imdb_movie

July 20, 2019

Author: Shashi Bala Table of contents

```
<ol>
    <li><a href="#about_dataset">About the dataset</a></li>
    <li><a href="#exploring_the_data">Data Exploration</a></li>
    <li><a href="#cleaning_the_data">Data Cleaning</a></li>
    <li><a href="#transforming_the_data">Data Transformation</a></li>
    <li><a href="#using_parameteric_and_non_parametric_models">Modeling</a></li>
    <li><a href="#recommendations">Recommendation</a></li>
</ol>
```

Import the Following Libraries:

```
<li> <b>numpy (as np)</b> </li>
<li> <b>pandas (as pd)</b> </li>
<li> <b>pandas_profiling</b> </li>
<li> <b>matplotlib.pyplot (as plt)</b> </li>
<li> <b>seaborn (as sns)</b> </li>
<li> <b>warnings</b> </li>
<li> <b>os</b> </li>
<li> <b>series, DataFrame</b> from <b>pandas</b> </li>
<li> <b>stats</b> from <b>scipy.stats</b> </li>
<li> <b>train_test_split</b> from <b>sklearn.model_selection</b> </li>
<li> <b>LinearRegression</b> from <b>sklearn.linear_model</b> </li>
<li> <b>r2_score</b> from <b>sklearn.metrics</b> </li>
<li> <b>statsmodels.api (as sm)</b> </li>
<li> <b>KNeighborsRegressor</b> from <b>sklearn.neighbors</b> </li>
<li> <b>mean_squared_error</b> from <b>sklearn.metrics</b> </li>
<li> <b>neighbors</b> from <b>sklearn</b> </li>
<li> <b>sqrt</b> from <b>math</b> </li>
<li> <b>RandomForestRegressor</b> from <b>sklearn.ensemble</b> </li>
```

```
In [2]: import pandas as pd
        import pandas_profiling
        from pandas import Series, DataFrame
        import numpy as np
        %matplotlib inline
        import matplotlib.pyplot as plt
```

```python
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
import scipy.stats as stats
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import statsmodels.api as sm
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.ensemble import RandomForestRegressor
```

<h2>1. About the dataset</h2>
The dataset is imdb_data_v2 which is in csv file. It contains 38 variables for 5787 movies, spa

<h2>2. Data Exploration</h2>

2.1. Data Load

In [3]:
```python
print(os.listdir("C:\Data Science_Interview\Dataset"))
```

```
['Data Scientist imdb_data_v2.csv']
```

We read in the data we've saved, passing the column names

In [4]:
```python
df = pd.read_csv('C:\Data Science_Interview\Dataset\Data Scientist imdb_data_v2.csv')
```

In [5]:
```python
print('Training data shape: ', df.shape)
```

```
Training data shape:  (5787, 38)
```

Let's check out the first few rows of data

In [6]:
```python
df.head()
```

Out[6]:
```
   id  stock_market_idx  days_since_last_tweet  pre_screen_viewers  \
0   1              1102                     67                  18
1   2              1117                     70                  18
2   3              1000                     90                  11
3   4              1007                     35                  10
4   5              1128                     85                  20

   characters_per_longest_review  priority  \
0                           1181         4
1                           1196         4
2                           1125         4
```

```
3                           1127          4
4                           1072          4

   longest_facebook_comment_review_char  color      director_name  \
0                                   250  Color      James Cameron
1                                   740  Color     Gore Verbinski
2                                  1779  Color         Sam Mendes
3                                  1074  Color  Christopher Nolan
4                                   813    NaN        Doug Walker

   num_critic_for_reviews     ...       country  content_rating  website_score  \
0                   723.0     ...           USA           PG-13            7.9
1                   302.0     ...           USA           PG-13            7.1
2                   602.0     ...            UK           PG-13            6.8
3                   813.0     ...           USA           PG-13            8.5
4                     NaN     ...           NaN             NaN            7.1

        budget  weighted_budget  title_year actor_2_facebook_likes  \
0  237000000.0        236999000      2009.0                  936.0
1  300000000.0        299999000      2007.0                 5000.0
2  245000000.0        244999000      2015.0                  393.0
3  250000000.0        249999000      2012.0                23000.0
4          NaN            -1000         NaN                   12.0

  aspect_ratio movie_facebook_likes  imdb_score
0         1.78                33000         7.9
1         2.35                    0         7.1
2         2.35                85000         6.8
3         2.35               164000         8.5
4          NaN                    0         7.1

[5 rows x 38 columns]
```

0.0.1 Data Profiling

```
In [7]: df.profile_report()
```

```
<IPython.lib.display.IFrame at 0x1fa28223cf8>
```

```
Out[7]:
```

We have 5787 observations of 38 variables in which 21 variables are numeric and 11 variables are categorical. The response variable "imdb_score" is numerical, and the predictors are mixed with numerical and categorical variables.

2.2. Remove Duplicates

In the IMDB dataset, There is 744 (12.9%) duplicate rows. I want to remove the 744 duplicated rows and keep the unique ones.

```
In [8]: df1 = df
```

```
In [9]: #drop the duplicates
        df1.drop_duplicates(inplace=True)
        # Check if done
        df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5043 entries, 0 to 5042
Data columns (total 38 columns):
id                                      5043 non-null int64
stock_market_idx                        5043 non-null int64
days_since_last_tweet                   5043 non-null int64
pre_screen_viewers                      5043 non-null int64
characters_per_longest_review           5043 non-null int64
priority                                5043 non-null int64
longest_facebook_comment_review_char    5043 non-null int64
color                                   5024 non-null object
director_name                           4939 non-null object
num_critic_for_reviews                  4993 non-null float64
duration                                5028 non-null float64
director_facebook_likes                 4939 non-null float64
actor_3_facebook_likes                  5020 non-null float64
actor_2_name                            5030 non-null object
actor_1_facebook_likes                  5036 non-null float64
gross                                   4159 non-null float64
genres                                  5043 non-null object
actor_1_name                            5036 non-null object
movie_title                             5043 non-null object
num_voted_users                         5043 non-null int64
cast_total_facebook_likes               5043 non-null int64
made_up_column                          5043 non-null float64
actor_3_name                            5020 non-null object
facenumber_in_poster                    5030 non-null float64
plot_keywords                           4890 non-null object
movie_imdb_link                         5043 non-null object
num_user_for_reviews                    5022 non-null float64
language                                5031 non-null object
country                                 5038 non-null object
content_rating                          4740 non-null object
website_score                           5043 non-null float64
budget                                  4551 non-null float64
weighted_budget                         5043 non-null int64
title_year                              4935 non-null float64
actor_2_facebook_likes                  5030 non-null float64
aspect_ratio                            4714 non-null float64
movie_facebook_likes                    5043 non-null int64
imdb_score                              5043 non-null float64
```

```
dtypes: float64(15), int64(11), object(12)
memory usage: 1.5+ MB
```

<h2>3. Data Cleaning</h2>

3.1 Missing Values
We can quickly check if we have any null values in our data

```
In [10]: def mis_values(df1):
             mis_value = df1.isnull().sum()
             mis_value_per = 100 * df1.isnull().sum() / len(df1)
             mis_value_column = pd.concat([mis_value, mis_value_per], axis=1)
             mis_val_tab_rename_cols = mis_value_column.rename(columns = {0 : 'Missing Values'
             mis_val_tab_rename_cols = mis_val_tab_rename_cols[mis_val_tab_rename_cols.iloc[:,
             print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"

                    "There are " + str(mis_val_tab_rename_cols.shape[0]) +
                    " cols that have missing values.")
             return mis_val_tab_rename_cols
```

```
In [11]: missing_values = mis_values(df1)
```

```
Your selected dataframe has 38 columns.
There are 21 cols that have missing values.
```

```
In [12]: print(missing_values)
```

	Missing Values	% of Total Missing Values
gross	884	17.5
budget	492	9.8
aspect_ratio	329	6.5
content_rating	303	6.0
plot_keywords	153	3.0
title_year	108	2.1
director_name	104	2.1
director_facebook_likes	104	2.1
num_critic_for_reviews	50	1.0
actor_3_name	23	0.5
actor_3_facebook_likes	23	0.5
num_user_for_reviews	21	0.4
color	19	0.4
duration	15	0.3
facenumber_in_poster	13	0.3
actor_2_name	13	0.3
actor_2_facebook_likes	13	0.3
language	12	0.2
actor_1_name	7	0.1

```
actor_1_facebook_likes                    7                        0.1
country                                   5                        0.1
```

