Reading all the previous tables

In [1]:

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

In [2]:

```
output_main_table = pd.read_csv("Main_table.csv").drop("Unnamed: 0",axis=1)
```

In [3]:

```
output_main_table.head()
```

Out[3]:

	Movie ID	Movie Name	Genre ID	Rating Score	Rating Count	Runtime	Box Office	Director ID	Writter ID	Studio ID	Movie Rating
0	M0	Joker (2019)	['G0', 'G1', 'G2']	68%	559	122 minutes	No Data	['D0']	['W0', 'W1']	S0	R
1	M1	Once Upon a Time In Hollywood (2019)	['G3', 'G1']	85%	535	159 minutes	No Data	['D1']	['W2']	S1	R
2	M2	Us (2019)	['G4', 'G2']	93%	520	120 minutes	No Data	['D2']	['W3']	S2	R
3	M3	Avengers: Endgame (2019)	['G0', 'G1', 'G5']	94%	514	182 minutes	No Data	['D3']	['W4', 'W5']	S3	PG-13
4	M4	Captain Marvel (2019)	['G0', 'G5']	78%	510	128 minutes	No Data	['D4', 'D5']	['W6', 'W7', 'W8']	S3	PG-13

In [4]:

```
movie_add_info = pd.read_csv("Add_Info_table.csv").drop("Unnamed: 0",axis=1)
movie_add_info.head()
```

Out[4]:

	Movie ID	In Theaters	On Disc/Streaming	Synopsis	Rating Description
0	M0	Oct 4, 2019	Dec 17, 2019	"Joker" centers around the iconic arch nemesis...	R (for strong bloody violence, disturbing beha...
1	M1	Jul 26, 2019	Nov 22, 2019	Quentin Tarantinos ninth feature film is a sto...	R (for language throughout, some strong graphi...
2	M2	Mar 22, 2019	Jun 18, 2019	Set in present day along the iconic Northern C...	R (for violence/terror, and language)
3	M3	Apr 26, 2019	Jul 30, 2019	The grave course of events set in motion by Th...	PG-13 (for sequences of sci-fi violence and ac...
4	M4	Mar 8, 2019	Jun 11, 2019	The story follows Carol Danvers as she becomes...	PG-13 (for sequences of sci-fi violence and ac...

In [5]:

```
studio_table= pd.read_csv("Studio_table.csv").drop("Unnamed: 0",axis=1)
studio_table.head()
```

Out[5]:

	Studio Name	Studio ID
0	Warner Bros. Pictures	S0
1	Columbia Pictures	S1
2	Universal Pictures	S2
3	Marvel Studios	S3
4	Walt Disney Pictures	S4

In [6]:

```
writter_table= pd.read_csv("writter_table.csv").drop("Unnamed: 0",axis=1)
writter_table.head()
```

Out[6]:

	Writter Name	Writter ID
0	Todd Phillips	W0
1	Scott Silver	W1
2	Quentin Tarantino	W2
3	Jordan Peele	W3
4	Christopher Markus	W4

In [7]:

```
director_table =  pd.read_csv("director_table.csv").drop("Unnamed: 0",axis=1)

director_table.head()
```

Out[7]:

	Director Name	Director ID
0	Todd Phillips	D0
1	Quentin Tarantino	D1
2	Jordan Peele	D2
3	Anthony Russo	D3
4	Anna Boden	D4

In [8]:

```
genre_table=pd.read_csv("genre_table.csv").drop("Unnamed: 0",axis=1)
genre_table.head()
```

Out[8]:

	Genre	Genre ID
0	Action & Adventure	G0
1	Drama	G1
2	Mystery & Suspense	G2
3	Comedy	G3
4	Horror	G4

In [9]:

```
cast_table_final=pd.read_csv("cast_table.csv").drop("Unnamed: 0",axis=1)
cast_table_final.head()
```

Out[9]:

	Movie ID	Cast ID	Cast Name
0	M0	A0	Joaquin Phoenix
1	M0	A1	Robert De Niro
2	M0	A2	Zazie Beetz
3	M0	A3	Bill Camp
4	M0	A4	Frances Conroy

In [10]:

```
critics_table= pd.read_csv("critics_table.csv").drop("Unnamed: 0",axis=1)
critics_table.head()
```

Out[10]:

	Critic Name	Critic ID
0	Josefine A.	C0
1	Lysalex Hernández A.	C1
2	Alex Abad-Santos	C2
3	Alana Joli Abbott	C3
4	Harrison Abbott	C4

In [11]:

```
critics_review_table= pd.read_csv("critics_review_table.csv").drop("Unnamed: 0",axis=1)
critics_review_table.head()
```

Out[11]:

	Movie ID	Critic ID	5 points score	Rating Score	Review	Date
0	M596	C0	4/5	88%	Shirley unquestionably does its subject justic...	Mar 2, 2020
1	M552	C0	5/5	99%	With a narrative that is both universal and de...	Mar 2, 2020
2	M880	C0	3/5	91%	With a set up as large as this, Bacurau would ...	Jul 8, 2019
3	M874	C0	4/5	99%	The film is a precious time capsule, preservin...	Feb 19, 2019
4	M536	C1	NaN	99%	an excellent opportunity to look at the past a...	Jul 29, 2019

Additional Data Cleaning & Data Transformation before Decision Tree

In [12]:

```
output_main_table
```

Out[12]:

	Movie ID	Movie Name	Genre ID	Rating Score	Rating Count	Runtime	Box Office	Director ID	Writer ID	St
0	M0	Joker (2019)	['G0', 'G1', 'G2']	68%	559	122 minutes	No Data	['D0']	['W0', 'W1']	
1	M1	Once Upon a Time In Hollywood (2019)	['G3', 'G1']	85%	535	159 minutes	No Data	['D1']	['W2']	
2	M2	Us (2019)	['G4', 'G2']	93%	520	120 minutes	No Data	['D2']	['W3']	
3	M3	Avengers: Endgame (2019)	['G0', 'G1', 'G5']	94%	514	182 minutes	No Data	['D3']	['W4', 'W5']	
4	M4	Captain Marvel (2019)	['G0', 'G5']	78%	510	128 minutes	No Data	['D4', 'D5']	['W6', 'W7', 'W8']	
...	...	...	...	...	...	...	...	...	...	
995	M995	Tammy (2014)	['G3']	24%	181	96 minutes	$51,033,986	['D465']	['W892', 'W893']	
996	M996	Harry Potter and the Order of the Phoenix (2007)	['G0', 'G1', 'G7', 'G5']	78%	255	138 minutes	$291,980,108	['D98']	['W835']	
997	M997	American Ultra (2015)	['G0', 'G3']	43%	173	99 minutes	No Data	['D612']	['W1160']	
998	M998	The Campaign (2012)	['G3']	66%	204	86 minutes	$86,897,182	['D103']	['W117', 'W1161', 'W1162', 'W1163']	
999	M999	The Incredible Burt Wonderstone (2013)	['G3']	37%	193	100 minutes	$22,525,921	['D613']	['W439', 'W90', 'W91']	

