In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xgb
from sklearn.metrics import mean_squared_error, accuracy_score, r2_score, mean_absol
from sklearn import metrics
```

In [2]:
```
df=pd.read_csv('movie_metadata.csv')
# Reading the dataset
```

In [3]:
```
pd.set_option('display.max_columns', None)
```

In [4]:
```
df
# Loading the dataset
```

Out[4]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebo
0	Color	James Cameron	723.0	178.0	0.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	
2	Color	Sam Mendes	602.0	148.0	0.0	
3	Color	Christopher Nolan	813.0	164.0	22000.0	
4	NaN	Doug Walker	NaN	NaN	131.0	
...	...	...	...	...	...	
5038	Color	Scott Smith	1.0	87.0	2.0	
5039	Color	NaN	43.0	43.0	NaN	
5040	Color	Benjamin Roberds	13.0	76.0	0.0	

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebo
5041	Color	Daniel Hsia	14.0	100.0	0.0	
5042	Color	Jon Gunn	43.0	90.0	16.0	

5043 rows × 28 columns

In [5]:
```
df.shape
# the dataset has around 5043 rows and 28 columns
```

Out[5]: (5043, 28)

In [6]:
```
df.info()
# Checking the information on the dataset
```

```
<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
```

In [7]:
```
df.isnull().sum()
# There are null values present in this dataset which have to be imputed.
```

Out[7]:
```
color                      19
director_name             104
num_critic_for_reviews     50
duration                   15
director_facebook_likes   104
```

```
actor_3_facebook_likes        23
actor_2_name                  13
actor_1_facebook_likes         7
gross                        884
genres                         0
actor_1_name                   7
movie_title                    0
num_voted_users                0
cast_total_facebook_likes      0
actor_3_name                  23
facenumber_in_poster          13
plot_keywords                153
movie_imdb_link                0
num_user_for_reviews          21
language                      12
country                        5
content_rating               303
budget                       492
title_year                   108
actor_2_facebook_likes        13
imdb_score                     0
aspect_ratio                 329
movie_facebook_likes           0
dtype: int64
```

In [8]:
```
df.columns
# Displaying all the column names present in the dataframe.
```

Out[8]:
```
Index(['color', 'director_name', 'num_critic_for_reviews', 'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
       'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
       'movie_imdb_link', 'num_user_for_reviews', 'language', 'country',
       'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
       'imdb_score', 'aspect_ratio', 'movie_facebook_likes'],
      dtype='object')
```

In [9]:
```
df['color'].value_counts()
# We have 4815 colour movies and 209 Black and white movies.
```

Out[9]:
```
Color               4815
 Black and White     209
Name: color, dtype: int64
```

In [10]:
```
df.describe()
# Using the describe function to check the min, max values as well as mean and stand
```

Out[10]:

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_fa
count	4993.000000	5028.000000	4939.000000	5020.000000	
mean	140.194272	107.201074	686.509212	645.009761	
std	121.601675	25.197441	2813.328607	1665.041728	
min	1.000000	7.000000	0.000000	0.000000	
25%	50.000000	93.000000	7.000000	133.000000	
50%	110.000000	103.000000	49.000000	371.500000	
75%	195.000000	118.000000	194.500000	636.000000	
max	813.000000	511.000000	23000.000000	23000.000000	6

In [11]:
```
df.nunique()
# Displays the total number of variables present in the data frames columns
```

Out[11]:
```
color                        2
director_name             2398
num_critic_for_reviews     528
duration                   191
director_facebook_likes    435
actor_3_facebook_likes     906
actor_2_name              3032
actor_1_facebook_likes     878
gross                     4035
genres                     914
actor_1_name              2097
movie_title               4917
num_voted_users           4826
cast_total_facebook_likes 3978
actor_3_name              3521
facenumber_in_poster        19
plot_keywords             4760
movie_imdb_link           4919
num_user_for_reviews       954
language                    47
country                     65
content_rating              18
budget                     439
title_year                  91
actor_2_facebook_likes     917
imdb_score                  78
aspect_ratio                22
movie_facebook_likes       876
dtype: int64
```

In [12]:
```
df['language'].value_counts()
# A lot of movies in this dataset are in English.
```

Out[12]:
```
English       4704
French          73
Spanish         40
Hindi           28
Mandarin        26
German          19
Japanese        18
Italian         11
Russian         11
Cantonese       11
Portuguese       8
Korean           8
Arabic           5
Danish           5
Swedish          5
Hebrew           5
Polish           4
Persian          4
Dutch            4
Norwegian        4
Thai             3
Chinese          3
Zulu             2
Aboriginal       2
Icelandic        2
None             2
Dari             2
Romanian         2
Indonesian       2
```

```
Aramaic          1
Filipino         1
Czech            1
Panjabi          1
Hungarian        1
Kazakh           1
Maya             1
Urdu             1
Tamil            1
Mongolian        1
Greek            1
Swahili          1
Vietnamese       1
Dzongkha         1
Kannada          1
Telugu           1
Slovenian        1
Bosnian          1
Name: language, dtype: int64
```

In [13]:

```
df['movie_title']
# Displaying all the movie names.
```

Out[13]:

```
0                                              Avatar
1            Pirates of the Caribbean: At World's End
2                                             Spectre
3                              The Dark Knight Rises
4      Star Wars: Episode VII - The Force Awakens    ...
                            ...
5038                         Signed Sealed Delivered
5039                        The Following
5040                             A Plague So Pleasant
5041                                 Shanghai Calling
5042                               My Date with Drew
Name: movie_title, Length: 5043, dtype: object
```

In [14]:

```
df.movie_title[df.language == 'Hindi']
# printing the name of Hindi movies present in the dataset.
```

Out[14]:

```
1056                                Earth
3075              Kabhi Alvida Naa Kehna
3085                            Housefull
3208                               Krrish
3276                    Jab Tak Hai Jaan
3344                     My Name Is Khan
3348                     Namastey London
3350            Yeh Jawaani Hai Deewani
3455                       Ta Ra Rum Pum
3510                          Veer-Zaara
3665                       Khiladi 786
3685                    Rang De Basanti
3695                      Dum Maaro Dum
3789                  Gandhi, My Father
3866                              Fugly
3870                            Airlift
3877                                Paa
4088                              Water
4160                Lage Raho Munna Bhai
4299              Hum To Mohabbat Karega
4305                      Roadside Romeo
4351           ABCD (Any Body Can Dance)
4385                        The Lunchbox
4490                     Monsoon Wedding
4528    Rocket Singh: Salesman of the Year
4572                                Fiza
4593          Chocolate: Deep Dark Secrets
```

