1. Analysis of A24 Movies data from Rotten Tomatoes

- correlation between audience and critic ratings?
- any changes in runtime, scores, etc over the years?
- best directors? best actors?

2. Simple models to predict scores

- linear model

Explore the data + Cleaning

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [ ]:
```
movies_df = pd.read_csv('full_scraped_a24_df.csv', index_col=0)
```

In [ ]:
```
## basic exploration
print(f'Size of dataset: {movies_df.shape}')

movies_df.describe()
```

```
Size of dataset: (136, 23)
```

Out[ ]:

	Release Year
count	136.000000
mean	2017.955882
std	3.236064
min	2012.000000
25%	2015.000000
50%	2017.000000
75%	2021.000000
max	2024.000000

Cleaning required -- there should be more numeric columns than this

In [ ]:
```
print(movies_df.columns)
pd.set_option('display.max_columns', 500)
(movies_df.head())
```

```
Index(['Title', 'Release Year', 'RT_score', 'URL', 'Director', 'Cast',
       'Director.1', 'Producer', 'Screenwriter', 'Distributor',
       'Production Co', 'Rating', 'Genre', 'Original Language',
       'Release Date (Theaters)', 'Release Date (Streaming)',
       'Box Office (Gross USA)', 'Runtime', 'Sound Mix', 'Aspect Ratio',
       'audience_score', 'number_audience_reviews',
       'Rerelease Date (Theaters)'],
      dtype='object')
```

Out[ ]:

	Title	Release Year	RT_score	URL	Director
0	Lady Bird	2017	99%	https://www.rottentomatoes.com/m/lady_bird	Greta Gerwig
1	Eighth Grade	2018	99%	https://www.rottentomatoes.com/m/eighth_grade	Bo Burnham
2	Moonlight	2016	98%	https://www.rottentomatoes.com/m/moonlight_2016	Barry Jenkins
3	Minari	2020	98%	https://www.rottentomatoes.com/m/minari	Lee Isaac Chung
4	Marcel the Shell with Shoes On	2021	98%	https://www.rottentomatoes.com/m/marcel_the_sh...	Dean Fleischer-Camp

Genres

In [ ]:

```
## Explore genres
from collections import Counter
movies_df['Genre'].value_counts()


all_genres = ','.join(movies_df['Genre']).split(',')
genre_counter = (Counter(all_genres))

print(f'Number of genres : {len(genre_counter)}')
genre_counter

## Sort out the counter to get arranged by frequency of genres
sorted(genre_counter.items(), key= lambda item: -item[1])
```

```
Number of genres : 22
```

Out[ ]:
```
[('Drama', 90),
 ('Mystery & Thriller', 46),
 ('Comedy', 45),
 ('Horror', 24),
 ('Romance', 15),
 ('Sci-Fi', 10),
 ('Crime', 9),
 ('Action', 7),
 ('Fantasy', 7),
 ('LGBTQ+', 5),
 ('Adventure', 5),
 ('Biography', 4),
 ('Music', 4),
 ('Western', 4),
 ('History', 4),
 ('War', 4),
 ('Documentary', 3),
 ('Holiday', 2),
 ('Kids & Family', 1),
 ('Animation', 1),
 ('Sports', 1),
 ('Musical', 1)]
```

In [ ]:
```
genre_freq = pd.DataFrame(genre_counter.items(), columns=['Genre', 'Frequenc
genre_freq['Cumulative Sum'] = genre_freq['Frequency'].cumsum()
genre_freq['Cumulative Sum Percentage'] = ((genre_freq['Cumulative Sum']/ger
genre_freq.reset_index(drop = True)
```

Out[ ]:

	Genre	Frequency	Cumulative Sum	Cumulative Sum Percentage
0	Drama	90	90	30.82
1	Mystery & Thriller	46	136	46.58
2	Comedy	45	181	61.99
3	Horror	24	205	70.21
4	Romance	15	220	75.34
5	Sci-Fi	10	230	78.77
6	Crime	9	239	81.85
7	Action	7	246	84.25
8	Fantasy	7	253	86.64
9	LGBTQ+	5	258	88.36
10	Adventure	5	263	90.07
11	War	4	267	91.44
12	Biography	4	271	92.81
13	Music	4	275	94.18
14	Western	4	279	95.55
15	History	4	283	96.92
16	Documentary	3	286	97.95
17	Holiday	2	288	98.63
18	Sports	1	289	98.97
19	Kids & Family	1	290	99.32
20	Animation	1	291	99.66
21	Musical	1	292	100.00

We can see that the big 4 genres by A24 are

- Drama : 30% of all genres
- Mystery and thriller : a personal favourite
- Comedy : another personal favourite
- Horror : what brought me to A24

These 4 genres made up 75% of all genres by A24.

Genre correlation

In [ ]:
```
print(f"Percentage of movies with more than 1 genre: {movies_df['Genre'].st
```

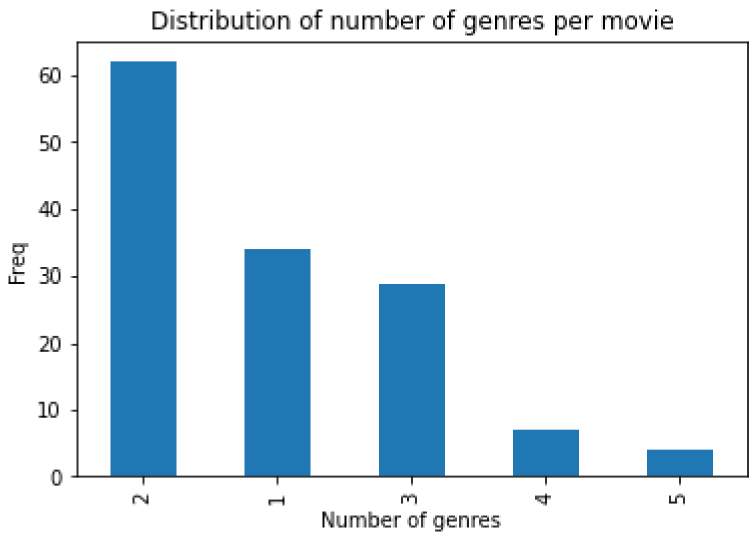
```
Percentage of movies with more than 1 genre: 75.0 %
```

As we can see most of the movies contain more than one genre. Let us see which are some popular combinations of genres

To do so we will need to split up the genres into a column for each possible genre, i.e. **one hot encoding**.