1000 rows × 11 columns

In [13]:

```
def join_list(x):
    return ",".join(x)
```

In [14]:

```
import ast
```

In [148]:

```
# Change of format for each columns
```

Reference: https://arxiv.org/ftp/arxiv/papers/1209/1209.6070.pdf (https://arxiv.org/ftp/arxiv/papers/1209/1209.6070.pdf)

Based on this we create rankings for Director, Writer, Genre and Studio

In [15]:

```
output_main_table["Director ID"]=output_main_table["Director ID"].apply(lambda x : ast.lite
```

In [16]:

```
output_main_table["Writter ID"]=output_main_table["Writter ID"].apply(lambda x : ast.litera
```

In [17]:

```
output_main_table["Genre ID"]=output_main_table["Genre ID"].apply(lambda x : ast.literal_ev
```

In [18]:

```
output_main_table["Genre ID"]=output_main_table["Genre ID"].apply(join_list)
```

In [19]:

```
output_main_table["Director ID"]=output_main_table["Director ID"].apply(join_list)
```

In [20]:

```
output_main_table["Writter ID"]=output_main_table["Writter ID"].apply(join_list)
```

In [21]:

```
output_main_table["Rating Score"]=output_main_table["Rating Score"].str.rstrip('%').astype(
```

In [22]:

```
output_main_table[output_main_table['Director ID'].str.contains(director_table["Director ID
```

Out[22]:

```
1      85.0
9      97.0
10     97.0
11     90.0
12     99.0
       ...
956    49.0
966    93.0
979    76.0
989    61.0
998    66.0
Name: Rating Score, Length: 234, dtype: float64
```

In [23]:

```
def avg_score(column_name,table_name):
    Avg_Score=[]
    for i in range(len(table_name)):
        Avg_Score.append(output_main_table[output_main_table[column_name].str.contains(tabl
    return Avg_Score
```

In [24]:

```
director_table['Avg_Score']=avg_score("Director ID",director_table)
```

In [25]:

```
director_table["Director_Rank"] = director_table["Avg_Score"].rank(ascending = 0)
```

In [26]:

```
director_table.sort_values("Director_Rank")
```

Out[26]:

	Director Name	Director ID	Avg_Score	Director_Rank
250	Debra Granik	D250	100.0	1.5
217	Paul King (VII)	D217	100.0	1.5
265	Hirokazu Koreeda	D265	99.0	5.0
121	Bo Burnham	D121	99.0	5.0
372	Todd Douglas Miller	D372	99.0	5.0
...	...	...	...	...
572	Pierre Morel	D572	14.5	610.0
459	Chris Addison	D459	14.0	611.5
498	David Frankel	D498	14.0	611.5
388	James Foley	D388	12.0	613.0
308	Josh Trank	D308	9.0	614.0

614 rows × 4 columns

In [27]:

```
genre_table['Avg_Score']=avg_score("Genre ID",genre_table)
genre_table["Genre_Rank"] = genre_table["Avg_Score"].rank(ascending = 0)
genre_table.sort_values("Genre_Rank")
```

Out[27]:

	Genre	Genre ID	Avg_Score	Genre_Rank
18	Television	G18	97.000000	1.0
17	Gay & Lesbian	G17	93.000000	2.0
13	Documentary	G13	90.333333	3.0
14	Special Interest	G14	86.333333	4.0
8	Art House & International	G8	85.194444	5.0
11	Sports & Fitness	G11	84.000000	6.0
6	Animation	G6	80.450000	7.0
15	Cult Movies	G15	77.000000	8.0
7	Kids & Family	G7	76.014286	9.0
1	Drama	G1	74.478049	10.0
10	Musical & Performing Arts	G10	73.851852	11.0
3	Comedy	G3	71.724806	12.0
2	Mystery & Suspense	G2	71.605405	13.0
9	Romance	G9	70.513889	14.0
4	Horror	G4	68.905405	15.0
5	Science Fiction & Fantasy	G5	65.486364	16.0
0	Action & Adventure	G0	63.869318	17.0
12	Western	G12	60.700000	18.0
16	Classics	G16	55.800000	19.0

In [28]:

```
writter_table['Avg_Score']=avg_score("Writter ID",writter_table)
writter_table["Writer_Rank"] = writter_table["Avg_Score"].rank(ascending = 0)
writter_table.sort_values("Writer_Rank")
```

Out[28]:

	Writter Name	Writter ID	Avg_Score	Writer_Rank
399	Simon Farnaby	W399	100.0	2.5
456	Debra Granik	W456	100.0	2.5
457	Anne Rosellini	W457	100.0	2.5
398	Paul King (VII)	W398	100.0	2.5
217	Bo Burnham	W217	99.0	7.5
...	...	...	...	...
1099	Gary Whitta	W1099	11.0	1160.0
566	Jeremy Slater	W566	9.0	1162.5
567	Josh Trank	W567	9.0	1162.5
1101	Hamish McColl	W1101	9.0	1162.5
1100	Conor McPherson	W1100	9.0	1162.5

1164 rows × 4 columns

In [29]:

```
studio_table
```

Out[29]:

	Studio Name	Studio ID
0	Warner Bros. Pictures	S0
1	Columbia Pictures	S1
2	Universal Pictures	S2
3	Marvel Studios	S3
4	Walt Disney Pictures	S4
...	...	...
159	Sony Pictures/Columbia Pictures	S159
160	Miramax	S160
161	Zeitgeist Films	S161
162	Paramount/Dreamworks Animation	S162
163	Participant Media	S163