Instead of dropping the rows, I used the median imputation because it's maintain the distribution of the variable.

```
In [13]: # fill missing values with median column values
         df1 = df1.fillna(df1.median())

In [14]: df1

Out[14]:         id  stock_market_idx  days_since_last_tweet  pre_screen_viewers  \
         0        1              1102                     67                  18
         1        2              1117                     70                  18
         2        3              1000                     90                  11
         3        4              1007                     35                  10
         4        5              1128                     85                  20
         5        6              1037                     75                  20
         6        7              1021                     80                  12
         7        8              1133                     30                  16
         8        9              1186                     14                  11
         9       10              1016                      3                  10
         10      11              1142                     99                  13
         11      12              1197                     35                  17
         12      13              1105                     17                  12
         13      14              1145                      8                  10
         14      15              1010                     14                  15
         15      16              1146                     89                  14
         16      17              1188                     12                  18
         17      18              1156                     17                  12
         18      19              1185                     59                  12
         19      20              1127                     77                  11
         20      21              1051                     36                  19
         21      22              1015                     61                  19
         22      23              1200                     75                  14
         23      24              1163                     45                  10
         24      25              1042                     73                  19
         25      26              1142                     96                  16
         26      27              1086                     28                  19
         27      28              1114                     41                  13
         28      29              1197                     66                  16
         29      30              1196                     23                  19
         ...    ...               ...                    ...                 ...
         5013  5014              1081                     57                  10
         5014  5015              1174                     77                  12
         5015  5016              1186                     15                  19
         5016  5017              1136                     59                  11
         5017  5018              1110                     19                  10
```

5018	5019	1040	66	19
5019	5020	1183	43	19
5020	5021	1155	61	11
5021	5022	1027	96	11
5022	5023	1190	13	13
5023	5024	1138	77	12
5024	5025	1055	13	16
5025	5026	1125	97	12
5026	5027	1170	58	10
5027	5028	1157	98	16
5028	5029	1122	69	19
5029	5030	1056	49	20
5030	5031	1193	97	18
5031	5032	1196	31	15
5032	5033	1026	76	20
5033	5034	1177	84	11
5034	5035	1060	59	16
5035	5036	1167	2	10
5036	5037	1062	45	12
5037	5038	1013	57	13
5038	5039	1131	42	19
5039	5040	1016	26	11
5040	5041	1146	33	18
5041	5042	1015	31	11
5042	5043	1048	9	19

	characters_per_longest_review	priority	\
0	1181	4	
1	1196	4	
2	1125	4	
3	1127	4	
4	1072	4	
5	1121	4	
6	1129	4	
7	1164	4	
8	1076	4	
9	1040	4	
10	1191	4	
11	1154	4	
12	1019	4	
13	1045	4	
14	1135	4	
15	1163	4	
16	1066	4	
17	1020	4	
18	1165	4	
19	1142	4	
20	1057	4	

```
21                              1147         4
22                              1103         4
23                              1092         4
24                              1005         4
25                              1134         4
26                              1035         4
27                              1125         4
28                              1158         4
29                              1022         4
...                              ...       ...
5013                            1171         4
5014                            1091         4
5015                            1101         4
5016                            1172         4
5017                            1012         4
5018                            1170         4
5019                            1074         4
5020                            1140         4
5021                            1002         4
5022                            1105         4
5023                            1064         4
5024                            1097         4
5025                            1086         4
5026                            1003         4
5027                            1095         4
5028                            1108         4
5029                            1146         4
5030                            1027         4
5031                            1036         4
5032                            1177         4
5033                            1159         4
5034                            1133         4
5035                            1152         4
5036                            1059         4
5037                            1187         4
5038                            1073         4
5039                            1002         4
5040                            1021         4
5041                            1020         4
5042                            1081         4

      longest_facebook_comment_review_char             color  \
0                                      250             Color
1                                      740             Color
2                                     1779             Color
3                                     1074             Color
4                                      813               NaN
5                                      508             Color
```

6	1189	Color
7	842	Color
8	1860	Color
9	832	Color
10	637	Color
11	1083	Color
12	429	Color
13	1506	Color
14	1389	Color
15	87	Color
16	829	Color
17	1256	Color
18	729	Color
19	737	Color
20	557	Color
21	659	Color
22	1742	Color
23	1770	Color
24	1028	Color
25	590	Color
26	390	Color
27	760	Color
28	1413	Color
29	91	Color
...	...	...
5013	1530	Color
5014	1897	Color
5015	999	Black and White
5016	1724	Color
5017	1022	Color
5018	1434	Color
5019	675	Color
5020	625	NaN
5021	1479	Color
5022	221	Black and White
5023	923	Color
5024	1828	Color
5025	1342	Color
5026	971	Color
5027	1145	Color
5028	311	Black and White
5029	1489	Color
5030	684	Color
5031	1121	Color
5032	1091	Color
5033	1365	Color
5034	181	Color
5035	1497	Color

```
5036                                        373          Color
5037                                        766          Color
5038                                         10          Color
5039                                        539          Color
5040                                       1558          Color
5041                                       1152          Color
5042                                       1587          Color

           director_name  num_critic_for_reviews     ...          country  \
0          James Cameron                   723.0     ...              USA
1         Gore Verbinski                   302.0     ...              USA
2             Sam Mendes                   602.0     ...               UK
3      Christopher Nolan                   813.0     ...              USA
4            Doug Walker                   110.0     ...              NaN
5         Andrew Stanton                   462.0     ...              USA
6              Sam Raimi                   392.0     ...              USA
7           Nathan Greno                   324.0     ...              USA
8            Joss Whedon                   635.0     ...              USA
9            David Yates                   375.0     ...               UK
10           Zack Snyder                   673.0     ...              USA
11          Bryan Singer                   434.0     ...              USA
12          Marc Forster                   403.0     ...               UK
13        Gore Verbinski                   313.0     ...              USA
14        Gore Verbinski                   450.0     ...              USA
15           Zack Snyder                   733.0     ...              USA
16        Andrew Adamson                   258.0     ...              USA
17           Joss Whedon                   703.0     ...              USA
18          Rob Marshall                   448.0     ...              USA
19      Barry Sonnenfeld                   451.0     ...              USA
20         Peter Jackson                   422.0     ...      New Zealand
21             Marc Webb                   599.0     ...              USA
22          Ridley Scott                   343.0     ...              USA
23         Peter Jackson                   509.0     ...              USA
24           Chris Weitz                   251.0     ...              USA
25         Peter Jackson                   446.0     ...      New Zealand
26         James Cameron                   315.0     ...              USA
27        Anthony Russo                    516.0     ...              USA
28            Peter Berg                   377.0     ...              USA
29       Colin Trevorrow                   644.0     ...              USA
...                  ...                     ...     ...              ...
5013          Eric Eason                    28.0     ...              USA
5014            Uwe Boll                    58.0     ...           Canada
5015   Richard Linklater                    61.0     ...              USA
5016     Joseph Mazzella                   110.0     ...              USA
5017         Travis Legge                    1.0     ...              USA
5018       Alex Kendrick                     5.0     ...              USA
5019       Marcus Nispel                    43.0     ...              USA
5020     Brandon Landers                   110.0     ...              USA
```

