1 Movie Recommendation System

- Student name: Vi Bui
- Student pace: Part-Time
- Scheduled project review date/time: Thur. 03/03/22 (Non-Technical); Tues. 03/08/22 (Technical)
- Instructor name: Claude Fried
- Blog post URL: [https://datasciish.com/ (https://datasciish.com/)](https://datasciish.com/)

1.1 Overview [...]

Vi(sion) Studios, a new streaming service, is looking to launch a concept called "Digital Cinema Night" where customers can build a "Cinema Night" around specific movies. They've hired us to build a Recommendation System in order to launch this concept.

Data, Methodology, and Analysis: we've explored data from MovieLens which captures Movies, Ratings, Genres, and Year. For this analysis, we used all metrics available with the exception of Tags which may be used for later analysis.

Results & Recommendations: After analyzing data from databases with movies heavily weighted in the 1990s through 2000s, we've built a recommendation system for Vi(sion) Studios' "Digital Cinema Night."

1.2 Business Objective

Build a model that provides top 5 movie recommendations to a user for the best Digital Cinema Night experience.

1.3 Data Overview

MovieLens Dataset

Data explored:

** denotes data used in current analysis
^ denotes data for future analysis

1. MovieLens - Movies**
2. MovieLens - Ratings**
3. MovieLens - Tags^
4. MovieLens - Links (will not be used)

2 Data Exploration, Cleansing, and Preparation

Data Exploration

- Outlined in comments: files explored, how and/or why data was chosen, which files will be used, and which files will be used in future analysis

Data Cleansing

- Checked for duplicates, NaN values; continuously cleansed data as necessary

Data Preparation

- Core variables: Movie Titles, Ratings, Year, Genre
- Merged Movies and Ratings datasets

In [1]:
```
# import libraries
# Surprise documentation - Surprise: https://surprise.readthedocs.io/en/sta

import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import train_test_split
from surprise.model_selection import cross_validate
from surprise.model_selection import GridSearchCV
from surprise.model_selection import KFold
from surprise import NormalPredictor
from surprise import NormalPredictor
from surprise import BaselineOnly
from surprise import KNNBasic
from surprise import KNNWithMeans
from surprise import KNNBaseline
from surprise import SVD
from surprise import SVDpp
from surprise import NMF
from surprise import SlopeOne
from surprise import CoClustering

%matplotlib inline
```

executed in 806ms, finished 04:49:14 2022-03-14

2.1 Data Exploration and Cleansing

In [2]:
```
# explore movies data

movies = pd.read_csv('movielens/movies.csv')
movies
```
executed in 18ms, finished 04:49:14 2022-03-14

Out[2]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

9742 rows × 3 columns

In [3]:
```
# explore ratings data

ratings = pd.read_csv('movielens/ratings.csv')
ratings.head()
```
executed in 31ms, finished 04:49:14 2022-03-14

Out[3]:

	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 [4]:
```
# explore links data - will not be used

links = pd.read_csv('movielens/links.csv')
links.head()
```
executed in 10ms, finished 04:49:14 2022-03-14

Out[4]:

	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 [5]:
```
# explore tags data - will not be used for this analysis

tags = pd.read_csv('movielens/tags.csv')
tags.head()
```
executed in 9ms, finished 04:49:14 2022-03-14

Out[5]:

	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

After exploring all datasets, Movies and Ratings datasets - which provides titles, ratings, userid, and genre - will be used

Movies and Ratings datasets will be merged

In [6]:
```
# merge movies and ratings data on movieId column

movies_and_ratings = pd.merge(left=movies,
                              right=ratings,
                              on='movieId')
```
executed in 17ms, finished 04:49:14 2022-03-14

In [7]:
```
# explore movies and ratings data

movies_and_ratings.info()
```
executed in 16ms, finished 04:49:14 2022-03-14

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100836 entries, 0 to 100835
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   movieId    100836 non-null  int64  
 1   title      100836 non-null  object 
 2   genres     100836 non-null  object 
 3   userId     100836 non-null  int64  
 4   rating     100836 non-null  float64
 5   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3), object(2)
memory usage: 5.4+ MB
```

In [8]:
```
# check for duplicates

movies_and_ratings.duplicated(keep='first').sum()
```
executed in 28ms, finished 04:49:14 2022-03-14

Out[8]:
```
0
```

There are zero duplicates!

In [9]:
```
# check for NaN values

movies_and_ratings.isna().sum()
```
executed in 11ms, finished 04:49:14 2022-03-14

Out[9]:
```
movieId      0
title        0
genres       0
userId       0
rating       0
timestamp    0
dtype: int64
```

There are zero NaN values!

In [10]:
```
# check number of unique MovieIds

movies_and_ratings['movieId'].nunique()
```
executed in 3ms, finished 04:49:14 2022-03-14

Out[10]:
```
9724
```

In [11]:
```
# check number of unique userIds

movies_and_ratings['userId'].nunique()
```
executed in 3ms, finished 04:49:14 2022-03-14

Out[11]: 610

The dataset contains 9724 unique entries (based on movieId) and 610 unique users. This is a solid start for our dataset.

In [12]:
```
# check dataset columns

movies_and_ratings.columns
```
executed in 3ms, finished 04:49:14 2022-03-14

Out[12]:
```
Index(['movieId', 'title', 'genres', 'userId', 'rating', 'timestamp'], dt
ype='object')
```

In [13]:
```
# add column for year
# parse out year using indexes (-5:-1)
# create a placeholder (1000000) for movies that do not have a year

year = []

for movie in movies_and_ratings['title']:
    year_separated = movie[-5:-1]
    try: year.append(int(year_separated))
    except: year.append(1000000)

movies_and_ratings['year'] = year
movies_and_ratings.head()
```
executed in 60ms, finished 04:49:14 2022-03-14

Out[13]:

	movieId	title	genres	userId	rating	timestamp	year
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1	4.0	964982703	1995
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	847434962	1995
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	7	4.5	1106635946	1995
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	15	2.5	1510577970	1995
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	17	4.5	1305696483	1995

In [14]:
```
# determine how many movies did not have a year in the title

print(len(movies_and_ratings[movies_and_ratings['year'] == 1000000]))
```
executed in 7ms, finished 04:49:14 2022-03-14

```
30
```

In [15]:
```
# eliminate the 30 titles that did not have a year attached

movies_and_ratings = movies_and_ratings.loc[movies_and_ratings['year']
                                             != 1000000]
```
executed in 7ms, finished 04:49:14 2022-03-14

In [16]:
```
# clean movie titles using indexes to trim the last 6 characters of each ti

movies_and_ratings['title'] = movies_and_ratings['title'].str[:-7]
movies_and_ratings['title']
```
executed in 31ms, finished 04:49:14 2022-03-14

Out[16]:
```
0                                   Toy Story
1                                   Toy Story
2                                   Toy Story
3                                   Toy Story
4                                   Toy Story
                         ...
100831    Black Butler: Book of the Atlantic
100832                 No Game No Life: Zero
100833                                 Flint
100834           Bungo Stray Dogs: Dead Apple
100835           Andrew Dice Clay: Dice Rules
Name: title, Length: 100806, dtype: object
```

In [17]:
```
# look at info for cleaned dataset

movies_and_ratings.info()
```
executed in 13ms, finished 04:49:14 2022-03-14

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100806 entries, 0 to 100835
Data columns (total 7 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   movieId    100806 non-null  int64  
 1   title      100806 non-null  object 
 2   genres     100806 non-null  object 
 3   userId     100806 non-null  int64  
 4   rating     100806 non-null  float64
 5   timestamp  100806 non-null  int64  
 6   year       100806 non-null  int64  
dtypes: float64(1), int64(4), object(2)
memory usage: 6.2+ MB
```

