**PREDICTING IMDB SCORE**

**INNOVATION OF SYSTEM DESIGN**

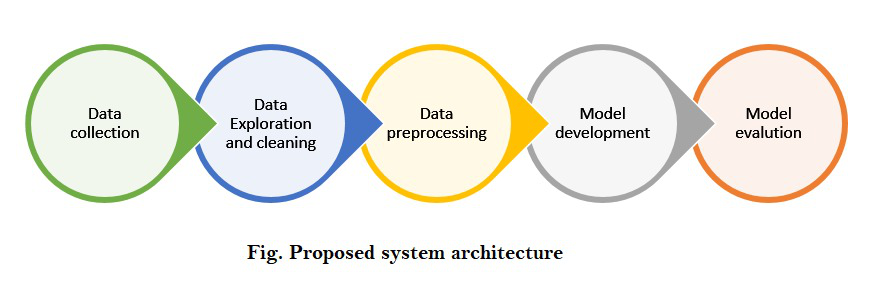
# IPL Score Prediction using Deep Learning

* Since the dawn of the IPL in 2008, it has attracted viewers all around the globe. A high level of uncertainty and last moment nail biters has urged fans to watch the matches. Within a short period, IPL has become the highest revenue-generating league of cricket. In a cricket match, we often see the scoreline showing the probability of the team winning based on the current match situation. This prediction is usually done with the help of Data Analytics. Before when there were no advancements in machine learning, the prediction was usually based on intuitions or some basic algorithms. The above picture clearly tells you how bad is taking run rate as a single factor to predict the final score in a limited-overs cricket match.
* **Tools used:**
* Jupyter Notebook / Google colab
* Visual Studio

### **Technology used:**

* Machine Learning.
* Deep Learning
* Flask (Front-end integration).
* Well, for the smooth running of the project we’ve used few libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and Matplotlib.

### **The architecture of model**



### **Step-by-step implementation:**

First, let’s import all the necessary libraries:

|  |
| --- |
| **import** pandas as pd  **import** numpy as np  **import** matplotlib.pyplot as plt  **import** seaborn as sns  **from** sklearn **import** preprocessing |

**Step 1: Understanding the dataset!**

When dealing with cricket data, Cricsheet is considered as an appropriate platform for gathering the data and thus we took the data from <https://cricsheet.org/downloads/ipl.zip>. It contains data from the year 2007 to 2021. For better accuracy of our model, we used IPL players’ stats to analyze their performance from [here](https://data.world/cclayford/cricinfo-statsguru-data/workspace/file?filename=IPL+Player+Stats+-+2016+till+2019.csv). This dataset contains details of every IPL player from the year 2016 – 2019.

**Step 2: Data cleaning and formatting**

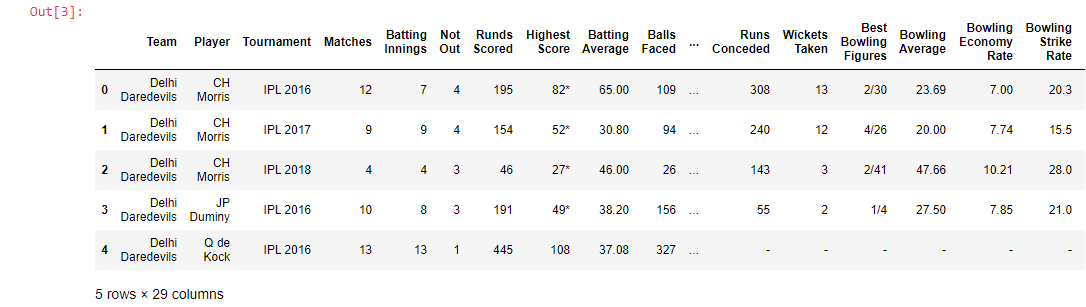
We imported both the datasets using *.read\_csv()* method into a dataframe using pandas and displayed the first 5 rows of each dataset. We did some changes to our dataset like added a new column named “y” which had the runs scored in the first 6 overs from that particular inning.