5021	Jay Duplass	51.0	...	USA
5022	Jim Chuchu	6.0	...	Kenya
5023	Daryl Wein	22.0	...	USA
5024	Jason Trost	42.0	...	USA
5025	John Waters	73.0	...	USA
5026	Olivier Assayas	81.0	...	France
5027	Jafar Panahi	64.0	...	Iran
5028	Ivan Kavanagh	12.0	...	Ireland
5029	Kiyoshi Kurosawa	78.0	...	Japan
5030	Tadeo Garcia	110.0	...	USA
5031	Thomas L. Phillips	13.0	...	USA
5032	Ash Baron-Cohen	10.0	...	USA
5033	Shane Carruth	143.0	...	USA
5034	Neill Dela Llana	35.0	...	Philippines
5035	Robert Rodriguez	56.0	...	USA
5036	Anthony Vallone	110.0	...	USA
5037	Edward Burns	14.0	...	USA
5038	Scott Smith	1.0	...	Canada
5039	NaN	43.0	...	USA
5040	Benjamin Roberds	13.0	...	USA
5041	Daniel Hsia	14.0	...	USA
5042	Jon Gunn	43.0	...	USA

	content_rating	website_score	budget	weighted_budget	title_year	\
0	PG-13	7.9	237000000.0	236999000	2009.0	
1	PG-13	7.1	300000000.0	299999000	2007.0	
2	PG-13	6.8	245000000.0	244999000	2015.0	
3	PG-13	8.5	250000000.0	249999000	2012.0	
4	NaN	7.1	20000000.0	-1000	2005.0	
5	PG-13	6.6	263700000.0	263699000	2012.0	
6	PG-13	6.2	258000000.0	257999000	2007.0	
7	PG	7.8	260000000.0	259999000	2010.0	
8	PG-13	7.5	250000000.0	249999000	2015.0	
9	PG	7.5	250000000.0	249999000	2009.0	
10	PG-13	6.9	250000000.0	249999000	2016.0	
11	PG-13	6.1	209000000.0	208999000	2006.0	
12	PG-13	6.7	200000000.0	199999000	2008.0	
13	PG-13	7.3	225000000.0	224999000	2006.0	
14	PG-13	6.5	215000000.0	214999000	2013.0	
15	PG-13	7.2	225000000.0	224999000	2013.0	
16	PG	6.6	225000000.0	224999000	2008.0	
17	PG-13	8.1	220000000.0	219999000	2012.0	
18	PG-13	6.7	250000000.0	249999000	2011.0	
19	PG-13	6.8	225000000.0	224999000	2012.0	
20	PG-13	7.5	250000000.0	249999000	2014.0	
21	PG-13	7.0	230000000.0	229999000	2012.0	
22	PG-13	6.7	200000000.0	199999000	2010.0	
23	PG-13	7.9	225000000.0	224999000	2013.0	

24	PG-13	6.1	180000000.0	179999000	2007.0
25	PG-13	7.2	207000000.0	206999000	2005.0
26	PG-13	7.7	200000000.0	199999000	1997.0
27	PG-13	8.2	250000000.0	249999000	2016.0
28	PG-13	5.9	209000000.0	208999000	2012.0
29	PG-13	7.0	150000000.0	149999000	2015.0
...	...	...	...	...	...
5013	NaN	7.0	24000.0	23000	2002.0
5014	R	6.3	20000000.0	-1000	2009.0
5015	R	7.1	23000.0	22000	1991.0
5016	NaN	4.8	25000.0	24000	2015.0
5017	NaN	3.3	22000.0	21000	2013.0
5018	NaN	6.9	20000.0	19000	2003.0
5019	R	4.6	20000000.0	-1000	2015.0
5020	NaN	3.0	17350.0	16350	2011.0
5021	R	6.6	15000.0	14000	2005.0
5022	NaN	7.4	15000.0	14000	2014.0
5023	NaN	6.2	15000.0	14000	2009.0
5024	Unrated	4.0	20000.0	19000	2011.0
5025	NC-17	6.1	10000.0	9000	1972.0
5026	R	6.9	4500.0	3500	2004.0
5027	Not Rated	7.5	10000.0	9000	2000.0
5028	NaN	6.7	10000.0	9000	2007.0
5029	NaN	7.4	1000000.0	999000	1997.0
5030	NaN	6.1	20000000.0	-1000	2004.0
5031	NaN	5.4	200000.0	199000	2012.0
5032	NaN	6.4	20000000.0	-1000	1995.0
5033	PG-13	7.0	7000.0	6000	2004.0
5034	Not Rated	6.3	7000.0	6000	2005.0
5035	R	6.9	7000.0	6000	1992.0
5036	PG-13	7.8	3250.0	2250	2005.0
5037	Not Rated	6.4	9000.0	8000	2011.0
5038	NaN	7.7	20000000.0	-1000	2013.0
5039	TV-14	7.5	20000000.0	-1000	2005.0
5040	NaN	6.3	1400.0	400	2013.0
5041	PG-13	6.3	20000000.0	-1000	2012.0
5042	PG	6.6	1100.0	100	2004.0

	actor_2_facebook_likes	aspect_ratio	movie_facebook_likes	imdb_score
0	936.0	1.78	33000	7.9
1	5000.0	2.35	0	7.1
2	393.0	2.35	85000	6.8
3	23000.0	2.35	164000	8.5
4	12.0	2.35	0	7.1
5	632.0	2.35	24000	6.6
6	11000.0	2.35	0	6.2
7	553.0	1.85	29000	7.8
8	21000.0	2.35	118000	7.5

9	11000.0	2.35	10000	7.5
10	4000.0	2.35	197000	6.9
11	10000.0	2.35	0	6.1
12	412.0	2.35	0	6.7
13	5000.0	2.35	5000	7.3
14	2000.0	2.35	48000	6.5
15	3000.0	2.35	118000	7.2
16	216.0	2.35	0	6.6
17	21000.0	1.85	123000	8.1
18	11000.0	2.35	58000	6.7
19	816.0	1.85	40000	6.8
20	972.0	2.35	65000	7.5
21	10000.0	2.35	56000	7.0
22	882.0	2.35	17000	6.7
23	972.0	2.35	83000	7.9
24	6000.0	2.35	0	6.1
25	919.0	2.35	0	7.2
26	14000.0	2.35	26000	7.7
27	19000.0	2.35	72000	8.2
28	10000.0	2.35	44000	5.9
29	2000.0	2.00	150000	7.0
...	...	...	...	...
5013	46.0	1.78	61	7.0
5014	918.0	2.35	0	6.3
5015	0.0	1.37	2000	7.1
5016	25.0	2.35	33	4.8
5017	184.0	1.78	200	3.3
5018	49.0	1.85	725	6.9
5019	512.0	1.85	0	4.6
5020	19.0	2.35	33	3.0
5021	224.0	2.35	297	6.6
5022	19.0	2.35	45	7.4
5023	212.0	2.35	324	6.2
5024	91.0	2.35	835	4.0
5025	143.0	1.37	0	6.1
5026	133.0	2.35	171	6.9
5027	0.0	1.85	697	7.5
5028	5.0	1.33	105	6.7
5029	13.0	1.85	817	7.4
5030	20.0	2.35	22	6.1
5031	98.0	16.00	424	5.4
5032	194.0	2.35	20	6.4
5033	45.0	1.85	19000	7.0
5034	0.0	2.35	74	6.3
5035	20.0	1.37	0	6.9
5036	44.0	2.35	4	7.8
5037	205.0	2.35	413	6.4
5038	470.0	2.35	84	7.7

```
5039                    593.0        16.00                    32000          7.5
5040                      0.0         2.35                       16          6.3
5041                    719.0         2.35                      660          6.3
5042                     23.0         1.85                      456          6.6

[5043 rows x 38 columns]
```

```
In [15]: df1.dtypes.value_counts()
```

```
Out[15]: float64    15
         object     12
         int64      11
         dtype: int64
```

