HW 1 – Frequent Pattern Mining

Name: Robb Alexander and Ryan Bailis Class: CSCI349
Semester: 2021SP
Instructor: Brian King

In [1]:
```
import numpy as np
import pandas as pd
import plotly.express as px
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules
```

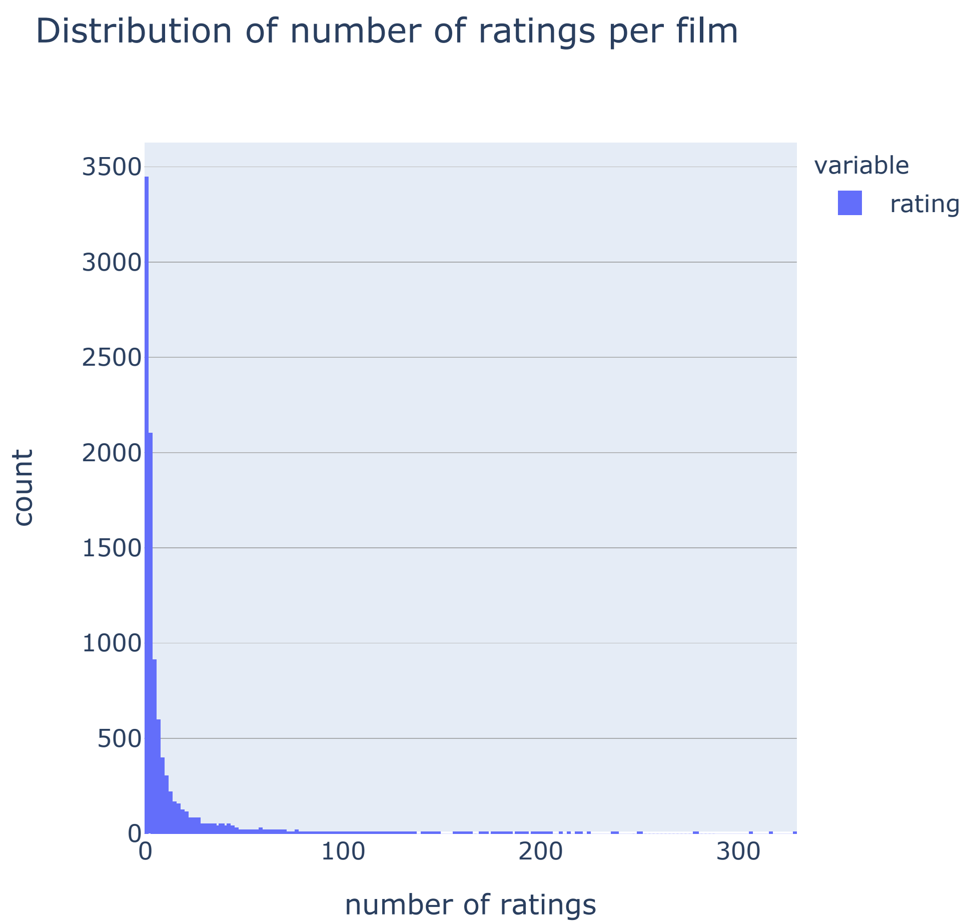
Phase I - EDA

In [2]:
```
ratings = pd.read_csv("/Users/rale/Documents/Programming/csci349_2021sp/
data/ml-latest-small/ratings.csv")
```

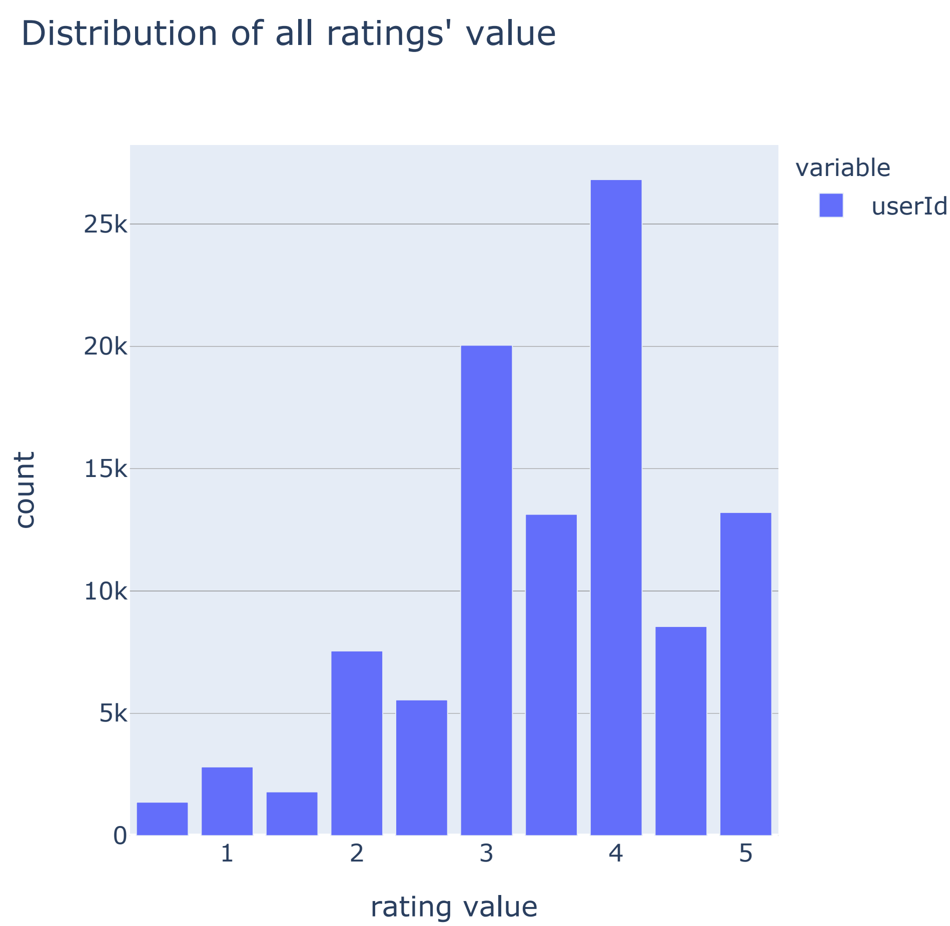
In [3]:
```
movies = pd.read_csv("/Users/rale/Documents/Programming/csci349_2021sp/d
ata/ml-latest-small/movies.csv")
movies.set_index("movieId", inplace=True)
movies.genres = movies.genres.apply(str.split, args="|")
```

EDA plots

In [4]:
```
num_ratings_distribution = ratings.groupby("movieId").count().rating
px.histogram(num_ratings_distribution, labels={"value":"number of rating
s"}, title="Distribution of number of ratings per film")
```



```
In [5]: ratings_distribution = ratings.groupby("rating").count().userId
px.bar(ratings_distribution, labels={"rating":"rating value","value":"co
unt"}, title="Distribution of all ratings' value")
# ratings_distribution
```



Max amount of Ratings

```
In [6]: max_ratings_id = ratings.groupby("movieId").count().rating.sort_values(a
scending=False).index[0]
movies.loc[max_ratings_id]
```

```
Out[6]: title                 Forrest Gump (1994)
genres     [Comedy, Drama, Romance, War]
Name: 356, dtype: object
```

Highest average ratings with over a 15 count

In [7]:
```
max_avg_ratings = ratings.groupby("movieId").mean()[ratings.groupby("mov
ieId").count().rating > 15].rating.sort_values(ascending=False)
max_avg_ratings_ids = max_avg_ratings.index[:5]
top_films = movies.loc[max_avg_ratings_ids]
top_films.drop("genres", axis=1, inplace=True)
top_films["ratings"] = max_avg_ratings
top_films
```

Out[7]:

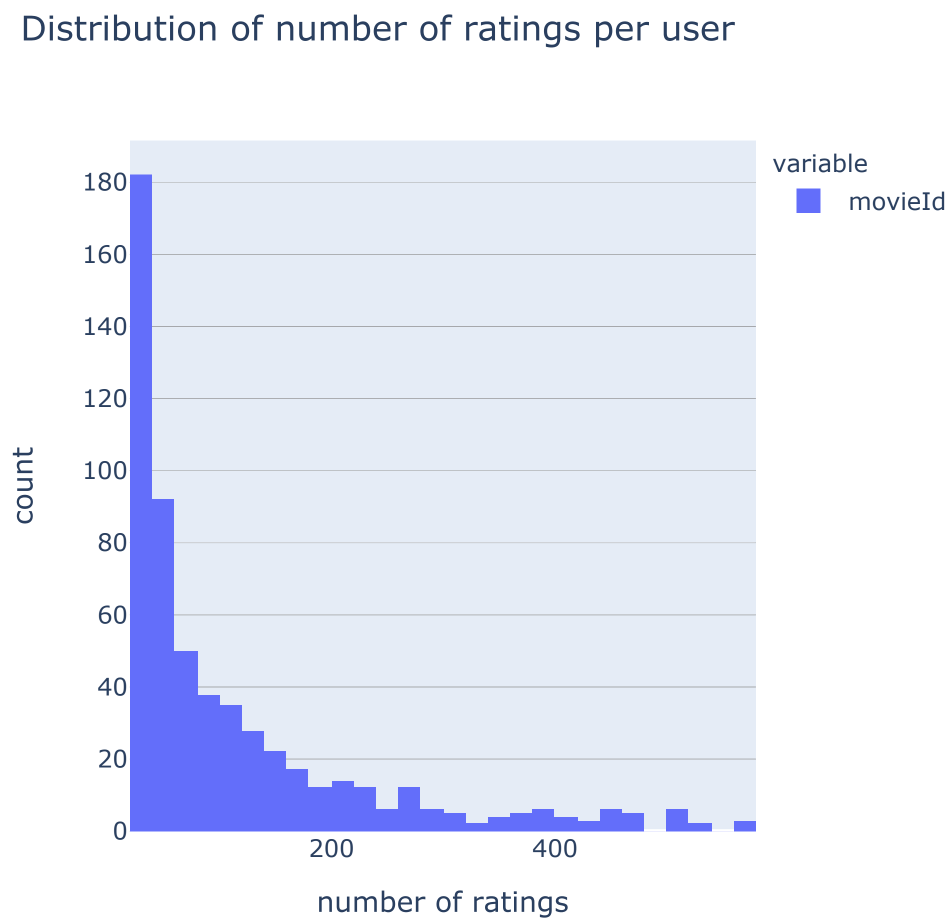
	title	ratings
movieId		
1104	Streetcar Named Desire, A (1951)	4.475000
318	Shawshank Redemption, The (1994)	4.429022
922	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	4.333333
3468	Hustler, The (1961)	4.333333
3435	Double Indemnity (1944)	4.323529

