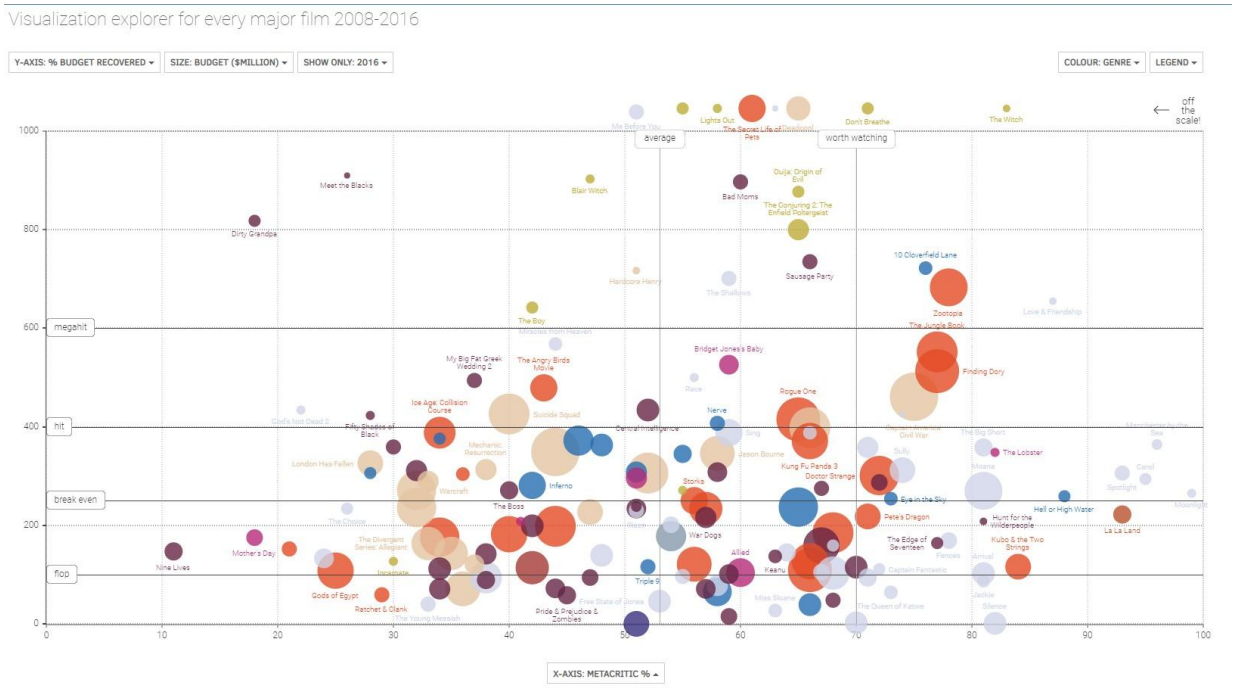
Group 5: Through the Lens

Intro to the Topic

Team 5 presents Through the Lens - a look at films and what drives their success. Success can be defined in two ways for this project: critical acclaim and profit. This project looks at patterns in film, seeking to understand which factors contribute to a film's success.

Users will be able to predict a film's critical success based on certain criteria, interact with a dashboard to explore the financial success of different films, and see the way films are rated on Rotten Tomatoes.

Inspiration



As our team discussed potential project ideas, movies just clicked for all of us. When we began our research, the most inspiring dashboards had the following in common: distinct color stories and tons of character, static and interactive visuals working in tandem, and fantastic dots that moved across the screen as you adjusted variables (see the image above.)

We sought out film data and settled into exploring Rotten Tomatoes scores and Box Office Mojo financial data.

Expectations

Going into the project, we did not have any major expectations, but rather a few things we were curious about. Ratings and genre had us pondering what would be the most successful, and questioning the potential for ratings like R or NC17 to draw in large enough audiences or repeat views to compete with the films with broader audiences.

Data Sources

Box Office Mojo

Rotten Tomatoes

Tools Required

- Python and Jupyter Notebook.
- Pandas, Numpy, Scikit-learn.
- Linear Regression, Random forest, LGB, and GB Regressor for model prediction.
- SQLite for Database
- Tableau for Dashboards
- HTML,CSS,Java Script for web page

EDA

Exploratory data analysis is performed on the Box Office Mojo budget update dataset. We have found that there are 3243 entries and 26 columns. We have columns like movie title, year, mpaa, release date, runtime, distributor, writer, producer, composer, main actors, genre, budget, domestic, international, and worldwide.

```
1 #Information about the data
2 mojo_upd_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3243 entries, 0 to 3242
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   movie_id         3243 non-null   object
 1   title            3243 non-null   object
 2   year             3243 non-null   int64
 3   trivia           3243 non-null   object
 4   mpaa             3082 non-null   object
 5   release_date     3242 non-null   object
 6   run_time         3243 non-null   object
 7   distributor      3228 non-null   object
 8   director         3243 non-null   object
 9   writer           3234 non-null   object
 10  producer         3230 non-null   object
 11  composer         3138 non-null   object
 12  cinematographer  3129 non-null   object
 13  main_actor_1     3243 non-null   object
 14  main_actor_2     3243 non-null   object
 15  main_actor_3     3243 non-null   object
 16  main_actor_4     3240 non-null   object
 17  budget           3243 non-null   float64
 18  domestic         3224 non-null   float64
 19  international    2833 non-null   float64
 20  worldwide        3236 non-null   float64
 21  genre_1          3243 non-null   object
 22  genre_2          2962 non-null   object
 23  genre_3          2221 non-null   object
 24  genre_4          1123 non-null   object
 25  html             3243 non-null   object
```

The initial data set with incomplete information or null values are fixed as follows:

1. Deleting the rows with null values.
2. Dropping the unnecessary columns.

This is the cleaned dataset we have used for analyzing the data:

```
1 mojo_upd_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3068 entries, 0 to 3067
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   movie_id       3068 non-null   object
 1   title          3068 non-null   object
 2   year           3068 non-null   int64
 3   mpaa           3068 non-null   object
 4   release_date   3068 non-null   object
 5   run_time       3068 non-null   object
 6   director       3068 non-null   object
 7   main_actor_1   3068 non-null   object
 8   main_actor_2   3068 non-null   object
 9   main_actor_3   3068 non-null   object
 10  main_actor_4   3068 non-null   object
 11  budget         3068 non-null   float64
 12  domestic       3068 non-null   float64
 13  genre_1        3068 non-null   object
```

The runtime is converted into minutes. When we see the mean runtime of all the genres, we can see that Biography has a long runtime by comparison.

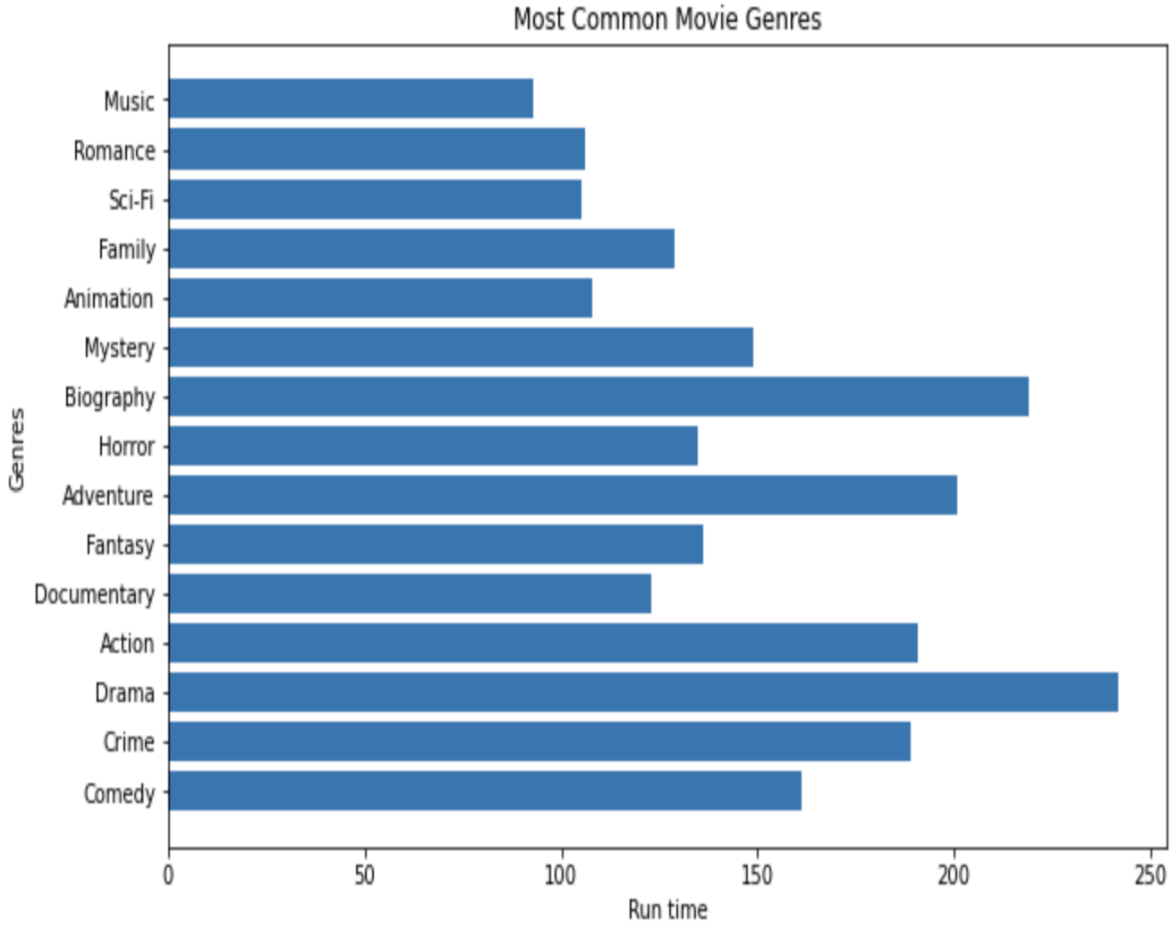
```
1 mojo_genre_runtime = mojo_upd_df.groupby("genre_1").runtime_conv.mean()
2 mojo_genre_runtime
```

```
genre_1
Action         112.985393
Adventure      103.168459
Animation       88.545455
Biography      123.445714
Comedy         102.736330
Crime          113.814433
Documentary     97.857143
Drama          115.224280
Family         118.000000
Fantasy        103.500000
Horror          97.880503
Music           93.000000
Mystery        118.222222
Romance        106.000000
Sci-Fi          96.333333
Name: runtime_conv, dtype: float64
```

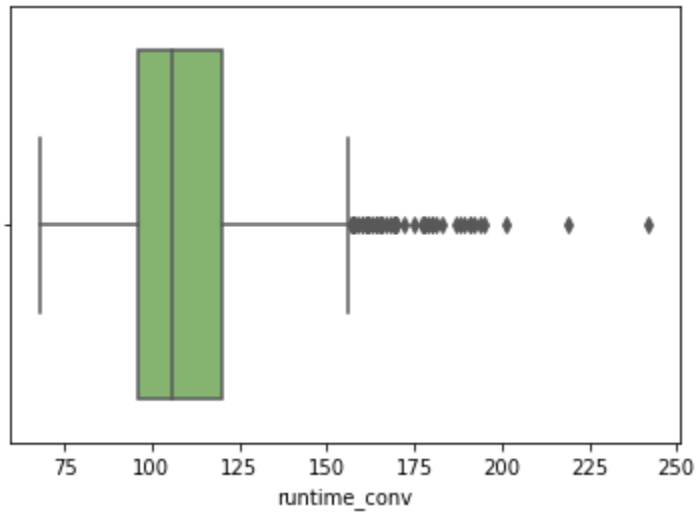
The runtime for each genre is plotted by using barchart. We could see that Drama's have a longer runtime.

```
x = mojo_upd_df['genre_1']
y = mojo_upd_df['runtime_conv']
import matplotlib.pyplot as plt

#create a figure with large width to display all genres clearly
plt.figure(figsize = (10,6))
#make a bar plot on the figure
plt.barh(x, y)
#label the axes, title, and ticks
plt.xlabel('Run time')
plt.ylabel('Genres')
plt.title('Most Common Movie Genres')
```

```
Text(0.5, 1.0, 'Most Common Movie Genres')
```



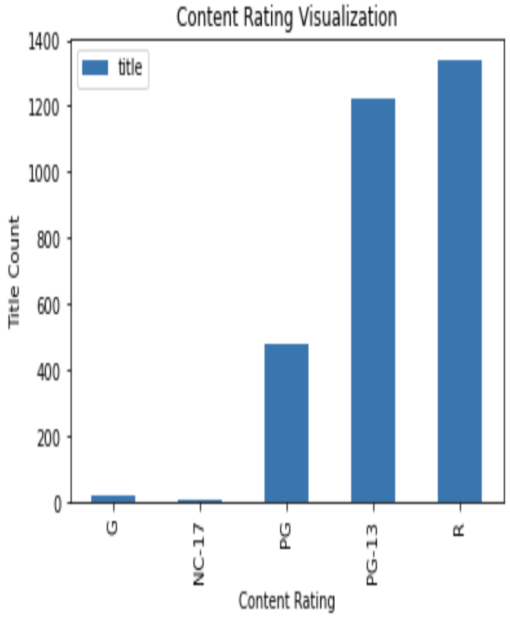
The box plot has been plotted to see the distribution of runtime data. From the box plot we can see that most of the movies have runtime between 100 to 125 minutes.



The Count of movies for each content rating is calculated and plotted below. We can see from the graph that movies with rating R are more.

```
# using the visualization to display content rating vs movie title
mojo_upd_df[['mpaa','title']].groupby('mpaa').count().plot(kind='bar', title='Content Rating Visualization')
plt.xlabel('Content Rating')
plt.ylabel('Title Count')
```

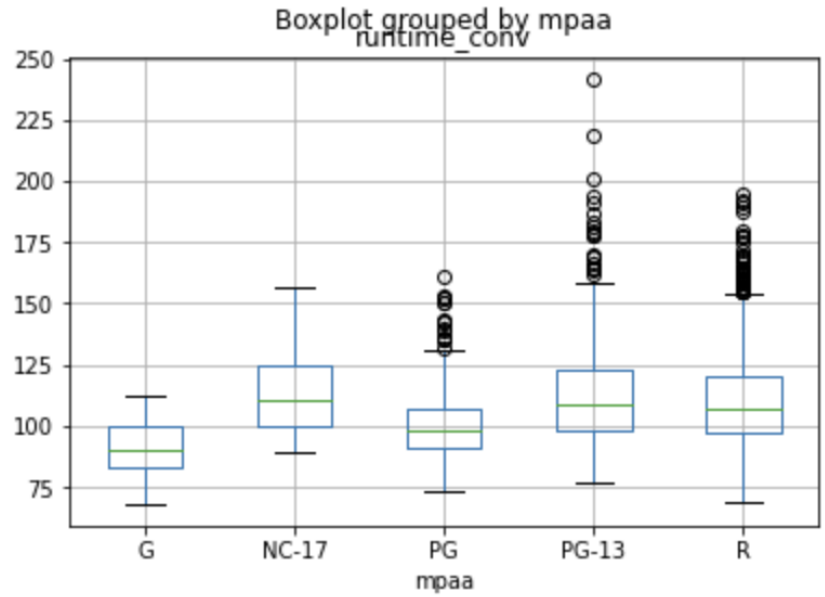
```
Text(0, 0.5, 'Title Count')
```



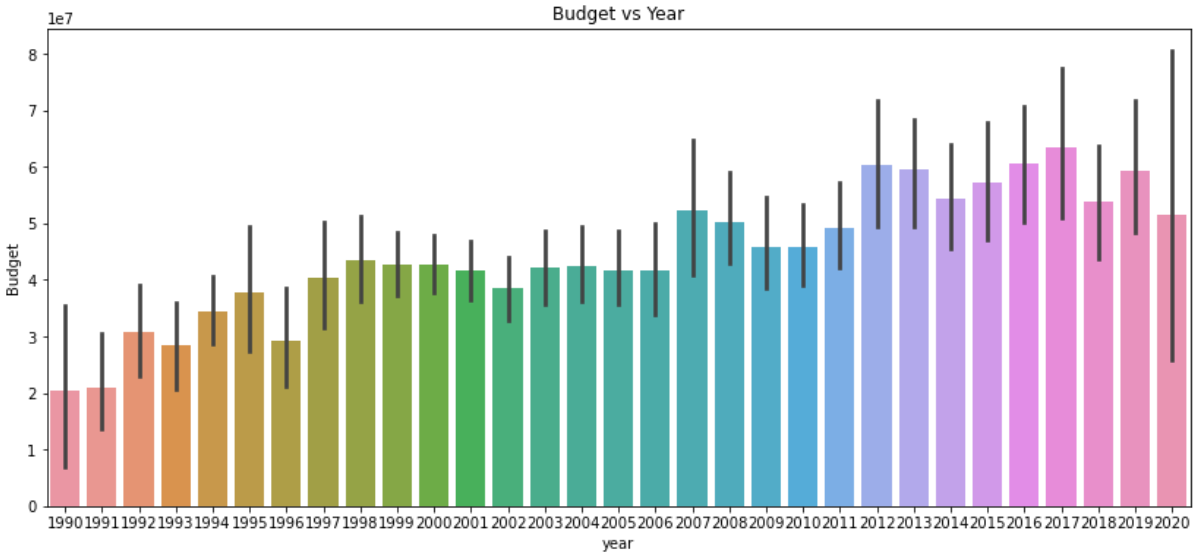
The box plot between content rating and runtime is plotted. We can see that NC-17 and PG-13 have runtime between 100 to 125 minutes.

```
# visualize the relationship between content rating and runtime
mojo_upd_df.boxplot(column ='runtime_conv', by='mpaa')
```