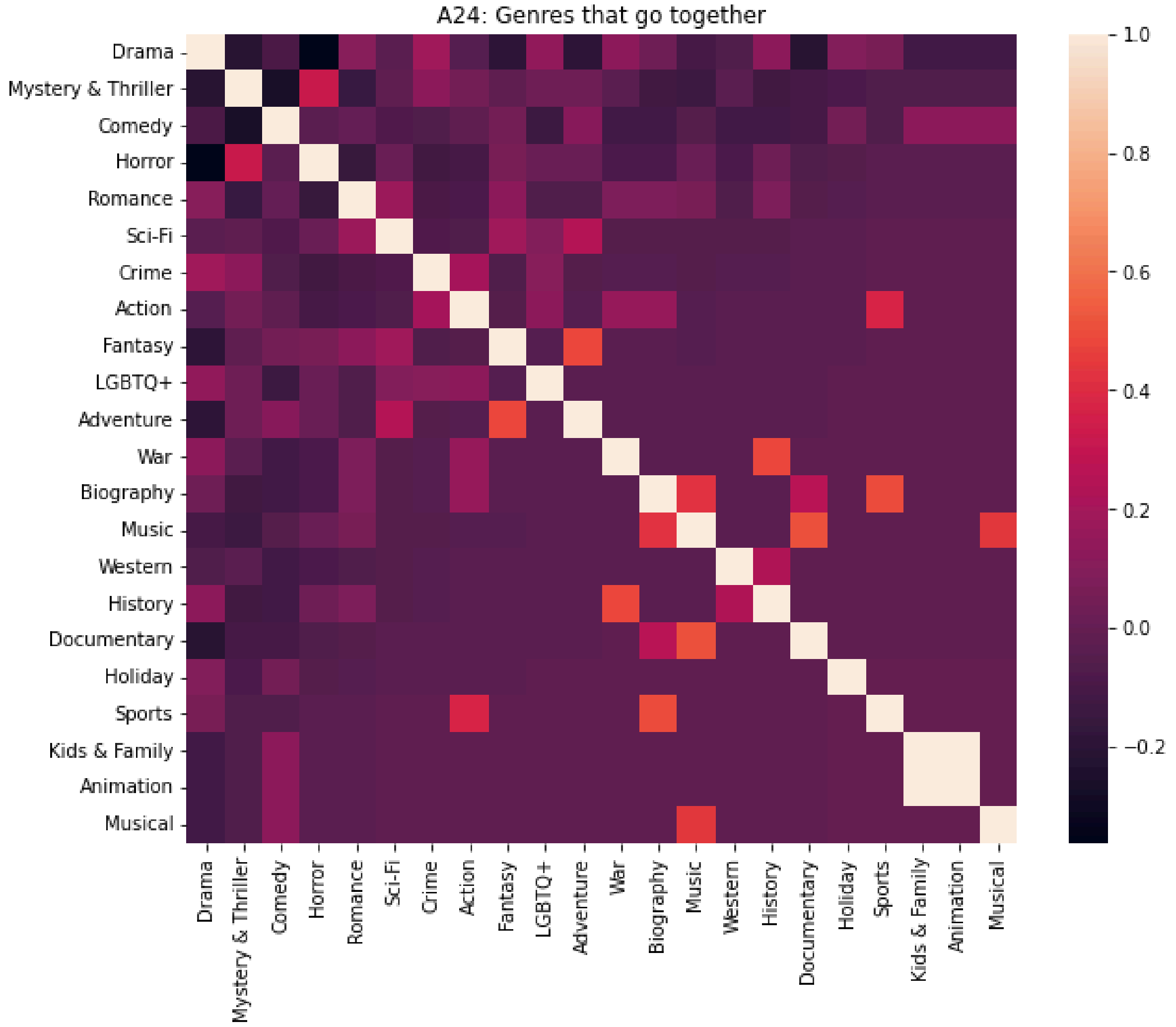
In [ ]:
```
unique_genres = genre_freq['Genre'].tolist()

for genre in unique_genres:
    movies_df[genre] = movies_df['Genre'].apply(lambda x: 1 if genre in x el

## Check results
## See total number of genres per movie by summing up the 1s horizontally
movies_df[unique_genres].sum(axis=1).value_counts().plot(kind = 'bar', xlabe
                                                         ylabel = 'Freq',
                                                         title = 'Distributi

plt.plot()
```

Out[ ]:
```
[]
```



Visualise genre correlation

In [ ]:
```
genre_corr = movies_df[unique_genres].corr()
plt.figure(figsize=(10,8))
sns.heatmap(genre_corr)
plt.title('A24: Genres that go together')
plt.show()
```



Most popular pairs

In [ ]:
```
## First, create a mask such that we only get the upper triangle for the
all_corr_df = genre_corr.unstack().reset_index()
all_corr_df.columns = ['Genre1', 'Genre2', 'Corr']
## Remove self correlations (i.e. Corr = 1)
unique_corr_df = all_corr_df[all_corr_df.Genre1 != all_corr_df.Genre2]
print(f'Shape before: {all_corr_df.shape} and after {unique_corr_df.shape}'
```

```
Shape before: (484, 3) and after (462, 3)
```

In [ ]:
```
## Sort genre 1 and 2 so that we can remove the same pair later (e.g. A and
## pair is just a helper column for us to remove duplicates -- remove this
unique_corr_df['pair'] = unique_corr_df.apply(lambda row: tuple(sorted([row
corr_pairs_df = unique_corr_df.drop_duplicates('pair')
corr_pairs_df.drop(columns = 'pair').sort_values('Corr').reset_index(drop =
```

```
/var/folders/4k/l4b4k90s7mb5fw_d4xc66bym0000gn/T/ipykernel_11399/682901453.
py:3: 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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  unique_corr_df['pair'] = unique_corr_df.apply(lambda row: tuple(sorted([r
ow['Genre1'], row['Genre2']])), axis=1)
```

Out[ ]:

	Genre1	Genre2	Corr
0	Drama	Horror	-0.362120
1	Mystery & Thriller	Comedy	-0.271528
2	Drama	Mystery & Thriller	-0.211594
3	Drama	Documentary	-0.210076
4	Drama	Adventure	-0.190682
...	...	...	...
226	War	History	0.484848
227	Fantasy	Adventure	0.485003
228	Biography	Sports	0.494413
229	Music	Documentary	0.502720
230	Kids & Family	Animation	1.000000

231 rows × 3 columns

Which genres score the highest?