In [8]:
```
#### Lowest average ratings with over a 15 count

min_avg_ratings = ratings.groupby("movieId").mean()[ratings.groupby("mov
ieId").count().rating > 15].rating.sort_values()
min_avg_ratings_ids = min_avg_ratings.index[:5]
low_films = movies.loc[min_avg_ratings_ids]
low_films.drop("genres", axis=1, inplace=True)
low_films["ratings"] = min_avg_ratings
low_films
```

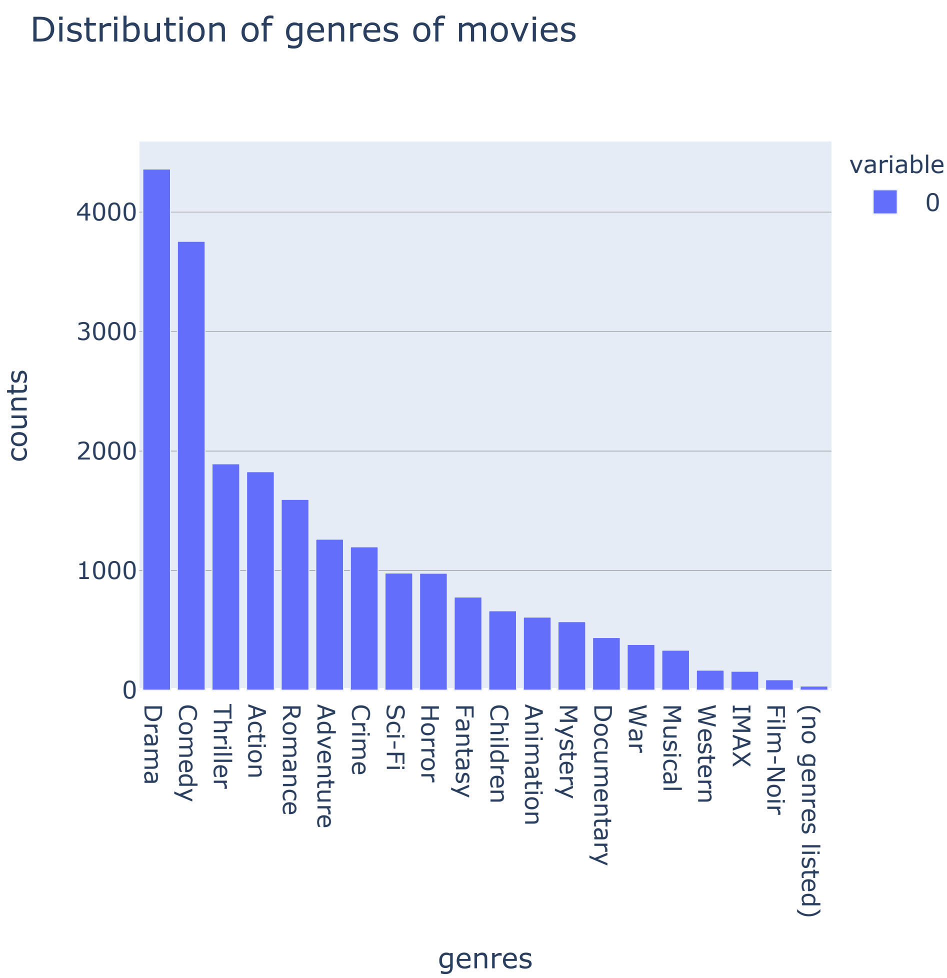
Out[8]:

	title	ratings
movieId		
1556	Speed 2: Cruise Control (1997)	1.605263
3593	Battlefield Earth (2000)	1.657895
2643	Superman IV: The Quest for Peace (1987)	1.687500
1499	Anaconda (1997)	1.925926
2412	Rocky V (1990)	1.941176

In [9]:
```python
# exclude outliers, cap at 600 for visuals
user_distribution = ratings.groupby("userId").count().movieId[ratings.groupby("userId").count().movieId < 600]
px.histogram(user_distribution, labels={"value":"number of ratings"}, title="Distribution of number of ratings per user")
```



In [10]:
```
te = TransactionEncoder()
te_ary = te.fit_transform(movies.genres)
genres_enc = pd.DataFrame(te_ary, columns=te.columns_, index=movies.inde
x)
genres_counts = genres_enc.sum().sort_values(ascending=False)
px.bar(genres_counts, labels={"index":"genres","value":"counts"}, title=
"Distribution of genres of movies")
```



Phase II - Rules

Itemsets generation

First, we group by the users, and take all the movie ids and apply them into one list. Now with the user_watched being a list of 'transactions' filled with all the films they've watched, we can run the itemset generation and rule generation (using fpgrowth this time)

In [11]:
```python
user_watched = ratings.groupby("userId")["movieId"].apply(list)
te_ary = te.fit_transform(user_watched)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)
watched_enc
```

Out[11]:

	1	2	3	4	5	6	7	8	9	10	...	193565	193567	1935
0	True	False	True	False	False	True	False	False	False	False	...	False	False	Fal
1	False	False	False	False	False	False	False	False	False	False	...	False	False	Fal
2	False	False	False	False	False	False	False	False	False	False	...	False	False	Fal
3	False	False	False	False	False	False	False	False	False	False	...	False	False	Fal
4	True	False	False	False	False	False	False	False	False	False	...	False	False	Fal
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
605	True	False	False	False	False	False	True	False	False	False	...	False	False	Fal
606	True	False	False	False	False	False	False	False	False	False	...	False	False	Fal
607	True	True	True	False	False	False	False	False	False	True	...	False	False	Fal
608	True	False	False	False	False	False	False	False	False	True	...	False	False	Fal
609	True	False	False	False	False	True	False	False	False	False	...	False	False	Fal

610 rows × 9724 columns

In [12]:
```python
def convert_ids(ids):
    """
    converts movie ids to their respective names
    in strings from the `movies` dataframe

    :param ids: array-like
    :return: pandas.Series
    """
    return movies.loc[ids].reset_index().title
```

In [13]:
```
items = fpgrowth(watched_enc, min_support=0.2, use_colnames=True)
items_20 = items[:20].copy()
items_20.itemsets = items_20.itemsets.apply(convert_ids)
items_20
```

Out[13]:

	support	itemsets
0	0.539344	Forrest Gump (1994)
1	0.503279	Pulp Fiction (1994)
2	0.457377	Silence of the Lambs, The (1991)
3	0.455738	Matrix, The (1999)
4	0.411475	Star Wars: Episode IV - A New Hope (1977)
5	0.390164	Jurassic Park (1993)
6	0.388525	Braveheart (1995)
7	0.360656	Schindler's List (1993)
8	0.357377	Fight Club (1999)
9	0.352459	Toy Story (1995)
10	0.345902	Star Wars: Episode V - The Empire Strikes Back...
11	0.334426	Usual Suspects, The (1995)
12	0.334426	American Beauty (1999)
13	0.332787	Seven (a.k.a. Se7en) (1995)
14	0.331148	Independence Day (a.k.a. ID4) (1996)
15	0.327869	Raiders of the Lost Ark (Indiana Jones and the...
16	0.321311	Star Wars: Episode VI - Return of the Jedi (1983)
17	0.311475	Fugitive, The (1993)
18	0.309836	Batman (1989)
19	0.308197	Saving Private Ryan (1998)

Association Rules Generation

We can then use the `association_rules()` function to generate them using the confidence metric set at 70%. Then filter based on lift threshold of 2, and sort them by lift/confidence.
After this, we have to convert the movie ids with the `convert_ids()` function written above with the use of `map()` and revert them back to the frozenset type.