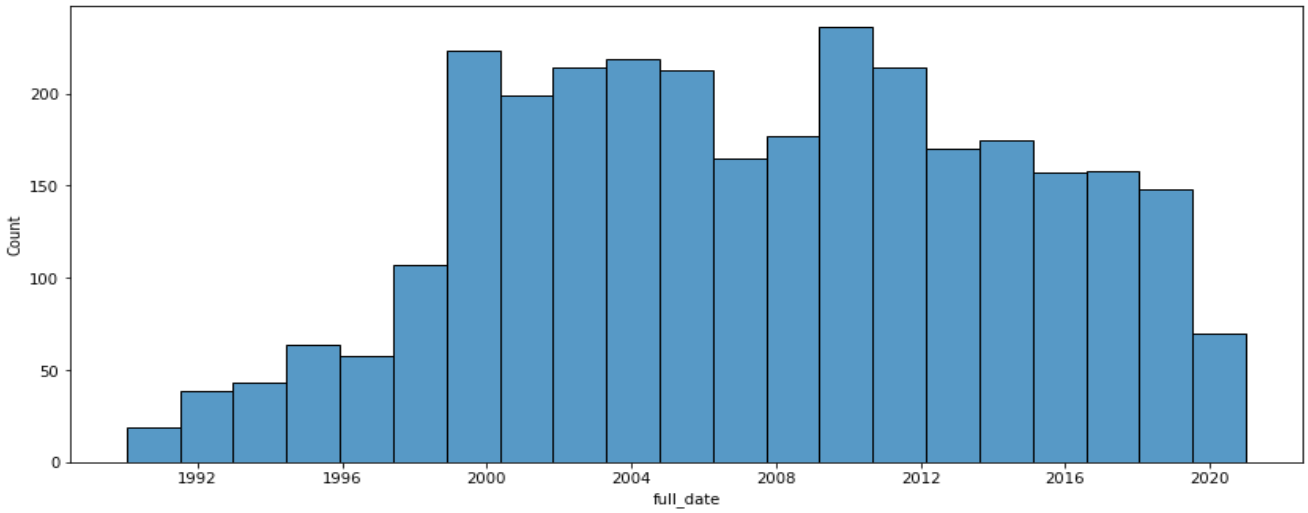
```
<AxesSubplot:title={'center':'runtime_conv'}, xlabel='mpaa'>
```



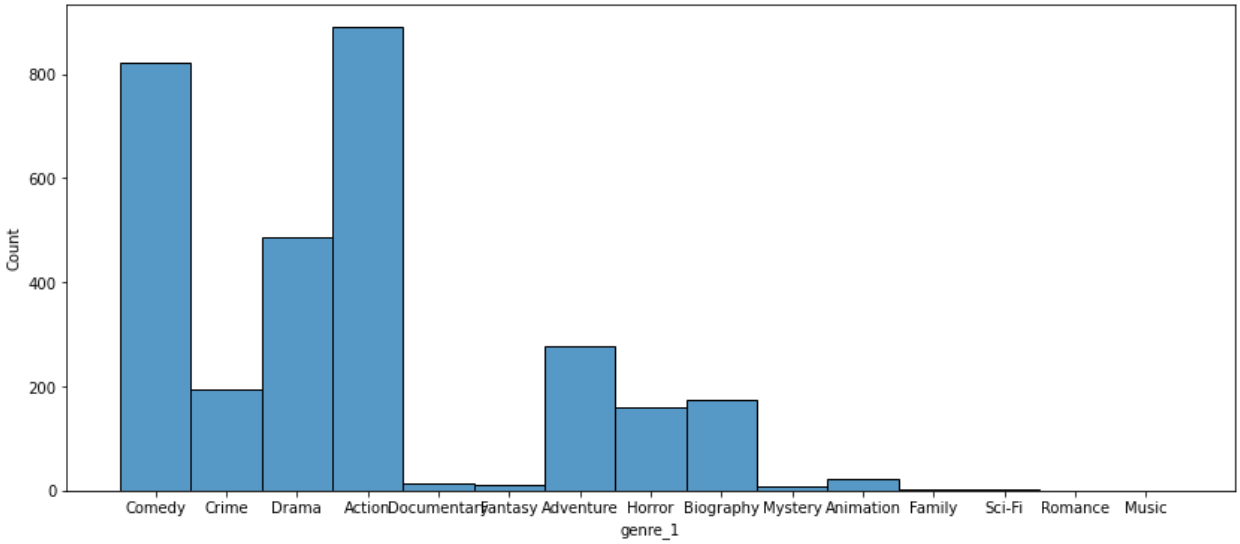
Using Sea born,the release year versus budget has been plotted. The budget for making movies was more in 2017.



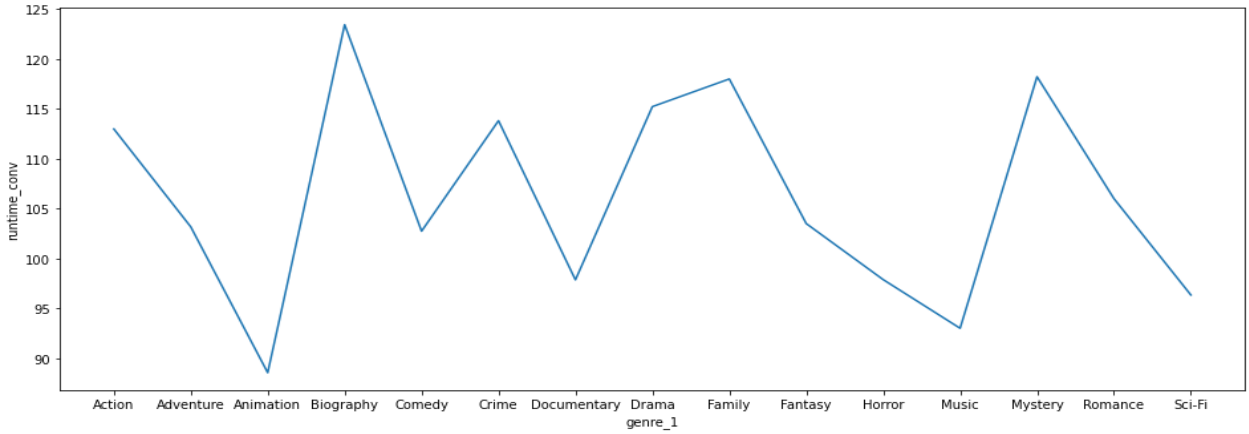
The histogram has been plotted to see how many movies have been released in each year.
We can see that more movies have been released in the year 2010.



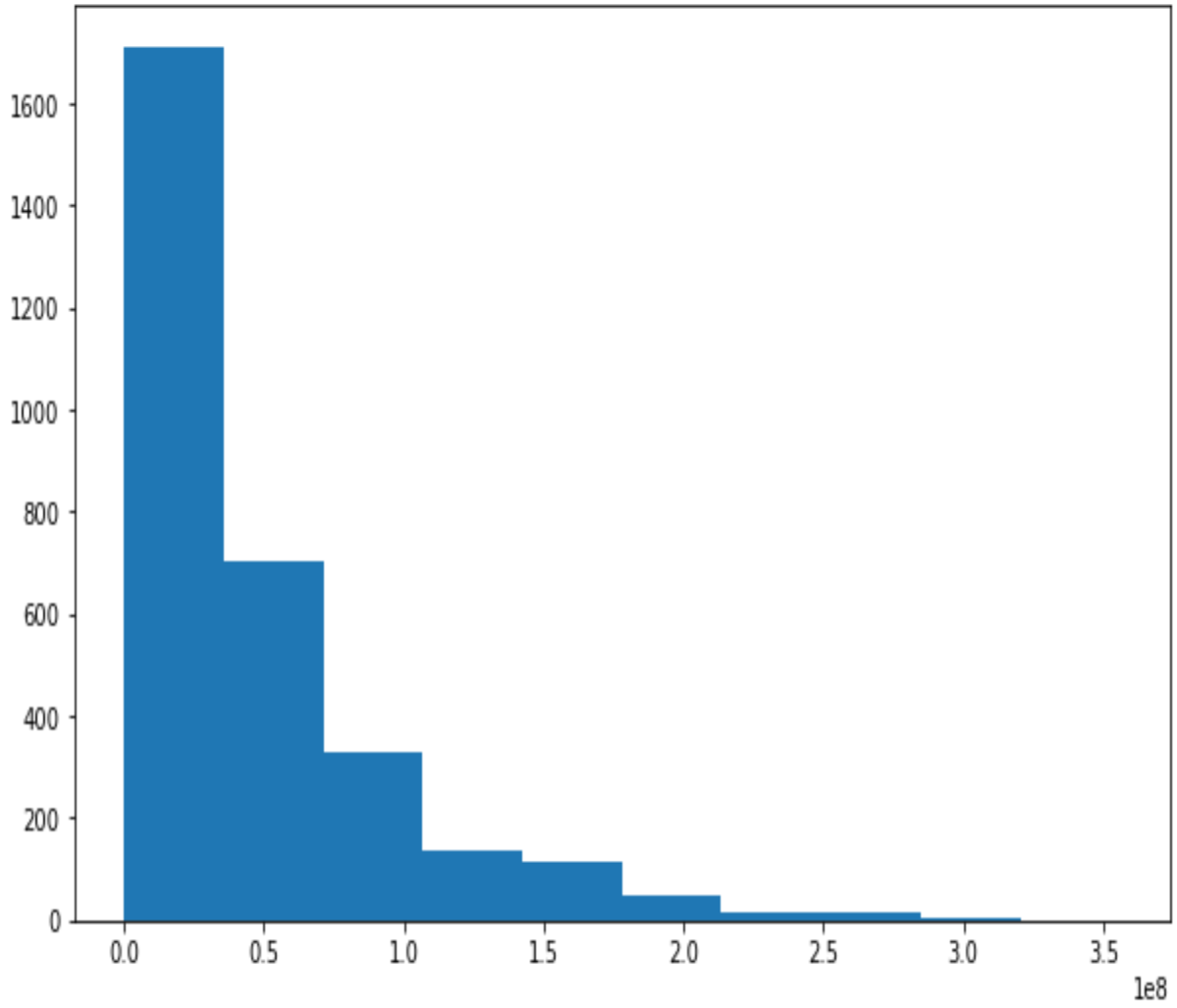
The histogram is plotted to see the count of movies of various genres.
We can see that the count of the Action genre values are more compared to other genres.



The runtime for various genres is plotted using a line graph.



We could see how the budgets of various movies are distributed using the graph below.

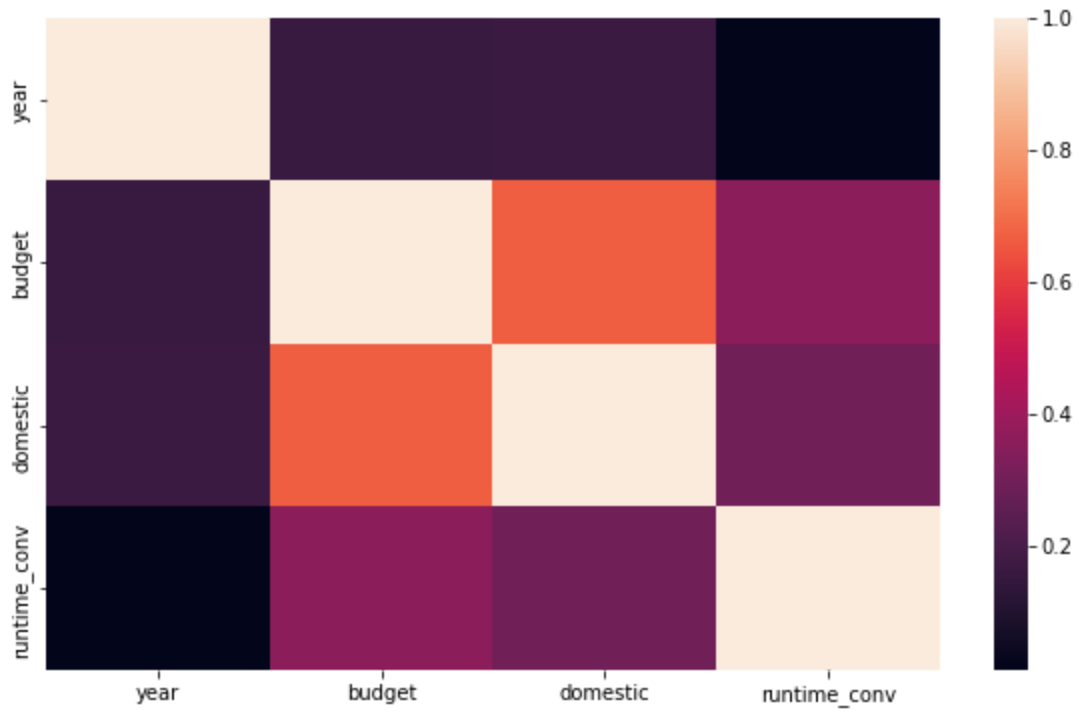


When we describe the dataframe, we can see the following details: the maximum movies are released in the year 2020 and the maximum runtime of movies is 242 minutes.

```
1 mojo_upd_df.describe()
```

	year	budget	domestic	runtime_conv
count	3068.000000	3.068000e+03	3.068000e+03	3068.000000
mean	2006.904824	4.727161e+07	6.192603e+07	109.282595
std	7.074530	4.740497e+07	8.047241e+07	18.465011
min	1990.000000	1.100000e+03	9.040000e+02	68.000000
25%	2001.000000	1.500000e+07	1.386308e+07	96.000000
50%	2007.000000	3.000000e+07	3.590826e+07	106.000000
75%	2013.000000	6.500000e+07	7.714734e+07	120.000000
max	2020.000000	3.560000e+08	9.366622e+08	242.000000

The correlation between numeric columns like runtime, year, budget, and domestic is generated using heatmap. From the heat map we can see that the budget and revenue are highly correlated.



After performing all the conversions like converting the release date to datetime datatype, converting the runtime to minutes, and by removing unnecessary columns, the final cleaned data frame looks like this:

```
1 cleaned_mojo_upd_df.head()
```