## Python3

|  |
| --- |
| ipl **=** pd.read\_csv('ipl\_dataset.csv')  ipl.head() |

## Python3

|  |
| --- |
| data **=** pd.read\_csv('IPL Player Stats - 2016 till 2019.csv')  data.head() |



## Python3

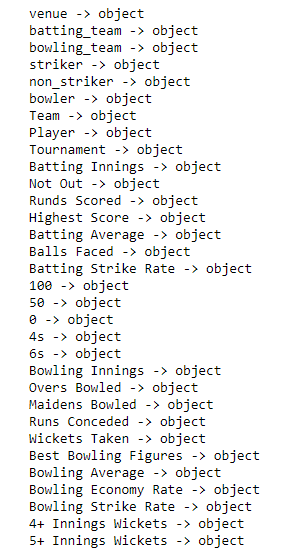
|  |
| --- |
| str\_cols **=** new\_ipl.columns[new\_ipl.dtypes**==**object]  new\_ipl[str\_cols] **=** new\_ipl[str\_cols].fillna('.') |

**Step 3: Encoding the categorical data to numerical values.**

For the columns to be able to assist the model in the prediction, the values should make some sense to the computers. Since they (still) don’t have the ability to understand and draw inferences from the text, we need to encode the strings to numeric categorical values. While we may choose to do the process manually, the **Scikit-learn library** gives us an option to use **LabelEncoder.**

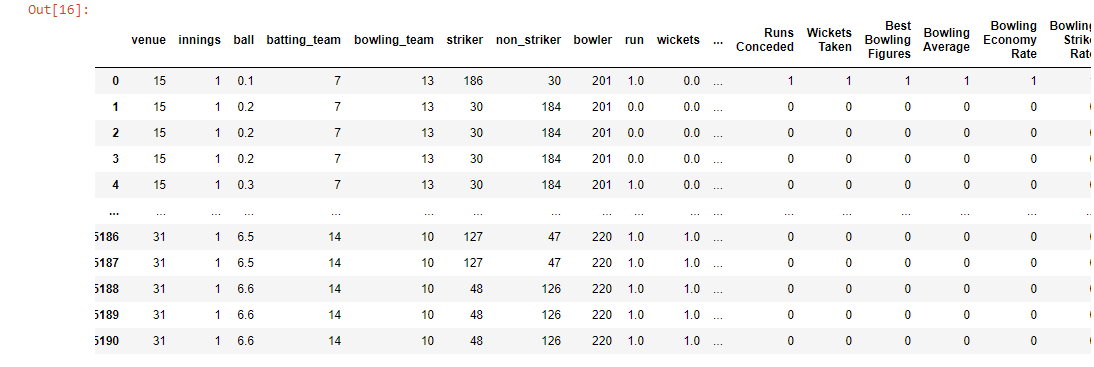
## Python3

|  |
| --- |
| listf **=** []    **for** c **in** new\_ipl.columns:  **if** new\_ipl.dtype**==**object:  **print**(c,"->" ,new\_ipl.dtype)  listf.append(c) |