In [18]:
```
movies_and_ratings.head()
```
executed in 8ms, finished 04:49:14 2022-03-14

Out[18]:

	movieId	title	genres	userId	rating	timestamp	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1	4.0	964982703	1995
1	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	847434962	1995
2	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	7	4.5	1106635946	1995
3	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	15	2.5	1510577970	1995
4	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	17	4.5	1305696483	1995

In [19]:
```
# drop timestamp column

movies_and_ratings = movies_and_ratings.drop(columns=['timestamp'])
movies_and_ratings
```
executed in 15ms, finished 04:49:14 2022-03-14

Out[19]:

	movieId	title	genres	userId	rating	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1	4.0	1995
1	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	1995
2	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	7	4.5	1995
3	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	15	2.5	1995
4	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	17	4.5	1995
...	...	...	...	...	...	...
100831	193581	Black Butler: Book of the Atlantic	Action|Animation|Comedy|Fantasy	184	4.0	2017
100832	193583	No Game No Life: Zero	Animation|Comedy|Fantasy	184	3.5	2017
100833	193585	Flint	Drama	184	3.5	2017
100834	193587	Bungo Stray Dogs: Dead Apple	Action|Animation	184	3.5	2018
100835	193609	Andrew Dice Clay: Dice Rules	Comedy	331	4.0	1991

100806 rows × 6 columns

In [20]:
```
# explore years in dataset - groupby year and title

movie_yr_count = movies_and_ratings.groupby('year').count()['title'
                                                            ].reset_index()

movie_yr_count.head()
```

executed in 18ms, finished 04:49:14 2022-03-14

Out[20]:

	year	title
0	1902	5
1	1903	2
2	1908	1
3	1915	1
4	1916	5

2.2 Data Visualizations

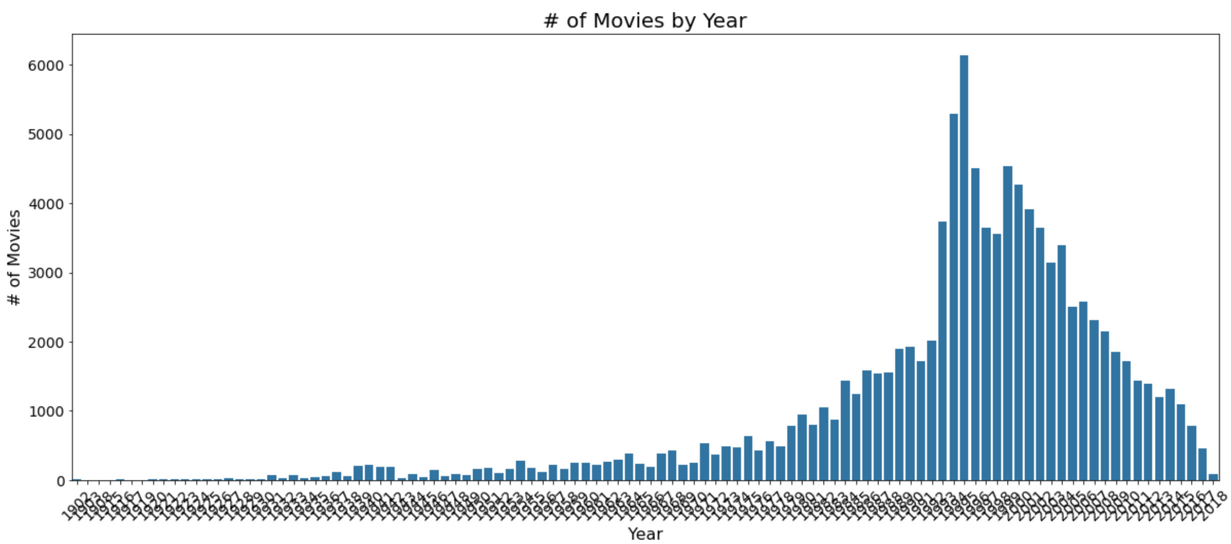
In [21]:
```
# plot the years - countplot

fig, ax = plt.subplots(figsize=(20,8))

sns.countplot(x='year',
              data=movies_and_ratings,
              color='tab:blue');

plt.xlabel("Year", fontsize=16)
plt.ylabel("# of Movies", fontsize=16)
plt.title("# of Movies by Year",fontsize=20)
plt.xticks(fontsize=14, rotation=45)
plt.yticks(fontsize=14)
ax.grid(False)
plt.show()
```

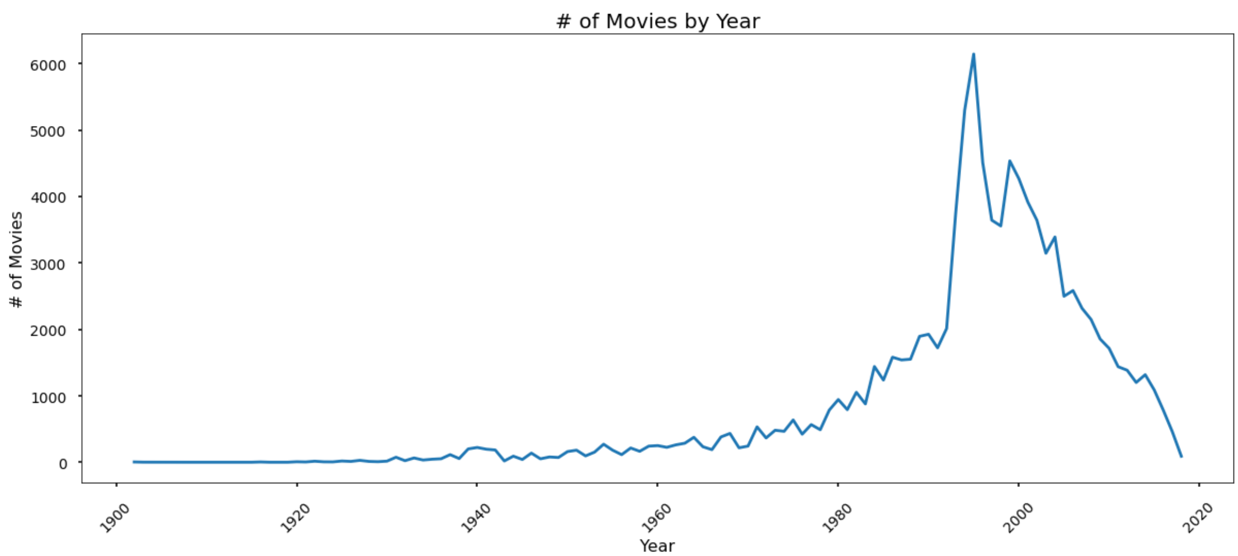
executed in 1.22s, finished 04:49:15 2022-03-14



In [22]:
```
# plot the years - lineplot

with plt.style.context('seaborn-poster'):
    fig, ax = plt.subplots(figsize=(20,8))
    sns.lineplot(x='year',
                 y = 'title',
                 data=movie_yr_count,
                 color='tab:blue');

    plt.xlabel("Year", fontsize=16)
    plt.ylabel("# of Movies", fontsize=16)
    plt.title("# of Movies by Year",fontsize=20)
    plt.xticks(fontsize=14, rotation=45)
    plt.yticks(fontsize=14)
    ax.grid(False)
    plt.show()
```
executed in 128ms, finished 04:49:15 2022-03-14