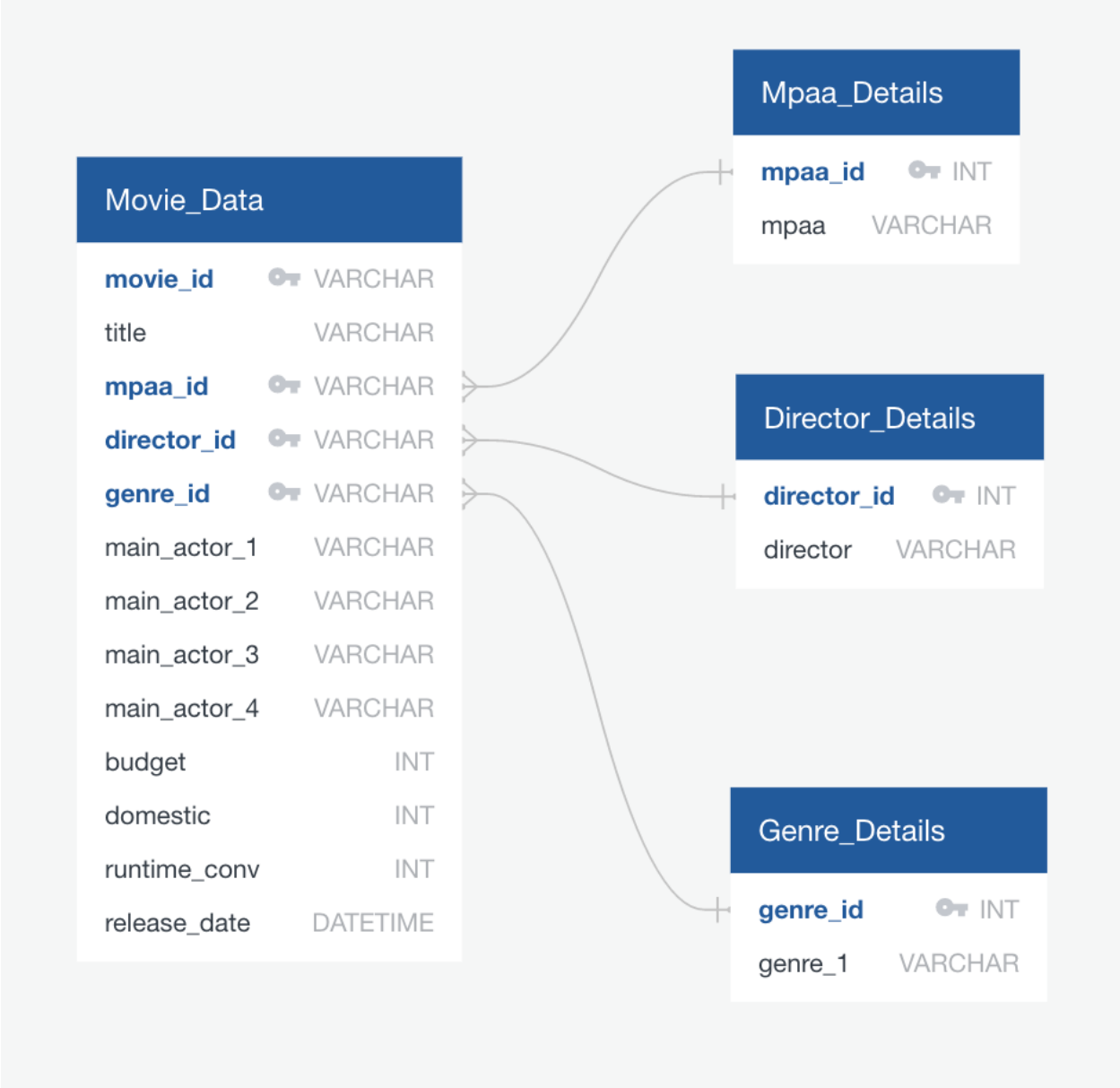
	movie_id	title	mpaa	director	main_actor_1	main_actor_2	main_actor_3	main_actor_4	budget	domestic	genre_1	runtime	release_date
0	tt0099165	The Bonfire of the Vanities	R	Brian De Palma	Tom Hanks	Bruce Willis	Melanie Griffith	Kim Cattrall	47000000.0	15691192.0	Comedy	125	1990-12-22
1	tt0099611	Frankenhooker	R	Frank Henenlotter	James Lorinz	Joanne Ritchie	Patty Mullen	J.J. Clark	2500000.0	205068.0	Comedy	85	1990-06-01
2	tt0099674	The Godfather: Part III	R	Francis Ford Coppola	Al Pacino	Diane Keaton	Andy Garcia	Talia Shire	54000000.0	66666062.0	Crime	162	1990-12-25
3	tt0099747	Havana	R	Sydney Pollack	Robert Redford	Lena Olin	Alan Arkin	Tomas Milian	40000000.0	9243140.0	Drama	144	1990-12-14
4	tt0100029	Lionheart	R	Sheldon Lettich	Jean-Claude Van Damme	Harrison Page	Deborah Rennard	Lisa Pelikan	6000000.0	24078196.0	Action	108	1990-01-11

The Exploratory Analysis is done to the Mojo budget update dataset and the data is loaded in the SQLite Database.

Database

The Cleaned Mojo Budget update data is loaded into the SQlite database. The Database contains four tables. Movies_Dable, Genre_Details, Directors_Details, and Mpaa_Details.

The following data will be output for our final database:



We could see that in the Movies_Data table: movie_id acts as a primary key, which is the unique key whereas mpaa_id, director_id, and genre_id act as foreign keys. We could access the details of mpaa, director, and genre by using these ids from the Movies_Data table.

The Genre dataframe is created by removing the duplicates and the data frame is loaded to the genre_details table.

```
genre_data_df.to_sql("genre_details", con=engine, method="multi", index=False, if_exists="replace")
```

```
# Make a connection to the SQL database
conn = engine.connect()

query = """
        SELECT
            *
        FROM
            genre_details;
        """

df1 = pd.read_sql(query, conn)
df1.head()
```

	genre_id	genre_1
0	1	Comedy
1	2	Crime
2	3	Drama
3	4	Action
4	5	Documentary

The Director details dataframe is created and the data is loaded into the director details table.

```
director_data_df.to_sql("director_details", con=engine, method="multi", index=False, if_exists="replace")
```

```
# Make a connection to the SQL database
conn = engine.connect()

query = """
        SELECT
            *
        FROM
            director_details;
        """

df3 = pd.read_sql(query, conn)
df3
```

	director_id	director
0	1	Brian De Palma
1	2	Frank Henenlotter
2	3	Francis Ford Coppola
3	4	Sydney Pollack
4	5	Sheldon Lettich
...	...	...
1415	1416	Lars Klevberg
1416	1417	Brian Kirk
1417	1418	Melina Matsoukas
1418	1419	Ari Aster
1419	1420	Chuck Konzelman

1420 rows × 2 columns

The mpaa dataframe is created and the data is loaded into the mpaa details table.

```
mpaa_data_df.to_sql("mpaa_details", con=engine, method="multi", index=False, if_exists="replace")
```

```
# Make a connection to the SQL database
conn = engine.connect()

query = """
        SELECT
            *
        FROM
            mpaa_details;
        """

df5 = pd.read_sql(query, conn)
df5.head()
```

	mpaa_id	mpaa
0	1	R
1	2	PG
2	3	PG-13
3	4	G
4	5	NC-17

The movies dataframe is created and the movies data is loaded into the movies data table.

```
movies_data_df.to_sql("movies_data", con=engine, method="multi", index=False, if_exists="replace")
```

```
# Make a connection to the SQL database
conn = engine.connect()

query = """
        SELECT
            *
        FROM
            movies_data;
        """

df6 = pd.read_sql(query, conn)
df6.head()
```

movie_id	title	mpaa_id	director_id	main_actor_1	main_actor_2	main_actor_3	main_actor_4	budget	domestic	genre_id	runtime	release_date
tt0099165	The Bonfire of the Vanities	1	1	Tom Hanks	Bruce Willis	Melanie Griffith	Kim Cattrall	47000000.0	15691192.0	1	125	1990-12-22
tt0099611	Frankenhooker	1	2	James Lorinz	Joanne Ritchie	Patty Mullen	J.J. Clark	2500000.0	205068.0	1	85	1990-06-01
tt0099674	The Godfather: Part III	1	3	Al Pacino	Diane Keaton	Andy Garcia	Talia Shire	54000000.0	66666062.0	2	162	1990-12-25
tt0099747	Havana	1	4	Robert Redford	Lena Olin	Alan Arkin	Tomas Milian	40000000.0	9243140.0	3	144	1990-12-14
tt0100029	Lionheart	1	5	Jean-Claude Van Damme	Harrison Page	Deborah Rennard	Lisa Pelikan	6000000.0	24078196.0	4	108	1990-01-11

To access the data from multiple tables we could use joins and query the table and get the desired result.

The maximum budget of the movie is calculated by executing the following query:

```
conn = engine.connect()

query = """
    SELECT
    max(m.budget),
    m.domestic,
    m.title,
    m.main_actor_1,
    m.main_actor_2,
    d.director,
    g.genre_1,
    m.release_date
    FROM
   movies_data as m
    INNER JOIN genre_details as g ON m.genre_id = g.genre_id
    INNER JOIN director_details as d ON m.director_id = d.director_id

    """
df10 = pd.read_sql(query, conn)
df10
```

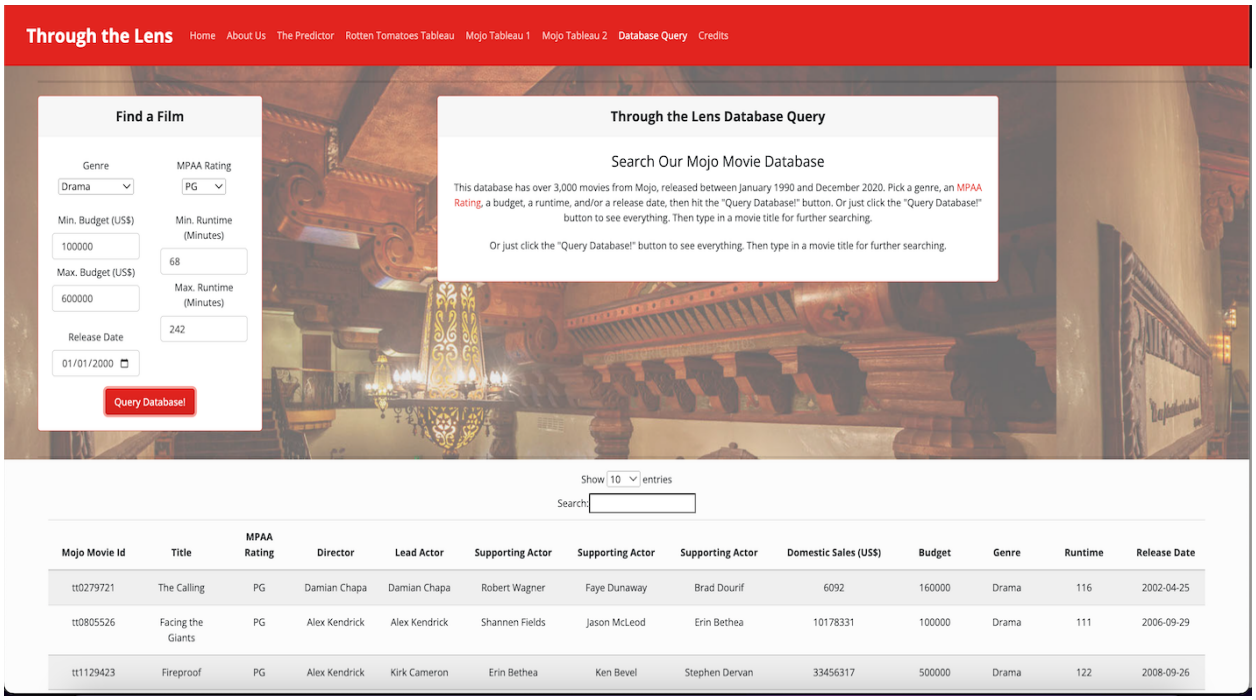
	max(m.budget)	domestic	title	main_actor_1	main_actor_2	director	genre_1	release_date
0	356000000.0	858373000.0	Avengers: Endgame	Robert Downey Jr.	Chris Evans	Anthony Russo	Action	2019-04-24

Based upon the genre, release date, minimum and maximum budget specified by the user, the following result is produced by querying the database.

```
conn = engine.connect()
genre = '"Drama"'
release_date = "2019-12-01"
min_budget = 100000
max_budget = 600000
query = f"""
    SELECT
    m.title,
    m.main_actor_1,
    m.main_actor_2,
    d.director,
    g.genre_1,
    m.release_date,
    m.budget
    FROM
   movies_data as m
    INNER JOIN genre_details as g ON m.genre_id = g.genre_id
    INNER JOIN director_details as d ON m.director_id = d.director_id
    where
    g.genre_1 in ({genre}) and
    m.release_date >= ({release_date}) and
    m.budget >= {min_budget} and m.budget <= {max_budget}
    order by m.title;

    """
df14 = pd.read_sql(query, conn)
df14
```

	title	main_actor_1	main_actor_2	director	genre_1	release_date	budget
0	A Ghost Story	Casey Affleck	Rooney Mara	David Lowery	Drama	2017-07-07	100000.0
1	A Long Way Off	Robert Davi	Edie McClurg	Michael Davis	Drama	2014-05-16	500000.0
2	Facing the Giants	Alex Kendrick	Shannen Fields	Alex Kendrick	Drama	2006-09-29	100000.0
3	Fireproof	Kirk Cameron	Erin Bethea	Alex Kendrick	Drama	2008-09-26	500000.0
4	Middle of Nowhere	Emayatzy Corinealdi	David Oyelowo	Ava DuVernay	Drama	2012-10-12	200000.0
5	Once	Glen Hansard	Markéta Irglová	John Carney	Drama	2007-03-23	150000.0
6	To Save a Life	Randy Wayne	Deja Kreutzberg	Brian Baugh	Drama	2009-01-22	500000.0

The final database in the website looks like this below:



Mojo Movie Id	Title	MPAA Rating	Director	Lead Actor	Supporting Actor	Supporting Actor	Supporting Actor	Domestic Sales (US$)	Budget	Genre	Runtime	Release Date
tt0279721	The Calling	PG	Damian Chapa	Damian Chapa	Robert Wagner	Faye Dunaway	Brad Dourif	6092	160000	Drama	116	2002-04-25
tt0805526	Facing the Giants	PG	Alex Kendrick	Alex Kendrick	Shannen Fields	Jason McLeod	Erin Bethea	10178331	100000	Drama	111	2006-09-29
tt1129423	Fireproof	PG	Alex Kendrick	Kirk Cameron	Erin Bethea	Ken Bevel	Stephen Dervan	33456317	500000	Drama	122	2008-09-26

Machine Learning

Predicting the Tomatometer score (critic and audience) for a film based on title length, runtime, production company, actors, authors, genres, and directors.

We tested multiple models and found that audience and critic reactions are difficult to predict reliably. Could this apply to multiple subjective pieces of art?

Our exploration also showcased a high success rating for NR films, which could be attributed to them being classics created before films were rated.

For Machine learning, the Rotten Tomatoes dataset has been used. The dataset has 17712 rows and 22 columns.It has columns like movie title, movie info, content rating, genres, directors, tomatometer rating, and audience rating.

```
rotten_tomatoes_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17712 entries, 0 to 17711
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   rotten_tomatoes_link            17712 non-null  object 
 1   movie_title                     17712 non-null  object 
 2   movie_info                      17391 non-null  object 
 3   critics_consensus               9134 non-null   object 
 4   content_rating                  17712 non-null  object 
 5   genres                          17693 non-null  object 
 6   directors                       17518 non-null  object 
 7   authors                         16170 non-null  object 
 8   actors                          17360 non-null  object 
 9   original_release_date           16546 non-null  object 
 10  streaming_release_date          17328 non-null  object 
 11  runtime                         17398 non-null  float64
 12  production_company              17213 non-null  object 
 13  tomatometer_status              17668 non-null  object 
 14  tomatometer_rating              17668 non-null  float64
 15  tomatometer_count               17668 non-null  float64
 16  audience_status                 17264 non-null  object 
 17  audience_rating                 17416 non-null  float64
 18  audience_count                  17415 non-null  float64
 19  tomatometer_top_critics_count   17712 non-null  int64  
 20  tomatometer_fresh_critics_count 17712 non-null  int64  
 21  tomatometer_rotten_critics_count 17712 non-null  int64  
dtypes: float64(5), int64(3), object(14)
memory usage: 3.0+ MB
```

Data Preprocessing:

We need to process the data before performing machine learning.

1. Removing the null values.
2. Dropping the irrelevant columns.

After preprocessing the dataset looks like this below:

```
df = df.loc[:,["movie_title", "movie_info", "content_rating", "genres", "directors", "authors", "actors",
               "original_release_date", "runtime", "production_company","tomatometer_rating","audience_rating"]]
df = df.dropna().reset_index(drop=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14639 entries, 0 to 14638
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   movie_title            14639 non-null  object
 1   movie_info             14639 non-null  object
 2   content_rating         14639 non-null  object
 3   genres                 14639 non-null  object
 4   directors              14639 non-null  object
 5   authors                14639 non-null  object
 6   actors                 14639 non-null  object
 7   original_release_date  14639 non-null  object
 8   runtime                14639 non-null  float64
 9   production_company     14639 non-null  object
 10  tomatometer_rating     14639 non-null  float64
 11  audience_rating        14639 non-null  float64
dtypes: float64(3), object(9)
memory usage: 1.3+ MB
```

Extracting year, month from the release date. Converting all the object data type columns to numeric columns by calculating length of the movie, length of the movie info, number of actors, number of genres, number of directors.

```
df["year"]=df.original_release_date.apply(lambda x: x.year)
```

```
df["month"]=df.original_release_date.apply(lambda x: x.month)
```

```
df["len_title"]=df.movie_title.apply(lambda x: len(x))
```

```
df["len_movie_info"]=df.movie_info.apply(lambda x: len(x))
```

```
df["num_directors"]=df.directors.apply(lambda x: getnum_ppl(x))
```

```
df["num_authors"]=df.authors.apply(lambda x: getnum_ppl(x))
```

```
df["num_actors"]=df.actors.apply(lambda x: getnum_ppl(x))
```

```
df["num_genres"]=df.genres.apply(lambda x: getnum_ppl(x))
```

```
def getnum_ppl(x):
    rtn = 0
    if len(x)>0:
        if "," in x:
            rtn = len(x.split(","))
        else:
            rtn = 1
    else:
        rtn = 0
    return(rtn)
```

```
def getfirst_genre(x):
    rtn = ""
    if len(x)>0:
        if "," in x:
            rtn = x.split(",")[0]
        else:
            rtn = x
    else:
        rtn = ""
    return(rtn)
```