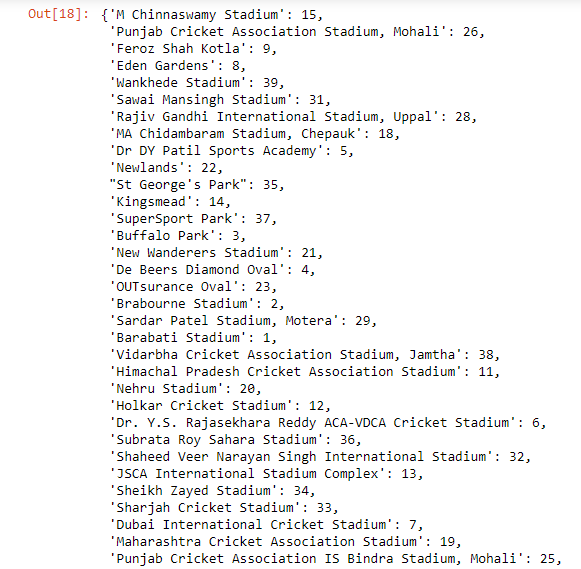
## Python3

|  |
| --- |
| a1 **=** new\_ipl['venue'].unique()  a2 **=** new\_ipl['batting\_team'].unique()  a3 **=** new\_ipl['bowling\_team'].unique()  a4 **=** new\_ipl['striker'].unique()  a5 **=** new\_ipl['bowler'].unique()    **def** labelEncoding(data):  dataset **=** pd.DataFrame(new\_ipl)  feature\_dict **=**{}    **for** feature **in** dataset:  **if** dataset[feature].dtype**==**object:  le **=** preprocessing.LabelEncoder()  fs **=** dataset[feature].unique()  le.fit(fs)  dataset[feature] **=** le.transform(dataset[feature])  feature\_dict[feature] **=** le    **return** dataset    labelEncoding(new\_ipl) |



## Python3

|  |
| --- |
| ip\_dataset **=** new\_ipl[['venue','innings', 'batting\_team',  'bowling\_team', 'striker', 'non\_striker',  'bowler']]    b1 **=** ip\_dataset['venue'].unique()  b2 **=** ip\_dataset['batting\_team'].unique()  b3 **=** ip\_dataset['bowling\_team'].unique()  b4 **=** ip\_dataset['striker'].unique()  b5 **=** ip\_dataset['bowler'].unique()  new\_ipl.fillna(0,inplace**=**True)    features**=**{}    **for** i **in** range(len(a1)):  features[a1[i]]**=**b1[i]  **for** i **in** range(len(a2)):  features[a2[i]]**=**b2[i]  **for** i **in** range(len(a3)):  features[a3[i]]**=**b3[i]  **for** i **in** range(len(a4)):  features[a4[i]]**=**b4[i]  **for** i **in** range(len(a5)):  features[a5[i]]**=**b5[i]    features |



**Step 4: Feature Engineering and Selection**

Our dataset contains multiple columns, but we can’t take these many inputs from users thus we have taken the selected amount of features as input and divided them into X and y. We will then divide our data into train sets and test set before using a machine learning algorithm.

## Python3

|  |
| --- |
| X **=** new\_ipl[['venue', 'innings','batting\_team',  'bowling\_team', 'striker','bowler']].values  y **=** new\_ipl['y'].values    **from** sklearn.model\_selection **import** train\_test\_split    X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(  X, y, test\_size**=**0.33, random\_state**=**42) |

Comparing these large numerical values by our model will be difficult so it is always a better choice to scale your data before processing it. Here we are using **MinMaxScaler** from **sklearn.preprocessing** which is recommended when dealing with deep learning.

## Python3

|  |
| --- |
| **from** sklearn.preprocessing **import** MinMaxScaler  scaler **=** MinMaxScaler()    X\_train **=** scaler.fit\_transform(X\_train)  X\_test **=** scaler.transform(X\_test) |

**Note:** We cannot fit X\_test as it is the data which is to be predicted.

**Step 5: Building, Training & Testing the Model**

Here comes the most exciting part of our project, ***Building our model!*** Firstly, we will import **Sequential** from **tensorflow.keras.models** Also, we will import **Dense & Dropout** from **tensorflow.keras.layers** as we will be using multiple layers.

## Python3

|  |
| --- |
| **from** tensorflow.keras.models **import** Sequential  **from** tensorflow.keras.layers **import** Dense,Dropout  **from** tensorflow.keras.callbacks **import** EarlyStopping |

EarlyStopping is used to avoid overfitting. What early stopping basically does is, it stops calculating the losses when ‘val\_loss’ increases than ‘loss’. Val\_loss curve should always be below val curve. When it is found that the difference between ‘val\_loss’ and ‘loss’ is becomes constant, it stops training.