In [14]:
```
rules = association_rules(items, metric="confidence", min_threshold=0.7)
# rules = rules[rules.antecedents.apply(lambda x: len(x) == 1)]
rules = rules[rules.lift > 2]
rules = rules.sort_values(by=["lift", "confidence"], ascending=False)

rules.antecedents = rules.antecedents.map(convert_ids).map(frozenset)
rules.consequents = rules.consequents.map(convert_ids).map(frozenset)
rules.head(5)

# formatted printing
def pretty_rules(df):
    for index, rule in df.iterrows():
        print(list(rule.antecedents))
        print("↓")
        print(list(rule.consequents))
        print('c' + str(round(rule.confidence* 100,1)) + '%', 'l' + str(
round(rule.lift,3)), '\n')
```

Association Rules Analysis

First we analyzed the rules with only one antecedent, which can narrow and focus a single type of film to more films. Some interesting ones are:

```
['Beauty and the Beast (1991)']
↓
['Aladdin (1992)']
c0.842 l2.808

['Mission: Impossible (1996)']
↓
['Independence Day (a.k.a. ID4) (1996)']
c79.6% l2.405
```

There was also a large number of LotR rules, but the highest one with one antecedent was the last film -> the first two films. This is a pattern we saw in most of the rules, if you watched the last film, it means that you normally have seen the prequels prior. This way, it might be good to think the consequents as necessary films to watch, before the consequent. While most of the first few films in a series as the antecedent had much lower lift values, which could mean that watching earlier film doesn't entail you are interested in the sequels, but if you have seen the sequels you are very likely to have watched the prequels.

```
['Lord of the Rings: The Return of the King, The (2003)']
↓
['Lord of the Rings: The Fellowship of the Ring, The (2001)', 'Lord of the R
ings: The Two Towers, The (2002)']
c83.2% l3.059

['Indiana Jones and the Last Crusade (1989)']
↓
['Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1
981)']
c87.9% l2.68

['Godfather: Part II, The (1974)']
↓
['Godfather, The (1972)']
c0.969 l3.079
```

The Star Wars rules also follow the idea of what was stated previously, but here there was an added interesting factor of having a different trilogy involved, i.e. Indiana Jones. Which I assume is from the deep relationship of Spielberg and Lucas.

```
['Star Wars: Episode VI - Return of the Jedi (1983)', 'Raiders of the Lost A
rk (Indiana Jones and the Raiders of the Lost Ark) (1981)']
↓
['Star Wars: Episode IV - A New Hope (1977)', 'Star Wars: Episode V - The Em
pire Strikes Back (1980)']
c96.1% l3.084
```

These rule also seems to be quite interesting, all of these movies have themes of memory and psychosis, 'mind-bending' at times, with big twists Very apt for them to be correlated.

```
['Memento (2000)']
↓
['Fight Club (1999)']
c77.4% l2.165

['Sixth Sense, The (1999)']
↓
['Fight Club (1999)'] c71.5% l2.001
```

These two below also illustrate the idea that genres will be a very powerful tool in recommending movies with similar interests. Here, I would assume it to be crime-action detective-y films.

```
['True Lies (1994)']
↓
['Batman (1989)']
c76.4% l2.466

['True Lies (1994)']
↓
['Fugitive, The (1993)']
c72.5% l2.327
```

Other rules with large groups of antecedents are all just groups of popular films that do not have much in common besides them being well received on average.

```
['Pulp Fiction (1994)', 'Fight Club (1999)', 'Forrest Gump (1994)']
↓
['Matrix, The (1999)']
c95.4% l2.093
```

Phase III - Genre

Helper Filtering Function

In [15]:
```
def filter_ratings(genre_name):
    """
    get ratings for movies that belong to a specific genre

    :param genre_name: string of genre in movies df column genres
    :return: pandas.DataFrame filtered ratings by genre
    """
    genre_ids = set(movies[movies.genres.apply(lambda x: genre_name in x
)].index)
    genre_ratings = ratings[ratings.movieId.apply(lambda x: x in genre_i
ds)]
    return genre_ratings
```

Rule Generation Function

In [16]:

```
def generate_rules(genre, min_sup=0.2, min_conf=0.7, min_lift=0, single_ant=False):
    """
    get ratings for movies that belong to a specific genre

    :param genre: string of genre in movies df column genres
    :param min_sup: min support for fpgrowth function
    :param min_conf: min confidence for association_rules function
    :param min_lift: min lift for filtering
    :param single_ant: filter rules to include only 1 antecedent
    :return: pandas.DataFrame of association rules
    """
    genre_ratings = filter_ratings(genre)
    genre_user_watched = genre_ratings.groupby("userId")["movieId"].apply(list)
    genre_te_ary = te.fit_transform(genre_user_watched)
    genre_watched_enc = pd.DataFrame(genre_te_ary, columns=te.columns_)

    genre_items = fpgrowth(genre_watched_enc, min_support=min_sup, use_colnames=True)
    genre_rules = association_rules(genre_items, metric="confidence", min_threshold=min_conf)
    genre_rules = genre_rules.sort_values(by=["confidence"], ascending=False)
    genre_rules = genre_rules[genre_rules.lift > min_lift]
    if single_ant:
        genre_rules = genre_rules[genre_rules.antecedents.apply(lambda x: len(x) == 1)]

    genre_rules.antecedents = genre_rules.antecedents.map(convert_ids).map(frozenset)
    genre_rules.consequents = genre_rules.consequents.map(convert_ids).map(frozenset)
    return genre_rules
```

Discussion

With the genre rules, what we notice instantly is that now we can differentiate the sub-genres of comedy itself. This is very good because we can recommend a type of sub-genre if that is detected in the user's movie habits. e.g. animated-comedy vs dark-adult-comedy vs family-comedy

It can be seen that this works much better than a general total rule-set since there is less bias for popular films, this seems like a good way to weight the films based on interest from the users themselves. This approach helps narrow down what the users would like much better than the original approach.

In [17]:

```
comedy = generate_rules("Comedy", min_sup=0.09,min_conf=0.85, min_lift=4)
# pretty_rules(comedy)
```

Comedy Rules

Animated-Comedy

```
['Incredibles, The (2004)', 'Monsters, Inc. (2001)', 'Shrek 2 (2004)']
↓
['Finding Nemo (2003)']
c94.9% l4.1
```

Family-Comedy

```
['Back to the Future Part II (1989)', 'Toy Story (1995)']
↓
['Back to the Future (1985)', 'Forrest Gump (1994)']
c96.5% l4.321

['Ghost (1990)', 'Mrs. Doubtfire (1993)', 'Sleepless in Seattle (1993)']
↓
['Pretty Woman (1990)', 'Forrest Gump (1994)']
c93.3% l4.547
```

Dark-Comedy

```
["Monty Python's Life of Brian (1979)", 'Fargo (1996)']
↓
['Monty Python and the Holy Grail (1975)']
c92.1% l4.123

['True Lies (1994)', 'Addams Family Values (1993)']
↓
['Batman Forever (1995)']
c91.7% l4.075

['Austin Powers: International Man of Mystery (1997)', 'Men in Black (a.k.a.
MIB) (1997)', 'Forrest Gump (1994)']
↓
['Austin Powers: The Spy Who Shagged Me (1999)']
c86.8% l4.367
```

In [18]:
```
romance = generate_rules("Romance", min_sup=0.08, min_conf=0.6, min_lift
=3)
# pretty_rules(romance)
```

Out of all the rules, the romance genre showed a trend of similar release years, much more than the other two . This could be related to how romance and eras have an interlinked connection.

Again we see sub-genres from the outputted rules.
Animated-Romance

```
['Shrek 2 (2004)']
↓
['Shrek (2001)']
c90.2% l3.216

['Cinderella (1950)']
↓
['Beauty and the Beast (1991)']
c81.0% l3.36
```

manic-pixie-dream-girl-Romance

```
['Garden State (2004)']
↓
['Eternal Sunshine of the Spotless Mind (2004)']
c85.4% l3.951

['Garden State (2004)']
↓
['Lost in Translation (2003)']
c70.8% l5.801
```

Action-Romance

```
['Twister (1996)', 'True Lies (1994)']
↓
['Speed (1994)']
c87.1% l3.088
```

Popular-Romance

```
['Sleepless in Seattle (1993)', 'Forrest Gump (1994)', 'Ghost (1990)']
↓
['Pretty Woman (1990)']
c92.6% l4.159
```

In [19]:

```
scifi = generate_rules("Sci-Fi", min_sup=0.2, min_conf=0.6)
# pretty_rules(scifi)
# scifi
```

Dystopian Sci-Fi

```
['RoboCop (1987)']
↓
['Terminator, The (1984)']
c87.1% l4.025

['Total Recall (1990)']
↓
['RoboCop (1987)']
c62.5% l5.402
```

Similar Concepts in Sci-Fi

```
['Unbreakable (2000)']
↓
['X-Men (2000)']
c80.8% l3.676

['Predator (1987)']
↓
['Aliens (1986)']
c85.2% l4.093
```

Same Series

```
['Back to the Future Part II (1989)']
↓
['Back to the Future (1985)']
c90.8% l3.213

['Spider-Man 2 (2004)']
↓
['Spider-Man (2002)']
c87.3% l4.331

['X2: X-Men United (2003)']
↓
['X-Men (2000)']
c86.8% l3.95

['Aliens (1986)']
↓
['Alien (1979)']
c82.5% l3.42
```

Phase IV - Genre Rules

In [20]:
```
def get_genres(ids):
    lists_of_lists_genres = movies.loc[ids].reset_index().genres
    series = lists_of_lists_genres.apply(pd.Series)
    return series.dropna().values.ravel()

users_genres = user_watched.apply(get_genres).apply(list)
users_genres
```

Out[20]:
```
userId
1      [Adventure, Animation, Children, Comedy, Crime...
2      [Action, Crime, Drama, Mystery, Sci-Fi, Thrill...
3         [Adventure, Animation, Children, Crime, Drama]
4      [Adventure, Animation, Children, Comedy, Drama...
5      [Adventure, Animation, Children, Drama, Musica...
                             ...
606    [Adventure, Animation, Children, Comedy, Crime...
607    [Action, Adventure, Comedy, Fantasy, Horror, T...
608    [Adventure, Animation, Children, Comedy, Drama...
609    [Adventure, Animation, Children, Comedy, Fantasy]
610    [Action, Adventure, Comedy, Crime, Drama, Film...
Name: movieId, Length: 610, dtype: object
```

Item generation

In [21]:
```python
te_ary = te.fit_transform(users_genres)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)

items = fpgrowth(watched_enc, min_support=0.3, use_colnames=True)
items
```

