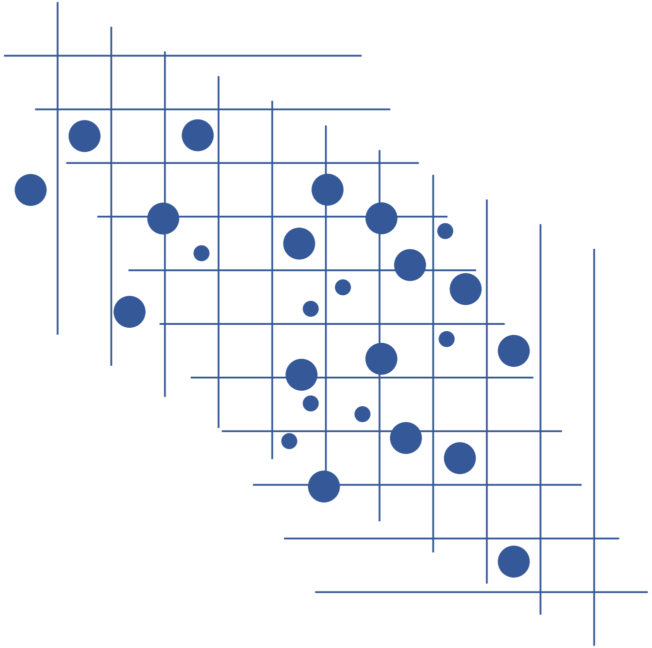
# Movie Recommendation System using MovieLens Data

Brian Lowell



# **Problem**: identifying movies that users will like is challenging

*Hypothesis*: If a user likes Movie A, they will also like movies similar to Movie A based on taggings and genres.

***Solution***: Develop a content-based recommender tool to identify similar movies.

*Hypothesis*: A movie's rating will be correlated with its features.

***Solution***: Use regression to predict ratings based on movie features.

*Hypothesis*: A movie can be recommended based on the preferences of similar users.

***Solution***: Use matrix factorization and SVD algorithm to predict ratings based on user ratings.

# MovieLens 20M Dataset

- MovieLens is run by GroupLens, a research lab at the University of Minnesota. By using MovieLens, you will help GroupLens develop new experimental tools and interfaces for data exploration and recommendation. MovieLens is non-commercial, and free of advertisements.
- The tag genome encodes how strongly movies exhibit particular properties represented by tags (atmospheric, thought-provoking, realistic, etc.). The tag genome was computed using a machine learning algorithm on user-contributed content including tags, ratings, and textual reviews.
- **genome_scores**: relevancy of a given tag to a movie
- **genome_tags**: tag descriptions and tag_ids
- **links**: includes IMDb and TMDB ids to bring in other data
- **movies**: includes movie id, title (year), and genre tags
- **ratings**: user reviews out of 5
- **tags**: user-generated tags, including user_id, movie_id and tag

Sources: Kaggle, MovieLens.com, Surprise package

**movies**

| movieId | title | genres |
|---|---|---|
| 1 | Toy Story (1995) | Adventure\|Animation\|Children\|Comedy\|Fantasy |
| 2 | Jumanji (1995) | Adventure\|Children\|Fantasy |
| 3 | Grumpier Old Men (1995) | Comedy\|Romance |
| 4 | Waiting to Exhale (1995) | Comedy\|Drama\|Romance |
| 5 | Father of the Bride Part II (1995) | Comedy |

**genome_tags**

| tagId | tag |
|---|---|
| 1 | 007 |
| 2 | 007 (series) |
| 3 | 18th century |
| 4 | 1920s |
| 5 | 1930s |

**ratings**

| userId | movieId | rating | timestamp |
|---|---|---|---|
| 1 | 1 | 4.0 | 964982703 |
| 1 | 3 | 4.0 | 964981247 |
| 1 | 6 | 4.0 | 964982224 |
| 1 | 47 | 5.0 | 964983815 |
| 1 | 50 | 5.0 | 964982931 |