## Python3

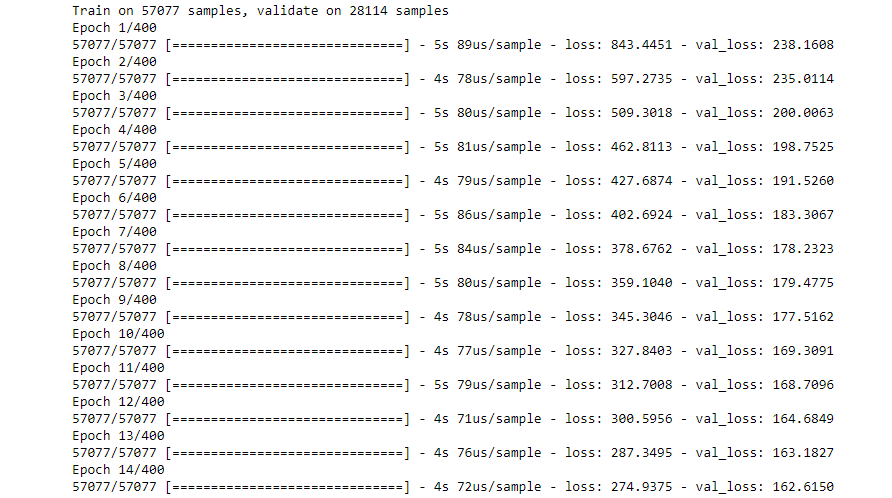
|  |
| --- |
| model **=** Sequential()    model.add(Dense(43, activation**=**'relu'))  model.add(Dropout(0.5))    model.add(Dense(22, activation**=**'relu'))  model.add(Dropout(0.5))    model.add(Dense(11, activation**=**'relu'))  model.add(Dropout(0.5))    model.add(Dense(1))    model.compile(optimizer**=**'adam', loss**=**'mse') |

Here, we have created 2 hidden layers and reduced the number of neurons as we want the final output to be 1. Then while compiling our model we used **adam optimizer** and loss as **mean squared error.**  Now, let’s start training our model with epochs=400.

## Python3

|  |
| --- |
| model.fit(x**=**X\_train, y**=**y\_train, epochs**=**400,  validation\_data**=**(X\_test,y\_test),  callbacks**=**[early\_stop] ) |

It will take some time because of a huge number of samples and epochs and will output the ‘loss’ and ‘val\_loss’ of each sample as below.



IMDB SCORE PREDICTION FOR MOVIES

*Import libraries*   
  
*import* numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from plotnine import \*

**1. Introduction**

**1.1 Background**

This dataset contains the information about the movies . For a movie to be commercial success , it depends on various factors like director, actors ,critic reviews and viewers reaction. Imdb score is one of the important factor to measure the movie's success.

**1.2 Description of dataset attributes**

Please find the details for the datset attributes:-

1. Color :- Movie is black or coloured
2. Director\_name:- Name of the movie director
3. num\_critic\_for\_reviews :- No of critics for the movie
4. duration:- movie duration in minutes
5. director\_facebook\_likes:-Number of likes for the Director on his Facebook Page
6. actor\_3\_facebook\_likes:- No of likes for the actor 3 on his/her facebook Page
7. actor2\_name:- name of the actor 2
8. actor\_1\_facebook\_likes:- No of likes for the actor 1 on his/her facebook Page
9. gross:- Gross earnings of the movie in Dollars
10. genres:- Film categorization like ‘Animation’, ‘Comedy’, ‘Romance’, ‘Horror’, ‘Sci-Fi’, ‘Action’, ‘Family’
11. actor\_1\_name:- Name of the actor 1
12. movie\_title:-Title of the movie
13. num\_voted\_users:-No of people who voted for the movie
14. cast\_total\_facebook\_likes:- Total facebook like for the movie
15. actor\_3\_name:- Name of the actor 3
16. facenumber\_in\_poster:- No of actors who featured in the movie poster
17. plot\_keywords:-Keywords describing the movie plots
18. movie\_imdb\_link:-Link of the movie link
19. num\_user\_for\_reviews:- Number of users who gave a review
20. language:- Language of the movie
21. country:- Country where movie is produced
22. content\_rating:- Content rating of the movie
23. budget:- Budget of the movie in Dollars
24. title\_year:- The year in which the movie is released
25. actor\_2\_facebook\_likes:- facebook likes for the actor 2
26. imdb\_score:- IMDB score of the movie
27. aspect\_ratio :- Aspect ratio the movie was made in
28. movie\_facebook\_likes:- Total no of facebook likes for the movie

