PREDICTIVE ANALYSIS ON IMDB MOVIES AND OSCAR AWARD MOVIES DATASET

https://github.com/HilKTL/UCDPA_HILALKATAL_project

1.ABSTRACT

The primary purpose of this project is to understand how different features play a role in Oscar-winning of movies by using the IMDb database features and Oscar database features. The descriptive and exploratory analysis will be made on the data, and Logistic Regression, a supervised classification algorithm, will be applied to data to predict whether a movie will win an Oscar or not by using IMDb features.

2.INTRODUCTION

The Academy Awards are the most prestigious and crucial film awards in the American and international film industry. Because all those who voted for the awards are members of the industry1
IMDb is the world's most popular and authoritative source for movie, TV and celebrity content designed to provide audience discover the movies and shows and choose what to watch.2
Considering the popularity of the IMDb database in movie selection and the prestige of Oscar awards in the film industry, the following question comes to mind. Can we predict the Oscar-winning of a film by IMDb features? What are the IMDb features of Oscar-winning films? In this analysis, we will try to find answers to these questions.

3.DATASETS

3.1 –IMDb-dataset-of-top-1000-movies-and-tv-show:

Data was scrapped from Kaggle ,an online community of data scientists and machine learning practitioners, which is a well-known public data platform.3 The link of the data ; https://www.kaggle.com/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows.
It has 1000 movies and tv series entries with IMDb scores of 7.6 and over, from 1920 to 2020. The data includes numerical and categorical variables.

FEATURES:

ATTRIBUTE	EXPLANATION
Poster-Link	Link of the poster
Series-Title	Name of movie
Released-Year	Released year of movie
Certificate	Certificate of movie
Runtime	Total runtime
Genre	Genre of movie
IMDB-Rating	IMDb score of film
Overview	Summary of movie
Meta-score	Metacritic score of movies
Director	Director's name
No-*of*-votes	The total number of votes for movie
Star 1-2-3-4	Actors' name
Gross	Money earned by that movie

3.2-Oscar-award-dataset:

The dataset was scrapped from the Kaggle Public Data Platform. The link of the data; 4 https://www.kaggle.com/unanimad/the-oscar-award. The primary source of the data is The Academy Awards Database operated by awardsdatabase.oscars.org.
It contains past Academy Award winners and nominees between 1927 and 2019. Three columns were used in the analysis: Year-Ceremony, Winner and Film.

FEATURES:

ATTRIBUTE	EXPLANATION
Year-Ceremony	Year of ceremony
Film	Title of movie
Winner	True if movie was awarded, false if was not

3.3 –IMDb-title-basics dataset:

Data was downloaded from the IMDb database, a well-known online database containing information about movies, television series, home videos etc. The link of data: 5 https://datasets.imdbws.com/title.basics.tsv.gz
There is no title type information in the IMDb-top-1000-movie-and-tv-show dataset. Two columns of the IMDb-title-base dataset were downloaded to determine the title types of the IMDb-top-1000-movie-and-tv-show dataset because only movies can win Oscars.

FEATURES:

ATTRIBUTE	EXPLANATION
Title-Type	the type/format of the title (movie, short, tv series)
Original-Title	name of the movie

4-IMPLEMENTATION PROCESS

4.1-DATA PREPROCESSING

4.1.1 - Web Scraping and Loading:

Datasets were imported from databases via using request library and read in data frames with pandas read function. After extracting films from IMDb-dataset-of-top-1000-movies-and-tv-shows using the title-type feature of the IMDb-title-basics dataset, it was merged with Oscar Award data. Info function was used to get dataset shape, data types, and null values. A generator expression was used instead of for-loop-iteration because it is shorter and faster and there is no need to assign an empty list and use the append function to get values. Besides, it generates items only in demand, making it more memory efficient than list comprehensions and faster. 18 out of 1000 titles are not movies in IMDb dataset.

4.1.1.1-_Imdb-dataset-of-top-1000-movies-and-tv-shows:

```
#read file and assign it to variable
#give write permission to the opened file
#write the contents of the data to the file, then close the file by appyling the changes
dfa = open("imdb_top_1000.csv", "w")
dfa.write(df)
dfa.close()
```

```
import requests
file = requests.get("https://www.kaggle.com/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows/download")
print(file.status_code)
df = file.text
```

```
import pandas as pd
dfa = pd.read_csv('C:\\Users\\serta\\Desktop\\IMDB_kaggle\\imdb_top_1000.csv')
```

```
dfa.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
```

```
dfa.head()
```

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview
0	https://m.media-amazon.com /images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142 min	Drama	9.3	Two imprisoned men bond over a number of years...

Meta_score	Director	Star1	Star2	Star3	Star4	No_of_Votes	Gross
80.0	Frank Darabont	Tim Robbins	Morgan Freeman	Bob Gunton	William Sadler	2343110	28,341,469

4.1.1.2- Oscar Award dataset

```
"""Sending http request with request module"""
import requests
file = requests.get("https://www.kaggle.com/unanimad/the-oscar-award/download")
print(file.status_code)
df = file.text
```

```
"""read file and assign it to variable
give write permission to the opened file
write the contents of the data to the file, then close the file by appyling the changes"""
oscar_df = open("oscar.csv", "w")
oscar_df.write(df)
oscar_df.close()
```

```
oscar_df = pd.read_csv("C:\\Users\\serta\\Desktop\\my_data\\the_oscar_award.csv")
```

```
oscar_df.head()
```

	year_film	year_ceremony	ceremony	category	name	film	winner
0	1927	1928	1	ACTOR	Richard Barthelmess	The Noose	False
1	1927	1928	1	ACTOR	Emil Jannings	The Last Command	True

4.1.1.3 – IMDb title basics dataset

```
r = requests.get('https://datasets.imdbws.com/title.basics.tsv.gz')
with open('C:\\Users\\serta\Downloads\\title.basics.tsv.gz', 'wb') as file:
    file.write(r.content)
print(r.status_code)
```

```
df = pd.read_csv('C:\\Users\\serta\Downloads\\title.basics.tsv.gz',usecols = ['titleType','originalTitle'],delimiter="\t")
```

```
# saving title.basics.tsv.gz with two columns as tsv file
df.to_csv('C:\\Users\\serta\Downloads\\imdb_title_basics.tsv', sep="\t")
```

<u>Extracting movies from Imdb-dataset-of-top-1000-movies-and-tv-shows:</u>

```
#getting the name of the movies of dfa file with generator expression
titles = (x for x in dfa['Series_Title'])
#titles
```

```
df['titleType'].unique()
```

```
array(['short', 'movie', 'tvEpisode', 'tvSeries', 'tvShort', 'tvMovie',
       'tvMiniSeries', 'tvSpecial', 'video', 'videoGame', 'tvPilot'],
      dtype=object)
```

```
df_movies = df[df['titleType']=='movie']
df_movies.head()
```

	titleType	originalTitle
498	movie	Bohemios
570	movie	The Story of the Kelly Gang

```
#dropping duplicates names of the films in imdb title file
df_movies = df_movies[df_movies['originalTitle'].isin(titles)].drop_duplicates(subset = 'originalTitle' )
#df_movies
#getting the names of the movies of df_movies file with generator expression
titles_movies = (x for x in df_movies['originalTitle'])
```

```
#extracting title type of movies from dfa file
dfa = dfa[dfa['Series_Title'].isin(titles_movies)]
```

```
dfa.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 982 entries, 0 to 999
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Poster_Link     982 non-null    object
 1   Series_Title    982 non-null    object
```

4.2- Merging IMDb and Oscar movies dataset

Two data frames are merged on title of the movies with outer method, which takes the union of two datasets.

```
#extracting Oscar awarded movies
oscar_winner_df = oscar_df[oscar_df['winner'] == True]
df = oscar_winner_df[['year_ceremony','film','winner']]
#Changing the names of the columns of df
df.columns = ['Ceremony Year','Series_Title','win']
#sorting according to series title
df.sort_values('Series_Title').head()
#sorting according to series title
dfa = dfa.sort_values(by = 'Series_Title')
df.head(1)
left = dfa
right = df
#merging two data frames on Series title column
#merging outer method for taking union of both datasets
merged_data = pd.merge(left,right,on = 'Series_Title',how = 'outer')
merged_data.head()
```

Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_score
https://m.media-amazon.com/images/M/MV5BMTk5Mj...	(500) Days of Summer	2009	UA	95 min	Comedy, Drama, Romance	7.7	An offbeat romantic comedy about a woman who d...	76.0

Meta_score	Director	Star1	Star2	Star3	Star4	No_of_Votes	Gross	Ceremony Year	win
76.0	Marc Webb	Zooey Deschanel	Joseph Gordon-Levitt	Geoffrey Arend	Chloë Grace Moretz	472242.0	32,391,374	NaN	NaN

4.3-DATA CLEANING

4.3.1-Removing duplicates and datatype manipulation

Duplicated rows were called on names of movies, and after dropping them, missing values of the poster-link column were removed. Because titles are repeated many times according to each nomination in Oscar Award data,these rows are duplicated. The data types of Runtime and Gross variables are incorrect because these columns are numerical. The released year column’s data type was converted to an integer to analyze its relationship between ceremony years.

```
#getting insight of duplicated rows
merged_data[merged_data.duplicated()].head()
```


```
#getting insight of duplicated rows
merged_data[merged_data.duplicated()].head()
```


	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	
3	https://m.media-amazon.com/images/M/MV5BMjExMT...	12 Years a Slave	2013	A	134 min	Biography, Drama, History	8.1	U
4	https://m.media-amazon.com/images/M/MV5BMjExMT...	12 Years a Slave	2013	A	134 min	Biography, Drama, History	8.1	U
6	https://m.media-amazon.com/images/M/MV5BOTdmNT...	1917	2019	R	119 min	Drama, Thriller, War	8.3	a
7	https://m.media-amazon.com/images/M/MV5BOTdmNT...	1917	2019	R	119 min	Drama, Thriller, War	8.3	a
19	https://m.media-amazon.com/images/M/MV5BMzcwYW...	A Beautiful Mind	2001	UA	135 min	Biography, Drama	8.2	

```
#dropping duplicates
merged_data.drop_duplicates(inplace= True)
#dropping nan values of Poster Link column
merged_data.dropna(subset = ['Poster_Link'],inplace = True)
```

```
#Getting information of missing values and datatypes
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3085 entries, 0 to 3084
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Poster_Link     1431 non-null   object
 1   Series_Title    2807 non-null   object
 2   Released_Year   1431 non-null   object
 3   Certificate     1326 non-null   object
 4   Runtime         1431 non-null   object
 5   Genre           1431 non-null   object
 6   IMDB_Rating     1431 non-null   float64
 7   Overview        1431 non-null   object
 8   Meta_score      1269 non-null   float64
 9   Director        1431 non-null   object
 10  Star1           1431 non-null   object
 11  Star2           1431 non-null   object
 12  Star3           1431 non-null   object
 13  Star4           1431 non-null   object
 14  No_of_Votes     1431 non-null   float64
 15  Gross           1246 non-null   object
 16  Ceremony Year   2357 non-null   float64
 17  win             2357 non-null   object
dtypes: float64(4), object(14)
memory usage: 457.9+ KB
```

```
merged_data.describe()
```