**genome_scores**

| movieId | tagId | relevance |
|---|---|---|
| 1 | 1 | 0.02500 |
| 1 | 2 | 0.02500 |
| 1 | 3 | 0.05775 |
| 1 | 4 | 0.09675 |
| 1 | 5 | 0.14675 |

**tags**

| userId | movieId | tag | timestamp |
|---|---|---|---|
| 2 | 60756 | funny | 1445714994 |
| 2 | 60756 | Highly quotable | 1445714996 |
| 2 | 60756 | will ferrell | 1445714992 |
| 2 | 89774 | Boxing story | 1445715207 |
| 2 | 89774 | MMA | 1445715200 |

# Key Data Challenges & Pre-processing

Problem: genres are pipe-delimited in a single string. They will need to be parsed to use them in a feature matrix.

| movieId | title | genres |
|---|---|---|
| 1 | Toy Story (1995) | Adventure\|Animation\|Children\|Comedy\|Fantasy |

```
cleaned_movies = movies.assign(genres=movies['genres'].str.split('|')).explode('genres')
cleaned_movies['year_released'] = cleaned_movies['title'].str.extract(r'\((\d{4})\)')
cleaned_movies['year_released'] = pd.to_numeric(cleaned_movies['year_released'], errors='coerce')
cleaned_movies = links.merge(cleaned_movies, on='movieId', how='outer')
cleaned_movies = cleaned_movies[['imdbId', 'title','year_released', 'genres']]
cleaned_movies.set_index('imdbId', inplace=True)
cleaned_movies.head()

    title   year_released   genres
imdbId
114709  Toy Story (1995)   1995.0  Adventure
114709  Toy Story (1995)   1995.0  Animation
114709  Toy Story (1995)   1995.0  Children
114709  Toy Story (1995)   1995.0  Comedy
114709  Toy Story (1995)   1995.0  Fantasy
```

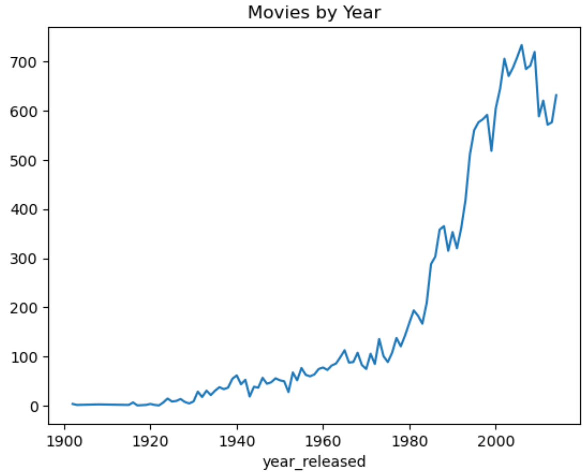
Problem: tags are included as observations, rather than variables. They will need to be pivoted to combine them into a feature matrix.

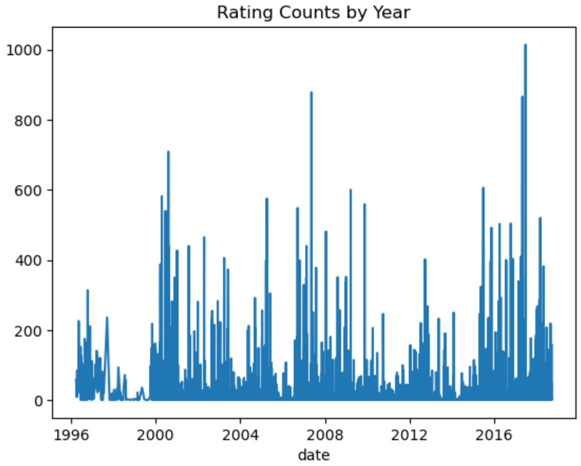
| title | relevance | tag |
|---|---|---|
| Toy Story (1995) | 0.02500 | 007 |
| Jumanji (1995) | 0.03975 | 007 |
| Grumpier Old Men (1995) | 0.04350 | 007 |

