1. import packages (numPy, pandas, matplotlib, seaborn)

In this case, I used these 4 packages. (sklearn also)

In [1]:
```
#Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Load ‘movie_metadata.csv’

In [2]:
```
#Reading the Data
df=pd.read_csv("movie_metadata.csv")
#Displaying the first 5 records
df.head(5)
```

Out[2]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes	gross
0	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	1000.0	760505847.0
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	40000.0	309404152.0
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	11000.0	200074175.0
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	27000.0	448130642.0
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	Rob Walker	131.0	NaN

5 rows × 28 columns

3. Drop irrelevant columns

In [3]:
```
#Dropping useless columns
df.drop('movie_title',axis=1,inplace=True)
df.drop('language',axis=1,inplace=True)
df.drop('plot_keywords',axis=1,inplace=True)
df.drop('genres',axis=1,inplace =True)
df.drop('movie_imdb_link', axis=1, inplace=True)
df.drop('color',axis=1,inplace=True)
df.drop('actor_1_name',axis=1,inplace=True)
df.drop('actor_2_name',axis=1,inplace=True)
df.drop('actor_3_name',axis=1,inplace=True)
df.drop('director_name', axis=1, inplace=True)
```

4. Deal with missing values

In [4]:
```
# check the null values
df.isna().sum()
```

Out[4]:
```
num_critic_for_reviews        50
duration                      15
director_facebook_likes      104
actor_3_facebook_likes        23
actor_1_facebook_likes         7
gross                        884
num_voted_users                0
cast_total_facebook_likes      0
facenumber_in_poster          13
num_user_for_reviews          21
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 [6]: # deal with missing values in the dataset
        df.dropna(axis=0,subset=['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', 'country', 'content_rating', 'budget',
               'title_year', 'actor_2_facebook_likes', 'imdb_score', 'aspect_ratio',
               'movie_facebook_likes'],inplace=True)

        #Replacing valus
        df["content_rating"].fillna("R18", inplace = True)
        df["aspect_ratio"].fillna(df["aspect_ratio"].median(),inplace=True)
        df["budget"].fillna(df["budget"].median(),inplace=True)
        df['gross'].fillna(df['gross'].median(),inplace=True)
```

```
In [7]: # check the null values again
        df.isna().sum()

Out[7]: num_critic_for_reviews       0
        duration                     0
        director_facebook_likes      0
        actor_3_facebook_likes       0
        actor_1_facebook_likes       0
        gross                        0
        num_voted_users              0
        cast_total_facebook_likes    0
        facenumber_in_poster         0
        num_user_for_reviews         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
```

I drop the NaN data and replace some value to several columns to the further steps.

5. Remove the duplicate value

```
In [8]: #Removing the duplicate values in the datset
        df.drop_duplicates(inplace=True)
```

6. Reorganize several columns