164 rows × 2 columns

In [30]:

```
studio_table['Avg_Score']=avg_score("Studio ID",studio_table)
studio_table["Studio_Rank"] = studio_table["Avg_Score"].rank(ascending = 0)
studio_table.sort_values("Studio_Rank")
```

Out[30]:

	Studio Name	Studio ID	Avg_Score	Studio_Rank
122	Drafthouse Recommends	S122	98.0	1.0
20	United Artists	S20	97.0	3.5
136	Motto Pictures	S136	97.0	3.5
53	Film 44	S53	97.0	3.5
147	Greenwich Entertainment	S147	97.0	3.5
...	...	...	...	...
159	Sony Pictures/Columbia Pictures	S159	36.0	160.0
123	Lionsgate and CBS Films	S123	34.0	161.0
152	Paramount/Dreamworks	S152	20.0	162.5
85	Aviron Pictures	S85	20.0	162.5
157	Disney+	S157	9.0	164.0

164 rows × 4 columns

In [31]:

```
Rating_rank = [output_main_table[output_main_table["Movie Rating"]==i]["Rating Score"].mean
Rating_list = list(output_main_table["Movie Rating"].unique())
data_rate_1 = {
    "Rating Score" : Rating_rank,
    "Movie Rating" : Rating_list
}
df_rate_1 = pd.DataFrame(data_rate_1)
```

In [32]:

```
df_rate_1["Rating_rank"]=df_rate_1["Rating Score"].rank(ascending=0)
df_rate_1 = df_rate_1.drop("Rating Score",axis=1)
df_rate_1
```

Out[32]:

	Movie Rating	Rating_rank
0	R	5.0
1	PG-13	6.0
2	G	2.0
3	PG	4.0
4	NR	1.0
5	NC17	3.0

In [33]:

```
output_main_table= output_main_table.merge(df_rate_1, on='Movie Rating', how='left')
```

In [34]:

```
output_main_table
```

Out[34]:

	Movie ID	Movie Name	Genre ID	Rating Score	Rating Count	Runtime	Box Office	Director ID	
0	M0	Joker (2019)	G0,G1,G2	68.0	559	122 minutes	No Data	D0	
1	M1	Once Upon a Time In Hollywood (2019)	G3,G1	85.0	535	159 minutes	No Data	D1	
2	M2	Us (2019)	G4,G2	93.0	520	120 minutes	No Data	D2	
3	M3	Avengers: Endgame (2019)	G0,G1,G5	94.0	514	182 minutes	No Data	D3	
4	M4	Captain Marvel (2019)	G0,G5	78.0	510	128 minutes	No Data	D4,D5	
...	...	...	...	...	...	...	...	...	
995	M995	Tammy (2014)	G3	24.0	181	96 minutes	$51,033,986	D465	
996	M996	Harry Potter and the Order of the Phoenix (2007)	G0,G1,G7,G5	78.0	255	138 minutes	$291,980,108	D98	
997	M997	American Ultra (2015)	G0,G3	43.0	173	99 minutes	No Data	D612	
998	M998	The Campaign (2012)	G3	66.0	204	86 minutes	$86,897,182	D103	W117,'
999	M999	The Incredible Burt Wonderstone (2013)	G3	37.0	193	100 minutes	$22,525,921	D613	

1000 rows × 12 columns

In [35]:

```
movie_tmp = output_main_table[["Movie ID","Rating Score"]]
movie_tmp
```

Out[35]:

	Movie ID	Rating Score
0	M0	68.0
1	M1	85.0
2	M2	93.0
3	M3	94.0
4	M4	78.0
...	...	...
995	M995	24.0
996	M996	78.0
997	M997	43.0
998	M998	66.0
999	M999	37.0

1000 rows × 2 columns

In [36]:

```
cast_table_final = cast_table_final.merge(movie_tmp, on='Movie ID', how='left')
```

In [37]:

```
cast_score = [cast_table_final[cast_table_final["Cast ID"]==i]["Rating Score"].mean() for i
cast_list = cast_table_final["Cast ID"].unique()
data_cast_1 = {
    "Cast Score" : cast_score,
    "Cast ID" : cast_list
}
df_cast_1 = pd.DataFrame(data_cast_1)
```

In [38]:

```
cast_table_final = cast_table_final.merge(df_cast_1, on= "Cast ID", how = "left")
cast_table_final.head()
```

Out[38]:

	Movie ID	Cast ID	Cast Name	Rating Score	Cast Score
0	M0	A0	Joaquin Phoenix	68.0	77.750000
1	M0	A1	Robert De Niro	68.0	75.200000
2	M0	A2	Zazie Beetz	68.0	76.000000
3	M0	A3	Bill Camp	68.0	74.578947
4	M0	A4	Frances Conroy	68.0	68.000000

In [39]:

```
cast_rank_table = cast_table_final[["Cast ID","Cast Name","Cast Score"]].drop_duplicates()
```

In [40]:

```
cast_rank_table["Cast_rank"] = cast_rank_table["Cast Score"].rank(ascending=0)
```

In [41]:

```
cast_table_final = cast_table_final.merge(cast_rank_table[["Cast ID","Cast_rank"]],on="Cast
cast_table_final.head()
```

Out[41]:

	Movie ID	Cast ID	Cast Name	Rating Score	Cast Score	Cast_rank
0	M0	A0	Joaquin Phoenix	68.0	77.750000	9442.0
1	M0	A1	Robert De Niro	68.0	75.200000	10134.0
2	M0	A2	Zazie Beetz	68.0	76.000000	9936.5
3	M0	A3	Bill Camp	68.0	74.578947	10413.0
4	M0	A4	Frances Conroy	68.0	68.000000	12416.0

In [42]:

```
movie_tmp["Cast_rank"]= [cast_table_final[cast_table_final["Movie ID"]==i]["Cast_rank"].mea
```

```
C:\Users\dsu.jianwei\AppData\Local\Continuum\anaconda3\lib\site-packages\ipy
kernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [43]:

```
movie_tmp
```

Out[43]:

	Movie ID	Rating Score	Cast_rank
0	M0	68.0	12340.750000
1	M1	85.0	8618.271739
2	M2	93.0	4735.552632
3	M3	94.0	6475.130769
4	M4	78.0	10902.970588
...	...	...	...
995	M995	24.0	18332.243902
996	M996	78.0	9094.045455
997	M997	43.0	15952.762500
998	M998	66.0	12841.989474
999	M999	37.0	17275.958333

1000 rows × 3 columns

In [44]:

```
DT_table = output_main_table.copy()
```

In [45]:

```
DT_table["Cast_rank"] = movie_tmp["Cast_rank"]
```

In [46]:

```
def Average(lst):
    return sum(lst) / len(lst)
```

In [47]:

```
import ast
```

In [48]:

```
director_Rank_tmp=[]
for i in range(1000):
    try:
        director_Rank_tmp.append(Average([float(director_table[director_table["Director ID"
    except:
        director_Rank_tmp.append(np.NaN)
DT_table["Director_rank"] = director_Rank_tmp
```

In [49]:

```
DT_table["Director ID"][0]
```

Out[49]:

'D0'

In [50]:

```
Average([float(director_table[director_table["Director ID"]==i]["Director_Rank"]) for i in
```

Out[50]:

504.5

In [51]:

```
writter_Rank_tmp=[]
for i in range(1000):
    try:
        writter_Rank_tmp.append(Average([float(writter_table[writter_table["Writter ID"]==i
    except:
        writter_Rank_tmp.append(np.NaN)
DT_table["Writter_rank"] = writter_Rank_tmp
```

In [52]:

```
genre_Rank_tmp=[]
for i in range(1000):
    try:
        genre_Rank_tmp.append(Average([float(genre_table[genre_table["Genre ID"]==i]["Genre
    except:
        genre_Rank_tmp.append(np.NaN)
DT_table["Genre_rank"] = genre_Rank_tmp
```

In [53]:

```
studio_table
```

Out[53]:

	Studio Name	Studio ID	Avg_Score	Studio_Rank
0	Warner Bros. Pictures	S0	64.232323	136.0
1	Columbia Pictures	S1	71.587537	110.0
2	Universal Pictures	S2	68.900000	123.0
3	Marvel Studios	S3	72.235294	108.0
4	Walt Disney Pictures	S4	76.864078	94.0
...	...	...	...	...
159	Sony Pictures/Columbia Pictures	S159	36.000000	160.0
160	Miramax	S160	80.000000	81.0
161	Zeitgeist Films	S161	88.000000	49.5
162	Paramount/Dreamworks Animation	S162	73.000000	106.0
163	Participant Media	S163	82.000000	76.0

164 rows × 4 columns

In [54]:

```
DT_table = DT_table.merge(studio_table[["Studio ID","Studio_Rank"]],on="Studio ID",how ="le
```

In [55]:

```
DT_table["Year"]=[(int(DT_table["Movie Name"][i].split("(")[-1].split(")")[0])) for i in ra
```

In [56]:

```
runtime = []
for i in range(1000):
    try:
        runtime.append(int(DT_table["Runtime"][i].split(" ")[0]))
    except:
        runtime.append(np.NaN)
```

In [57]:

```
DT_table["Runtime"] = runtime
```

In [58]:

```
DT_table_cleaned = DT_table.copy()
```

In [59]:

```
DT_table_cleaned = DT_table_cleaned[['Movie ID', 'Rating Score', 'Rating Count',
       'Runtime', 'Rating_rank', 'Cast_rank', 'Director_rank',
       'Writter_rank', 'Genre_rank', 'Studio_Rank', 'Year']]
```

In [60]:

```
# Cleaned data for DT_table with the all the avg rankings for cast,director,writer,genre an
DT_table_cleaned
```

Out[60]:

	Movie ID	Rating Score	Rating Count	Runtime	Rating_rank	Cast_rank	Director_rank	Writter_rank	G
0	M0	68.0	559	122.0	5.0	12340.750000	504.5	785.500000	
1	M1	85.0	535	159.0	5.0	8618.271739	305.0	544.000000	
2	M2	93.0	520	120.0	5.0	4735.552632	324.0	576.000000	
3	M3	94.0	514	182.0	6.0	6475.130769	373.0	631.500000	
4	M4	78.0	510	128.0	6.0	10902.970588	391.5	665.333333	
...	...	...	...	...	...	...	...	...	
995	M995	24.0	181	96.0	5.0	18332.243902	588.0	1106.500000	
996	M996	78.0	255	138.0	6.0	9094.045455	372.0	927.000000	
997	M997	43.0	173	99.0	5.0	15952.762500	544.0	1020.000000	
998	M998	66.0	204	86.0	5.0	12841.989474	356.5	674.125000	
999	M999	37.0	193	100.0	6.0	17275.958333	563.5	872.666667	

1000 rows × 11 columns

Create category for Rating Score

In [61]:

```
def category(x):
    if x>=0 and x<=25:
        return "Terrible"
    elif x>25 and x<=50:
        return "Poor"
    elif x>50 and x<=75:
        return "Average"
    elif x>75:
        return "Excellent"
```

In [62]:

```
DT_table_cleaned["Rating Score"]=DT_table_cleaned["Rating Score"].apply(category)
```

In [63]:

```
DT_table_cleaned= DT_table_cleaned.dropna()
#Remove all NA data
```

In [64]:

```
DT_table_cleaned
```

Out[64]:

	Movie ID	Rating Score	Rating Count	Runtime	Rating_rank	Cast_rank	Director_rank	Writter_rank
0	M0	Average	559	122.0	5.0	12340.750000	504.5	785.500000
1	M1	Excellent	535	159.0	5.0	8618.271739	305.0	544.000000
2	M2	Excellent	520	120.0	5.0	4735.552632	324.0	576.000000
3	M3	Excellent	514	182.0	6.0	6475.130769	373.0	631.500000
4	M4	Excellent	510	128.0	6.0	10902.970588	391.5	665.333333
...	...	...	...	...	...	...	...	...
995	M995	Terrible	181	96.0	5.0	18332.243902	588.0	1106.500000
996	M996	Excellent	255	138.0	6.0	9094.045455	372.0	927.000000
997	M997	Poor	173	99.0	5.0	15952.762500	544.0	1020.000000
998	M998	Average	204	86.0	5.0	12841.989474	356.5	674.125000
999	M999	Poor	193	100.0	6.0	17275.958333	563.5	872.666667

933 rows × 11 columns

Start performing Decision tree

In [65]:

```
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
#a) Import and build a decision tree classifier.
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
```

In [66]:

```
DT_table_cleaned.head()
```

Out[66]:

	Movie ID	Rating Score	Rating Count	Runtime	Rating_rank	Cast_rank	Director_rank	Writter_rank	G
0	M0	Average	559	122.0	5.0	12340.750000	504.5	785.500000	
1	M1	Excellent	535	159.0	5.0	8618.271739	305.0	544.000000	
2	M2	Excellent	520	120.0	5.0	4735.552632	324.0	576.000000	
3	M3	Excellent	514	182.0	6.0	6475.130769	373.0	631.500000	
4	M4	Excellent	510	128.0	6.0	10902.970588	391.5	665.333333	

In [67]:

```
y = DT_table_cleaned.iloc[:,1]
X = DT_table_cleaned.iloc[:,2:]
```

In [68]:

```
X.head()
```

Out[68]:

	Rating Count	Runtime	Rating_rank	Cast_rank	Director_rank	Writter_rank	Genre_rank	Studio
0	559	122.0	5.0	12340.750000	504.5	785.500000	13.333333	
1	535	159.0	5.0	8618.271739	305.0	544.000000	11.000000	
2	520	120.0	5.0	4735.552632	324.0	576.000000	14.000000	
3	514	182.0	6.0	6475.130769	373.0	631.500000	14.333333	
4	510	128.0	6.0	10902.970588	391.5	665.333333	16.500000	

In [69]:

```
y.head()
```

Out[69]:

```
0      Average
1    Excellent
2    Excellent
3    Excellent
4    Excellent
Name: Rating Score, dtype: object
```

In [70]:

```
# Set random state = 719
rs = 719
X_mat = np.asmatrix(X)
X_train, X_test, y_train, y_test = train_test_split(X_mat, y, test_size=0.3, stratify=y, ra
```

In [71]:

```
# simple decision tree training
model = DecisionTreeClassifier(random_state=rs)
#Fit against the training data
model.fit(X_train, y_train)
```

Out[71]:

```
DecisionTreeClassifier(random_state=719)
```

In [72]:

```
#Performance of the model against training data
print("Train accuracy:", model.score(X_train, y_train), ", Test accuracy:", model.score(X_t
```

```
Train accuracy: 1.0 , Test accuracy: 0.8642857142857143
```

In [73]:

```
#Performance on the test data
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
#The accuracy is around 0.52 which indicates that the model is not overfitting
```

```
              precision    recall  f1-score   support

     Average       0.81      0.75      0.78        72
   Excellent       0.93      0.95      0.94       156
        Poor       0.74      0.83      0.78        41
    Terrible       0.75      0.55      0.63        11

    accuracy                           0.86       280
   macro avg       0.81      0.77      0.78       280
weighted avg       0.86      0.86      0.86       280
```

In [74]:

```
# Top feature
import numpy as np

# grab feature importances from the model and feature name from the original X
importances = model.feature_importances_
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

# limit to 20 features, you can leave this out to print out everything
indices = indices[:20]

for i in indices:
    print(feature_names[i], ':', importances[i])
```

```
Cast_rank : 0.7180913794009575
Writter_rank : 0.08210529585924431
Director_rank : 0.06620122219198238
Year : 0.040622487398997154
Rating Count : 0.03558015399719346
Runtime : 0.02056783795575847
Studio_Rank : 0.02030019763346319
Genre_rank : 0.012445962428546554
Rating_rank : 0.0040854631338567895
```

In [75]:

```
import os
os.environ["PATH"] += os.pathsep +  r"C:\Program Files (x86)\Graphviz2.38\bin"
```

In [76]:

```
fn= X.columns
```

In [77]:

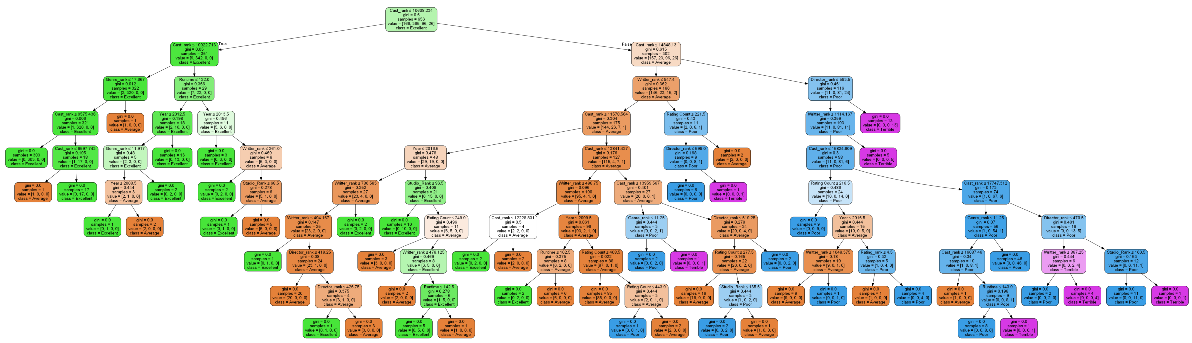
```
cn = y.unique()
```

In [78]:

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = fn,class_names=cn)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('MovieRating.png')
Image(graph.create_png())
```

```
C:\Users\dsu.jianwei\AppData\Local\Continuum\anaconda3\lib\site-packages\skl
earn\externals\six.py:31: FutureWarning: The module is deprecated in version
0.21 and will be removed in version 0.23 since we've dropped support for Pyt
hon 2.7. Please rely on the official version of six (https://pypi.org/projec
t/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
```

Out[78]:



In [167]:

```
#retrain with a small max_depth limit
def check_best(x):
    model = DecisionTreeClassifier(max_depth=x, random_state=rs)
    model.fit(X_train, y_train)

    print("Train accuracy:", model.score(X_train, y_train),", Test accuracy:", model.score(
    print()

    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))
```

In [178]:

```
for i in range(1,10):
    check_best(i)
```

```
   Excellent       0.97      0.92      0.95       156
        Poor       0.00      0.00      0.00        41
    Terrible       0.00      0.00      0.00        11

    accuracy                           0.76       280
   macro avg       0.37      0.47      0.40       280
weighted avg       0.67      0.76      0.70       280

Train accuracy: 0.8713629402756509 , Test accuracy: 0.8571428571428571

              precision    recall  f1-score   support

     Average       0.79      0.81      0.80        72
   Excellent       0.97      0.92      0.95       156
        Poor       0.64      0.93      0.76        41
    Terrible       0.00      0.00      0.00        11

    accuracy                           0.86       280
   macro avg       0.60      0.66      0.63       280
weighted avg       0.84      0.86      0.84       280
```