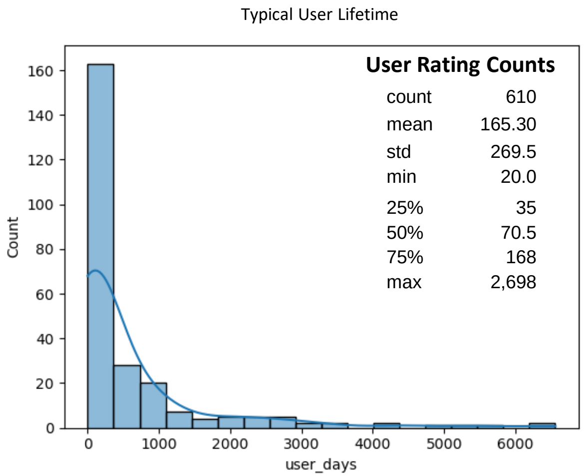
```
pivoted_movie_tags = movie_tags.pivot_table(index='title',
columns='tag', values='relevance')
pivoted_movie_tags.head()
```

| tag | 007 | 007 (series) | 18th century | 1920s | 1930s | 1950s | 1960s | 1970s | 1980s | 19th century | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| title | | | | | | | | | | | |
| '71 (2014) | 0.01350 | 0.01250 | 0.04650 | 0.05525 | 0.11375 | 0.04425 | 0.02600 | 0.35250 | 0.07150 | 0.03175 | ... |
| 'Round Midnight (1986) | 0.01950 | 0.02075 | 0.02100 | 0.12100 | 0.51250 | 0.25750 | 0.41800 | 0.27575 | 0.08675 | 0.01525 | ... |
| 'Til | | | | | | | | | | | |