We do! Let's use the "describe" method to find them, amongst other interesting information

```
In [16]: df1.describe()
```

```
Out[16]:                 id  stock_market_idx  days_since_last_tweet  \
         count  5043.000000        5043.00000            5043.000000
         mean   2522.000000        1101.16042              49.908190
         std    1455.933034          58.48476              28.432368
         min       1.000000        1000.00000               1.000000
         25%    1261.500000        1051.00000              26.000000
         50%    2522.000000        1101.00000              49.000000
         75%    3782.500000        1153.00000              74.000000
         max    5043.000000        1200.00000              99.000000

                pre_screen_viewers  characters_per_longest_review  priority  \
         count         5043.000000                    5043.000000    5043.0
         mean            14.979576                    1100.040849       4.0
         std              3.163246                      57.299452       0.0
         min             10.000000                    1000.000000       4.0
         25%             12.000000                    1051.000000       4.0
         50%             15.000000                    1099.000000       4.0
         75%             18.000000                    1149.000000       4.0
         max             20.000000                    1200.000000       4.0

                longest_facebook_comment_review_char  num_critic_for_reviews  \
         count                           5043.000000             5043.000000
         mean                             962.646242              139.894904
         std                              541.710282              121.034214
         min                                6.000000                1.000000
         25%                              511.000000               50.000000
         50%                              962.000000              110.000000
         75%                             1424.000000              194.000000
         max                             1900.000000              813.000000

                   duration  director_facebook_likes     ...         \
```

```
count  5043.000000             5043.000000     ...
mean    107.188578              673.362086     ...
std      25.160972             2785.636586     ...
min       7.000000                0.000000     ...
25%      93.000000                7.000000     ...
50%     103.000000               49.000000     ...
75%     118.000000              189.000000     ...
max     511.000000            23000.000000     ...

       facenumber_in_poster  num_user_for_reviews  website_score  \
count           5043.000000           5043.000000    5043.000000
mean               1.370216            272.284553       6.559925
std                2.011066            377.269873       8.433695
min                0.000000              1.000000       1.600000
25%                0.000000             65.000000       5.800000
50%                1.000000            156.000000       6.600000
75%                2.000000            324.000000       7.200000
max               43.000000           5060.000000     600.000000

             budget  weighted_budget   title_year  actor_2_facebook_likes  \
count  5.043000e+03     5.043000e+03  5043.000000             5043.000000
mean   3.782554e+07     3.587332e+07  2002.531033             1649.030339
std    1.958882e+08     1.961555e+08    12.359307             4037.579765
min    2.180000e+02    -1.000000e+03  1916.000000                0.000000
25%    7.000000e+06     2.999000e+06  1999.000000              281.000000
50%    2.000000e+07     1.499900e+07  2005.000000              595.000000
75%    4.000000e+07     3.999900e+07  2011.000000              918.000000
max    1.221550e+10     1.221550e+10  2045.000000           137000.000000

       aspect_ratio  movie_facebook_likes   imdb_score
count   5043.000000           5043.000000  5043.000000
mean       2.228858           7525.964505     6.559925
std        1.339542          19320.445110     8.433695
min        1.180000              0.000000     1.600000
25%        1.850000              0.000000     5.800000
50%        2.350000            166.000000     6.600000
75%        2.350000           3000.000000     7.200000
max       16.000000         349000.000000   600.000000

[8 rows x 26 columns]
```

I dropped the columns because these columns had high cardinality or many levels and also many zero values.

```
In [17]: # Drop extraneous columns
         col = ['id', 'priority', 'director_name', 'actor_2_name', 'color', 'actor_1_name', 'ac
                'director_name', 'facenumber_in_poster', 'movie_facebook_likes', 'website_scor
                'movie_imdb_link', 'content_rating', 'language', 'plot_keywords', 'cast_total_
```

```
                'director_facebook_likes', 'actor_2_facebook_likes', 'actor_3_facebook_likes',
                'aspect_ratio', 'website_score', 'genres', 'actor_1_facebook_likes']
        df1.drop(col, axis=1, inplace=True)
```

In [18]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5043 entries, 0 to 5042
Data columns (total 13 columns):
stock_market_idx                        5043 non-null int64
days_since_last_tweet                   5043 non-null int64
pre_screen_viewers                      5043 non-null int64
characters_per_longest_review           5043 non-null int64
longest_facebook_comment_review_char    5043 non-null int64
num_critic_for_reviews                  5043 non-null float64
duration                                5043 non-null float64
gross                                   5043 non-null float64
num_voted_users                         5043 non-null int64
made_up_column                          5043 non-null float64
num_user_for_reviews                    5043 non-null float64
budget                                  5043 non-null float64
imdb_score                              5043 non-null float64
dtypes: float64(7), int64(6)
memory usage: 551.6 KB
```

<h2>4. Data Transformation</h2>

To normalize the above variables, I used log transformation.

In [19]:
```
df1['budget'] = np.log(df1.budget)
df1['imdb_score'] = np.log(df1.imdb_score)
df1['made_up_column'] = np.log(df1.imdb_score)
df1['stock_market_idx'] = np.log(df1.stock_market_idx)
df1['days_since_last_tweet'] = np.log(df1.days_since_last_tweet)
df1['pre_screen_viewers'] = np.log(df1.pre_screen_viewers)
df1['characters_per_longest_review'] = np.log(df1.characters_per_longest_review)
df1['longest_facebook_comment_review_char'] = np.log(df1.longest_facebook_comment_revi
df1['num_critic_for_reviews'] = np.log(df1.num_critic_for_reviews)
df1['duration'] = np.log(df1.duration)
df1['gross'] = np.log(df1.gross)
df1['num_voted_users'] = np.log(df1.num_voted_users)
df1['num_user_for_reviews'] = np.log(df1.num_user_for_reviews)
```

In [20]:
```
df1.describe()
```

Out[20]:
```
       stock_market_idx  days_since_last_tweet  pre_screen_viewers  \
count       5043.000000            5043.000000         5043.000000
mean           7.002705               3.630552            2.683450
```

```
std             0.053234             0.913946             0.218117
min             6.907755             0.000000             2.302585
25%             6.957497             3.258097             2.484907
50%             7.003974             3.891820             2.708050
75%             7.050123             4.304065             2.890372
max             7.090077             4.595120             2.995732

       characters_per_longest_review  longest_facebook_comment_review_char  \
count                    5043.000000                           5043.000000
mean                        7.001743                              6.583890
std                         0.052173                              0.944422
min                         6.907755                              1.791759
25%                         6.957497                              6.236370
50%                         7.002156                              6.869014
75%                         7.046647                              7.261225
max                         7.090077                              7.549609

       num_critic_for_reviews     duration        gross  num_voted_users  \
count             5043.000000  5043.000000  5043.000000      5043.000000
mean                 4.467420     4.647760    16.501699        10.096277
std                  1.163810     0.242031     2.125449         1.990129
min                  0.000000     1.945910     5.087596         1.609438
25%                  3.912023     4.532599    15.950977         9.058761
50%                  4.700480     4.634729    17.054875        10.444619
75%                  5.267858     4.770685    17.754313        11.475317
max                  6.700731     6.236370    20.449494        14.340099

       made_up_column  num_user_for_reviews       budget   imdb_score
count     5043.000000           5043.000000  5043.000000  5043.000000
mean         0.605564              4.871133    16.478166     1.845742
std          0.126912              1.392721     1.635744     0.210021
min         -0.755015              0.000000     5.384495     0.470004
25%          0.564096              4.174387    15.761421     1.757858
50%          0.635025              5.049856    16.811243     1.887070
75%          0.680103              5.780744    17.504390     1.974081
max          1.855818              8.529122    23.225971     6.396930
```

In [21]: `df1.profile_report()`

```
<IPython.lib.display.IFrame at 0x1fa2e8a0080>
```

Out[21]:

Some of the columns have missing values. We can deal with this in a few different ways. The simpliest solution is to remove them, though we lose many examples in doing so. Alternatively, we could impute the values, replacing the NaN values with an average (mean or median).

For the purpose of this simple notebook, the variable num_user_for_reviews has 51 zeros which was replaced by median and also remaining numerical variable are imputed by median imputation.

```
In [22]: nonzero_median = df1[ df1.num_user_for_reviews != 0 ].median()

In [23]: df1.loc[ df1.num_user_for_reviews == 0, "num_user_for_reviews" ] = nonzero_median

In [24]: # fill missing values with median column values
         df1 = df1.fillna(df1.median())

In [25]: df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5043 entries, 0 to 5042
Data columns (total 13 columns):
stock_market_idx                        5043 non-null float64
days_since_last_tweet                   5043 non-null float64
pre_screen_viewers                      5043 non-null float64
characters_per_longest_review           5043 non-null float64
longest_facebook_comment_review_char    5043 non-null float64
num_critic_for_reviews                  5043 non-null float64
duration                                5043 non-null float64
gross                                   5043 non-null float64
num_voted_users                         5043 non-null float64
made_up_column                          5043 non-null float64
num_user_for_reviews                    5043 non-null float64
budget                                  5043 non-null float64
imdb_score                              5043 non-null float64
dtypes: float64(13)
memory usage: 551.6 KB
```

<h2>5. Data Modeling</h2>

5.1. Data Splitting

0.0.2 Test & Train Split

The purpose of splitting the data is to be able to assess the quality of a predictive model when it is used on unseen data. When training, you will try to build a model that fits to the data as closely as possible, to be able to most accurately make a prediction. However, without a test set you run the risk of overfitting - the model works very well for the data it has seen but not for new data.

The split ratio is often debated and in practice you might split your data into three sets: train, validation and test. You would use the training data to understand which classifier you wish to use; the validation set to test on whilst tweaking parameters; and the test set to get an understanding of how your final model would work in practice. Furthermore, there are techniques such as K-Fold cross validation that also help to reduce bias.

For the purpose of this demonstration, we will only be randomly splitting our data into test and train, with a 80/20 split.

We import the required library from scikit-learn, train_test_split

```
In [26]: X = df1.iloc[:,0:12].values
         y = df1.iloc[:,12:13].values
```

```
In [27]: X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.20,
```

```
In [28]: print(X_trainset.shape)
         print(y_trainset.shape)
```

```
(4034, 12)
(4034, 1)
```

```
In [29]: print(X_testset.shape)
         print(y_testset.shape)
```

```
(1009, 12)
(1009, 1)
```

<h2> Parametric Machine Learning Algorithm</h2>

5.2. Using Linear Regression

Linear regression attempts to fit a straight hyperplane to your dataset that is closest to all data points. It is most suitable when there are linear relationships between the variables in the dataset.