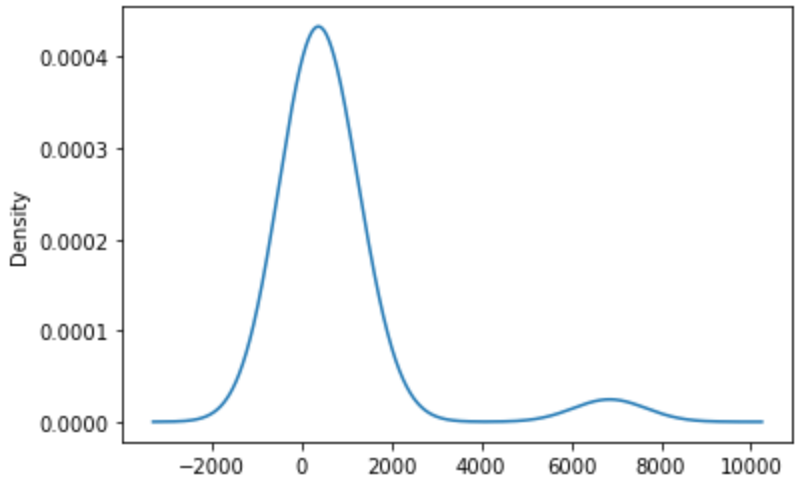
```
df["main_genre"]=df.genres.apply(lambda x: getfirst_genre(x))
```

The duplicates in the production company column are removed and then binning is applied as there are more than 10 different companies

.

```
production_cmp_count = df_prod_cmp.production_company.value_counts()
production_cmp_count
```

```
Other               6858
Warner Bros.        1156
Sony                 969
20th Century Fox     853
Paramount            749
Universal            745
MGM                  572
IFC                  455
Lionsgate            412
Columbia             254
Magnolia             232
Disney               231
Buena Vista          221
New Line             190
United Artists       173
Focus                144
Samuel Goldwyn       128
Netflix              104
Kino                 100
DreamWorks            93
Name: production_company, dtype: int64
```




```
# Determine which values to replace if counts are less than ...?
replace_production = list(production_cmp_count[production_cmp_count<230].index)

# Replace in dataframe
for app in replace_production:
    df_prod_cmp.production_company = df_prod_cmp.production_company.replace(app,"Other")

# Check to make sure binning was successful
df_prod_cmp.production_company.value_counts()
```

```
Other               8011
Warner Bros.        1156
Sony                 969
20th Century Fox     853
Paramount            749
Universal            745
MGM                  572
IFC                  455
Lionsgate            412
Columbia             254
Magnolia             232
Disney               231
Name: production_company, dtype: int64
```

After converting the featured columns to numeric. We need to perform one hot encoding. After that the dataset looks like this:

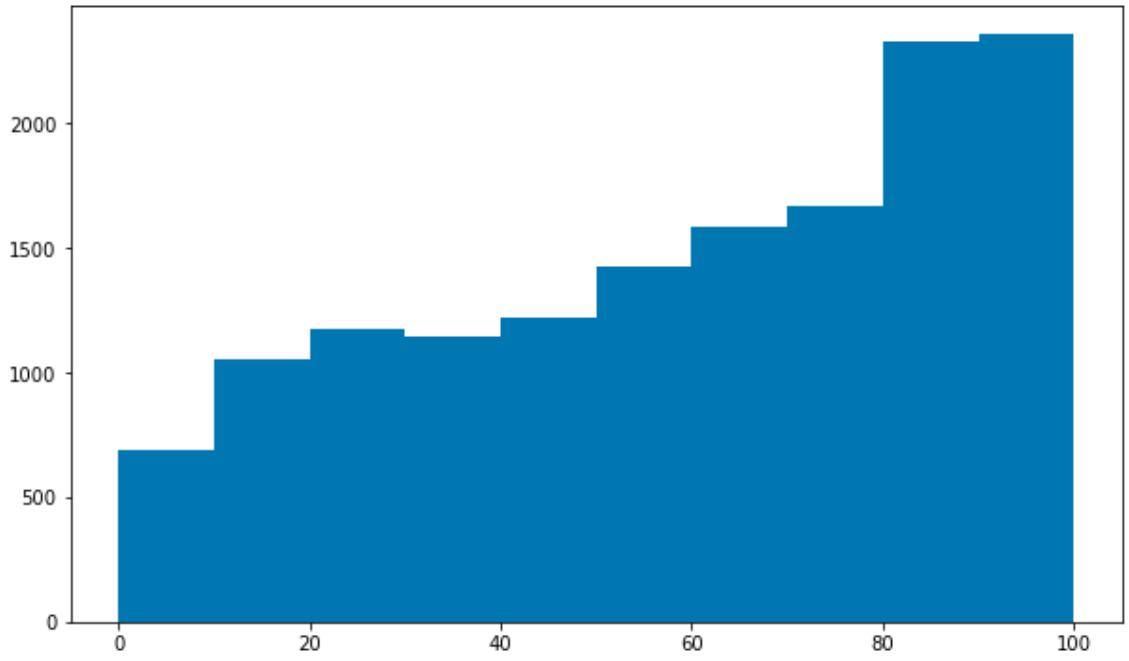
```
df_sub = df_prod_cmp.loc[:,['runtime','production_company',
       'tomatometer_rating',
       'month', 'len_title', 'len_movie_info', 'num_directors', 'num_authors',
       'num_actors', 'num_genres', 'main_genre']]

df_sub = pd.get_dummies(df_sub)
df_sub.head()
```

runtime	tomatometer_rating	month	len_title	len_movie_info	num_directors	num_authors	num_actors	num_genres	production_company_20th Century Fox	production_comp
119	49	2	50	454	1	3	87	4	1	
90	87	4	11	486	1	1	8	1	0	
122	67	10	2	279	1	1	46	2	0	
95	100	4	31	450	1	1	16	2	0	
127	89	1	28	489	1	1	16	3	0	

Model 1 - Tomatometer Rating as the target column

When we plot histogram for tomatometer rating we can see most of the movies are rated between 80 to 100.

```
plt.figure(figsize=(10,6))
plt.hist(df_sub["tomatometer_rating"])
plt.show()
```

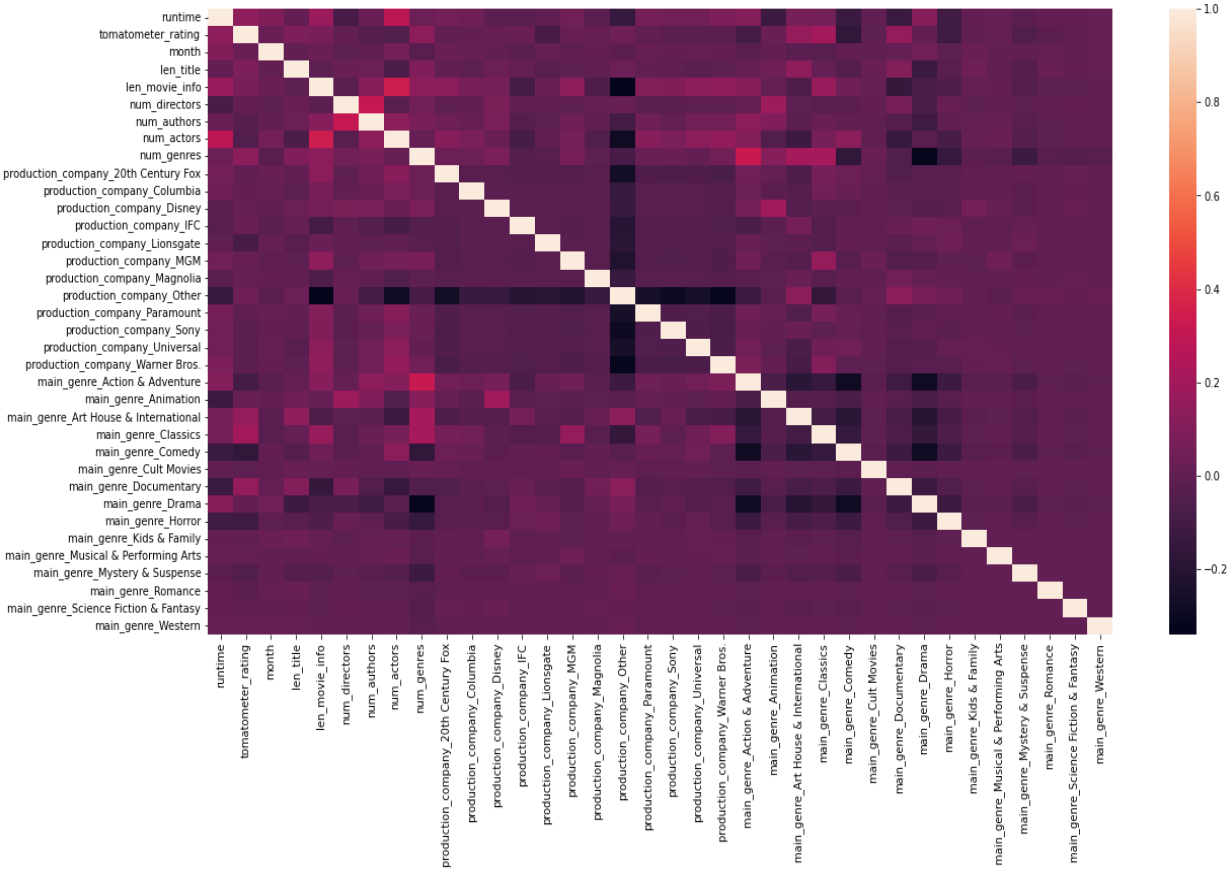


The correlation between tomatometer rating and various columns are found and the values are shown below. We can see that tomatometer rating is correlated with genres, runtime.

```
1 df_sub.corr().tomatometer_rating.sort_values()
```

```
main_genre_Comedy                       -0.164851
main_genre_Horror                       -0.115201
main_genre_Action & Adventure           -0.095710
production_company_Lionsgate            -0.084324
num_actors                              -0.055644
main_genre_Mystery & Suspense           -0.054285
num_authors                             -0.039303
production_company_Universal            -0.017550
production_company_Sony                 -0.017182
main_genre_Romance                      -0.016106
production_company_Warner Bros.         -0.015780
main_genre_Cult Movies                  -0.013276
production_company_Paramount            -0.009076
production_company_20th Century Fox     -0.002848
main_genre_Science Fiction & Fantasy    -0.002561
main_genre_Kids & Family                -0.001513
main_genre_Drama                         0.002825
num_directors                            0.003784
production_company_Columbia              0.005419
main_genre_Western                       0.006519
main_genre_Musical & Performing Arts     0.008029
production_company_Disney                0.015174
production_company_Magnolia              0.015972
production_company_MGM                   0.017223
main_genre_Animation                     0.019288
month                                    0.026198
production_company_IFC                   0.026344
production_company_Other                 0.033138
len_movie_info                           0.069530
len_title                                0.085048
num_genres                               0.128084
runtime                                  0.139216
main_genre_Documentary                   0.158738
main_genre_Art House & International     0.160886
main_genre_Classics                      0.203152
tomatometer_rating                       1.000000
Name: tomatometer_rating, dtype: float64
```

The sea born heatmap shows the correlation between tomatometer rating and various other columns.



Model 2 -Audience Rating as the target column

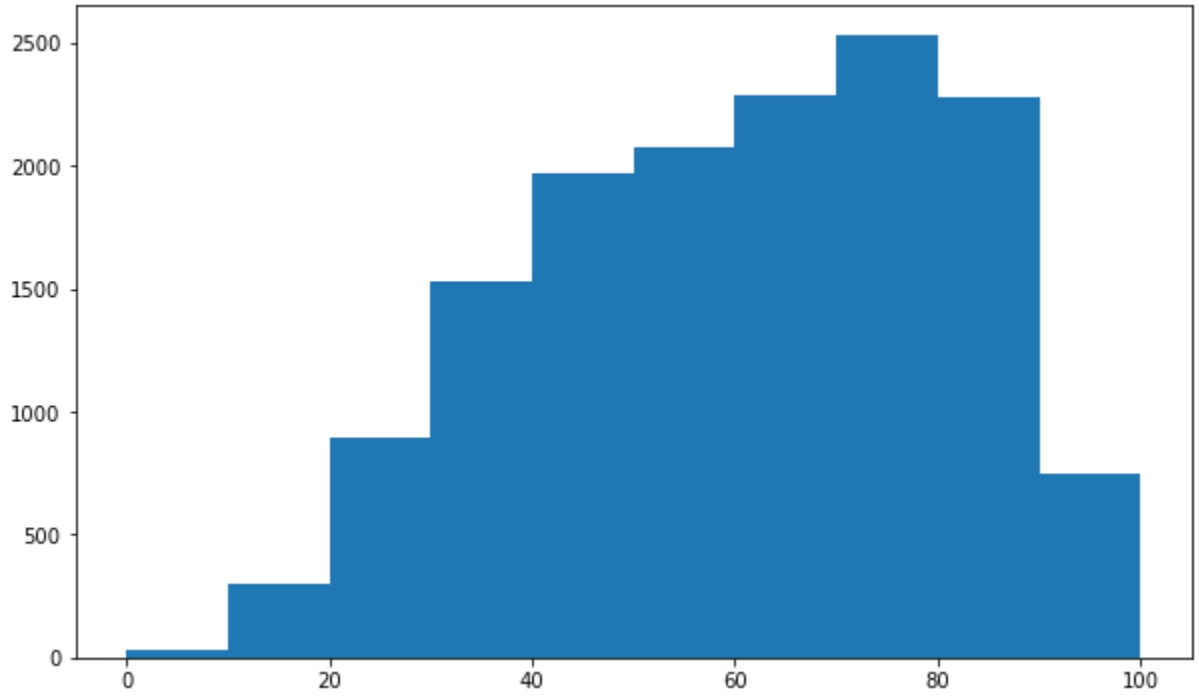
Similarly we are going to build a machine learning model for audience rating by following the same steps as we did in tomatometer rating.

```
df_sub = df_prod_cmp.loc[:,['runtime','production_company','audience_rating',
       'month', 'len_title', 'len_movie_info', 'num_directors', 'num_authors',
       'num_actors', 'num_genres', 'main_genre']]
df_sub = pd.get_dummies(df_sub)
df_sub.head()
```

	runtime	audience_rating	month	len_title	len_movie_info	num_directors	num_authors	num_actors	num_genres	production_company_20th Century Fox	production_comp
0	119	53	2	50	454	1	3	87	4	1	
1	90	64	4	11	486	1	1	8	1	0	
2	122	53	10	2	279	1	1	46	2	0	
3	95	97	4	31	450	1	1	16	2	0	
4	127	74	1	28	489	1	1	16	3	0	

When we plot histogram for audience rating we can see most of the movies are rated between 60 and 90.

```
plt.figure(figsize=(10,6))
plt.hist(df_sub["audience_rating"])
plt.show()
```

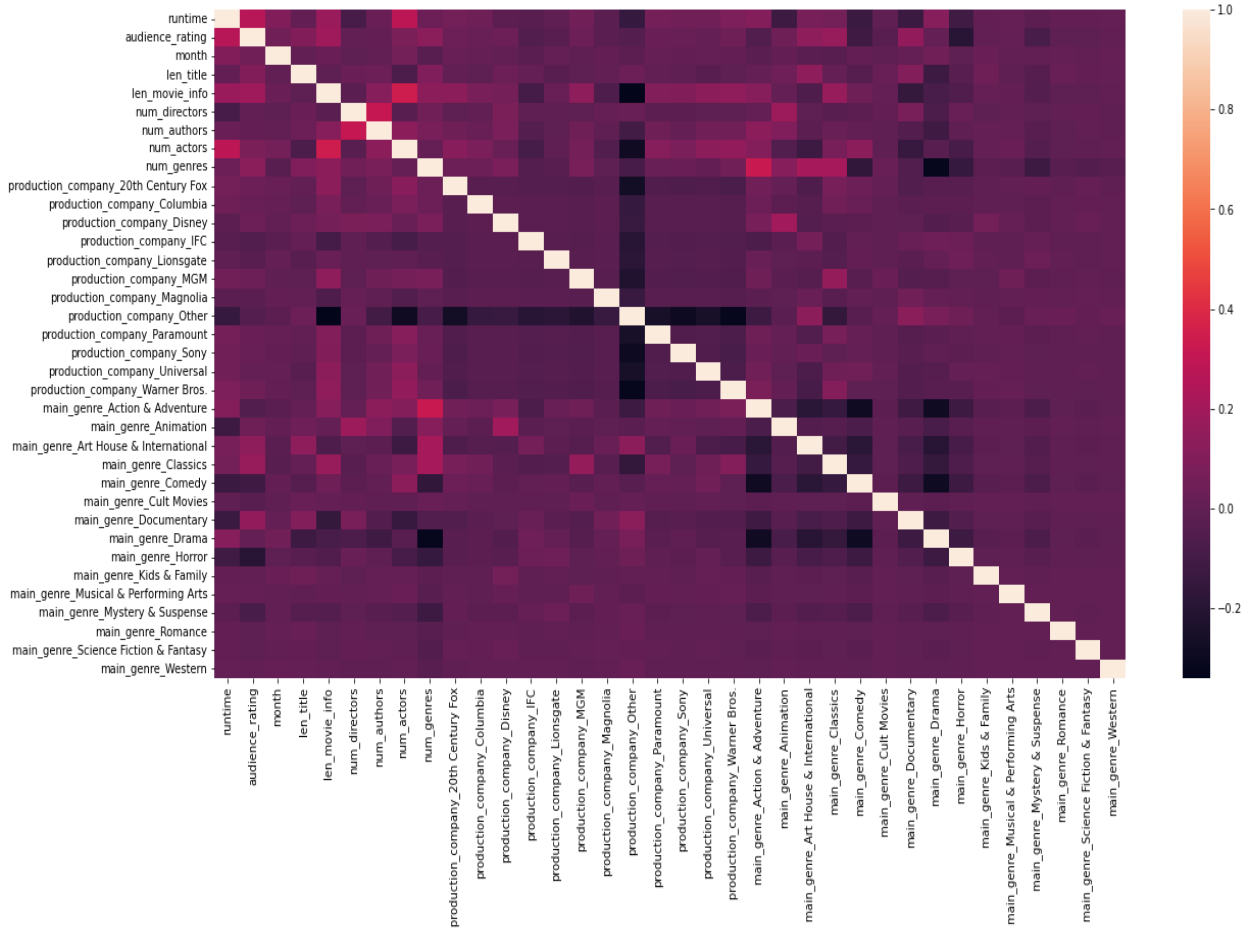


The correlation between audience rating and various columns are found and the values are shown below. We can see that audience rating is correlated with genres, runtime.

```
1 df_sub.corr().audience_rating.sort_values()
```

```
main_genre_Horror                        -0.196638
main_genre_Comedy                        -0.116856
main_genre_Mystery & Suspense            -0.079243
main_genre_Action & Adventure            -0.053654
production_company_IFC                   -0.053204
production_company_Other                 -0.043499
main_genre_Cult Movies                   -0.028917
production_company_Lionsgate             -0.027729
production_company_Magnolia              -0.024323
main_genre_Romance                       -0.013452
main_genre_Science Fiction & Fantasy     -0.012949
main_genre_Western                       -0.004438
main_genre_Kids & Family                 -0.003904
num_directors                            -0.001226
num_authors                               0.004470
production_company_Universal              0.008584
main_genre_Drama                          0.009311
main_genre_Musical & Performing Arts      0.011115
production_company_Columbia               0.017843
production_company_Sony                   0.018412
production_company_Paramount              0.022254
production_company_MGM                    0.028526
production_company_20th Century Fox       0.028555
production_company_Disney                 0.029762
production_company_Warner Bros.           0.032948
main_genre_Animation                      0.034611
month                                     0.044519
num_actors                                0.080935
len_title                                 0.098803
num_genres                                0.121378
main_genre_Art House & International      0.137251
main_genre_Documentary                    0.150187
main_genre_Classics                       0.162990
len_movie_info                            0.193659
runtime                                   0.266074
audience_rating                           1.000000
Name: audience_rating, dtype: float64
```