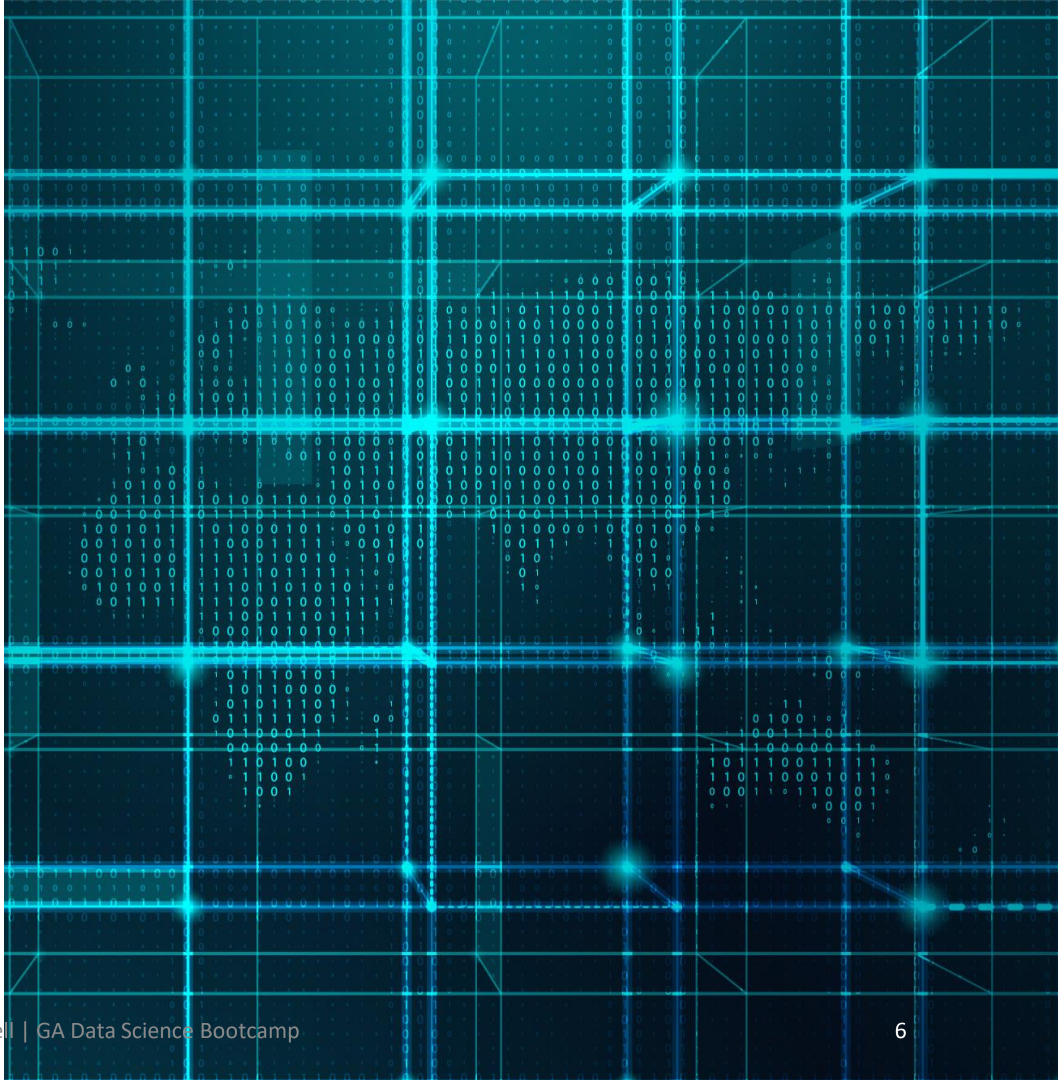
# Exploratory Data Analysis



# Develop a content-based recommender tool to identify similar movies



# Create feature matrix

*one-hot encode genres*

```
one_hot = pd.get_dummies(cleaned_movies['genres'],
dtype=float)
cleaned_movies = cleaned_movies.drop('genres',axis = 1)
cleaned_movies = cleaned_movies.join(one_hot)
cleaned_movies = cleaned_movies.groupby(['title']).max()
```

| | year_released | Action | Adventure | Animation | Children | ... | Thriller | War | Western |
|---|---|---|---|---|---|---|---|---|---|
| **title** | | | | | | | | | |
| **'71 (2014)** | 2014.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 1.0 | 0.0 |
| **'Hellboy': The Seeds of Creation (2004)** | 2004.0 | 1.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 |
| **'Round Midnight (1986)** | 1986.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 |
| **'Salem's Lot (2004)** | 2004.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 0.0 |
| **'Til There Was You (1997)** | 1997.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 |

*merging with scaled tag genome scores*

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(pivoted_movie_tags)
scaled_tags = pd.DataFrame(scaled_features,
index=pivoted_movie_tags.index,
columns=pivoted_movie_tags.columns)

scaled_tags.head()
```

| title | Action | Adventure | ... | wwii | zombies |
|---|---|---|---|---|---|
| **'71 (2014)** | 1.0 | 0.0 | ... | 0.340927 | -0.152156 |
| **'Hellboy': The Seeds of Creation (2004)** | 1.0 | 1.0 | ... | 0.000000 | 0.000000 |
| **'Round Midnight (1986)** | 0.0 | 0.0 | ... | -0.278982 | -0.167342 |
| **'Salem's Lot (2004)** | 0.0 | 0.0 | ... | 0.000000 | 0.000000 |
| **'Til There Was You (1997)** | 0.0 | 0.0 | ... | -0.261763 | -0.161268 |

# Use cosine similarity matrix to identify similar movies

```python
from sklearn.metrics.pairwise import cosine_similarity

# Compute the cosine similarity matrix
similarity_matrix = cosine_similarity(data)
similarity_matrix
```

```python
def recommend_movies(movie_title, num_movies=5):
    # Ensure the movie_title exists in the dataset
    if movie_title not in data.index:
        return "Movie title not found in the dataset."

    # Find the row index of the movie
    movie_idx = list(data.index).index(movie_title)

    # Get similarity values with other movies
    similarity_scores = list(enumerate(similarity_matrix[movie_idx]))

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

    # Get the scores of the top-n most similar movies, excluding the first one (itself)
    most_similar_movies = similarity_scores[1:num_movies+1]

    # Get movie titles for the most similar movies
    movie_titles = [data.index[i[0]] for i in most_similar_movies]
    return movie_titles
```

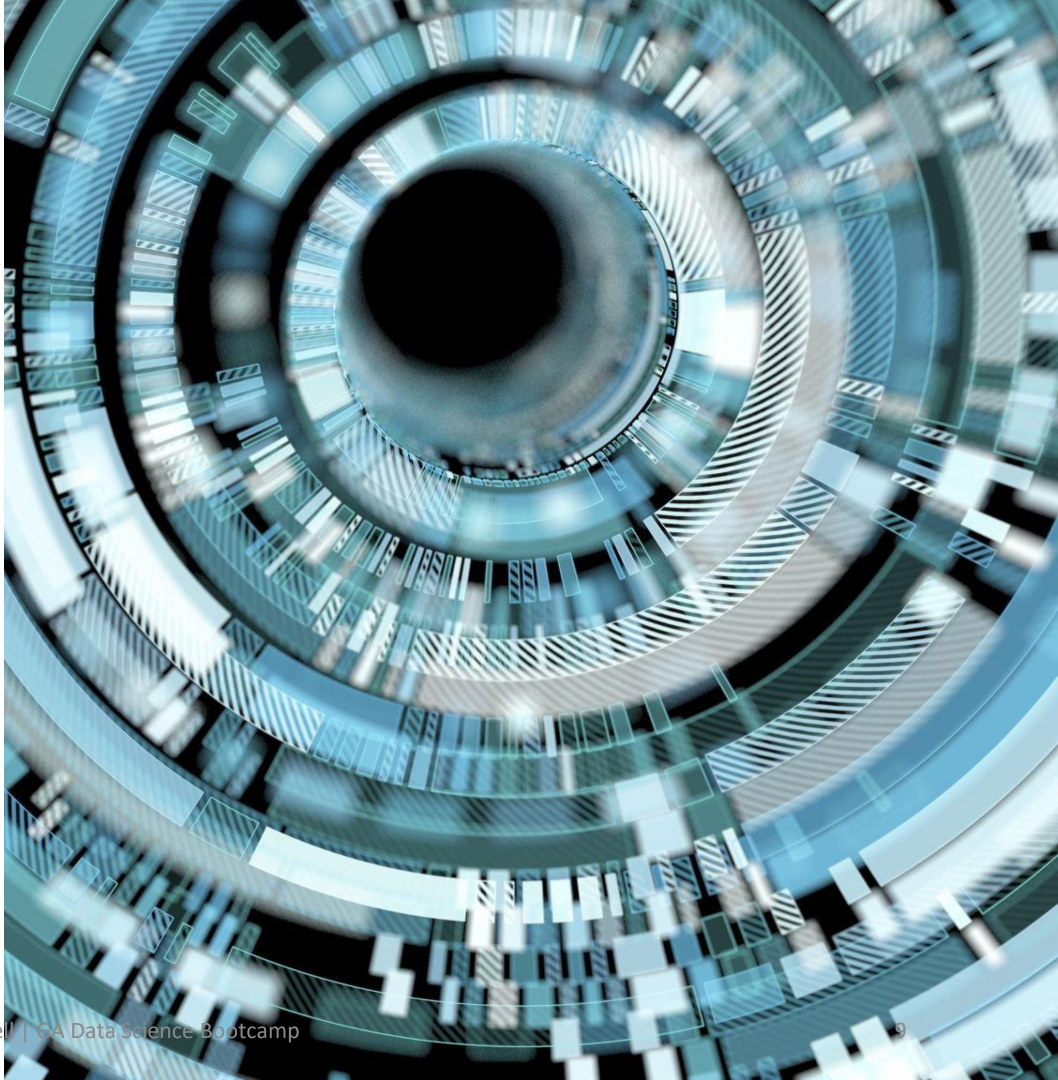
```python
recommend_movies('Toy Story (1995)', 10)
['Toy Story 2 (1999)',
 'Monsters, Inc. (2001)',
 'Toy Story 3 (2010)',
 "Bug's Life, A (1998)",
 'Up (2009)',
 'Ice Age (2002)',
 'Cars (2006)',
 'Incredibles, The (2004)',
 'Finding Nemo (2003)',
 'Monsters University (2013)']
```

```python
recommend_movies('Godfather, The (1972)', 10)
['Godfather: Part II, The (1974)',
 'Goodfellas (1990)',
 'Scarface (1983)',
 'On the Waterfront (1954)',
 'Departed, The (2006)',
 'Road to Perdition (2002)',
 'Raging Bull (1980)',
 'Taxi Driver (1976)',
 'Scarface (1932)',
 'Rocco and His Brothers (Rocco e i suoi fratelli) (1960)']
```

```python
recommend_movies('Lord of the Rings: The Fellowship of the Ring, The (2001)', 10)
['Lord of the Rings: The Two Towers, The (2002)',
 'Lord of the Rings: The Return of the King, The (2003)',
 'Hobbit: An Unexpected Journey, The (2012)',
 'Hobbit: The Desolation of Smaug, The (2013)',
 'The Hobbit: The Battle of the Five Armies (2014)',
 'Star Wars: Episode IV - A New Hope (1977)',
 'Hunt For Gollum, The (2009)',
 'Willow (1988)',
 'Star Wars: Episode V - The Empire Strikes Back (1980)',
 'Star Wars: Episode VI - Return of the Jedi (1983)']
```

```python
recommend_movies('Inception (2010)', 10)
['Memento (2000)',
 'Usual Suspects, The (1995)',
 'Predestination (2014)',
 'Eternal Sunshine of the Spotless Mind (2004)',
 'Matrix, The (1999)',
 'Prestige, The (2006)',
 'Interstellar (2014)',
 'Donnie Darko (2001)',
 'Mr. Nobody (2009)',
 'Looper (2012)']
```

# Using ML models to predict ratings



# Content-based Filtering Recommendation System

- This Linear Regression model aims to predict ratings based solely on a movie's characteristics. In this implementation, the only features considered are tag scores and genres.
- Other features could be used to help improve the accuracy of the model. However, the algorithm does have inherent limitations which make it a less desirable candidate for solving this problem.

```
from sklearn.model_selection import train_test_split

model_data = data.merge(processed_ratings, on='title', how='outer')
model_data = model_data[model_data['rating'].notna()]

X = model_data.drop(['userId', 'title', 'rating'], axis=1)
y = model_data['rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict ratings on the test set
y_pred = model.predict(X_test)

# Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"RMSE: {rmse}")
RMSE: 0.9294030117250892
```

# Collaborative Filtering Recommendation System

- Collaborative filtering attempts to recommend items based on the preferences of similar users.
- Surprise is a Python scikit for building and analyzing recommender systems that deal with explicit rating data.
- The first algorithm is the Singular Value Decomposition model, which is a matrix factorization technique. This approach performed the best in terms of minimizing the RMSE.
- The second algorithm from this package was the KNN algorithm – this performed a bit worse.

```
ratings_matrix = ratings.pivot(index='userId', columns='movieId', values='rating')

from surprise import SVD, KNNBasic, KNNWithMeans, SVDpp, NMF, BaselineOnly, Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import accuracy

sim_options = {
    'name': 'cosine',
    'user_based': True
}

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

trainset, testset = train_test_split(data, test_size=0.25)

algo = SVD()

algo.fit(trainset)
predictions = algo.test(testset)

accuracy.rmse(predictions)
RMSE: 0.876

algo = KNNBasic(k=40, sim_options=sim_options)

algo.fit(trainset)
predictions = algo.test(testset)

accuracy.rmse(predictions)
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 0.9783
```

# Key Data Challenges & Next Steps

| | Issue |
|---|---|
| 1 | The dataset is large: ~10,000 movies and 1,100 tags. To feed the feature matrix into a matrix factorization model, more memory and compute is needed. |
| 2 | Genome tags are unprocessed, including very similar tags differentiated by typos or extra text. When put into a feature matrix, this could have the effect of overweighting a tag when calculating similarity |
| 3 | Tags may or may not include principal cast and crew (directors, writers, etc.). These are likely correlated with user preferences but missing from the current feature matrix. |

| tagId | tag |
|---|---|
| 1 | 007 |
| 2 | 007 (series) |