```
In [30]: reg = LinearRegression()
         reg.fit(X, y)
```

```
Out[30]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
In [31]: y_pred1 = reg.predict(X_testset)
```

5.2.1. Performance Metrics of Linear Regression

We wish to understand how good our model is; there are a few different metrics we can use. We will evaluate mean squared error (MSE) and mean absolute error (MAE)

We import scikit-learn's mean squared error and sckit-learn's mean absolute error

```
In [32]: from sklearn import metrics
         print('Mean Absolute Error:', metrics.mean_absolute_error(y_testset, y_pred1))
         print('Mean Squared Error:', metrics.mean_squared_error(y_testset, y_pred1))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_testset, y_pre
```

```
Mean Absolute Error: 0.018131519955082531
Mean Squared Error: 0.0014731432685910185
Root Mean Squared Error: 0.03838154854342147
```

5.3. Using Ridge Regression

```
In [33]: from sklearn.model_selection import GridSearchCV
         from sklearn.linear_model import Ridge
```

```
In [34]: ridge = Ridge()
         parameters = {'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 5, 10, 20]}
         ridge_regressor = GridSearchCV(ridge, parameters, scoring='mean_squared_error')

         ridge_regressor.fit(X, y)
```

```
Out[34]: GridSearchCV(cv=None, error_score='raise',
                estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
            normalize=False, random_state=None, solver='auto', tol=0.001),
                fit_params=None, iid=True, n_jobs=1,
                param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.0001, 0.001, 0.01, 1, 5, 10, 20]}
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='mean_squared_error', verbose=0)
```

```
In [35]: print(ridge_regressor.best_params_)
         print(ridge_regressor.best_score_)
```

```
{'alpha': 1}
-0.00248930108763342
```

In this case, the optimal value for alpha is 1, and the negative MSE is -0.0024893.

5.4. Using LASSO Regression

```
In [36]: from sklearn.linear_model import Lasso
```

```
In [37]: lasso = Lasso()
         parameters = {'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 5, 10, 20]}
         lasso_regressor = GridSearchCV(lasso, parameters, scoring='mean_squared_error')

         lasso_regressor.fit(X, y)
```

```
Out[37]: GridSearchCV(cv=None, error_score='raise',
                estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
            normalize=False, positive=False, precompute=False, random_state=None,
            selection='cyclic', tol=0.0001, warm_start=False),
                fit_params=None, iid=True, n_jobs=1,
                param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.0001, 0.001, 0.01, 1, 5, 10, 20]}
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='mean_squared_error', verbose=0)
```

```
In [38]: print(lasso_regressor.best_params_)
         print(lasso_regressor.best_score_)
```

```
{'alpha': 0.0001}
-0.0024895672740462694
```

In this case, the optimal value for alpha is 0.0001, and the negative MSE is -0.0024895.

Note: After use linear, lasso, and ridge regression. We have seen that ridge is the best fitting method, with a regularization value of 1.

<h2>Nonparametric ML Algorithms</h2>

5.5. Using KNRegressor

In [39]:
```
clf=KNeighborsRegressor(5)
clf.fit(X_trainset,y_trainset)
y_pred=clf.predict(X_testset)
print(mean_squared_error(y_testset,y_pred))
```

```
0.031100714262353453
```

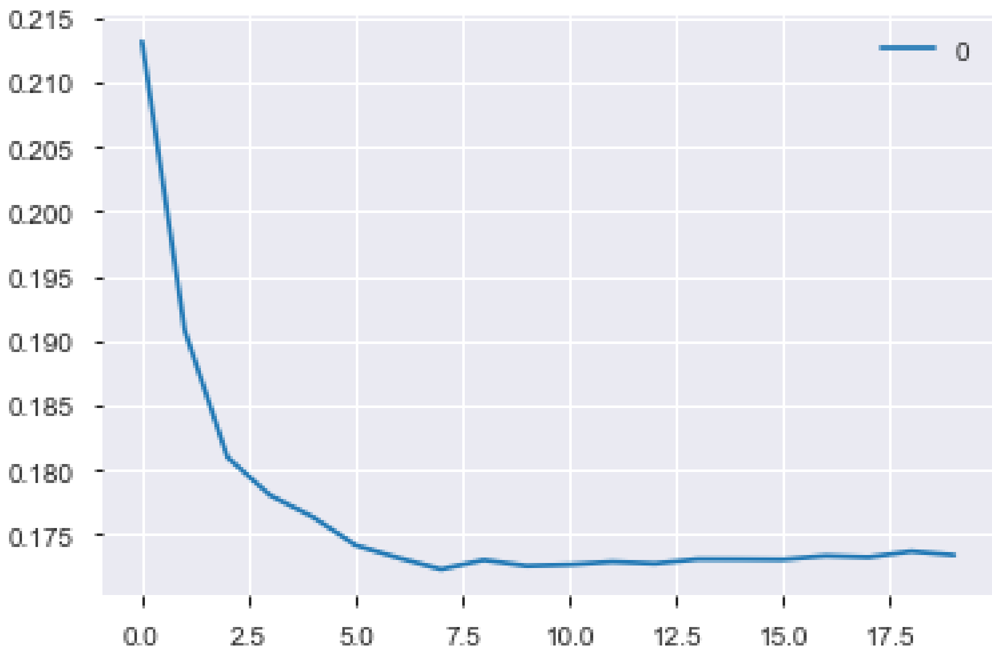
In [42]:
```
from sklearn import neighbors
rmse_val = [] #to store rmse values for different k
for K in range(20):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)

    model.fit(X_trainset, y_trainset)  #fit the model
    pred=model.predict(X_testset) #make prediction on test set
    error = sqrt(mean_squared_error(y_testset,pred)) #calculate rmse
    rmse_val.append(error) #store rmse values
    print('RMSE value for k= ' , K , 'is:', error)
```

```
RMSE value for k=  1 is: 0.21315240080835277
RMSE value for k=  2 is: 0.1908013268343305
RMSE value for k=  3 is: 0.18098856364713534
RMSE value for k=  4 is: 0.17803795746340828
RMSE value for k=  5 is: 0.17635394597896994
RMSE value for k=  6 is: 0.17416175135818351
RMSE value for k=  7 is: 0.1731860406522612
RMSE value for k=  8 is: 0.17230452241536612
RMSE value for k=  9 is: 0.17302494104578212
RMSE value for k=  10 is: 0.17256339138196394
RMSE value for k=  11 is: 0.17266004948251795
RMSE value for k=  12 is: 0.1728858069932387
RMSE value for k=  13 is: 0.17274587714990983
RMSE value for k=  14 is: 0.1730896798547398
RMSE value for k=  15 is: 0.17308010745515268
RMSE value for k=  16 is: 0.1730563602443598
RMSE value for k=  17 is: 0.17335005398973438
RMSE value for k=  18 is: 0.17323566744883404
RMSE value for k=  19 is: 0.1736963168112529
RMSE value for k=  20 is: 0.17340505276908819
```

In [43]:
```
#plotting the rmse values against k values
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa2e148588>



Comparing with other Number of clusters, k=8 and RMSE is 0.172 is better.
5.6. Using of Random Forest

```
In [44]: regressor = RandomForestRegressor(n_estimators=20, random_state=0)
         regressor.fit(X_trainset, y_trainset)
         y_pred = regressor.predict(X_testset)
```

```
In [45]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_testset, y_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(y_testset, y_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_testset, y_pre
```

```
Mean Absolute Error: 0.00044979607149377013
Mean Squared Error: 4.4625280188515924e-05
Root Mean Squared Error: 0.006680215579494117
```

When estimator is 20 then RMSE is 0.007 and MSE is 4.46.
Note:
After comparing non-parametric models RMSE, Random forest performed better.