```
4724                    Faith Connections
Name: movie_title, dtype: object
```

In [15]:
```
bw=df[df.language == 'Hindi']
# Saving the Hindi movies into a separate dataframe for futher analysis
```

In [16]:
```
bw
# Dataframe of all hindi movies with high imdb scores
```

Out[16]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebo
1056	Color	Deepa Mehta	34.0	110.0	375.0	
3075	Color	Karan Johar	20.0	193.0	160.0	
3085	Color	Sajid Khan	10.0	144.0	0.0	
3208	Color	Rakesh Roshan	20.0	168.0	53.0	
3276	Color	Yash Chopra	50.0	176.0	147.0	
3344	Color	Karan Johar	210.0	128.0	160.0	
3348	Color	Vipul Amrutlal Shah	15.0	128.0	11.0	
3350	Color	Ayan Mukerji	25.0	160.0	0.0	
3455	Color	Siddharth Anand	16.0	153.0	5.0	
3510	Color	Yash Chopra	29.0	192.0	147.0	
3665	Color	Ashish R. Mohan	23.0	141.0	2.0	
3685	Color	Rakeysh Omprakash Mehra	33.0	157.0	85.0	
3695	Color	Rohan Sippy	24.0	128.0	4.0	
3789	Color	Feroz Abbas Khan	16.0	136.0	0.0	
3866	Color	Kabir Sadanand	9.0	134.0	0.0	
3870	Color	Raja Menon	39.0	130.0	6.0	
3877	Color	R. Balki	12.0	133.0	12.0	
4088	Color	Deepa Mehta	91.0	117.0	375.0	

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebo
4160	Color	Rajkumar Hirani	15.0	144.0	124.0	
4299	Color	Kundan Shah	1.0	NaN	4.0	
4305	Color	Jugal Hansraj	6.0	93.0	32.0	
4351	Color	Remo	15.0	160.0	168.0	
4385	Color	Ritesh Batra	195.0	104.0	25.0	
4490	Color	Mira Nair	137.0	114.0	300.0	
4528	Color	Shimit Amin	14.0	150.0	6.0	
4572	Black and White	Khalid Mohamed	1.0	167.0	10.0	
4593	Color	Vivek Agnihotri	4.0	160.0	5.0	
4724	Color	Pan Nalin	19.0	115.0	95.0	

In [17]:
```
scores=bw.imdb_score[bw.imdb_score>=7]
# storing movies with imdb score equal to 8 or above 8
```

In [18]:
```
name=bw.movie_title[bw.imdb_score>=7]
# storing movie_titles with imdb scores above 8 or equal to 8
```

In [19]:
```
genre=bw.genres[bw.imdb_score>=7]
# storing movie genre with imdb score above or equal to 8
```

In [20]:
```
bollywoodmovieratings=pd.concat([name,genre,scores],axis=1)
# creating a dataframe with all values concatenated into one dataframe
```

In [21]:
```
bollywoodmovieratings
# final concatenated dataframe of Indian movies and their genres and imdb_scores
```

Out[21]:

	movie_title	genres	imdb_score
1056	Earth	Drama|Romance|War	7.8

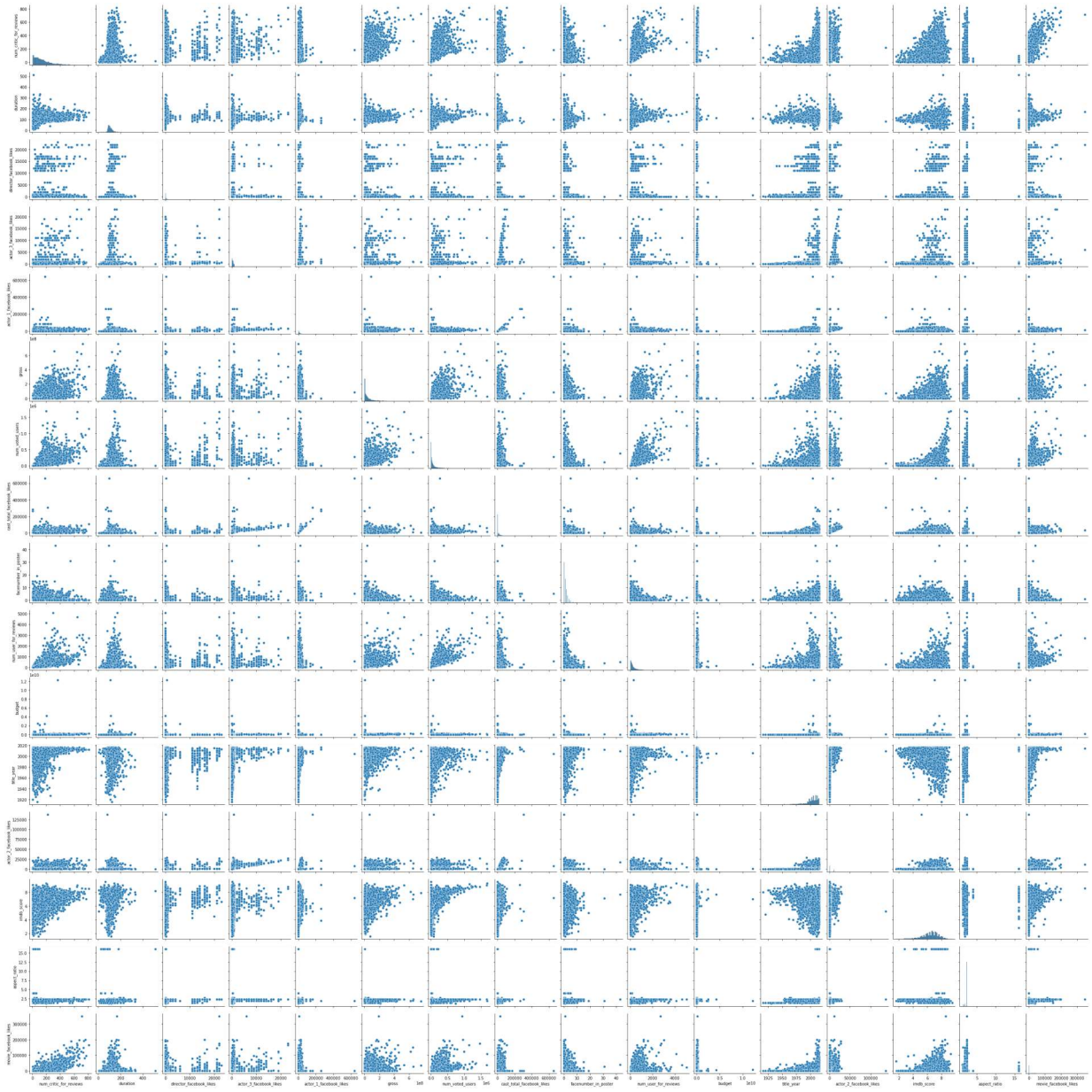
	movie_title	genres	imdb_score
3344	My Name Is Khan	Adventure|Drama|Thriller	8.0
3348	Namastey London	Comedy|Drama|Romance	7.3
3510	Veer-Zaara	Drama|Musical|Romance	7.9
3685	Rang De Basanti	Comedy|Drama|History|Romance	8.4
3789	Gandhi, My Father	Biography|Drama|History	7.4
3870	Airlift	Action|Drama|History|Thriller|War	8.5
3877	Paa	Comedy|Drama	7.2
4088	Water	Drama|Romance	7.8
4160	Lage Raho Munna Bhai	Comedy|Drama|Romance	8.2
4385	The Lunchbox	Drama|Romance	7.8
4490	Monsoon Wedding	Comedy|Drama|Romance	7.4
4528	Rocket Singh: Salesman of the Year	Comedy|Drama	7.5
4724	Faith Connections	Biography|Documentary|Drama	7.0

As seen above Indian movies have a higher imdb score if it has a genre with Action|Drama|History|Thriller|War included in it.

Data visualization

In [22]:
```
sns.pairplot(data=df)
```

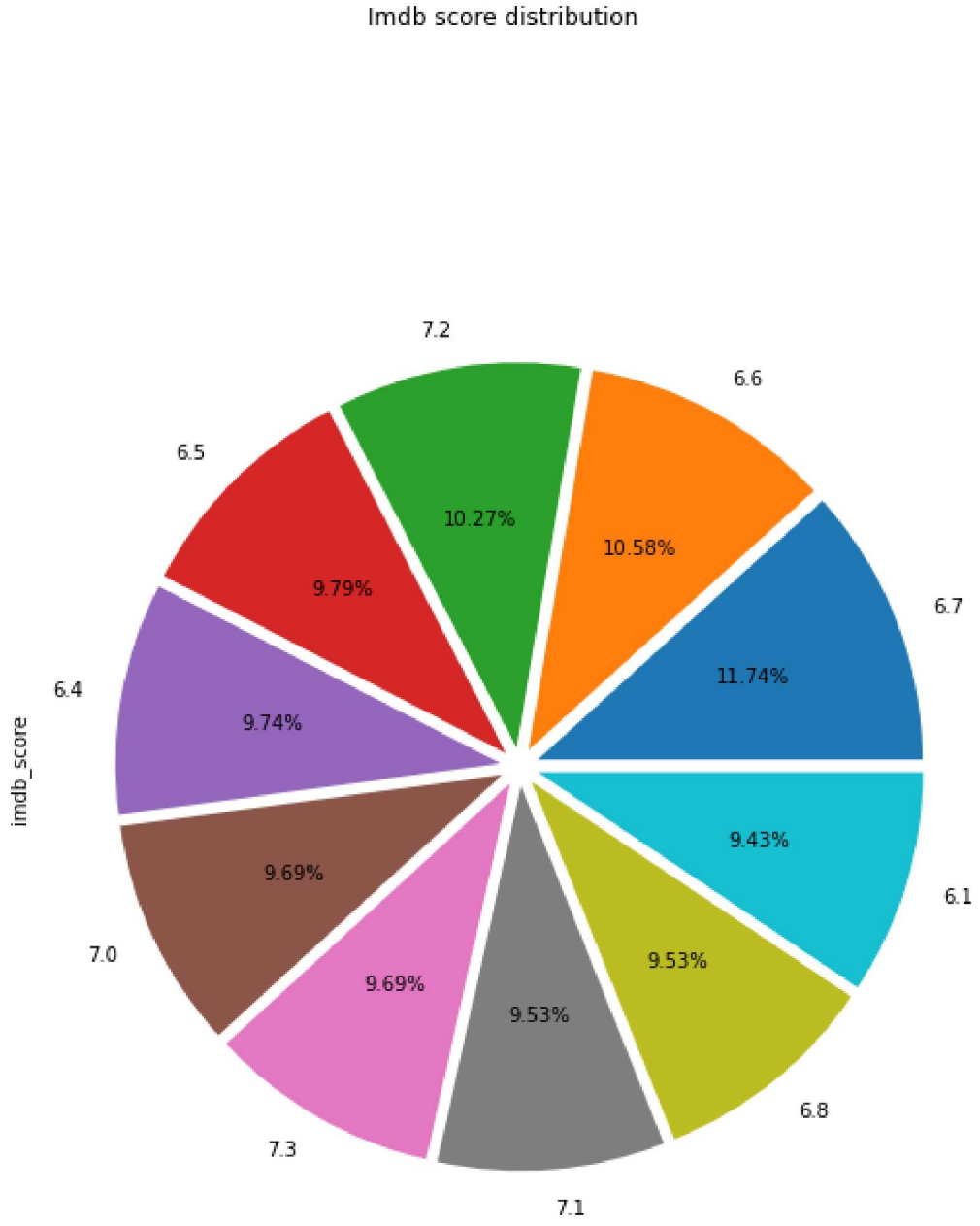
Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x2083c322af0>
```



In [24]:

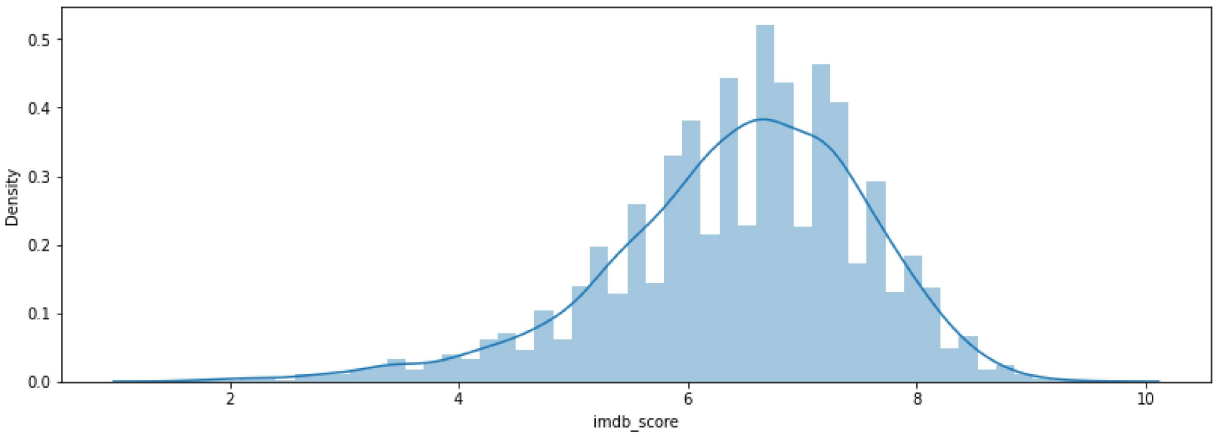
```
f,ax=plt.subplots(figsize=(10,20))
ax1=plt.subplot(211)
f.suptitle("Imdb score distribution")
explode=(0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05)
df['imdb_score'].value_counts(ascending=False).head(10).plot(kind='pie',autopct="%0.
# Pie plot for top 10 Imdb_scores.
# Around 11.74 percent of the data has a imdb_score which is 6.7
```

Out[24]: <AxesSubplot:ylabel='imdb_score'>

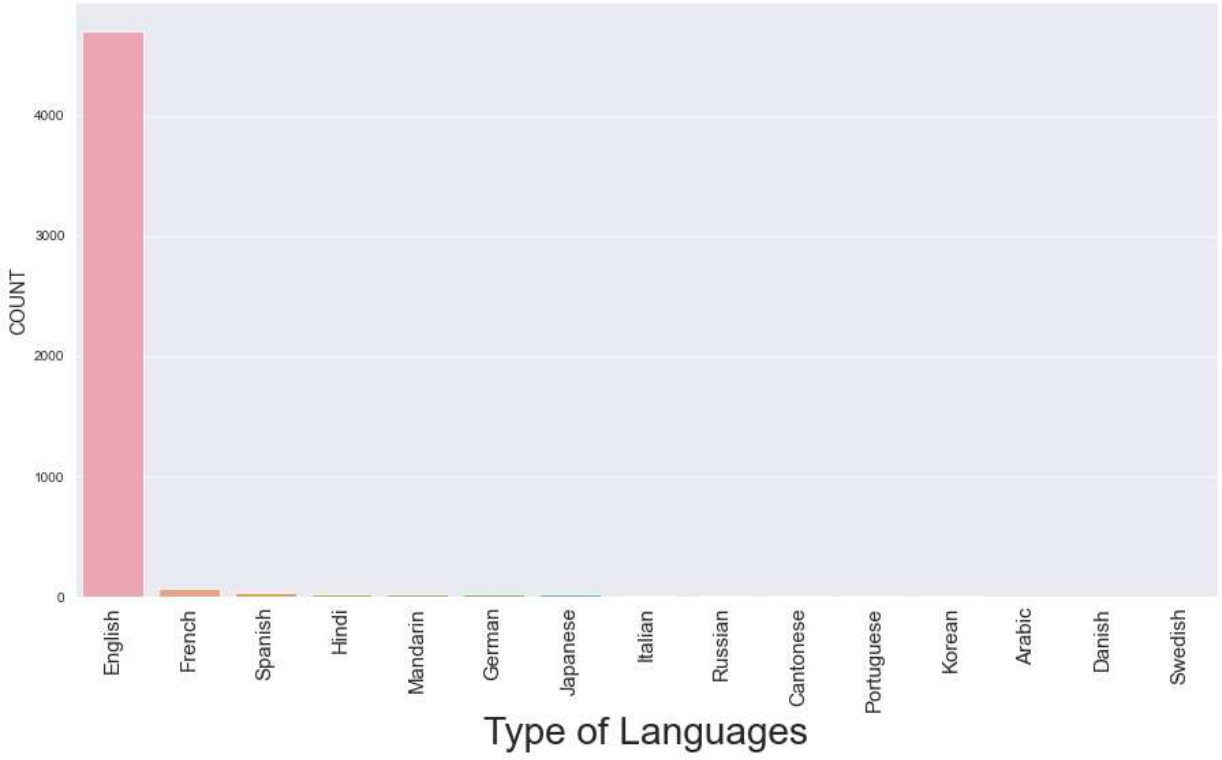


In [25]:
```
# Distribution plot for all prices.
f,ax=plt.subplots(figsize=(30,10))
ax3=plt.subplot(224)
sns.distplot(df['imdb_score'],ax=ax3)
# Using the distribution plot we can see that the imdb score is the highest at a imd
```

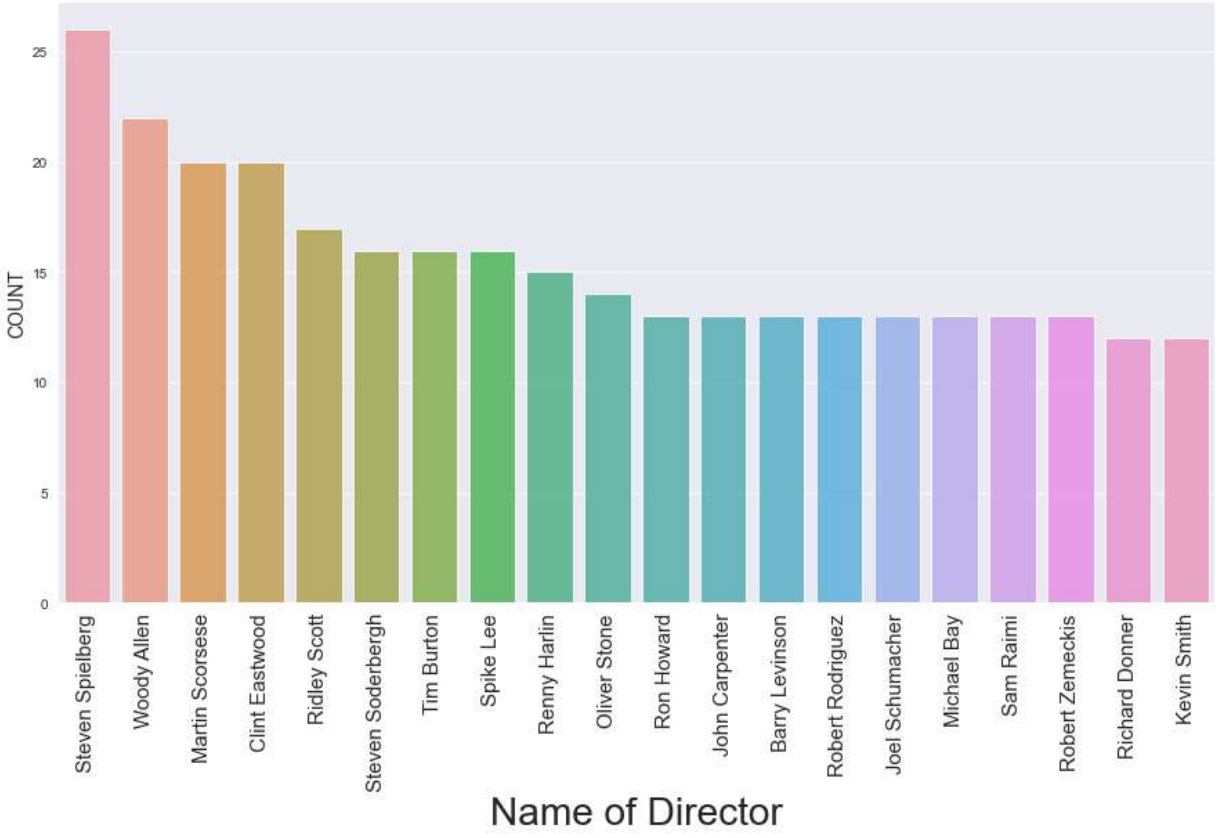
Out[25]: <AxesSubplot:xlabel='imdb_score', ylabel='Density'>



In [26]:
```
sns.set_style("darkgrid")
ls=df['language'].value_counts().head(15).sort_values(ascending=False)
plt.figure(figsize=(15,8))
temp =sns.barplot(ls.index, ls.values, alpha=0.8)
plt.ylabel('COUNT', fontsize=14)
plt.xlabel('Type of Languages', fontsize=28)
temp.set_xticklabels(rotation=90,labels=ls.index,fontsize=15)
plt.show()
# Visualizing the count of languages in the dataset.
```



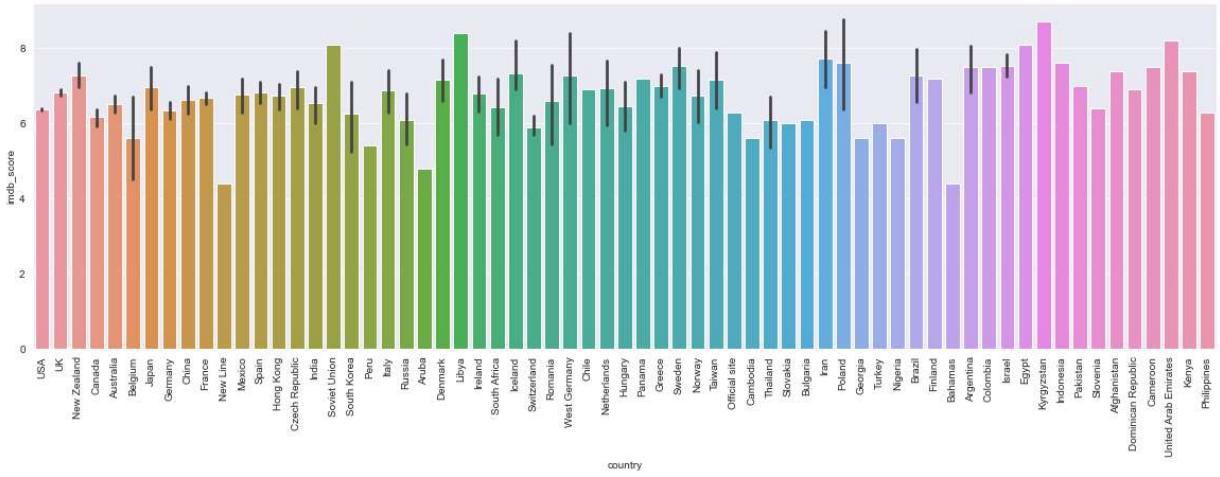
In [27]:
```
sns.set_style("darkgrid")
ls=df['director_name'].value_counts().head(20).sort_values(ascending=False)
plt.figure(figsize=(15,8))
temp =sns.barplot(ls.index, ls.values, alpha=0.8)
plt.ylabel('COUNT', fontsize=14)
plt.xlabel('Name of Director', fontsize=28)
temp.set_xticklabels(rotation=90,labels=ls.index,fontsize=15)
plt.show()
# Visualizing the Directors names present in the dataset.
```



In [28]:
```
plt.figure(figsize=(20, 6))
sns.barplot(x='country',y='imdb_score',data=df);
plt.xticks(rotation=90)
# Visualizing the barplot of countries and the imdb scores.
```

Out[28]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64]),
 [Text(0, 0, 'USA'),
  Text(1, 0, 'UK'),
  Text(2, 0, 'New Zealand'),
  Text(3, 0, 'Canada'),
  Text(4, 0, 'Australia'),
  Text(5, 0, 'Belgium'),
  Text(6, 0, 'Japan'),
  Text(7, 0, 'Germany'),
  Text(8, 0, 'China'),
  Text(9, 0, 'France'),
  Text(10, 0, 'New Line'),
  Text(11, 0, 'Mexico'),
  Text(12, 0, 'Spain'),
  Text(13, 0, 'Hong Kong'),
  Text(14, 0, 'Czech Republic'),
  Text(15, 0, 'India'),
  Text(16, 0, 'Soviet Union'),
  Text(17, 0, 'South Korea'),
  Text(18, 0, 'Peru'),
  Text(19, 0, 'Italy'),
  Text(20, 0, 'Russia'),
  Text(21, 0, 'Aruba'),
  Text(22, 0, 'Denmark'),
  Text(23, 0, 'Libya'),
  Text(24, 0, 'Ireland'),
  Text(25, 0, 'South Africa'),
  Text(26, 0, 'Iceland'),
  Text(27, 0, 'Switzerland'),
  Text(28, 0, 'Romania'),
```

```
 Text(29, 0, 'West Germany'),
 Text(30, 0, 'Chile'),
 Text(31, 0, 'Netherlands'),
 Text(32, 0, 'Hungary'),
 Text(33, 0, 'Panama'),
 Text(34, 0, 'Greece'),
 Text(35, 0, 'Sweden'),
 Text(36, 0, 'Norway'),
 Text(37, 0, 'Taiwan'),
 Text(38, 0, 'Official site'),
 Text(39, 0, 'Cambodia'),
 Text(40, 0, 'Thailand'),
 Text(41, 0, 'Slovakia'),
 Text(42, 0, 'Bulgaria'),
 Text(43, 0, 'Iran'),
 Text(44, 0, 'Poland'),
 Text(45, 0, 'Georgia'),
 Text(46, 0, 'Turkey'),
 Text(47, 0, 'Nigeria'),
 Text(48, 0, 'Brazil'),
 Text(49, 0, 'Finland'),
 Text(50, 0, 'Bahamas'),
 Text(51, 0, 'Argentina'),
 Text(52, 0, 'Colombia'),
 Text(53, 0, 'Israel'),
 Text(54, 0, 'Egypt'),
 Text(55, 0, 'Kyrgyzstan'),
 Text(56, 0, 'Indonesia'),
 Text(57, 0, 'Pakistan'),
 Text(58, 0, 'Slovenia'),
 Text(59, 0, 'Afghanistan'),
 Text(60, 0, 'Dominican Republic'),
 Text(61, 0, 'Cameroon'),
 Text(62, 0, 'United Arab Emirates'),
 Text(63, 0, 'Kenya'),
 Text(64, 0, 'Philippines')])
```



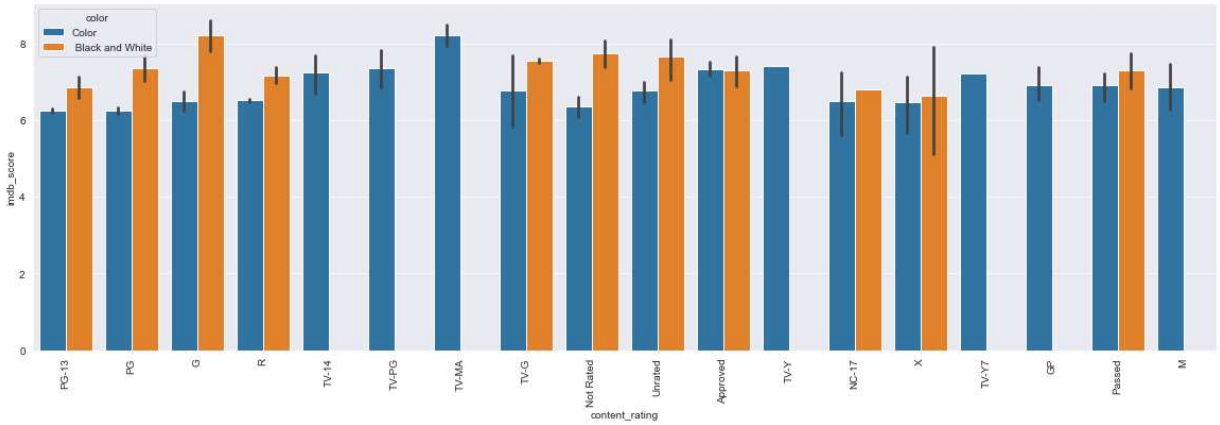
In [29]:

```
plt.figure(figsize=(20, 6))
sns.barplot(x='content_rating',y='imdb_score',hue='color',data=df);
plt.xticks(rotation=90)
# This visualization shows the type of content having higher imdbscore and shows us
```

Out[29]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17]),
 [Text(0, 0, 'PG-13'),
  Text(1, 0, 'PG'),
  Text(2, 0, 'G'),
  Text(3, 0, 'R'),
  Text(4, 0, 'TV-14'),
  Text(5, 0, 'TV-PG'),
  Text(6, 0, 'TV-MA'),
```

```
 Text(7, 0, 'TV-G'),
 Text(8, 0, 'Not Rated'),
 Text(9, 0, 'Unrated'),
 Text(10, 0, 'Approved'),
 Text(11, 0, 'TV-Y'),
 Text(12, 0, 'NC-17'),
 Text(13, 0, 'X'),
 Text(14, 0, 'TV-Y7'),
 Text(15, 0, 'GP'),
 Text(16, 0, 'Passed'),
 Text(17, 0, 'M')])
```

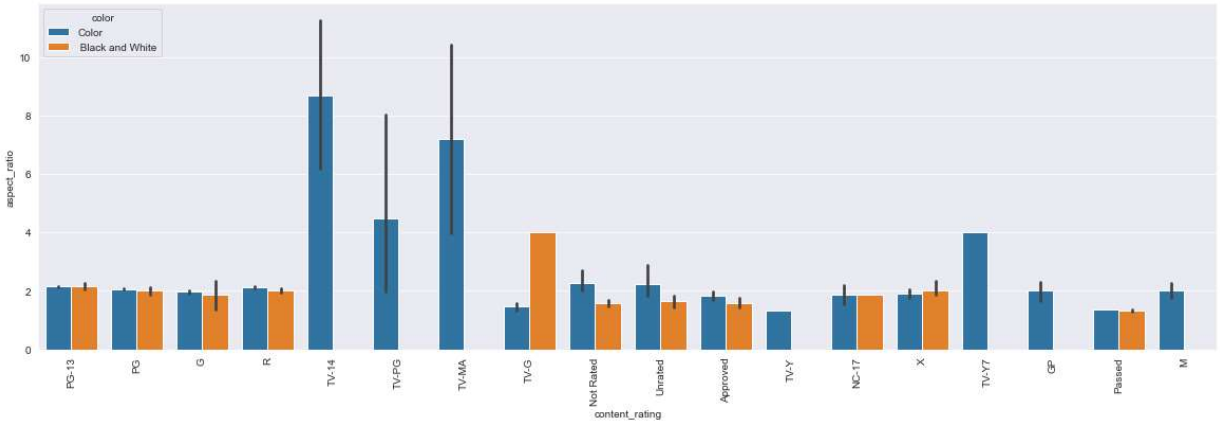


In [30]:

```
plt.figure(figsize=(20, 6))
sns.barplot(x='content_rating',y='aspect_ratio',hue='color',data=df);
plt.xticks(rotation=90)
# This visualization shows the type of content rating on X-axis having aspect ratio
```

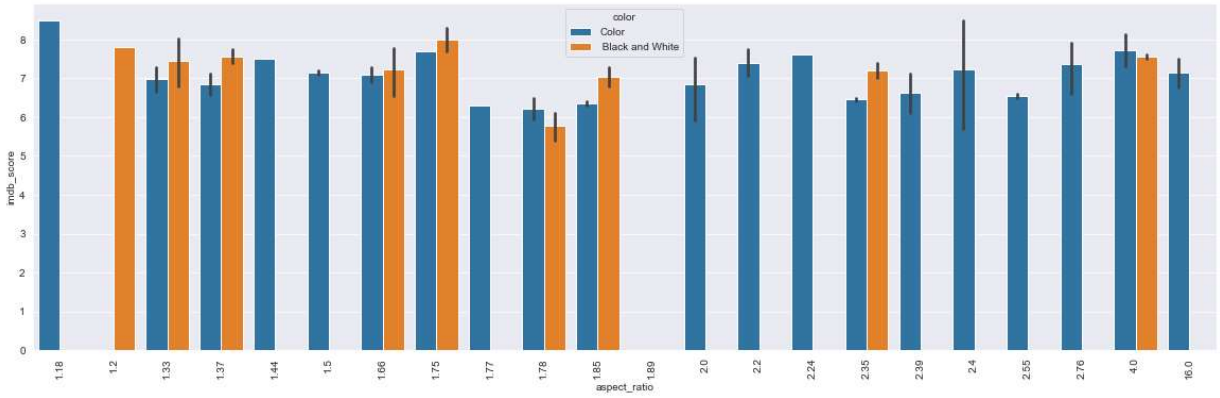
Out[30]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17]),
 [Text(0, 0, 'PG-13'),
  Text(1, 0, 'PG'),
  Text(2, 0, 'G'),
  Text(3, 0, 'R'),
  Text(4, 0, 'TV-14'),
  Text(5, 0, 'TV-PG'),
  Text(6, 0, 'TV-MA'),
  Text(7, 0, 'TV-G'),
  Text(8, 0, 'Not Rated'),
  Text(9, 0, 'Unrated'),
  Text(10, 0, 'Approved'),
  Text(11, 0, 'TV-Y'),
  Text(12, 0, 'NC-17'),
  Text(13, 0, 'X'),
  Text(14, 0, 'TV-Y7'),
  Text(15, 0, 'GP'),
  Text(16, 0, 'Passed'),
  Text(17, 0, 'M')])
```



In [31]:
```
plt.figure(figsize=(20, 6))
sns.barplot(x='aspect_ratio',y='imdb_score',hue='color',data=df);
plt.xticks(rotation=90)
# This visualization shows the aspect ratio and its imdb score with hue as the color
```

Out[31]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]),
 [Text(0, 0, '1.18'),
  Text(1, 0, '1.2'),
  Text(2, 0, '1.33'),
  Text(3, 0, '1.37'),
  Text(4, 0, '1.44'),
  Text(5, 0, '1.5'),
  Text(6, 0, '1.66'),
  Text(7, 0, '1.75'),
  Text(8, 0, '1.77'),
  Text(9, 0, '1.78'),
  Text(10, 0, '1.85'),
  Text(11, 0, '1.89'),
  Text(12, 0, '2.0'),
  Text(13, 0, '2.2'),
  Text(14, 0, '2.24'),
  Text(15, 0, '2.35'),
  Text(16, 0, '2.39'),
  Text(17, 0, '2.4'),
  Text(18, 0, '2.55'),
  Text(19, 0, '2.76'),
  Text(20, 0, '4.0'),
  Text(21, 0, '16.0')])
```



In [33]:
```
df=df.dropna()
# Removing null values
```

In [34]:
```
df.isnull().sum()
# As you can see there is no null values present in the dataset now.
```

Out[34]:
```
color                      0
director_name              0
num_critic_for_reviews     0
duration                   0
director_facebook_likes    0
actor_3_facebook_likes     0
actor_2_name               0
actor_1_facebook_likes     0
gross                      0
genres                     0
actor_1_name               0
movie_title                0
num_voted_users            0
cast_total_facebook_likes  0
actor_3_name               0
facenumber_in_poster       0
```

```
plot_keywords               0
movie_imdb_link             0
num_user_for_reviews        0
language                    0
country                     0
content_rating              0
budget                      0
title_year                  0
actor_2_facebook_likes      0
imdb_score                  0
aspect_ratio                0
movie_facebook_likes        0
dtype: int64
```

In [35]:
```
df.columns
```

Out[35]:
```
Index(['color', 'director_name', 'num_critic_for_reviews', 'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
       'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
       'movie_imdb_link', 'num_user_for_reviews', 'language', 'country',
       'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
       'imdb_score', 'aspect_ratio', 'movie_facebook_likes'],
      dtype='object')
```

In [36]:
```
df=df.drop(columns=['movie_imdb_link','color','movie_title','facenumber_in_poster',
                    'actor_3_name','movie_imdb_link','aspect_ratio','language'])
```

In [37]:
```
df
```

Out[37]:

	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes
0	James Cameron	723.0	178.0	0.0	855.0
1	Gore Verbinski	302.0	169.0	563.0	1000.0
2	Sam Mendes	602.0	148.0	0.0	161.0
3	Christopher Nolan	813.0	164.0	22000.0	23000.0
5	Andrew Stanton	462.0	132.0	475.0	530.0
...	...	...	...	...	..
5026	Olivier Assayas	81.0	110.0	107.0	45.0
5027	Jafar Panahi	64.0	90.0	397.0	0.0
5033	Shane Carruth	143.0	77.0	291.0	8.0
5035	Robert Rodriguez	56.0	81.0	0.0	6.0
5042	Jon Gunn	43.0	90.0	16.0	16.0

3756 rows × 20 columns

In [38]:
```
df.shape
# The number of columns have now been reduced to 20
```

Out[38]: (3756, 20)

Label Encoding Categorical data

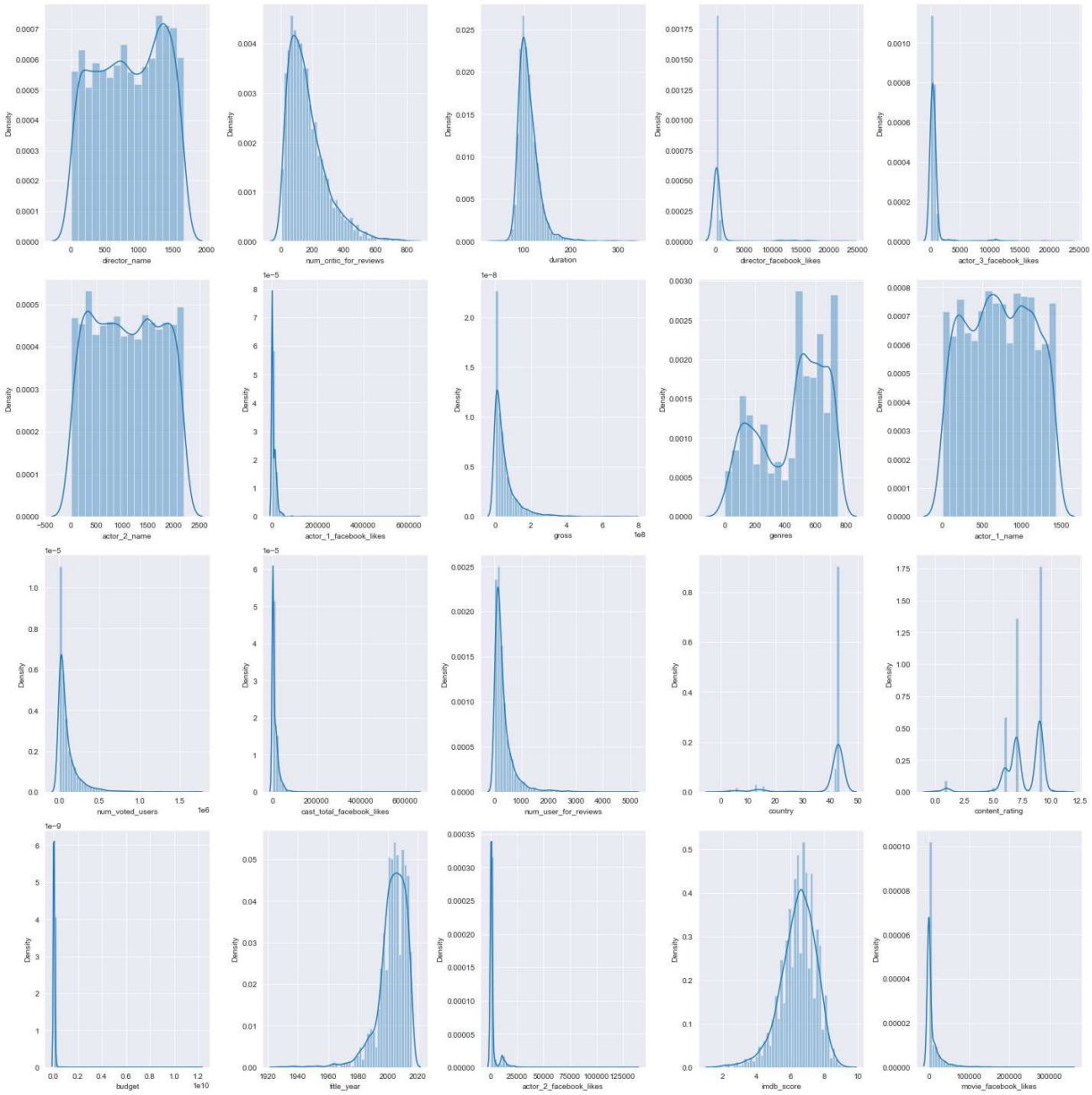
In [39]:
```
cat_cols=['content_rating','director_name','genres','actor_1_name','actor_2_name','c
le=LabelEncoder()
for i in cat_cols:
    df[i]=le.fit_transform(df[i])
df.dtypes
## We have label encoded the categorical columns in the dataset and transformed them
```

Out[39]:
```
director_name                int32
num_critic_for_reviews     float64
duration                   float64
director_facebook_likes    float64
actor_3_facebook_likes     float64
actor_2_name                 int32
actor_1_facebook_likes     float64
gross                      float64
genres                       int32
actor_1_name                 int32
num_voted_users              int64
cast_total_facebook_likes    int64
num_user_for_reviews       float64
country                      int32
content_rating               int32
budget                     float64
title_year                 float64
actor_2_facebook_likes     float64
imdb_score                 float64
movie_facebook_likes         int64
dtype: object
```

Distribution Plot

In [40]:
```
rows=4
cols=5
fig, ax=plt.subplots(nrows=rows,ncols=cols,figsize=(20,20))
col=df.columns
index=0
for i in range(rows):
    for j in range(cols):
        sns.distplot(df[col[index]],ax=ax[i][j])
        index=index+1

plt.tight_layout()
# The distribution plot shows us the overall distribution of the data.
```



Log Transformation

In [43]:
```
df.columns
# Displaying all column names,copypaste this in the next cell.
```

Out[43]:
```
Index(['director_name', 'num_critic_for_reviews', 'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'num_voted_users', 'cast_total_facebook_likes', 'num_user_for_reviews',
       'country', 'content_rating', 'budget', 'title_year',
       'actor_2_facebook_likes', 'imdb_score', 'movie_facebook_likes'],
      dtype='object')
```

In [44]:
```
skewed_features=['director_name', 'num_critic_for_reviews', 'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'num_voted_users', 'cast_total_facebook_likes', 'num_user_for_reviews',
       'country', 'content_rating', 'budget', 'title_year',
       'actor_2_facebook_likes', 'imdb_score', 'movie_facebook_likes']
# Selecting all features which are skewed and storing them i n the skewed_features
```

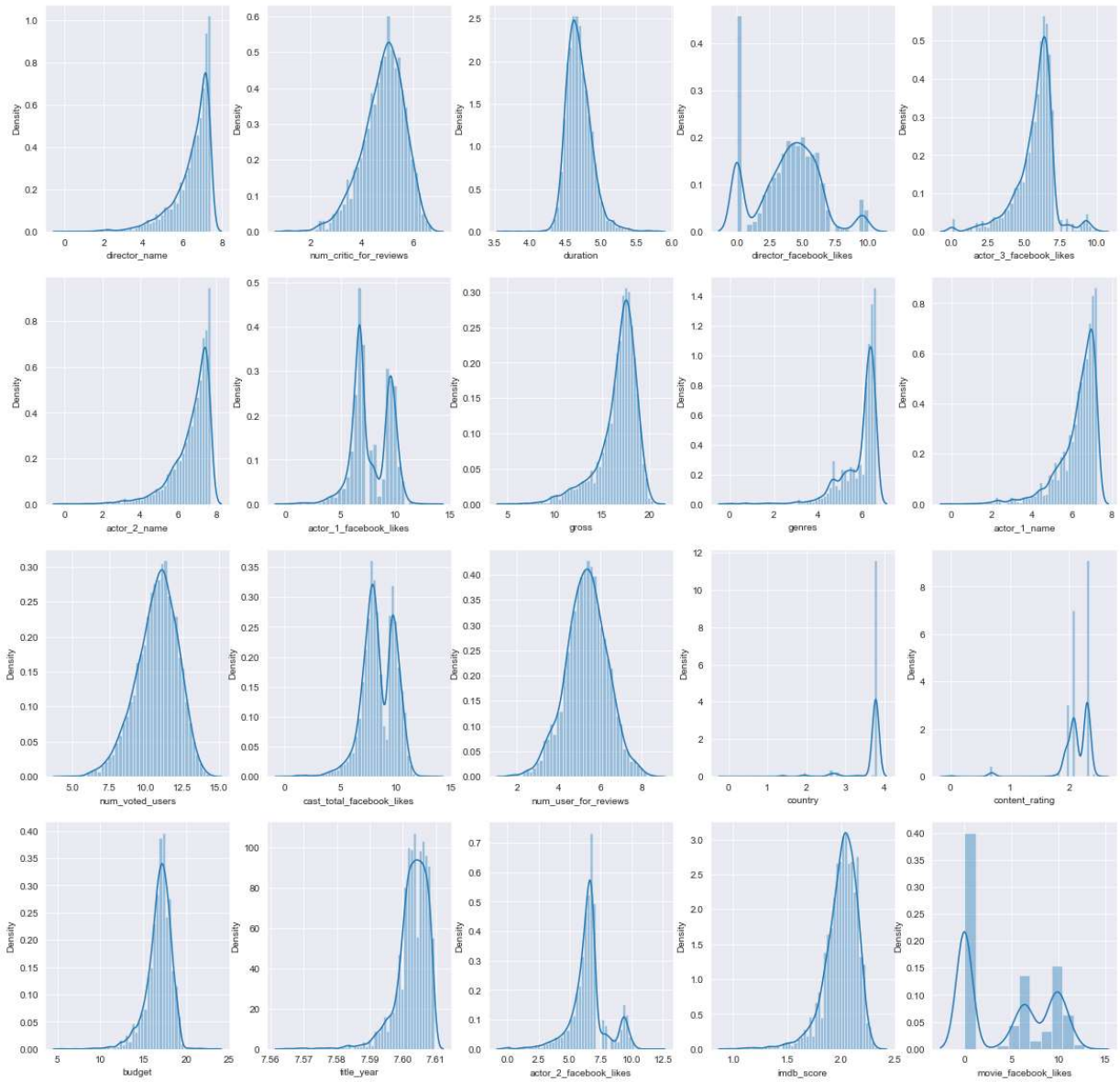
In [45]:

```
for i in skewed_features:
    df[i]=np.log(df[i]+1)
# Applying log transformation on the skewed features
```