In [23]:
```
# continue exploring the years in movie dataset
# use (normalize=True) so you do not need to do what you did in cell below

movies_and_ratings['year'].value_counts(normalize=True)*100
```
executed in 5ms, finished 04:49:15 2022-03-14

Out[23]:
```
1995    6.093883
1994    5.253656
1999    4.498740
1996    4.472948
2000    4.233875
          ...
1903    0.001984
1908    0.000992
1915    0.000992
1919    0.000992
1917    0.000992
Name: year, Length: 106, dtype: float64
```

In [24]:
```
# reference of what not to do
# calculate % of movies by year

movies_years_composition = (
    movies_and_ratings['year'].value_counts() /
    len(movies_and_ratings['year']))*100

movies_years_composition
```
executed in 6ms, finished 04:49:15 2022-03-14

Out[24]:
```
1995    6.093883
1994    5.253656
1999    4.498740
1996    4.472948
2000    4.233875
          ...
1903    0.001984
1908    0.000992
1915    0.000992
1919    0.000992
1917    0.000992
Name: year, Length: 106, dtype: float64
```

In [25]:
```
# look at standard metrics for ratings in dataset

movies_and_ratings['rating'].describe()
```
executed in 9ms, finished 04:49:15 2022-03-14

Out[25]:
```
count    100806.000000
mean          3.501592
std           1.042414
min           0.500000
25%           3.000000
50%           3.500000
75%           4.000000
max           5.000000
Name: rating, dtype: float64
```

In [26]:

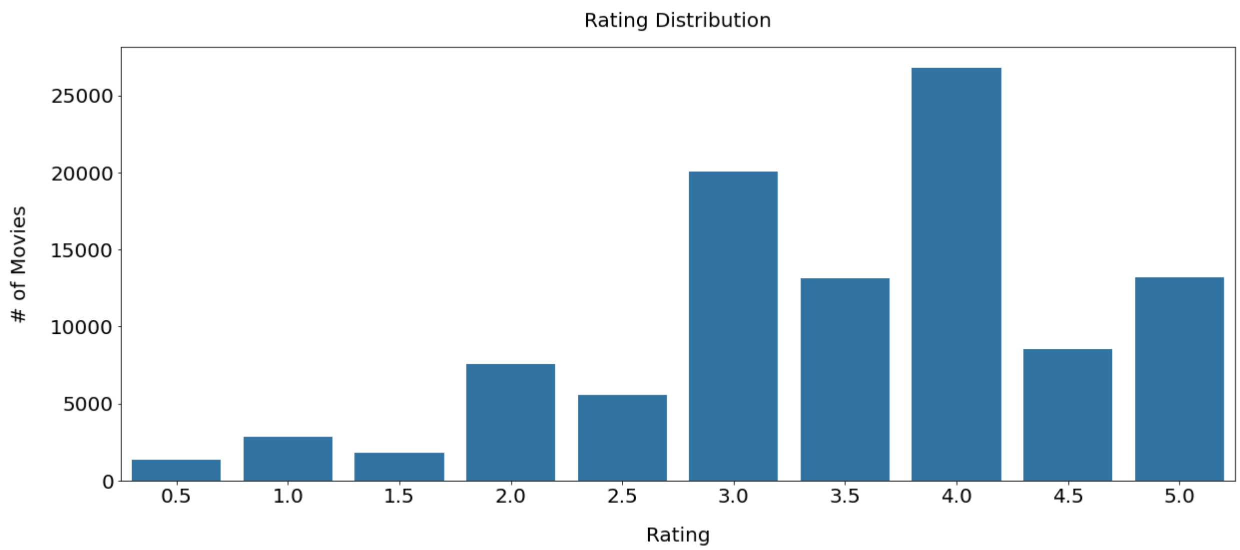
```
# plot number of ratings

fig, ax = plt.subplots(figsize=(20,8))

sns.countplot(x='rating',
              data=movies_and_ratings,
              color='tab:blue');

plt.title("Rating Distribution",fontsize=20, pad=20)
plt.xlabel("Rating", fontsize=20, labelpad=20)
plt.ylabel("# of Movies", fontsize=20, labelpad=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
ax.grid(False)
plt.show()
```

executed in 137ms, finished 04:49:15 2022-03-14



In [27]:

```
# CLEAN GENRE DATA
# separate genres from each other and see how many genres there are

genre_df = pd.DataFrame(movies_and_ratings['genres'].
                        str.split('|').
                        tolist(),
                        index=movies_and_ratings['movieId']
                       ).stack()

# reset index

genre_df = genre_df.reset_index([0, 'movieId'])

# create columns movieId and genre

genre_df.columns = ['movieId', 'genre']
```

executed in 243ms, finished 04:49:16 2022-03-14

In [28]:
```
# explore genre data

genre_df.head()
```
executed in 5ms, finished 04:49:16 2022-03-14

Out[28]:

	movieId	genre
0	1	Adventure
1	1	Animation
2	1	Children
3	1	Comedy
4	1	Fantasy

In [29]:
```
# explore genre value counts

genre_df['genre'].value_counts()
```
executed in 39ms, finished 04:49:16 2022-03-14

Out[29]:
```
Drama                 41923
Comedy                39049
Action                30623
Thriller              26446
Adventure             24157
Romance               18124
Sci-Fi                17233
Crime                 16679
Fantasy               11831
Children               9207
Mystery                7674
Horror                 7287
Animation              6982
War                    4858
IMAX                   4145
Musical                4138
Western                1930
Documentary            1219
Film-Noir               870
(no genres listed)       38
Name: genre, dtype: int64
```

In [30]:
```
# genre data data to explore most popular genres

genre_df_sorted = (
    genre_df['genre']
    .value_counts()
    .sort_values(ascending=False)
    .reset_index()
    )

genre_df_sorted
```

executed in 33ms, finished 04:49:16 2022-03-14

Out[30]:

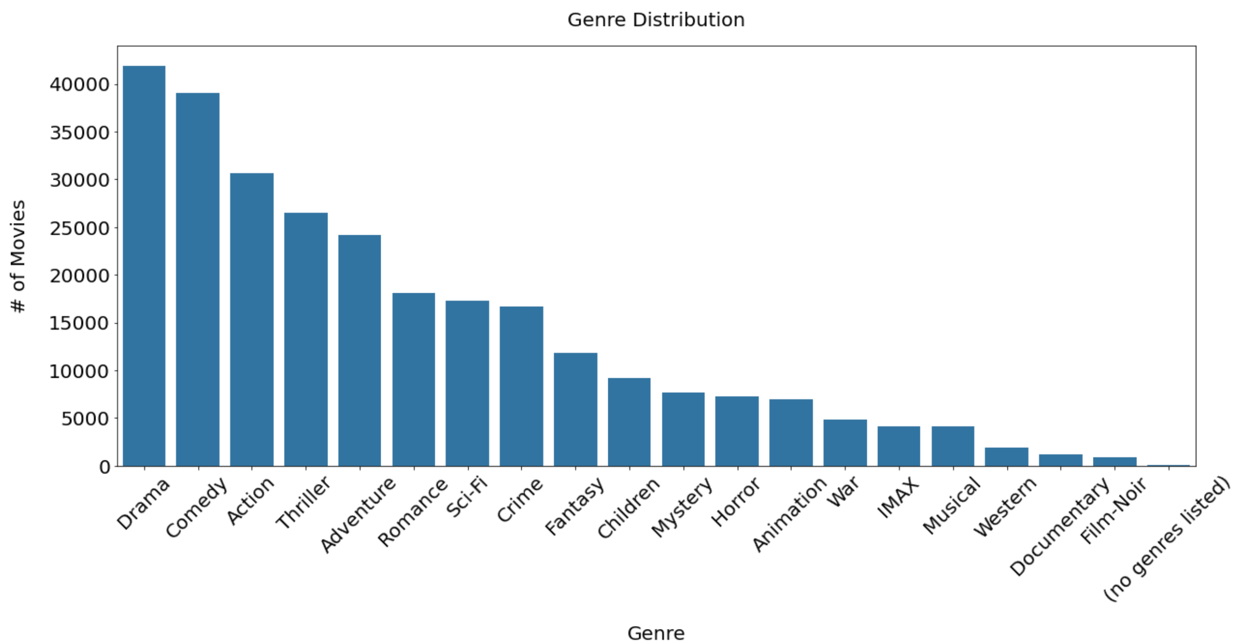
	index	genre
0	Drama	41923
1	Comedy	39049
2	Action	30623
3	Thriller	26446
4	Adventure	24157
5	Romance	18124
6	Sci-Fi	17233
7	Crime	16679
8	Fantasy	11831
9	Children	9207
10	Mystery	7674
11	Horror	7287
12	Animation	6982
13	War	4858
14	IMAX	4145
15	Musical	4138
16	Western	1930
17	Documentary	1219
18	Film-Noir	870
19	(no genres listed)	38