Out[21]:

	support	itemsets
0	0.714754	(Adventure)
1	0.642623	(Children)
2	0.640984	(Fantasy)
3	0.636066	(Animation)
4	0.611475	(Comedy)
...	...	...
146	0.314754	(Musical, Romance, Animation)
147	0.314754	(Musical, Children, Romance, Animation)
148	0.316393	(Musical, Children, Adventure)
149	0.309836	(Musical, Adventure, Animation)
150	0.308197	(Musical, Children, Adventure, Animation)

151 rows × 2 columns

In [22]:
```python
rules = association_rules(items, metric="confidence", min_threshold=0.8)
rules = rules.sort_values(by=["confidence"], ascending=False)
rules = rules[rules.lift > 1.5]

# rules = rules[rules.antecedents.apply(lambda x: len(x) == 1)]
# rules = rules[rules.antecedents.apply(lambda x: "Film-Noir" in x)]
# pretty_rules(rules)
```

As expected, we can see that it puts genres together that have similar tones, we can see the difference in the two section below and their respective rules.

```
['Sci-Fi']
↓
['Action']
c89.6% l1.523

['Crime', 'Action']
↓
['Thriller']
c97.3% l1.761

['Thriller']
↓
['Action']
c94.4% l1.603
```

vs

```
['Mystery']
↓
['Crime']
c92.2% l1.645


['Film-Noir']
↓
['Crime', 'Mystery', 'Drama']
c90.0% l2.429
```

vs

```
['Fantasy', 'Comedy', 'Animation']
↓
['Children', 'Adventure']
c95.3% l1.647

['Musical']
↓
['Children', 'Animation']
c95.3% l1.538

['Comedy', 'Romance']
↓
['Adventure']
c89.8% l1.257
```

We also took a look at low Confidence and low lift rules to see the inverse correlation of genres, which resulted in obviously uncommon pairings of genres

```
['Crime', 'Drama']
↓
['Children']
c38.1% l0.592

['Crime', 'Action', 'Thriller', 'Mystery']
↓
['Adventure']
c42.9% l0.6
```

Phase V – Incorporating Additional Variables

Bad Movies Recommendations

In [23]:
```
user_watched = ratings[ratings.rating <= 2].groupby("userId")["movieId"]
.apply(list)
te_ary = te.fit_transform(user_watched)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)
items = fpgrowth(watched_enc, min_support=0.015, use_colnames=True)
rules = association_rules(items, metric="confidence", min_threshold=0.4)
rules = rules[rules.lift > 10]
rules = rules.sort_values(by=["confidence"], ascending=False)
rules.antecedents = rules.antecedents.map(convert_ids).map(frozenset)
rules.consequents = rules.consequents.map(convert_ids).map(frozenset)
rules.head(5)
```

Out[23]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverag
44	(Rocky III (1982))	(Rocky IV (1985))	0.019011	0.022814	0.015209	0.800000	35.066667	0.01477
29	(Superman III (1983))	(Batman & Robin (1997))	0.022814	0.047529	0.017110	0.750000	15.780000	0.01602
33	(Lord of the Rings: The Two Towers, The (2002))	(Lord of the Rings: The Fellowship of the Ring...	0.022814	0.024715	0.017110	0.750000	30.346154	0.01654
25	(Superman IV: The Quest for Peace (1987))	(Batman & Robin (1997))	0.028517	0.047529	0.020913	0.733333	15.429333	0.01955
31	(Sister Act 2: Back in the Habit (1993))	(Angels in the Outfield (1994))	0.020913	0.026616	0.015209	0.727273	27.324675	0.01465

Low Ratings Discussion

We can improve on this, how about we incoorpe

HW 1 – Frequent Pattern Mining

Name: Robb Alexander and Ryan Bailis Class: CSCI349 Semester: 2021SP Instructor: Brian King

In [24]:
```
import numpy as np
import pandas as pd
import plotly.express as px
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules
```

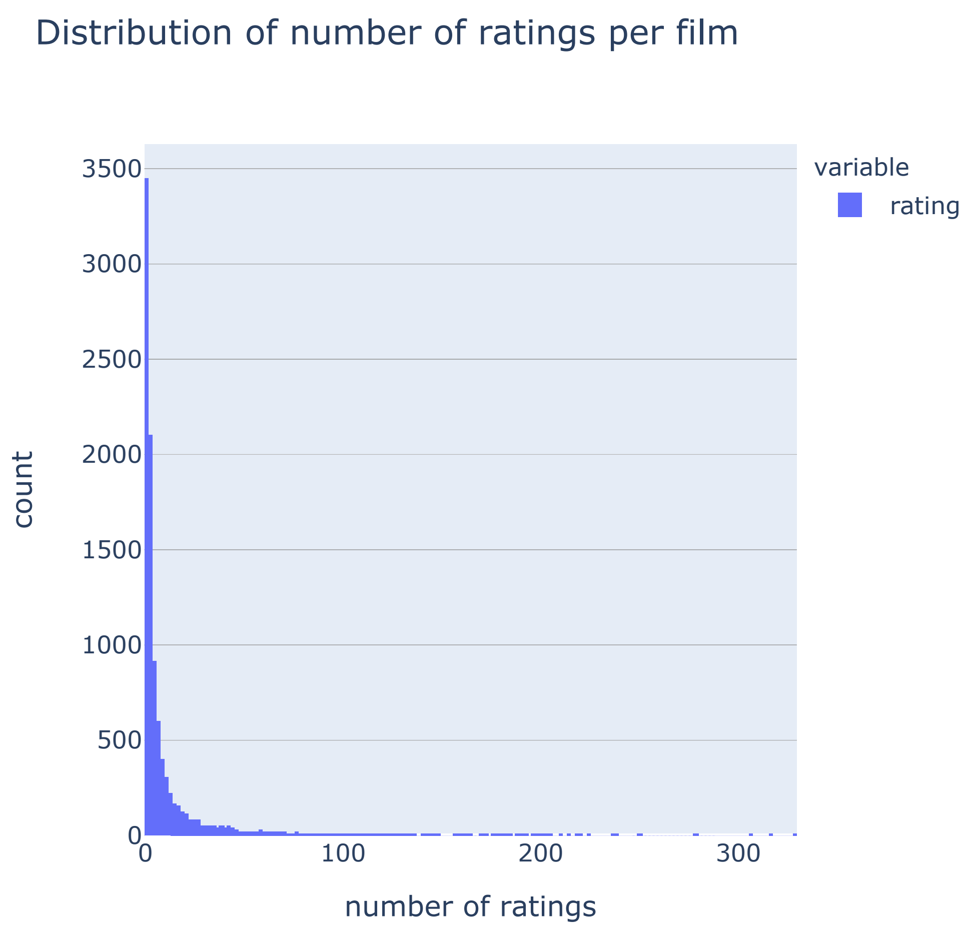
Phase I - EDA

In [25]:
```
ratings = pd.read_csv("/Users/rale/Documents/Programming/csci349_2021sp/data/ml-latest-small/ratings.csv")
```

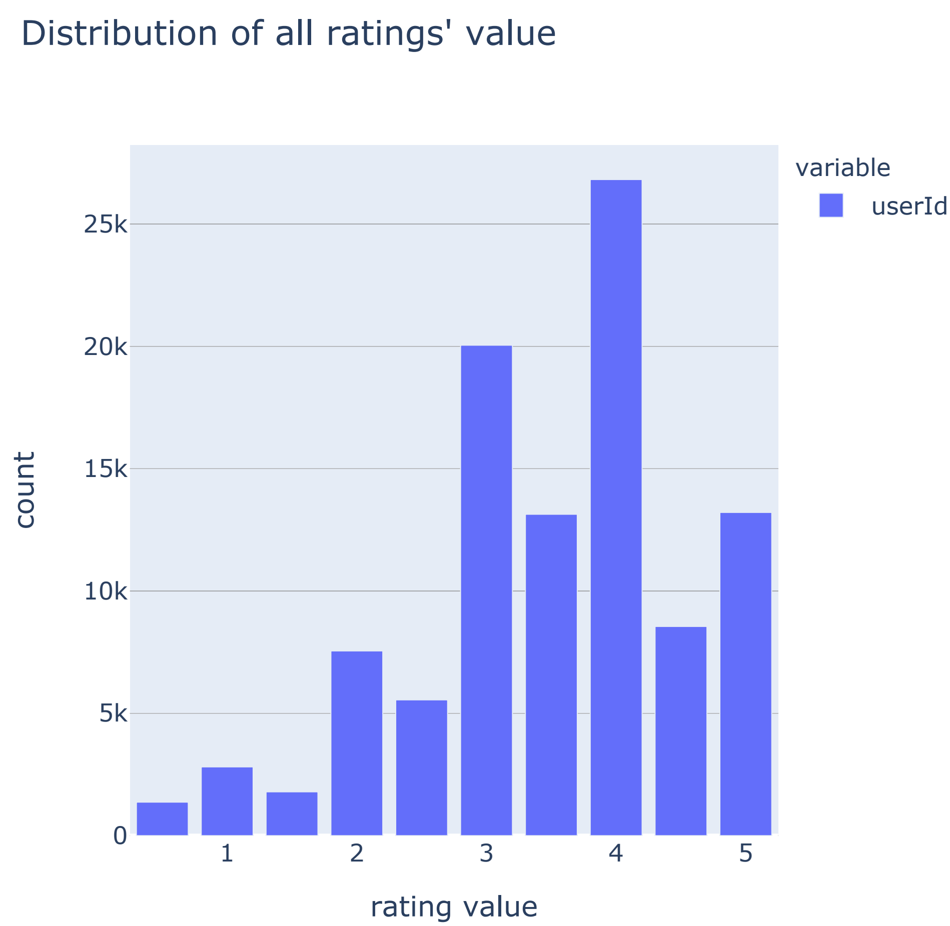
In [26]:
```
movies = pd.read_csv("/Users/rale/Documents/Programming/csci349_2021sp/data/ml-latest-small/movies.csv")
movies.set_index("movieId", inplace=True)
movies.genres = movies.genres.apply(str.split, args="|")
```