```
In [9]: #combine facebook likes of actor 2 and actor 3
        df['Other_actor_facebbok_likes']=df["actor_2_facebook_likes"] + df['actor_3_facebook_likes']
        df.drop('actor_2_facebook_likes',axis=1,inplace=True)
        df.drop('actor_3_facebook_likes',axis=1,inplace=True)
        df.drop('cast_total_facebook_likes',axis=1,inplace=True)
        #create the ratio of num_user_for_reviews and num_critic_for_reviews.
        df['critic_review_ratio']=df['num_critic_for_reviews']/df['num_user_for_reviews']
        df.drop('num_critic_for_reviews',axis=1,inplace=True)
        df.drop('num_user_for_reviews',axis=1,inplace=True)
```

I combined and deal with several relative columns to more meaningful features.

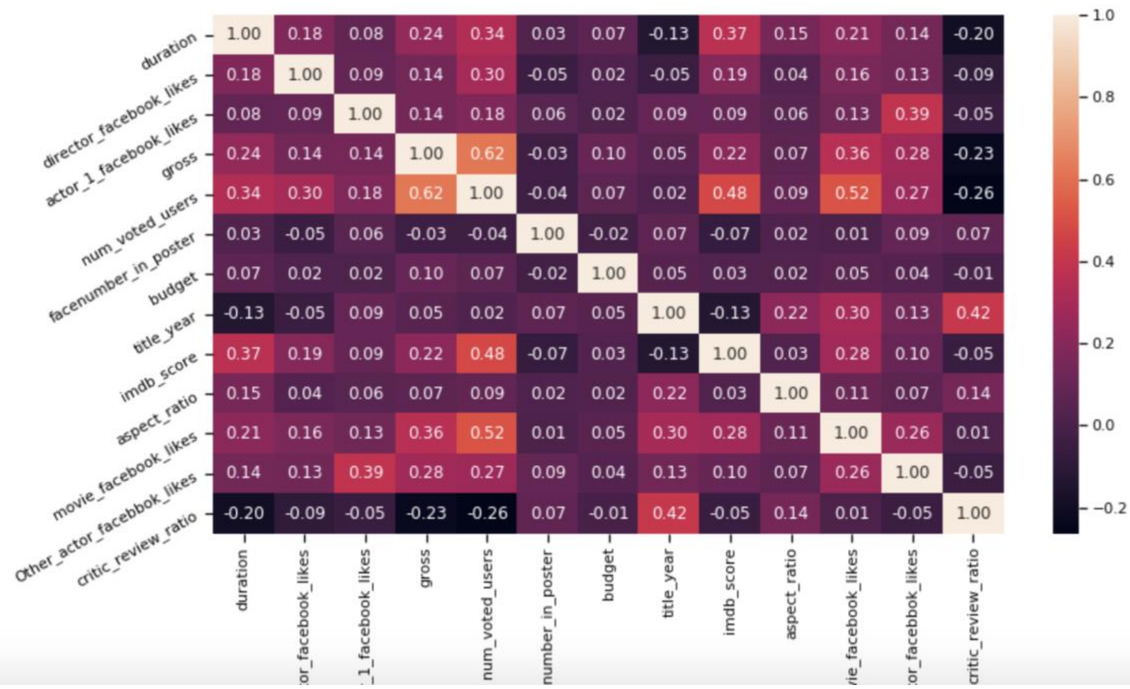
7. Deal with correlation problem

Since the cast_total_facebook_likes & actor_1_facebook_like and num_voted_users, num_user_for_reviews & num_critic_for_reviews are highly correlated. I dealt with them like the following picture.

```
#combine facebook likes of actor 2 and actor 3
df['Other_actor_facebbok_likes']=df["actor_2_facebook_likes"] + df['actor_3_facebook_likes']
df.drop('actor_2_facebook_likes',axis=1,inplace=True)
df.drop('actor_3_facebook_likes',axis=1,inplace=True)
df.drop('cast_total_facebook_likes',axis=1,inplace=True)
#create the ratio of num_user_for_reviews and num_critic_for_reviews.
df['critic_review_ratio']=df['num_critic_for_reviews']/df['num_user_for_reviews']
df.drop('num_critic_for_reviews',axis=1,inplace=True)
df.drop('num_user_for_reviews',axis=1,inplace=True)
```

After removing and combing the columns which are highly correlated to each other. The correlation matrix is good now.

```
In [10]: # Correlation matrix shown in the figure
corr = df.corr()
sns.set_context("notebook", font_scale=1.0, rc={"lines.linewidth": 2.5})
plt.figure(figsize=(13,7))
a = sns.heatmap(corr, annot=True, fmt='.2f')
rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```



