numVotes, movies.MinimumVotes, movies.AverageTotalRating, movi
WeightedRating
```

```
Column<b'weighted(numVotes, MinimumVotes, AverageTotalRating, averageRating)'>
```

```
movies = movies.withColumn("WeightedRating", WeightedRating.cast(FloatType()))
```

```
movies = movies.select('title','titleType','startYear','genres','averageRating','WeightedRating')
movies.show(10)
```

```
movies.createOrReplaceTempView("movies_sql")
```

```
recommendations_sql = spark.sql('''
  SELECT title, titleType, startYear, genres, WeightedRating
  FROM movies_sql
  WHERE genres LIKE '%Comedy%' AND titleType = 'movie' AND (startYear > 2000)
  ORDER BY WeightedRating desc
  LIMIT (5)
''')
```

```
recommendations_sql.show()
```

```
+--------------------+---------+---------+--------------------+--------------+
|               title|titleType|startYear|              genres|WeightedRating|
+--------------------+---------+---------+--------------------+--------------+
|CM101MMXI Fundame...|    movie|   2013.0|  Comedy,Documentary|      9.193336|
|               Shibu|    movie|   2019.0|              Comedy|      8.924001|
|          Anbe Sivam|    movie|   2003.0|Adventure,Comedy,...|      8.782449|
|The Weight of Cha...|    movie|   2014.0|Comedy,Documentar...|      8.780598|
|Nuvvu Naaku Nachchav|    movie|   2001.0|Comedy,Family,Mus...|      8.628696|
+--------------------+---------+---------+--------------------+--------------+
```

Findings & Conclusions

- Number of people voting on a movie is the most influential factor in a high scoring film
    - Popular films are perceived as good films
- The most popular movie genre is drama
- Deep categorical variables such as directors and actors could add value to the model via creating dummy columns for important figures in each area.
- Further Improvements for Deployment
    - Serialize the models and use a Flask API to generate a usable web app. A user could visit the website, select a genre, type, and the age of the film. The user would see the movie recommendations given by the model.