EDA plots

In [27]:
```
num_ratings_distribution = ratings.groupby("movieId").count().rating
px.histogram(num_ratings_distribution, labels={"value":"number of ratings"}, title="Distribution of number of ratings per film")
```



In [28]:
```
ratings_distribution = ratings.groupby("rating").count().userId
px.bar(ratings_distribution, labels={"rating":"rating value","value":"count"}, title="Distribution of all ratings' value")
# ratings_distribution
```



Max amount of Ratings

In [29]:
```
max_ratings_id = ratings.groupby("movieId").count().rating.sort_values(ascending=False).index[0]
movies.loc[max_ratings_id]
```

Out[29]:
```
title                Forrest Gump (1994)
genres     [Comedy, Drama, Romance, War]
Name: 356, dtype: object
```

Highest average ratings with over a 15 count

In [30]:

```
max_avg_ratings = ratings.groupby("movieId").mean()[ratings.groupby("movieId").count().rating > 15].rating.sort_values(ascending=False)
max_avg_ratings_ids = max_avg_ratings.index[:5]
top_films = movies.loc[max_avg_ratings_ids]
top_films.drop("genres", axis=1, inplace=True)
top_films["ratings"] = max_avg_ratings
top_films
```

Out[30]:

movieId	title	ratings
1104	Streetcar Named Desire, A (1951)	4.475000
318	Shawshank Redemption, The (1994)	4.429022
922	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	4.333333
3468	Hustler, The (1961)	4.333333
3435	Double Indemnity (1944)	4.323529

In [31]:

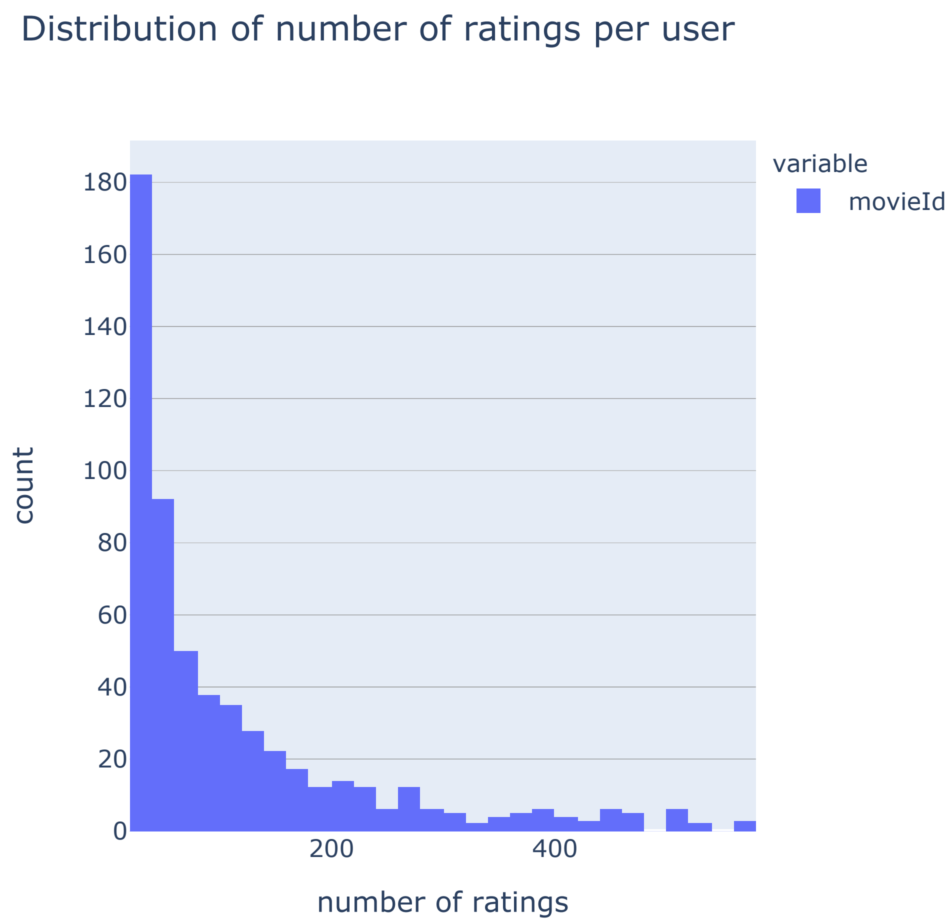
```
#### Lowest average ratings with over a 15 count

min_avg_ratings = ratings.groupby("movieId").mean()[ratings.groupby("movieId").count().rating > 15].rating.sort_values()
min_avg_ratings_ids = min_avg_ratings.index[:5]
low_films = movies.loc[min_avg_ratings_ids]
low_films.drop("genres", axis=1, inplace=True)
low_films["ratings"] = min_avg_ratings
low_films
```

Out[31]:

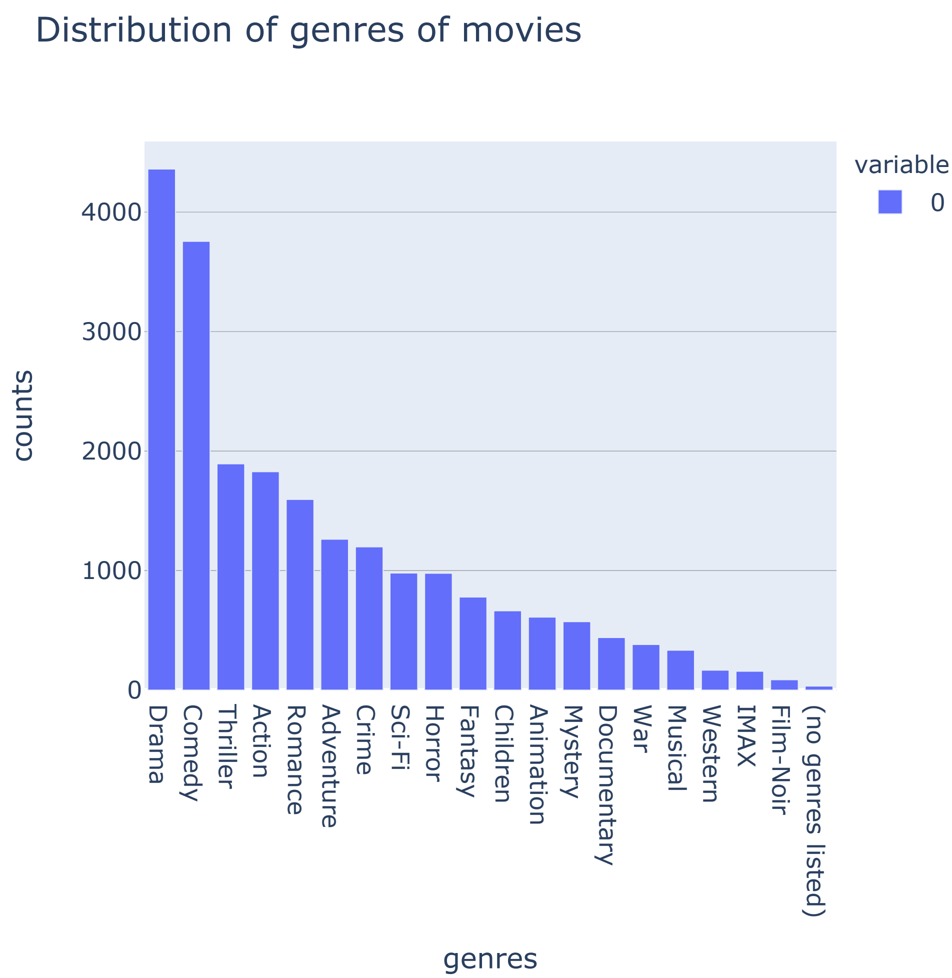
movieId	title	ratings
1556	Speed 2: Cruise Control (1997)	1.605263
3593	Battlefield Earth (2000)	1.657895
2643	Superman IV: The Quest for Peace (1987)	1.687500
1499	Anaconda (1997)	1.925926
2412	Rocky V (1990)	1.941176

In [32]:

```
# exclude outliers, cap at 600 for visuals
user_distribution = ratings.groupby("userId").count().movieId[ratings.gr
oupby("userId").count().movieId < 600]
px.histogram(user_distribution, labels={"value":"number of ratings"}, ti
tle="Distribution of number of ratings per user")
```



In [33]:
```
te = TransactionEncoder()
te_ary = te.fit_transform(movies.genres)
genres_enc = pd.DataFrame(te_ary, columns=te.columns_, index=movies.inde
x)
genres_counts = genres_enc.sum().sort_values(ascending=False)
px.bar(genres_counts, labels={"index":"genres","value":"counts"}, title=
"Distribution of genres of movies")
```



Phase II - Rules

Itemsets generation

First, we group by the users, and take all the movie ids and apply them into one list. Now with the user_watched being a list of 'transactions' filled with all the films they've watched, we can run the itemset generation and rule generation (using fpgrowth this time)

In [34]:
```
user_watched = ratings.groupby("userId")["movieId"].apply(list)
te_ary = te.fit_transform(user_watched)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)
watched_enc
```