In [ ]:

```
cleaned_df = movies_df.copy()

## Clean scores
cleaned_df['audience_score'] = cleaned_df['audience_score'].str.replace('%',
cleaned_df['RT_score'] = cleaned_df['RT_score'].str.replace('%', '')
## Drop extra director column
cleaned_df = cleaned_df.drop(columns='Director.1')
movies_df['audience_score_%'] = pd.to_numeric(movies_df['audience_score'].st
movies_df['RT_score_%'] = pd.to_numeric(movies_df['RT_score'].str.replace('%
```

In [ ]:

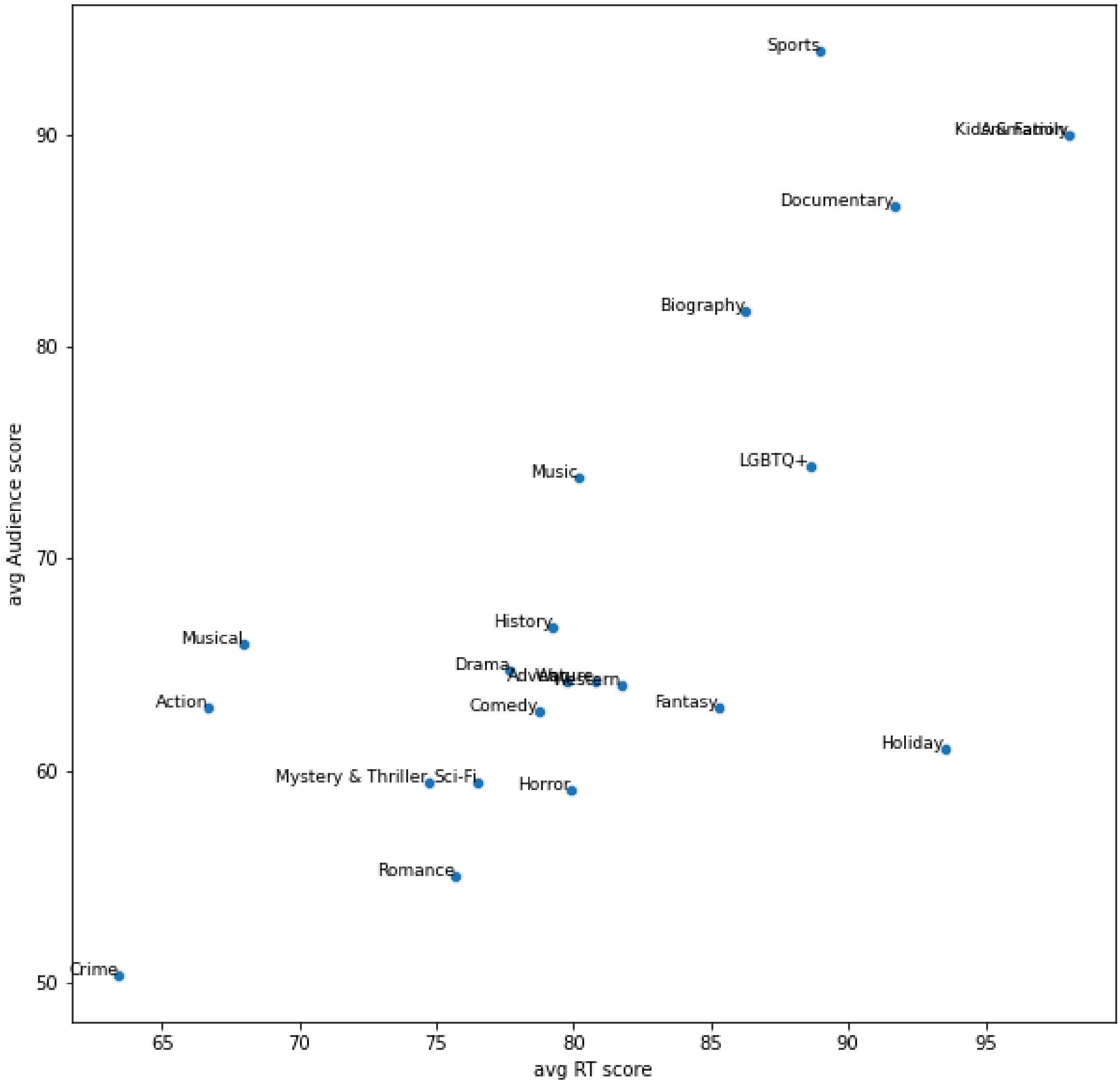
```
rt_genre_scores = []
aud_genre_scores = []
for genre in unique_genres:
    rt_genre_scores.append(round(cleaned_df[movies_df[genre] == 1]["RT_score
    aud_genre_scores.append(round(cleaned_df[movies_df[genre] == 1]["audienc
genre_scores_df = pd.DataFrame({'Genre' : unique_genres, 'avg RT score': rt_
              , 'avg Audience score' : aud_genre_scores
              }).sort_values('avg RT score', ascending = False)
genre_scores_df
```

Out[ ]:

	Genre	avg RT score	avg Audience score
20	Animation	98.00	90.00
19	Kids & Family	98.00	90.00
17	Holiday	93.50	61.00
16	Documentary	91.67	86.67
18	Sports	89.00	94.00
9	LGBTQ+	88.60	74.40
12	Biography	86.25	81.75
8	Fantasy	85.29	63.00
14	Western	81.75	64.00
10	Adventure	80.80	64.20
13	Music	80.20	73.80
3	Horror	79.88	59.08
11	War	79.75	64.25
15	History	79.25	66.75
2	Comedy	78.76	62.84
0	Drama	77.66	64.78
5	Sci-Fi	76.50	59.40
4	Romance	75.67	55.00
1	Mystery & Thriller	74.70	59.46
21	Musical	68.00	66.00
7	Action	66.71	63.00
6	Crime	63.44	50.33

In [ ]:

```
plt.figure(figsize=(10,10))
sns.scatterplot(data = genre_scores_df, x = 'avg RT score', y = 'avg Audienc
## annotate with genre
for i, row in genre_scores_df.iterrows():
    plt.text(row['avg RT score'], row['avg Audience score'], row['Genre'], 
```



Genres that do well on both accounts include: Sports, Kids & Family, Documentary

Genres that fare badly : Crime

There is generally a correlation between critics and audiences on the scores they give to genres (on average, for each genre). However one that they disagree on is the genre **Holiday**, which achieves high critics but low audience scores.

Some examples are below:

In [ ]:
```
movies_df[movies_df['Holiday'] == 1][['Title', 'Release Year', 'RT_score',
```

Out[ ]:

	Title	Release Year	RT_score	audience_score
16	Krisha	2015	95%	77%
29	The Humans	2021	92%	45%

1. Actors/ cast

In [ ]:
```
all_actors_combined_list = ','.join(movies_df['Cast'])
all_actors_combined_list
```

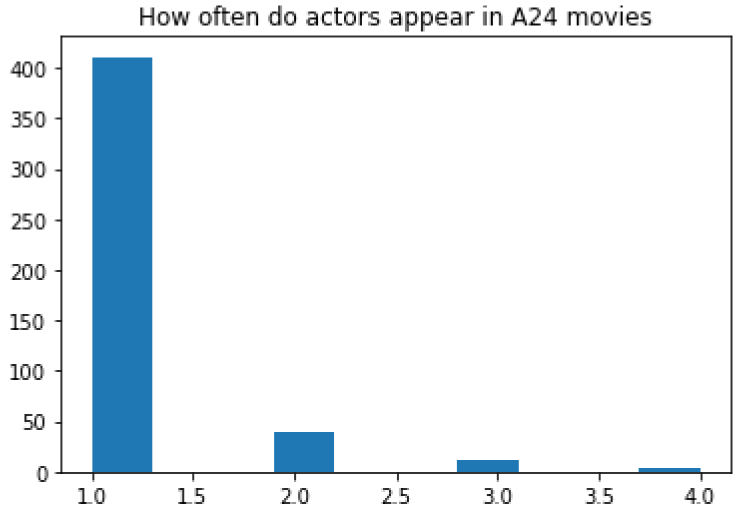
```
actors_counter = Counter(all_actors_combined_list.split(','))
actors_counter.most_common(5)
```