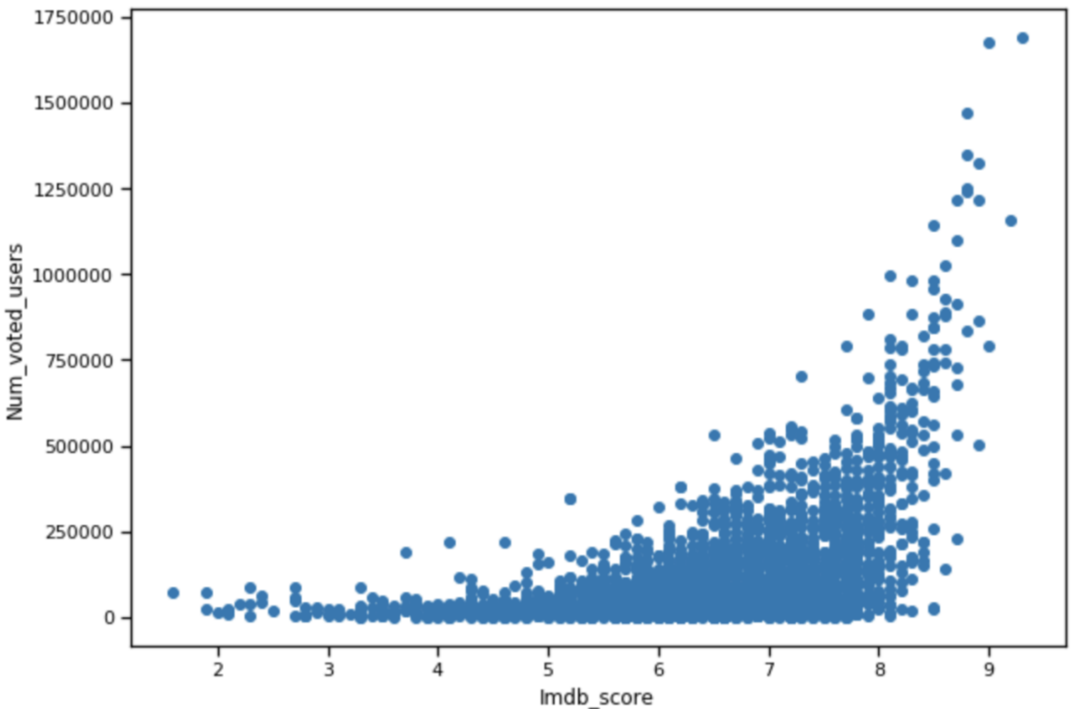
8. Deal with categorical columns (dummy)
To one-hot code for the following models.

```
In [11]: #deal with categorical data
value_counts=df["country"].value_counts()
vals = value_counts[:2].index
df['country'] = df.country.where(df.country.isin(vals), 'other')
df = pd.get_dummies(data = df, columns = ['country'] , prefix = ['country'] , drop_first = True)
df = pd.get_dummies(data = df, columns = ['content_rating'] , prefix = ['content_rating'] , drop_first = True)
```

9. Data virtualization

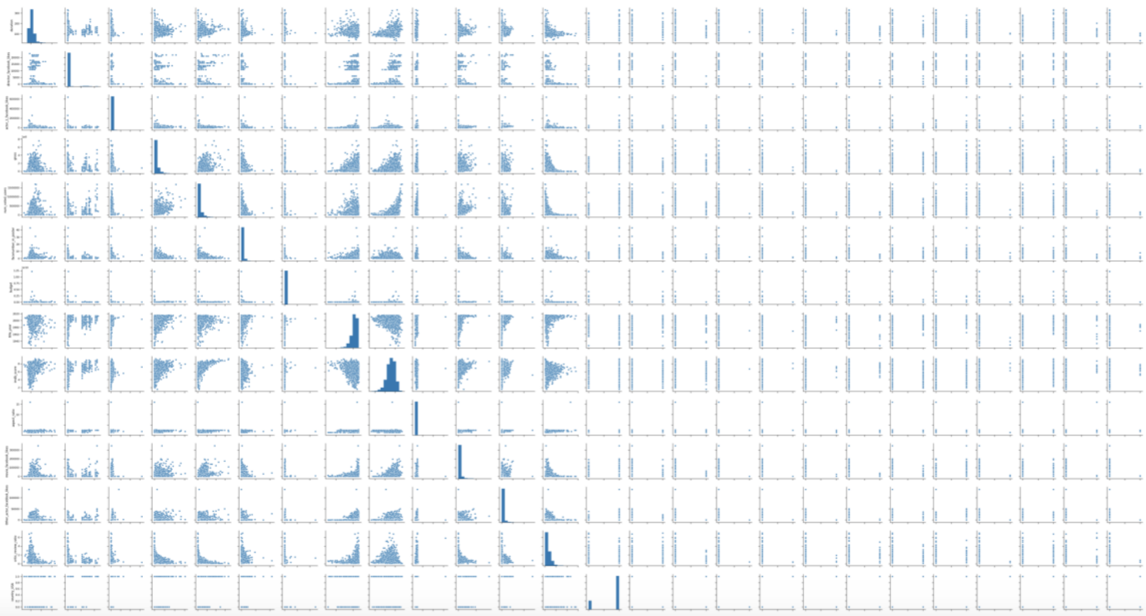
Show the relationship between Num_voted_users and imdb_score:

```
#Plot imdb_score vs num_voted_users
plt.figure(figsize=[10,7])
plt.scatter(df.imdb_score, df.num_voted_users)
plt.xlabel("Imdb_score")
plt.ylabel("Num_voted_users")
plt.show()
```