0.0.3 Stepwise Selection

Used backward selection method for multiple regression to find better R-square and all p-value of variable should be significantly significant (less than 0.05)

```
In [46]: X = np.column_stack((df1['budget'], df1['stock_market_idx'], df1['days_since_last_twe
                              df1['characters_per_longest_review'], df1['longest_facebook_comme
                              df1['num_critic_for_reviews'], df1['duration'], df1['gross'], df1
                              df1['made_up_column'], df1['num_user_for_reviews']))
         y = df1['imdb_score']
         X2 = sm.add_constant(X)
         est = sm.OLS(y, X2)
         est2 = est.fit()
         print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              imdb_score   R-squared:                       0.948
Model:                             OLS   Adj. R-squared:                  0.948
Method:                  Least Squares   F-statistic:                     7691.
Date:                 Sat, 20 Jul 2019   Prob (F-statistic):               0.00
Time:                         17:39:57   Log-Likelihood:                 8184.7
No. Observations:                 5043   AIC:                        -1.634e+04
Df Residuals:                     5030   BIC:                        -1.626e+04
Df Model:                           12
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.0315      0.129      7.993      0.000       0.778       1.284
x1            -0.0032      0.001     -6.456      0.000      -0.004      -0.002
x2            -0.0086      0.013     -0.678      0.498      -0.033       0.016
x3            -0.0001      0.001     -0.139      0.890      -0.002       0.001
x4            -0.0003      0.003     -0.106      0.915      -0.006       0.006
x5            -0.0137      0.013     -1.059      0.290      -0.039       0.012
x6            -0.0008      0.001     -1.065      0.287      -0.002       0.001
x7            -0.0059      0.001     -5.906      0.000      -0.008      -0.004
x8             0.0068      0.003      2.282      0.023       0.001       0.013
x9         -5.332e-05      0.000     -0.143      0.886      -0.001       0.001
x10            0.0041      0.001      5.531      0.000       0.003       0.006
x11            1.5940      0.006    279.342      0.000       1.583       1.605
x12            0.0037      0.001      3.635      0.000       0.002       0.006
==============================================================================
Omnibus:                    13727.176   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        583187610.018
Skew:                          33.242   Prob(JB):                         0.00
Kurtosis:                    1667.636   Cond. No.                     5.72e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.72e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [47]: col = ['pre_screen_viewers']
         df1.drop(col, axis=1, inplace=True)
```

```
In [48]: X = np.column_stack((df1['budget'], df1['stock_market_idx'], df1['days_since_last_twe
                             df1['characters_per_longest_review'], df1['longest_facebook_comme
                             df1['num_critic_for_reviews'], df1['duration'], df1['gross'], df1
                             df1['made_up_column'], df1['num_user_for_reviews']))
         y = df1['imdb_score']
         X2 = sm.add_constant(X)
         est = sm.OLS(y, X2)
         est2 = est.fit()
         print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             imdb_score   R-squared:                       0.948
Model:                            OLS   Adj. R-squared:                  0.948
Method:                 Least Squares   F-statistic:                     8391.
Date:                Sat, 20 Jul 2019   Prob (F-statistic):               0.00
Time:                        17:39:59   Log-Likelihood:                 8184.7
No. Observations:                5043   AIC:                        -1.635e+04
Df Residuals:                    5031   BIC:                        -1.627e+04
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.0307      0.129      8.001      0.000       0.778       1.283
x1            -0.0032      0.001     -6.456      0.000      -0.004      -0.002
x2            -0.0086      0.013     -0.678      0.498      -0.033       0.016
x3            -0.0001      0.001     -0.140      0.889      -0.002       0.001
x4            -0.0137      0.013     -1.060      0.289      -0.039       0.012
x5            -0.0008      0.001     -1.066      0.287      -0.002       0.001
x6            -0.0059      0.001     -5.906      0.000      -0.008      -0.004
x7             0.0068      0.003      2.282      0.023       0.001       0.013
x8          -5.33e-05      0.000     -0.143      0.886      -0.001       0.001
x9             0.0041      0.001      5.533      0.000       0.003       0.006
x10            1.5940      0.006    279.385      0.000       1.583       1.605
x11            0.0037      0.001      3.634      0.000       0.002       0.006
==============================================================================
Omnibus:                    13727.412   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        583262636.992
Skew:                          33.244   Prob(JB):                         0.00
Kurtosis:                    1667.743   Cond. No.                     5.69e+03
==============================================================================
```

```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.69e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [49]: col = ['gross']
         df1.drop(col, axis=1, inplace=True)
```

```
In [50]: X = np.column_stack((df1['budget'], df1['stock_market_idx'], df1['days_since_last_twe
                              df1['characters_per_longest_review'], df1['longest_facebook_comme
                              df1['num_critic_for_reviews'], df1['duration'], df1['num_voted_us
                              df1['made_up_column'], df1['num_user_for_reviews']))
         y = df1['imdb_score']
         X2 = sm.add_constant(X)
         est = sm.OLS(y, X2)
         est2 = est.fit()
         print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             imdb_score   R-squared:                       0.948
Model:                            OLS   Adj. R-squared:                  0.948
Method:                 Least Squares   F-statistic:                     9232.
Date:                Sat, 20 Jul 2019   Prob (F-statistic):               0.00
Time:                        17:40:00   Log-Likelihood:                 8184.7
No. Observations:                5043   AIC:                        -1.635e+04
Df Residuals:                    5032   BIC:                        -1.628e+04
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.0306      0.129      8.002      0.000       0.778       1.283
x1            -0.0033      0.000     -6.673      0.000      -0.004      -0.002
x2            -0.0086      0.013     -0.681      0.496      -0.033       0.016
x3            -0.0001      0.001     -0.138      0.890      -0.002       0.001
x4            -0.0137      0.013     -1.063      0.288      -0.039       0.012
x5            -0.0008      0.001     -1.067      0.286      -0.002       0.001
x6            -0.0058      0.001     -5.953      0.000      -0.008      -0.004
x7             0.0069      0.003      2.291      0.022       0.001       0.013
x8             0.0041      0.001      5.572      0.000       0.003       0.006
x9             1.5940      0.006    280.066      0.000       1.583       1.605
x10            0.0037      0.001      3.667      0.000       0.002       0.006
==============================================================================
Omnibus:                    13726.670   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        582997176.363
Skew:                          33.239   Prob(JB):                         0.00
```

```
Kurtosis:                    1667.364   Cond. No.                     4.71e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.71e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [51]: col = ['days_since_last_tweet']
         df1.drop(col, axis=1, inplace=True)
```

```
In [52]: X = np.column_stack((df1['budget'], df1['stock_market_idx'],
                              df1['characters_per_longest_review'], df1['longest_facebook_comme
                              df1['num_critic_for_reviews'], df1['duration'], df['num_voted_use
                              df1['made_up_column'], df1['num_user_for_reviews']))
         y = df1['imdb_score']
         X2 = sm.add_constant(X)
         est = sm.OLS(y, X2)
         est2 = est.fit()
         print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             imdb_score   R-squared:                       0.949
Model:                            OLS   Adj. R-squared:                  0.949
Method:                 Least Squares   F-statistic:                 1.043e+04
Date:                Sat, 20 Jul 2019   Prob (F-statistic):               0.00
Time:                        17:40:01   Log-Likelihood:                 8223.5
No. Observations:                5043   AIC:                        -1.643e+04
Df Residuals:                    5033   BIC:                        -1.636e+04
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.0974      0.128      8.583      0.000       0.847       1.348
x1            -0.0028      0.000     -6.085      0.000      -0.004      -0.002
x2            -0.0113      0.013     -0.897      0.370      -0.036       0.013
x3            -0.0151      0.013     -1.177      0.239      -0.040       0.010
x4            -0.0007      0.001     -1.023      0.307      -0.002       0.001
x5            -0.0035      0.001     -4.009      0.000      -0.005      -0.002
x6             0.0036      0.003      1.209      0.227      -0.002       0.009
x7           6.57e-08   6.27e-09     10.472      0.000    5.34e-08     7.8e-08
x8             1.5878      0.006    279.653      0.000       1.577       1.599
x9             0.0033      0.001      3.751      0.000       0.002       0.005
==============================================================================
Omnibus:                    13897.771   Durbin-Watson:                   1.989
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         640334751.739
Skew:                          34.339   Prob(JB):                          0.00
Kurtosis:                    1747.329   Cond. No.                      3.11e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.11e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [53]: col = ['stock_market_idx']
         df1.drop(col, axis=1, inplace=True)

In [54]: X = np.column_stack((df1['budget'],
                              df1['characters_per_longest_review'], df1['longest_facebook_comme
                              df1['num_critic_for_reviews'], df1['duration'], df1['num_voted_us
                              df1['made_up_column'], df1['num_user_for_reviews']))
         y = df1['imdb_score']
         X2 = sm.add_constant(X)
         est = sm.OLS(y, X2)
         est2 = est.fit()
         print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             imdb_score   R-squared:                       0.948
Model:                            OLS   Adj. R-squared:                  0.948
Method:                 Least Squares   F-statistic:                 1.154e+04
Date:                Sat, 20 Jul 2019   Prob (F-statistic):               0.00
Time:                        17:40:01   Log-Likelihood:                 8184.5
No. Observations:                5043   AIC:                        -1.635e+04
Df Residuals:                    5034   BIC:                        -1.629e+04
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.9687      0.091     10.589      0.000       0.789       1.148
x1            -0.0033      0.000     -6.688      0.000      -0.004      -0.002
x2            -0.0136      0.013     -1.049      0.294      -0.039       0.012
x3            -0.0008      0.001     -1.076      0.282      -0.002       0.001
x4            -0.0059      0.001     -5.975      0.000      -0.008      -0.004
x5             0.0068      0.003      2.284      0.022       0.001       0.013
x6             0.0041      0.001      5.605      0.000       0.003       0.006
x7             1.5940      0.006    280.140      0.000       1.583       1.605
x8             0.0037      0.001      3.668      0.000       0.002       0.006
==============================================================================
```

```
Omnibus:                    13728.449   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        583620066.860
Skew:                          33.250   Prob(JB):                         0.00
Kurtosis:                    1668.253   Cond. No.                     3.18e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.18e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [55]: col = ['characters_per_longest_review']
         df1.drop(col, axis=1, inplace=True)
```

```
In [56]: X = np.column_stack((df1['budget'],
                              df1['longest_facebook_comment_review_char'],
                              df1['num_critic_for_reviews'], df1['duration'], df1['num_voted_u
                              df1['made_up_column'], df1['num_user_for_reviews']))
         y = df1['imdb_score']
         X2 = sm.add_constant(X)
         est = sm.OLS(y, X2)
         est2 = est.fit()
         print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             imdb_score   R-squared:                       0.948
Model:                            OLS   Adj. R-squared:                  0.948
Method:                 Least Squares   F-statistic:                 1.319e+04
Date:                Sat, 20 Jul 2019   Prob (F-statistic):               0.00
Time:                        17:40:02   Log-Likelihood:                 8183.9
No. Observations:                5043   AIC:                        -1.635e+04
Df Residuals:                    5035   BIC:                        -1.630e+04
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.8740      0.015     59.630      0.000       0.845       0.903
x1            -0.0032      0.000     -6.657      0.000      -0.004      -0.002
x2            -0.0008      0.001     -1.110      0.267      -0.002       0.001
x3            -0.0058      0.001     -5.962      0.000      -0.008      -0.004
x4             0.0068      0.003      2.268      0.023       0.001       0.013
x5             0.0041      0.001      5.598      0.000       0.003       0.006
x6             1.5938      0.006    280.199      0.000       1.583       1.605
x7             0.0037      0.001      3.666      0.000       0.002       0.006
==============================================================================
```

```
Omnibus:                    13731.611   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        584547592.701
Skew:                          33.270   Prob(JB):                         0.00
Kurtosis:                    1669.576   Cond. No.                         489.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In [57]: col = ['longest_facebook_comment_review_char']
         df1.drop(col, axis=1, inplace=True)
```

```
In [58]: X = np.column_stack((df1['budget'], df1['num_critic_for_reviews'], df1['duration'], d
                              df1['made_up_column'], df1['num_user_for_reviews']))
         y = df1['imdb_score']
         X2 = sm.add_constant(X)
         est = sm.OLS(y, X2)
         est2 = est.fit()
         print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             imdb_score   R-squared:                       0.948
Model:                            OLS   Adj. R-squared:                  0.948
Method:                 Least Squares   F-statistic:                 1.539e+04
Date:                Sat, 20 Jul 2019   Prob (F-statistic):               0.00
Time:                        17:40:04   Log-Likelihood:                 8183.3
No. Observations:                5043   AIC:                        -1.635e+04
Df Residuals:                    5036   BIC:                        -1.631e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.8689      0.014     62.356      0.000       0.842       0.896
x1            -0.0032      0.000     -6.660      0.000      -0.004      -0.002
x2            -0.0058      0.001     -5.964      0.000      -0.008      -0.004
x3             0.0068      0.003      2.263      0.024       0.001       0.013
x4             0.0042      0.001      5.614      0.000       0.003       0.006
x5             1.5937      0.006    280.208      0.000       1.583       1.605
x6             0.0036      0.001      3.643      0.000       0.002       0.006
==============================================================================
Omnibus:                    13733.185   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        585152979.355
Skew:                          33.280   Prob(JB):                         0.00
Kurtosis:                    1670.440   Cond. No.                         445.
==============================================================================
```

```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Note: The target variable or dependent variable and independent variables were continuous. I used lof transformation to normalize the variables. After verifing Linear Regression assumptions, I move forward with predictive modeling and got adjusted R-square of 95%. After using backward selection technique, the remaining independent variables p-value is < 0.05 which shows that variables are significant. RMSE is also less which is 0.04.

After checking RMSE of the other algorithm, Random Forset root mean squared error is less which is 0.007. It shows that random forset perform better comparing with other models.

<h2> Modules</h2>