Out[34]:

	1	2	3	4	5	6	7	8	9	10	...	193565	193567	1935
0	True	False	True	False	False	True	False	False	False	False	...	False	False	Fal
1	False	False	False	False	False	False	False	False	False	False	...	False	False	Fal
2	False	False	False	False	False	False	False	False	False	False	...	False	False	Fal
3	False	False	False	False	False	False	False	False	False	False	...	False	False	Fal
4	True	False	False	False	False	False	False	False	False	False	...	False	False	Fal
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
605	True	False	False	False	False	False	True	False	False	False	...	False	False	Fal
606	True	False	False	False	False	False	False	False	False	False	...	False	False	Fal
607	True	True	True	False	False	False	False	False	False	True	...	False	False	Fal
608	True	False	False	False	False	False	False	False	False	True	...	False	False	Fal
609	True	False	False	False	False	True	False	False	False	False	...	False	False	Fal

610 rows × 9724 columns

In [35]:
```
def convert_ids(ids):
    """
    converts movie ids to their respective names
    in strings from the `movies` dataframe

    :param ids: array-like
    :return: pandas.Series
    """
    return movies.loc[ids].reset_index().title
```

In [36]:
```
items = fpgrowth(watched_enc, min_support=0.2, use_colnames=True)
items_20 = items[:20].copy()
items_20.itemsets = items_20.itemsets.apply(convert_ids)
items_20
```

Out[36]:

	support	itemsets
0	0.539344	Forrest Gump (1994)
1	0.503279	Pulp Fiction (1994)
2	0.457377	Silence of the Lambs, The (1991)
3	0.455738	Matrix, The (1999)
4	0.411475	Star Wars: Episode IV - A New Hope (1977)
5	0.390164	Jurassic Park (1993)
6	0.388525	Braveheart (1995)
7	0.360656	Schindler's List (1993)
8	0.357377	Fight Club (1999)
9	0.352459	Toy Story (1995)
10	0.345902	Star Wars: Episode V - The Empire Strikes Back...
11	0.334426	Usual Suspects, The (1995)
12	0.334426	American Beauty (1999)
13	0.332787	Seven (a.k.a. Se7en) (1995)
14	0.331148	Independence Day (a.k.a. ID4) (1996)
15	0.327869	Raiders of the Lost Ark (Indiana Jones and the...
16	0.321311	Star Wars: Episode VI - Return of the Jedi (1983)
17	0.311475	Fugitive, The (1993)
18	0.309836	Batman (1989)
19	0.308197	Saving Private Ryan (1998)

Association Rules Generation

We can then use the `association_rules()` function to generate them using the confidence metric set at 70%. Then filter based on lift threshold of 2, and sort them by lift/confidence.
After this, we have to convert the movie ids with the `convert_ids()` function written above with the use of `map()` and revert them back to the frozenset type.

In [37]:

```
rules = association_rules(items, metric="confidence", min_threshold=0.7)
# rules = rules[rules.antecedents.apply(lambda x: len(x) == 1)]
rules = rules[rules.lift > 2]
rules = rules.sort_values(by=["lift", "confidence"], ascending=False)

rules.antecedents = rules.antecedents.map(convert_ids).map(frozenset)
rules.consequents = rules.consequents.map(convert_ids).map(frozenset)
rules.head(5)

# formatted printing
def pretty_rules(df):
    for index, rule in df.iterrows():
        print(list(rule.antecedents))
        print("↓")
        print(list(rule.consequents))
        print('c' + str(round(rule.confidence* 100,1)) + '%', 'l' + str(
round(rule.lift,3)), '\n')
```

Association Rules Analysis

First we analyzed the rules with only one antecedent, which can narrow and focus a single type of film to more films. Some interesting ones are:

```
['Beauty and the Beast (1991)']
↓
['Aladdin (1992)']
c0.842 l2.808

['Mission: Impossible (1996)']
↓
['Independence Day (a.k.a. ID4) (1996)']
c79.6% l2.405
```

There was also a large number of LotR rules, but the highest one with one antecedent was the last film -> the first two films. This is a pattern we saw in most of the rules, if you watched the last film, it means that you normally have seen the prequels prior. This way, it might be good to think the consequents as necessary films to watch, before the consequent. While most of the first few films in a series as the antecedent had much lower lift values, which could mean that watching earlier film doesn't entail you are interested in the sequels, but if you have seen the sequels you are very likely to have watched the prequels.

```
['Lord of the Rings: The Return of the King, The (2003)']
↓
['Lord of the Rings: The Fellowship of the Ring, The (2001)', 'Lord of the R
ings: The Two Towers, The (2002)']
c83.2% l3.059

['Indiana Jones and the Last Crusade (1989)']
↓
['Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1
981)']
c87.9% l2.68

['Godfather: Part II, The (1974)']
↓
['Godfather, The (1972)']
c0.969 l3.079
```

The Star Wars rules also follow the idea of what was stated previously, but here there was an added interesting factor of having a different trilogy involved, i.e. Indiana Jones. Which I assume is from the deep relationship of Spielberg and Lucas.

```
['Star Wars: Episode VI - Return of the Jedi (1983)', 'Raiders of the Lost A
rk (Indiana Jones and the Raiders of the Lost Ark) (1981)']
↓
['Star Wars: Episode IV - A New Hope (1977)', 'Star Wars: Episode V - The Em
pire Strikes Back (1980)']
c96.1% l3.084
```

These rule also seems to be quite interesting, all of these movies have themes of memory and psychosis, 'mind-bending' at times, with big twists Very apt for them to be correlated.

```
['Memento (2000)']
↓
['Fight Club (1999)']
c77.4% l2.165

['Sixth Sense, The (1999)']
↓
['Fight Club (1999)'] c71.5% l2.001
```

These two below also illustrate the idea that genres will be a very powerful tool in recommending movies with similar interests. Here, I would assume it to be crime-action detective-y films.

```
['True Lies (1994)']
↓
['Batman (1989)']
c76.4% l2.466

['True Lies (1994)']
↓
['Fugitive, The (1993)']
c72.5% l2.327
```

Other rules with large groups of antecedents are all just groups of popular films that do not have much in common besides them being well received on average.

```
['Pulp Fiction (1994)', 'Fight Club (1999)', 'Forrest Gump (1994)']
↓
['Matrix, The (1999)']
c95.4% l2.093
```

Phase III - Genre

Helper Filtering Function

In [38]:
```
def filter_ratings(genre_name):
    """
    get ratings for movies that belong to a specific genre

    :param genre_name: string of genre in movies df column genres
    :return: pandas.DataFrame filtered ratings by genre
    """
    genre_ids = set(movies[movies.genres.apply(lambda x: genre_name in x
)].index)
    genre_ratings = ratings[ratings.movieId.apply(lambda x: x in genre_i
ds)]
    return genre_ratings
```

Rule Generation Function

In [39]:
```
def generate_rules(genre, min_sup=0.2, min_conf=0.7, min_lift=0, single_
ant=False):
    """
    get ratings for movies that belong to a specific genre

    :param genre: string of genre in movies df column genres
    :param min_sup: min support for fpgrowth function
    :param min_conf: min confidence for association_rules function
    :param min_lift: min lift for filtering
    :param single_ant: filter rules to include only 1 antecedent
    :return: pandas.DataFrame of association rules
    """
    genre_ratings = filter_ratings(genre)
    genre_user_watched = genre_ratings.groupby("userId")["movieId"].appl
y(list)
    genre_te_ary = te.fit_transform(genre_user_watched)
    genre_watched_enc = pd.DataFrame(genre_te_ary, columns=te.columns_)

    genre_items = fpgrowth(genre_watched_enc, min_support=min_sup, use_c
olnames=True)
    genre_rules = association_rules(genre_items, metric="confidence", mi
n_threshold=min_conf)
    genre_rules = genre_rules.sort_values(by=["confidence"], ascending=F
alse)
    genre_rules = genre_rules[genre_rules.lift > min_lift]
    if single_ant:
        genre_rules = genre_rules[genre_rules.antecedents.apply(lambda x
: len(x) == 1)]

    genre_rules.antecedents = genre_rules.antecedents.map(convert_ids).m
ap(frozenset)
    genre_rules.consequents = genre_rules.consequents.map(convert_ids).m
ap(frozenset)
    return genre_rules
```

Discussion

With the genre rules, what we notice instantly is that now we can differentiate the sub-genres of comedy itself. This is very good because we can recommend a type of sub-genre if that is detected in the user's movie habits. e.g. animated-comedy vs dark-adult-comedy vs family-comedy

It can be seen that this works much better than a general total rule-set since there is less bias for popular films, this seems like a good way to weight the films based on interest from the users themselves. This approach helps narrow down what the users would like much better than the original approach.

In [40]:
```
comedy = generate_rules("Comedy", min_sup=0.09,min_conf=0.85, min_lift=4
)
# pretty_rules(comedy)
```

Comedy Rules

Animated-Comedy

```
['Incredibles, The (2004)', 'Monsters, Inc. (2001)', 'Shrek 2 (2004)']
↓
['Finding Nemo (2003)']
c94.9% l4.1
```

Family-Comedy

```
['Back to the Future Part II (1989)', 'Toy Story (1995)']
↓
['Back to the Future (1985)', 'Forrest Gump (1994)']
c96.5% l4.321

['Ghost (1990)', 'Mrs. Doubtfire (1993)', 'Sleepless in Seattle (1993)']
↓
['Pretty Woman (1990)', 'Forrest Gump (1994)']
c93.3% l4.547
```

Dark-Comedy

```
["Monty Python's Life of Brian (1979)", 'Fargo (1996)']
↓
['Monty Python and the Holy Grail (1975)']
c92.1% l4.123

['True Lies (1994)', 'Addams Family Values (1993)']
↓
['Batman Forever (1995)']
c91.7% l4.075

['Austin Powers: International Man of Mystery (1997)', 'Men in Black (a.k.a.
MIB) (1997)', 'Forrest Gump (1994)']
↓
['Austin Powers: The Spy Who Shagged Me (1999)']
c86.8% l4.367
```

In [41]:
```
romance = generate_rules("Romance", min_sup=0.08, min_conf=0.6, min_lift
=3)
# pretty_rules(romance)
```

Out of all the rules, the romance genre showed a trend of similar release years, much more than the other two . This could be related to how romance and eras have an interlinked connection.