In [180]:

```
importances = model.feature_importances_
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

# limit to 20 features, you can leave this out to print out everything
indices = indices[:len(X.columns)]

for i in indices:
    print(feature_names[i], ':', importances[i])

# visualize
dotfile = StringIO()
export_graphviz(model, out_file=dotfile, feature_names=X.columns)
graph = pydot.graph_from_dot_data(dotfile.getvalue())
graph[0].write_png("DT_MovieRating.png") # saved in the following file
```

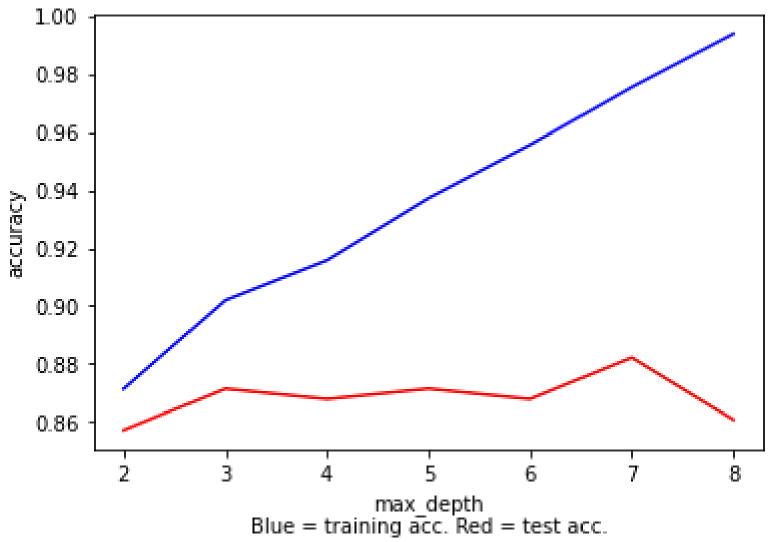
```
Cast_rank : 0.7686976933123122
Writter_rank : 0.07460588326696838
Director_rank : 0.06073479602458012
Year : 0.04346125095219195
Rating Count : 0.02171683646099267
Genre_rank : 0.013315705932423228
Studio_Rank : 0.013067945701250801
Runtime : 0.00439988834928055
Rating_rank : 0.0
```

In [181]:

```
test_score = []
train_score = []

# check the model performance for max depth from 2-20
for max_depth in range(2, len(X.columns)):
    model = DecisionTreeClassifier(max_depth=max_depth, random_state=rs)
    model.fit(X_train, y_train)

    test_score.append(model.score(X_test, y_test))
    train_score.append(model.score(X_train, y_train))
```

In [182]:

```
import matplotlib.pyplot as plt

# plot max depth hyperparameter values vs training and test accuracy score
plt.plot(range(2, len(X.columns)), train_score, 'b', range(2,len(X.columns)), test_score, '
plt.xlabel('max_depth\nBlue = training acc. Red = test acc.')
plt.ylabel('accuracy')
plt.show()
```



In [183]:

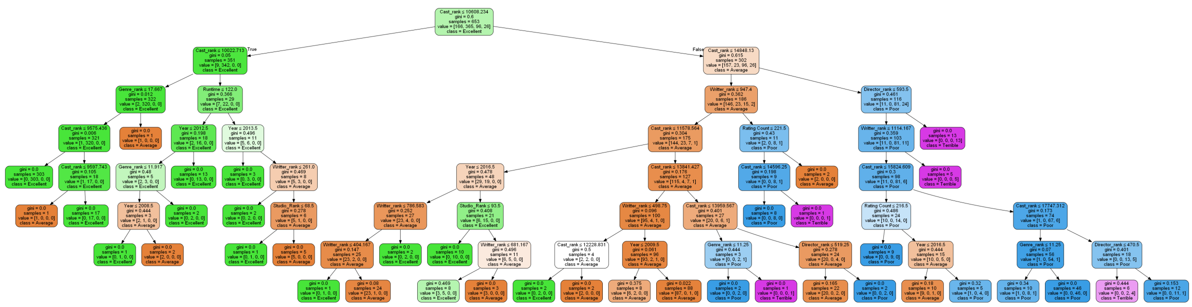
```
#retrain with a small max_depth limit

model = DecisionTreeClassifier(max_depth=7, random_state=rs)
model.fit(X_train, y_train)

print("Train accuracy:", model.score(X_train, y_train),", Test accuracy:", model.score(X_te
print()

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Train accuracy: 0.9754977029096478 , Test accuracy: 0.8821428571428571

              precision    recall  f1-score   support

     Average       0.85      0.78      0.81        72
   Excellent       0.94      0.96      0.95       156
        Poor       0.77      0.88      0.82        41
    Terrible       0.75      0.55      0.63        11

    accuracy                           0.88       280
   macro avg       0.83      0.79      0.80       280
weighted avg       0.88      0.88      0.88       280
```

In [184]:

```
dot_data = StringIO()
export_graphviz(model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = fn,class_names=cn)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('MovieRating_7depth.png')
Image(graph.create_png())
```

Out[184]:



In [185]:

```
from sklearn.model_selection import GridSearchCV
# grid search CV
params = {'criterion': ['gini', 'entropy'],
          'max_depth': range(2, len(X.columns)),
          'min_samples_leaf': range(20, 60, 10)}

cv = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs), cv=
cv.fit(X_train, y_train)

print("Train accuracy:", cv.score(X_train, y_train))
print("Test accuracy:", cv.score(X_test, y_test))

# test the best model
y_pred = cv.predict(X_test)
print(classification_report(y_test, y_pred))

# print parameters of the best model
print(cv.best_params_)
```

```
Train accuracy: 0.8836140888208269
Test accuracy: 0.8678571428571429
              precision    recall  f1-score   support

     Average       0.87      0.67      0.76        72
   Excellent       0.95      0.96      0.96       156
        Poor       0.67      0.88      0.76        41
    Terrible       0.69      0.82      0.75        11

    accuracy                           0.87       280
   macro avg       0.80      0.83      0.80       280
weighted avg       0.88      0.87      0.87       280

{'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 20}
```