In [31]:
```
# plot value counts by genre

fig, ax = plt.subplots(figsize=(20,8))

sns.barplot(x='index',
            y='genre',
            data=genre_df_sorted,
            color='tab:blue');

plt.title("Genre Distribution",fontsize=20, pad=20)
plt.xlabel("Genre", fontsize=20, labelpad=20)
plt.ylabel("# of Movies", fontsize=20, labelpad=20)
plt.xticks(fontsize=20, rotation=45)
plt.yticks(fontsize=20)
ax.grid(False)
plt.show()
```
executed in 239ms, finished 04:49:16 2022-03-14



In [32]:
```
# create movie_popularity variable using value counts

movie_popularity = movies_and_ratings["title"].value_counts()

# create variable "popular_movies" for only movies appearing > 50 times
popular_movies = movie_popularity[movie_popularity > 50]
```
executed in 14ms, finished 04:49:16 2022-03-14

In [33]:
```
# explore movie_popularity statistics

movie_popularity.describe()
```
executed in 6ms, finished 04:49:16 2022-03-14

Out[33]:
```
count    9423.000000
mean       10.697867
std        22.837215
min         1.000000
25%         1.000000
50%         3.000000
75%         9.000000
max       329.000000
Name: title, dtype: float64
```

In [34]:
```
# explore popular_movies statistics

popular_movies.describe()
```
executed in 5ms, finished 04:49:16 2022-03-14

Out[34]:
```
count    444.000000
mean      93.189189
std       45.947219
min       51.000000
25%       61.000000
50%       77.000000
75%      108.250000
max      329.000000
Name: title, dtype: float64
```

In [35]:
```
# create a column called 'RatingsCounts' - number of ratings for each movie

num_ratings = pd.DataFrame(
    movies_and_ratings
    .groupby('movieId')
    .count()['rating']
    ).reset_index()

movies_and_ratings = pd.merge(left=movies_and_ratings,
                              right=num_ratings,
                              on='movieId')

movies_and_ratings.rename(columns={'rating_x': 'rating',
                                   'rating_y': 'RatingsCount'},
                          inplace=True)
```
executed in 65ms, finished 04:49:16 2022-03-14

In [36]:
```
movies_and_ratings.info()
```
executed in 17ms, finished 04:49:16 2022-03-14

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100806 entries, 0 to 100805
Data columns (total 7 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   movieId       100806 non-null  int64  
 1   title         100806 non-null  object 
 2   genres        100806 non-null  object 
 3   userId        100806 non-null  int64  
 4   rating        100806 non-null  float64
 5   year          100806 non-null  int64  
 6   RatingsCount  100806 non-null  int64  
dtypes: float64(1), int64(4), object(2)
memory usage: 6.2+ MB
```

In [37]:
```
# sort by ratings counts

movies_and_ratings_sorted = (
    movies_and_ratings
    .sort_values(by='RatingsCount', ascending=False)
    .drop_duplicates('movieId')
    )

movies_and_ratings_sorted
```
executed in 27ms, finished 04:49:16 2022-03-14

Out[37]:

	movieId	title	genres	userId	rating	year	RatingsCount
10332	356	Forrest Gump	Comedy|Drama|Romance|War	589	5.0	1994	329
8860	318	Shawshank Redemption, The	Crime|Drama	400	5.0	1994	317
7886	296	Pulp Fiction	Comedy|Crime|Drama|Thriller	45	5.0	1994	307
16321	593	Silence of the Lambs, The	Crime|Horror|Thriller	201	5.0	1991	279
45053	2571	Matrix, The	Action|Sci-Fi|Thriller	80	4.5	1999	278
...	...	...	...	...	...	...	...
79447	32442	Greedy	Comedy	599	2.5	1994	1
79446	32440	If Looks Could Kill	Action|Comedy	599	2.0	1991	1
79445	32392	800 Bullets (800 Balas)	Comedy|Crime|Drama|Western	387	2.5	2002	1
97640	115667	Love, Rosie	Comedy|Romance	563	3.5	2014	1
100805	193609	Andrew Dice Clay: Dice Rules	Comedy	331	4.0	1991	1

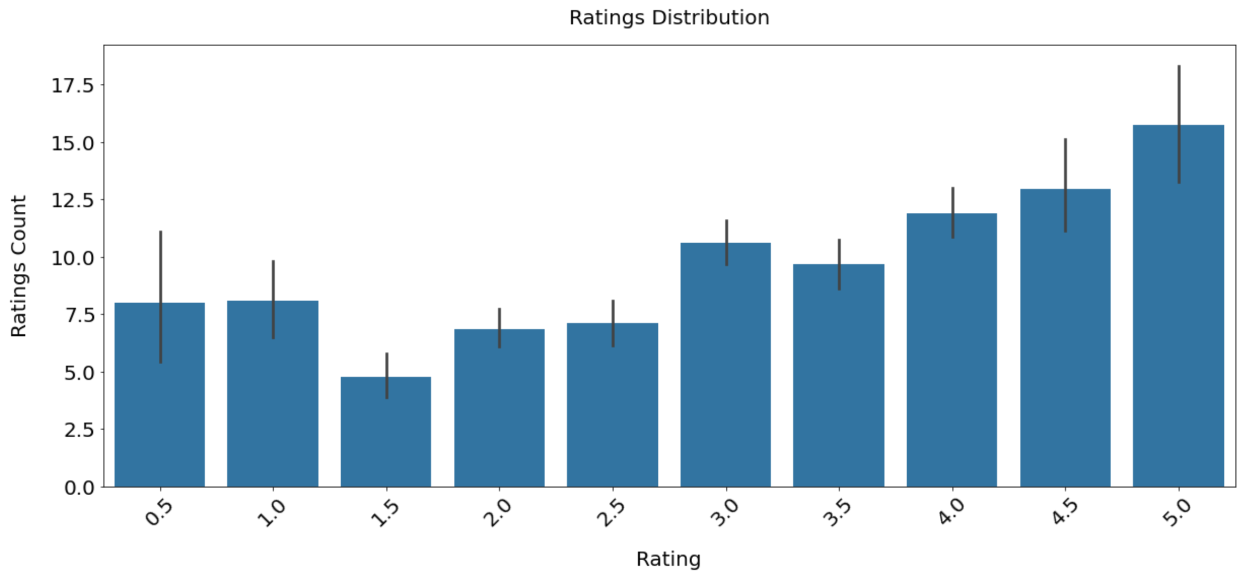
9701 rows × 7 columns

In [38]:
```
# plot value counts by RatingsCount

