IMDB Rating Predictor

Data Preparation & Visualization

Presenter: Yasin Zahir, Ted Wu

Project Background

The purpose of this project is to predict the IMDB rating score for over 5000 popularized movies. The model will be built based on certain features such as title, released year, alphabetized certificate, runtime in minutes, genre, IMDB rating, overview of the movie, the casts, and the director.

Therefore, the film review machine we will design can find the best combination of the film information and accurately predict the IMDB rating based on the customer preferences. Moreover, our model can benefit the user by displaying the attributes and features they prefer to find a high ranking movie.

Outline

- Data Preparation - Yasin Zahir
  - Feature Engineering
  - Cleaning
  - Imputation
  - Encoding
- Data Visualization - Ted Wu
  - Numerical diagram and Bar plot
  - Histograms and Analyzed
  - Scatter plot and Scatter matrix
  - Box Plot and Heatmaps

Initial Dataset

- We started with a dataset of 1000 entries and 15 features

In [16]:
```
1  # Gather info regarding dtypes of features
2  data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Poster_Link    1000 non-null   object
 1   Series_Title   1000 non-null   object
 2   Released_Year  1000 non-null   object
 3   Certificate    899 non-null    object
 4   Runtime        1000 non-null   object
 5   Genre          1000 non-null   object
 6   IMDB_Rating    1000 non-null   float64
 7   Overview       1000 non-null   object
 8   Meta_score     843 non-null    float64
 9   Director       1000 non-null   object
 10  Star1          1000 non-null   object
 11  Star2          1000 non-null   object
 12  Star3          1000 non-null   object
 13  Star4          1000 non-null   object
 14  No_of_Votes    1000 non-null   int64
 15  Gross          831 non-null    object
dtypes: float64(2), int64(1), object(13)
memory usage: 125.1+ KB
```

Initial Feature Engineering

- Dropped poster_link, overview, meta_score, star4, num_of_votes

```python
#2. Drop the 1000 IMDB features that we don't need (listed above)
# Poster_Link, Overview, Meta_score, Star4, No_of_Votes,

data = data.drop(['Poster_Link', 'Overview', 'Meta_score', 'Star4', 'No_of_Votes'], axis=1)
```

Initial Data Cleaning

- Random entry in Released_Year & needed to replace non-numeric entry with null value using lambda function

In [2]:

```python
# We begin preprocessing the Released Year column

# Check for missing or invalid values in the "Released Year" feature

#---------------------RESOLVED ISSUE---------------------
# Unique values generated below for year column are strings rather than ints
# We want to change that into an int

print(data['Released_Year'].isna().sum()) # number of missing values = 0
print(data['Released_Year'].unique()) # unique values in the column are strings entries with 1 'PG' entry
```

```
0
['1994' '1972' '2008' '1974' '1957' '2003' '1993' '2010' '1999' '2001'
 '1966' '2002' '1990' '1980' '1975' '2020' '2019' '2014' '1998' '1997'
 '1995' '1991' '1977' '1962' '1954' '1946' '2011' '2006' '2000' '1988'
 '1985' '1968' '1960' '1942' '1936' '1931' '2018' '2017' '2016' '2012'
 '2009' '2007' '1984' '1981' '1979' '1971' '1963' '1964' '1950' '1940'
 '2013' '2005' '2004' '1992' '1987' '1986' '1983' '1976' '1973' '1965'
 '1959' '1958' '1952' '1948' '1944' '1941' '1927' '1921' '2015' '1996'
 '1989' '1978' '1961' '1955' '1953' '1925' '1924' '1982' '1967' '1951'
 '1949' '1939' '1937' '1934' '1928' '1926' '1920' '1970' '1969' '1956'
 '1947' '1945' '1930' '1938' '1935' '1933' '1932' '1922' '1943' 'PG']
```

Data Cleaning

- Imputed Released_Year using mean and converted to int

```
In [6]:
# Take in account the new NaN value for the incorrect "PG" entry

# Fill in the NaN value with the mean for the column as the correct imputation for numerical data
data['Released_Year'] = data['Released_Year'].fillna(data["Released_Year"].mean()).astype(int)

# Update the new released year column with the imputation accounted for
data['Released_Year'] = data['Released_Year'].astype(int)

# Print the new unique values
print(data['Released_Year'].unique()) # No more strings,floats, or NaN values - all are integers
```

```
[1994 1972 2008 1974 1957 2003 1993 2010 1999 2001 1966 2002 1990 1980
 1975 2020 2019 2014 1998 1997 1995 1991 1977 1962 1954 1946 2011 2006
 2000 1988 1985 1968 1960 1942 1936 1931 2018 2017 2016 2012 2009 2007
 1984 1981 1979 1971 1963 1964 1950 1940 2013 2005 2004 1992 1987 1986
 1983 1976 1973 1965 1959 1958 1952 1948 1944 1941 1927 1921 2015 1996
 1989 1978 1961 1955 1953 1925 1924 1982 1967 1951 1949 1939 1937 1934
 1928 1926 1920 1970 1969 1956 1947 1945 1930 1938 1935 1933 1932 1922
 1943]
```

Loading & Merging Secondary Dataset

- List of 5k IMDB Movies, no order: 28 features, range of 5043 entries

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5043 entries, 0 to 5042
Data columns (total 28 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   color                      5024 non-null   object
 1   director_name              4939 non-null   object
 2   num_critic_for_reviews     4993 non-null   float64
 3   duration                   5028 non-null   float64
 4   director_facebook_likes    4939 non-null   float64
 5   actor_3_facebook_likes     5020 non-null   float64
 6   actor_2_name               5030 non-null   object
 7   actor_1_facebook_likes     5036 non-null   float64
 8   gross                      4159 non-null   float64
 9   genres                     5043 non-null   object
 10  actor_1_name               5036 non-null   object
 11  movie_title                5043 non-null   object
 12  num_voted_users            5043 non-null   int64
 13  cast_total_facebook_likes  5043 non-null   int64
 14  actor_3_name               5020 non-null   object
 15  facenumber_in_poster       5030 non-null   float64
 16  plot_keywords              4890 non-null   object
 17  movie_imdb_link            5043 non-null   object
 18  num_user_for_reviews       5022 non-null   float64
 19  language                   5031 non-null   object
 20  country                    5038 non-null   object
 21  content_rating             4740 non-null   object
 22  budget                     4551 non-null   float64
 23  title_year                 4935 non-null   float64
 24  actor_2_facebook_likes     5030 non-null   float64
 25  imdb_score                 5043 non-null   float64
 26  aspect_ratio               4714 non-null   float64
 27  movie_facebook_likes       5043 non-null   int64
dtypes: float64(13), int64(3), object(12)
memory usage: 1.1+ MB
```

Rename Features To Match Prior To Merge

- Initially I merged without proper cleaning and had errors due to duplicates
- These were compatible features but needed individual cleaning to match

In [12]:

```
#3. Rename the features in 5000 IMDB dataframe to match the first
# Series_Title = movie_title, Released_Year = title_year, Certificate = content_rating, Genre = genres
# IMDB_Rating = imdb_score, Runtime = duration, Director = director_name, Star1 = actor_1_name
# Star2 = actor_2_name, Star3 = actor_3_name, Gross = gross

data = data.rename(columns={
    # Renamed features
    'Series_Title': 'movie_title',
    'Released_Year': 'title_year',
    'Certificate': 'content_rating',
    'Genre': 'genres',
    'IMDB_Rating': 'imdb_score',
    'Runtime': 'duration',
    'Director': 'director_name',
    'Star1': 'actor_1_name',
    'Star2': 'actor_2_name',
    'Star3': 'actor_3_name',
    'Gross': 'gross'
})
```

Prior To Dropping Duplicates

- Before dropping duplicate entries in each df
    - The string entries needed to be cleaned of capitalization and trailing white spaces

```python
#1. Normalize the movie titles in both dataframes to get rid of inconsistencies so it matches
data['movie_title'] = data['Series_Title'].str.lower().str.strip()
data2['movie_title'] = data2['movie_title'].str.lower().str.strip()
```

Dropping Duplicate Tuples

- Dropped duplicates rows based off movie_title feature
  - Result Df1 = 1 row removed
  - Result Df2 = 127 rows removed

```python
# Prior to merging both dataframes I got the error that there were duplicate movie entries in each dataframe
# ERROR "The column label 'movie_title' is not unique."

#Now individually droping duplicates has worked and prepared the environment for a merge

data = data.drop_duplicates(subset=['movie_title'])
data2 = data2.drop_duplicates(subset=['movie_title'])

# Aftering generating info on both dataframes, I see that only 1 duplicate entry existed for 1st dataframe
# Dataframe 2, however, has gone from a range of 5043 entries to 4916 --> this had the most duplicate entries
```

Further Data Cleaning in Primary Dataset

- Get rid of commas in gross
- Get rid of ‘mins’ in duration

```
# delete 'mins'
data['duration'] = data['duration'].str.replace('min', '')
data['duration'] = data['duration'].astype('Int64')
```

```
# delete commas in the gross feature
data['gross'] = data['gross'].str.replace(',', '')
data['gross'] = data['gross'].astype('Int64')
```

Impute Missing Values in Primary Dataset

- STILL CANNOT MERGE YET!
  - content_rating imputation - object
  - gross imputation - numeric value requires skew
    - Not listed: ‘budget’

```
In [ ]:
# Here we can perform imputation on missing values prior to merging
```

```
In [33]:
# imputation for content_rating using mode
data['content_rating'].fillna(value=data['content_rating'].mode()[0], inplace=True)
```

```
In [44]:
# The skew method shows that it has an extreme positive skew over 3 so we will use median rather than mean
data['gross'].skew()

median_gross = int(data['gross'].median())
data['gross'].fillna(median_gross, inplace=True)

```

Feature Engineering in Secondary Dataset

- Drop Irrelevant features
  - director_facebook_likes
  - actor_facebook_likes (1-3)
  - movie_imdb_link
  - movie_facebook_likes
  - etc

```
data2.drop(['director_facebook_likes', 'actor_3_facebook_likes', 'actor_1_facebook_likes',
            'cast_total_facebook_likes', 'facenumber_in_poster', 'plot_keywords',
            'movie_imdb_link', 'actor_2_facebook_likes', 'aspect_ratio', 'movie_facebook_likes'],
           axis=1, inplace=True)
```

Convert Numerical Data into Integers in Secondary Dataset

- num_critic_for _reviews
- num_voted_users
- num_user_for_reviews
- Budget
- etc

```
data2['num_critic_for_reviews'] = data2['num_critic_for_reviews'].astype('Int64')
data2['num_voted_users'] = data2['num_voted_users'].astype('Int64')
data2['num_user_for_reviews'] = data2['num_user_for_reviews'].astype('Int64')
data2['budget'] = data2['budget'].astype('Int64')

```

```
data2['title_year'] = data2['title_year'].astype('Int64')
```

```
data2['duration'] = data2['duration'].astype('Int64')
```

```
data2['gross'] = data2['gross'].astype('Int64')
```

Imputation in Secondary Dataset

- Categorical objects= mode
- Skewed distribution positive or negative = median
- No symmetrical distribution was found for mean

```
# Impute 100 missing values in director name column with mode
data2['director_name'].fillna(value=data2['director_name'].mode()[0], inplace=True)
```

```
# The skew method shows that it has an extreme positive skew of 1.5 so we will use median rather than mean
# imputing for 15 missing values
data2['duration'].skew()

median_duration= int(data2['duration'].median())
data2['duration'].fillna(median_duration, inplace=True)
```

```
# imputation for actor_2_name using mode
data2['actor_2_name'].fillna(value=data2['actor_2_name'].mode()[0], inplace=True)
```

```
# The skew method shows that it has an extreme positive skew of over 3 so we will use median rather than mean
# imputing for 862 missing values

data2['gross'].skew()

median_gross = int(data2['gross'].median())
data2['gross'].fillna(median_gross, inplace=True)
```

Merge Datasets

- Merged_df = 5915 range of entries & 18 total features

```
1 merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5915 entries, 0 to 5042
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   movie_title             5915 non-null   object
 1   title_year              5915 non-null   Int64
 2   content_rating          5915 non-null   object
 3   duration                5915 non-null   Int64
 4   genres                  5915 non-null   object
 5   imdb_score              5915 non-null   float64
 6   director_name           5915 non-null   object
 7   actor_1_name            5915 non-null   object
 8   actor_2_name            5915 non-null   object
 9   actor_3_name            5915 non-null   object
 10  gross                   5915 non-null   Int64
 11  color                   4916 non-null   object
 12  num_critic_for_reviews  4916 non-null   Int64
 13  num_voted_users         4916 non-null   Int64
 14  num_user_for_reviews    4916 non-null   Int64
 15  language                4916 non-null   object
 16  country                 4916 non-null   object
 17  budget                  4916 non-null   Int64
dtypes: Int64(7), float64(1), object(10)
memory usage: 918.4+ KB
```

Prepare For Model-Based Imputation

- Perform hot-encoding on categorical data from the features containing missing values
  - Continued research on KNN, Random Forest, etc.

```
# select categorical columns
cat_cols = ['language', 'country', 'color']

# perform one-hot encoding
# country and language will still be objects even after being encoded because they are not binary like color
encoded_df = pd.get_dummies(merged_df, columns=cat_cols)


```

K- Nearest Neighbors

- Pros:
  - Simple implementation and easy to use in comparison to Random Forest
  - Can handle nonlinear decision boundaries
  - Can handle noisy data
- Cons:
  - Sensitive to irrelevant features
  - Expensive with large training set
  - Could require more time with feature scaling
    - (subtracting the mean and dividing by standard deviation)

Random Forest

- Pros:
  - Knows how to work with high dimensional data
    - Features with complex interactions
  - Slow to overfitting
  - Works with both small and large datasets
- Cons:
  - Difficult to interpret
  - Requires longer training time than KNN
  - May not perform as well on small datasets

Numerical Diagram and Bar Plot

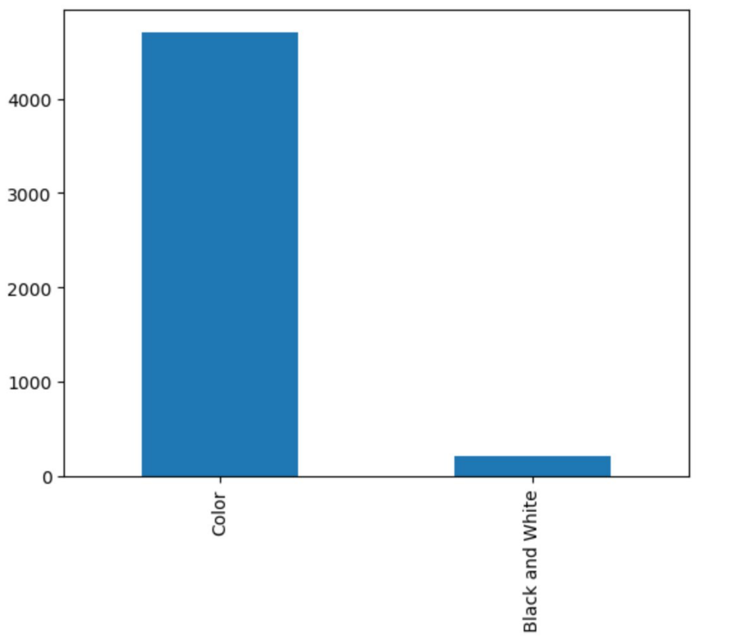
Within our 5915 movies data, the majority of the movie are colored and produced in USA

```
# Numerical data shows the amount of the original region of the flim

print(merged_df['country'].value_counts())
```

```
USA              3713
UK                434
France            154
Canada            124
Germany            94
                 ...
Slovakia            1
Chile               1
Cambodia            1
Official site       1
Philippines         1
Name: country, Length: 65, dtype: int64
```

```
# bar plot shows the numbers of difference between color movie

merged_df['color'].value_counts().plot(kind='bar')
```

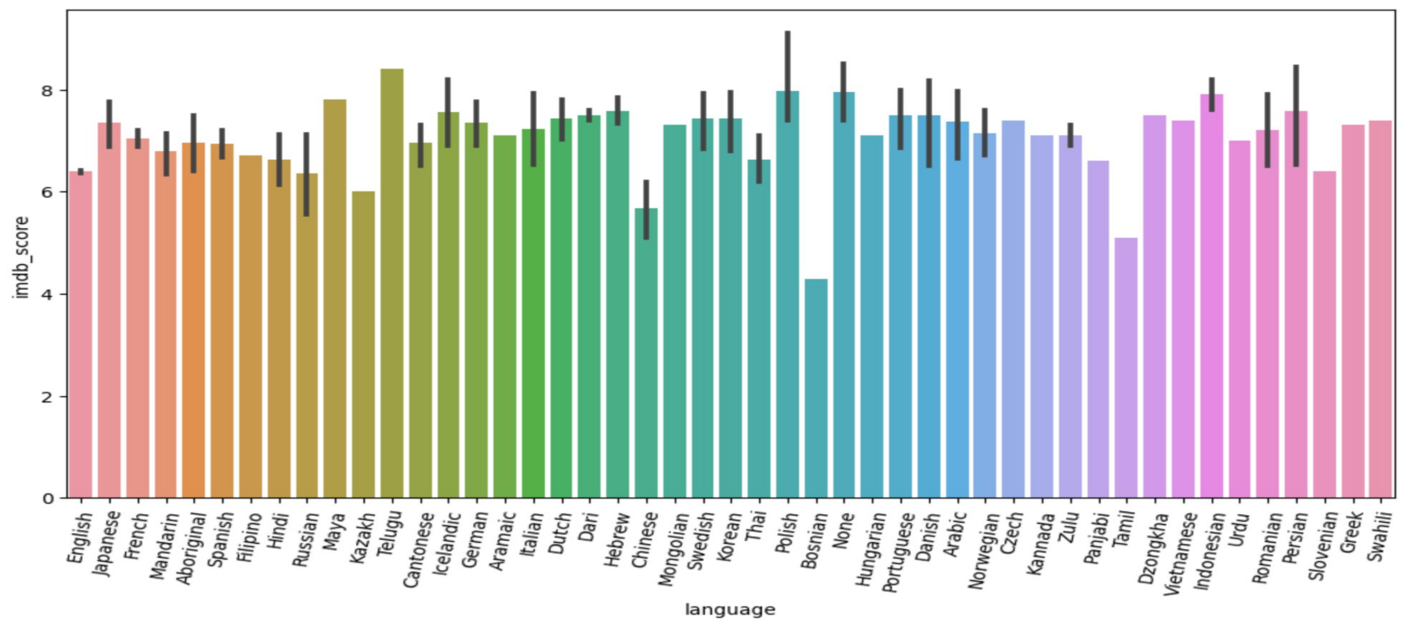
```
<AxesSubplot:>
```



Numerical Diagram and Bar Plot

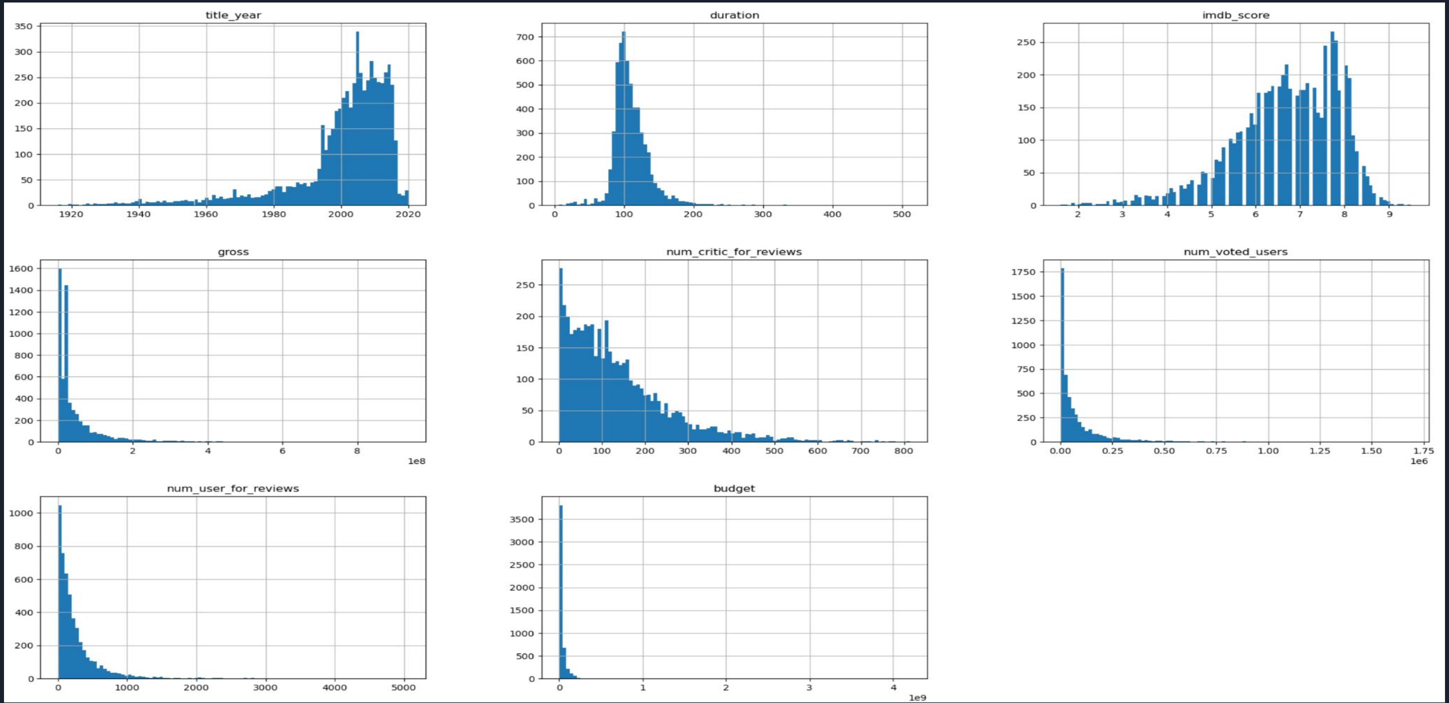
Bar plot shows the correlation for each language type of the movie and IMDB score

```
#Barplot that shows the correlation between 'language' and 'imdb_score'

plt.figure(figsize = (12, 6))
plot = sns.barplot(x='language', y='imdb_score', data= merged_df)
plt.setp(plot.get_xticklabels(),rotation=80)
plt.show()
```



Histogram and Analyzed

```
#Histogram for numerical attribute

import matplotlib.pyplot as plt
merged_df.hist(bins=100, figsize=(25,15))
plt.show()
```

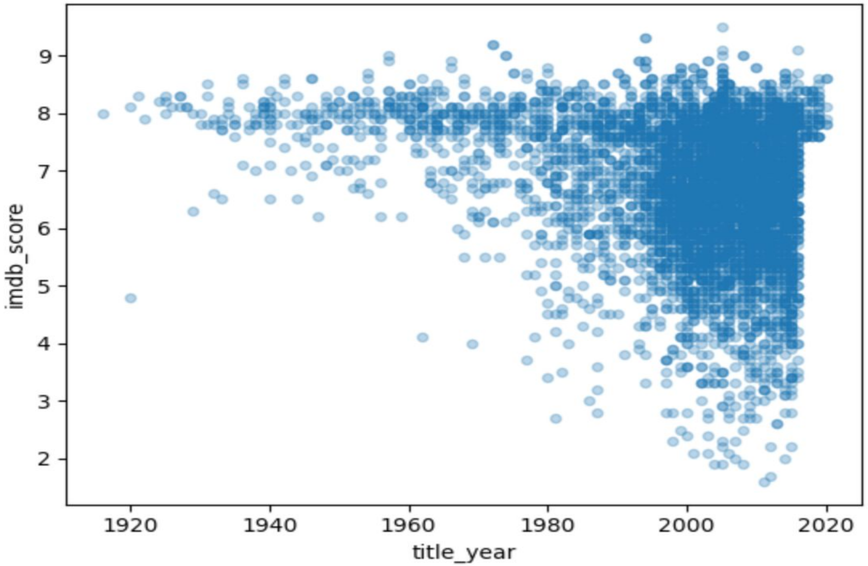


Scatter Plot and Scatter Matrix

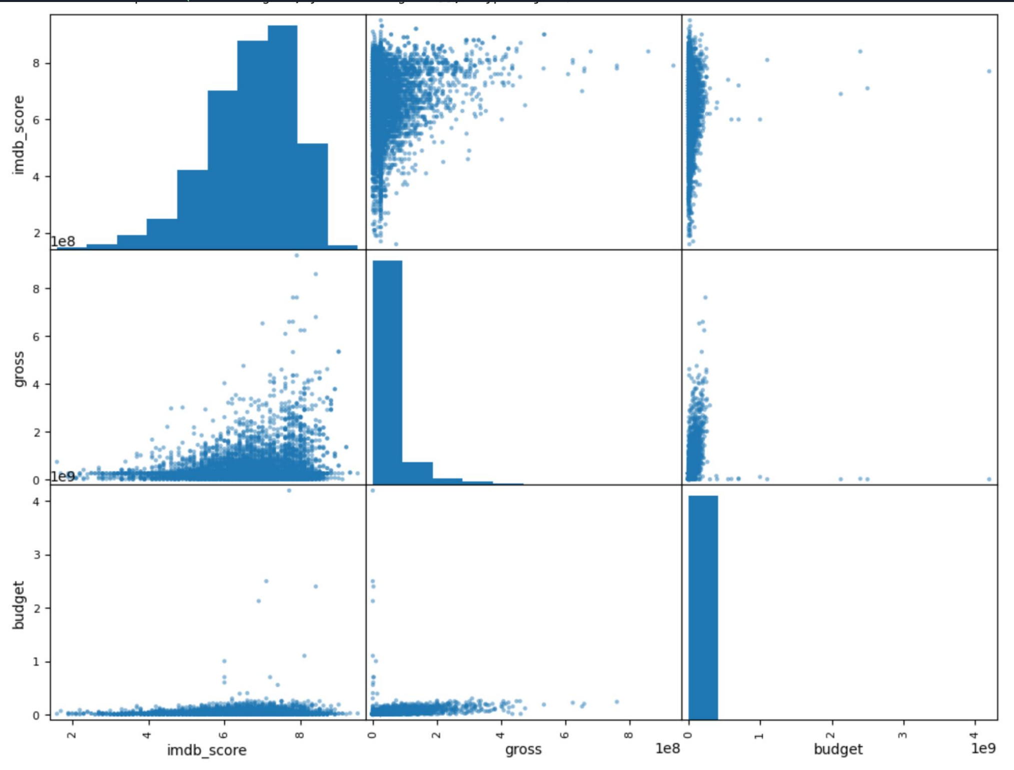
Using the scatter plot for the year of the movie and the IMDB Score

```python
#Scatterplot of the movie year distribution

merged_df.plot(kind = "scatter", x = 'title_year', y = 'imdb_score', alpha=0.3)
```

<AxesSubplot:xlabel='title_year', ylabel='imdb_score'>



Scatter Plot and Scatter Matrix



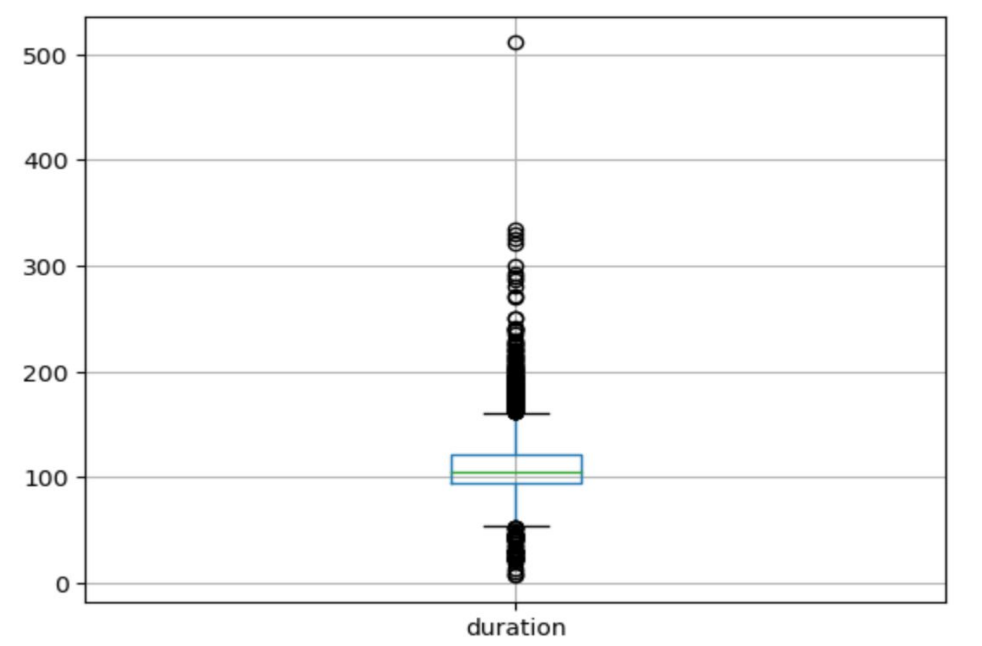
Using Scatter Matrix between the IMDB Score, gross, and the budget of the movie.