In the following pairplot picture, we can find the variations in each plot. The plots are in matrix format. The X axis means row and y axis means column. The main-diagonal subplots are the univariate histograms for each attribute.

In [16]:
```
sns.pairplot(df[num_cols])
```
Out[16]: <seaborn.axisgrid.PairGrid at 0x7fc0bdbec8d0>



10. split dataset into training and testing dataset

I use df.columns to find all columns and split the dataset by 7:3 ratio.

In [19]:
```
# Splitting the data into training and testing data
X=pd.DataFrame(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', 'aspect_ratio', 'movie_facebook_likes',
       'country_USA', 'country_other', 'content_rating_G',
       'content_rating_GP', 'content_rating_M', 'content_rating_NC-17',
       'content_rating_Not Rated', 'content_rating_PG', 'content_rating_PG-13',
       'content_rating_Passed', 'content_rating_R', 'content_rating_Unrated',
       'content_rating_X'],data=df)
y=pd.DataFrame(columns=['imdb_score_y'],data=df)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.3,random_state=100)
```

In [20]:
```
#Feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

11. Create models (Logistic Regression, KNN, Random Forest)

Logistic Regression (Accuracy: 72%)

In [51]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
logit =LogisticRegression()
logit.fit(X_train,np.ravel(y_train,order='C'))
y_pred=logit.predict(X_test)

#Confusion matrix for logistic regression**
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
[[  0  20   5   0]
 [  0 126 157   0]
 [  0 106 657   3]
 [  0   0  15  33]]
Accuracy: 0.7272727272727273
```

KNN (Accuracy: 68%)

In [52]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=22)
knn.fit(X_train, np.ravel(y_train,order='C'))
knnpred = knn.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, knnpred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, knnpred))
```

```
[[  0  15  10   0]
 [  0 120 163   0]
 [  0 124 642   0]
 [  0   0  39   9]]
Accuracy: 0.6871657754010695
```

Random Forest (Accuracy: 78%)

```
In [56]: #Random Forest
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 200)#criterion = entopy,gini
rfc.fit(X_train, np.ravel(y_train,order='C'))
rfcpred = rfc.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, rfcpred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, rfcpred))
```

```
[[  0  19   6   0]
 [  0 144 139   0]
 [  0  53 708   5]
 [  0   0  24  24]]
Accuracy: 0.7807486631016043
```

12. Model comparison

```
Logistic  Reports
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        25
           2       0.50      0.45      0.47       283
           3       0.79      0.86      0.82       766
           4       0.92      0.69      0.79        48

    accuracy                           0.73      1122
   macro avg       0.55      0.50      0.52      1122
weighted avg       0.70      0.73      0.71      1122

KNN Reports
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        25
           2       0.46      0.42      0.44       283
           3       0.75      0.84      0.79       766
           4       1.00      0.19      0.32        48

    accuracy                           0.69      1122
   macro avg       0.55      0.36      0.39      1122
weighted avg       0.67      0.69      0.67      1122

Random Forests Reports
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        25
           2       0.63      0.50      0.56       283
           3       0.80      0.91      0.85       766
           4       0.82      0.48      0.61        48

    accuracy                           0.77      1122
   macro avg       0.56      0.47      0.50      1122
weighted avg       0.74      0.77      0.75      1122
```

13. Conclusion
(#The conclusion is that Random Forest Algorithm have best accuracy which is around 78%)