Again we see sub-genres from the outputted rules.
Animated-Romance

```
['Shrek 2 (2004)']
↓
['Shrek (2001)']
c90.2% l3.216

['Cinderella (1950)']
↓
['Beauty and the Beast (1991)']
c81.0% l3.36
```

manic-pixie-dream-girl-Romance

```
['Garden State (2004)']
↓
['Eternal Sunshine of the Spotless Mind (2004)']
c85.4% l3.951

['Garden State (2004)']
↓
['Lost in Translation (2003)']
c70.8% l5.801
```

Action-Romance

```
['Twister (1996)', 'True Lies (1994)']
↓
['Speed (1994)']
c87.1% l3.088
```

Popular-Romance

```
['Sleepless in Seattle (1993)', 'Forrest Gump (1994)', 'Ghost (1990)']
↓
['Pretty Woman (1990)']
c92.6% l4.159
```

In [42]:

```
scifi = generate_rules("Sci-Fi", min_sup=0.2, min_conf=0.6)
# pretty_rules(scifi)
# scifi
```

Dystopian Sci-Fi

```
['RoboCop (1987)']
↓
['Terminator, The (1984)']
c87.1% l4.025

['Total Recall (1990)']
↓
['RoboCop (1987)']
c62.5% l5.402
```

Similar Concepts in Sci-Fi

```
['Unbreakable (2000)']
↓
['X-Men (2000)']
c80.8% l3.676

['Predator (1987)']
↓
['Aliens (1986)']
c85.2% l4.093
```

Same Series

```
['Back to the Future Part II (1989)']
↓
['Back to the Future (1985)']
c90.8% l3.213

['Spider-Man 2 (2004)']
↓
['Spider-Man (2002)']
c87.3% l4.331

['X2: X-Men United (2003)']
↓
['X-Men (2000)']
c86.8% l3.95

['Aliens (1986)']
↓
['Alien (1979)']
c82.5% l3.42
```

Phase IV - Genre Rules

In [43]:
```python
def get_genres(ids):
    lists_of_lists_genres = movies.loc[ids].reset_index().genres
    series = lists_of_lists_genres.apply(pd.Series)
    return series.dropna().values.ravel()

users_genres = user_watched.apply(get_genres).apply(list)
users_genres
```

Out[43]:
```
userId
1      [Adventure, Animation, Children, Comedy, Crime...
2      [Action, Crime, Drama, Mystery, Sci-Fi, Thrill...
3         [Adventure, Animation, Children, Crime, Drama]
4      [Adventure, Animation, Children, Comedy, Drama...
5      [Adventure, Animation, Children, Drama, Musica...
                             ...
606    [Adventure, Animation, Children, Comedy, Crime...
607    [Action, Adventure, Comedy, Fantasy, Horror, T...
608    [Adventure, Animation, Children, Comedy, Drama...
609    [Adventure, Animation, Children, Comedy, Fantasy]
610    [Action, Adventure, Comedy, Crime, Drama, Film...
Name: movieId, Length: 610, dtype: object
```

Item generation

In [44]:

```python
te_ary = te.fit_transform(users_genres)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)

items = fpgrowth(watched_enc, min_support=0.3, use_colnames=True)
items
```

Out[44]:

	support	itemsets
0	0.714754	(Adventure)
1	0.642623	(Children)
2	0.640984	(Fantasy)
3	0.636066	(Animation)
4	0.611475	(Comedy)
...	...	...
146	0.314754	(Musical, Romance, Animation)
147	0.314754	(Musical, Children, Romance, Animation)
148	0.316393	(Musical, Children, Adventure)
149	0.309836	(Musical, Adventure, Animation)
150	0.308197	(Musical, Children, Adventure, Animation)

151 rows × 2 columns

In [45]:

```python
rules = association_rules(items, metric="confidence", min_threshold=0.8)
rules = rules.sort_values(by=["confidence"], ascending=False)
rules = rules[rules.lift > 1.5]

# rules = rules[rules.antecedents.apply(lambda x: len(x) == 1)]
# rules = rules[rules.antecedents.apply(lambda x: "Film-Noir" in x)]
# pretty_rules(rules)
```

As expected, we can see that it puts genres together that have similar tones, we can see the difference in the two section below and their respective rules.

```
['Sci-Fi']
↓
['Action']
c89.6% l1.523

['Crime', 'Action']
↓
['Thriller']
c97.3% l1.761

['Thriller']
↓
['Action']
c94.4% l1.603
```

vs

```
['Mystery']
↓
['Crime']
c92.2% l1.645


['Film-Noir']
↓
['Crime', 'Mystery', 'Drama']
c90.0% l2.429
```

vs

```
['Fantasy', 'Comedy', 'Animation']
↓
['Children', 'Adventure']
c95.3% l1.647

['Musical']
↓
['Children', 'Animation']
c95.3% l1.538

['Comedy', 'Romance']
↓
['Adventure']
c89.8% l1.257
```

We also took a look at low Confidence and low lift rules to see the inverse correlation of genres, which resulted in obviously uncommon pairings of genres

```
['Crime', 'Drama']
↓
['Children']
c38.1% l0.592

['Crime', 'Action', 'Thriller', 'Mystery']
↓
['Adventure']
c42.9% l0.6
```

Phase V.i – Incorporating Additional Variables

Bad Movies Recommendations

In [46]:
```
user_watched = ratings[ratings.rating <= 2].groupby("userId")["movieId"]
.apply(list)
te_ary = te.fit_transform(user_watched)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)
items = fpgrowth(watched_enc, min_support=0.015, use_colnames=True)
rules = association_rules(items, metric="confidence", min_threshold=0.4)
rules = rules[rules.lift > 10]
rules = rules.sort_values(by=["confidence"], ascending=False)
rules.antecedents = rules.antecedents.map(convert_ids).map(frozenset)
rules.consequents = rules.consequents.map(convert_ids).map(frozenset)
rules.head()
```

Out[46]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverag
44	(Rocky III (1982))	(Rocky IV (1985))	0.019011	0.022814	0.015209	0.800000	35.066667	0.01477
29	(Superman III (1983))	(Batman & Robin (1997))	0.022814	0.047529	0.017110	0.750000	15.780000	0.01602
33	(Lord of the Rings: The Two Towers, The (2002))	(Lord of the Rings: The Fellowship of the Ring...	0.022814	0.024715	0.017110	0.750000	30.346154	0.01654
25	(Superman IV: The Quest for Peace (1987))	(Batman & Robin (1997))	0.028517	0.047529	0.020913	0.733333	15.429333	0.01955
31	(Sister Act 2: Back in the Habit (1993))	(Angels in the Outfield (1994))	0.020913	0.026616	0.015209	0.727273	27.324675	0.01465

Low Ratings Discussion

We can improve on this, how about we incorporate the time era to bad movies to narrow our focus.

Time Era and Low ratings

Add year as a string ranging from 1920s~2010s

In [47]:
```
import re
movies["year"] = movies.title.apply(lambda x: re.findall(r'.*\((\d\d\d
\d)\)', x))
bad_years = movies[movies.year.map(len) == 0].index
movies.drop(bad_years, inplace=True)
movies.year = movies.year.apply(lambda x: int(x[0]))
movies = movies[movies.year > 1920]
movies.year = movies.year.apply(lambda x: str(x/100)[3:])
movies.year = movies.year.apply(lambda x: x + '0'if len(x) == 1 else x)
movies.year = movies.year.apply(lambda x: str(int(x) // 10) + "0s")
movies
```

Out[47]:

	title	genres	year
movieId			
1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	90s
2	Jumanji (1995)	[Adventure, Children, Fantasy]	90s
3	Grumpier Old Men (1995)	[Comedy, Romance]	90s
4	Waiting to Exhale (1995)	[Comedy, Drama, Romance]	90s
5	Father of the Bride Part II (1995)	[Comedy]	90s
...	...	...	...
193581	Black Butler: Book of the Atlantic (2017)	[Action, Animation, Comedy, Fantasy]	10s
193583	No Game No Life: Zero (2017)	[Animation, Comedy, Fantasy]	10s
193585	Flint (2017)	[Drama]	10s
193587	Bungo Stray Dogs: Dead Apple (2018)	[Action, Animation]	10s
193609	Andrew Dice Clay: Dice Rules (1991)	[Comedy]	90s

9717 rows × 3 columns

Get people who give bad reviews for 90s films

In [48]:
```
decade_string = "90s"
decade_ids = set(movies[movies.year == decade_string].index)
decade = ratings[ratings.movieId.apply(lambda x: x in decade_ids)]
decade_ratings = decade[decade.rating <= 3]

decade_watched = decade_ratings.groupby("userId")["movieId"].apply(list)
decade_watched
```