```
# Scatter matrix show the correclation between imdb_score, gross, and budget

import pandas
from pandas.plotting import scatter_matrix
attributes = ["imdb_score", "gross", "budget"]
scatter_matrix(merged_df[attributes], figsize=(12,9))
```

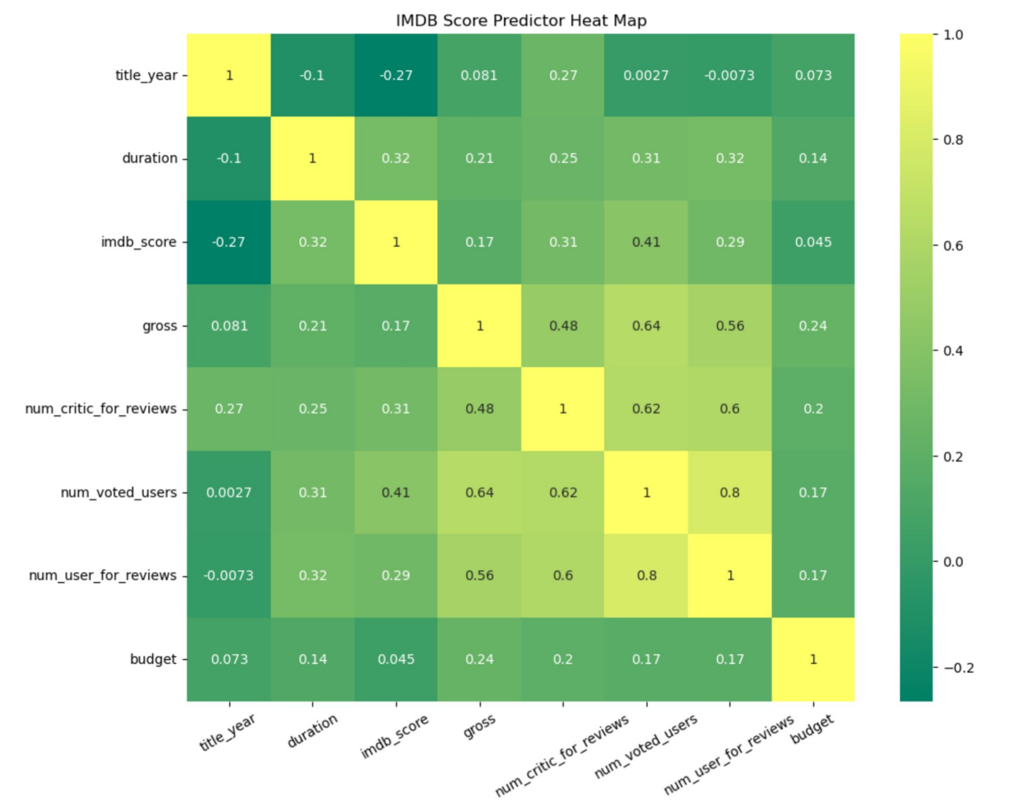
Box Plot and Heat Map

Using the box plot to highlight the outliner of the duration of the movie

```
#Boxplot to check the outliner of the duration

Duration_Ol = merged_df.boxplot(column = 'duration')
```



Box Plot and Heat Map



Using Heat Map to show the correlation between each numerical attribute and help the analysis

```
# heat map
plt.figure(figsize=(12,9))
plot = sns.heatmap(merged_df.corr(), annot=True, cmap="summer", square=True)
plt.setp(plot.get_xticklabels(),rotation=30)
plt.title("IMDB Score Predictor Heat Map")
plt.show()

# the number of the voted users, number of critic for review and duration has high positive correlation between the imdb_score
# the budgt has the lowest of the correlation between the imdb_score, and the title_year has neg. correlation between imdb_score
# title_year has low correlation between number of reviews and users
```

The End