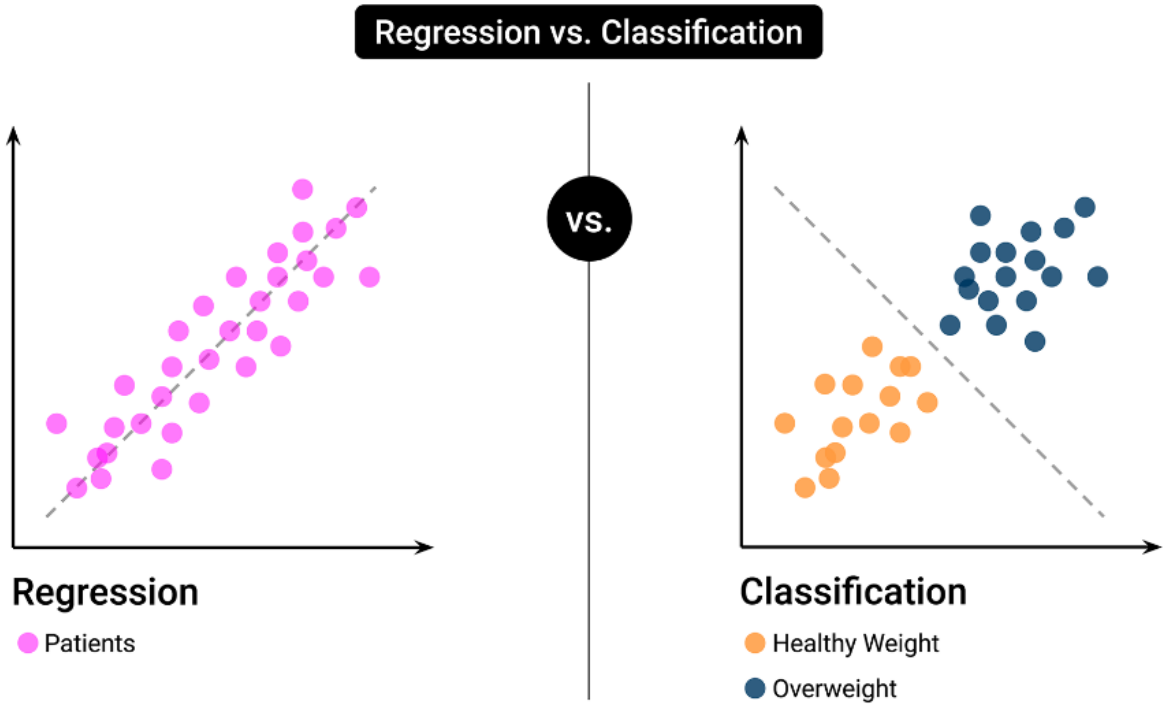
The sea born heatmap shows the correlation between audience rating and various other columns.



Since we know the target column as the tomatometer rating in model 1 (audience rating in model 2), we are using supervised machine learning.It deals with labeled data. Supervised machine learning are classified into two types:

1. Classification
2. Regression

Regression is used to predict continuous variables. We want to predict the tomatometer rating in model 1 (audience rating in model 2) based on movie length,number of actors,number of genres.To accomplish this task, we would collect data on a number of movies. The regression model's algorithms would attempt to learn patterns that exist among these factors. If presented with the data of a new movie, the model would make a prediction of that movie's tomatometer rating in model 1 (audience rating in model 2) based on previously learned patterns from the dataset.



A dataset is divided into features and target. Features are the variables used to make a prediction. Target is the predicted outcome. The basic procedure for implementing a supervised learning model: **create a model, train the model, and then create predictions.** We will be using Scikit-learn, a Python machine learning library for implementing linear regression model.

```
# data science libraries
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

# loading linear algorithms
from sklearn.linear_model import LinearRegression, ElasticNet, Lasso, Ridge

# tree based algorithms
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor, GradientBoostingRegress
#from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

# machine learning processing and metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

The target variable is `tomatometer rating for model 1` (audience rating for model 2) meaning that the goal of the linear regression model is to predict tomatometer rating for model 1(audience rating for model 2) based on the movie length, movie info,genre etc

.

```
# get features and target
X = df_final_train.drop("tomatometer_rating", axis=1)
y = df_final_train["tomatometer_rating"]
```

```
# this function takes in a model algorithm and the training data and prints graphs and metrics
def evaluateModel(model, X_train, y_train, X_test, y_test):
    # FIT THE MODEL
    model.fit(X_train, y_train)

    # EVALUATE the MODEL
    train_preds = model.predict(X_train)
    test_preds = model.predict(X_test)

    # PRINT the METRICS
    print("TRAINING SET")

    # Score the prediction with MSE and R2
    mse_train = np.sqrt(mean_squared_error(y_train, train_preds))
    r2_train = r2_score(y_train, train_preds)
    mae_train = mean_absolute_error(y_train, train_preds)

    print(f"root mean squared error (RMSE): {mse_train}")
    print(f"R-squared (R2 ): {r2_train}")
    print(f"MAE {mae_train}")

    print()

    # PRINT the METRICS
    print("Testing SET")

    # Score the prediction with MSE and R2
    mse_test = np.sqrt(mean_squared_error(y_test, test_preds))
    r2_test = r2_score(y_test, test_preds)
    mae_test = mean_absolute_error(y_test, test_preds)

    print(f"root mean squared error (RMSE): {mse_test}")
    print(f"R-squared (R2 ): {r2_test}")
    print(f"MAE {mae_test}")

    plt.figure(figsize=(10,6))
    plt.scatter(y_test, test_preds)
    plt.title("Actual vs Predicted Plot")
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.plot(y_test, y_test)
    plt.show()

    # Plot Residuals
    plt.figure(figsize=(10,6))
    plt.scatter(test_preds, test_preds - y_test)
    plt.hlines(y=0, xmin=test_preds.min(), xmax=test_preds.max(), color="r")
    plt.title("Residuals")
    plt.xlabel("Prediction")
    plt.ylabel("Error")
    plt.show()

    return(model)
```

The dataset has been separated into X and y components: X is the input data, and y is the output. We may have noticed that X is capitalized, while y is not,it's a common practice when using Scikit-learn.

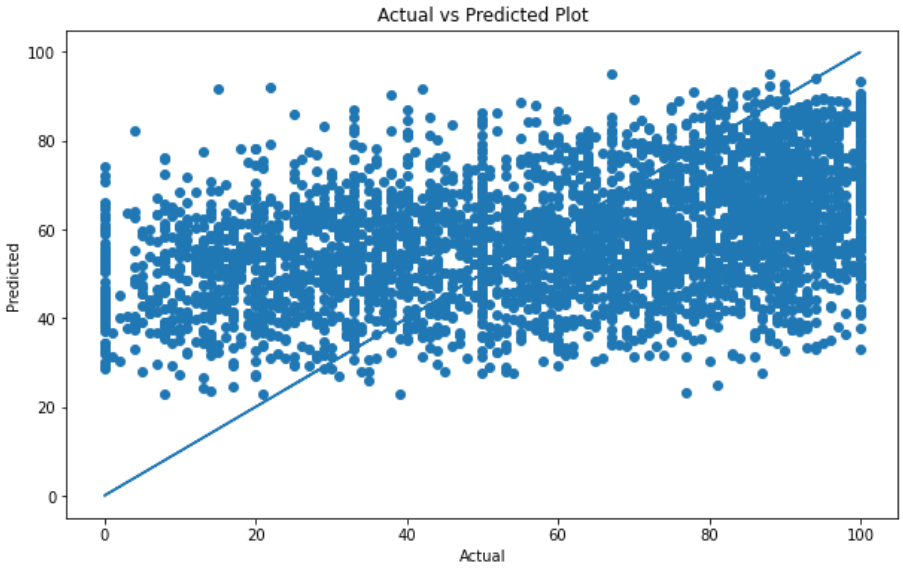
The next step is to create an instance of the linear regression model. An object is instantiated, or created, from `sklearn.linear_model`'s `LinearRegression` class. Instantiation here means that the `LinearRegression` class is like a template that contains the algorithms required to perform linear regression. From this template, a specific object called `model` is created that will analyze the data and store information specific to this dataset.After a model is instantiated, it will analyze the data and attempt to learn patterns in the data. This learning stage is alternatively called fitting or training.After the learning stage, the `predict()` method is used to generate predictions: given any data related to the movie, the model will predict the tomatometer rating for model 1(audience rating for model 2).

Model 1 - Tomatometer Rating as Target Variable : RESULTS

Linear Regression Results

```
lr = LinearRegression()
lr = evaluateModel(lr, X_train, y_train, X_test, y_test)
```

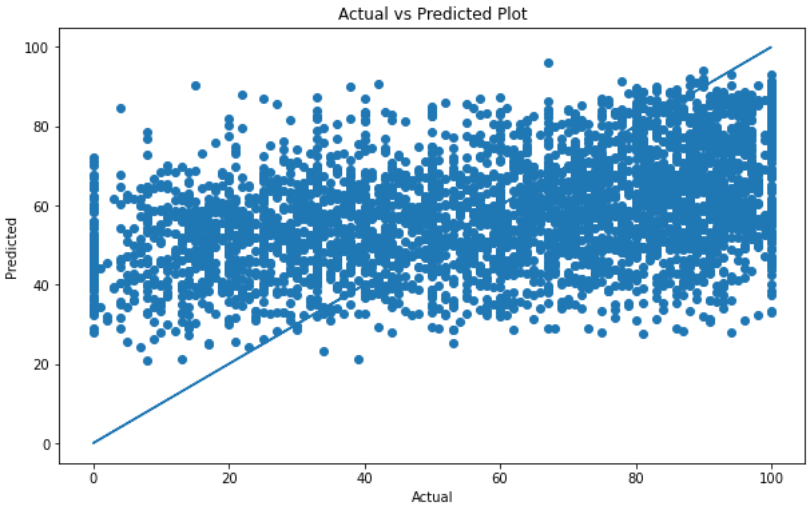
```
TRAINING SET
root mean squared error (RMSE): 25.85470746430405
R-squared (R2 ): 0.1669906495988981
MAE 21.687670612910022

Testing SET
root mean squared error (RMSE): 26.015956808034094
R-squared (R2 ): 0.1636576888461201
MAE 21.946257426629323
```



Random Forest Results:

```
rf = RandomForestRegressor()
rf = evaluateModel(rf, X_train, y_train, X_test, y_test)
```

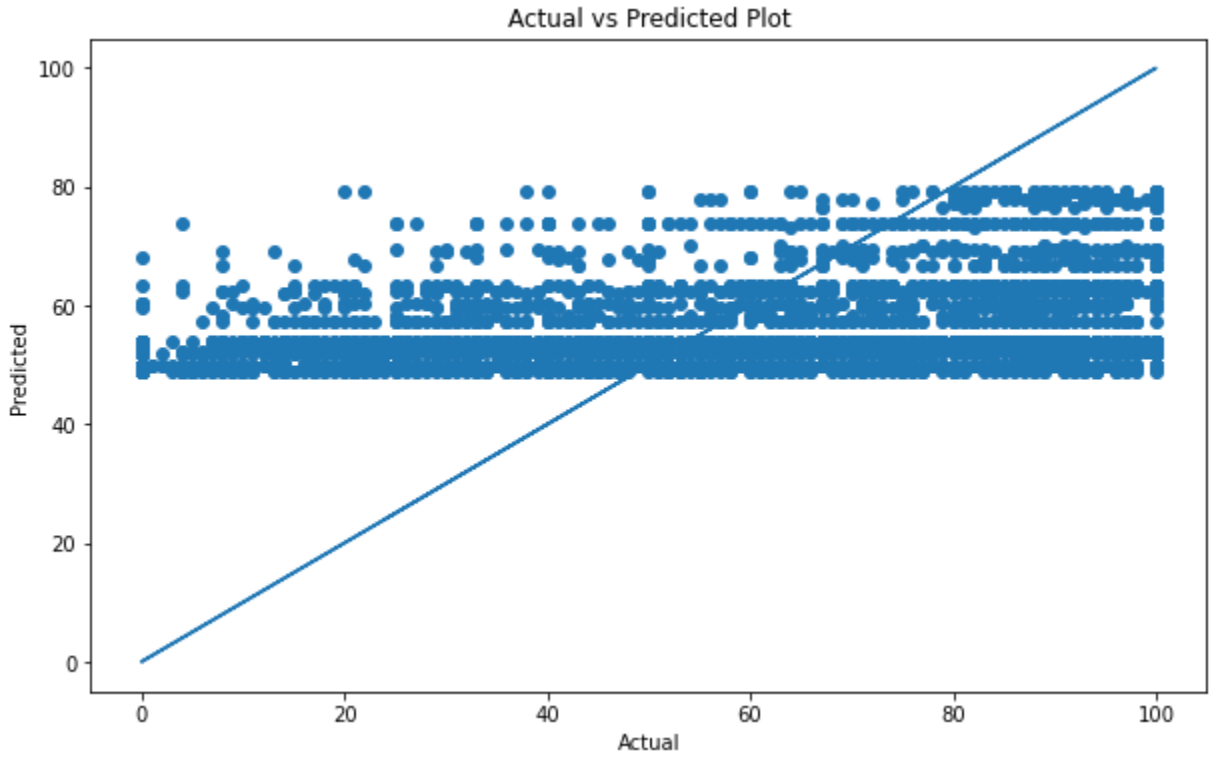
```
TRAINING SET
root mean squared error (RMSE): 9.703441319929354
R-squared (R2 ): 0.8826664562635935
MAE 7.9589479916203665
Testing SET
root mean squared error (RMSE): 26.183890586710575
R-squared (R2 ): 0.1528256114026858
MAE 21.734745901639343
```



AdaBoost Results