	IMDB_Rating	Meta_score	No_of_Votes	Ceremony Year
count	1431.000000	1269.000000	1.431000e+03	2357.000000
mean	7.987911	80.134752	3.412366e+05	1976.071701
std	0.299231	11.864308	3.969824e+05	25.717570
min	7.600000	28.000000	2.508800e+04	1928.000000
25%	7.800000	73.000000	6.579900e+04	1954.000000
50%	8.000000	82.000000	1.741250e+05	1976.000000
75%	8.100000	89.000000	4.730640e+05	1998.000000
max	9.300000	100.000000	2.343110e+06	2020.000000

```
#Removing 'minutes' and changing datatype from object to integer
merged_data['Runtime'] = [int(str(i).replace("min", "")) for i in merged_data['Runtime']]

#filling nan values with 0 to be able to change the datatype from object to float,0 values will be imputed later.
merged_data['Gross'] = merged_data['Gross'].fillna(0)
merged_data['Gross'] = [float(str(i).replace(",", "")) for i in merged_data['Gross']]
merged_data['Gross'] = [np.nan if x == 0 else x for x in merged_data['Gross']]
numerical_attributes = ['Ceremony Year', 'Runtime', 'No_of_Votes']
"""there is no day or month on ceremony year column so instead of date format,it will be changed to integer,it will be dropped later"""
merged_data[numerical_attributes] = merged_data[numerical_attributes].astype('int64')
"""replacing values with list comprehension"""
merged_data['win'] = [1 if x == True else 0 for x in merged_data['win']]
merged_data.info()
#filling nan with 1900 ,it is a random year which I pick because there was no Oscar awards in that years
merged_data['Ceremony Year'].fillna(1900,inplace = True)
#filling missing win values with false ,because not awarded movies are null
merged_data['win'].fillna('False',inplace = True)
merged_data.head()
```

```
#looking for unique values for detecting anormal values
merged_data['Released_Year'].unique()
```

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

```
#finding out the series title of released year with the value of 'PG'
merged_data[merged_data['Released_Year'] == 'PG']
```

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview
103	https://m.media-amazon.com/images/M/MV5BNjEzYj...	Apollo 13	PG	U	140	Adventure, Drama, History	7.6	NASA must devise a strategy to return Apollo 1...

```
#replacing 'PG' with correct year value after finding out the correct value of released year via internet
merged_data['Released_Year'] = merged_data['Released_Year'].str.replace('PG','1995')
"Changing datatype from object to datetime"""
merged_data['Released_Year'] = pd.to_datetime(merged_data['Released_Year'])
"""Creating new column by taking year values as integers to be able to  analyze relation between ceremony year and released year"""
merged_data['Year_of_release'] = merged_data['Released_Year'].dt.year
merged_data['Year_of_release'].head(1)
```

```
#dropping the released year column
merged_data.drop('Released_Year',inplace = True,axis = 1)
```

Data after dropping duplicates and manipulating data types

```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 989 entries, 0 to 1430
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Poster_Link     989 non-null    object
 1   Series_Title    989 non-null    object
 2   Released_Year   989 non-null    object
 3   Certificate     891 non-null    object
 4   Runtime         989 non-null    int64
 5   Genre           989 non-null    object
 6   IMDB_Rating     989 non-null    float64
 7   Overview        989 non-null    object
 8   Meta_score      839 non-null    float64
 9   Director        989 non-null    object
 10  Star1           989 non-null    object
 11  Star2           989 non-null    object
 12  Star3           989 non-null    object
 13  Star4           989 non-null    object
 14  No_of_Votes     989 non-null    int64
 15  Gross           989 non-null    float64
 16  Ceremony Year   989 non-null    int64
 17  win             989 non-null    int64
dtypes: float64(3), int64(4), object(11)
```

4.3.2-Data Quality Assessment

Merged data should have 982 entries (since the IMDb dataset has 982 rows), but it has 989, so there are irrelevant seven rows in merged data that will be examined.

```
#extracting duplicated movies on series title
duplicated_titles = merged_data[merged_data.duplicated(subset='Series_Title' )]
duplicated_titles
```

Series_Title	Certificate	Runtime	Genre	IMDB_Rating	Overview	No_of_Votes	Gross	Ceremony Year	win	Year_of_release
A Star Is Born	UA	136	Drama, Music, Romance	7.6	A musician helps a young singer find fame as a...	334312	215288866.0	1977	1	2018
A Star Is Born	UA	136	Drama, Music, Romance	7.6	A musician helps a young singer find fame as a...	334312	215288866.0	2019	1	2018
Drishyam	U	160	Crime, Drama, Thriller	8.3	A man goes to extreme lengths to save his fami...	30722	0.0	1900	0	2013
King Kong	Passed	100	Adventure, Horror, Sci-Fi	7.9	A film crew goes to a tropical island for an e...	78991	10000000.0	2006	1	1933
Little Women	U	135	Drama, Romance	7.8	Jo March reflects back and forth on her life, ...	143250	108101214.0	1950	1	2019
Little Women	U	135	Drama, Romance	7.8	Jo March reflects back and forth on her life, ...	143250	108101214.0	2020	1	2019
Titanic	UA	194	Drama, Romance	7.8	A seventeen-year-old aristocrat falls in love ...	1046089	659325379.0	1998	1	1997
Up	U	96	Animation, Adventure, Comedy	8.2	78-year-old Carl Fredricksen travels to Paradi...	935507	293004164.0	2010	1	2009

```
merged_data[merged_data['Series_Title'].isin(['Titanic','Little Women','A Star Is Born','Drishyam','King Kong','Up'])].iloc[:,[1,15,16,17]]
```

	Series_Title	Ceremony Year	win	Year_of_release
35	A Star Is Born	1938	1	2018
36	A Star Is Born	1977	1	2018
37	A Star Is Born	2019	1	2018
331	Drishyam	1900	0	2015
332	Drishyam	1900	0	2013
593	King Kong	1977	1	1933
594	King Kong	2006	1	1933
676	Little Women	1933	1	2019
677	Little Women	1950	1	2019
678	Little Women	2020	1	2019
1317	Titanic	1954	1	1997
1318	Titanic	1998	1	1997
1368	Up	1985	1	2009
1369	Up	2010	1	2009

```python
#calling duplicated movies from imdb movies dataset
dfa[dfa['Series_Title'].isin(['Titanic','Little Women','A Star Is Born','Drishyam','King Kong','Up'])]
```

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview
903	https://m.media-amazon.com /images/M/MV5BNmE5Zm...	A Star Is Born	2018	UA	136 min	Drama, Music, Romance	7.6	A musician helps a young singer find fame as a...
136	https://m.media-amazon.com /images/M/MV5BYmJhZm...	Drishyam	2015	UA	163 min	Crime, Drama, Mystery	8.2	Desperate measures are taken by a man who trie...
87	https://m.media-amazon.com /images/M/MV5BYmY3Mz...	Drishyam	2013	U	160 min	Crime, Drama, Thriller	8.3	A man goes to extreme lengths to save his fami...
566	https://m.media-amazon.com /images/M/MV5BZTY3Yj...	King Kong	1933	Passed	100 min	Adventure, Horror, Sci-Fi	7.9	A film crew goes to a tropical island for an e...
585	https://m.media-amazon.com /images/M/MV5BY2QzYT...	Little Women	2019	U	135 min	Drama, Romance	7.8	Jo March reflects back and forth on her life, ...
652	https://m.media-amazon.com /images/M/MV5BMDdmZG...	Titanic	1997	UA	194 min	Drama, Romance	7.8	A seventeen-year-old aristocrat falls in love ...
146	https://m.media-amazon.com /images/M/MV5BMTk3ND...	Up	2009	U	96 min	Animation, Adventure, Comedy	8.2	78-year-old Carl Fredricksen travels to Paradi...

The Ceremony-year of the awarded movie cannot be smaller than the year of release, so the index of 35, 36, 676, 677,1317,1368 was removed from the data. There is one King Kong movie in the IMDb dataset, and it was awarded in 2006, so the 593rd row was removed.

```python
#duplicated rows and films that are not in imdb movies will be drop from merged data(Oscar+imdb movies data)
merged_data.drop(index = [35,36,593,676,677,1317,1368],inplace = True)
#Ceremony year of awarded movie cannot be smaller than year.
```

Since first Oscar award was given to 1927 and 1928 films and the films in the Oscar data are up to 2019, years smaller than 1927 and bigger than 2019 were deleted from merged data.

```python
#getting information of released year range of Oscar df
print(oscar_df['year_film'].nsmallest(2),oscar_df['year_film'].nlargest(2))
```

```
0     1927
1     1927
Name: year_film, dtype: int64 10267    2019
10268    2019
Name: year_film, dtype: int64
```

```python
#getting information of released year range of merged data
print(merged_data['Year_of_release'].nsmallest(2),merged_data['Year_of_release'].nlargest(2))
```

```
281     1920
1150    1921
Name: Year_of_release, dtype: int64 308    2020
334    2020
Name: Year_of_release, dtype: int64
```

```python
index_of_inappropriate_dates = merged_data[merged_data['Year_of_release'].isin([1920,1921,1922,1923,1924,1925,1926,2020])].index
index_of_inappropriate_dates
merged_data = merged_data.drop(index=index_of_inappropriate_dates, axis=0)
```

4.3.3-Handling missing data

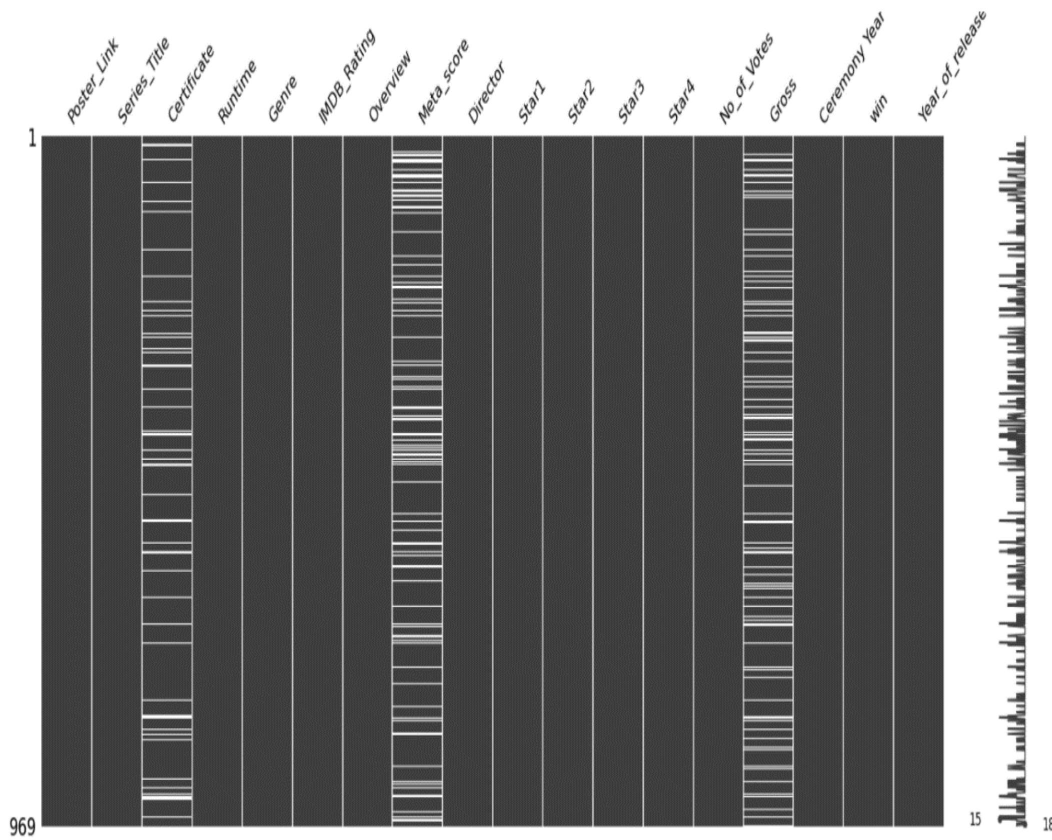
Around %40 of data is missing in gross, meta-score, and certificate columns.

```
merged_data.isna().sum()
```

```
Poster_Link          0
Series_Title         0
Certificate         94
Runtime              0
Genre                0
IMDB_Rating          0
Overview             0
Meta_score         142
Director             0
Star1                0
Star2                0
Star3                0
Star4                0
No_of_Votes          0
Gross              154
Ceremony Year        0
win                  0
Year_of_release      0
dtype: int64
```

```
#calculating percentage of missing values
def missing_values_percentage(df):
    return sum(df.isna().sum())/len(df.values)*100
print(missing_values_percentage(merged_data))
```

```
40.24767801857585
```



4.3.3.1-Imputation of Gross column:

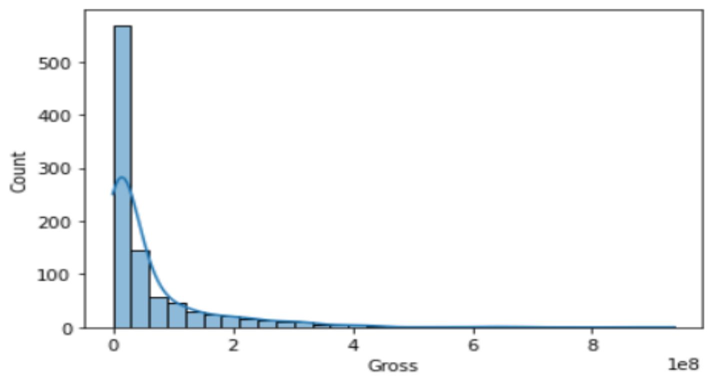
Since the gross column distribution is highly skewed and there is a high correlation between gross and number-of-votes column, imputation was made according to the number-of-votes column with linear regression. SciPy library lingress method was used to get a linear equation between two columns. Then, with the help of slope and intercept results of linear equation, we can determine the null gross values that fits on line by putting corresponding vote values in the equation. After imputation median decreased from 2.447542e+07 to 2.153785e+07 and correlation between two columns increased from 0.57 to 0.59.

```
#Finding out the correlation between gross column and other numerical columns
columns = ['IMDB_Rating','Meta_score','No_of_Votes','Gross']
subset = merged_data[columns]
subset.corr()
#according to results,there is a high correlation between gross and number of votes column
```

	IMDB_Rating	Meta_score	No_of_Votes	Gross
IMDB_Rating	1.000000	0.274106	0.509862	0.105226
Meta_score	0.274106	1.000000	-0.013446	-0.029833
No_of_Votes	0.509862	-0.013446	1.000000	0.574909
Gross	0.105226	-0.029833	0.574909	1.000000

```
#Distribution of non null gross values
not_null_gross = merged_data[~merged_data['Gross'].isnull()]['Gross']
#plotting hisplot and assign bin numbers to square root of the entries
bins = round(merged_data.shape[0]**0.5)
sns.histplot(not_null_gross,kde = True,bins = bins)
plt.xlabel('Gross')
title = print('Distribution of Gross')
plt.title(title)
plt.show()
```

```
Distribution of Gross
```

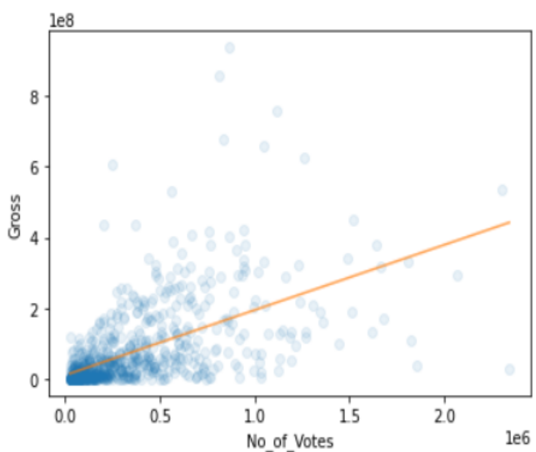



```
merged_data['Gross'].describe()
```

```
count    8.150000e+02
mean     6.902748e+07
std      1.104879e+08
min      1.305000e+03
25%      3.333484e+06
50%      2.447542e+07
75%      8.302631e+07
max      9.366622e+08
Name: Gross, dtype: float64
```

<u>Plotting best fit line on the data points of meta-scores and IMDb-scores before imputation:</u>

```
#Scatterplot between no_of_votes and gross
plt.plot(xi,yi,'o',alpha = 0.1)

#plot the best linear fit line on scatter plot
#creating numpy array including min to max values of number of votes
x = np.array([xi.min(),xi.max()])
#creating linear equation according to linear regression slope and intercept values
y = results.intercept+results.slope*x
#plotting the best fit line
plt.plot(x,y,'-',alpha = 0.7)
plt.xlabel('No_of_Votes')
plt.ylabel('Gross')
plt.show()
```



Getting linear equation:

```
#extracting data in which gross isnot null
data = merged_data[~merged_data['Gross'].isnull()]

#getting linear regression equation between gross and no_o_votes column
from scipy.stats import linregress

xi = data['No_of_Votes']
yi = data['Gross']


# Compute the linear regression
results = linregress(xi,yi)
print(results)
```

```
LinregressResult(slope=184.16528980872417, intercept=10661783.398254469,
```

According to results, gross increases 184.16 points per a vote.

Imputation according to linear regression results:

```
#getting index number of null gross rows
gross_null = merged_data[merged_data['Gross'].isnull()]['Gross']
#creating list of index numbers
gross_index_lst = gross_null.index
```

```
#extracting null values of gross
gross_scores_null = merged_data.loc[gross_index_lst]['Gross']
#extracting no of votes values of null gross values
votes_null = merged_data.loc[gross_index_lst]['No_of_Votes']
```

```
#calculating gross values according to intercept and slope
gross_scores_null = round((results.intercept) + ((results.slope)*votes_null))
```

```
#imputating null gross rows
merged_data.loc[gross_index_lst,'Gross'] = gross_scores_null
```

```
After imputaion gross values:
```

```
merged_data.loc[gross_index_lst].head()
```

Pearson coefficients and visualization before and after imputation:

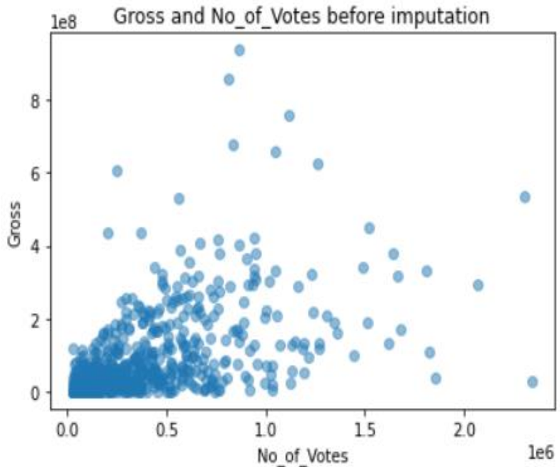
Before

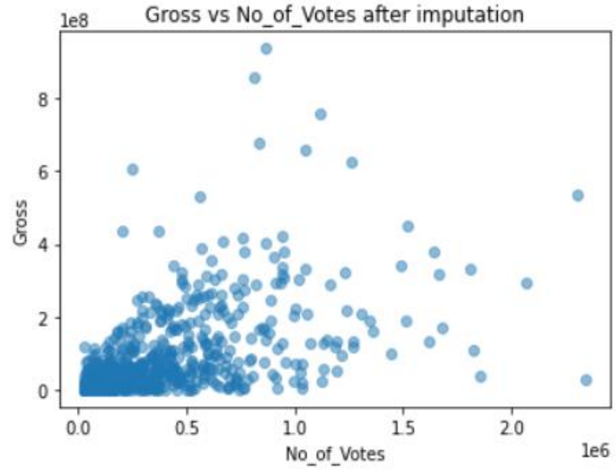
	IMDB_Rating	Meta_score	No_of_Votes	Gross
IMDB_Rating	1.000000	0.274106	0.509862	0.105226
Meta_score	0.274106	1.000000	-0.013446	-0.029833
No_of_Votes	0.509862	-0.013446	1.000000	0.574909
Gross	0.105226	-0.029833	0.574909	1.000000

After

	No_of_Votes	Gross
No_of_Votes	1.000000	0.591154
Gross	0.591154	1.000000

```
#Group by numbers of votes
grouped = merged_data.groupby('No_of_Votes')
#Getting mean values of gross
mean_gross_by_votes = grouped['Gross'].mean()
#plotting of mean values of gross vs number of votes
plt.plot(mean_gross_by_votes,'o',alpha = 0.5)
plt.xlabel('No_of_Votes')
plt.title('Gross and No_of_Votes before imputation' )
plt.ylabel('Gross')
plt.show()
```





After imputation, correlation between two columns increased by 0.016 points.

4.3.3.2-Imputation of Certificate Column

Imputation was made according to the genre column with a function that takes the mode value of the certificate in a particular genres group. Since both genre and certificate columns are strings, 'contains function' was used with regex expressions for imputation. ^ was used to determine beginning and $ was used to determine end of genres group. A negative look ahead was used to extract only Drama genre without matching groups.

```
merged_data.head(2)
```

	Poster_Link	Series_Title	Certificate	Runtime	Genre
0	https://m.media-amazon.com/images/M/MV5BMTk5Mj...	(500) Days of Summer	UA	95	Comedy, Drama, Romance
1	https://m.media-amazon.com/images/M/MV5BMWU4N2...	12 Angry Men	U	96	Crime, Drama

```
#Defining a function for certificate values imputation

import re

mask2 = merged_data['Certificate'].isnull()

def certificate_imp(reg):

    """Finding mode value of certificate according to genres group"""

    mask1 = merged_data['Genre'].str.contains(reg)

    #creating dictionary to take the mode value of certificate values
    x = dict(merged_data[mask1].iloc[:,2].value_counts())
    #print(list(x.keys())[0])

    result = list(x.keys())[0]

    return result
```

<u>Count of genre groups where certificate values are missing:</u>

```
merged_data[merged_data['Certificate'].isnull()]['Genre'].value_counts()
```

Drama	12	Animation, Adventure, Family	1
Comedy, Drama	7	Drama, Fantasy, Mystery	1
Drama, War	5	Drama, Western	1
Action, Crime, Drama	4	Film-Noir, Mystery	1
Mystery, Thriller	3	Comedy, Crime	1
Drama, Thriller	3	Action, Adventure, War	1
Drama, Romance	3	Biography, Drama, History	1
Crime, Drama, Thriller	3	Crime, Mystery, Thriller	1
Action, Drama, Thriller	3	Drama, Family	1
Crime, Drama	3	Action, Drama, Mystery	1
Crime, Drama, Mystery	3	Action, Adventure, Drama	1
Drama, Horror, Thriller	2	Action, Adventure, Crime	1
Comedy, Romance	2	Comedy, Musical, War	1
Crime, Drama, Film-Noir	2	Horror, Thriller	1
Comedy, Drama, War	2	Comedy, Drama, Family	1
Drama, Music, Romance	2	Drama, Sci-Fi	1
Comedy, Drama, Romance	2	Crime, Drama, Horror	1
Drama, Horror	2	Drama, History	1
Crime, Drama, Romance	2	Action, Crime, Comedy	1
Drama, Fantasy	1	Adventure, Drama	1
Drama, Horror, Sci-Fi	1	Drama, Film-Noir, Mystery	1
Comedy, Crime, Thriller	1	Action, Adventure, Biography	1
		Action, Crime, Thriller	1
		Crime, Drama, Fantasy	1
		Adventure, Comedy, Sci-Fi	1
		Thriller	1

Some codes of imputation:

```
mask2 = merged_data['Certificate'].isnull()
```

```
#extracting only drama genres without any genre (negative loohahead assertion)
mask1 =merged_data['Genre'].str.contains(r"^Drama(?!,)")
#visualize drama films
merged_data[mask1].head()
```

```
#certificate imputation according to genre
merged_data.loc[mask1&mask2,'Certificate'] = certificate_imp(r"^Drama(?!,)")
#(?!,) ---> negative loohahead assertion (), '^' ----> Starts with
mask1 =merged_data['Genre'].str.contains(r"(Action, Crime, Drama)")
merged_data[mask1].head()
merged_data.loc[mask1&mask2,'Certificate'] = certificate_imp(r"(Action, Crime, Drama)")
```

```
mask1 =merged_data['Genre'].str.contains(r"(^Drama, Thriller$)")
merged_data[mask1].head()
merged_data.loc[mask1&mask2,'Certificate'] = certificate_imp(r"(^Drama, Thriller$)")
```

After imputation, certificates were put in groups by creating a dictionary(to assign a key to a value) .6

```
#Creating certificate groups by mapping
mapping = {'G': 'All_ages_group','U' : 'All_ages_group' , 'PG':'All ages(kids with PG)','TV-PG':'All ages(kids with PG)',
          'UA':'All ages(kids with PG)','U/A':'All ages(kids with PG)','GP':'All ages(kids with PG)',
          'Passed' : 'All ages(kids with PG)','PG-13' : 'Teens', 'TV-14': 'Teens','16' : 'Teens','R' : 'Adult' ,'TV-MA' : 'Adult','A' : 'Adult'}
merged_data['Certificate']= merged_data['Certificate'].replace(mapping)
```

```
merged_data['Certificate'].value_counts()
```

```
Adult                     404
All_ages_group            256
All ages(kids with PG)    254
Teens                      43
Approved                   11
Unrated                     1
Name: Certificate, dtype: int64
```

```
merged_data['Certificate'].isnull().sum()
```

```
0
```

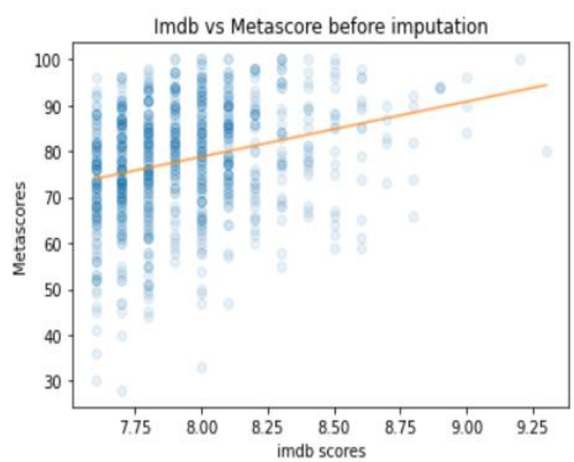
4.3.3.4 -Meta-score column imputation

There is a weak correlation between meta-score and IMDB ratings. Therefore, we could use multiple regression, but there is nearly no correlation between the meta-score and other columns. So, imputation was made with linear regression. After imputation, correlation increased by 0.013 points and the median value remained same.

Imputation according to linear regression results:

```
#extracting data in which metascores isnot null
data = merged_data[~merged_data['Meta_score'].isnull()]

#getting linear regression equation between metascore and imdb scores
from scipy.stats import linregress

xi = data['IMDB_Rating']
yi = data['Meta_score']


# Compute the linear regression
results = linregress(xi,yi)
print(results)
```

```
LinregressResult(slope=12.005067620755545, intercept=-17.269931628675778
```




```
#getting index number of null metascores rows
metascore_null = merged_data[merged_data['Meta_score'].isnull()]['Meta_score']
#creating list of index numbers
metascore_index_lst = metascore_null.index
```

```
#extracting null values of metascores
meta_scores_null = merged_data.loc[metascore_index_lst]['Meta_score']
#extracting null values of imdb scores
IMDB_Rating_null = merged_data.loc[metascore_index_lst]['IMDB_Rating']
```

```
#calculating metascores values according to linear equation
meta_scores_null = round((results.intercept) + ((results.slope)*IMDB_Rating_null))
```

```
#imputating null metascores rows
merged_data.loc[metascore_index_lst,'Meta_score'] = meta_scores_null
```

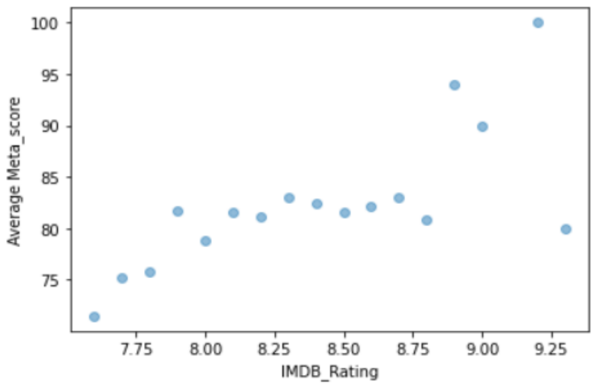
Pearson correlation coefficients and average-meta-score values vs IMDb-ratings visualization before and after imputation:

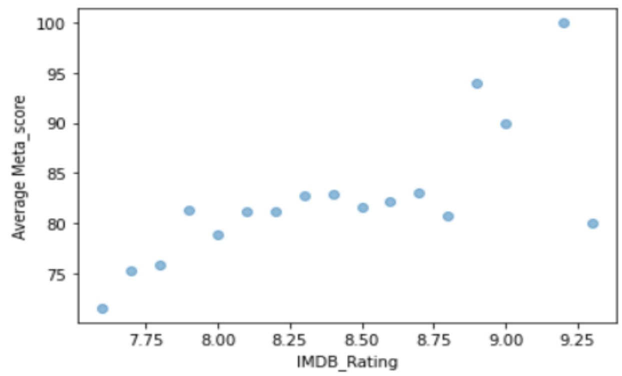
BEFORE

	IMDB_Rating	Meta_score
IMDB_Rating	1.000000	0.274106
Meta_score	0.274106	1.000000
No_of_Votes	0.509862	-0.013446
Gross	0.099605	-0.044344

AFTER

	IMDB_Rating	Meta_score
IMDB_Rating	1.000000	0.287654
Meta_score	0.287654	1.000000





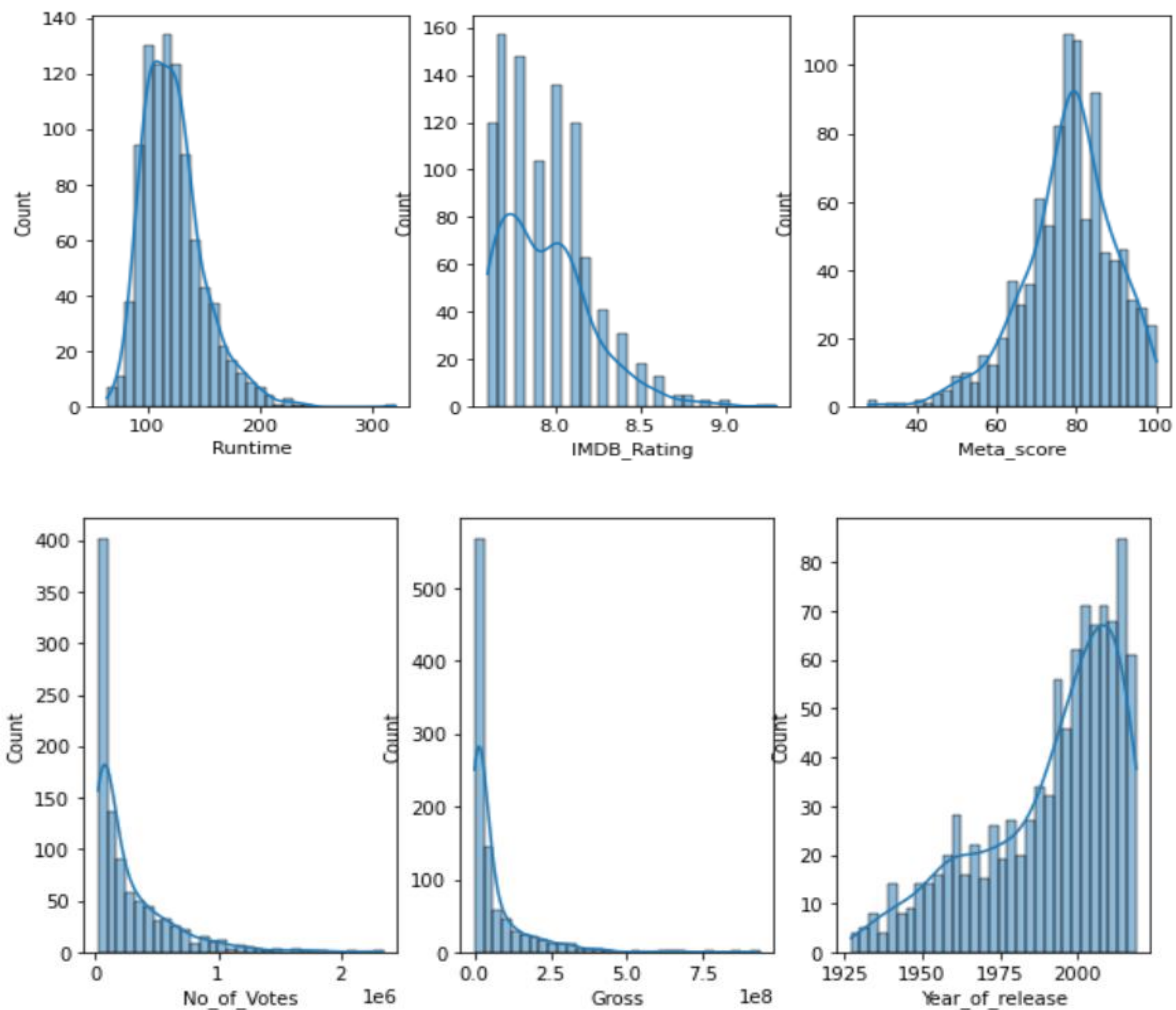
After imputation and data type manipulation, there are non-null 969 entries and 18 columns in the data.

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Poster_Link     969 non-null    object
 1   Series_Title    969 non-null    object
 2   Certificate     969 non-null    object
 3   Runtime         969 non-null    int64
 4   Genre           969 non-null    object
 5   IMDB_Rating     969 non-null    float64
 6   Overview        969 non-null    object
 7   Meta_score      969 non-null    float64
 8   Director        969 non-null    object
 9   Star1           969 non-null    object
 10  Star2           969 non-null    object
 11  Star3           969 non-null    object
 12  Star4           969 non-null    object
 13  No_of_Votes     969 non-null    int64
 14  Gross           969 non-null    float64
 15  Ceremony Year   969 non-null    int64
 16  win             969 non-null    int64
 17  Year_of_release 969 non-null    int64
```

4.3.4-DATA RANGE ASSESSMENT

Distribution of numerical values were plotted with seaborn library hisplot function. Descriptive statistical results were determined by describe and unique categorical values were observed with pandas unique function. There are no irrelevant values in the data.

```
import seaborn as sns
numerical_columns = ['Runtime','IMDB_Rating','Meta_score','No_of_Votes','Gross','Year_of_release']
for x in numerical_columns:
    sns.histplot(merged_data[x],kde = True,bins = round(merged_data.shape[0]**0.5))
    plt.xlabel(x)
    title = print('Distribution of',x)
    plt.title(title)
    plt.show()
```



```
#Descriptive analytics
merged_data[numerical_columns].describe()
```

	Runtime	IMDB_Rating	Meta_score	No_of_Votes	Gross	Year_of_release
count	969.000000	969.000000	969.000000	9.690000e+02	9.690000e+02	969.000000
mean	123.147575	7.944995	78.112487	2.773749e+05	6.095474e+07	1991.578947
std	27.842345	0.274837	11.488529	3.299243e+05	1.031012e+08	22.484573
min	64.000000	7.600000	28.000000	2.508800e+04	1.305000e+03	1927.000000
25%	103.000000	7.700000	72.000000	5.662500e+04	5.009677e+06	1978.000000
50%	119.000000	7.900000	79.000000	1.415160e+05	1.809570e+07	1999.000000
75%	137.000000	8.100000	86.000000	3.778840e+05	6.625700e+07	2009.000000
max	321.000000	9.300000	100.000000	2.343110e+06	9.366622e+08	2019.000000

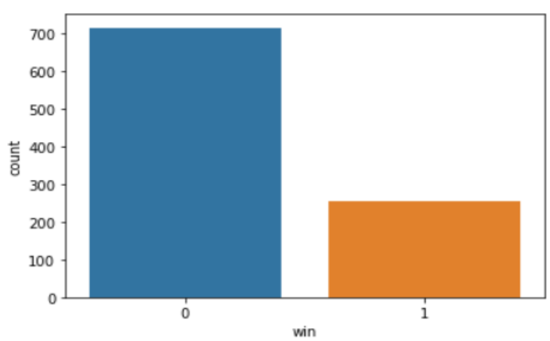
```
merged_data['Certificate'].unique()
```

```
array(['All ages(kids with PG)', 'All_ages_group', 'Adult', 'Approved',
       'Teens', 'Unrated'], dtype=object)
```

Data is not in balance. The number of unawarded movies are nearly three times more than awarded films.

```
"""Determining data balance (according to plot we see that data is not in balance
because number of awarded movies are by far bigger than not awarded ones"""
sns.countplot(data = merged_data,x = 'win')
```

```
<AxesSubplot:xlabel='win', ylabel='count'>
```



6-PREDICTIVE ANALYSIS

In the data, independent variables contain multinominal and continuous values, and the target variable is labelled and is binary. Therefore, Logistic regression, a supervised learning classification model, was used as a predictive model. In logistic regression, we fit the curve by using maximum likelihood estimation calculated by the probability of each observed sample and multiplication of these likelihoods provides the likelihood of the best fit line.

6.1- Feature Engineering:

Resetting index and dropping unnecessary columns:

```
#changing index as titles of movies
merged_data = merged_data.set_index('Series_Title')
#dropping columns which will be not used in predictive analysis
merged_data.drop(columns = ['Overview','Ceremony Year','Poster_Link','Year_of_release'],axis =1,inplace = True)
```

6.1.1-One-Hot encoding on categorical features

Since the machine learning algorithms require numerical values, categorical variables were encoded. The categorical values are nominal and not ordinal, so the pandas get dummies function was used instead of the label encoder.7

```
Genre = merged_data['Genre']
Genre = Genre.str.get_dummies()
Certificate = merged_data['Certificate']
Certificate = Certificate.str.get_dummies()
Star1 = merged_data['Star1']
Star1 = Star1.str.get_dummies()
Star2 = merged_data['Star2']
Star2 = Star2.str.get_dummies()
Star3 = merged_data['Star3']
Star3 = Star3.str.get_dummies()
Star4 = merged_data['Star4']
Star4 = Star4.str.get_dummies()
Director = merged_data['Director']
Director = Director.str.get_dummies()


merged_data_encoded = pd.concat(
    [merged_data.drop(
        ['Genre','Certificate','Star4','Director','Star1','Star2','Star3'],
        axis=1
    ),
     Genre,Certificate,Star1,Star2,Star3,Star4,Director],

    axis=1,

)
merged_data_encoded.head()
```

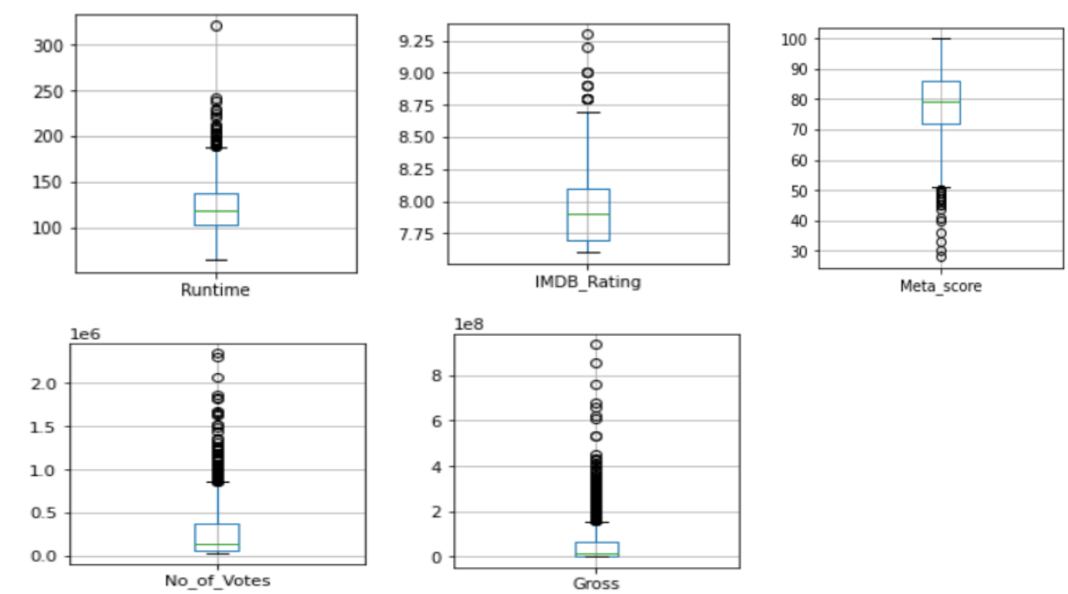
	Runtime	IMDB_Rating	Meta_score	No_of_Votes	Gross	win	Action, Adventure	Action, Adventure, Biography	Action, Adventure, Comedy	Action, Adventure, Crime	...	Yilmaz Erdogan	Yimou Zhang	Yoshiaki Kawajiri	Yoshifumi Kondô	Yôjirô Takita	Zack Snyder
Series_Title																	
(500) Days of Summer	95	7.7	76.0	472242	32391374.0	0	0	0	0	0	...	0	0	0	0	0	0
12 Angry Men	96	9.0	96.0	689845	4360000.0	0	0	0	0	0	...	0	0	0	0	0	0
12 Years a Slave	134	8.1	96.0	640533	56671993.0	1	0	0	0	0	...	0	0	0	0	0	0
1917	119	8.3	78.0	425844	159227644.0	1	0	0	0	0	...	0	0	0	0	0	0
2001: A Space Odyssey	149	8.3	84.0	603517	56954992.0	1	0	0	0	0	...	0	0	0	0	0	0

5 rows × 3974 columns

6.1.2-OUTLIERS

The outlier value is the value that is over 1.5 times the interquartile, which is the difference between the third quartile and first quartile. The outliers can affect statistical results and model assumptions. Therefore, before scaling, outliers were examined. 25.39% of the data have outlier values.

```
"""PLOTTING OUTLIERS"""
for x in ['Runtime','IMDB_Rating','Meta_score','No_of_Votes','Gross']:
    boxplot = merged_data_encoded.boxplot(column=[x],figsize =(3,3))
    plt.show()
```



Defining a function to find upper and lower outliers:

```
def outliers_percentage(x):
    """finding upper and lower outlier values"""
    #first quantile
    q1 = merged_data_encoded[x].quantile(0.25)
    #third quantile
    q3 = merged_data_encoded[x].quantile(0.75)
    #interquar
    Interquar=q3-q1
    #print(q1)
    #print(q3)
    #print(Interquar)
    #lower outlier value(1.5 times interquar lower than first quantile)
    Lower_outlier_value = q1-(1.5*Interquar)
    #Upper outlier value(1.5 times interquar upper than third quantile)
    Upper_outlier_value = q3+(1.5*Interquar)
    #percentage of outlier values
    percentage = round((len(merged_data_encoded[merged_data_encoded[x] > Upper_outlier_value]) +
                        len(merged_data_encoded[merged_data_encoded[x] < Lower_outlier_value]))/merged_data_encoded.shape[0]*100,2)
    return percentage
```

```
outliers_percentage('No_of_Votes')
```

6.71

Calculating total percentage of outliers:

```python
def sum_per_outliers(lst):
    """calculating total percentage of outliers"""
    #define empty list for percentage of outliers
    percentage_lst = []
    for x in lst:
        #getting percentage of outliers
        percentage = outliers_percentage(x)
        #collecting percentage for each column
        percentage_lst.append(percentage)
        #calculating total percentage of outliers with numpy array sum function
    return np.sum(percentage_lst)
```

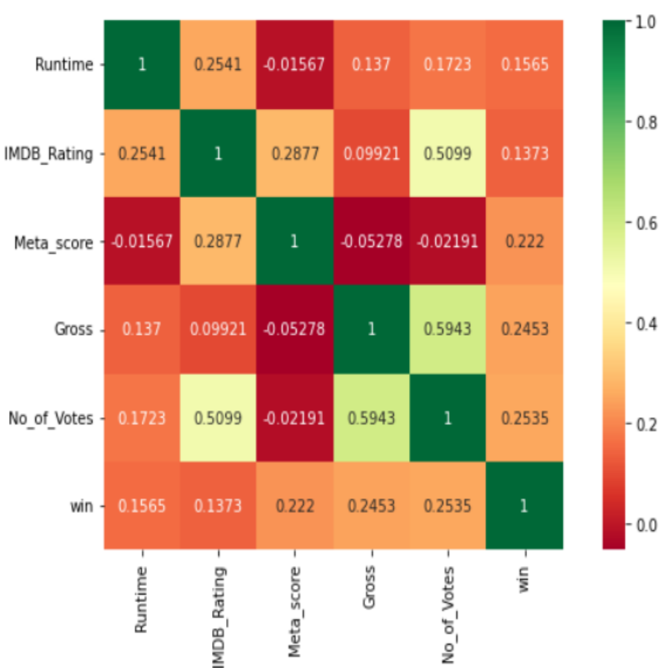
```python
#total percentage of outliers of all numerical columns
sum_per_outliers(['Runtime','IMDB_Rating','Meta_score','No_of_Votes','Gross'])
```

25.39

6.1.3- Determining high correlated features

```python
#determining list of numerical columns to scale
columns_to_scale  = ['Runtime', 'IMDB_Rating','Meta_score', 'Gross', 'No_of_Votes','win']
#getting pearson correlation coefficient between the columns
cor = merged_data_encoded[columns_to_scale].corr(method = 'pearson')
#plotting heatmap with correlation values
plt.figure(figsize = (10,6))
sns.heatmap(cor, annot = True ,square=True, cmap='RdYlGn',fmt='.4g')
```

<AxesSubplot:>



The-number-of-votes column has high correlation with Gross and IMDb columns. It was deleted to avoid multicollinearity which could produce overfitting problem on predictive model.

6.1.4-Standardization

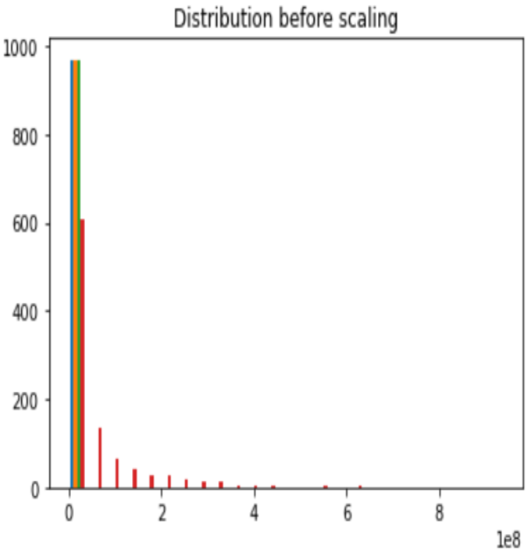
The gross column has large values, dominating other numerical columns. Since most numerical columns were distributed highly skewed and logistic regression assumes the variables to have Gaussian (normal distribution), standardization is made by robust scaling, which scales the data according to the interquartile range because the outlier's ratio is too high to remove (%25 of the data).8 After scaling, although the distribution is not skewed, the outliers remain in the data. Model accuracy increased by %2.4 after scaling.
Since the data is not in balance, Repeated-Stratified-KFold cross-validation was used to have the same ratio between target classes in each fold as in the entire dataset. Besides, target variable was split to train and test data with stratified method because of balance problem. The pipeline was used for applying scaling and model respectively into the validation process.

<u>Getting accuracy scores of different quantiles ranges with cross validation:</u>

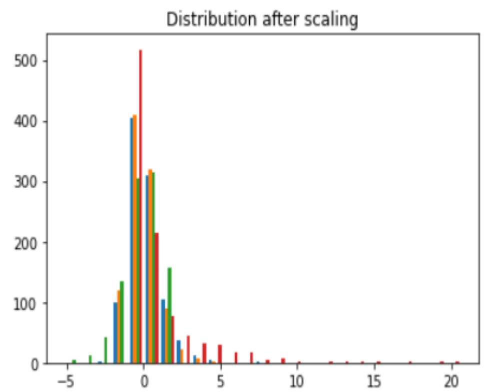
```
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
#determining independent variables
columns_to_scale  = merged_data_encoded[['Runtime', 'IMDB_Rating','Meta_score', 'Gross']]
#determining target variable
win = merged_data_encoded['win']
#looping in quantile ranges

for ranges in ((1.0, 99.0),(2.0,98.0),(5.0, 95.0),(10.0, 90.0),(15.0, 85.0),(25.0, 75.0),(30,70),(35,65)):
    #instantiate Robust Scaler
    scaler = RobustScaler(with_centering=True,
    with_scaling=True,
    quantile_range= ranges,copy=True)
    #instantiate model
    model =LogisticRegression()
    #initiate pipeline for two steps(first scaling will be done,then model fitting)
    pipeline = Pipeline(steps=[('scaler', scaler), ('model', model)])
    ##instantiate cross validation
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    #getting cross validation scores (scoring is accuracy because dependent variable is binary)
    scores = cross_val_score(pipeline, columns_to_scale, win, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
    #rounding scores numbers to 4 digit
    scores = round(np.mean(scores),4)

    print('For quantile range:',ranges,'average score is:',scores)
```

```
For quantile range: (1.0, 99.0) average score is: 0.7479
For quantile range: (2.0, 98.0) average score is: 0.7499
For quantile range: (5.0, 95.0) average score is: 0.7517
For quantile range: (10.0, 90.0) average score is: 0.753
For quantile range: (15.0, 85.0) average score is: 0.7534
For quantile range: (25.0, 75.0) average score is: 0.7551
For quantile range: (30, 70) average score is: 0.7554
For quantile range: (35, 65) average score is: 0.7554
```

<u>Robust scaling:</u>




```python
from matplotlib import pyplot
columns_to_scale  = ['Runtime', 'IMDB_Rating','Meta_score', 'Gross']
scaler = RobustScaler(with_centering=True,
    with_scaling=True,
    quantile_range=(30.0, 70.0),copy=True)
merged_data_encoded[columns_to_scale] = scaler.fit_transform(merged_data_encoded[columns_to_scale])
merged_data_encoded[columns_to_scale].head()
# histogram of the transformed data
pyplot.hist(merged_data_encoded[columns_to_scale], bins=25)
pyplot.title("Distribution after scaling")
pyplot.show()
```



In cross validation we divide data into n folds. Each time, one fold of n folds is held out for testing and the rest are for training. So we can take average of cross validation scores to get more accurate evaluation scores.

```python
"""Defining cross validation model accuracy function"""
from sklearn import model_selection
from sklearn. linear_model import LogisticRegression
def models_accuracy_scores(model,independent,dependent):
    model = model
    #create crossvalidation with 10 splits and 10 repeats with RepeatedstratifiedKfold
    cv = RepeatedStratifiedKFold(n_splits=10, random_state=0)
    #getting validation scores(scoring is accuracy because dependent variable is binary and classification analysis will be applied)
    scores = model_selection.cross_val_score(model, independent, dependent, cv=cv, scoring='accuracy')
    #rounding average score to 3 decimal places
    average_accuracy = round(scores.mean(),3)
    #printing the results
    print('Average accuracy score of',model,'is:',average_accuracy)
```

Accuracy score before scaling

```
models_accuracy_scores(LogisticRegression(),X,y)

Average accuracy score of LogisticRegression() is: 0.738
```

Accuracy score after scaling

```
models_accuracy_scores(LogisticRegression(),X,y)

Average accuracy score of LogisticRegression() is: 0.762
```

6.1.5-Feature Selection

Irrelevant features can decrease the model's performance and increase overfitting (variance) caused by the noise of less redundant data. L1 penalty forces the least related coefficients with the response variable to zero and provides a sparsity solution. However, in our data number of features is higher than the number of samples, and dual formulation can only be applied to l2 regularization, which adds a penalty equal to the square of the magnitude of coefficients. To reduce the dimensionality of the data, a Logistic Regression classifier (with L2 penalty) was used along with a Select-From-Model meta-transformer that chooses attributes according to importance weights. After feature reduction, model accuracy increased to %80. The number of features decreased from 3972 to 487.[9]

Getting avg. accuracy scores with 10-fold cross-validation to select best C-index (inverse of regularization index) :

```
X = merged_data_encoded.drop(columns = ['win'],axis =1)
y = merged_data_encoded['win']
from sklearn.feature_selection import SelectFromModel

#determine 15 c-index values between 0.0001 and 1000
c_ = np.logspace(-4,3,15)
#looping in c-index
for c in c_:
    #instantiate logistic regression
    logreg = LogisticRegression(C=c,penalty = 'l2',solver = 'liblinear',dual = True,max_iter = 100000).fit(X, y)
    #instantiate select from model
    selector = SelectFromModel(logreg, prefit=True)
    #model.get_support()
    X = selector.transform(X)
    #getting validation scores
    scores = model_selection.cross_val_score(logreg, X, y, cv=10, scoring='accuracy')
    avg_score = round(np.mean(scores),4)
    print("for",c,"avg_score is",avg_score)
```

```
for 0.0001 avg_score is 0.7441
for 0.00031622776601683794 avg_score is 0.7472
for 0.001 avg_score is 0.7482
for 0.0031622776601683794 avg_score is 0.7358
for 0.01 avg_score is 0.7317
for 0.03162277660168379 avg_score is 0.7368
for 0.1 avg_score is 0.7368
for 0.31622776601683794 avg_score is 0.7368
for 1.0 avg_score is 0.7348
for 3.1622776601683795 avg_score is 0.7337
for 10.0 avg_score is 0.7337
for 31.622776601683793 avg_score is 0.7337
for 100.0 avg_score is 0.7337
for 316.22776601683796 avg_score is 0.7337
for 1000.0 avg_score is 0.7337
```

```
X = merged_data_encoded.drop(columns = ['win'],axis =1)
y = merged_data_encoded['win']


from sklearn.feature_selection import SelectFromModel

#instantiate model
logreg = LogisticRegression(C= 0.001, penalty = 'l2',solver = 'liblinear').fit(X, y)
#instantiate select from model
selector = SelectFromModel(estimator = logreg, prefit=True,importance_getter = "coef_")
#scale data
X = selector.transform(X)
X.shape
```

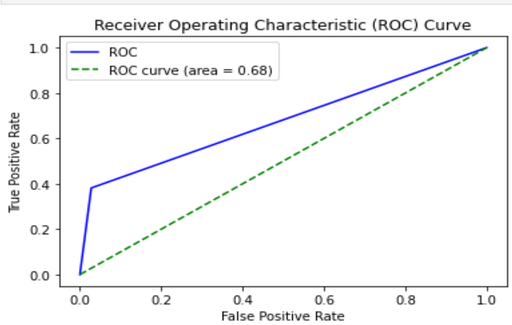
```
(969, 487)
```

Model accuracy and roc curve plotting after feature selection

```
models_accuracy_scores(LogisticRegression(),X,y)
```

```
Average accuracy score of LogisticRegression() is: 0.799
```

```
roc_after_feature_selection = roc_curve_plot(LogisticRegression(),X,y)
roc_after_feature_selection
```



Most correlated 20 columns

```
#assigning columns names to coef_df
coef_df.columns = columns
#getting the first 20 largest coefficient values with column names
coef_df.iloc[0].nlargest(20)
```

```
Meta_score                    0.094753
Gross                         0.033115
Runtime                       0.031331
IMDB_Rating                   0.013589
Dustin Hoffman                0.002753
Biography, Drama, Music       0.002324
Tom Hanks                     0.002186
Biography, Drama, History     0.002150
Sam Mendes                    0.002116
Drama, Romance, War           0.002087
Kate Winslet                  0.002075
Billy Wilder                  0.001862
Paul Newman                   0.001706
Steven Spielberg              0.001706
Audrey Hepburn                0.001685
Norman Jewison                0.001685
Elia Kazan                    0.001640
Joaquin Phoenix               0.001619
Harrison Ford                 0.001619
Katharine Hepburn             0.001613
Name: 0, dtype: float64
```

7-MODEL APPLY

7.1- Hyperparameter Tuning:

Grid Search was used to generate a model for each specified combination of hyperparameters and select the best-performing ones with cross-validation.

<u>*Best Performing Hyperparameters:*</u>

OPTIMIZATION-ALGORITHM	REGULARIZATION-PENALTY	CONCORDANCE STATISTICS	ACCURACY	TUNED HYPERPARAMETERS
SAGA	None	51.795	81.44	C-index (15 different values from 0.01 to 100) Penalty (L1, L2, None, Elastic-net)
NEWTON_CG	L2	7.197	83.16	C-index (15 different values from 0.01 to 100) Penalty (L2, None)
LBFGS	L2	7.197	83.16	C-index (15 different values from 0.01 to 100) Penalty (L2, None)
LIBLINEAR	L2	3.73	83.50	C-index (15 different values from 0.01 to 100) Penalty (L1, L2)

Before tuning, cross-validation was applied to the model with the default hyperparameters to get the train and test accuracy to determine if there is over-or under-fitting after tuning.

```
cross_val_eval(LogisticRegression(),X,y)
```

```
for k =  1
train_score is :  0.8727064220183486 and test score is :  0.711340206185567
-----------------------------------------------------
for k =  2
train_score is :  0.8612385321100917 and test score is :  0.8041237113402062
-----------------------------------------------------
for k =  3
train_score is :  0.8715596330275229 and test score is :  0.7938144329896907
-----------------------------------------------------
for k =  4
train_score is :  0.8669724770642202 and test score is :  0.865979381443299
-----------------------------------------------------
for k =  5
train_score is :  0.8681192660550459 and test score is :  0.8041237113402062
-----------------------------------------------------
for k =  6
train_score is :  0.8658256880733946 and test score is :  0.7835051546391752
-----------------------------------------------------
for k =  7
train_score is :  0.8669724770642202 and test score is :  0.8144329896907216
-----------------------------------------------------
for k =  8
train_score is :  0.8635321100917431 and test score is :  0.7628865979381443
-----------------------------------------------------
for k =  9
train_score is :  0.8658256880733946 and test score is :  0.845360824742268
-----------------------------------------------------
for k =  10
train_score is :  0.8671248568155785 and test score is :  0.71875
-----------------------------------------------------
Average train score is :   0.866987715039356
Average test score is :   0.7904317010309279
```

<u>Tuning:</u>

```
warnings. simplefilter(action='ignore', category=UserWarning)
# Create the classifier: logreg
logreg_lib = LogisticRegression(max_iter = 10000)
# Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=0,stratify = y)
# Create the hyperparameter grid
#creating 15 c-index between 0.01 to 100
c_space = np.logspace(-2, 2, 15)
param_grid = {'C': c_space}
param_grid['solver'] = ['liblinear']
param_grid['penalty'] = ['l1','l2']
#define k-fold cross validation evaluation with 9 folds (RepeatedStratifiedKFold - classification)
cv = RepeatedStratifiedKFold(n_splits =9,n_repeats = 3,random_state = 0)
#creating Gridsearch for determining the best parameters of logistic regression with 9 fold cross validation
logreg_lib_cv = GridSearchCV(logreg_lib,param_grid,cv=cv,scoring = 'accuracy')
#fit logr. with train data
logreg_lib_cv.fit(X_train,y_train)
#making predictions with test data
y_pred = logreg_lib_cv.predict(X_test)
#print accuracy
accuracy = logreg_lib_cv.score(X_test, y_test)
print("Accuracy of ",param_grid,"is :",accuracy)
#print classification report
print(classification_report(y_test, y_pred))
#print best parameters determined by gridsearch
logreg_lib_cv.best_params_
```

```
Accuracy of  {'C': array([1.00000000e-02, 1.93069773e-02, 3.72759372e-02, 7.19685673e-02,
       1.38949549e-01, 2.68269580e-01, 5.17947468e-01, 1.00000000e+00,
       1.93069773e+00, 3.72759372e+00, 7.19685673e+00, 1.38949549e+01,
       2.68269580e+01, 5.17947468e+01, 1.00000000e+02]), 'solver': ['liblinear'], 'penalty': ['l1', 'l2']} is : 0.8350515463917526
              precision    recall  f1-score   support

           0       0.85      0.95      0.89       215
           1       0.78      0.51      0.62        76

    accuracy                           0.84       291
   macro avg       0.81      0.73      0.76       291
weighted avg       0.83      0.84      0.82       291
```

```
{'C': 3.727593720314938, 'penalty': 'l2', 'solver': 'liblinear'}
```

Model apply:

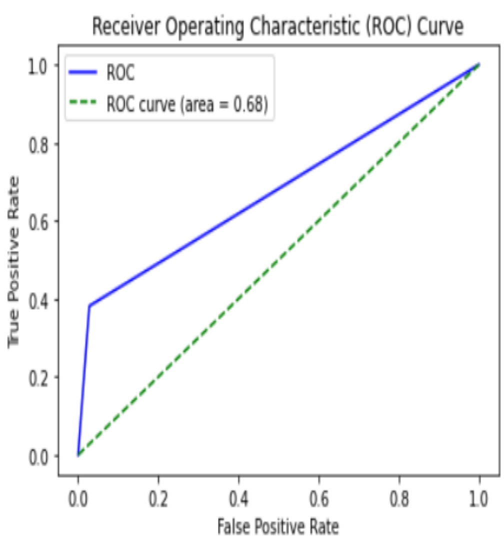
```
logreg = LogisticRegression(C = 3.73, penalty = 'l2',solver = 'liblinear',random_state = 0)

# #split data into train-test data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3,random_state =0,stratify = y)
# Fit it to the training data
logreg.fit(X_train,y_train)
#make prediction
y_pred = logreg.predict(X_test)
#print accuracy of train and test data
print("Training accuracy: {}".format(logreg.score(X_train, y_train)))
print("Testing accuracy : {}" .format(accuracy_score(y_pred,y_test)))
#print classification report
print(classification_report(y_test, y_pred))
```

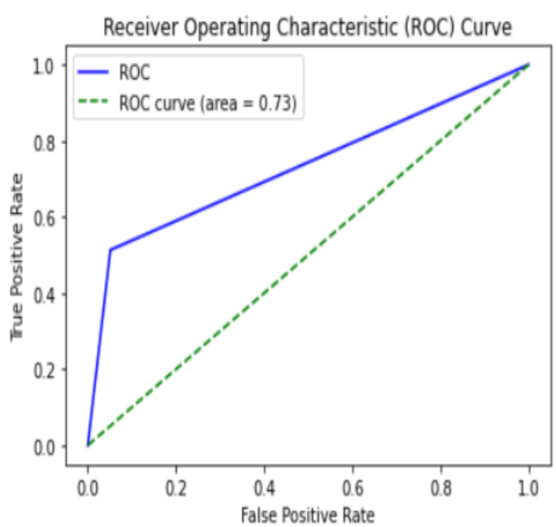
```
Training accuracy: 0.9026548672566371
Testing accuracy : 0.8350515463917526
              precision    recall  f1-score   support

           0       0.85      0.95      0.89       215
           1       0.78      0.51      0.62        76

    accuracy                           0.84       291
   macro avg       0.81      0.73      0.76       291
weighted avg       0.83      0.84      0.82       291
```

7.2-RESULTS

After tuning, both test and train accuracy scores increased, meaning that the model performs well and there is no over or under fitting after tuning. AUC (Area under curve) represents the degree or measure of separability.

Before	**After**
Average train score is : 0.866987715039356	Training accuracy: 0.9026548672566371
Average test score is : 0.7904317010309279	Testing accuracy : 0.8350515463917526





Since the model predicts Oscar-winning movies, we value the false-negative and true-positive predictions (positive class), so recall will be used as an evaluation score. The recall score is 84.65%.

```
# Confusion Matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

```
array([[204,  11],
       [ 37,  39]], dtype=int64)
```

```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

```
Recall = true_positive/(true_positive+false_negative)
Recall
```

```
0.8464730290456431
```

```
"""False negative predicted films"""
false_nega_titles
```

```
Index(['Chinatown', 'Little Women',
       'Birdman or (The Unexpected Virtue of Ignorance)',
       'The Lives of Others', 'Once', 'The Red Shoes', 'Bohemian Rhapsody',
       'The Last Emperor', 'The Fighter', 'Bonnie and Clyde',
       'Lost in Translation', 'The Curious Case of Benjamin Button',
       'East of Eden', 'My Cousin Vinny', 'Inception', 'Jojo Rabbit',
       'As Good as It Gets', 'Ex Machina', 'The Usual Suspects', 'Aladdin',
       'Rocky', 'No Man's Land', 'Scent of a Woman', 'Dog Day Afternoon',
       'Mary Poppins', 'True Grit', 'Rosemary's Baby', 'Ed Wood', 'Hamlet',
       'Ran', 'Charade', 'Arrival', 'Marriage Story', 'Life of Pi',
       'Mad Max: Fury Road', 'The Sound of Music', 'On Golden Pond'],
      dtype='object', name='Series_Title')
```

8-INSIGHTS

8.1 –According to descriptive statistics, we can hypothesize that awarded and unawarded films have almost similar IMDb scores. Is it true or not?

```
#group data by win column
merged_data.groupby('win')['IMDB_Rating'].describe()
```

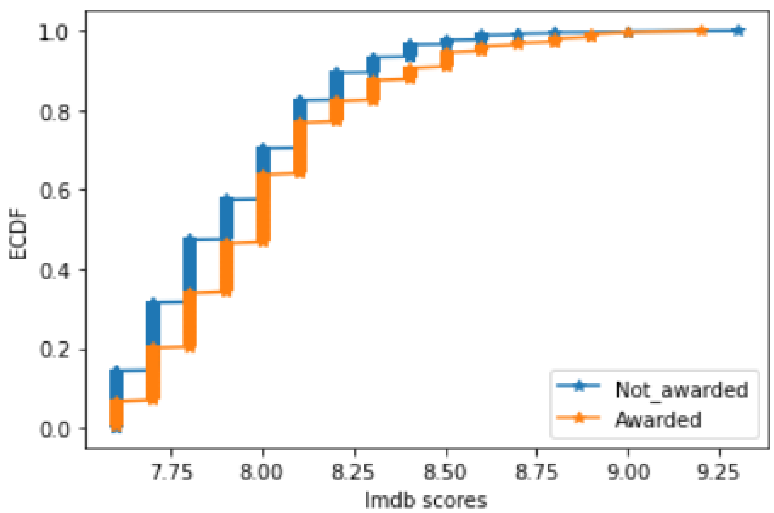
win	count	mean	std	min	25%	50%	75%	max
0	715.0	7.922517	0.262522	7.6	7.7	7.9	8.1	9.3
1	254.0	8.008268	0.298432	7.6	7.8	8.0	8.1	9.2

```
#oscar awarded data
awarded_data = merged_data[merged_data['win']==1]
#not awarded data
not_oscar_awarded_data = merged_data[merged_data['win']==0]
#imdb scores of awarded movies
imdb_oscar = awarded_data['IMDB_Rating']
#imdb scores of not awarded movies
imdb_not_oscar = not_oscar_awarded_data['IMDB_Rating']
```

Defining empirical cumulative density function:

```python
def ecdf(df):
    """Compute ECDF for a one-dimensional array of measurements."""
    # Number of data points
    length = len(df)

    # sorting x array: x
    x = np.sort(df)

    # y-data for the ECDF: y
    y = np.arange(1, length+1) / length

    return x, y
```

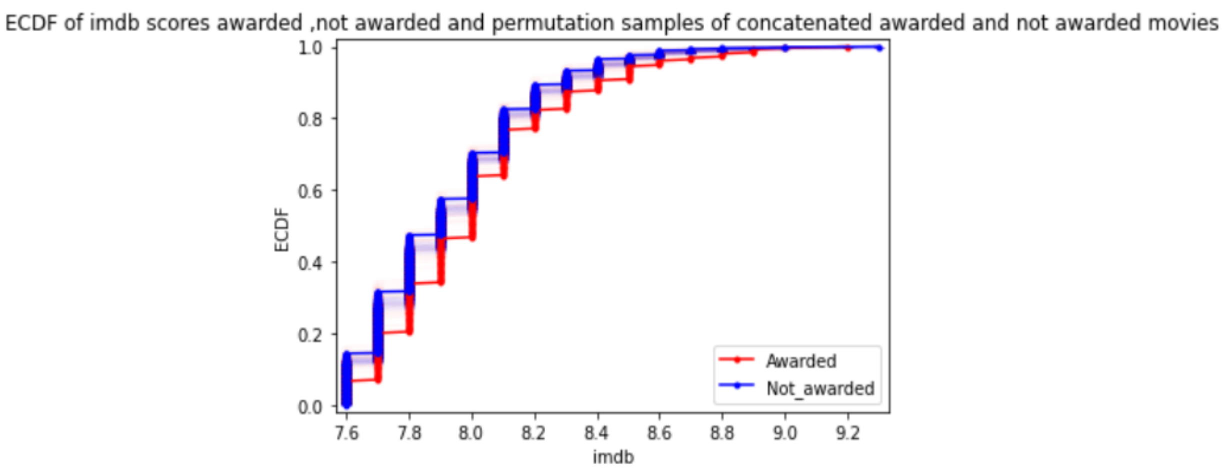
```python
"""Plotting ecdf of imdb scores of awarded and not awarded data"""
x,y = ecdf(imdb_not_oscar)
x_oscar,y_oscar = ecdf(imdb_oscar)
_ = plt.plot(x,y,marker = '*',linestyle = '-',label = 'Not_awarded')
_ = plt.plot(x_oscar,y_oscar,marker = '*',linestyle = '-',label = 'Awarded')

plt.legend(loc = 'lower right')
_ = plt.xlabel('Imdb scores')
_ = plt.ylabel('ECDF')
plt.show()
```



Awarded movies have higher scores than unawarded films. Unawarded and awarded films were concatenated and permutation samples of IMDb scores were created to investigate if they would overlap with the observed data.

```python
"""Creating permutation samples for 50 times"""
for _ in range(50):


    # Concatenate two datasets with numpy concatanenate function
    df =np.concatenate((imdb_oscar,imdb_not_oscar))

    # Permute the concatenated array: permuted_data
    permuted_df = np.random.permutation(df)

    # Split the permuted array into two: perm_sample_1, perm_sample_2
    perm_sample_awarded = permuted_df[:len(imdb_oscar)]
    perm_sample_not_awarded = permuted_df[len(imdb_oscar):]


    # Compute ECDFs
    x_awarded, y_awarded = ecdf(perm_sample_awarded)
    x_not_awarded, y_not_awarded = ecdf(perm_sample_not_awarded)

    # Plot ECDFs of permutation samples
    _ = plt.plot(x_awarded, y_awarded, marker='_',
                 color='red', alpha=0.01)
    _ = plt.plot(x_not_awarded, y_not_awarded, marker='_',
                 color='blue', alpha=0.01)



# Create and plot ECDFs from merged data(original data)
x_org_awarded, y_org_awarded = ecdf(imdb_oscar)
x_not_org_awarded, y_not_org_awarded = ecdf(imdb_not_oscar)
_ = plt.plot(x_org_awarded, y_org_awarded, marker='.', color='red',label = 'Awarded')
_ = plt.plot(x_not_org_awarded, y_not_org_awarded, marker='.', color='blue',label = 'Not_awarded')
```

```python
# Label axes, set margin, and show plot
plt.margins(0.02)
_ = plt.xlabel('imdb')
_ = plt.ylabel('ECDF')
plt.legend(loc = 'lower right')
plt.title('ECDF of imdb scores awarded ,not awarded and permutation samples of concatenated awarded and not awarded movies')
plt.show()
```



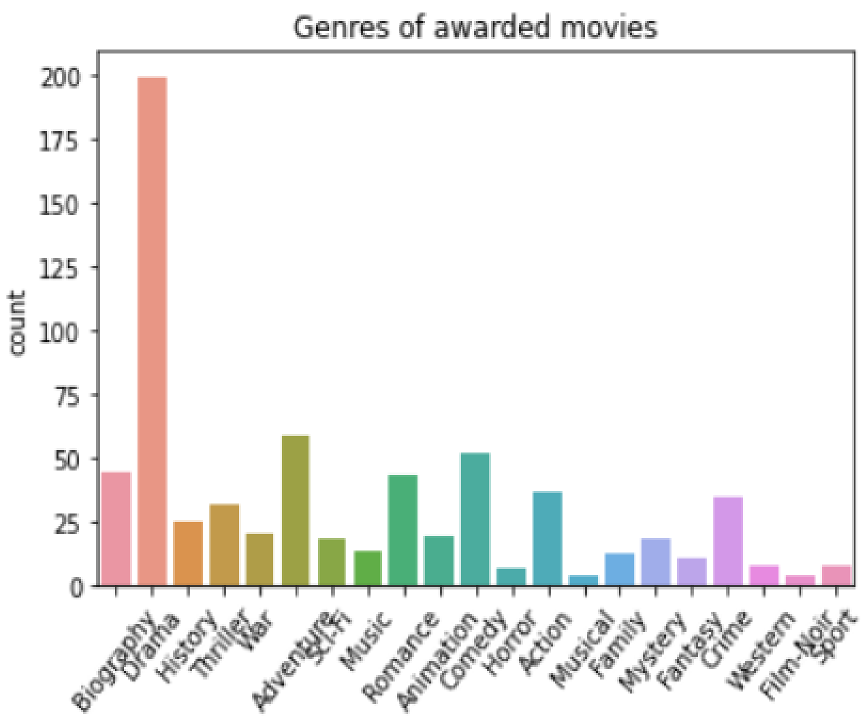
According to the graphic, none of the permutation samples overlaps on observed data. They remain between ecdf line of awarded and unawarded films till 8.8, showing that scores are not identically distributed between two datasets. Awarded movies have higher scores.

2-Dramas are the most awarded genres.

Getting dictionary of each genre counts:

```python
from collections import Counter

"""Counter is a sub-class to count hashable objects
as key: value pairs (as a dictionary)"""

# extract awarded movies' genres
dtseries = awarded_data["Genre"]
#print(dtseries)
#creating empty list to collect genres
counts_lst = []
for entry in dtseries:
    #print(entry)
    #extract each genre from groups of genres
    a = entry.split(",")

    #print(a)
    for genre in a:
        #print(genre)
        #strip white space
        genre = genre.strip()
        #print(genre)
        counts_lst.append(genre)
#getting dictionary of genres and counts
Counter(counts_lst)
```

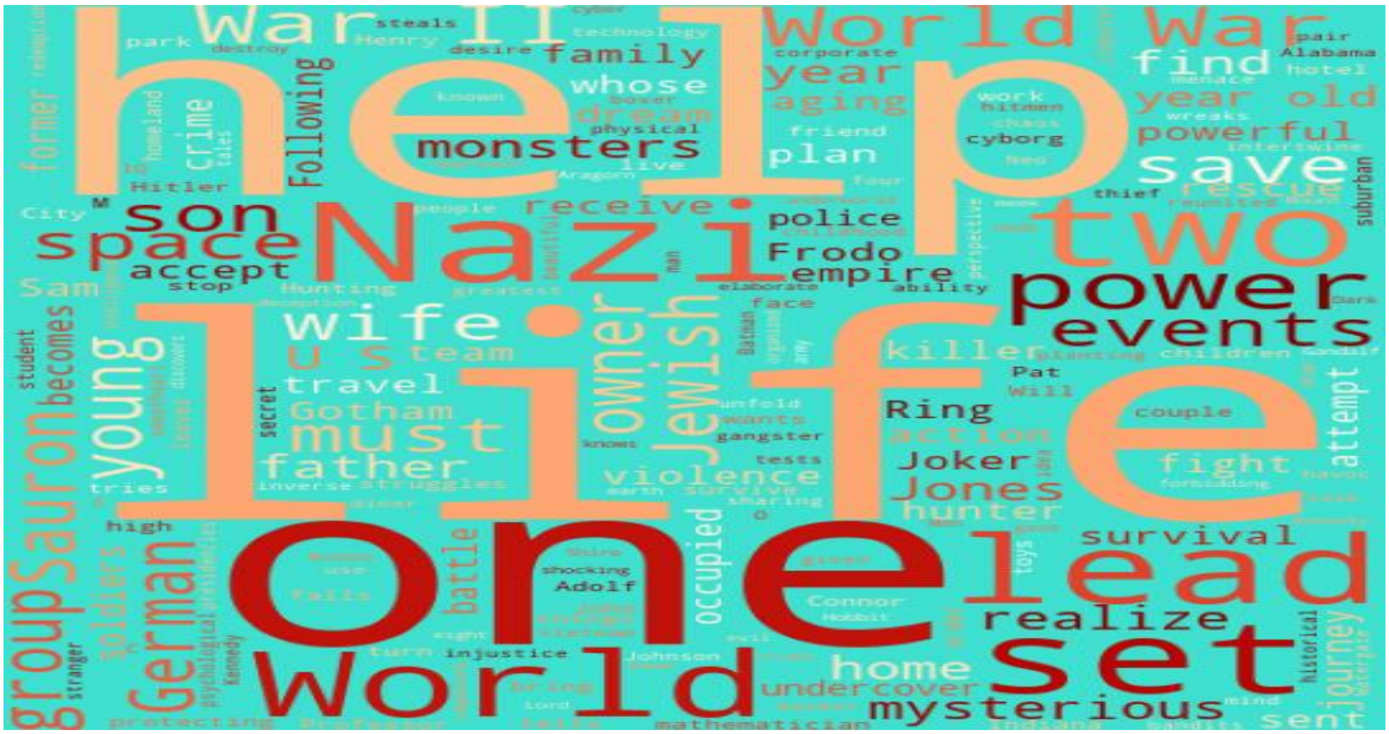
```
Counter({'Biography': 44,
         'Drama': 199,
         'History': 25,
         'Thriller': 31,
         'War': 20,
         'Adventure': 58,
         'Sci-Fi': 18,
         'Music': 13,
         'Romance': 43,
         'Animation': 19,
         'Comedy': 52,
         'Horror': 6,
         'Action': 36,
         'Musical': 4,
         'Family': 12,
         'Mystery': 18,
         'Fantasy': 10,
         'Crime': 34,
         'Western': 7,
         'Film-Noir': 4,
         'Sport': 7})
```




```python
#creating series of counted list
awarded_genre = pd.Series(counts_lst)
#plotting genres according to counts of each genre
sns.countplot(x = awarded_genre)
plt.xticks(rotation = 50)
plt.title('Genres of awarded movies')
```

3-In the overviews of the most first fifty voted award-winning films, the words relating to second world war and life were the most mentioned ones.

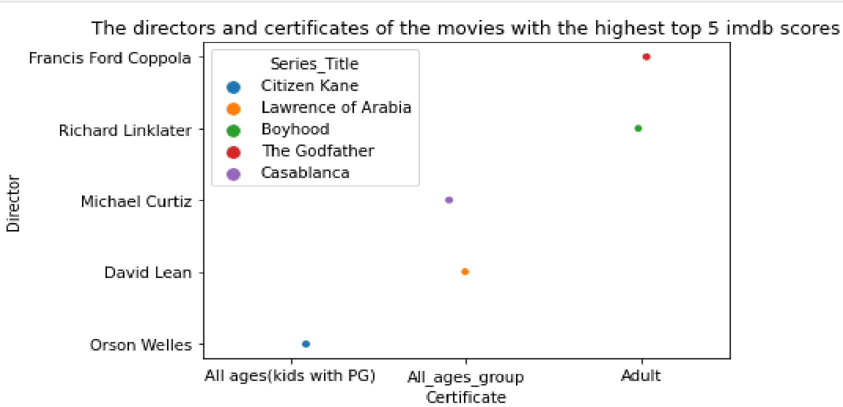
```python
awarded = merged_data[merged_data['win']== 1]
awarded = awarded.sort_values(by= 'No_of_Votes',ascending = False).head(50)
def plot_cloud(wordcloud):
    plt.figure(figsize=(10, 10))
    plt.imshow(wordcloud)
    plt.axis("off");
wordcloud = WordCloud(width = 500, height = 500,
background_color='#40E0D0', colormap="OrRd").generate(' '.join(awarded['Overview']))
plot_cloud(wordcloud)
```



4-The directors, titles, and certificates of awarded movies with the 5 highest meta-score values:

```python
"""extracting oscar awarded movies"""
awarded = merged_data[merged_data['win']== 1]
awarded_first_five_metascore = awarded.sort_values(by= 'Meta_score',ascending = False).head()
#awarded_first_five_metascore
```

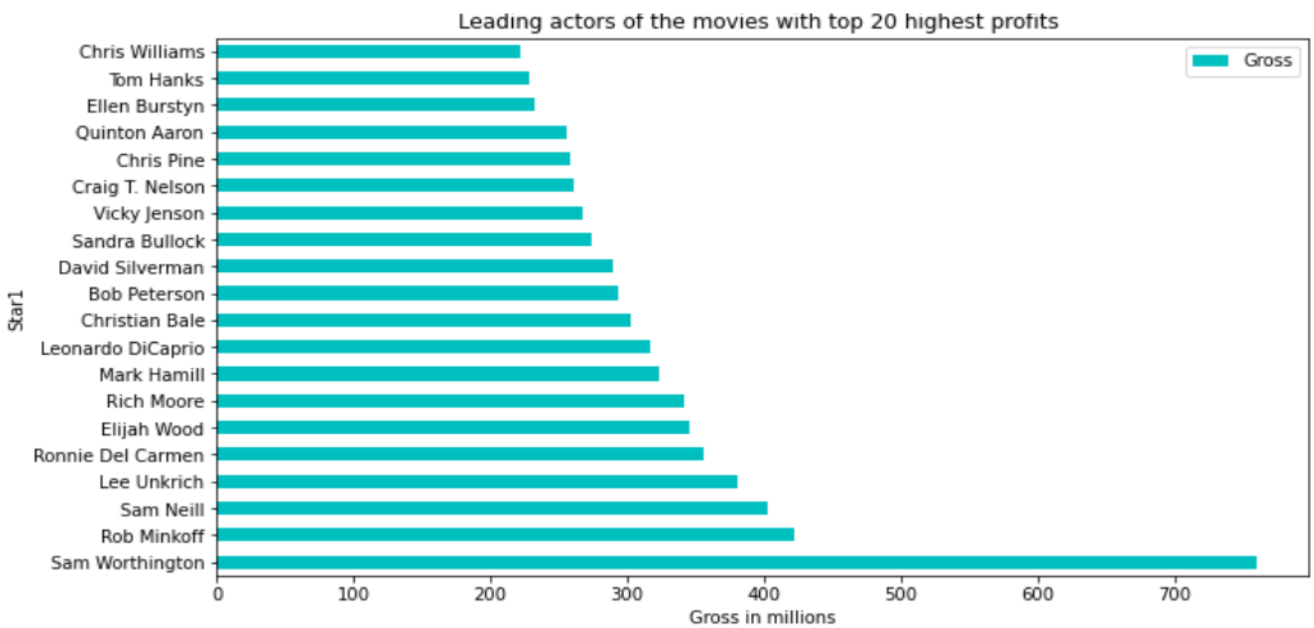
```python
sns.stripplot(x = 'Certificate',y = 'Director',hue ='Series_Title',data = awarded_first_five_metascore)
plt.title('The directors and certificates of the movies with the highest top 5 meta_scores')
plt.show()
```



5- Leading actors of top award-winning 20 movies with highest gross values:

```
#extracting Oscar awarded movies
awarded_data = merged_data[merged_data['win']  == 1]
#extracting leading actor and gross columns
star1_gross= awarded_data[['Star1','Gross']]
#grouping data by actors and getting mean of gross
mean_gross = star1_gross.groupby('Star1').mean()/1000000
#sorting gross values
mean_gross = mean_gross.sort_values('Gross', ascending=False)
#extracting first 20 top gross values
mean_gross_first_twenty = mean_gross.head(20)
```

```
mean_gross_first_twenty.plot.barh(title = 'Leading actors of the movies with top 20 highest profits ',color = 'c',figsize=(11, 6))
plt.xlabel('Gross in millions')
plt.show()
```



9-REFERENCES:

1. https://www.thesun.co.uk/tvandshowbiz/2589715/what-are-the-oscars-academy-awards/
2. https://help.imdb.com/article/imdb/general-information/what-is-imdb/G836CY29Z4SGNMK5?ref_=helpart_nav_1#
3. https://en.wikipedia.org/wiki/Kaggle
4. https://www.kaggle.com/unanimad/the-oscar-award
5. https://datasets.imdbws.com/title.basics.tsv.gz
6. https://simple.wikipedia.org/wiki/Motion_Picture_Association_of_America_film_rating_system
7. https://towardsdatascience.com/understanding-feature-engineering-part-2-categorical-data-f54324193e63
8. https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html
9. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c