fig, ax = plt.subplots(figsize=(20,8))

sns.barplot(x='rating',
            y='RatingsCount',
            data=movies_and_ratings_sorted,
            color='tab:blue');

plt.title("Ratings Distribution",fontsize=20, pad=20)
plt.xlabel("Rating", fontsize=20, labelpad=20)
plt.ylabel("Ratings Count", fontsize=20, labelpad=20)
plt.xticks(fontsize=20, rotation=45)
plt.yticks(fontsize=20)
ax.grid(False)
plt.show()
```
executed in 382ms, finished 04:49:17 2022-03-14



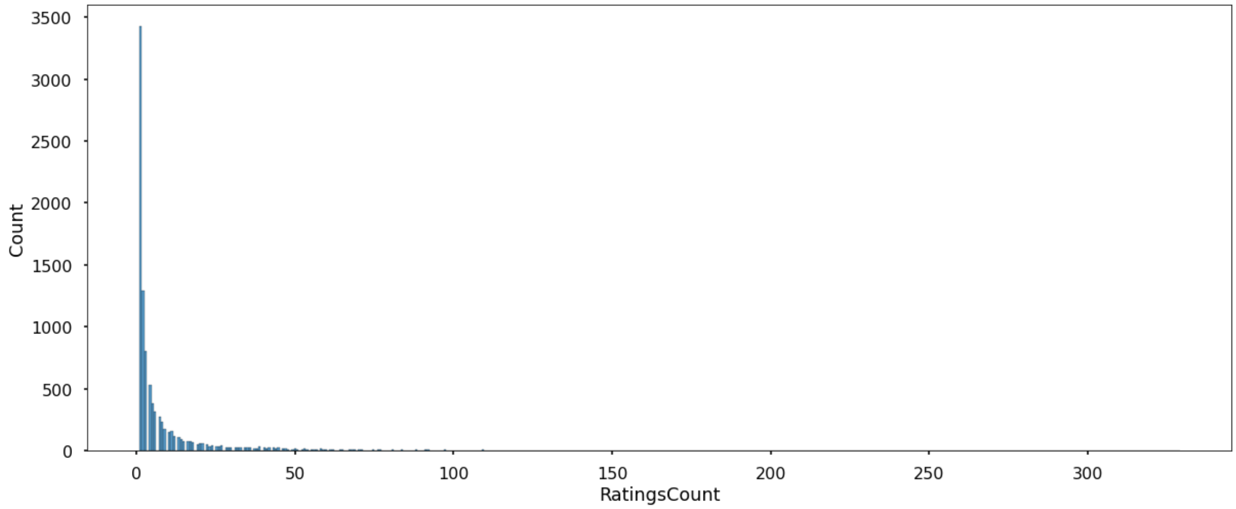
In [39]:
```
# explore latest dataset

movies_and_ratings_sorted.info()
```
executed in 9ms, finished 04:49:17 2022-03-14

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9701 entries, 10332 to 100805
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   movieId       9701 non-null   int64  
 1   title         9701 non-null   object 
 2   genres        9701 non-null   object 
 3   userId        9701 non-null   int64  
 4   rating        9701 non-null   float64
 5   year          9701 non-null   int64  
 6   RatingsCount  9701 non-null   int64  
dtypes: float64(1), int64(4), object(2)
memory usage: 926.3+ KB
```

In [40]:
```
# plot ratings counts

with plt.style.context('seaborn-poster'):
    fig, ax = plt.subplots(figsize=(20,8))
    sns.histplot(x='RatingsCount',
                 data=movies_and_ratings_sorted,
                 color='tab:blue');
    ax.grid(False)
```
executed in 563ms, finished 04:49:17 2022-03-14



In [41]:
```
# explore most rated movies

most_rated_movies = movies_and_ratings_sorted[
    movies_and_ratings['RatingsCount']>2]

most_rated_movies.info()
```
executed in 9ms, finished 04:49:17 2022-03-14

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4979 entries, 10332 to 53319
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   movieId       4979 non-null   int64  
 1   title         4979 non-null   object 
 2   genres        4979 non-null   object 
 3   userId        4979 non-null   int64  
 4   rating        4979 non-null   float64
 5   year          4979 non-null   int64  
 6   RatingsCount  4979 non-null   int64  
dtypes: float64(1), int64(4), object(2)
memory usage: 311.2+ KB

<ipython-input-41-91ba2498a61a>:3: UserWarning: Boolean Series key will b
e reindexed to match DataFrame index.
  most_rated_movies = movies_and_ratings_sorted[
```

3 Build Recommendation System

3.1 Prepare Data for Modeling

3.1.1 Read in dataset

In [42]:
```
# instantiate a reader

reader = Reader(rating_scale=(0,5))

# load data

movies = Dataset.load_from_df(
    movies_and_ratings_sorted[['userId', 'title', 'rating']],
    reader)
```
executed in 11ms, finished 04:49:17 2022-03-14

3.1.2 Train Test Split

In [43]:
```
# create train and test datasets

train, test = train_test_split(movies, test_size=.2)
```
executed in 13ms, finished 04:49:17 2022-03-14

In [44]:
```
# look at train data

train
```
executed in 3ms, finished 04:49:17 2022-03-14

Out[44]:
```
<surprise.trainset.Trainset at 0x7fde7973e940>
```

In [45]:
```
# look at test data

test
```
executed in 109ms, finished 04:49:17 2022-03-14

Out[45]:
```
[(474, 'Wildcats', 1.0),
 (534, '21 Jump Street', 4.0),
 (567, "Ivan's Childhood (a.k.a. My Name is Ivan) (Ivanovo detstvo)",
0.5),
 (113, 'Saving Grace', 5.0),
 (91, 'Enter the Dragon', 4.0),
 (448, 'Perfect Score, The', 1.5),
 (544, 'That Thing You Do!', 5.0),
 (448, 'Albatross', 2.5),
 (448, '10 Cent Pistol', 2.0),
 (140, 'Lilies of the Field', 4.0),
 (414, 'Skin Deep', 3.0),
 (268, 'Muse, The', 3.0),
 (599, 'Wicked City (Yôjû toshi)', 2.5),
 (68, 'Hobbit: The Desolation of Smaug, The', 3.0),
 (89, 'A Perfect Day', 5.0),
 (599, 'King Ralph', 1.5),
 (326, 'Last King of Scotland, The', 4.5),
 (599, 'With Great Power: The Stan Lee Story', 3.0),
```

3.2 MODELS

3.2.1 SVD Model (Singular Value Decomposition)

This is the famous SVD algorithm used by Simon Funk for a Netflix competition which he won

When baselines are not used, this is equivalent to Probabilistic Matrix Factorization

In [46]:
```
# instantiate SVD model
svd = SVD()

# fit data on train set
svd.fit(train)

# predict/test on test aset
svd_predictions = svd.test(test)

# check RMSE and MAE
accuracy.rmse(svd_predictions)
accuracy.mae(svd_predictions)
```
executed in 388ms, finished 04:49:18 2022-03-14