Out[ ]:
```
[('Tilda Swinton', 4),
 ('Robert Pattinson', 4),
 ('Riley Keough', 4),
 ('Elle Fanning', 4),
 ('Lucas Hedges', 3)]
```

In [ ]:
```
print(f'Total number of unique actors: {len(actors_counter)}')
```

```
Total number of unique actors: 466
```

In [ ]:
```
plt.hist(actors_counter.values())
plt.title('How often do actors appear in A24 movies')
plt.show()
```



It is very right skewed - only a handful of actors have appeared more than once in the movies

In [ ]:
```
recurrent_actors = [(actors, count) for actors,count in actors_counter.items
sorted(recurrent_actors, key = lambda item: -item[1])
```

Out[ ]:
```
[('Tilda Swinton', 4),
 ('Robert Pattinson', 4),
 ('Riley Keough', 4),
 ('Elle Fanning', 4),
 ('Lucas Hedges', 3),
 ('Daniel Zolghadri', 3),
 ('Isabella Rossellini', 3),
 ('John Magaro', 3),
 ('Mia Goth', 3),
 ('Brie Larson', 3),
 ('Oscar Isaac', 3),
 ('Alicia Vikander', 3),
 ('Joe Cole', 3),
 ('James Franco', 3),
 ('Colin Farrell', 3)]
```

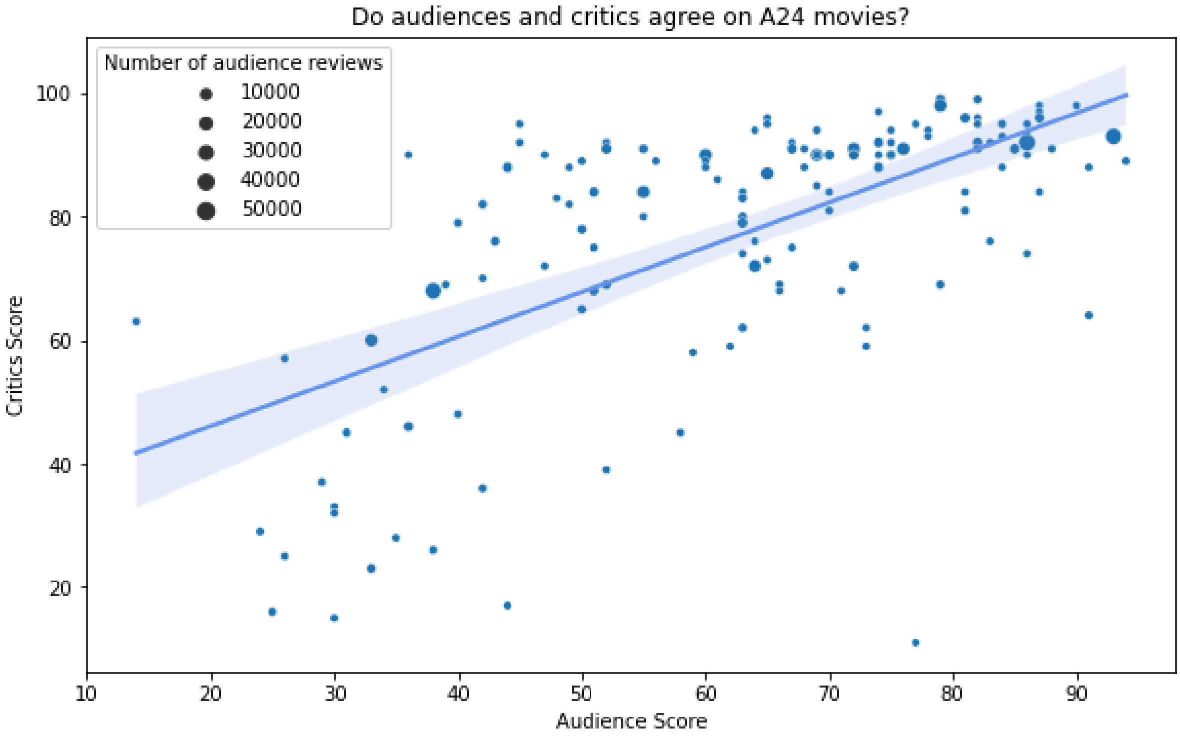
Critics vs Audience

Let us clean the number of audience reviews such that we glean some insights from it - as a proxy of popularity

In [ ]:
```
movies_df['number_audience_reviews_censored'] = movies_df['number_audience_r
movies_df['number_audience_reviews_censored'][movies_df['number_audience_rev

## Replace text 'Fewer than 50' with just 50
movies_df.loc[movies_df['number_audience_reviews_censored'].str.contains('Fe

## Now convert everything to numeric
movies_df['number_audience_reviews_censored'] = pd.to_numeric(movies_df['nun
```

In [ ]:
```
plt.figure(figsize=(10,6))

sns.scatterplot(data = movies_df, x = 'audience_score_%',
                y = 'RT_score_%', size = 'number_audience_reviews_censored')
sns.regplot(data = movies_df, x = 'audience_score_%',
                y = 'RT_score_%', scatter = False, color = 'cornflowerblue')
plt.title('Do audiences and critics agree on A24 movies?')
plt.xlabel('Audience Score')
plt.ylabel('Critics Score')
plt.legend(title = 'Number of audience reviews')
plt.plot()
```

Out[ ]: []



There is high variance amongst critics for low audience scoring movies. However, as the audience scores increase, the critics tend to agree more that the movies are better, showing a decreasing variance.

There is only one anomaly - a high (~78) rated movie by audiences, but below 10 by critics.

As for audience popularity, we do see that the more commonly reviewed movies are the ones in the top right hand corner, where both audience and critics view more favourably.

In [ ]:
```
anomaly = (movies_df['RT_score_%'] <20) & (movies_df['audience_score_%'] > 7
movies_df[anomaly]
```

Out[ ]:

	Title	Release Year	RT_score	URL	Directo
135	The Vanishing of Sidney Hall	2017	11%	https://www.rottentomatoes.com/m/the_vanishing...	Shaw Christense

Run times : are they getting longer? Are they popular with fans and critics?