```
In [59]: from sklearn import preprocessing
         from sklearn.metrics import precision_recall_fscore_support
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix
```

<h2>Creating new dataframe as df2</h2>

```
In [60]: df2 = df
```

<h2> Clean & Transform</h2>
The target variable is numeric and have duplicates. My goal is to build a model, which can hel

```
In [61]: ### data clean up/ transformation
         # 1 = Good and 0 = when Bad
         df2.loc[df.imdb_score < 8.0, 'imdb_score'] = 0
         df2.loc[df.imdb_score >= 8.0, 'imdb_score'] = 1
```

```
In [62]: df2.imdb_score.unique()
```

```
Out[62]: array([0., 1.])
```

Now, imdb_score is dichotomous now.

```
In [63]: #create dummy variables
         df2 = pd.get_dummies(df2, columns = ['color', 'director_name',
                'actor_2_name', 'genres', 'actor_1_name', 'movie_title', 'actor_3_name',
                'plot_keywords', 'language', 'country', 'content_rating'], drop_first = True)
```

Some of the columns have missing values. We can deal with this in a few different ways. The simpliest solution is to remove them, though we lose many examples in doing so. Alternatively, we could impute the values, replacing the NaN values with an average (mean or median). For the purpose of this simple notebook, we will simply remove them.