```
RMSE: 0.9507
MAE:  0.7398
```

Out[46]:
```
0.7397612974244089
```

Starting SVD Model Performance: average predictions are 0.95 stars away from the actual rating

In [47]:
```
# run 5-fold cross_validation

cross_validate(svd, movies, measures =['rmse', 'mae'], cv=5, verbose=True)
```
executed in 2.08s, finished 04:49:20 2022-03-14

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9595  0.9374  0.9349  0.9351  0.9409  0.9415  0.009
2
MAE (testset)     0.7455  0.7323  0.7266  0.7288  0.7368  0.7340  0.006
7
Fit time          0.37    0.41    0.38    0.38    0.37    0.38    0.01
Test time         0.01    0.01    0.01    0.01    0.06    0.02    0.02
```

Out[47]:
```
{'test_rmse': array([0.95946107, 0.93736966, 0.93491585, 0.93505028, 0.94
094627]),
 'test_mae': array([0.74553923, 0.73226289, 0.72662335, 0.72877489, 0.736
80258]),
 'fit_time': (0.37498903274536133,
  0.4071052074432373,
  0.38315701484680176,
  0.3824949264526367,
  0.37239694595336914),
 'test_time': (0.007213830947875976 6,
  0.007066965103149414,
  0.007613182067871094,
  0.010246038436889648,
  0.0639491081237793)}
```

In [48]:
```
# GRIDSEARCH - SVD Model

param_grid = {'n_epochs': [5, 10], 'lr_all': [0.002, 0.005],
              'reg_all': [0.4, 0.6]}

gs_svd = GridSearchCV(SVD,
                      param_grid,
                      measures=['rmse', 'mae'],
                      cv=5)

gs_svd.fit(movies)

# best RMSE score
print(gs_svd.best_score['rmse'])

# combination of parameters that gave the best RMSE score
print(gs_svd.best_params['rmse'])
```
executed in 6.27s, finished 04:49:26 2022-03-14

```
0.9579050537876668
{'n_epochs': 10, 'lr_all': 0.005, 'reg_all': 0.4}
```

In [49]:
```
# FINAL SVD MODEL

svd_algo = gs_svd.best_estimator['rmse']
svd_algo.fit(movies.build_full_trainset())

svd_best_predictions = svd_algo.test(test)

# check RMSE and MAE of final SVD model

accuracy.rmse(svd_best_predictions)
accuracy.mae(svd_best_predictions)
```
executed in 262ms, finished 04:49:26 2022-03-14

```
RMSE: 0.8601
MAE:  0.6738
```

Out[49]:
```
0.6737755408300745
```

Final SVD Model Performance: average predictions are 0.86 stars away from the actual rating

View five predictions

r_ui: actual rating
est: estimated rating

In [50]:
```
# check five predictions

svd_best_predictions[:5]
```
executed in 4ms, finished 04:49:26 2022-03-14

Out[50]:
```
[Prediction(uid=474, iid='Wildcats', r_ui=1.0, est=3.2312788476843943, de
tails={'was_impossible': False}),
 Prediction(uid=534, iid='21 Jump Street', r_ui=4.0, est=3.63673703448861
9, details={'was_impossible': False}),
 Prediction(uid=567, iid="Ivan's Childhood (a.k.a. My Name is Ivan) (Ivan
ovo detstvo)", r_ui=0.5, est=2.2375535472515216, details={'was_impossibl
e': False}),
 Prediction(uid=113, iid='Saving Grace', r_ui=5.0, est=3.692164164558661,
details={'was_impossible': False}),
 Prediction(uid=91, iid='Enter the Dragon', r_ui=4.0, est=3.3431697623439
4, details={'was_impossible': False})]
```

Test/make a prediction with user and movie id

In [51]:
```
# get prediction for specific user and item

user_id = str(111)
movie_id = str(111)

svd_predictor = svd_algo.predict(user_id, movie_id, r_ui=4, verbose=True)
```
executed in 3ms, finished 04:49:26 2022-03-14

```
user: 111        item: 111        r_ui = 4.00   est = 3.27   {'was_imposs
ible': False}
```

For reference: function provided by Surprise to find top_n recommendations for users

To be improved in Final Concept Launch

In [52]:
```
from collections import defaultdict


def get_top_n(svd__best_predictions, n=10):
    """Return the top-N recommendation for each user from a set of predicti

    Args:
        predictions(list of Prediction objects): The list of predictions, a
            returned by the test method of an algorithm.
        n(int): The number of recommendation to output for each user. Defau
            is 10.

    Returns:
    A dict where keys are user (raw) ids and values are lists of tuples:
        [(raw item id, rating estimation), ...] of size n.
    """

    # Map the predictions to each user
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in svd_best_predictions:
        top_n[uid].append((iid, est))

    # Sort the predictions for each user and retrieve the k highest ones
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n

top_n = get_top_n(svd_best_predictions, n=10)

# Print the recommended items for each user
for uid, user_ratings in top_n.items():
    print(uid, [iid for (iid, _) in user_ratings])
```

executed in 30ms, finished 04:49:26 2022-03-14

```
474 ['Mildred Pierce', 'Safety Last!', 'Auntie Mame', 'Jane Eyre', 'Ste
amboat Bill, Jr.', 'Anna Karenina', 'Divided We Fall (Musíme si pomáha
t)', 'Dark Water (Honogurai mizu no soko kara)', 'Zelary', 'Dark Passag
e']
534 ['10th Kingdom, The', '21 Jump Street', 'Oz the Great and Powerfu
l', 'Doomsday', 'Absolutely Anything', 'Lone Ranger, The', 'Sinbad: Leg
end of the Seven Seas', '47 Ronin', 'Stalingrad', 'Star Wars: Episode I
I - Attack of the Clones']
567 ['Eye in the Sky', 'Frances Ha', 'Everest', 'Only Yesterday (Omohid
e poro poro)', 'Lilya 4-Ever (Lilja 4-ever)', 'The Martian', 'Fences',
"The Devil's Candy", 'Jack and Jill', "God's Not Dead"]
113 ["For Roseanna (Roseanna's Grave)", 'Crying Game, The', 'Saving Gra
ce', 'Morning After, The', 'Living Out Loud']
91 ['Pirates of Silicon Valley', 'Enter the Dragon', 'Body Snatcher, Th
e', 'Martin', 'Robot Carnival (Roboto kânibauru)', 'Startup.com', 'Craz
ies, The (a.k.a. Code Name: Trixie)', 'Fist of Fury (Chinese Connectio
n, The) (Jing wu men)', 'Outlaw Josey Wales, The', 'Godzilla, King of t
he Monsters! (Kaijû-ô Gojira)']
448 ['42', 'City Island', 'Florence Foster Jenkins', 'Alan Partridge: A
```

3.2.2 KNN Baseline Model (k-Nearest Neighbor)

k-Nearest Neighbor Baseline Model

Collaborative filtering algorithm that uses Baseline rating

In [53]:
```
# instantiate KNN Baseline model
knnb = KNNBaseline(k=50)

# fit data on train set
knnb.fit(train)

# predict/test on test aset
knnb_predictions = knnb.test(test)

# check RMSE and MAE
accuracy.rmse(knnb_predictions)
accuracy.mae(knnb_predictions)
```
executed in 43ms, finished 04:49:26 2022-03-14

```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9607
MAE:  0.7492
```

Out[53]:
```
0.7491622763262366
```

Starting KNN Baseline Model Performance: average predictions are 0.96 stars away from the actual rating