In [46]:

```
rows=4
cols=5
fig, ax=plt.subplots(nrows=rows,ncols=cols,figsize=(20,20))
col=df.columns
index=0
for i in range(rows):
    for j in range(cols):
        sns.distplot(df[col[index]],ax=ax[i][j])
        index=index+1

plt.show()
# Checking the changes in the distribution of data after applying log transformation
```



Splitting dataset

In [47]:

```
X=df.drop(labels=['imdb_score'],axis=1)
Y=df['imdb_score']
X.head()
# splitting data into dependent and independent variables
```

Out[47]:

	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
0	6.431331	6.584791	5.187386	0.000000	6.752270	
1	6.289716	5.713733	5.135798	6.335054	6.908755	
2	7.241366	6.401917	5.003946	0.000000	5.087596	
3	5.529429	6.701960	5.105945	9.998843	10.043293	
5	4.143135	6.137727	4.890349	6.165418	6.274762	

In [48]:

```
Y.head()
# target column
```

Out[48]:

```
0    2.186051
1    2.091864
2    2.054124
3    2.251292
5    2.028148
Name: imdb_score, dtype: float64
```

In [49]:

```
# Train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=40)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
# Splitting data set into training and testing.
```

```
(3004, 19) (752, 19) (3004,) (752,)
```

Machine Learning

Linear Regression

In [50]:

```
lm=LinearRegression()
lm = lm.fit(X_train,Y_train)

#Traindata Predictions
train_pred = lm.predict(X_train)

#testdata predictions
test_pred = lm.predict(X_test)


RMSE_test = np.sqrt(mean_squared_error(Y_test, test_pred))
RMSE_train= np.sqrt(mean_squared_error(Y_train,train_pred))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',lm.score(X_train, Y_train))
print('RSquared value on test:',lm.score(X_test, Y_test))
```

```
RMSE TrainingData =  0.12041595568987461
RMSE TestData =  0.11861105307571095
--------------------------------------------------
RSquared value on train: 0.40800950550852366
RSquared value on test: 0.4021001552231912
```

In [51]:

```
errors = abs(test_pred - Y_test)
# Calculating errors for using error values in mean absolute percentage error
```

In [52]:
```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / Y_test)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 95.5 %.
```

Decision Tree Regressor

In [53]:
```
DT=DecisionTreeRegressor(max_depth=9)
DT.fit(X_train,Y_train)

#predicting train
train_preds=DT.predict(X_train)
#predicting on test
test_preds=DT.predict(X_test)

RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',DT.score(X_train, Y_train))
print('RSquared value on test:',DT.score(X_test, Y_test))
```

```
RMSE TrainingData =  0.08302711763715251
RMSE TestData =  0.1249030143248399
--------------------------------------------------
RSquared value on train: 0.7185595074222233
RSquared value on test: 0.3369840807616663
```

In [54]:
```
errors = abs(test_preds - Y_test)
# Calculating errors for using error values in mean absolute percentage error
```

In [55]:
```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / Y_test)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 95.48 %.
```

Random Forest Regressor

In [56]:
```
RF=RandomForestRegressor().fit(X_train,Y_train)

#predicting train
train_preds1=RF.predict(X_train)
#predicting on test
test_preds1=RF.predict(X_test)

RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds1)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds1)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
```

```
print('RSquared value on train:',RF.score(X_train, Y_train))
print('RSquared value on test:',RF.score(X_test, Y_test))
```

```
RMSE TrainingData =  0.04139656392848419
RMSE TestData =  0.10102168619593156
-------------------------------------------------
RSquared value on train: 0.9300359004959
RSquared value on test: 0.5662821027007214
```

In [57]:
```
errors = abs(test_preds1 - Y_test)
# Calculating errors for using error values in mean absolute percentage error
```

In [58]:
```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / Y_test)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 96.25 %.
```

K-Nearest Neighbours

In [59]:
```
knn=KNeighborsRegressor()
knn.fit(X_train,Y_train)

#predicting train
train_preds2=knn.predict(X_train)
#predicting on test
test_preds2=knn.predict(X_test)

RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds2)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds2)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',knn.score(X_train, Y_train))
print('RSquared value on test:',knn.score(X_test, Y_test))
```

```
RMSE TrainingData =  0.10856954037953365
RMSE TestData =  0.13173731630991498
-------------------------------------------------
RSquared value on train: 0.518758958219276
RSquared value on test: 0.2624427420400328
```

In [60]:
```
# More machine learning algorithms.
```

In [61]:
```
from sklearn.linear_model import LassoCV
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import ElasticNetCV
from sklearn.ensemble import GradientBoostingRegressor
```

Lasso Regression

In [62]:
```
lasso = LassoCV(cv=10).fit(X_train, Y_train)


#predicting train
train_preds3=lasso.predict(X_train)
```

```
#predicting on test
test_preds3=lasso.predict(X_test)

RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds3)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds3)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',lasso.score(X_train, Y_train))
print('RSquared value on test:',lasso.score(X_test, Y_test))
```

```
RMSE TrainingData =  0.12196766732315875
RMSE TestData =  0.11809803934941587
--------------------------------------------------
RSquared value on train: 0.3926541120673017
RSquared value on test: 0.4072610148044429
```

Ridge Regression

In [63]:

```
ridge = RidgeCV(cv=10).fit(X_train, Y_train)
#predicting train
train_preds4=ridge.predict(X_train)
#predicting on test
test_preds4=ridge.predict(X_test)

RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds4)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds4)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',ridge.score(X_train, Y_train))
print('RSquared value on test:',ridge.score(X_test, Y_test))
```

```
RMSE TrainingData =  0.12123661279697662
RMSE TestData =  0.11784386021079911
--------------------------------------------------
RSquared value on train: 0.3999129590876712
RSquared value on test: 0.4098097404701734
```

Elastic Net

In [64]:

```
elastic_net = ElasticNetCV(cv = 10).fit(X_train, Y_train)
#predicting train
train_preds5=elastic_net.predict(X_train)
#predicting on test
test_preds5=elastic_net.predict(X_test)

RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds5)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds5)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',elastic_net.score(X_train, Y_train))
print('RSquared value on test:',elastic_net.score(X_test, Y_test))
```

```
RMSE TrainingData =  0.12196899402304891
RMSE TestData =  0.11810830677152079
--------------------------------------------------
RSquared value on train: 0.3926408992196211
RSquared value on test: 0.4071579450846382
```

XG-Boost Regressor

In [65]:
```
xgbr =xgb.XGBRegressor().fit(X_train, Y_train)
#predicting train
train_preds6=xgbr.predict(X_train)
#predicting on test
test_preds6=xgbr.predict(X_test)

RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds6)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds6)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',xgbr.score(X_train, Y_train))
print('RSquared value on test:',xgbr.score(X_test, Y_test))
```

```
RMSE TrainingData =  0.016453210384851504
RMSE TestData =  0.09549969503973747
--------------------------------------------------
RSquared value on train: 0.9889478196861716
RSquared value on test: 0.612401500913349
```

In [66]:
```
errors = abs(test_preds6 - Y_test)
# Calculating errors for using error values in mean absolute percentage error
```

In [67]:
```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / Y_test)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 96.49 %.
```

In [68]:
```
# Before log transformation the xg boost Accuracy was 91.87 % and now accuracy has i
```

In [ ]:
```

```