```
ada = AdaBoostRegressor(random_state=42)
ada = evaluateModel(ada, X_train, y_train, X_test, y_test)
```

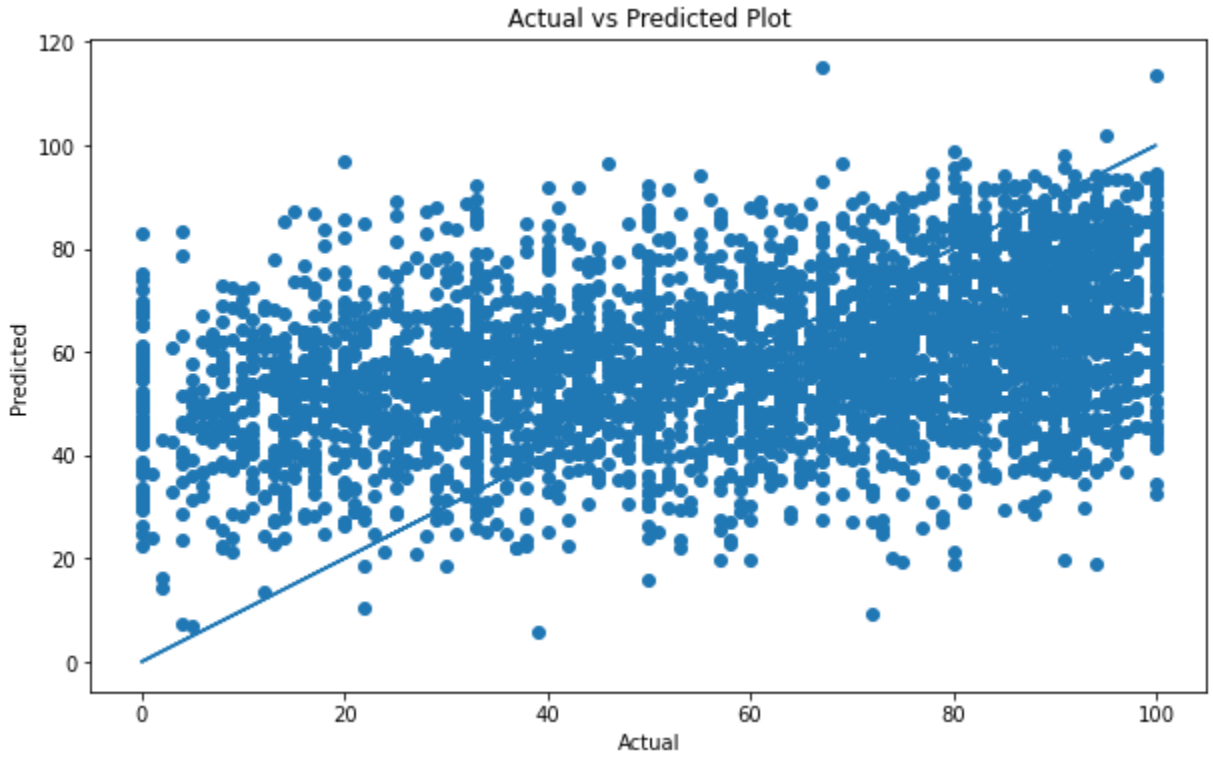
```
TRAINING SET
root mean squared error (RMSE): 26.287612679564386
R-squared (R2 ): 0.1388616822190002
MAE 22.575954485579626
Testing SET
root mean squared error (RMSE): 26.65646119042229
R-squared (R2 ): 0.12196980749719721
MAE 22.916505751949753
```



XGBRegressor Results:

```
1 xgb = XGBRegressor(random_state=42)
2 xgb = evaluateModel(xgb, X_train, y_train, X_test, y_test)
```

```
TRAINING SET
root mean squared error (RMSE): 18.566554052714608
R-squared (R2 ): 0.5704307037101941
MAE 14.840976566299206
Testing SET
root mean squared error (RMSE): 26.697014016191073
R-squared (R2 ): 0.11929625757916262
MAE 22.06131490436408
```

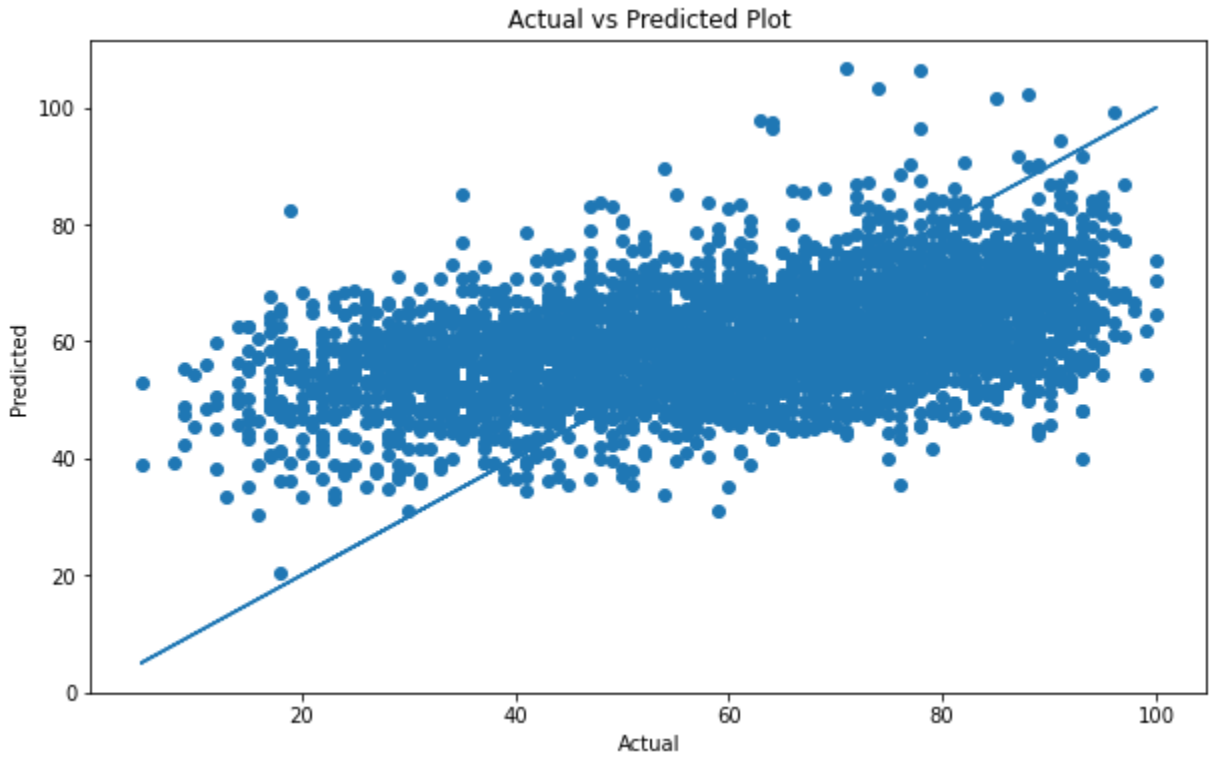


We have trained the same dataset with different models like Ada Boost Regressor, XGB Regressor, Gradient Boosting Regressor. But the R-squared value was less when compared to the linear regression model. **So we have finalized linear regression as our predicted model.**

Model 2 - Audience Rating as Target Variable : RESULTS

```
lr = LinearRegression()
lr = evaluateModel(lr, X_train, y_train, X_test, y_test)
```

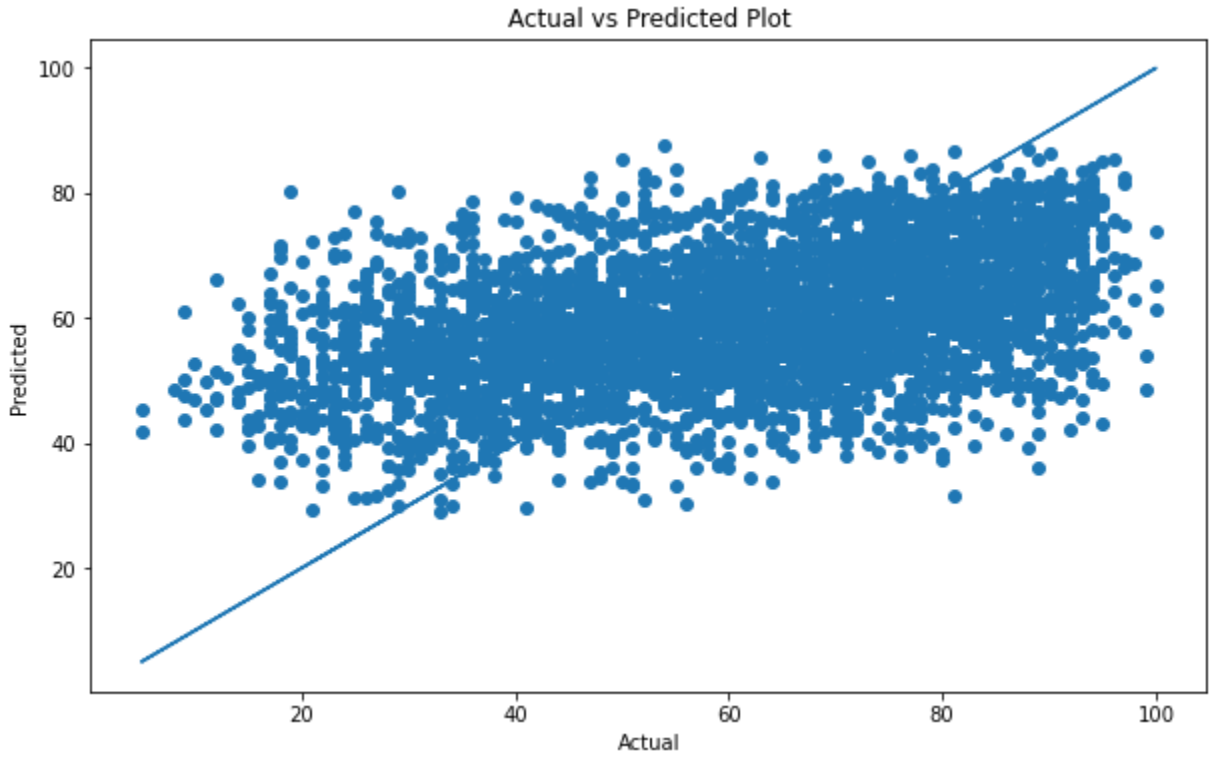
```
TRAINING SET
root mean squared error (RMSE): 17.98286252387678
R-squared (R2 ): 0.21746188540883715
MAE 14.83577584317367

Testing SET
root mean squared error (RMSE): 18.273162065333693
R-squared (R2 ): 0.2143786173628518
MAE 15.13769594909468
```



Random Forest Results:

```
rf = RandomForestRegressor()
rf = evaluateModel(rf, X_train, y_train, X_test, y_test)
```

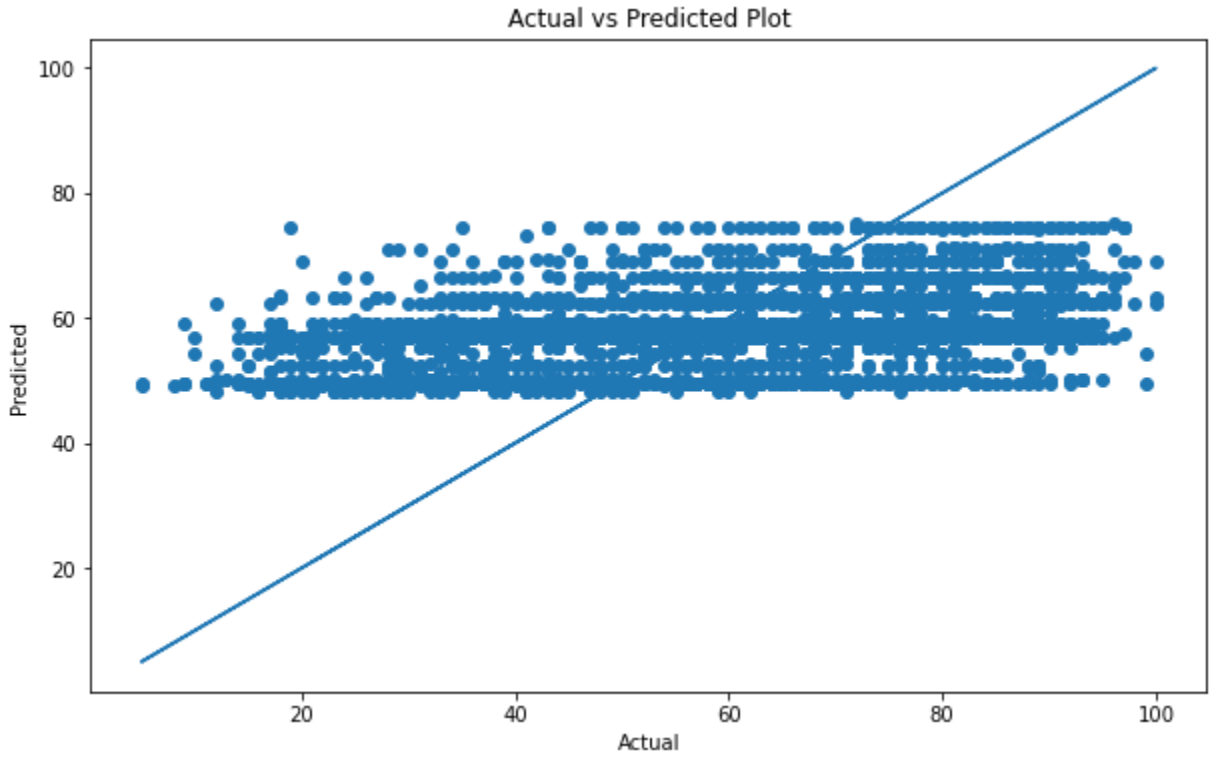
```
TRAINING SET
root mean squared error (RMSE): 6.77696351233956
R-squared (R2 ): 0.8888631339294428
MAE 5.482260679479006

Testing SET
root mean squared error (RMSE): 18.645077867590807
R-squared (R2 ): 0.1820734866441226
MAE 15.274366120218577
```



Ada Boost Results:

```
ada = AdaBoostRegressor(random_state=42)
ada = evaluateModel(ada, X_train, y_train, X_test, y_test)
```

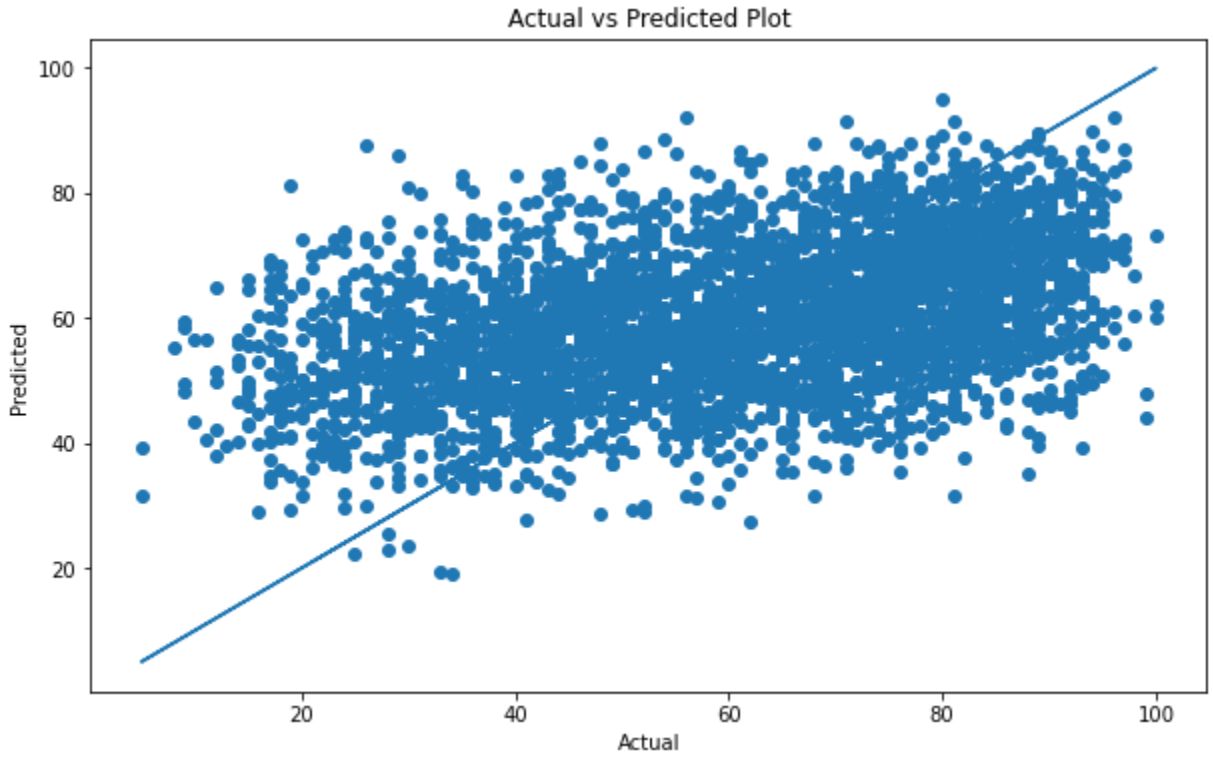
```
TRAINING SET
root mean squared error (RMSE): 18.637670741691736
R-squared (R2 ): 0.15943535057515323
MAE 15.730694333522669

Testing SET
root mean squared error (RMSE): 19.08154433057927
R-squared (R2 ): 0.14333125036082306
MAE 16.12231835724114
```



XGBRegressor Results:

```
1 xgb = XGBRegressor(random_state=42)
2 xgb = evaluateModel(xgb, X_train, y_train, X_test, y_test)
```

```
TRAINING SET
root mean squared error (RMSE): 12.752772361178652
R-squared (R2 ): 0.6064525829546246
MAE 10.107138761591093

Testing SET
root mean squared error (RMSE): 18.955512032544355
R-squared (R2 ): 0.15461035533251533
MAE 15.465407696187171
```



We have trained the same dataset with different models like Ada Boost Regressor, XGB Regressor, Gradient Boosting Regressor. But the R-squared value was less when compared to the linear regression model. **So we have finalized linear regression as our predicted model.**

The final predicted tomatometer rating for model 1 and audience rating for model 2 are merged and added in to a csv file.

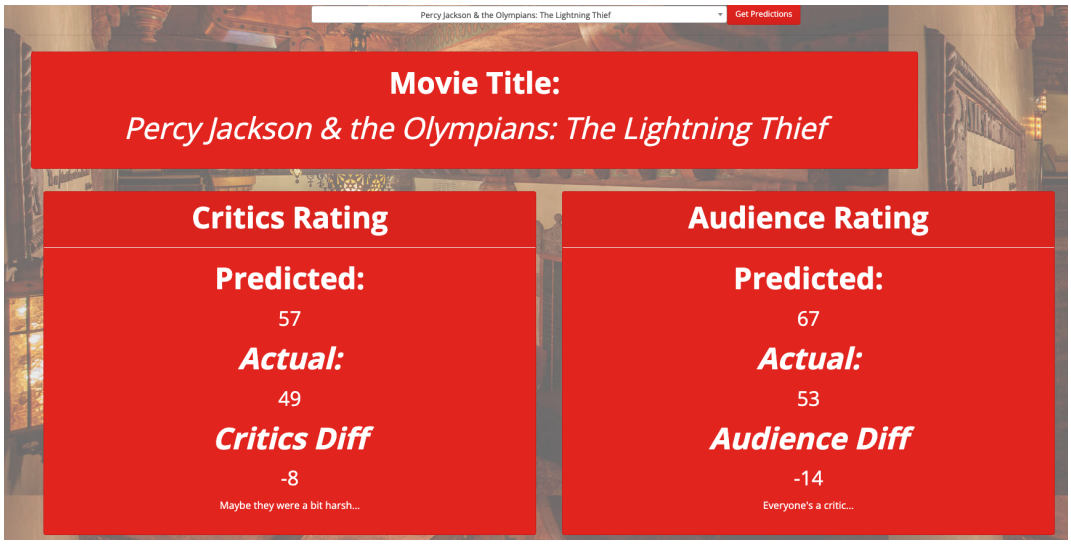
```
all_preds.loc[:,["movie_title",'tomatometer_rating', 'audience_rating','pred_tomatometer_rating',
       'pred_aud_rating']].head()
```

	movie_title	tomatometer_rating	audience_rating	pred_tomatometer_rating	pred_aud_rating
0	Percy Jackson & the Olympians: The Lightning T...	49.0	53.0	56.555671	67.412158
1	Please Give	87.0	64.0	47.565039	53.921476
2	10	67.0	53.0	49.614574	58.075423
3	12 Angry Men (Twelve Angry Men)	100.0	97.0	80.741472	70.288125
4	20,000 Leagues Under The Sea	89.0	74.0	68.127017	70.556199

Conclusion:

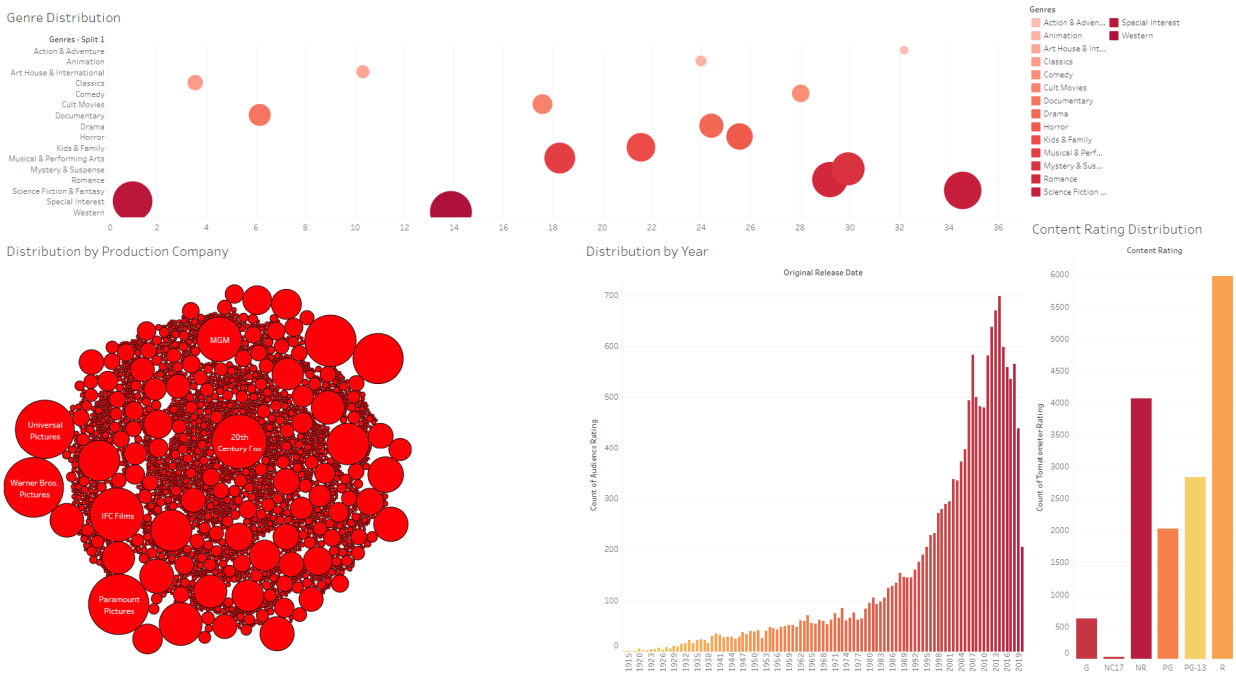
We have used various models like Ada Boost Regressor, XGB Regressor, Gradient Boosting Regressor for predicting the tomatometer rating for model 1 and audience rating for model 2. But we could not get the accuracy closer. We used tensor flow too for prediction, but the accuracy was very low there too. We may need to reduce the features and do some feature engineering in the future for better accuracy.

The Machine learning model on the website works like this:



Tableau

Rotten Tomatoes Dashboard



The Rotten Tomatoes Dashboard looks at the number of Rotten Tomatoes reviews by production company, genre, content rating, and date.

Rotten Tomatoes is a highly visited website that captures the critic and audience reviews of the many movies available. The Rotten Tomatoes dataset helped us in identifying trends based on the typical attributes of a movie such as genre, duration, director, as well as production company. We found that the audience rating and the critic rating averages were identical in outcome. The largest gap found when comparing the rating averages were in the Special Interest and Documentary genres. This dataset captured over 14,000 movies in total which gave us a large spread of release dates, even prior to the widespread usage of the world wide web. The content rating distribution showed a significant number of movies which were not rated, which could be due to the fact that content rating did not emerge until 1968 (G, PG, R, & X) and expanded in 1984 to include PG-13.

Box Office Mojo Dashboards

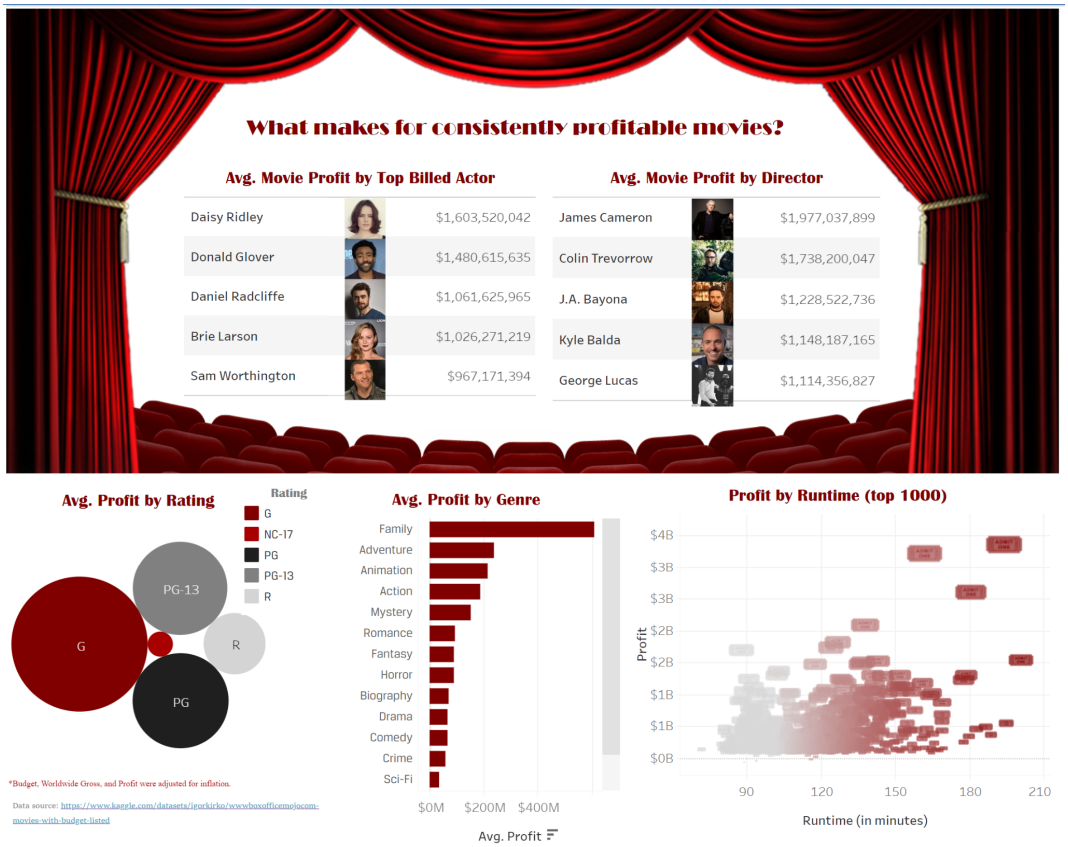
Before visualization began, the Box Office Mojo data was cleaned to adjust for inflation. To do this, we used the CPI with guidance from this article.

```
mojo_upd_df["infl_budget"] = mojo_upd_df.apply(lambda x: cpi.inflate(x["budget"], x["year"]), axis =1)
```

```
mojo_upd_df["infl_domestic"] = mojo_upd_df.apply(lambda x: cpi.inflate(x["domestic"], x["year"]), axis =1)
```

```
mojo_upd_df["infl_worldwide"] = mojo_upd_df.apply(lambda x: cpi.inflate(x["worldwide"], x["year"]), axis =1)
```

Box Office Mojo Dashboard 1: Average Profit by: Rating, Genre, Runtime, Director, and Actor



Overview

For this first visualization, we chose more static visuals to discover overarching trends in the data. One consistent decision made throughout this dashboard was to base trends on average profit. Utilizing the sum of profit would skew the top actors and directors to those with longer careers. The exception was for 'Profit by Runtime,' which has the top 1000 films by profit plotted individual.y.

Analysis

At the top, the first two visualizations explore the top actors and directors by average profit. These visualizations forced us to learn the names of the directors for all the new *Jurassic World* films, along with some un-surprising names (George Lucas, for example.) The actors, however, left me with a few major questions:

1. Donald Glover is fantastic, but his name was unexpected at the top of the list.
    a. Donald provided the voice for Simba in the modern remake of The Lion King, which had a large profit. Donald isn't billed as the first actor for many films, so his stock rose.
2. Sam Worthing-who?
    a. *Avatar*'s lead actor. He has other leading credits to his name, but this most certainly skewed his position.
3. Would a film starring Daisy and Daniel be an automatic money-maker?

The obvious take-aways from both of these visualizations are that franchises seem to be the key to a high average box office - Harry Potter, Jurassic World, and Star Wars reign both lists.

The bottom 3 visualizations tell two distinct stories. Average profit according to ratings and genre indicate that the broader your audience, the more likely your film is to be profitable. G ratings and the Family genre top the list.

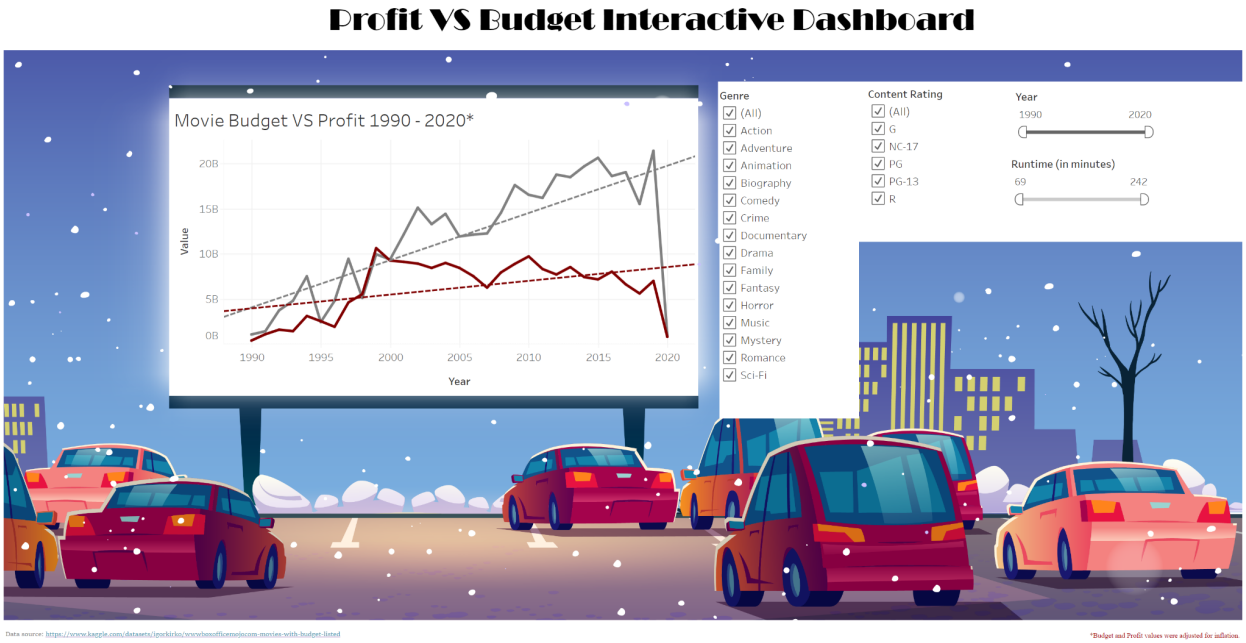
The visualization demonstrating average profit by runtime supports the theory that a successful film is most likely going to be a franchise or big budget remake. The visualization shows the top 1000 films, and a vast majority of the films fall in a brick near the center, demonstrating that runtime does not appear to significantly impact profit.

Next Steps

These visualizations provide a fascinating jumping off point that could prove useful if a soulless Hollywood executive wants to seek consistently profitable directors and actors. One way to deepen the analysis would be to incorporate the number of films as some type of filter. That *Avatar* guy struck gold once, but who knows if he'll manage it again?

It could prove interesting to expand the actors selection to include more than just top billing versus average profit. Another style of visualization could provide something more interactive to narrow parameters and find that perfect actor to match a project.

Box Office Mojo Dashboard 2: Profit vs Budget Interactive Dashboard



Overview

This interactive dashboard is designed to be an accompaniment to the first Box Office Mojo dashboard. Users can bring all their questions to this interactive dashboard and look at profit trends over time, including genre, content rating, year, and runtime.

Analysis

In its static form, the interactive dashboard shows an ongoing trend of movies yielding more and more profitable films as time goes on, with an interesting point in the late 90s where it would appear the industry was not profitable overall. There's a (to be expected) dropoff where the data ends in 2020, when the industry was put in a chokehold.

Next Steps

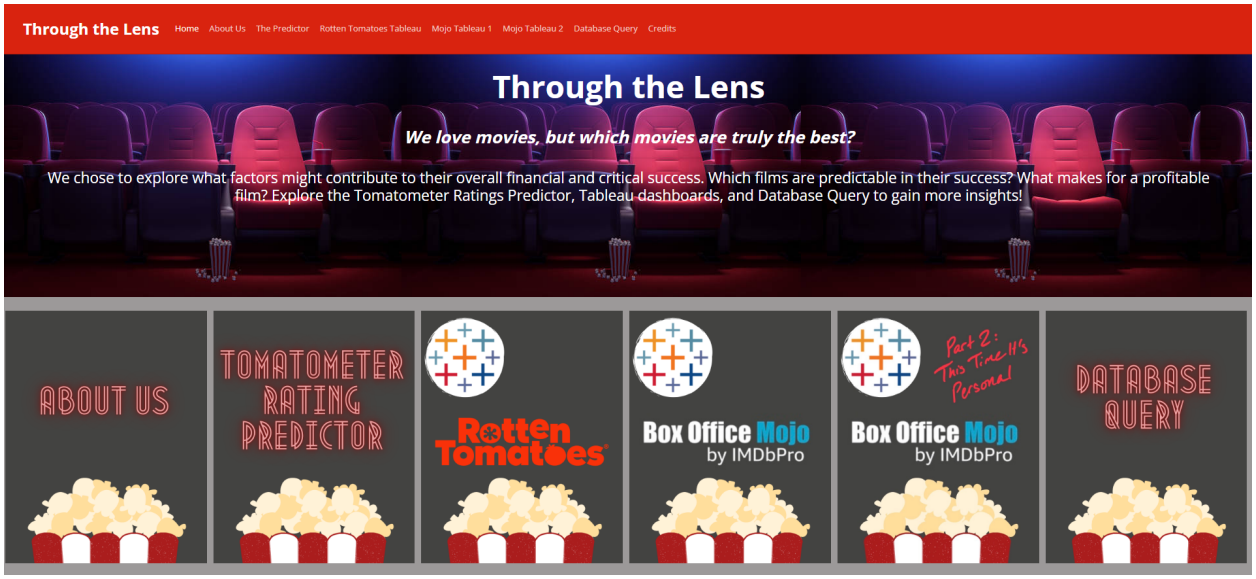
This dashboard would likely benefit from more data as well - perhaps awards data or ratings data could be integrated to create a more dynamic chart.

Conclusions

The Box Office Mojo Tableau dashboards seem to highlight this long-standing issue within the movie industry: franchise films or familiar concepts seem to out-perform the rest. Now this begs the question - are data-driven filmmaking decisions poised to demolish artistry? Has it already?

To deepen the analysis, it would be interesting to tie together audience and critic ratings, alongside the budget and profit analysis. Breaking things down further by studio could bring another layer of depth. Which studios are playing it safe and which are less risk-averse? Finally, it would be useful to see how certain films are performing on streaming versus at the movie theater. Is streaming killing the movie theater? Are movie theaters going the way of the dinosaur? There are so many more questions to answer with this data.

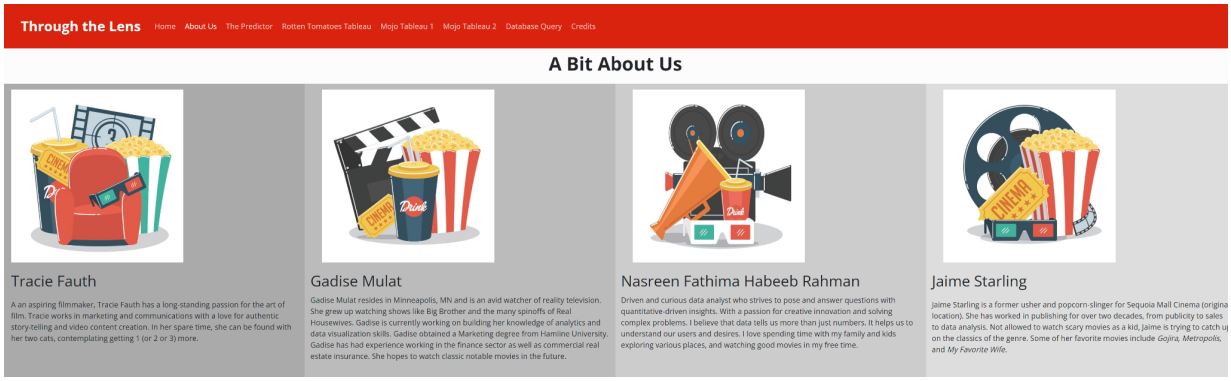
Web Application



The Through the Lens website was built and launched utilizing VS Code, Jupyter Notebooks, Flask, and Heroku.

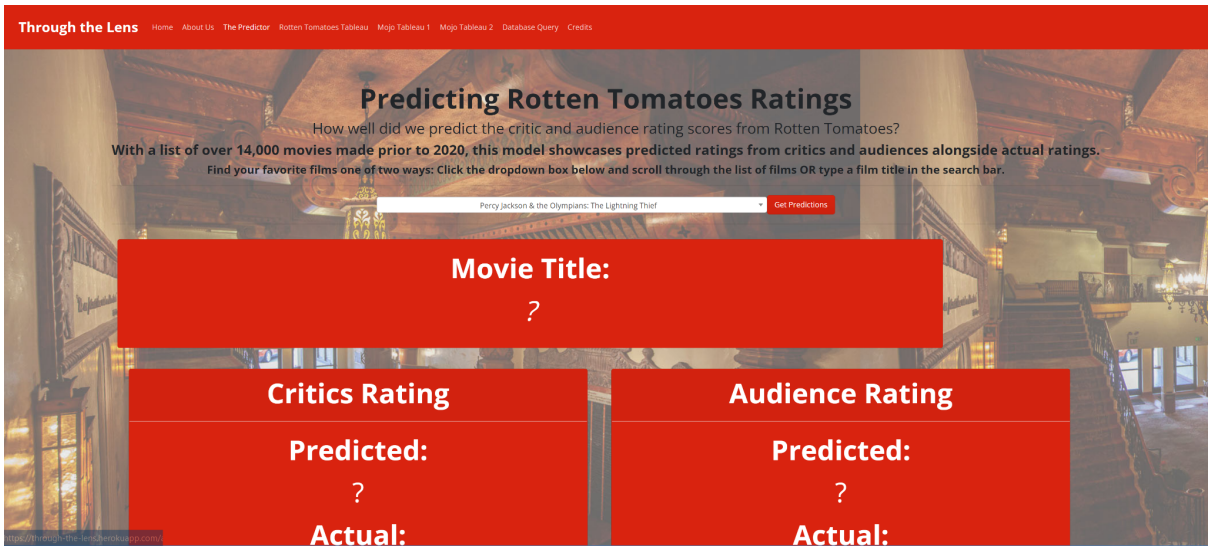
The final product of this project is the above website, which includes the following sections:

About Us



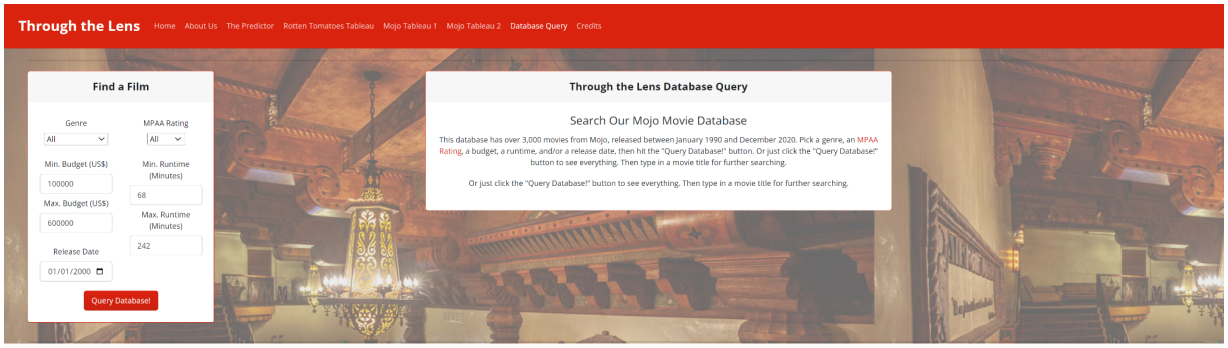
- Meet the team who worked on Through the Lens!

The Predictor



- Use our machine learning model to predict the audience and critic Tomatometer (Rotten Tomatoes) scores from a list of 14,000+ movies. You can also see the film's actual score to see how well our machine predicts critical behavior.

Database Query

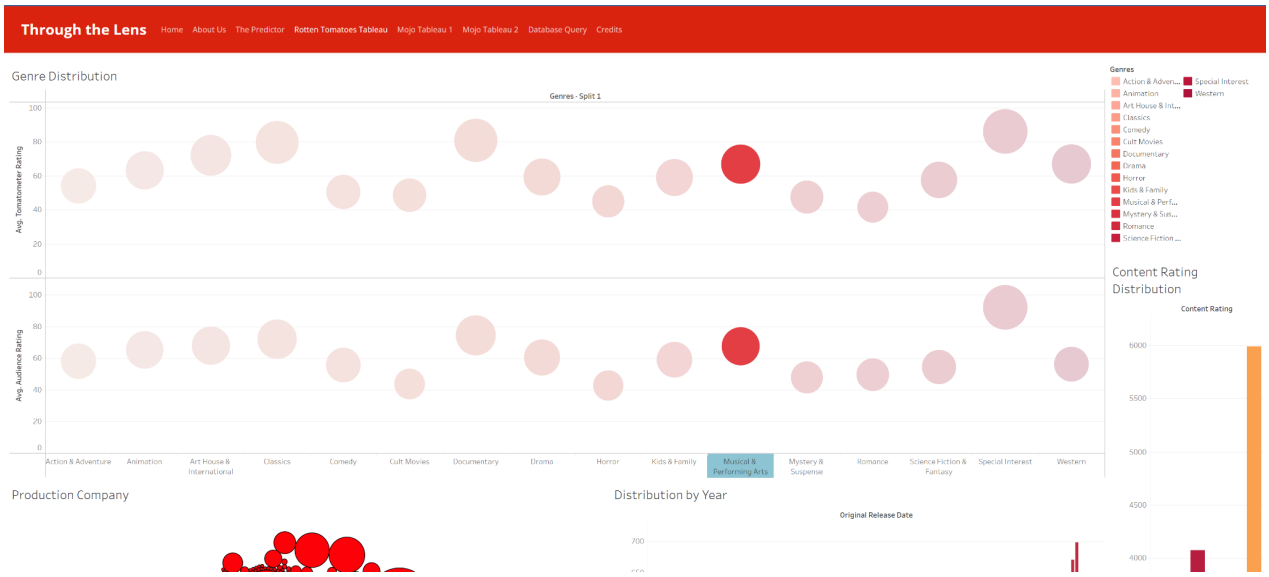


Show 10 entries
Search:

Mojo Movie Id	Title	MPAA Rating	Director	Lead Actor	Supporting Actor	Supporting Actor	Supporting Actor	Domestic Sales (US$)	Budget	Genre	Runtime	Release Date
tt0243415	Mercy Streets	PG-13	Jon Gunn	Eric Roberts	David A.R. White	Cynthia Watros	Shiek Mahmud-Bey	173599	600000	Action	106	2000-10-31
tt0279721	The Calling	PG	Damian Chapa	Damian Chapa	Robert Wagner	Faye Dunaway	Brad Dourif	6092	160000	Drama	116	2002-04-25
tt0280477	Better Luck Tomorrow	R	Justin Lin	Parry Shen	Jason Tobin	Sung Kang	Shirley Anderson	3802390	250000	Crime	101	2002-04-11
tt0285728	Dahmer	R	David Jacobson	Jeremy Renner	Bruce Davison	Artel Great	Matt Newton	144008	250000	Biography	101	2002-06-21

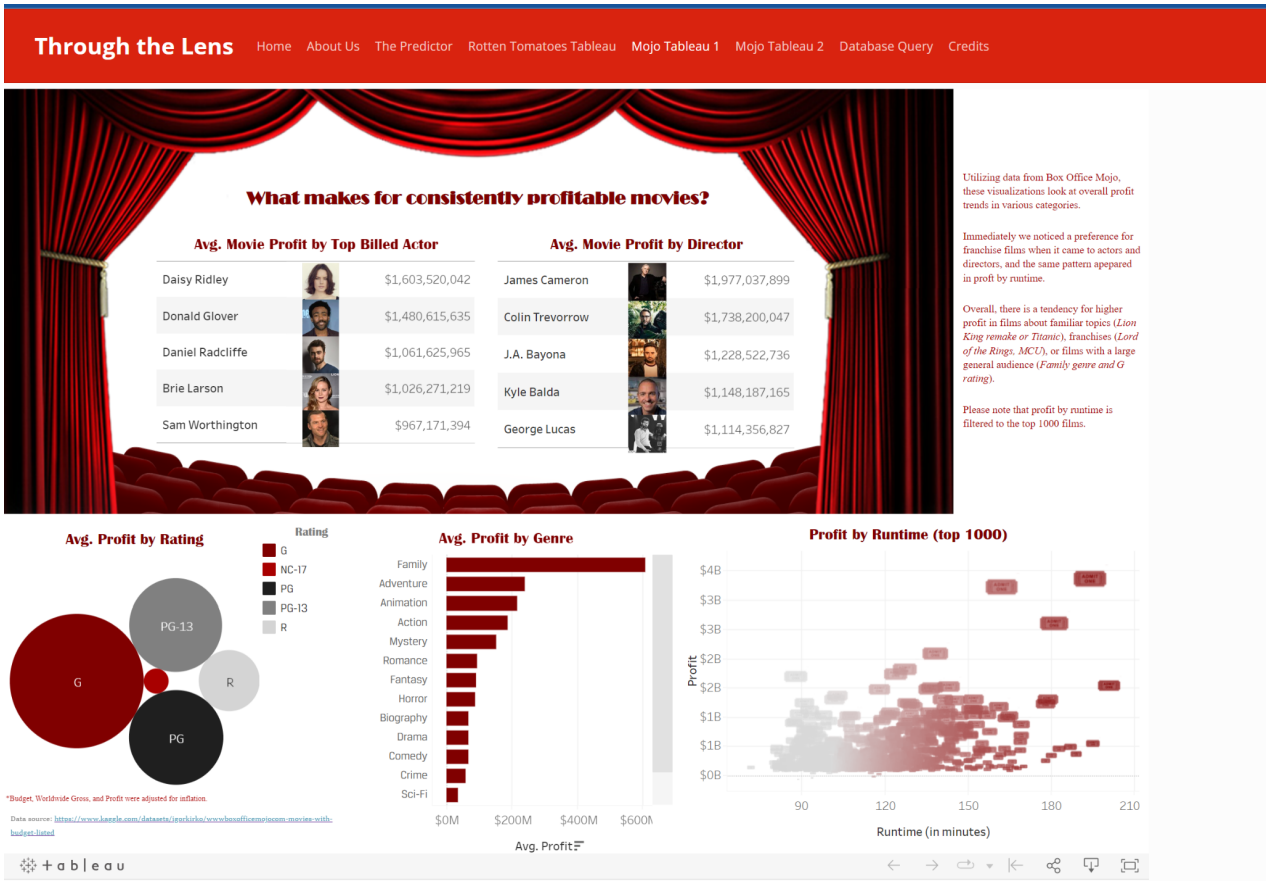
- Filter through the Box Office Mojo Data and explore the budget data.

Rotten Tomatoes Tableau



- See the evolution of Rotten Tomatoes, exploring the number of reviews by genre, studio, over time, and by content rating.

Mojo Tableau 1

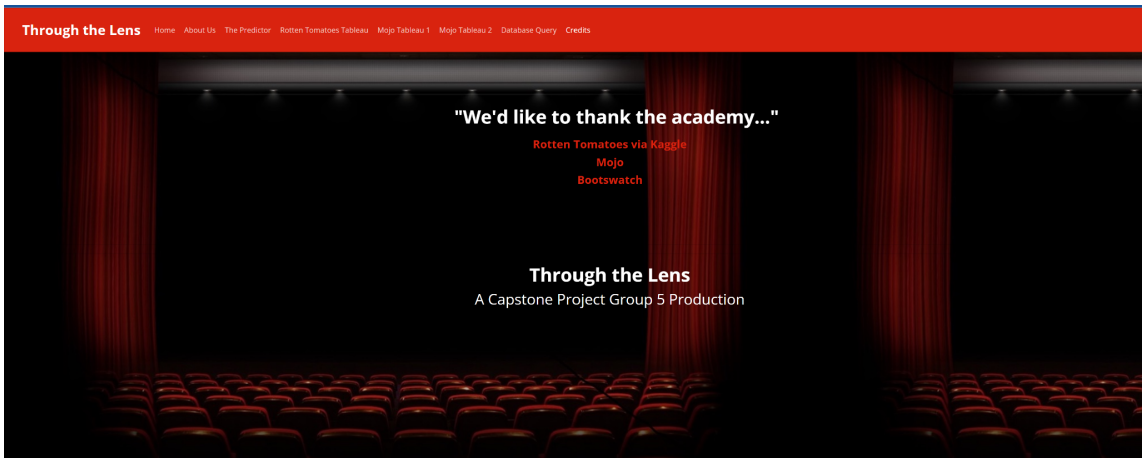


- Start with the static dashboard exploring the average profit by top billed actor, director, content rating, runtime, and genre.

Mojo Tableau 2

- Explore the Box Office Mojo data further, exploring film budgets and profits over time with filters for content rating, runtime, and genre.

Credits



- See our resources.

Background images and buttons for our pages were sourced or created via Google, Adobe Stock, or Canva. One important note is that the background image on the machine learning page The Predictor and Database Query is from team member Jaime's hometown. She frequented the theater as a child and was struck by its old school charm and beauty.

Conclusions

Our analysis genuinely concluded that human behavior can be difficult to predict, especially when it comes to their reactions to art and that franchises / familiar properties tend to dominate at the box office.

To deepen the overall analysis, the first thing we would do is tie together our data, connecting the financial and Rotten Tomatoes data, and perhaps pull in some additional ratings data. Seeing how these metrics interact might help us understand audience and critic behavior on a more meaningful level.

The Box Office Mojo Tableau dashboards seem to highlight this long-standing issue within the movie industry: franchise films or familiar concepts seem to out-perform the rest. Now this begs the question - are data-driven filmmaking decisions poised to demolish artistry? Has it already? Perhaps the data could help figure out ways to make new media that has a lessened risk without the promise of a big box-office payout.