**1.3 Case Study**

The dataset here gives the massive information about the movies and their IMDB scores respectively. We are going to analyze each and every factors which can influence the imdb ratings so that we can predict better results.The movie with the higher imdb score is more successful as compared to the movies with low imdb score.

**2. Data Preprocessing**

In [2]:

*#Reading the Data*   
  
*movie\_df*=pd.read\_csv("/kaggle/input/imdb-5000-movie-dataset/movie\_metadata.csv")

In [3]:

*#Displaying the first 10 records*  
  
*movie\_df*.head(10)

Out[3]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | color | director\_name | num\_critic\_for\_reviews | duration | director\_facebook\_likes | actor\_3\_facebook\_likes | actor\_2\_name | actor\_1\_facebook\_likes | gross | genres | ... | num\_user\_for\_reviews | language | country | content\_rating | budget | title\_year | actor\_2\_facebook\_likes | imdb\_score | aspect\_ratio | movie\_facebook\_likes |
| 0 | Color | James Cameron | 723.0 | 178.0 | 0.0 | 855.0 | Joel David Moore | 1000.0 | 760505847.0 | Action|Adventure|Fantasy|Sci-Fi | ... | 3054.0 | English | USA | PG-13 | 237000000.0 | 2009.0 | 936.0 | 7.9 | 1.78 | 33000 |
| 1 | Color | Gore Verbinski | 302.0 | 169.0 | 563.0 | 1000.0 | Orlando Bloom | 40000.0 | 309404152.0 | Action|Adventure|Fantasy | ... | 1238.0 | English | USA | PG-13 | 300000000.0 | 2007.0 | 5000.0 | 7.1 | 2.35 | 0 |
| 2 | Color | Sam Mendes | 602.0 | 148.0 | 0.0 | 161.0 | Rory Kinnear | 11000.0 | 200074175.0 | Action|Adventure|Thriller | ... | 994.0 | English | UK | PG-13 | 245000000.0 | 2015.0 | 393.0 | 6.8 | 2.35 | 85000 |
| 3 | Color | Christopher Nolan | 813.0 | 164.0 | 22000.0 | 23000.0 | Christian Bale | 27000.0 | 448130642.0 | Action|Thriller | ... | 2701.0 | English | USA | PG-13 | 250000000.0 | 2012.0 | 23000.0 | 8.5 | 2.35 | 164000 |
| 4 | NaN | Doug Walker | NaN | NaN | 131.0 | NaN | Rob Walker | 131.0 | NaN | Documentary | ... | NaN | NaN | NaN | NaN | NaN | NaN | 12.0 | 7.1 | NaN | 0 |
| 5 | Color | Andrew Stanton | 462.0 | 132.0 | 475.0 | 530.0 | Samantha Morton | 640.0 | 73058679.0 | Action|Adventure|Sci-Fi | ... | 738.0 | English | USA | PG-13 | 263700000.0 | 2012.0 | 632.0 | 6.6 | 2.35 | 24000 |
| 6 | Color | Sam Raimi | 392.0 | 156.0 | 0.0 | 4000.0 | James Franco | 24000.0 | 336530303.0 | Action|Adventure|Romance | ... | 1902.0 | English | USA | PG-13 | 258000000.0 | 2007.0 | 11000.0 | 6.2 | 2.35 | 0 |
| 7 | Color | Nathan Greno | 324.0 | 100.0 | 15.0 | 284.0 | Donna Murphy | 799.0 | 200807262.0 | Adventure|Animation|Comedy|Family|Fantasy|Musi... | ... | 387.0 | English | USA | PG | 260000000.0 | 2010.0 | 553.0 | 7.8 | 1.85 | 29000 |
| 8 | Color | Joss Whedon | 635.0 | 141.0 | 0.0 | 19000.0 | Robert Downey Jr. | 26000.0 | 458991599.0 | Action|Adventure|Sci-Fi | ... | 1117.0 | English | USA | PG-13 | 250000000.0 | 2015.0 | 21000.0 | 7.5 | 2.35 | 118000 |
| 9 | Color | David Yates | 375.0 | 153.0 | 282.0 | 10000.0 | Daniel Radcliffe | 25000.0 | 301956980.0 | Adventure|Family|Fantasy|Mystery | ... | 973.0 | English | UK | PG | 250000000.0 | 2009.0 | 11000.0 | 7.5 | 2.35 | 10000 |