In [ ]:
```
hours = pd.to_numeric(movies_df['Runtime'].str[0])
mins = pd.to_numeric(movies_df['Runtime'].str[3:-1])
movies_df['Runtime_hours'] = hours + (mins/60)
movies_df['Runtime_mins'] = hours*60 + mins
movies_df['Runtime_mins'].head()
```

Out[ ]:
```
0     94
1     93
2    111
3    115
4     89
Name: Runtime_mins, dtype: int64
```

Longest and shortest running movies

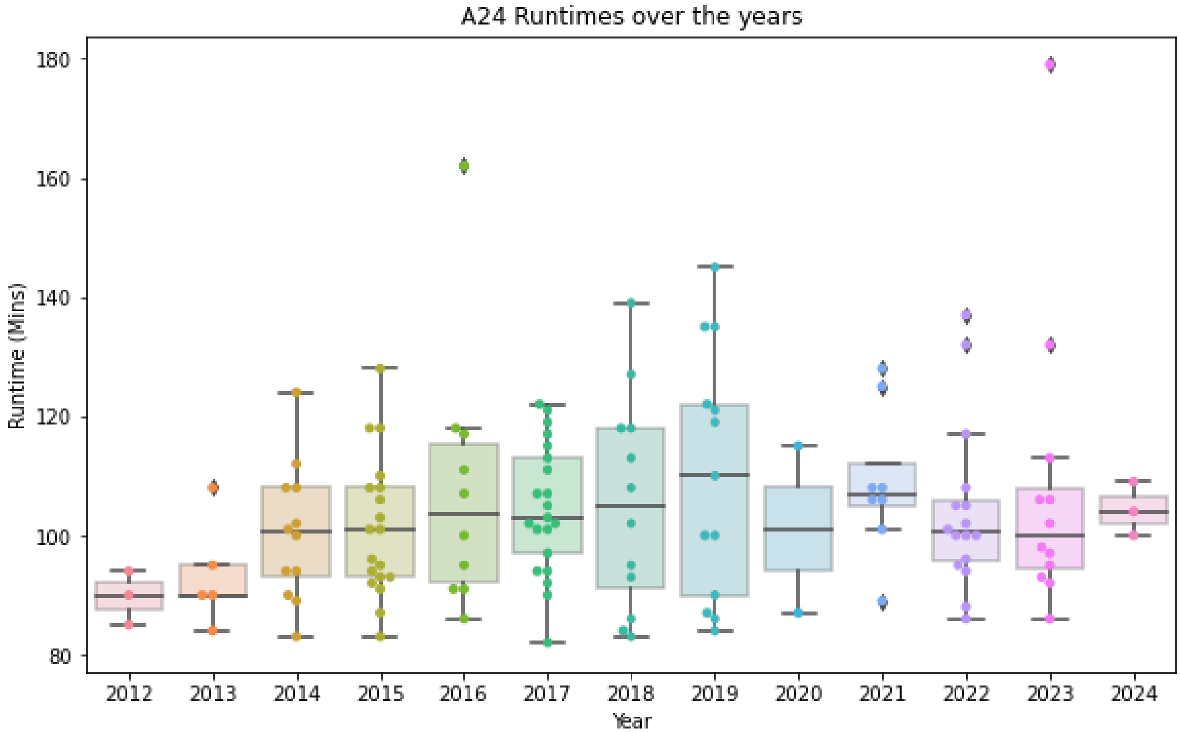
In [ ]:
```
longest = movies_df.sort_values('Runtime_mins').head(3)
shortest = movies_df.sort_values('Runtime_mins').tail(3)

pd.concat([longest, shortest], axis=0)[['Title', 'Director','Runtime_mins',
```

Out[ ]:

	Title	Director	Runtime_mins	Release Year	RT_score	audience_score
14	Menashe	Joshua Z Weinstein	82	2017	95%	65%
51	Obvious Child	Gillian Robespierre	83	2014	90%	72%
117	Slice	Austin Vesely	83	2018	52%	34%
75	Midsommar	Ari Aster	145	2019	83%	63%
82	American Honey	Andrea Arnold	162	2016	80%	63%
105	Beau Is Afraid	Ari Aster	179	2023	68%	71%

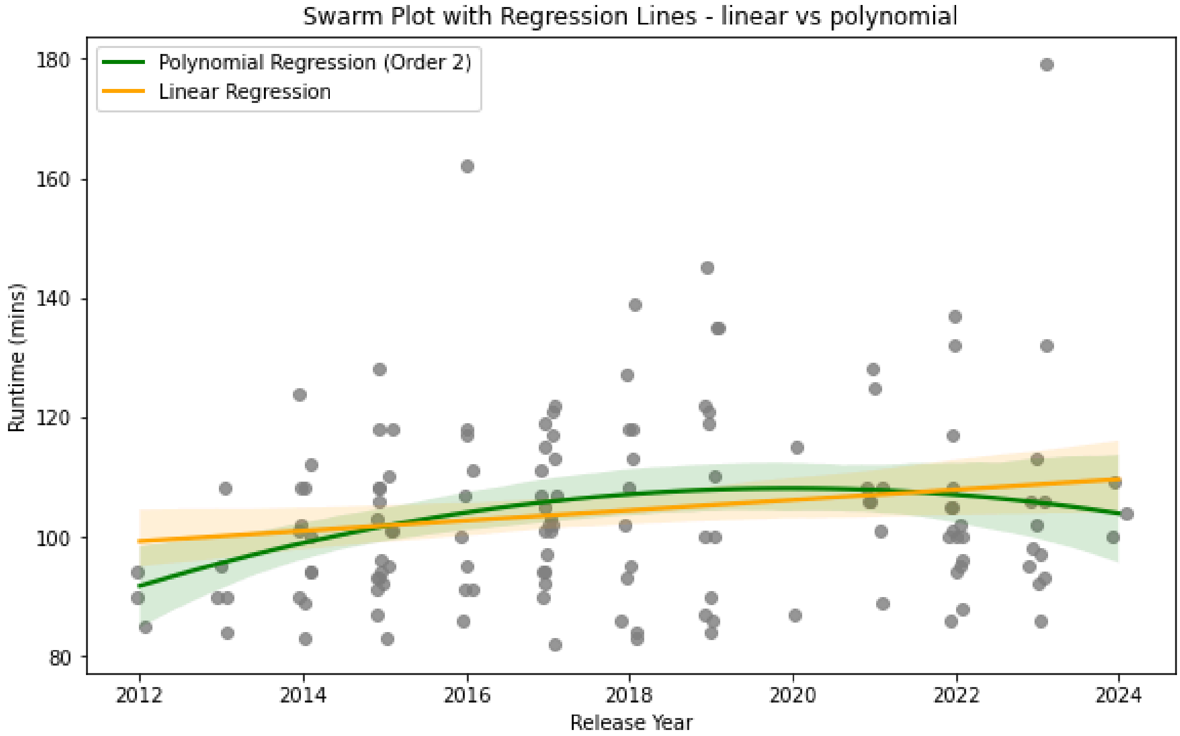
In [ ]:
```
plt.figure(figsize=(10,6))
sns.regplot(x='Release Year', y='Runtime_mins', data=movies_df, scatter=Fals
sns.swarmplot(x = movies_df['Release Year'],y = movies_df['Runtime_mins'])
sns.boxplot(x = movies_df['Release Year'],y = movies_df['Runtime_mins'], bo
# sns.regplot(x = movies_df['Release Year'],y = movies_df['Runtime_mins'])
plt.title('A24 Runtimes over the years')
plt.xlabel('Year')
```

```
plt.ylabel('Runtime (Mins)')
plt.show()
```



Can we see a general trend?