Out[48]:
```
userId
1      [70, 223, 296, 316, 423, 500, 648, 673, 736, 7...
2                                                 [318]
3              [31, 527, 647, 688, 720, 1093, 2424, 6238]
4      [21, 32, 45, 47, 52, 58, 126, 171, 190, 222, 2...
5      [39, 150, 153, 253, 265, 266, 300, 316, 318, 3...
                             ...
606    [1, 7, 11, 19, 47, 140, 168, 172, 202, 225, 23...
607    [11, 25, 34, 112, 153, 204, 208, 296, 316, 337...
608    [1, 2, 3, 19, 24, 31, 39, 44, 48, 63, 65, 70, ...
609    [1, 110, 116, 137, 150, 161, 185, 208, 231, 28...
610    [153, 303, 318, 332, 344, 356, 412, 519, 849, ...
Name: movieId, Length: 561, dtype: object
```

Generate Rules

In [49]:
```
te_ary = te.fit_transform(decade_watched)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)
items = fpgrowth(watched_enc, min_support=0.03, use_colnames=True)

rules = association_rules(items, metric="confidence", min_threshold=0.6)
rules = rules[rules.antecedents.apply(lambda x: len(x) == 1)]
rules = rules[rules.lift > 2]
rules = rules.sort_values(by=["lift", "confidence"], ascending=False)

rules.antecedents = rules.antecedents.map(convert_ids).map(frozenset)
rules.consequents = rules.consequents.map(convert_ids).map(frozenset)
# pretty_rules(rules)
```

Again we see that the movies even if 'bad' and from the same decade still focuses most of the relations on genres. This could be used to suggest bad movies that people don't like but the user likes the genre of movie itself. You dont always have to suggest perfect movies cause the user might not be looking for that.

```
['Deep Impact (1998)']
↓
['Armageddon (1998)']
c65.4% l6.435

['Austin Powers: International Man of Mystery (1997)']
↓
['Austin Powers: The Spy Who Shagged Me (1999)']
c60.6% l5.667

['Maverick (1994)']
↓
['Die Hard: With a Vengeance (1995)']
c66.7% l5.582

['Congo (1995)']
↓
['Cliffhanger (1993)']
c70.7% l5.221

['Client, The (1994)']
↓
['Net, The (1995)']
c67.7% l5.067

['Brady Bunch Movie, The (1995)']
↓
['Mask, The (1994)']
c65.5% l4.039
```

Now lets look at what types of movies get the best ratings from that era

In [50]:
```
people_who_like_decade = decade[decade.rating >= 4.5].groupby("userId")[
"movieId"].apply(list)

transactions = []
for i in people_who_like_decade:
    for j in i:
        genres = movies.loc[j].genres.copy()
        # genres.append(decade_string)
        # transactions.append(genres)
        for k in genres:
            transactions.append([k, decade_string])

te_ary = te.fit_transform(transactions)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)
items = fpgrowth(watched_enc, min_support=0.05, use_colnames=True)
rules = association_rules(items, metric="confidence", min_threshold=0.0)
rules = rules[rules.antecedents.apply(lambda x: len(x) == 1 and decade_s
tring in x)]
rules = rules.sort_values(by=["lift", "confidence"], ascending=False)
rules.loc[:, ["antecedents","consequents","confidence"]].head(10)
```

Out[50]:

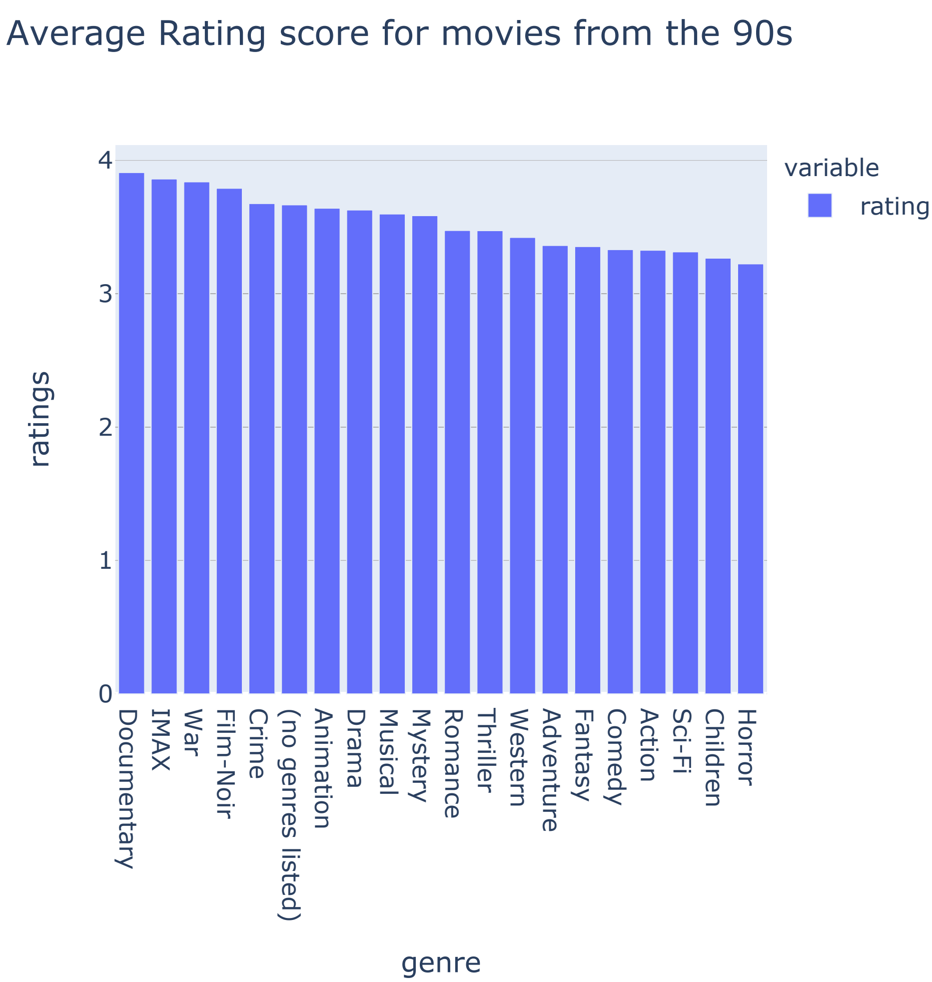
	antecedents	consequents	confidence
13	(90s)	(Drama)	0.192583
7	(90s)	(Comedy)	0.131944
0	(90s)	(Thriller)	0.115402
3	(90s)	(Crime)	0.096390
11	(90s)	(Action)	0.088934
8	(90s)	(Romance)	0.073922
5	(90s)	(Adventure)	0.057380

We can do a weighted calculation normally and redo this with a plot

find average ratings of 90s films for comparison. And we can see that this weighted average works much better than the botched association rules generated above.

In [51]:
```
alls = []
for index, i in decade.iterrows():
    genres = movies.loc[i.movieId].genres
    rs = [(x, i.rating) for x in genres]
    alls.extend(rs)
```

In [52]:
```
df = pd.DataFrame(alls, columns=['genre', 'rating'])
px.bar(df.groupby("genre").mean().sort_values(by="rating", ascending=False), labels={"value":"ratings"}, title="Average Rating score for movies from the 90s")
```



Discussion

In the end we can see that the bad movie within an era can be a good way to recommend niche films. This is reinforced by the fact that we couldn't find a relationship between a decade and its desirability by the people that enjoy them itself.

Phase V.ii – Incorporating Additional Variables

Tags

In [53]:
```
tags = pd.read_csv("/Users/rale/Documents/Programming/csci349_2021sp/data/ml-latest-small/tags.csv")
```

Get the most common tags (>5 occurances)

In [127]:
```
tags.tag = tags.tag.apply(str.lower)
common_tags = set(tags.tag.value_counts()[tags.tag.value_counts() > 3].i
ndex)
common_tags = tags[tags.tag.apply(lambda x: x in common_tags)]
common_tags
```

Out[127]:

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
2	2	60756	will ferrell	1445714992
6	2	106782	drugs	1445715054
7	2	106782	leonardo dicaprio	1445715051
8	2	106782	martin scorsese	1445715056
...	...	...	...	...
3670	599	2959	violence	1498456904
3671	599	2959	violent	1498456914
3672	600	273	gothic	1237739064
3673	606	1357	music	1176765393
3677	606	6107	world war ii	1178473747

2035 rows × 4 columns

Convert to vertical formatted data

In [128]:
```
transactions = common_tags.groupby("movieId").tag.apply(list)
te_ary = te.fit_transform(transactions)
watched_enc = pd.DataFrame(te_ary, columns=te.columns_)
items = fpgrowth(watched_enc, min_support=0.003, use_colnames=True)
```

In [130]:
```
rules = association_rules(items, metric="confidence", min_threshold=0.6)
rules = rules[rules.antecedents.apply(lambda x: len(x) == 1)]
rules = rules.sort_values(by=["lift", "confidence"], ascending=False)
# pretty_rules(rules)
```

Coorelated tags

Here we can see more complex rules that we can see from the analysis, these ideas are not synonymous concepts but rather associations that us humans have implicitly on films and genres in real life.

```
['will ferrell']
↓
['comedy']
c83.3% l40.789

['comic book']
↓
['superhero']
c60.0% l16.773

['wizards']
↓
['magic']
c100.0% l172.333

['mindfuck']
↓
['psychological']
c100.0% l61.5

['artificial intelligence']
↓
['robots', 'sci-fi']
c80.0% l178.24

['philosophy']
↓
['thought-provoking']
c80.0% l46.905
```

Problematic

But the issue we can see is that this type of analysis is dangerous because many of the rules returned were just similar wording of the same concepts.

['dreamlike']
↓
['atmospheric']
c63.6% l10.577

['hallucinatory']
↓
['surreal']
c100.0% l42.273