In [54]:
```
# GRIDSEARCH - KNN Baseline Model

param_grid = {'k': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
              'sim_options': {'user_based': [True, False]},\
              'bsl_options': {'method': ['als', 'sgd']}}

gs_knnb = GridSearchCV(KNNBaseline,
                       param_grid,
                       measures=['rmse', 'mae'],
                       cv=5)

gs_knnb.fit(movies)

print(gs_knnb.best_score['rmse'])
print(gs_knnb.best_params['rmse'])
```

executed in 1m 42.6s, finished 04:51:09 2022-03-14

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

In [55]:
```
# FINAL KNN MODEL

knnb_algo = gs_knnb.best_estimator['rmse']
knnb_algo.fit(movies.build_full_trainset())

knnb_best_predictions = knnb_algo.test(test)

# check RMSE and MAE of final KNN model

accuracy.rmse(knnb_best_predictions)
accuracy.mae(knnb_best_predictions)
```
executed in 2.39s, finished 04:51:11 2022-03-14

```
Estimating biases using sgd...
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.3926
MAE:  0.2647
```

Out[55]:
```
0.2646965200335826
```

Final KNN Baseline Model Performance: average predictions are 0.39 stars away from the actual rating

View five predictions

r_ui: actual rating
est: estimated rating

In [56]:
```
knnb_best_predictions[:5]
```
executed in 3ms, finished 04:51:11 2022-03-14

Out[56]:
```
[Prediction(uid=474, iid='Wildcats', r_ui=1.0, est=1.8593421262239793, de
tails={'actual_k': 100, 'was_impossible': False}),
 Prediction(uid=534, iid='21 Jump Street', r_ui=4.0, est=3.85720076277837
93, details={'actual_k': 44, 'was_impossible': False}),
 Prediction(uid=567, iid="Ivan's Childhood (a.k.a. My Name is Ivan) (Ivan
ovo detstvo)", r_ui=0.5, est=1.007962576668691, details={'actual_k': 88,
'was_impossible': False}),
 Prediction(uid=113, iid='Saving Grace', r_ui=5.0, est=4.612170066874563
5, details={'actual_k': 21, 'was_impossible': False}),
 Prediction(uid=91, iid='Enter the Dragon', r_ui=4.0, est=3.7147285147535
354, details={'actual_k': 67, 'was_impossible': False})]
```

3.2.3 NMF Model (Non-Negative Matrix Factorization)

Non-Negative Matrix Factorization Model

Collaborative filtering model based on Non-Negative Matrix Factorization

In [57]:
```
# instantiate NMF model
nmf = NMF()

# fit data on train set
nmf.fit(train)

# predict/test on test aset
nmf_predictions = nmf.test(test)

# check RMSE and MAE

accuracy.rmse(nmf_predictions)
accuracy.mae(nmf_predictions)
```
executed in 831ms, finished 04:51:12 2022-03-14

```
RMSE: 1.1151
MAE:  0.8960
```

Out[57]:
```
0.8960196508519168
```

Starting NMF Model Performance: average predictions are 1.12 stars away from the actual rating

In [58]:
```
# GRIDSEARCH - NMF Model

param_grid = {'n_factors': [1,2,3,4,5,6,7,8,9,10],
              'n_epochs': [100],
              'biased': [True],
              'reg_bu': [0.1],
              'reg_bi': [0.1]}

gs_nmf = GridSearchCV(NMF, param_grid, measures=['rmse', 'mae'], cv=5)

gs_nmf.fit(movies)

#nmfb = gs_nmfb.best_estimator['rmse']
print(gs_nmf.best_score['rmse'])
print(gs_nmf.best_params['rmse'])
```
executed in 44.4s, finished 04:51:57 2022-03-14

```
1.0967863468536287
{'n_factors': 1, 'n_epochs': 100, 'biased': True, 'reg_bu': 0.1, 'reg_b
i': 0.1}
```

In [59]:
```
# FINAL NMF MODEL

nmf_algo = gs_nmf.best_estimator['rmse']
nmf_algo.fit(movies.build_full_trainset())

nmf_best_predictions = nmf_algo.test(test)

# check RMSE and MAE of final NMF model

accuracy.rmse(nmf_best_predictions)
accuracy.mae(nmf_best_predictions)
```
executed in 817ms, finished 04:51:57 2022-03-14

```
RMSE: 0.3064
MAE:  0.1708
```

Out[59]:
```
0.1707548649981007
```

Final NMF Model Performance: average predictions are 0.30 stars away from the actual rating

View five predictions

r_ui: actual rating
est: estimated rating

In [60]:
```
nmf_best_predictions[:5]
```
executed in 3ms, finished 04:51:57 2022-03-14

Out[60]:
```
[Prediction(uid=474, iid='Wildcats', r_ui=1.0, est=2.216686139297967, det
ails={'was_impossible': False}),
 Prediction(uid=534, iid='21 Jump Street', r_ui=4.0, est=3.93075269893139
94, details={'was_impossible': False}),
 Prediction(uid=567, iid="Ivan's Childhood (a.k.a. My Name is Ivan) (Ivan
ovo detstvo)", r_ui=0.5, est=1.1027548082478766, details={'was_impossibl
e': False}),
 Prediction(uid=113, iid='Saving Grace', r_ui=5.0, est=4.895730843115939,
details={'was_impossible': False}),
 Prediction(uid=91, iid='Enter the Dragon', r_ui=4.0, est=3.9308589487980
776, details={'was_impossible': False})]
```

4 Evaluation and Conclusions

We've built a Recommendation System to execute on Vi(sion) Studios "Digital Cinema Night" concept!

Model Evaluation

- Singular Vector Decomposition Model (SVD):
  * Pre-Tuned RMSE: 0.9502
  * Tuned RMSE: 0.8622

- k-Nearest Neighbor Baseline Model (KNNB):
  * Pre-Tuned RMSE: 0.9604
  * Tuned RMSE: 0.3863

- Non-Negative Matrix Factorization Model (NMF):
  * Pre-Tuned RMSE: 1.1232
  * Tuned RMSE: 0.2964

We've decided to use the SVD Model to launch "Digital Cinema Night." While all models showed improvement after tuning and KNN Baseline and NMF Models have low RMSEs (predictions are close to actual ratings), in order to reduce the risk of over-fitting, we will use the SVD model. With an RMSE of 0.86, we feel best our recommendation system make more expansive recommendations to users.

Summary of recommendations

- Use SVD Model to launch

- Test for 6 months and re-evaluate
  *Revisit KNN Baseline and/or NMF Models if necessary

- Launch concept to target Millennial audience
  *Movies weighted toward movies from 1990s to early 2000s
  *Build early loyalty with influential demographic

Further considerations:

- Explore launching next round with genre and year selections as a function

5 Future Work

This is just the beginning!