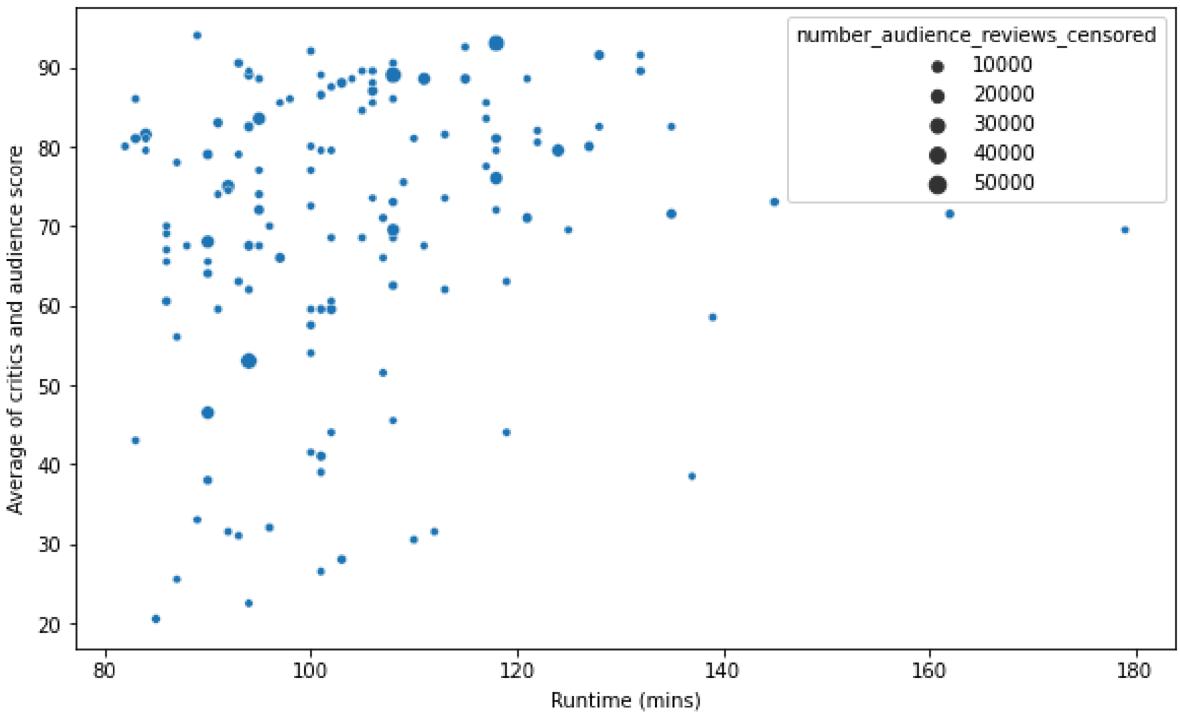
In [ ]:
```
from matplotlib.lines import Line2D
plt.figure(figsize=(10, 6))

# sns.scatterplot(x='Release Year', y='Runtime_mins', data=movies_df,y_jitte

# Overlay the regression line
sns.regplot(x='Release Year', y='Runtime_mins', data=movies_df, scatter=True
            scatter_kws={"color": "grey"}, line_kws={"color": "green"})
sns.regplot(x='Release Year', y='Runtime_mins', data=movies_df, scatter=Fals

plt.xlabel('Release Year')
plt.ylabel('Runtime (mins)')
legend_elements = [Line2D([0], [0], color='green', lw=2, label='Polynomial R
                   Line2D([0], [0], color='orange', lw=2, label='Linear Regr

# Add the legend to the plot
plt.legend(handles=legend_elements)
plt.title('Swarm Plot with Regression Lines - linear vs polynomial')
plt.show()
```



The most general trend is that the run times are increasing, but when we increase the order of the fitted curve (i.e. quadratic line), we see that there is a dip after year 2020. As expected from higher order polynomial fits, the standard errors are wider at the ends too.

In [ ]:
```
plt.figure(figsize=(10,6))
movies_df['critics_audience_avg_score_%'] = (movies_df['audience_score_%'] -
sns.scatterplot(x = movies_df['Runtime_mins'], y = movies_df['critics_audier
                 size = movies_df['number_audience_reviews_censored'])
plt.ylabel('Average of critics and audience score')
plt.xlabel('Runtime (mins)')
```

Out[ ]:
```
Text(0.5, 0, 'Runtime (mins)')
```



There is no clear evidence to show that longer runtimes are associated with increased scores or higher popularity.

Box office: over the years

Clean Box Office column by standardising all to in terms of millions USD

In [ ]:
```
def convert_to_millions(value):
    if isinstance(value, str):
        value = value.replace('$', '')
        if 'M' in value:
            return float(value.replace('M', ''))
        elif 'K' in value:
            return float(value.replace('K', '')) / 1000
    return value

movies_df['box_office_millions'] = movies_df['Box Office (Gross USA)'].apply

## check
movies_df['box_office_millions'].describe()
```

Out[ ]:
```
count    107.000000
mean       8.005124
std       14.029848
min        0.005600
25%        0.205550
50%        2.000000
75%        8.150000
max       76.700000
Name: box_office_millions, dtype: float64
```