In [186]:

```
def analyse_feature_importance(dm_model, feature_names, n_to_display=20):
    # grab feature importances from the model
    importances = dm_model.feature_importances_

    # sort them out in descending order
    indices = np.argsort(importances)
    indices = np.flip(indices, axis=0)

    # limit to 20 features, you can leave this out to print out everything
    indices = indices[:n_to_display]

    for i in indices:
        print(feature_names[i], ':', importances[i])

def visualize_decision_tree(dm_model, feature_names, save_name):
    dotfile = StringIO()
    export_graphviz(dm_model, out_file=dotfile, feature_names=feature_names)
    graph = pydot.graph_from_dot_data(dotfile.getvalue())
    graph[0].write_png(save_name) # saved in the following file
```

Since the Using Grid Search we found out that depth =4 has the highest accuracy.

In [187]:

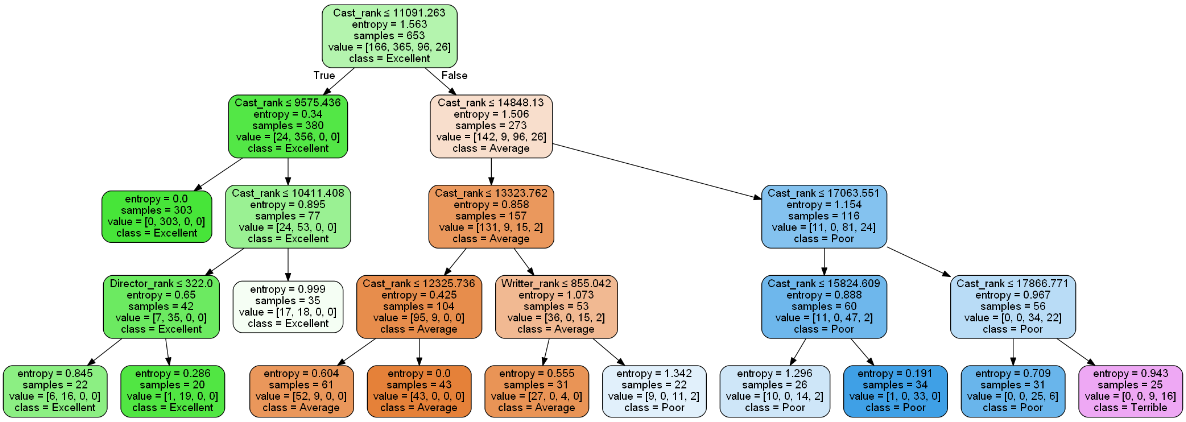
```
# do the feature importance and visualization analysis on GridSearchCV's best model

analyse_feature_importance(cv.best_estimator_, X.columns, 20)
visualize_decision_tree(cv.best_estimator_, X.columns, "lab2_optimize_d_tree.png")
```

```
Cast_rank : 0.9834479616351093
Writter_rank : 0.012796127562112975
Director_rank : 0.0037559108027778645
Year : 0.0
Studio_Rank : 0.0
Genre_rank : 0.0
Rating_rank : 0.0
Runtime : 0.0
Rating Count : 0.0
```

In [188]:

```python
dot_data = StringIO()
export_graphviz(cv.best_estimator_, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = X.columns,class_names=cn)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('MovieRating_2.png')
Image(graph.create_png())
```

Out[188]:



Question 4 Apriori and F&P algorithm

Data cleansing and data exploration

In [121]:

```python
critics_review_table= critics_review_table.merge(output_main_table[["Movie ID","Movie Name"
```

In [125]:

```python
critics_review_table.head()
```

Out[125]:

	Movie ID	Critic ID	5 points score	Rating Score	Review	Date	Movie Name
0	M596	C0	4/5	88%	Shirley unquestionably does its subject justic...	Mar 2, 2020	Shirley (2020)
1	M552	C0	5/5	99%	With a narrative that is both universal and de...	Mar 2, 2020	Never Rarely Sometimes Always (2020)
2	M880	C0	3/5	91%	With a set up as large as this, Bacurau would ...	Jul 8, 2019	Bacurau (Nighthawk) (2020)
3	M874	C0	4/5	99%	The film is a precious time capsule, preservin...	Feb 19, 2019	Amazing Grace (2019)
4	M536	C1	NaN	99%	an excellent opportunity to look at the past a...	Jul 29, 2019	Apollo 11 (2019)

In [122]:

```
len(critics_review_table["Critic ID"].unique())
```

Out[122]:

```
2528
```

In [123]:

```
critics_review_table_tmp = critics_review_table.copy()
```

In [124]:

```
critics_review_table_tmp=critics_review_table_tmp.sort_values(["Critic ID",'Date'])
```

In [126]:

```
critics_review_table_tmp_2 = critics_review_table_tmp[["Movie Name","Critic ID","Rating Sco
```

In [127]:

```
import datetime
```

In [128]:

```
critics_review_table_tmp_2["Date"]=pd.to_datetime(critics_review_table_tmp_2["Date"])
```

```
C:\Users\dsu.jianwei\AppData\Local\Continuum\anaconda3\lib\site-packages\ipy
kernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [129]:

```
critics_review_table_tmp_2= critics_review_table_tmp_2[critics_review_table_tmp_2["Rating S
```

In [130]:

```
critics_review_table_tmp_2["Rating Score"]=critics_review_table_tmp_2["Rating Score"].apply
```

In [133]:

```
critics_review_table_tmp_2.head()
```

Out[133]:

	Movie Name	Critic ID	Rating Score	Date
3	Amazing Grace (2019)	C0	99.0	2019-02-19
2	Bacurau (Nighthawk) (2020)	C0	91.0	2019-07-08
0	Shirley (2020)	C0	88.0	2020-03-02
1	Never Rarely Sometimes Always (2020)	C0	99.0	2020-03-02
4	Apollo 11 (2019)	C1	99.0	2019-07-29

In [134]:

```
critics_review_table_tmp_3 = critics_review_table_tmp_2[["Critic ID","Movie Name","Rating S
```

In [135]:

```
critics_review_table_tmp_3.head()
```

Out[135]:

	Critic ID	Movie Name	Rating Score
3	C0	Amazing Grace (2019)	99.0
2	C0	Bacurau (Nighthawk) (2020)	91.0
0	C0	Shirley (2020)	88.0
1	C0	Never Rarely Sometimes Always (2020)	99.0
4	C1	Apollo 11 (2019)	99.0

In [136]:

```
basket = (critics_review_table_tmp_3.groupby(['Critic ID', 'Movie Name'])['Rating Score']
          .sum().unstack().reset_index().fillna(0)
          .set_index('Critic ID'))
```

In [189]:

```
basket
```

Out[189]:

Movie Name	10 Cloverfield Lane (2016)	12 Strong (2018)	12 Years a Slave (2013)	127 Hours (2010)	13 Hours: The Secret Soldiers Of Benghazi (2016)	1917 (2020)	2 Guns (2013)	2012 (2009)	20th Century Women (2017)	21 Jump Street (2012)	...
Critic ID											
C0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
C1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
C10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	39.0	0.0	0.0	...
C100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
C1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
...	...	...	...	...	...	...	...	...	...	...	...
C995	0.0	0.0	0.0	0.0	0.0	0.0	64.0	0.0	0.0	0.0	...
C996	90.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
C997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
C998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
C999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

2528 rows × 1000 columns

Convert list to dataframe with boolean values

In [137]:

```
# For boolean values
def encode_units(x):
    if x <= 0:
        return False
    if x >= 1:
        return True


basket_sets = basket.applymap(encode_units)
```

In [138]:

```
basket_sets.head()
```

Out[138]:

Movie Name	10 Cloverfield Lane (2016)	12 Strong (2018)	12 Years a Slave (2013)	127 Hours (2010)	13 Hours: The Secret Soldiers Of Benghazi (2016)	1917 (2020)	2 Guns (2013)	2012 (2009)	20th Century Women (2017)	21 Jump Street (2012)	...
Critic ID											
C0	False	False	False	False	False	False	False	False	False	False	...
C1	False	False	False	False	False	False	False	False	False	False	...
C10	False	False	False	False	False	False	False	True	False	False	...
C100	False	False	False	False	False	False	False	False	False	False	...
C1000	False	False	False	False	False	False	False	False	False	False	...

5 rows × 1000 columns

Apriori Rules

In [139]:

```
from mlxtend.frequent_patterns import apriori
```

In [140]:

```
frequent_itemsets = apriori(basket_sets, min_support=0.05, use_colnames=True)
```

In [141]:

```
frequent_itemsets
```

Out[141]:

	support	itemsets
0	0.088608	(10 Cloverfield Lane (2016))
1	0.058544	(12 Strong (2018))
2	0.102057	(12 Years a Slave (2013))
3	0.058940	(127 Hours (2010))
4	0.067247	(13 Hours: The Secret Soldiers Of Benghazi (2...
...	...	...
72546	0.054984	(Captain Marvel (2019), Star Wars: The Rise o...
72547	0.050633	(Captain Marvel (2019), The Lion King (2019)...
72548	0.050633	(Star Wars: The Rise of Skywalker (2019), Us ...
72549	0.050633	(Star Wars: The Rise of Skywalker (2019), The...
72550	0.052215	(Captain Marvel (2019), Star Wars: The Rise o...

72551 rows × 2 columns

F&P Algorithm

In [142]:

```
from mlxtend.frequent_patterns import fpgrowth
```

In [143]:

```
frequent_itemsets_2 = fpgrowth(basket_sets, min_support=0.05, use_colnames=True)
```

In [144]:

```
frequent_itemsets_2
```

Out[144]:

	support	itemsets
0	0.066060	(Never Rarely Sometimes Always (2020))
1	0.064082	(Shirley (2020))
2	0.054984	(Amazing Grace (2019))
3	0.054193	(Bacurau (Nighthawk) (2020))
4	0.067247	(Apollo 11 (2019))
...	...	...
72546	0.051028	(Rogue One: A Star Wars Story (2016), Allied ...
72547	0.052215	(Dunkirk (2017), Life (2017))
72548	0.051820	(Logan (2017), Life (2017))
72549	0.051424	(Alien: Covenant (2017), Life (2017))
72550	0.051028	(Ready Player One (2018), Life (2017))

72551 rows × 2 columns

Mining Association Rules

In [145]:

```
from mlxtend.frequent_patterns import association_rules
```

In [146]:

```
first_rule = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
first_rule.head()
```

Out[146]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverag
0	(Avengers: Endgame (2019))	(10 Cloverfield Lane (2016))	0.187500	0.088608	0.053797	0.286920	3.238095	0.03718
1	(10 Cloverfield Lane (2016))	(Avengers: Endgame (2019))	0.088608	0.187500	0.053797	0.607143	3.238095	0.03718
2	(Avengers: Infinity War (2018))	(10 Cloverfield Lane (2016))	0.162579	0.088608	0.051424	0.316302	3.569691	0.03701
3	(10 Cloverfield Lane (2016))	(Avengers: Infinity War (2018))	0.088608	0.162579	0.051424	0.580357	3.569691	0.03701
4	(Batman v Superman: Dawn of Justice (2016))	(10 Cloverfield Lane (2016))	0.128956	0.088608	0.055775	0.432515	4.881245	0.04434

In [147]:

```
second_rule = association_rules(frequent_itemsets_2, metric="lift", min_threshold=1)
second_rule.head()
```

Out[147]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverag
0	(Ant-Man and the Wasp (2018))	(Avengers: Infinity War (2018))	0.147152	0.162579	0.102453	0.696237	4.282448	0.07852
1	(Avengers: Infinity War (2018))	(Ant-Man and the Wasp (2018))	0.162579	0.147152	0.102453	0.630170	4.282448	0.07852
2	(Ant-Man and the Wasp (2018))	(Avengers: Endgame (2019))	0.147152	0.187500	0.100870	0.685484	3.655914	0.07327
3	(Avengers: Endgame (2019))	(Ant-Man and the Wasp (2018))	0.187500	0.147152	0.100870	0.537975	3.655914	0.07327
4	(Ant-Man and the Wasp (2018))	(Black Panther (2018))	0.147152	0.168908	0.098101	0.666667	3.946916	0.07324