Future work:

- Integrate genre and year functionality into recommendation system
- Launch website allowing users to build "Digital Cinema Night" by typing in movie names, genre, and/or year

5.1 APPENDIX

Output to explore and aspire toward

In [61]:
```
# cross-validation results dataframe

results_df = pd.DataFrame.from_dict(gs_svd.cv_results)
results_df
```

executed in 97ms, finished 04:51:58 2022-03-14

Out[61]:

	split0_test_rmse	split1_test_rmse	split2_test_rmse	split3_test_rmse	split4_test_rmse	mean_test
0	0.977595	0.989719	0.979794	0.982992	0.995029	0.9
1	0.984173	0.995288	0.984269	0.989404	0.999870	0.9
2	0.961932	0.971756	0.961698	0.964057	0.975037	0.9

In [62]:
```
# create matrix for movies and ratings

matrix = movies_and_ratings.pivot_table(
    index='userId',
    columns='title',
    values='rating'
)

matrix.head()
```
executed in 172ms, finished 04:51:58 2022-03-14

Out[62]:

title	'71	'Hellboy': The Seeds of Creation	'Round Midnight	'Salem's Lot	'Til There Was You	'Tis the Season for Love	'burbs, The	'night Mother	(500) Days of Summer	*batteries not included	...
userId											
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

5 rows × 9423 columns

In [63]:
```
# create recommendation system to make 5 recommendations to users

def cinema_night_for(title, ratings_count_filter=100, number_recommendation
    similar = matrix.corrwith(matrix[title])
    corr_similar = pd.DataFrame(similar, columns=['correlation'])
    corr_similar.dropna(inplace=True)

    orig = movies_and_ratings.copy()

    corr_with_movie = pd.merge(
        left=corr_similar,
        right=orig,
        on='title')[['title', 'correlation', 'RatingsCount']].drop_duplicat

    result = corr_with_movie[corr_with_movie['RatingsCount'] > ratings_coun

    return result.head(number_recommendations)
```
executed in 4ms, finished 04:51:58 2022-03-14

In [64]:
```
cinema_night_for('Toy Story')
```
executed in 1.75s, finished 04:51:59 2022-03-14

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/l
ib/function_base.py:2526: RuntimeWarning: Degrees of freedom <= 0 for sli
ce
  c = cov(x, y, rowvar)
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/l
ib/function_base.py:2455: RuntimeWarning: divide by zero encountered in t
rue_divide
  c *= np.true_divide(1, fact)
```

Out[64]:

	title	correlation	RatingsCount
4485	Toy Story	1.000000	215
2174	Incredibles, The	0.643301	125
1505	Finding Nemo	0.618701	141
138	Aladdin	0.611892	183
2914	Monsters, Inc.	0.490231	132
2960	Mrs. Doubtfire	0.446261	144

In [65]:
```
# TEST RECOMMENDATION SYSTEM using Toy Story

cinema_night_for('Toy Story')
```
executed in 1.79s, finished 04:52:01 2022-03-14

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/l
ib/function_base.py:2526: RuntimeWarning: Degrees of freedom <= 0 for sli
ce
  c = cov(x, y, rowvar)
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/l
ib/function_base.py:2455: RuntimeWarning: divide by zero encountered in t
rue_divide
  c *= np.true_divide(1, fact)
```

Out[65]:

	title	correlation	RatingsCount
4485	Toy Story	1.000000	215
2174	Incredibles, The	0.643301	125
1505	Finding Nemo	0.618701	141
138	Aladdin	0.611892	183
2914	Monsters, Inc.	0.490231	132
2960	Mrs. Doubtfire	0.446261	144

In [66]:
```
# Test for The Shawshank Redemption

cinema_night_for('Shawshank Redemption, The')
```
executed in 1.76s, finished 04:52:03 2022-03-14

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/l
ib/function_base.py:2526: RuntimeWarning: Degrees of freedom <= 0 for sli
ce
  c = cov(x, y, rowvar)
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/l
ib/function_base.py:2455: RuntimeWarning: divide by zero encountered in t
rue_divide
  c *= np.true_divide(1, fact)
```

Out[66]:

	title	correlation	RatingsCount
3890	Shawshank Redemption, The	1.000000	317
1598	Four Weddings and a Funeral	0.446212	103
3787	Schindler's List	0.402202	220
4616	Usual Suspects, The	0.394294	204
3175	Ocean's Eleven	0.391546	119
1838	Green Mile, The	0.382818	111

In [67]:
```
cinema_night_for('Godfather, The')
```
executed in 1.75s, finished 04:52:05 2022-03-14

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/l
ib/function_base.py:2526: RuntimeWarning: Degrees of freedom <= 0 for sli
ce
  c = cov(x, y, rowvar)
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/l
ib/function_base.py:2455: RuntimeWarning: divide by zero encountered in t
rue_divide
  c *= np.true_divide(1, fact)
```

Out[67]:

	title	correlation	RatingsCount
1720	Godfather, The	1.000000	192
1721	Godfather: Part II, The	0.782643	129
3740	Schindler's List	0.456661	220
1475	Fight Club	0.445205	218
3718	Saving Private Ryan	0.441377	188
1755	Goodfellas	0.439937	126

In [68]:
```
from itertools import permutations

# Create the function to find all permutations
def find_movie_pairs(x):
  pairs = pd.DataFrame(list(permutations(x.values, 2)),
                       columns=['movie_a', 'movie_b'])
  return pairs

# Apply the function to the title column and reset the index
movie_combinations = movies_and_ratings.groupby('userId')['title'].apply(
  find_movie_pairs).reset_index(drop=True)

print(movie_combinations)
```

executed in 17.4s, finished 04:52:22 2022-03-14

```
                         movie_a                      movie_b
0                      Toy Story             Grumpier Old Men
1                      Toy Story                         Heat
2                      Toy Story         Seven (a.k.a. Se7en)
3                      Toy Story          Usual Suspects, The
4                      Toy Story           From Dusk Till Dawn
...                          ...                          ...
60731789  The Fate of the Furious  Rogue One: A Star Wars Story
60731790  The Fate of the Furious                        Split
60731791  The Fate of the Furious         John Wick: Chapter Two
60731792  The Fate of the Furious                      Get Out
60731793  The Fate of the Furious                        Logan

[60731794 rows x 2 columns]
```

In [69]:
```
# calculate how often each item in movie_a occurs with the items in movie_b
combination_counts = movie_combinations.groupby(['movie_a', 'movie_b']).siz

# convert the results to a DataFrame and reset the index
combination_counts_df = combination_counts.to_frame(name= 'size').reset_ind
print(combination_counts_df.head())
```

executed in 18.6s, finished 04:52:41 2022-03-14

```
  movie_a                                movie_b  size
0     '71                    (500) Days of Summer     1
1     '71                     10 Cloverfield Lane     1
2     '71                               127 Hours     1
3     '71  13 Assassins (Jûsan-nin no shikaku)     1
4     '71                                13 Hours     1
```

In [70]:
```
# calculate ratings by genre

values = defaultdict(list)
for ind, row in movies_and_ratings.iterrows():
    for genre in row['genres'].split('|'):
        values[genre].append(row['rating'])


genre_list, rating_list = [], []
for key, item in values.items():
    if key not in [0, 1]:
        genre_list.append(key)
        rating_list.append(np.mean(item))


genres_and_ratings = pd.DataFrame([genre_list, rating_list]).T
genres_and_ratings.columns = ['genre', 'avg_rating']
```

executed in 6.39s, finished 04:52:47 2022-03-14

In [71]:
```
genres_and_ratings
```

executed in 6ms, finished 04:52:47 2022-03-14

Out[71]:

	genre	avg_rating
0	Adventure	3.50886
1	Animation	3.62998
2	Children	3.41311
3	Comedy	3.38471
4	Fantasy	3.491
5	Romance	3.50651
6	Drama	3.65619
7	Action	3.44798
8	Crime	3.65849
9	Thriller	3.49376
10	Horror	3.25834
11	Mystery	3.63246
12	Sci-Fi	3.45601
13	War	3.80815
14	Musical	3.56368
15	Documentary	3.79779
16	IMAX	3.61834
17	Western	3.58394
18	Film-Noir	3.92011
19	(no genres listed)	3.42105