Clean date as well

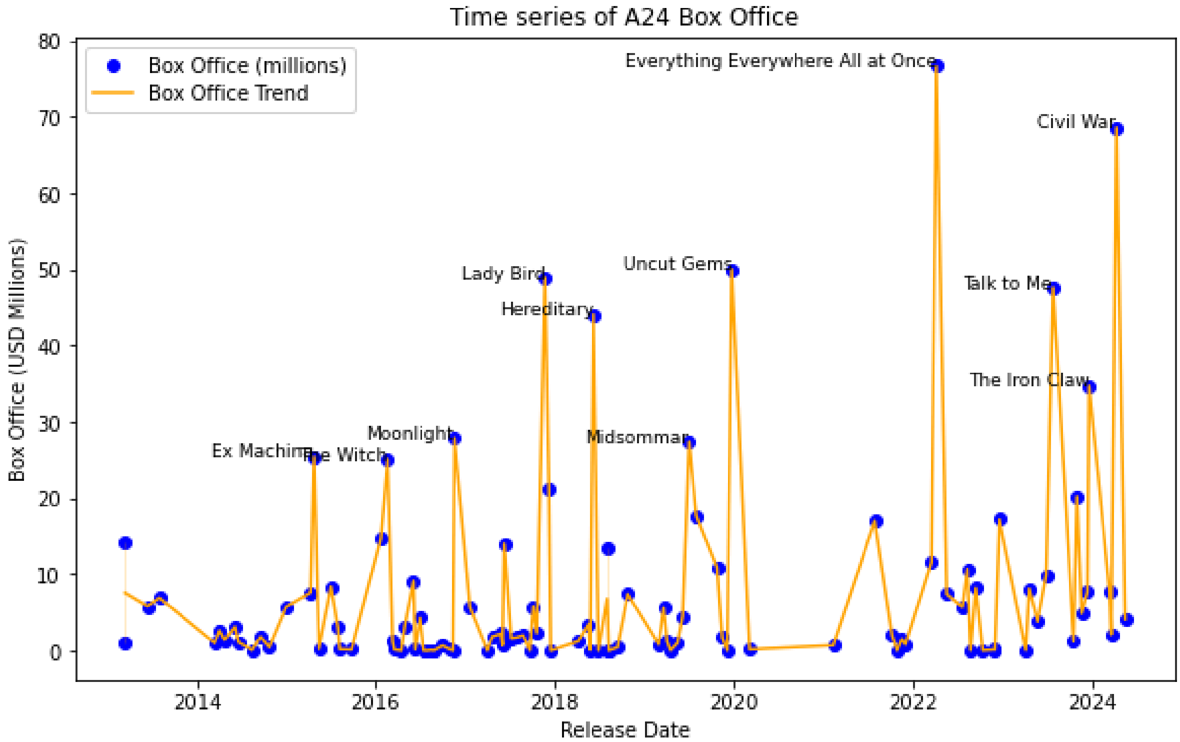
In [ ]:
```
movies_df['Release Date'] = movies_df['Release Date (Theaters)'].apply(lambd
movies_df['Release Date'] = pd.to_datetime(movies_df['Release Date'], format
movies_df['Release Date']
```

Out[ ]:
```
0      2017-11-22
1      2018-08-03
2      2016-11-18
3      2021-02-12
4      2022-07-22
          ...
131    2015-08-07
132    2016-08-26
133    2013-02-08
134    2014-10-24
135    2018-03-02
Name: Release Date, Length: 136, dtype: datetime64[ns]
```

Visualising blockbusters

In [ ]:
```
plt.figure(figsize = (10,6))
timeseries = movies_df.set_index('Release Date')
plt.scatter(movies_df['Release Date'], movies_df['box_office_millions'], co
for i, row in movies_df.iterrows():
    if row['box_office_millions'] > 25:
        plt.text(row['Release Date'], row['box_office_millions'], row['Title
```

```
sns.lineplot(data=movies_df, x='Release Date', y='box_office_millions', colo
plt.title('Time series of A24 Box Office')
plt.ylabel('Box Office (USD Millions)')
plt.show()
```

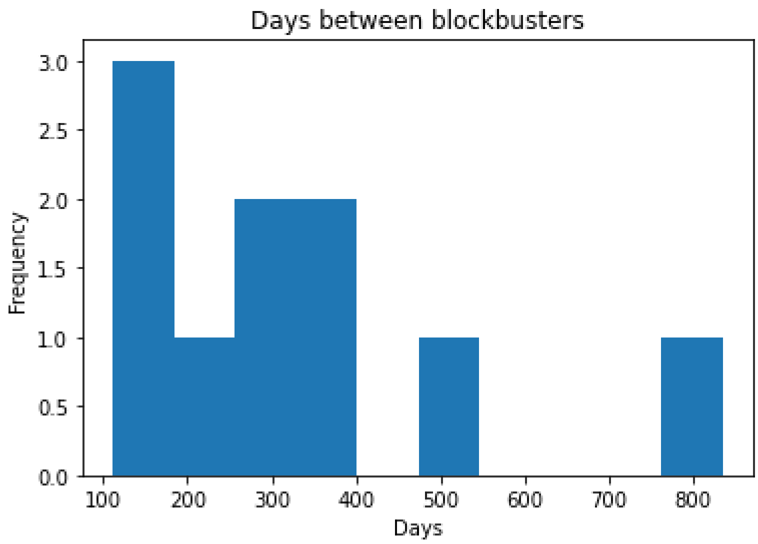


If we use a more modest definition of a blockbuster for A24 movies being more than 25 million, we see that we get one or more a year, with the exception of the Covid-19 period between 2020 and 2022.

We can further the analysis to see 'time to blockbuster'

In [ ]:
```
movies_df['blockbuster'] = movies_df['box_office_millions'].apply(lambda x:
blockbusters = movies_df[movies_df['blockbuster']].sort_values('Release Date
time_to_blockbuster =blockbusters['Release Date'] - blockbusters['Release Da
time_to_blockbuster = time_to_blockbuster.dt.days.dropna()
time_to_blockbuster.plot(kind = 'hist')
plt.title('Days between blockbusters')
plt.xlabel('Days')
print(f"Average number of days between blockbusters: {time_to_blockbuster.me
```

```
Average number of days between blockbusters: 327.6
```



We see that the average wait is about less than a year, a value that is upwardly biased due to the Covid pause.