```
In [64]: #drop columns
         df2 = df2.drop(columns=['cast_total_facebook_likes', 'made_up_column', 'priority', 'we
                                'movie_imdb_link']).drop_duplicates()
```

```
In [65]: def calculate_metrics(y_true,y_pred):
             print(precision_recall_fscore_support(y_true, y_pred,average='macro'))
             print(accuracy_score(y_true, y_pred))
             print(confusion_matrix(y_true, y_pred,labels=[1,0]))

         #df = pd.read_excel("../data/dataset_exercise.xlsx",header=0)
         #df = df.drop("id",axis=1)
         target = df2["imdb_score"]
         df2_x = df2.drop("imdb_score",axis=1)

         df2_x = df2_x.dropna(thresh=int(len(df2_x)*0.5), axis=1)
         print(df2_x.shape)
         df2_x = df2_x.fillna(df2_x.mean())

         x = df2_x.values #returns a numpy array
         min_max_scaler = preprocessing.MinMaxScaler()
         # stand = preprocessing.StandardScaler()
         x_scaled = min_max_scaler.fit_transform(x)
         df2_x_pre = pd.DataFrame(x_scaled)
```

```
(5043, 21780)
```

```
In [66]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import chi2
         bestfeatures = SelectKBest(score_func=chi2, k=10)
         fit = bestfeatures.fit(df2_x_pre,target)
         df2scores = pd.DataFrame(fit.scores_)
         df2columns = pd.DataFrame(df2_x_pre.columns)
         #concat two dataframes for better visualization
         featureScores = pd.concat([df2columns,df2scores],axis=1)
         df2_x_backup = df2_x_pre.copy()
```

```
In [67]: df2_x_pre = df2_x_backup.copy()
         # df_new =
         conf_sum = 0
         index = 0
         for i in df2scores.values:
             if i[0]<0.05:
                 conf_sum+=i[0]
                 df2_x_pre = df2_x_pre.drop(df2_x_pre.columns[index], axis=1)
                 index-=1
             index+=1

         df2_x_pre.shape
```

```
Out[67]: (5043, 21745)
```

<h2>2. Split Data on df2 dataset</h2>

```
In [68]: x_train, x_test, y_train, y_test = train_test_split(df2_x_pre, target, test_size=0.20

         print(x_train.shape)
         print(x_test.shape)
         print(y_train.shape)
         print(y_test.shape)

         print("testing data=")
         print("1=",np.sum(y_test))
         print("0=",len(y_test)-np.sum(y_test))
```

```
(4034, 21745)
(1009, 21745)
(4034,)
(1009,)
testing data=
1= 72.0
0= 937.0
```

<h2> Logistic Regression</h2> (Parametric Model)

```
In [69]: from sklearn.linear_model import LogisticRegression
         clf1 = LogisticRegression(random_state=0, solver='lbfgs').fit(x_train, y_train)
         y_pred= clf1.predict(x_test)

         calculate_metrics(y_test,y_pred)
```

```
(0.8927698032961191, 0.6095102573224238, 0.6605154057151981, None)
0.9415262636273538
[[ 16  56]
 [  3 934]]
```

<h2> Performance Matrix of Logistic Regression </h2>

```
In [70]: from sklearn.metrics import classification_report
         predictions = clf1.predict(x_test)
         print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

        0.0       0.94      1.00      0.97       937
        1.0       0.84      0.22      0.35        72

avg / total       0.94      0.94      0.93      1009
```

In [71]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(y_test,predictions)
```

Out[71]: 0.9415262636273538

<h2> K Nearest Neighbor</h2> (Nonparametric Model)

In [72]:
```python
from sklearn.neighbors import KNeighborsClassifier
clf2 = KNeighborsClassifier(n_neighbors=3).fit(x_train, y_train)
y_pred= clf2.predict(x_test)
calculate_metrics(y_test,y_pred)
```

```
(0.7628676470588236, 0.5657091189375074, 0.594474636098345, None)
0.931615460852329
[[ 10  62]
 [  7 930]]
```

<h2> Performance Matrix of KNN </h2>

In [73]:
```python
predict2 = clf2.predict(x_test)
print(classification_report(y_test,predict2))
```

```
             precision    recall  f1-score   support

        0.0       0.94      0.99      0.96       937
        1.0       0.59      0.14      0.22        72

avg / total       0.91      0.93      0.91      1009
```

In [74]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(y_test,predict2)
```

Out[74]: 0.931615460852329

<h2> Gaussian Process Classifier</h2> (Nonparametric Model)

In [75]:
```python
from sklearn.gaussian_process import GaussianProcessClassifier
clf3 = GaussianProcessClassifier(random_state=0).fit(x_train, y_train)
y_pred= clf3.predict(x_test)
calculate_metrics(y_test,y_pred)
```

```
(0.9652432969215492, 0.5138888888888888, 0.5090229118006895, None)
0.9306243805748265
[[  2  70]
 [  0 937]]
```

<h2> Performance Matrix of Gaussian </h2>

In [76]:
```python
predict3 = clf3.predict(x_test)
print(classification_report(y_test,predict3))
```

```
             precision    recall  f1-score   support

        0.0       0.93      1.00      0.96       937
        1.0       1.00      0.03      0.05        72

avg / total       0.94      0.93      0.90      1009
```

In [77]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(y_test,predict3)
```

Out[77]: 0.9306243805748265

<h2> Multinominal Naive Bayes</h2> (Parametric Model)

In [78]:
```python
from sklearn.naive_bayes import MultinomialNB
clf4 = MultinomialNB().fit(x_train, y_train)
y_pred= clf4.predict(x_test)
calculate_metrics(y_test,y_pred)
```

```
(0.4643211100099108, 0.5, 0.4815005138746146, None)
0.9286422200198216
[[  0  72]
 [  0 937]]
```

<h2> Performance Matrix of MNB </h2>

In [79]:
```python
predict4 = clf4.predict(x_test)
print(classification_report(y_test,predict4))
```

```
             precision    recall  f1-score   support

        0.0       0.93      1.00      0.96       937
        1.0       0.00      0.00      0.00        72

avg / total       0.86      0.93      0.89      1009
```

In [80]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(y_test,predict4)
```

Out[80]: 0.9286422200198216

<h2> Decision Tree</h2> (Nonparametric Model)

```
In [81]: from sklearn import tree
         clf5 = tree.DecisionTreeClassifier().fit(x_train, y_train)
         y_pred= clf5.predict(x_test)
         calculate_metrics(y_test,y_pred)
```

```
(0.7666868015705225, 0.7355923159018143, 0.7499380421313506, None)
0.9375619425173439
[[ 36  36]
 [ 27 910]]
```

<h2> Performance Matrix of Decision Tree </h2>

```
In [82]: predict5 = clf5.predict(x_test)
         print(classification_report(y_test,predict5))
```

```
             precision    recall  f1-score   support

        0.0       0.96      0.97      0.97       937
        1.0       0.57      0.50      0.53        72

avg / total       0.93      0.94      0.94      1009
```

```
In [83]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test,predict5)
```

```
Out[83]: 0.9375619425173439
```

<h2> Random_Forest</h2> (Nonparametric Model)

```
In [84]: from sklearn.ensemble import RandomForestClassifier
         clf6 = RandomForestClassifier(n_estimators=10, max_depth=None,min_samples_split=2, ra
         y_pred= clf6.predict(x_test)
         calculate_metrics(y_test,y_pred)
```

```
(0.9703815261044177, 0.5902777777777778, 0.6376799245305985, None)
0.9415262636273538
[[ 13  59]
 [  0 937]]
```

<h2> Performance Matrix of Random Forest </h2>

```
In [85]: predict6 = clf6.predict(x_test)
         print(classification_report(y_test,predict6))
```

```
             precision    recall  f1-score   support

        0.0       0.94      1.00      0.97       937
        1.0       1.00      0.18      0.31        72

avg / total       0.94      0.94      0.92      1009
```

```
In [86]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test,predict6)
```

```
Out[86]: 0.9415262636273538
```

<h2> Gradient Boosting</h2> (Nonparametric Model)

```
In [87]: from sklearn.ensemble import GradientBoostingClassifier
         clf7 = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, ra
         y_pred= clf7.predict(x_test)
         calculate_metrics(y_test,y_pred)
```

```
(0.6842335224688165, 0.728129076248073, 0.7030046467018339, None)
0.9117938553022795
[[ 37  35]
 [ 54 883]]
```

<h2> Performance Matrix of GB </h2>

```
In [88]: predict7 = clf7.predict(x_test)
         print(classification_report(y_test,predict7))
```

```
             precision    recall  f1-score   support

        0.0       0.96      0.94      0.95       937
        1.0       0.41      0.51      0.45        72

avg / total       0.92      0.91      0.92      1009
```

```
In [89]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test,predict7)
```

```
Out[89]: 0.9117938553022795
```

Note: After seeing confusion matrix of parametric model, I found that Logistic Regression performed better where accuray is 94% and F1-score is 0.93.

After seeing confusion matrix of nonparametric model, I found that Random Forest performed better where accuracy was 94% and F1-score is 0.92.

<h2>Recommendation</h2>
Findings

1. Based on continuous dependent and independent variable, random forest (Regressor) performed better.
2. After leveling the continuous dependent variable and creating dummies of independent variables, random forest performed better.