ceflynn / Movie-Recommendation-System Public

Code Issues Pull requests Actions Projects Wiki Security

main

Movie-Recommendation-System / student.ipynb

ceflynn updating image files, pdf, student notebook History

1 contributor

7171 lines (7171 sloc) 268 KB

Netflix Movie Reccomendation System

Business Understanding

Netflix is looking to improve their reccomendation system for new users. As part of a new trial membership program Netflix is looking to maximize their customer retention by providing the best possible reccomendations.

Netflix has attracted new users by using a free weekly trial membership. In order to maximize the number of customers that continue their membership, the reccomendations must be match the customers preferences. If the reccomendations are on point the customer is more likely to feel like there are enough options to continue the service past the free trial.

In [1]:

```
#initial imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Data

Import the four datasets to inspect and eventually combine into one dataframe for modeling.

The data is in the data folder:

- data/links.csv
- data/movies.csv
- data/ratings.csv
- data/tags.csv

Links dataframe

this dataframe will come in handy if we end up using additional data from imdb and the tmd for features in our model.

In [111…

```
links = pd.read_csv('data/links.csv')
links.head()
```

Out[111…

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

In [112…

```
links.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   movieId  9742 non-null   int64  
 1   imdbId   9742 non-null   int64  
 2   tmdbId   9734 non-null   float64
dtypes: float64(1), int64(2)
memory usage: 228.5 KB
```

Movies DataFrame

this contains the title and genre of the movies. The movieId column matches with our links dataframe. For example movieId 1 matches with movieId Toystory.

In [113…

```
movies = pd.read_csv('data/movies.csv')
movies.head()
```

Out[113…

	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Children|Comec
1	2	Jumanji (1995)	Adventure|Childre
2	3	Grumpier Old Men (1995)	Comedy
3	4	Waiting to Exhale (1995)	Comedy|Drama
		Father of	

4	5	the Bride Part II (1995)

In [114…

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

In [115…

```
#extract the year of film from the title usir
movies['year'] = movies.title.str.extract(r'(
movies.head()
```

Out[115…

	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Children|Comec
1	2	Jumanji (1995)	Adventure|Childre
2	3	Grumpier Old Men (1995)	Comedy
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	

In [116…

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
 3   year     9729 non-null   object
dtypes: int64(1), object(3)
memory usage: 304.6+ KB
```

In [117…

```
#find the movies without years -
```

```
movies[movies.year.isna()]
```

Out[117…

	movieId	title	genres	year
6059	40697	Babylon 5	Sci-Fi	NaN
9031	140956	Ready Player One	Action|Sci-Fi|Thriller	NaN
9091	143410	Hyena Road	(no genres listed)	NaN
9138	147250	The Adventures of Sherlock Holmes and Doctor W...	(no genres listed)	NaN
9179	149334	Nocturnal Animals	Drama|Thriller	NaN
9259	156605	Paterson	(no genres listed)	NaN
9367	162414	Moonlight	Drama	NaN
9448	167570	The OA	(no genres listed)	NaN
9514	171495	Cosmos	(no genres listed)	NaN
9515	171631	Maria Bamford: Old Baby	(no genres listed)	NaN
9518	171749	Death Note: Desu nôto (2006–2007)	(no genres listed)	NaN
9525	171891	Generation Iron 2	(no genres listed)	NaN
9611	176601	Black Mirror	(no genres listed)	NaN

In [118…

```
# create dictionary to add the years for thes
year_fix_dict ={40697:1998,
                140956:2018,
                143410:2015,
                147250:1939,
                149334:2016,
                156605:2016,
                162414:2002,
                167570:2016,
                171495:1996,
                171631:2017,
                171749:2006,
                171891:2013,
                176601:2013}
```

In [119…

```
movies['year'] = movies['movieId'].map(year_f
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
 3   year     9742 non-null   object
dtypes: int64(1), object(3)
memory usage: 304.6+ KB
```

In [120…

```
movies.head()
```

Out[120…

	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Children|Comec
1	2	Jumanji (1995)	Adventure|Childre
2	3	Grumpier Old Men (1995)	Comedy
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	

Ratings DataFrame

This dataframe contains userId, movieId, rating and a timestamp.

In [121…

```
ratings = pd.read_csv('data/ratings.csv')
ratings.head()
```

Out[121…

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

In [122…

```
ratings.rating.value_counts()
```

Out[122…
```
4.0     26818
3.0     20047
5.0     13211
3.5     13136
4.5      8551
2.0      7551
2.5      5550
1.0      2811
1.5      1791
0.5      1370
Name: rating, dtype: int64
```

In [123…
```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

In [124…
```
ratings.movieId.value_counts()
```

Out[124…
```
356       329
318       317
296       307
593       279
2571      278
         ... 
5986        1
100304      1
34800       1
83976       1
8196        1
Name: movieId, Length: 9724, dtype: int64
```

Tags DataFrame

The tags dataframe has userId, movieId, tag and timestamp

In [125…
```
tags = pd.read_csv('data/tags.csv')
tags.head()
```

Out[125…

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207

4	2	89774	MMA	1445715200

In [126…

```
tags.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3683 entries, 0 to 3682
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   userId     3683 non-null   int64 
 1   movieId    3683 non-null   int64 
 2   tag        3683 non-null   object
 3   timestamp  3683 non-null   int64 
dtypes: int64(3), object(1)
memory usage: 115.2+ KB
```

Tags may be an important feature we will want to explore the tags and see if we can pinpoint some of the most used tags to add to our data

In [127…

```
#tag value counts
tags.tag.value_counts()
```

Out[127…

```
In Netflix queue     131
atmospheric           36
superhero             24
thought-provoking     24
surreal               23
                    ... 
California             1
Mexico                 1
TERRORISM              1
purity of essence      1
Dinosaur               1
Name: tag, Length: 1589, dtype: int64
```

In [128…

```
#create tag dictionary
keys = tags['tag'].value_counts(dropna=False)
vals = tags['tag'].value_counts(dropna=False)
tag_dict = dict(zip(keys, vals))
tag_dict
```

Out[128…

```
{'In Netflix queue': 131,
 'atmospheric': 36,
 'superhero': 24,
 'thought-provoking': 24,
 'surreal': 23,
 'funny': 23,
 'Disney': 23,
 'religion': 22,
 'dark comedy': 21,
 'quirky': 21,
 'psychology': 21,
 'sci-fi': 21,
 'suspense': 20,
 'twist ending': 19
```

```
 'twist ending': 19,
 'visually appealing': 19,
 'crime': 19,
 'politics': 18,
 'time travel': 16,
 'mental illness': 16,
 'music': 16,
 'comedy': 15,
 'dark': 15,
 'aliens': 15,
 'space': 14,
 'mindfuck': 14,
 'dreamlike': 14,
 'emotional': 13,
 'black comedy': 13,
 'heist': 13,
 'Shakespeare': 12,
 'satire': 12,
 'action': 12,
 'court': 12,
 'Stephen King': 12,
 'anime': 12,
 'high school': 12,
 'disturbing': 12,
 'journalism': 12,
 'adolescence': 11,
 'comic book': 11,
 'imdb top 250': 11,
 'boxing': 11,
 'classic': 11,
 'Holocaust': 11,
 'adultery': 11,
 'psychological': 11,
 'cinematography': 10,
 'Mafia': 10,
 'ghosts': 10,
 'England': 10,
 'Australia': 10,
 'remake': 10,
 'drugs': 10,
 'Leonardo DiCaprio': 10,
 'philosophical': 10,
 'India': 10,
 'Vietnam': 10,
 'animation': 10,
 'robots': 10,
 'tense': 9,
 'murder': 9,
 'racism': 9,
 'bittersweet': 9,
 'hallucinatory': 9,
 'military': 9,
 'World War II': 9,
 'sexuality': 9,
 'stylized': 9,
 'creepy': 9,
 'heartwarming': 8,
 'Christmas': 8,
 'movie business': 8,
 'revenge': 8,
 'bad': 8,
```

```
'adventure': 8,
'serial killer': 8,
'sequel': 8,
'spoof': 8,
'divorce': 8,
'violence': 8,
'martial arts': 8,
'race': 8,
'cult film': 7,
'assassination': 7,
'clever': 7,
'intelligent': 7,
'predictable': 7,
'Quentin Tarantino': 7,
'disability': 7,
'inspirational': 7,
'prostitution': 7,
'Animal movie': 7,
'gritty': 7,
'dark humor': 7,
'romance': 7,
'family': 7,
'social commentary': 7,
'police': 7,
'Coen Brothers': 7,
'philosophy': 6,
'fantasy': 6,
'pregnancy': 6,
'Bible': 6,
'black and white': 6,
'wedding': 6,
'zombies': 6,
'business': 6,
'future': 6,
'Magic': 6,
'remade': 6,
'gothic': 6,
'New York': 6,
'cerebral': 6,
'television': 6,
'Astaire and Rogers': 6,
'twins': 6,
'mystery': 6,
'great soundtrack': 6,
'men in drag': 6,
'Nick and Nora Charles': 6,
'witty': 6,
'death': 6,
'hit men': 6,
'touching': 6,
'kidnapping': 6,
'post-apocalyptic': 5,
'paranoia': 5,
'Will Ferrell': 5,
'lawyers': 5,
'Judaism': 5,
'Civil War': 5,
'gambling': 5,
'Atmospheric': 5,
'stylish': 5,
```

```
 'space opera': 5,
 'artificial intelligence': 5,
 'death penalty': 5,
 'cross dressing': 5,
 'sarcasm': 5,
 'poignant': 5,
 'existentialism': 5,
 'fun': 5,
 'Ireland': 5,
 'sports': 5,
 'good dialogue': 5,
 'thriller': 5,
 'biopic': 5,
 'baseball': 5,
 'alcoholism': 5,
 'corruption': 5,
 'Dickens': 5,
 'marriage': 5,
 'swashbuckler': 5,
 'beautiful': 5,
 'humorous': 5,
 'organized crime': 5,
 'Al Pacino': 5,
 'based on a book': 5,
 'amnesia': 5,
 'terrorism': 5,
 'Adam Sandler': 5,
 'friendship': 5,
 'Brad Pitt': 5,
 'Hepburn and Tracy': 5,
 'Africa': 5,
 'dystopia': 5,
 'Jason': 5,
 'samurai': 5,
 'Ryan Reynolds': 4,
 'Ben Stiller': 4,
 'King Arthur': 4,
 'blindness': 4,
 'enigmatic': 4,
 'president': 4,
 'weird': 4,
 'will ferrell': 4,
 'show business': 4,
 'unique': 4,
 'loneliness': 4,
 'Tolkein': 4,
 'Seth Rogen': 4,
 'melancholy': 4,
 'sad': 4,
 'horror': 4,
 'Action': 4,
 'Post apocalyptic': 4,
 'archaeology': 4,
 'parody': 4,
 'Tom Hanks': 4,
 'horses': 4,
 'prison': 4,
 'Jane Austen': 4,
 'movies': 4,
 'true story': 4,
 'birds': 4,
```

```
'birds': 4,
'demons': 4,
'bad plot': 4,
'Wizards': 4,
'visually stunning': 4,
'intense': 4,
'screwball': 4,
'basketball': 4,
'controversial': 4,
'based on a TV show': 4,
'Alfred Hitchcock': 4,
'assassin': 4,
'dogs': 4,
'Oscar (Best Actress)': 4,
'generation X': 4,
'Aardman': 4,
'Samuel L. Jackson': 4,
'soundtrack': 4,
'High School': 4,
'John Grisham': 4,
'heartbreaking': 4,
'fatherhood': 4,
'plot holes': 4,
'survival': 4,
'circus': 4,
'Martin Scorsese': 4,
'interesting': 4,
'feel-good': 4,
'depressing': 4,
'homeless': 4,
'violent': 4,
'Christian Bale': 4,
'paranoid': 4,
'christmas': 4,
'Comedy': 4,
'Pixar': 4,
'immigrants': 4,
'memory': 4,
'Mystery': 3,
'artistic': 3,
'multiple storylines': 3,
'good soundtrack': 3,
'Michael Cera': 3,
'blind': 3,
'Tim Burton': 3,
'Beautiful': 3,
'Screwball': 3,
'obsession': 3,
'books': 3,
'spying': 3,
'Tennessee Williams': 3,
'Clousseau': 3,
'NASA': 3,
'Hollywood': 3,
'based on a true story': 3,
'alternate reality': 3,
'Shakespeare sort of': 3,
'Liam Neeson': 3,
'sweet': 3,
'terminal illness': 3,
'whimsical': 3,
```

```
 '06 Oscar Nominated Best Movie - Animation':
3,
 'Rachel Weisz': 3,
 'golf': 3,
 'gangsters': 3,
 'Highly quotable': 3,
 'bloody': 3,
 'classic sci-fi': 3,
 'motherhood': 3,
 'mafia': 3,
 'orphans': 3,
 'Robin Williams': 3,
 'slasher': 3,
 'overrated': 3,
 'psychedelic': 3,
 'Girl Power': 3,
 'love story': 3,
 'child abuse': 3,
 'inspiring': 3,
 'Rome': 3,
 'nightclub': 3,
 'british comedy': 3,
 'smart': 3,
 'Cold War': 3,
 'food': 3,
 'crude humor': 3,
 'nonlinear': 3,
 'ensemble cast': 3,
 'hilarious': 3,
 'great acting': 3,
 'Steve Carell': 3,
 'dance': 3,
 'hitman': 3,
 'claustrophobic': 3,
 'class': 3,
 '1970s': 3,
 'Tarantino': 3,
 'photography': 3,
 'boring': 3,
 'silly': 3,
 'evil children': 3,
 'off-beat comedy': 3,
 'masterpiece': 3,
 'irreverent': 3,
 'Robert De Niro': 3,
 'football': 3,
 'mockumentary': 3,
 'children': 3,
 'moving': 3,
 'Christopher Nolan': 3,
 'anti-Semitism': 3,
 'priest': 3,
 'fantasy world': 3,
 'mathematics': 3,
 'brutality': 3,
 'transplants': 3,
 'Steve Buscemi': 3,
 'Japan': 3,
 'drama': 3,
 'writing': 3,
```

```
'M. Night Shyamalan': 2,
'figure skating': 2,
'chick flick': 2,
'Jude Law': 2,
'brainwashing': 2,
'indiana jones': 2,
'Hemingway': 2,
'tragic': 2,
'Hugh Jackman': 2,
'radio': 2,
'bad acting': 2,
'characters': 2,
'heroin': 2,
'reciprocal spectator': 2,
'conspiracy theory': 2,
'morality': 2,
'drug abuse': 2,
'Tom Clancy': 2,
'coma': 2,
'1950s': 2,
'rape': 2,
'new york': 2,
'heroine in tight suit': 2,
'original': 2,
'art': 2,
'neo-noir': 2,
'downbeat': 2,
'great dialogue': 2,
'schizophrenia': 2,
'AIDs': 2,
'sexy female scientist': 2,
'space action': 2,
'Nick Hornby': 2,
'college': 2,
'alternate endings': 2,
'POW': 2,
'Paul Giamatti': 2,
'poetic': 2,
'Nazis': 2,
'George Bernard Shaw': 2,
'E.M. Forster': 2,
'twist': 2,
'espionage': 2,
'moon': 2,
'cyberpunk': 2,
'Marvel': 2,
'Cambodia': 2,
'last man on earth': 2,
'satirical': 2,
'personals ads': 2,
'too long': 2,
'Robert Downey Jr.': 2,
'jack nicholson': 2,
'rasicm': 2,
'great ending': 2,
'documentary': 2,
'theater': 2,
'made me cry': 2,
'romantic': 2,
'Marx brothers': 2,
'Seann William Scott': 2,
```

```
'reunion': 2,
'music business': 2,
'jazz': 2,
'Michael Bay': 2,
'Arnold Schwarzenegger': 2,
'France': 2,
'cult': 2,
'Jim Carrey': 2,
'holocaust': 2,
'island': 2,
'lawyer': 2,
'Hannibal Lecter': 2,
'Capote': 2,
'L.A.': 2,
'meditative': 2,
'psychiatrist': 2,
'unconventional': 2,
'cancer': 2,
'apocalypse': 2,
'depression': 2,
'bad script': 2,
'darth vader': 2,
'sentimental': 2,
'halloween': 2,
'Jessica Alba': 2,
'Mark Ruffalo': 2,
'psychopaths': 2,
'dialogue': 2,
'surrealism': 2,
'EPIC': 2,
'marvel': 2,
'moody': 2,
'Christopher Lloyd': 2,
'Will Smith': 2,
'long shots': 2,
'Tolkien': 2,
'non-linear': 2,
'powerful ending': 2,
'Stanley Kubrick': 2,
'sniper': 2,
'Jean Reno': 2,
'Europe': 2,
'Italy': 2,
'lyrical': 2,
'claymation': 2,
'artsy': 2,
'white guilt': 2,
'Jake Gyllenhaal': 2,
'Christina Ricci': 2,
'guns': 2,
'original plot': 2,
'Steven Spielberg': 2,
'courtroom drama': 2,
'understated': 2,
'narrated': 2,
'audience intelligence underestimated': 2,
'1920s': 2,
'humor': 2,
'ballet': 2,
'love': 2,
```

```
'Studio Ghibli': 2,
'James Stewart': 2,
'Agatha Christie': 2,
'beautiful scenery': 2,
'twists & turns': 2,
'eerie': 2,
'scary': 2,
'vampire': 2,
'fugitive': 2,
'Jared Leto': 2,
'South America': 2,
'Anne Hathaway': 2,
'bromance': 2,
'surfing': 2,
'intellectual': 2,
'freaks': 2,
'trippy': 2,
'secret society': 2,
'Christoph Waltz': 2,
'gangster': 2,
'fairy tales': 2,
'Nudity (Full Frontal)': 2,
'Ray Bradbury': 2,
'Star Wars': 2,
'Graham Greene': 2,
'retro': 2,
'bleak': 2,
'Henry James': 2,
'luke skywalker': 2,
'Amish': 2,
'TV': 2,
'fish': 2,
'scifi cult': 2,
'violence in america': 2,
'Rob Zombie': 2,
'Chris Evans': 2,
'harsh': 2,
'FBI': 2,
'character study': 2,
'Keanu Reeves': 2,
'Morgan Freeman': 2,
'goofy': 2,
'Charlize Theron': 2,
'complicated': 2,
'Edith Wharton': 2,
'spaghetti western': 2,
'babies': 2,
'school': 2,
'helena bonham carter': 2,
'tear jerker': 2,
'war': 2,
'virginity': 2,
'awesome': 2,
'vampires': 2,
'time-travel': 2,
'Star Trek': 2,
'Not available from Netflix': 2,
'Bittersweet': 2,
'dark hero': 2,
'treasure hunt': 2,
```

```
 'James Franco': 2,
 'symbolism': 2,
 'edward norton': 2,
 'Visually stunning': 2,
 'pixar': 2,
 'brutal': 2,
 'Rogers and Hammerstein': 2,
 'reflective': 2,
 'factory': 2,
 'Bruce Willis': 2,
 'offensive': 2,
 'Soundtrack': 2,
 'train': 2,
 'trains': 2,
 'New York City': 2,
 'very funny': 2,
 'teen': 2,
 'chess': 2,
 'Myth': 2,
 'Edward Norton': 2,
 'futuristic': 2,
 'cynical': 2,
 'dating': 2,
 'insanity': 2,
 'alternate universe': 2,
 'Olympics': 2,
 'seen more than once': 2,
 'imagination': 2,
 'confrontational': 2,
 'Tom Hardy': 2,
 'Ben Affleck': 2,
 'hugh jackman': 2,
 'Paul Rudd': 2,
 'space travel': 2,
 'epic': 2,
 'graphic design': 2,
 'big budget': 2,
 'plot twist': 2,
 'slick': 2,
 '1980s': 2,
 'doctors': 2,
 'notable soundtrack': 2,
 'dc comics': 2,
 'ridiculous': 2,
 'submarine': 2,
 'C.S. Lewis': 2,
 'scandal': 2,
 'ironic': 2,
 'suspenseful': 2,
 'Jeff Bridges': 2,
 'adoption': 2,
 'weddings': 2,
 'dinosaurs': 2,
 'Wall Street': 2,
 'Peter Pan': 2,
 'widows/widowers': 2,
 'e-mail': 1,
 'childish naivity': 1,
 'deaf': 1,
 'Psychological Thriller': 1,
 'Thanksgiving': 1,
```

```
 'motherfucker': 1,
 'gore': 1,
 'Sci-Fi': 1,
 'fast-paced': 1,
 'Emma Stone': 1,
 'fatalistic': 1,
 'Saturday Night Live': 1,
 'Broadway': 1,
 'chilly': 1,
 'ending': 1,
 'CGI': 1,
 'southern US': 1,
 'Margot Robbie': 1,
 'Bill Murray': 1,
 'villain nonexistent or not needed for good
story': 1,
 'Loretta Lynn': 1,
 'really bad': 1,
 'lies': 1,
 'Music': 1,
 'cult classic': 1,
 'Dr. Strange': 1,
 'In Your Eyes': 1,
 'stupid ending': 1,
 'Well Done': 1,
 'tension building': 1,
 'matchmaker': 1,
 'whales': 1,
 'first was much better': 1,
 'Great villain': 1,
 'Istanbul': 1,
 'oil': 1,
 'Motivational': 1,
 'based on a play': 1,
 'grim': 1,
 'Brooch': 1,
 'golfing': 1,
 'PTSD': 1,
 'hula hoop': 1,
 'Las Vegas': 1,
 'Documentary': 1,
 'crime scene scrubbing': 1,
 'stop looking at me swan': 1,
 'addiction': 1,
 'macho': 1,
 'new society': 1,
 'not funny': 1,
 'interracial marriage': 1,
 'fucked up': 1,
 'Alicia Vikander': 1,
 'Favelas': 1,
 'video games': 1,
 'predictible plot': 1,
 'Gangs': 1,
 'nanny': 1,
 'Sci-fi': 1,
 'purposefulness': 1,
 'pizza beer': 1,
 'Oscar (Best Effects - Visual Effects)': 1,
 '2001-like': 1,
```

```
 'Afghanistan': 1,
 'live action/animation': 1,
 'daniel radcliffe': 1,
 'Funny': 1,
 'Suspense': 1,
 'Dodie Smith': 1,
 'Nun': 1,
 'masculinity': 1,
 'big top': 1,
 'revolutionary': 1,
 'Something for everyone in this one... saw i
t without and plan on seeing it with kids!':
1,
 'jackie chan': 1,
 'royal with cheese': 1,
 'vertriloquism': 1,
 'threesome': 1,
 'dark fairy tale': 1,
 'gintama': 1,
 'opera': 1,
 'indie record label': 1,
 'Deep Throat': 1,
 'saint': 1,
 'biography': 1,
 'Joker': 1,
 'Roger Avary': 1,
 'Witty': 1,
 'Unique': 1,
 'consumerism': 1,
 'Romans': 1,
 'Dumas': 1,
 'assassin-in-training (scene)': 1,
 'memory loss': 1,
 'dumpster diving': 1,
 'ben stiller': 1,
 'black humour': 1,
 'Hal': 1,
 'history': 1,
 'allegorical': 1,
 'parenthood': 1,
 'emma thompson': 1,
 'Horror': 1,
 'western': 1,
 'nonlinear storyline': 1,
 'quotable': 1,
 'sophisticated': 1,
 'geeky': 1,
 'Bad writing': 1,
 'women': 1,
 'Oscar Wilde': 1,
 'romantic comedy': 1,
 'James Fennimore Cooper': 1,
 'Anthony Hopkins': 1,
 'updated classics': 1,
 'rap': 1,
 'art house': 1,
 'police corruption': 1,
 'painter': 1,
 'bad-ass': 1,
 'magic board game': 1,
 'zoe kazan': 1
```

```
 'Zoe Kazan': 1,
 'union': 1,
 'lesbian subtext': 1,
 'gruesome': 1,
 'Modern war': 1,
 'con men': 1,
 'nostalgia': 1,
 'Western': 1,
 'embarassing scenes': 1,
 'beautiful cinematography': 1,
 'cool': 1,
 'Colin Farrell': 1,
 'innovative': 1,
 'rebellion': 1,
 'Up series': 1,
 'muppets': 1,
 'Ben Kingsley': 1,
 'remix culture': 1,
 'confusing ending': 1,
 'Morrow': 1,
 'fairy tale': 1,
 'american idolatry': 1,
 'governess': 1,
 'jake gyllenhaal': 1,
 'plastic surgery': 1,
 'bad language': 1,
 'Food': 1,
 'r:strong language': 1,
 'roald dahl': 1,
 'It was melodramatic and kind of dumb': 1,
 'pudding': 1,
 'money': 1,
 'Hannibal Lector': 1,
 'ransom': 1,
 'rug': 1,
 'goretastic': 1,
 'MMA': 1,
 'Enterprise': 1,
 'Gulf War': 1,
 'unusual': 1,
 'double life': 1,
 'conversation': 1,
 'Monty Python': 1,
 'Norman Bates': 1,
 'Everything you want is here': 1,
 'Stock Market': 1,
 'the catholic church is the most corrupt org
anization in history': 1,
 'space adventure': 1,
 'film-noir': 1,
 'Mark Wahlberg': 1,
 'suburbia': 1,
 'Philip K. Dick': 1,
 'sexy': 1,
 'Charles Dickens': 1,
 'noir': 1,
 'mobster': 1,
 'ethics': 1,
 'big wave': 1,
 'moldy': 1,
 'Francis Ford Coppola': 1,
```

```
 'missing children': 1,
 'SNL': 1,
 'Halloween': 1,
 'restaurant': 1,
 'adorable': 1,
 'invisibility': 1,
 'mermaid': 1,
 'Batman': 1,
 'smart writing': 1,
 'r:violence': 1,
 'stephen king': 1,
 'Harvey Keitel': 1,
 'hip hop': 1,
 'zither': 1,
 'Nudity (Topless)': 1,
 'steve carell': 1,
 'stupid': 1,
 'Ichabod Crane': 1,
 'Visually appealing': 1,
 'Adventure': 1,
 'jim carrey': 1,
 'beautiful visuals': 1,
 'absorbing': 1,
 'china': 1,
 'remaster': 1,
 'multiple stories': 1,
 'nocturnal': 1,
 'Doc Ock': 1,
 'deadpan': 1,
 'Depressing': 1,
 'wry': 1,
 'Robert Penn Warren': 1,
 'singletons': 1,
 'Anne Boleyn': 1,
 'awkward': 1,
 'dumb': 1,
 'hotel': 1,
 'non-linear timeline': 1,
 'creativity': 1,
 'freedom of expression': 1,
 'dance marathon': 1,
 'comics': 1,
 'great performances': 1,
 'Shark': 1,
 'wizards': 1,
 'FIGHTING THE SYSTEM': 1,
 'somber': 1,
 'prom': 1,
 'independent': 1,
 'blood': 1,
 'Scotland': 1,
 'wine': 1,
 'video': 1,
 'Tolstoy': 1,
 'ark of the covenant': 1,
 'Jim Morrison': 1,
 'Animation': 1,
 'best comedy': 1,
 'Dystopia': 1,
 'anthology': 1,
```

```
'menacing': 1,
'casual violence': 1,
'batman': 1,
'Chile': 1,
'multiple personalities': 1,
'conspiracy': 1,
'annoying': 1,
'happy ending': 1,
'tragedy': 1,
'ocean': 1,
'humour': 1,
'carnival': 1,
'McDonalds': 1,
'oldie but goodie': 1,
'bad dialogue': 1,
'Middle East': 1,
'nonsense': 1,
'too much love interest': 1,
'weather forecaster': 1,
'70mm': 1,
'r:some violence': 1,
'Scifi masterpiece': 1,
'cool style': 1,
'drug overdose': 1,
'Lou Gehrig': 1,
'doll': 1,
'Jack Nicholson': 1,
'AS Byatt': 1,
'achronological': 1,
'start of a beautiful friendship': 1,
'Navy': 1,
'system holism': 1,
'magic': 1,
'setting:space/space ship': 1,
'leopard': 1,
'casino': 1,
'Stupid ending': 1,
'skiing': 1,
'Bette Davis': 1,
'planes': 1,
'foul language': 1,
'Savannah': 1,
'Salieri': 1,
'ummarti2006': 1,
'Politics': 1,
'Robert Ludlum': 1,
'transvestite': 1,
'inhumane': 1,
'twisted': 1,
'dreams': 1,
'Titanic': 1,
'Orlando Bloom': 1,
'cia': 1,
'Great movie': 1,
'soccer': 1,
'a clever chef rat': 1,
'leonardo DiCarpio': 1,
'Slim Pickens': 1,
'Michael Crichton': 1,
'cheating': 1,
'Nuclear disaster': 1,
```

```
 'Twist Ending': 1,
 'haunting': 1,
 'truckers': 1,
 'Lolita theme': 1,
 'unoriginal': 1,
 'big name actors': 1,
 'Hilary Swank': 1,
 'Canada': 1,
 'Ralph Fiennes': 1,
 'sisters': 1,
 'Quakers': 1,
 'Mount Rushmore': 1,
 'cheeky': 1,
 'shipwreck': 1,
 'Bugs Bunny': 1,
 'different': 1,
 'Rosebud': 1,
 'dreamy': 1,
 'HOT actress': 1,
 'ancient Rome': 1,
 'Quotable': 1,
 'heroic bloodshed': 1,
 'Jesse Eisenberg': 1,
 'Matrix': 1,
 'Eva Green': 1,
 'kung fu': 1,
 'Dull': 1,
 'I am your father': 1,
 'passion': 1,
 'werewolf': 1,
 'tearjerking': 1,
 'Death': 1,
 'gun-fu': 1,
 'Academy award (Best Supporting Actress)':
1,
 'Palahnuik': 1,
 'Jennifer Connelly': 1,
 'wapendrama': 1,
 'special effects': 1,
 'Henry Darger': 1,
 'robbery': 1,
 'teacher': 1,
 'busniess': 1,
 'Shangri-La': 1,
 'Michigan': 1,
 'pigs': 1,
 'slow': 1,
 'Rogue': 1,
 'cruel characters': 1,
 'Guardians of the Galaxy': 1,
 'prodigies': 1,
 'Maggie Gyllenhaal': 1,
 'poor dialogue': 1,
 'unnecessary sequel': 1,
 'fighting': 1,
 'philosophical issues': 1,
 'spacecraft': 1,
 'World War I': 1,
 'hippies': 1,
 'blood splatters': 1,
```

```
'acting': 1,
'Eric Bana': 1,
'suicide': 1,
'convent': 1,
'Train': 1,
'McCarthy hearings': 1,
'challenging': 1,
'virtual reality': 1,
'Amtrak': 1,
'David Thewlis': 1,
'Captain Kirk': 1,
'Missionary': 1,
'Philip Seymour Hoffman': 1,
'representation of children': 1,
'Tokyo': 1,
'black hole': 1,
'Insane': 1,
'Not Seen': 1,
'celebrity fetishism': 1,
'John Cusack': 1,
'slavery': 1,
'Surreal': 1,
'bizarre': 1,
'Rachel McAdams': 1,
'game': 1,
'contemplative': 1,
'diner': 1,
'Heartwarming': 1,
'ben affleck': 1,
'sword fight': 1,
'rich guy - poor girl': 1,
'multiple short stories': 1,
'sofia coppola': 1,
'voyeurism': 1,
'Nabokov': 1,
'stone age': 1,
'Notable Nudity': 1,
'Kurt Russell': 1,
'bluegrass': 1,
'May-December romance': 1,
'lack of plot': 1,
'bears': 1,
'falling': 1,
'strange': 1,
'mining': 1,
'David Bowie': 1,
'camp': 1,
'Epic': 1,
'Stoner Movie': 1,
'Lloyd Dobbler': 1,
'families': 1,
'cryptic': 1,
'Harrison Ford': 1,
'Visually Striking': 1,
'Thanos': 1,
'spelling bee': 1,
'amazing artwork': 1,
'genocide': 1,
'splatter': 1,
'test tag': 1,
```

```
'unintelligent': 1,
'flood': 1,
'pool': 1,
'Josh Brolin': 1,
'italy': 1,
'GIVE ME BACK MY SON!': 1,
'dust bowl': 1,
'lieutenant dan': 1,
'bowling': 1,
'visual': 1,
'surprise ending': 1,
'secrets': 1,
'Pearl S Buck': 1,
'1990s': 1,
'Hungary': 1,
'kids': 1,
'Day and Hudson': 1,
'Creature Feature': 1,
'alone in the world': 1,
'mice': 1,
'a dingo ate my baby': 1,
'evolution': 1,
'mad scientist': 1,
"Palme d'Or": 1,
'Ed Harris': 1,
'engrossing adventure': 1,
'golden watch': 1,
...}
```

We may come back to the tags later.

Combined DataFrame

Below we will add the movie titles and genres to the ratings data to make a combined data frame

1. start the ratings dataframe and drop the timestamp.
2. use the movieId column to add the title and genre of the movie

In [129…

```
#combined dataframe
#drop the timestamp column
df = ratings.drop('timestamp', axis=1)
#add title,genre and year using merge how=lei
df = df.merge(movies, on='movieId', how='left
df.head()
```

Out[129…

	userId	movieId	rating	title	
0	1	1	4.0	Toy Story (1995)	Adventure|Animati
1	1	3	4.0	Grumpier Old Men (1995)	

2	1	6	4.0	Heat (1995)
3	1	47	5.0	Seven (a.k.a. Se7en) (1995)
4	1	50	5.0	Usual Suspects, The (1995)

In [130…
```
df.shape
```
Out[130…
```
(100836, 6)
```

In [131…
```
df.describe()
```
Out[131…

	userId	movieId	rating
count	100836.000000	100836.000000	100836.000000
mean	326.127564	19435.295718	3.501557
std	182.618491	35530.987199	1.042529
min	1.000000	1.000000	0.500000
25%	177.000000	1199.000000	3.000000
50%	325.000000	2991.000000	3.500000
75%	477.000000	8122.000000	4.000000
max	610.000000	193609.000000	5.000000

User ids range from 1-610. We will need to create new user ids that are outside of this range.

Further Data Exploration

In [132…
```
#number of unique users
n_users = df.userId.nunique()
print(n_users,'users that have rated movies.'
##movies rated
mov_rat=df.movieId.nunique()
print(mov_rat,'different movies rated')
```
```
610 users that have rated movies.
9724 different movies rated
```

In [133…
```
##top 20 best rated movies
agg_function = {'rating':['mean','count']}
movie_ratings = df.groupby(['movieId','title'
movie_ratings.sort_values(by=('rating','mean'
```

Out[133…

			me
movieId	**title**	**genres**	
88448	**Paper Birds (Pájaros de papel) (2010)**	**Comedy|Drama**	
100556	**Act of Killing, The (2012)**	**Documentary**	
143031	**Jump In! (2007)**	**Comedy|Drama|Romance**	
143511	**Human (2015)**	**Documentary**	
143559	**L.A. Slasher (2015)**	**Comedy|Crime|Fantasy**	
...	**...**	**...**	
157172	**Wizards of the Lost Kingdom II (1989)**	**Action|Fantasy**	
85334	**Hard Ticket to Hawaii (1987)**	**Action|Comedy**	
53453	**Starcrash (a.k.a. Star Crash) (1978)**	**Action|Adventure|Fantasy|Sci-Fi**	
8494	**Cincinnati Kid, The (1965)**	**Drama**	
71810	**Legionnaire (1998)**	**Action|Adventure|Drama|War**	

9724 rows × 2 columns

We can see that we have many 5 rated movies as well as many .5 rated movies. It is good to know how many times each movie was rated as I have never heard of any of the movies that are currently listed at the top of the rating list. We have added the count to the agg function so now we can sort the movies by count.

In [134…

```
movie_ratings.sort_values(by=('rating','count
```

Out[134…

movieId	title	genres	
356	Forrest Gump (1994)	Comedy|Drama|Romance|War	4.
318	Shawshank Redemption, The (1994)	Crime|Drama	4.4
296	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller	4.
593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller	4.
2571	Matrix, The (1999)	Action|Sci-Fi|Thriller	4.
...	...	...	
4093	Cop (1988)	Thriller	1.5
4089	Born in East L.A. (1987)	Comedy	2.0
58351	City of Men (Cidade dos Homens) (2007)	Drama	4.0
4083	Best Seller (1987)	Thriller	4.0
193609	Andrew Dice Clay: Dice Rules (1991)	Comedy	4.0

9724 rows × 2 columns

Now the movies at the top of the list are recognizable. Now we have an idea of movies that have been rated alot and most likely watched the most. This will be helpful when selecting movies for new users to rate. We want to only suggest movies that we currently have a good number of ratings for. This will make it more likely that they have seen the movie and it will make our model more useful because their will me users that have rated those movies.

Currently we have movies with anywhere from 1-329 rankings.

Finding the Best Movies for User Survey

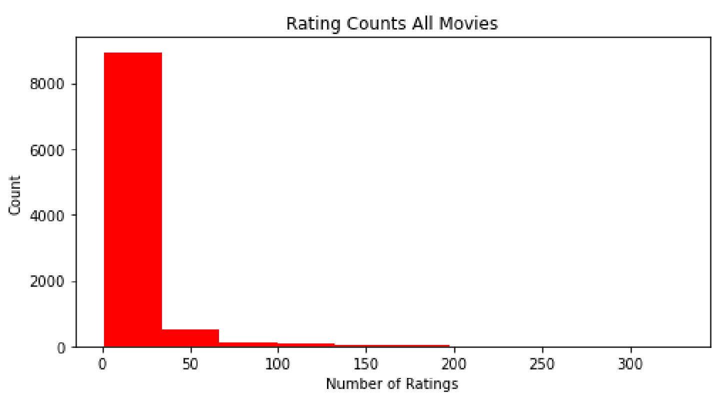
It may be interesting to find movies that have a good balance amoung ratings. These movies may be better at pinpointing what a new user may like. for example movies that get mostly ratings of 4 or 5 may not tell us as much about a viewer as movies that recieve an equal amount of ratings from 1-5 or polarizing ratings. How do we do this...

For the sake of time we will limit the movies included in the user survey to movies that have atleast n ratings.

We can plot rating counts to see what a good number will be.

In [135…

```
#histogram of rating count
fig, ax = plt.subplots(figsize=(8,4))
ax.hist(movie_ratings[('rating','count')],bir
ax.set_title('Rating Counts All Movies')
ax.set_ylabel('Count')
ax.set_xlabel('Number of Ratings')
plt.show()
```



It looks like if we limit the movies for the survey to any movie that has more than 20 ratings we will have a good number of movies that the user has possibly seen and that enough other people have rated.

In [136…

```
survey_movies = movie_ratings[movie_ratings[(
survey_movies
```

Out[136…

movieId	title
	Toy

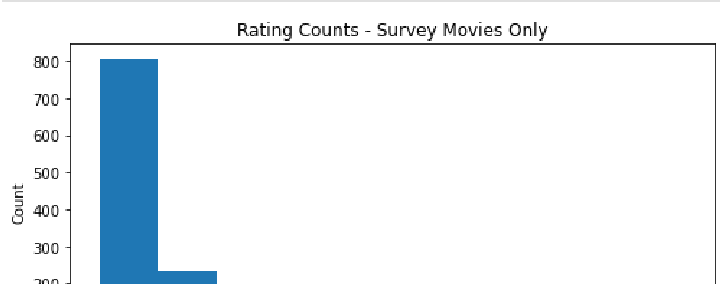
1	**Story (1995)**	**Adventure|Animation|Children|Comec**
2	**Jumanji (1995)**	**Adventure|Childre**
3	**Grumpier Old Men (1995)**	**Comedy**
5	**Father of the Bride Part II (1995)**	
6	**Heat (1995)**	**Action|Crir**
...	**...**	
148626	**Big Short, The (2015)**	
152081	**Zootopia (2016)**	**Action|Adventure|Animation|Childre**
164179	**Arrival (2016)**	
166528	**Rogue One: A Star Wars Story (2016)**	**Action|Adventure|Fant**
168252	**Logan (2017)**	**Ac**

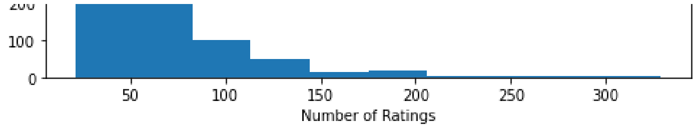
1235 rows × 2 columns

1235 movies will be included in our user rating survey.

In [137…

```
#histogram of rating count
fig, ax = plt.subplots(figsize=(8,4))
ax.hist(survey_movies[('rating','count')],bir
ax.set_title('Rating Counts - Survey Movies O
ax.set_ylabel('Count')
ax.set_xlabel('Number of Ratings')
plt.show()
```





Ratings Distribution Breakdown

Visualizing the ratings.

In [138…

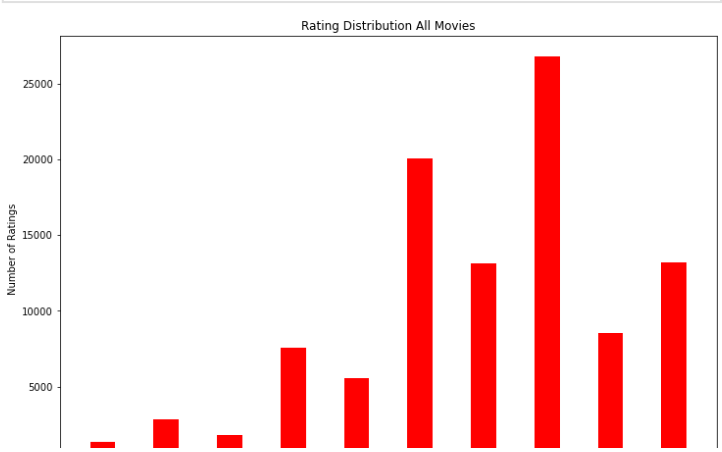
```
rating_table = pd.DataFrame(df.groupby(['rati
rating_table
```

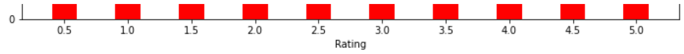
Out[138…

	rating	Count
0	0.5	1370
1	1.0	2811
2	1.5	1791
3	2.0	7551
4	2.5	5550
5	3.0	20047
6	3.5	13136
7	4.0	26818
8	4.5	8551
9	5.0	13211

In [139…

```
## ratings histogram
xs = rating_table['rating']
ys = rating_table['Count']
fig, ax = plt.subplots(figsize=(12,8))
ax.bar(xs,ys,tick_label=xs, width=0.2, color=
ax.set_title('Rating Distribution All Movies'
ax.set_ylabel('Number of Ratings')
ax.set_xlabel('Rating')
plt.show()
```





Movies By Year

In [140…

```
movies.year.value_counts()
```

Out[140…

```
2002      311
2006      295
2001      294
2007      284
2000      283
         ...
1917        1
1921        1
1908        1
1922        1
1939.0      1
Name: year, Length: 116, dtype: int64
```

In [141…

```
movies['year'] = movies['year'].astype(int)
```

In [142…

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  9742 non-null   int64
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
 3   year     9742 non-null   int64
dtypes: int64(2), object(2)
memory usage: 304.6+ KB
```

In [143…

```
max_year = movies.year.max()
min_year = movies.year.min()
print('Data includes movies from {} to {}'.fc
```

```
Data includes movies from 1902 to 2018
```

In [144…

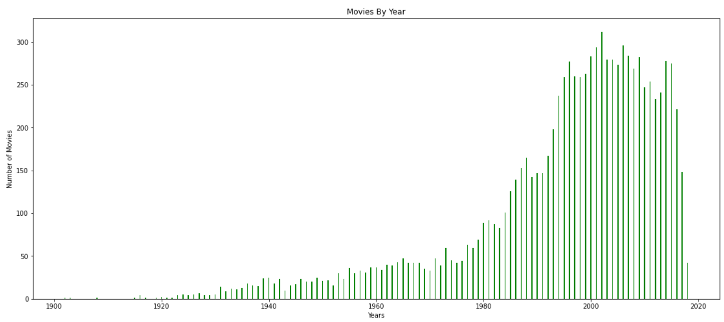
```
by_year = pd.DataFrame(movies.groupby(['year'
by_year.head()
```

Out[144…

	year	Count
0	1902	1
1	1903	1
2	1908	1
3	1915	1
4	1916	4

In [145…

```
## ratings bar plot
xs = by_year['year']
ys = by_year['Count']
fig, ax = plt.subplots(figsize=(19,8))
ax.bar(xs,ys, width=0.2, color='green')
ax.set_title('Movies By Year')
ax.set_ylabel('Number of Movies')
ax.set_xlabel('Years')
plt.show()
```



I wonder if the year a movie was made or was rated has an influence on the average ratings.

Create User - Rating Matrix

We will create a matrix that has users and columns for each movie with that user ratings. This will be a very large sparse matrix. -- lots of zeros...

use df and pivot userId,movieId,rating

In [146…

```
##create matrix from
model_matrix = df.pivot(index='userId',columr
model_matrix.head()
```

Out[146…

movieId	1	2	3	4	5	6	7	8	9	10
userId										
1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 9724 columns

In [147…

```
model_matrix.shape
```

Out[147…
```
(610, 9724)
```

In [148…
```
non_zero = np.count_nonzero(model_matrix)
sparse_percentage = 1-(non_zero/(model_matrix
print('matrix sparse percentage: {}%'.format(
```

```
matrix sparse percentage: 98%
```

As expected this is a sparse matrix will help in deciding which direction we will move in our iterative modeling process. Using the surprise library we will not need the model_matrix but it is interesting to see that our ratings data is 98% empty.

Surprise

We will import the needed tools from the Surprise libray below and begin our iterative modeling process.

In [149…
```
from surprise import Reader, Dataset
from surprise.model_selection import cross_va
from surprise.prediction_algorithms import SV
from surprise.model_selection import GridSear
```

We need our ratings data here. Lets make sure that we have the right data. We mostlikely need to drop the timestamp column still.

In [150…
```
ratings.head()
```

Out[150…

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

In [151…
```
# drop timestamp
rate_df = ratings.drop('timestamp', axis=1)
```

In [152…
```
#create the surprise dataset
```

```
reader=Reader()
data=Dataset.load_from_df(rate_df,reader)
dataset=data.build_full_trainset()
dataset
```

Out[152…

```
<surprise.trainset.Trainset at 0x7fd3d93878e0
>
```

Lets explore the new surprise dataset to see if everything looks correct. We can look at the items and users to see how it compares to our original data.

In [153…

```
#print out users and items
items = dataset.n_items
users = dataset.n_users
print('Users: {}\tItems: {}'.format(users,ite
```

```
Users: 610      Items: 9724
```

This matches with our matrix above. We are ready to model.

Iterative Modeling Process

For our modeling process we will begin with our baseline model. Because we have seen this data before we will start with our best parameters from a SVD model and then grid search around those values to see if we can do better.

SVD

Singular Value Decomposition is a widely used dimensionality reduction tool.
In our previous work we found by using gridsearch that {'n_factors': 50, 'reg_all': 0.05} were the best parameters. We will run that first for our baseline model.

In [44]:

```
#svd baseline
baseline_model = SVD(n_factors=50,reg_all=0.0
baseline_model.fit(dataset)
```

Out[44]:

```
<surprise.prediction_algorithms.matrix_factor
ization.SVD at 0x7fd3ea80a820>
```

In [45]:

```
#cross-validate baseline model
baseline_cv = cross_validate(baseline_model,c
```

In [46]:
```
#print out results
for i in baseline_cv.items():
    print(i,'/n')
```

```
('test_rmse', array([0.86407823, 0.8721453 ,
0.87308509, 0.8758768 , 0.85983875])) /n
('test_mae', array([0.66634417, 0.6711445 ,
0.67079666, 0.67285405, 0.66148722])) /n
('fit_time', (2.6540627479553223, 2.805015802
383423, 2.5758962631225586, 2.579800128936767
6, 2.585649251937866)) /n
('test_time', (0.09466719627380371, 0.0928702
3544311523, 0.08849716186523438, 0.0943369865
4174805, 0.08001995086669922)) /n
```

In [47]:
```
model_avg = np.mean(baseline_cv['test_rmse'])
model_avg
```

Out[47]:
```
0.8690048325664689
```

Create a Dictionary To Store Model Results

We want to store our model name and rmse in a dictionary to easily compare. We will also create a function to add further scores to our dictionary.

In [48]:
```
#score_dict will be used to store
score_dict={}
def add_to_dict(dict,model,score):
    dict[model]=score
    return dict
add_to_dict(score_dict,'baseline_model',model
```

Out[48]:
```
{'baseline_model': 0.8690048325664689}
```

SVD GridSearch

Lets see if we can improve on on our model by using a parameter grid search.

We used n_factors = 50(number of factors) and reg_all=0.05(regularization term) for these we will include values closer to these to test, because we had a wider range in our previous work.

lets include:

- n_epochs - The number of iterations default

- n_epochs - The number of iterations default-100
- lr_all – Parameter learning rate default - 0.005

Best Params

{'rmse': 0.8521706144532271, 'mae': 0.6524552323348267} {'rmse': {'n_factors': 60, 'reg_all': 0.075, 'n_epochs': 50, 'lr_all': 0.01}, 'mae': {'n_factors': 60, 'reg_all': 0.075, 'n_epochs': 50, 'lr_all': 0.01}}

*don't run the next cell to save time...

In [50]:

```
#svd gridsearch  -  This will take some time.

#params = {'n_factors':[40,50,60],
         #'reg_all':[0.025,0.05, 0.075],
         #'n_epochs':[25,50,100],
         #'lr_all':[0.0025,0.005,0.01]}
#svd_gs = GridSearchCV(SVD,param_grid=params,
#svd_gs.fit(data)
```

In [51]:

```
#print(svd_gs.best_score)
#print(svd_gs.best_params)
```

Function to Fit and Get RMSE Scores From Model

The model_process function will perform the following steps:

1. Fit the model
2. Train the model
3. Cross Validate the model
4. Store the mean RMSE for the model

In [154…

```
#our best model.
best_svd = SVD(n_factors=60, reg_all=0.075, l
```

In [155…

```
##function
def model_process(model,name,train=dataset,fu
    '''
    model- actual model
    name - string of model name for storing i
    train - training data
    full_data - all of the data
    dict - scoring dictionary
```

```
    '''
    #fit the model
    model.fit(train)

    #cross-validate the model
    model_cv = cross_validate(model,full_data

    #score RMSE
    rmse = np.mean(model_cv['test_rmse'])

    #add to score dictionary
    add_to_dict(dict,name,rmse)

    return dict
```

In [156…

```
#our best svd model just changes the reg_all.
best_svd = SVD(n_factors=60, reg_all=0.075,lr
#fit, score and add to dictionary using funct
model_process(best_svd,'best_svd',dataset,dat
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 sp
lit(s).

                  Fold 1  Fold 2  Fold 3  Fol
d 4  Fold 5  Mean    Std
RMSE (testset)    0.8589  0.8502  0.8686  0.8
600  0.8622  0.8600  0.0060
MAE (testset)     0.6619  0.6538  0.6656  0.6
600  0.6620  0.6606  0.0039
Fit time          3.02    3.21    2.93    2.9
9    2.90    3.01    0.11
Test time         0.09    0.08    0.10    0.0
9    0.08    0.09    0.01
```

Out[156…

```
{'baseline_model': 0.8690048325664689,
 'best_svd': 0.859974911395814,
 'knn_basic': 0.9724861639611996,
 'knn_baseline': 0.8774229661900085,
 'knn_wm': 0.8967312758649276,
 'knn_wzs': 0.8922658679927045,
 'knn_baseline_min_k_5': 0.8657669128835253,
 'knn_baseline_k_30': 0.865934148375465,
 'knn_baseline_best': 0.8664549712431681}
```

This is only a slight improvement in our model. Rememeber that our rating scale is 1-5. So we are still off by about .86 of rating point.

Below we will try some other models

KNN Algorithms

lets compare

- KNNBasic
- KNNBaseline

- KNNBaseline
- KNNWithMeans
- KNNWithZScore

We can do gridsearch with these to see if we can do better.

KNNBasic

basic KNN model

In [52]:

```
##KNNBasic
knn_basic = KNNBasic(sim_options={'name': 'pe
model_process(knn_basic,'knn_basic')
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBasic on
5 split(s).

                  Fold 1  Fold 2  Fold 3  Fol
d 4  Fold 5  Mean    Std
RMSE (testset)    0.9735  0.9665  0.9744  0.9
734  0.9746  0.9725  0.0030
MAE (testset)     0.7550  0.7451  0.7547  0.7
505  0.7500  0.7511  0.0036
Fit time          0.35    0.34    0.35    0.3
1    0.30    0.33    0.02
Test time         0.98    1.03    1.02    1.0
0    0.91    0.99    0.04
```

Out[52]:

```
{'baseline_model': 0.8690048325664689,
 'best_svd': 0.8584878623389438,
 'knn_basic': 0.9724861639611996}
```

KNNBaseline

KNN that takes a baseline rating into account.

In [53]:

```
knn_baseline = KNNBaseline(sim_options={'name
model_process(knn_baseline,'knn_baseline')
```

```
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBaseline
on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fol
d 4  Fold 5  Mean    Std
RMSE (testset)    0.8838  0.8779  0.8774  0.8
698  0.8781  0.8774  0.0045
MAE (testset)     0.6736  0.6704  0.6676  0.6
654  0.6715  0.6697  0.0029
Fit time          0.39    0.41    0.39    0.3
5    0.35    0.38    0.02
```

```
Test time          1.40    1.41    1.39    1.3
7    1.37    1.39    0.02
```

Out[53]:

```
{'baseline_model': 0.8690048325664689,
 'best_svd': 0.8584878623389438,
 'knn_basic': 0.9724861639611996,
 'knn_baseline': 0.8774229661900085}
```

KNNWithMeans

Takes into account mean rating for each user

In [54]:

```
#KNNWithMeans
knn_wm = KNNWithMeans(sim_options={'name': 'p
model_process(knn_wm,'knn_wm')
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithMean
s on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fol
d 4  Fold 5  Mean    Std
RMSE (testset)    0.8930  0.8842  0.8929  0.9
002  0.9134  0.8967  0.0098
MAE (testset)     0.6821  0.6771  0.6798  0.6
837  0.6906  0.6827  0.0045
Fit time          0.32    0.33    0.32    0.3
1    0.30    0.32    0.01
Test time         1.20    1.21    1.17    1.1
2    1.09    1.16    0.05
```

Out[54]:

```
{'baseline_model': 0.8690048325664689,
 'best_svd': 0.8584878623389438,
 'knn_basic': 0.9724861639611996,
 'knn_baseline': 0.8774229661900085,
 'knn_wm': 0.8967312758649276}
```

KNNWithZScore

Takes into account the z-score normalization of each user.

In [55]:

```
knn_wzs = KNNWithZScore(sim_options={'name':
model_process(knn_wzs,'knn_wzs')
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithZSco
re on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fol
d 4  Fold 5  Mean    Std
RMSE (testset)    0.8885  0.8922  0.8925  0.8
972  0.8909  0.8923  0.0029
MAE (testset)     0.6721  0.6718  0.6740  0.6
800  0.6733  0.6742  0.0030
Fit time          0.34    0.34    0.34    0.3
```

```
5    0.34    0.34    0.00
Test time          1.24    1.24    1.24    1.1
9    1.17    1.22    0.03
```

Out[55]:

```
{'baseline_model': 0.8690048325664689,
 'best_svd': 0.8584878623389438,
 'knn_basic': 0.9724861639611996,
 'knn_baseline': 0.8774229661900085,
 'knn_wm': 0.8967312758649276,
 'knn_wzs': 0.8922658679927045}
```

the knn_baseline model was the best of the 3 and just a little bit higher than our best svd model. We can try to tune that model to see if we can improve the performance.

KNNBaseline HyperTuning

In [56]:

```
#KNNBaseline with more parameters
knn_baseline_min_k_5 = KNNBaseline(min_k=5,si
model_process(knn_baseline_min_k_5,'knn_basel
```

```
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBaseline
on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fol
d 4  Fold 5  Mean    Std
RMSE (testset)    0.8637  0.8696  0.8626  0.8
611  0.8718  0.8658  0.0042
MAE (testset)     0.6618  0.6646  0.6600  0.6
610  0.6661  0.6627  0.0023
Fit time          0.35    0.36    0.39    0.3
8    0.32    0.36    0.02
Test time         1.40    1.40    1.38    1.3
5    1.35    1.37    0.02
```

Out[56]:

```
{'baseline_model': 0.8690048325664689,
 'best_svd': 0.8584878623389438,
 'knn_basic': 0.9724861639611996,
 'knn_baseline': 0.8774229661900085,
 'knn_wm': 0.8967312758649276,
 'knn_wzs': 0.8922658679927045,
 'knn_baseline_min_k_5': 0.8657669128835253}
```

In [57]:

```
#KNNBaseline with more parameters
knn_baseline_k_30 = KNNBaseline(min_k=30,sim_
model_process(knn_baseline_k_30,'knn_baseline
```

```
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBaseline
on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fol
d 4  Fold 5  Mean    Std
```

```
RMSE (testset)    0.8716  0.8579  0.8694  0.8
646  0.8662  0.8659  0.0047
MAE (testset)     0.6666  0.6602  0.6677  0.6
646  0.6658  0.6650  0.0026
Fit time          0.36    0.36    0.36    0.3
4    0.33    0.35    0.01
Test time         1.41    1.40    1.38    1.3
5    1.34    1.38    0.03
```

Out[57]:
```
{'baseline_model': 0.8690048325664689,
 'best_svd': 0.8584878623389438,
 'knn_basic': 0.9724861639611996,
 'knn_baseline': 0.8774229661900085,
 'knn_wm': 0.8967312758649276,
 'knn_wzs': 0.8922658679927045,
 'knn_baseline_min_k_5': 0.8657669128835253,
 'knn_baseline_k_30': 0.865934148375465}
```

These both slightly improved our RMSE. It may warrant taking the time to run a gridsearch with different values of k, min_k

GridSearchCV with KNNBaseline

In [58]:
```
#this will take a minute or so....
#params = {'k': [15,30,40],
          #'min_k': [1,3,5]}
#kb = KNNBaseline(sim_options = {'name': 'pea
#knnbaseline_gs = GridSearchCV(KNNBaseline,pa
#knnbaseline_gs.fit(data)
```

```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
```

```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
```

In [59]:

```
#Get the scores and best params
print(knnbaseline_gs.best_params)
print(knnbaseline_gs.best_score)
```

```
{'rmse': {'k': 30, 'min_k': 5}, 'mae': {'k':
40, 'min_k': 5}}
{'rmse': 0.865606545398092, 'mae': 0.66317525
06118159}
```

In [60]:

```
#best KNN Baseline model
knn_baseline_best = KNNBaseline(k=30,min_k=5,
```

```
model_process(knn_baseline_best,'knn_baseline
```

```
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBaseline
on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fol
d 4  Fold 5  Mean    Std
RMSE (testset)    0.8726  0.8684  0.8637  0.8
637  0.8639  0.8665  0.0035
MAE (testset)     0.6660  0.6639  0.6646  0.6
641  0.6607  0.6639  0.0017
Fit time          0.37    0.40    0.39    0.3
5    0.36    0.37    0.02
Test time         1.29    1.27    1.27    1.2
6    1.23    1.26    0.02
```

Out[60]:

```
{'baseline_model': 0.8690048325664689,
 'best_svd': 0.8584878623389438,
 'knn_basic': 0.9724861639611996,
 'knn_baseline': 0.8774229661900085,
 'knn_wm': 0.8967312758649276,
 'knn_wzs': 0.8922658679927045,
 'knn_baseline_min_k_5': 0.8657669128835253,
 'knn_baseline_k_30': 0.865934148375465,
 'knn_baseline_best': 0.8664549712431681}
```

Final Model Selection

Our best_svd model has the best results. Since there is not a lot of progress it is best to move on with this model.

We will use this model to make predicitons and reccomendations

Making Predicitons

Below will test code for predictions before building a function.

In [204…

```
best_svd.predict(12,12)
```

Out[204…

```
Prediction(uid=12, iid=12, r_ui=None, est=3.7
386642974645277, details={'was_impossible': F
alse})
```

In [158…

```
#making predictions for user 10 movie 1
pred = best_svd.predict(10,1)
pred
```

Out[158…

```
Prediction(uid=10, iid=1, r_ui=None, est=3.48
```

```
43803919078966, details={'was_impossible': Fa
lse})
```

In [159…

```
#use the movies dataframe to get actual movie
movies.head()
```

Out[159…

	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Children|Comec
1	2	Jumanji (1995)	Adventure|Childre
2	3	Grumpier Old Men (1995)	Comedy
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	

In [160…

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
 3   year     9742 non-null   int64 
dtypes: int64(2), object(2)
memory usage: 304.6+ KB
```

Displaying Movies

Need to use the movies dataframe to get the information for the movie. The surprise format gives us the the item id (iid) which can be matched to the index of our movies dataframe.

Use .loc to find the index that matches and return the title and genre

pred[0] - userId, pred[1] - index, pred[2] - actual user rating pred[3] - predicted rating

In [198…

```
#retrieve movie information from our predicti
title = movies.loc[pred[1]]['title']
genre = movies.loc[pred[1]]['genres']
pred_rating=round(pred[3],1)
print('Title: {}\nGenre: {}\nProjected Rating
```

```
Title: Jumanji (1995)
Genre: Adventure|Children|Fantasy
Projected Rating: 3.5
```

Find n Movie Reccomendations For Specific User

Below we will create a function that will return n movies for a specific user.

In [162…

```
#get number of items
dataset.n_items
```

Out[162…

```
9724
```

In [163…

```
#import heapq for getting top items from list
import heapq
#import regex
import re

#define reccomendation function
def n_movies_rec(user,model, movie_list,n=5):

    '''
    this will return n number of movies for a

    user - which user
    model - model to make the predictions
    n- number of reccomendations(default 5)
    movie_list -df of movies to pick from -
    '''
    #change display text based on user choice
    rec_title = 'YOUR CUSTOMIZED RECOMENDATIO
    if movie_list is movies:
        rec_title = 'STANDARD RECOMENDATIONS'

    display_movies = []
    #get the items from the dataset
    total_items = dataset.n_items


    #list for movie ratings
    movie_ratings = []


    #create a list of movieId's
    ids_list = pd.unique(movie_list['movieId'


    #populate movie ratings for raw list
```

```
    for item in range(total_items):
        #append the movie to the movie_rating
        movie_ratings.append(model.predict(us

    #remove any movies from movie_ratings not
    final_ratings=[]

    for mov in movie_ratings:
        mov_id = movies.iloc[mov[1]]['movieId
        if mov_id in ids_list:
            #add movie to the final_ratings
            final_ratings.append(mov)


    print('final ratings include {} ratings c
    print('----------------------------------
    print('----------------------------------
    print(' ')
    print(rec_title)
    print(' ')

    #use heapq.nlagerst to get the N highest
    raw_list = heapq.nlargest(n,final_ratings
    # get movie info from movies dataframe

    for p in raw_list:
        title = movies.iloc[p[1]]['title']
        genre = movies.iloc[p[1]]['genres']
        pred_rating=round(p[3],1)
        print('Title: {}\nGenre: {}\nProjecte
        print('\n')
    return None

#test the function for 10 movies for user 110
n_movies_rec(110,best_svd, movies, 10)
```

```
final ratings include 9724 ratings out of 972
4
---------------------------------------------
----
---------------------------------------------
----

STANDARD RECOMENDATIONS

Title: My Best Friend's Wedding (1997)
Genre: Comedy|Romance
Projected Rating: 4.4


Title: Marat/Sade (1966)
Genre: Drama|Musical
Projected Rating: 4.4


Title: Virtuosity (1995)
Genre: Action|Sci-Fi|Thriller
Projected Rating: 4.4
```

```
Title: RocketMan (a.k.a. Rocket Man) (1997)
Genre: Children|Comedy|Romance|Sci-Fi
Projected Rating: 4.4


Title: I Love Trouble (1994)
Genre: Action|Comedy
Projected Rating: 4.4


Title: Escape from New York (1981)
Genre: Action|Adventure|Sci-Fi|Thriller
Projected Rating: 4.4


Title: Cement Garden, The (1993)
Genre: Drama
Projected Rating: 4.3


Title: Withnail & I (1987)
Genre: Comedy
Projected Rating: 4.3


Title: Amistad (1997)
Genre: Drama|Mystery
Projected Rating: 4.3


Title: Hoodlum (1997)
Genre: Crime|Drama|Film-Noir
Projected Rating: 4.3
```

In [164…

```
#try out another user to make sure this is wo
n_movies_rec(8,best_svd,movies, 10)
```

```
final ratings include 9724 ratings out of 972
4
--------------------------------------------
----
--------------------------------------------
----

STANDARD RECOMENDATIONS

Title: I Love Trouble (1994)
Genre: Action|Comedy
Projected Rating: 4.5


Title: Marat/Sade (1966)
Genre: Drama|Musical
Projected Rating: 4.5


Title: Cement Garden, The (1993)
Genre: Drama
```

```
Projected Rating: 4.5


Title: Escape from New York (1981)
Genre: Action|Adventure|Sci-Fi|Thriller
Projected Rating: 4.4


Title: My Best Friend's Wedding (1997)
Genre: Comedy|Romance
Projected Rating: 4.4


Title: Lady Vengeance (Sympathy for Lady Veng
eance) (Chinjeolhan geumjassi) (2005)
Genre: Crime|Drama|Mystery|Thriller
Projected Rating: 4.4


Title: Star Wars: Episode V - The Empire Stri
kes Back (1980)
Genre: Action|Adventure|Sci-Fi
Projected Rating: 4.4


Title: Uncle Buck (1989)
Genre: Comedy
Projected Rating: 4.4


Title: Indian Summer (a.k.a. Alive & Kicking)
(1996)
Genre: Comedy|Drama
Projected Rating: 4.4


Title: Pompatus of Love, The (1996)
Genre: Comedy|Drama
Projected Rating: 4.4
```

This will give us reccomendations for existing users. We will then need to add by category and beable to create a new user and provide results

Obtain New User Ratings

We will create a function to obtain new user ratings. We will want to gather at least 5 ratings in order to make reccomendations from our model. The user will be given movies to rate on a scale of 0.5 to 5. They should have the option to skip a movie if they have not seen it.

- use survey_movies dataframe for this function

- use survey_movies dataframe for this function. This should make it more likely that the new user has seen the movie

We need to be able to do the following:

- get new user ratings
- add ratings to the ratings 'data'
- read the data into a suprise dataset
- train the model
- return the predicitons (5)

We will create funcitons to work through this process and then create a master function to complete the entire process.

Get New User Ratings

We need to use the original rate_df to add the ratings of the new user.

In [165…

```
rate_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   userId   100836 non-null  int64  
 1   movieId  100836 non-null  int64  
 2   rating   100836 non-null  float64
dtypes: float64(1), int64(2)
memory usage: 2.3 MB
```

In [166…

```
np.max(rate_df.userId)
```

Out[166…

```
610
```

In [167…

```
sm_df = survey_movies.drop('rating',axis=1)
```

In [168…

```
sm_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 1235 entries, (1, 'Toy Story (199
5)', 'Adventure|Animation|Children|Comedy|Fan
tasy') to (168252, 'Logan (2017)', 'Action|Sc
i-Fi')
Empty DataFrame
```

In [169…

```
sm_df = sm_df.reset_index()
```

In [170…

```
sm_df
```

Out[170…

	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Children|Co
1	2	Jumanji (1995)	Adventure|Ch
2	3	Grumpier Old Men (1995)	Com
3	5	Father of the Bride Part II (1995)	
4	6	Heat (1995)	Action
...	...	...	
1230	148626	Big Short, The (2015)	
1231	152081	Zootopia (2016)	Action|Adventure|Animation|Chi
1232	164179	Arrival (2016)	
1233	166528	Rogue One: A Star Wars Story (2016)	Action|Adventure|
1234	168252	Logan (2017)	

1235 rows × 3 columns

New User Rating Function

The new user rating function selects movies to be rated and returns a new dataframe of that users ratings.

In [171…

```
def new_user_ratings(movies,n,genre=None):
    '''
    movies - selected movie dataframe
    n - (int) number of movies
```

```
    n - (int) number of movies
    genre- (str) can specify a genre to furth

    '''

    #narrow down the movies if genre is selec
    to_be_rated = movies
    if genre:
        to_be_rated = movies[movies['genres']

    #create new user id one higher than the m
    user_id = np.max(rate_df.userId)+1

    #list to store the new ratings
    ratings = []

    #use while loop to get n ratings from our
    while n > 0:
        # select a sample movie
        movie = to_be_rated.sample(1)
        #clean up the presentation of the mov
        title = movie.title.to_string()
        title = re.sub("[^a-zA-Z0-9(),'']+",
        print(title)
        rating = input("Rate the movie from 1
        # make sure user enters an acceptable
        if rating not in ['1','2','3','4','5'
            continue
        else:
            #need to use column names from ra
            rated = {'userId':user_id,'movieI
            ratings.append(rated)
            n -=1

    return pd.DataFrame(ratings)
```

In [205…

```
#test out the ratings
new_ratings = new_user_ratings(sm_df,5)
new_ratings
```

```
845 Amelie (Fabuleux destin d'Am lie Poulain,
Le)
Rate the movie from 1-5.  enter 'x' if you ha
ven't seen the film 1
263 Ghost and the Darkness, The (1996)
Rate the movie from 1-5.  enter 'x' if you ha
ven't seen the film 4
468 Last Emperor, The (1987)
Rate the movie from 1-5.  enter 'x' if you ha
ven't seen the film 3
609 Tarzan (1999)
Rate the movie from 1-5.  enter 'x' if you ha
ven't seen the film 2
443 Wedding Singer, The (1998)
Rate the movie from 1-5.  enter 'x' if you ha
ven't seen the film 5
```

Out[205…

	userId	movieId	rating
0	619	4973	1

1	619	1049	4
2	619	1960	3
3	619	2687	2
4	619	1777	5

Add Ratings to the Original Ratings Data

We want to add these to rate_df

In [173…

```
rate_df.head()
```

Out[173…

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0

Add Ratings Function

This functions will add the new user ratings to the original rating data.

It will return the data in both surprise and pandas form.

In [174…

```
#define function to add movies to the Data -
def add_ratings(new_rate, existing_data):
    existing_data = pd.concat([new_rate, exis
    updated_ratings = Dataset.load_from_df(ex
    rate_df = existing_data
    #return both the dataframe and surprise c
    return existing_data,updated_ratings
```

Fit Model

best_svd = SVD(n_factors=60, reg_all=0.075, lr_all=0.01, random_state=42)

use the new add_ratings function to fit the the model

In [175…

```
#fit the model using the add_ratings functior
best_svd = SVD(n_factors=60, reg_all=0.075, l
ratings,surprise_ratings = add_ratings(new_ra
best_svd.fit(surprise_ratings.build_full_trai
```

Out[175…

```
<surprise.prediction_algorithms.matrix_factor
ization.SVD at 0x7fd3dc1d7760>
```

In [84]:

```
ratings.describe()
```

Out[84]:

	userId	movieId
count	100841.000000	100841.000000
mean	326.141688	19434.454547
std	182.624980	35530.307623
min	1.000000	1.000000
25%	177.000000	1199.000000
50%	325.000000	2991.000000
75%	477.000000	8121.000000
max	611.000000	193609.000000

Return Predictions

use 'n_movies_rec' function to return 5 movies.

In [176…

```
n_movies_rec(np.max(ratings.userId),best_svd,
```

```
final ratings include 9724 ratings out of 972
4
---------------------------------------------
----
---------------------------------------------
----

STANDARD RECOMENDATIONS

Title: Marat/Sade (1966)
Genre: Drama|Musical
Projected Rating: 4.5


Title: I Love Trouble (1994)
Genre: Action|Comedy
Projected Rating: 4.5


Title: Cement Garden, The (1993)
Genre: Drama
Projected Rating: 4.4


Title: Quiz Show (1994)
```

```
Title: Quiz Show (1994)
Genre: Drama
Projected Rating: 4.4


Title: Barefoot Contessa, The (1954)
Genre: Drama
Projected Rating: 4.4
```

Put it all Together

Function to get some user preferences. This is being added to narrow down the types of movies being reccomended. We can ask the user a few questions to limit the pool of movies using year and genre. the function should return a dataframe that can be used in the user_survey() function

In [177…

```
movies[movies['year']>1980]
```

Out[177…

	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Children|Cc
1	2	Jumanji (1995)	Adventure|Ch
2	3	Grumpier Old Men (1995)	Com
3	4	Waiting to Exhale (1995)	Comedy|Dr
4	5	Father of the Bride Part II (1995)	
...	...	...	
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Cc
9738	193583	No Game No Life: Zero (2017)	Animation|Cc
9739	193585	Flint (2017)	

9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Ac
9741	193609	Andrew Dice Clay: Dice Rules (1991)	

8092 rows × 4 columns

Genre List

need to make a list of genres that users can select or eliminate

In [178…

```
##go through genres column and create a list
#list to hold the genres
genre_list = []
#function to get all the genres separated and
def get_genre(row):
    words = row.split('|')
    for w in words:
        w = w.lower()
        if w not in genre_list:
            genre_list.append(w)

#lambda function to get the entire dataframe
movies['genres'].map(lambda x: get_genre(x))
genre_list.remove('(no genres listed)')
genre_list
```

Out[178…

```
['adventure',
 'animation',
 'children',
 'comedy',
 'fantasy',
 'romance',
 'drama',
 'action',
 'crime',
 'thriller',
 'horror',
 'mystery',
 'sci-fi',
 'war',
 'musical',
 'documentary',
 'imax',
 'western',
 'film-noir']
```

Questionnaire

The questionnaire will allow the user to limit the range of years. -- may need to build in something to prevent bad inputs and make sure that the user cannot limit the data beyound the output of 5 movies.

Then the questionnaire will allow the user to include only certain genres.

In [179…

```
def questionnaire():
    """
    this funciton will take the movies datafr
    questions.

    """
    # year  -  change the range of the movies
    print('Our movie library contains films f
    year_range=input("Type 'yes' if you would

    if year_range == 'yes':
        oldest = int(input('Enter the first y
        newest = int(input('Enter the last ye
        custom_movies = movies[(movies['year'
    else:
        custom_movies = movies

    # genres -  limit the genres included in

    print(' ')
    print('Our movie library contains films f
    print(' ')
    print(' ')

    print(genre_list)

    #ask if they want to modify
    limit_genre = input("Type 'yes' if you wo

    mod_genre = []

    # check if they want to limit genres
    if limit_genre == 'yes':
        print('')
        print("Enter any genres that you woul
        print("leave blank and hit enter to s

        #continue looping until they don't en
        while True:
            word = input()
            if word:
                mod_genre.append(word.lower()
            else:
                break
```

```
#change custom movies to remove movies i

    custom_movies = custom_movies[custom_
        lambda x: any(substring in x.lowe

return custom_movies
```