Missing data analysis

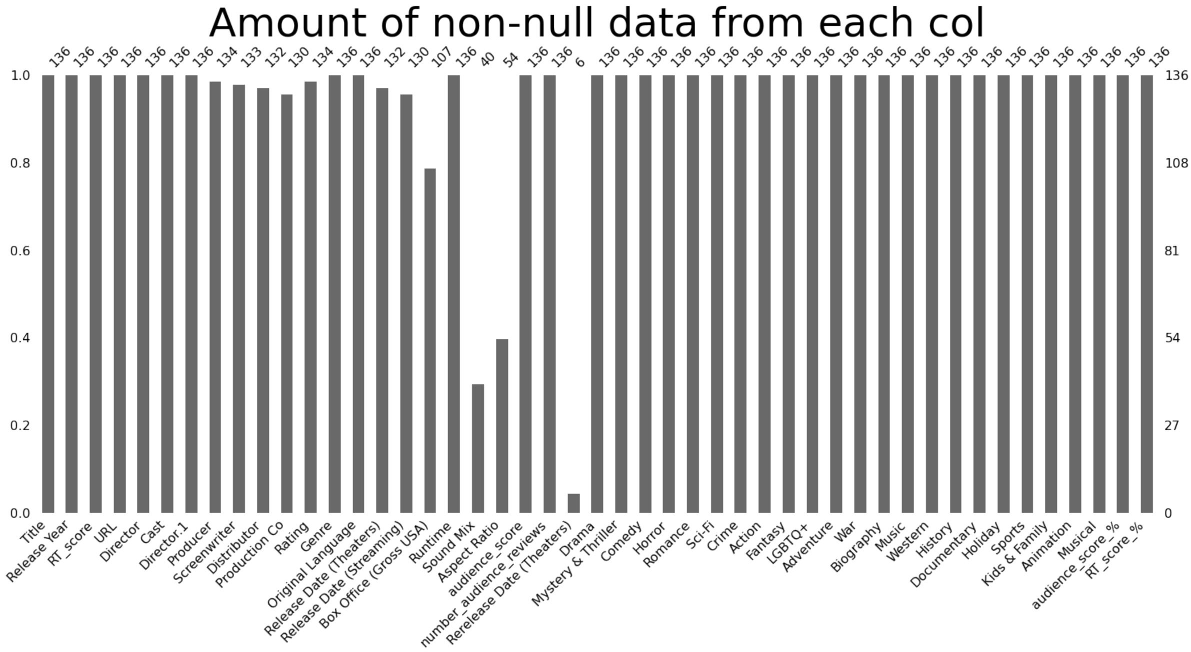
In [ ]:
```
original_movies_df = pd.read_csv('full_scraped_a24_df.csv', index_col=0)
## Percentage missing
original_movies_df.isna().sum(axis = 0).sort_values(ascending=
                                                    False)/movies_df.shape[0]
```

Out[ ]:
```
Rerelease Date (Theaters)    0.955882
Sound Mix                    0.705882
Aspect Ratio                 0.602941
Box Office (Gross USA)       0.213235
Release Date (Streaming)     0.044118
Production Co                0.044118
Release Date (Theaters)      0.029412
Distributor                  0.029412
Screenwriter                 0.022059
Producer                     0.014706
Rating                       0.014706
Biography                    0.000000
War                          0.000000
Adventure                    0.000000
Music                        0.000000
Western                      0.000000
LGBTQ+                       0.000000
Fantasy                      0.000000
Title                        0.000000
History                      0.000000
Crime                        0.000000
Documentary                  0.000000
Holiday                      0.000000
Sports                       0.000000
Kids & Family                0.000000
Animation                    0.000000
Musical                      0.000000
audience_score_%             0.000000
Action                       0.000000
Drama                        0.000000
Sci-Fi                       0.000000
Original Language            0.000000
RT_score                     0.000000
URL                          0.000000
Director                     0.000000
Cast                         0.000000
Director.1                   0.000000
Genre                        0.000000
Runtime                      0.000000
Romance                      0.000000
audience_score               0.000000
number_audience_reviews      0.000000
Release Year                 0.000000
Mystery & Thriller           0.000000
Comedy                       0.000000
Horror                       0.000000
RT_score_%                   0.000000
dtype: float64
```

Visualise the missing data

In [ ]:
```
import missingno as msno
```

In [ ]:
```
## visualising nullity bu column
msno.bar(original_movies_df)
plt.title('Amount of non-null data from each col', size = 50)
plt.show()
```

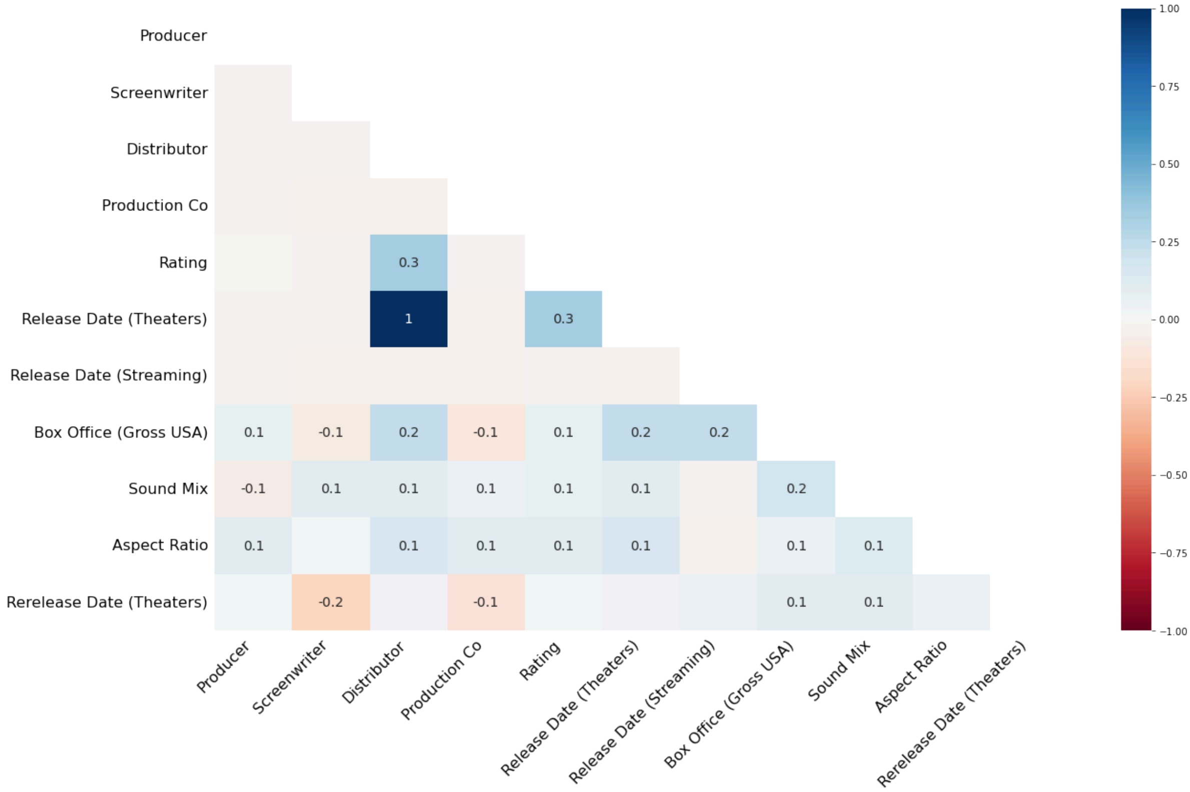


Do the missing data come from the same places - are they correlated?

This could reveal insights into the way we collected the data, i.e. via simple html webscrapping. It is possible that the pages where most data weren't captured had different html structure for the data that we were trying to get, hence causing correlated nullity.

In [ ]:
```
msno.heatmap(original_movies_df)
```

Out[ ]:
```
<AxesSubplot:>
```



The missingness in the missing cols are quite highly correlated. For example, Genre and Duration are perfectly correlated, suggesting that they go missing together.

```
In [ ]:
```