10 rows × 28 columns

In [4]:

*#Shape of the dataset (no of rows and no of columns)*  
  
*movie\_df*.shape

Out[4]:

(5043, 28)

In [5]:

*#Displaying the data type of the dataset attributes*   
  
*movie\_df*.dtypes

Out[5]:

color object  
director\_name object  
num\_critic\_for\_reviews float64  
duration float64  
director\_facebook\_likes float64  
actor\_3\_facebook\_likes float64  
actor\_2\_name object  
actor\_1\_facebook\_likes float64  
gross float64  
genres object  
actor\_1\_name object  
movie\_title object  
num\_voted\_users int64  
cast\_total\_facebook\_likes int64  
actor\_3\_name object  
facenumber\_in\_poster float64  
plot\_keywords object  
movie\_imdb\_link object  
num\_user\_for\_reviews float64  
language object  
country object  
content\_rating object  
budget float64  
title\_year float64  
actor\_2\_facebook\_likes float64  
imdb\_score float64  
aspect\_ratio float64  
movie\_facebook\_likes int64  
dtype: object

\*\*We can say we have the datset divided into categorical and numeric columns "

**Categorical Columns**

Color,Director name, actor name,genres,movie\_title,language,country,content\_rating.

**Numerical Columns**

num\_critic\_for\_reviews,duration,director\_facebook\_likes ,actor\_3\_facebook\_likes,actor\_1\_facebook\_likes ,gross,num\_voted\_users,cast\_total\_facebook\_likes,facenumber\_in\_poster,num\_user\_for\_reviews ,budget,title\_year, actor\_2\_facebook\_likes ,imdb\_score,aspect\_ratio,movie\_facebook\_likes

In [6]:

*#Five point summary for the numerical columns in the dataset*  
  
*movie\_df*.describe().T

Out[6]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | count | mean | std | min | 25% | 50% | 75% | max |
| num\_critic\_for\_reviews | 4993.0 | 1.401943e+02 | 1.216017e+02 | 1.00 | 50.00 | 110.00 | 195.00 | 8.130000e+02 |
| duration | 5028.0 | 1.072011e+02 | 2.519744e+01 | 7.00 | 93.00 | 103.00 | 118.00 | 5.110000e+02 |
| director\_facebook\_likes | 4939.0 | 6.865092e+02 | 2.813329e+03 | 0.00 | 7.00 | 49.00 | 194.50 | 2.300000e+04 |
| actor\_3\_facebook\_likes | 5020.0 | 6.450098e+02 | 1.665042e+03 | 0.00 | 133.00 | 371.50 | 636.00 | 2.300000e+04 |
| actor\_1\_facebook\_likes | 5036.0 | 6.560047e+03 | 1.502076e+04 | 0.00 | 614.00 | 988.00 | 11000.00 | 6.400000e+05 |
| gross | 4159.0 | 4.846841e+07 | 6.845299e+07 | 162.00 | 5340987.50 | 25517500.00 | 62309437.50 | 7.605058e+08 |
| num\_voted\_users | 5043.0 | 8.366816e+04 | 1.384853e+05 | 5.00 | 8593.50 | 34359.00 | 96309.00 | 1.689764e+06 |
| cast\_total\_facebook\_likes | 5043.0 | 9.699064e+03 | 1.816380e+04 | 0.00 | 1411.00 | 3090.00 | 13756.50 | 6.567300e+05 |
| facenumber\_in\_poster | 5030.0 | 1.371173e+00 | 2.013576e+00 | 0.00 | 0.00 | 1.00 | 2.00 | 4.300000e+01 |
| num\_user\_for\_reviews | 5022.0 | 2.727708e+02 | 3.779829e+02 | 1.00 | 65.00 | 156.00 | 326.00 | 5.060000e+03 |
| budget | 4551.0 | 3.975262e+07 | 2.061149e+08 | 218.00 | 6000000.00 | 20000000.00 | 45000000.00 | 1.221550e+10 |
| title\_year | 4935.0 | 2.002471e+03 | 1.247460e+01 | 1916.00 | 1999.00 | 2005.00 | 2011.00 | 2.016000e+03 |
| actor\_2\_facebook\_likes | 5030.0 | 1.651754e+03 | 4.042439e+03 | 0.00 | 281.00 | 595.00 | 918.00 | 1.370000e+05 |
| imdb\_score | 5043.0 | 6.442138e+00 | 1.125116e+00 | 1.60 | 5.80 | 6.60 | 7.20 | 9.500000e+00 |
| aspect\_ratio | 4714.0 | 2.220403e+00 | 1.385113e+00 | 1.18 | 1.85 | 2.35 | 2.35 | 1.600000e+01 |
| movie\_facebook\_likes | 5043.0 | 7.525965e+03 | 1.932045e+04 | 0.00 | 0.00 | 166.00 | 3000.00 | 3.490000e+05 |

In [7]:

*#Dropping the Imdb link from the dataset*  
  
*movie\_df*.drop('movie\_imdb\_link', axis=1, inplace=True)

In [8]:

*#Removing the color section as most of the movies is colored*  
  
*movie\_df*["color"].value\_counts()  
  
movie\_df.drop('color',axis=1,inplace=True)

In [9]:

*#Checking for the columns present in the datset*  
*movie\_df*.columns

Out[9]:

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',  
 'movie\_title', 'num\_voted\_users', 'cast\_total\_facebook\_likes',  
 'actor\_3\_name', 'facenumber\_in\_poster', 'plot\_keywords',  
 '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 [10]:

*#Checking for the missing values in the dataset*  
  
*movie\_df*.isna().any()

Out[10]:

director\_name True  
num\_critic\_for\_reviews True  
duration True  
director\_facebook\_likes True  
actor\_3\_facebook\_likes True  
actor\_2\_name True  
actor\_1\_facebook\_likes True  
gross True  
genres False  
actor\_1\_name True  
movie\_title False  
num\_voted\_users False  
cast\_total\_facebook\_likes False  
actor\_3\_name True  
facenumber\_in\_poster True  
plot\_keywords True  
num\_user\_for\_reviews True  
language True  
country True  
content\_rating True  
budget True  
title\_year True

actor\_2\_facebook\_likes True  
imdb\_score False  
aspect\_ratio True  
movie\_facebook\_likes False  
dtype: bool

In [11]:

*#No of the missing values in the dataset*  
  
*movie\_df*.isna().sum()

Out[11]:

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  
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 [12]:

*# We can remove the null values from the dataset where the count is less . so that we don't loose much data*   
  
*movie\_df*.dropna(axis=0,subset=['director\_name', 'num\_critic\_for\_reviews','duration','director\_facebook\_likes','actor\_3\_facebook\_likes','actor\_2\_name','actor\_1\_facebook\_likes','actor\_1\_name','actor\_3\_name','facenumber\_in\_poster','num\_user\_for\_reviews','language','country','actor\_2\_facebook\_likes','plot\_keywords'],inplace=True)

In [13]:

movie\_df.shape

Out[13]:

(4737, 26)

**We lost only 6% of the data which is acceptable**

In [14]:

*#Replacing the content rating with Value R as it has highest frequency*  
  
*movie\_df*["content\_rating"].fillna("R", inplace = True)

In [15]:

*#Replacing the aspect\_ratio with the median of the value as the graph is right skewed*   
  
*movie\_df*["aspect\_ratio"].fillna(movie\_df["aspect\_ratio"].median(),inplace=True)

In [16]:

*#We need to replace the value in budget with the median of the value*  
  
*movie\_df*["budget"].fillna(movie\_df["budget"].median(),inplace=True)

In [17]:

*# We need to replace the value in gross with the median of the value*   
  
*movie\_df*['gross'].fillna(movie\_df['gross'].median(),inplace=True)

In [18]:

*# Recheck that all the null values are removed*  
  
*movie\_df*.isna().sum()

Out[18]:

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  
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

**We don't have any null values in the datset anymore**

In [19]:

*#Removing the duplicate values in the datset*  
  
*movie\_df*.drop\_duplicates(inplace=True)  
movie\_df.shape

Out[19]:

(4695, 26)

In [20]:

*#Count of the language values*   
  
*movie\_df*["language"].value\_counts()

Out[20]:

English 4405  
French 69  
Spanish 35  
Hindi 25  
Mandarin 24  
German 18  
Japanese 16  
Russian 11  
Italian 10  
Cantonese 10  
Portuguese 8  
Korean 8  
Danish 5  
Norwegian 4  
Swedish 4  
Hebrew 4  
Dutch 4  
Persian 4  
Arabic 3  
Thai 3  
Indonesian 2  
None 2  
Aboriginal 2  
Dari 2  
Zulu 2  
Hungarian 1  
Mongolian 1  
Greek 1  
Romanian 1  
Bosnian 1  
Telugu 1  
Maya 1  
Polish 1  
Filipino 1  
Czech 1  
Dzongkha 1  
Kazakh 1  
Vietnamese 1  
Icelandic 1  
Aramaic 1  
Name: language, dtype: int64

*# Graphical presentaion*   
*plt*.figure(figsize=(40,10))  
sns.countplot(movie\_df["language"])  
plt.show()

*# Graphical presentaion*   
*plt*.figure(figsize=(40,10))  
sns.countplot(movie\_df["language"])  
plt.show()

*Creating a new column to check the net profit made by the company (Gross-Budget)*   
  
*movie\_df*["Profit"]=movie\_df['budget'].sub(movie\_df['gross'], axis = 0)   
  
movie\_df.head(5)

*Creating a new column to check the profit percentage made by the company*   
  
*Creating a new column to check the profit percentage made by the company*   
  
*movie\_df*['Profit\_Percentage']=(movie\_df["Profit"]/movie\_df["gross"])\*100  
movie\_df

Out[24]:

*#Value counts for the countries*   
  
*value\_counts*=movie\_df["country"].value\_counts()  
print(value\_counts)

*##get top 2 values of index*  
*vals* = value\_counts[:2].index  
print (vals)  
movie\_df['country'] = movie\_df.country.where(movie\_df.country.isin(vals), 'other')

Index(['USA', 'UK'], dtype='object')

In [27]:

linkcode

*#Successfully divided the country into three catogories*   
*movie\_df*["country"].value\_counts()

Out[27]: