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

In [2]:
```
df=pd.read_csv('movie_success_rate.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	...	Music	Musical	Mystery	Ro
0	1.0	Guardians of the Galaxy	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014.0	121.0	8.1	757074.0	...	0.0	0.0	0.0	
1	2.0	Prometheus	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012.0	124.0	7.0	485820.0	...	0.0	0.0	1.0	
2	3.0	Split	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016.0	117.0	7.3	157606.0	...	0.0	0.0	0.0	
3	4.0	Sing	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2016.0	108.0	7.2	60545.0	...	0.0	0.0	0.0	
4	5.0	Suicide Squad	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016.0	123.0	6.2	393727.0	...	0.0	0.0	0.0	

5 rows × 33 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 839 entries, 0 to 838
Data columns (total 33 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Rank                838 non-null    float64
 1   Title               838 non-null    object 
 2   Genre               838 non-null    object 
 3   Description         838 non-null    object 
 4   Director            838 non-null    object 
 5   Actors              838 non-null    object 
 6   Year                838 non-null    float64
 7   Runtime (Minutes)   838 non-null    float64
 8   Rating              839 non-null    float64
 9   Votes               839 non-null    float64
 10  Revenue (Millions)  839 non-null    float64
 11  Metascore           838 non-null    float64
 12  Action              838 non-null    float64
 13  Adventure           838 non-null    float64
 14  Aniimation          838 non-null    float64
 15  Biography           838 non-null    float64
 16  Comedy              838 non-null    float64
 17  Crime               838 non-null    float64
 18  Drama               838 non-null    float64
 19  Family              838 non-null    float64
 20  Fantasy             838 non-null    float64
 21  History             838 non-null    float64
 22  Horror              838 non-null    float64
 23  Music               838 non-null    float64
 24  Musical             838 non-null    float64
 25  Mystery             838 non-null    float64
 26  Romance             838 non-null    float64
 27  Sci-Fi              838 non-null    float64
 28  Sport               838 non-null    float64
 29  Thriller            838 non-null    float64
 30  War                 838 non-null    float64
 31  Western             838 non-null    float64
 32  Success             838 non-null    float64
dtypes: float64(28), object(5)
memory usage: 216.4+ KB
```

In [5]:
```
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
Rank	838.0	485.247017	286.572065	1.0	238.250	475.50	729.75	1000.00
Year	838.0	2012.507160	3.172360	2006.0	2010.000	2013.00	2015.00	2016.00
Runtime (Minutes)	838.0	114.638425	18.470922	66.0	101.000	112.00	124.00	187.00
Rating	839.0	6.814320	0.877230	1.9	6.300	6.90	7.50	9.00
Votes	839.0	193230.251790	192983.756508	178.0	61455.000	137117.00	270865.00	1791916.00
Revenue (Millions)	839.0	84.564558	104.457845	0.0	13.975	48.24	116.73	936.63
Metascore	838.0	59.575179	16.952416	11.0	47.000	60.00	72.00	100.00
Action	838.0	0.330549	0.470692	0.0	0.000	0.00	1.00	1.00
Adventure	838.0	0.291169	0.454573	0.0	0.000	0.00	1.00	1.00
Aniimation	838.0	0.053699	0.225558	0.0	0.000	0.00	0.00	1.00
Biography	838.0	0.079952	0.271381	0.0	0.000	0.00	0.00	1.00
Comedy	838.0	0.298329	0.457798	0.0	0.000	0.00	1.00	1.00
Crime	838.0	0.150358	0.357635	0.0	0.000	0.00	0.00	1.00
Drama	838.0	0.500000	0.500299	0.0	0.000	0.50	1.00	1.00
Family	838.0	0.057279	0.232514	0.0	0.000	0.00	0.00	1.00
Fantasy	838.0	0.109785	0.312809	0.0	0.000	0.00	0.00	1.00
History	838.0	0.029833	0.170228	0.0	0.000	0.00	0.00	1.00
Horror	838.0	0.103819	0.305207	0.0	0.000	0.00	0.00	1.00
Music	838.0	0.023866	0.152724	0.0	0.000	0.00	0.00	1.00
Musical	838.0	0.005967	0.077059	0.0	0.000	0.00	0.00	1.00
Mystery	838.0	0.102625	0.303650	0.0	0.000	0.00	0.00	1.00
Romance	838.0	0.143198	0.350484	0.0	0.000	0.00	0.00	1.00
Sci-Fi	838.0	0.127685	0.333938	0.0	0.000	0.00	0.00	1.00
Sport	838.0	0.017900	0.132666	0.0	0.000	0.00	0.00	1.00
Thriller	838.0	0.176611	0.381567	0.0	0.000	0.00	0.00	1.00
War	838.0	0.011933	0.108650	0.0	0.000	0.00	0.00	1.00
Western	838.0	0.004773	0.068965	0.0	0.000	0.00	0.00	1.00
Success	838.0	0.177804	0.382576	0.0	0.000	0.00	0.00	1.00

In [6]:
```
df.drop('Rank',axis=1,inplace=True)
```

In [7]:
```
df.isnull().sum()/len(df)*100
```

Out[7]:
```
Title                 0.11919
Genre                 0.11919
Description           0.11919
Director              0.11919
Actors                0.11919
Year                  0.11919
Runtime (Minutes)     0.11919
Rating                0.00000
Votes                 0.00000
Revenue (Millions)    0.00000
Metascore             0.11919
Action                0.11919
Adventure             0.11919
Aniimation            0.11919
Biography             0.11919
Comedy                0.11919
Crime                 0.11919
Drama                 0.11919
Family                0.11919
Fantasy               0.11919
History               0.11919
Horror                0.11919
Music                 0.11919
Musical               0.11919
Mystery               0.11919
Romance               0.11919
Sci-Fi                0.11919
Sport                 0.11919
Thriller              0.11919
War                   0.11919
Western               0.11919
Success               0.11919
dtype: float64
```

In [8]:
```
num_col=df.select_dtypes(include=['int64','float64']).columns
num_col
```

Out[8]:
```
Index(['Year', 'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
       'Metascore', 'Action', 'Adventure', 'Aniimation', 'Biography', 'Comedy',
       'Crime', 'Drama', 'Family', 'Fantasy', 'History', 'Horror', 'Music',
       'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Sport', 'Thriller', 'War',
       'Western', 'Success'],
      dtype='object')
```

In [9]:
```
cat_col=df.select_dtypes(include=['O']).columns
cat_col
```

Out[9]:
```
Index(['Title', 'Genre', 'Description', 'Director', 'Actors'], dtype='object')
```

In [10]:
```
for i in num_col:
    df[i]=df[i].fillna(df[i].mean())
```

In [11]:
```
for i in cat_col:
    df[i]=df[i].fillna(df[i].mode()[0])
```

In [12]:
```
df.isnull().sum()/len(df)*100
```

Out[12]:
```
Title                 0.0
Genre                 0.0
Description           0.0
Director              0.0
Actors                0.0
Year                  0.0
Runtime (Minutes)     0.0
Rating                0.0
Votes                 0.0
Revenue (Millions)    0.0
Metascore             0.0
Action                0.0
Adventure             0.0
Aniimation            0.0
Biography             0.0
Comedy                0.0
Crime                 0.0
Drama                 0.0
Family                0.0
Fantasy               0.0
History               0.0
Horror                0.0
Music                 0.0
Musical               0.0
Mystery               0.0
Romance               0.0
Sci-Fi                0.0
Sport                 0.0
Thriller              0.0
War                   0.0
Western               0.0
Success               0.0
dtype: float64
```

In [13]:
```
df['Success'].unique()
```

Out[13]:
```
array([1.       , 0.       , 0.1778043])
```

In [14]:
```
df['Success'].replace(0.17780429594272076,0,inplace=True)
```

In [15]:
```
df['Success'].unique()
```

Out[15]:
```
array([1., 0.])
```

In [16]:
```
df.head()
```

Out[16]:

	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	...	Music	Musical	Mystery
0	Guardians of the Galaxy	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014.0	121.0	8.1	757074.0	333.13	...	0.0	0.0	0.0
1	Prometheus	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012.0	124.0	7.0	485820.0	126.46	...	0.0	0.0	1.0
2	Split	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016.0	117.0	7.3	157606.0	138.12	...	0.0	0.0	0.0
3	Sing	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2016.0	108.0	7.2	60545.0	270.32	...	0.0	0.0	0.0
4	Suicide Squad	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016.0	123.0	6.2	393727.0	325.02	...	0.0	0.0	0.0

5 rows × 32 columns

In [17]:
```
df['Year']=df['Year'].astype('int64')
df['Votes']=df['Votes'].astype('int64')
df['Runtime (Minutes)']=df['Runtime (Minutes)'].astype('int64')
```

In [18]:
```
df.head()
```

Out[18]:

	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	...	Music	Musical	Mystery	R
0	Guardians of the Galaxy	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1	757074	333.13	...	0.0	0.0	0.0	
1	Prometheus	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012	124	7.0	485820	126.46	...	0.0	0.0	1.0	
2	Split	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016	117	7.3	157606	138.12	...	0.0	0.0	0.0	
3	Sing	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2016	108	7.2	60545	270.32	...	0.0	0.0	0.0	
4	Suicide Squad	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016	123	6.2	393727	325.02	...	0.0	0.0	0.0	

5 rows × 32 columns

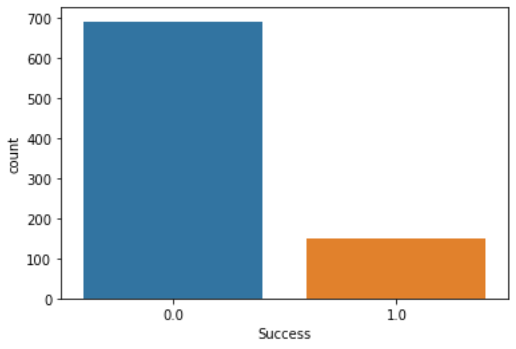
In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 839 entries, 0 to 838
Data columns (total 32 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Title               839 non-null    object 
 1   Genre               839 non-null    object 
 2   Description         839 non-null    object 
 3   Director            839 non-null    object 
 4   Actors              839 non-null    object 
 5   Year                839 non-null    int64  
 6   Runtime (Minutes)   839 non-null    int64  
 7   Rating              839 non-null    float64
 8   Votes               839 non-null    int64  
 9   Revenue (Millions)  839 non-null    float64
 10  Metascore           839 non-null    float64
 11  Action              839 non-null    float64
 12  Adventure           839 non-null    float64
 13  Aniimation          839 non-null    float64
 14  Biography           839 non-null    float64
 15  Comedy              839 non-null    float64
 16  Crime               839 non-null    float64
 17  Drama               839 non-null    float64
 18  Family              839 non-null    float64
 19  Fantasy             839 non-null    float64
 20  History             839 non-null    float64
 21  Horror              839 non-null    float64
 22  Music               839 non-null    float64
 23  Musical             839 non-null    float64
 24  Mystery             839 non-null    float64
 25  Romance             839 non-null    float64
 26  Sci-Fi              839 non-null    float64
 27  Sport               839 non-null    float64
 28  Thriller            839 non-null    float64
 29  War                 839 non-null    float64
 30  Western             839 non-null    float64
 31  Success             839 non-null    float64
dtypes: float64(24), int64(3), object(5)
memory usage: 209.9+ KB
```

Visulization

In [20]:
```
sns.countplot(df['Success'])
```

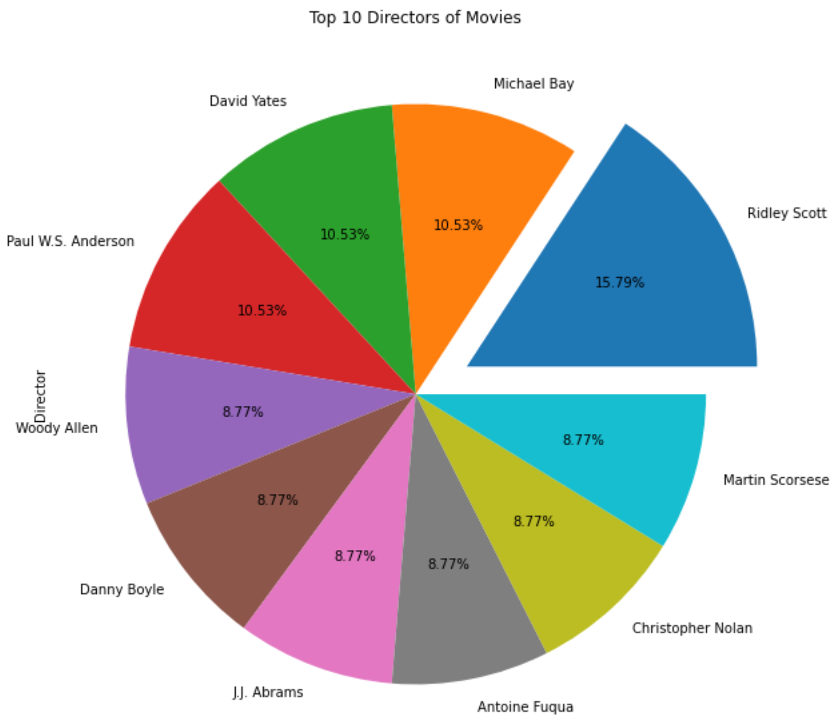
Out[20]:
```
<AxesSubplot:xlabel='Success', ylabel='count'>
```



By seeing above graph the given data is imbalanced.

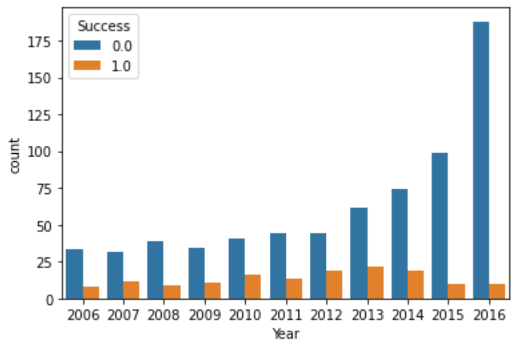
```
In [21]: print(df['Director'].value_counts()[:10])
         plt.figure(figsize=(10,10))
         df['Director'].value_counts()[:10].plot.pie(autopct="%1.2f%%",explode=(0.2,0,0,0,0,0,0,0,0,0))
         plt.title('Top 10 Directors of Movies')
         plt.show()
```

```
Ridley Scott          9
Michael Bay           6
David Yates           6
Paul W.S. Anderson    6
Woody Allen           5
Danny Boyle           5
J.J. Abrams           5
Antoine Fuqua         5
Christopher Nolan     5
Martin Scorsese       5
Name: Director, dtype: int64
```



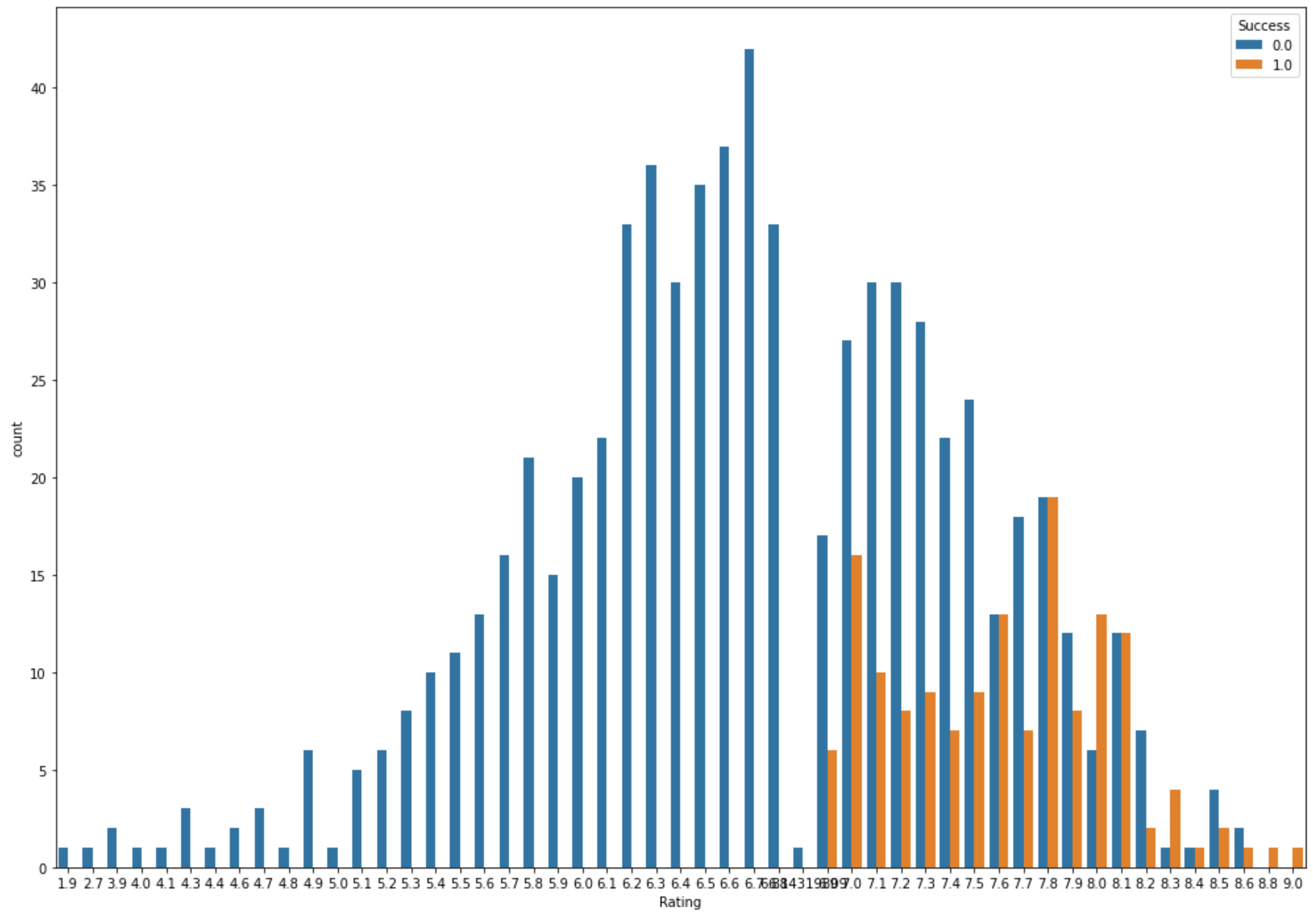
```
In [22]: sns.countplot(df['Year'],hue=df['Success'])
```

```
Out[22]: <AxesSubplot:xlabel='Year', ylabel='count'>
```



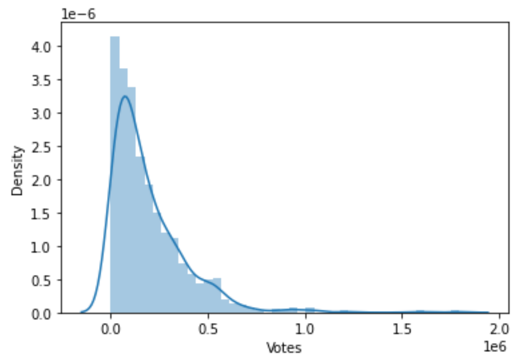
In [23]:
```
plt.figure(figsize=(17,12))
sns.countplot(df['Rating'],hue=df['Success'])
```

Out[23]:
```
<AxesSubplot:xlabel='Rating', ylabel='count'>
```



In [67]:
```
sns.distplot(df['Votes'])
```

Out[67]:
```
<AxesSubplot:xlabel='Votes', ylabel='Density'>
```

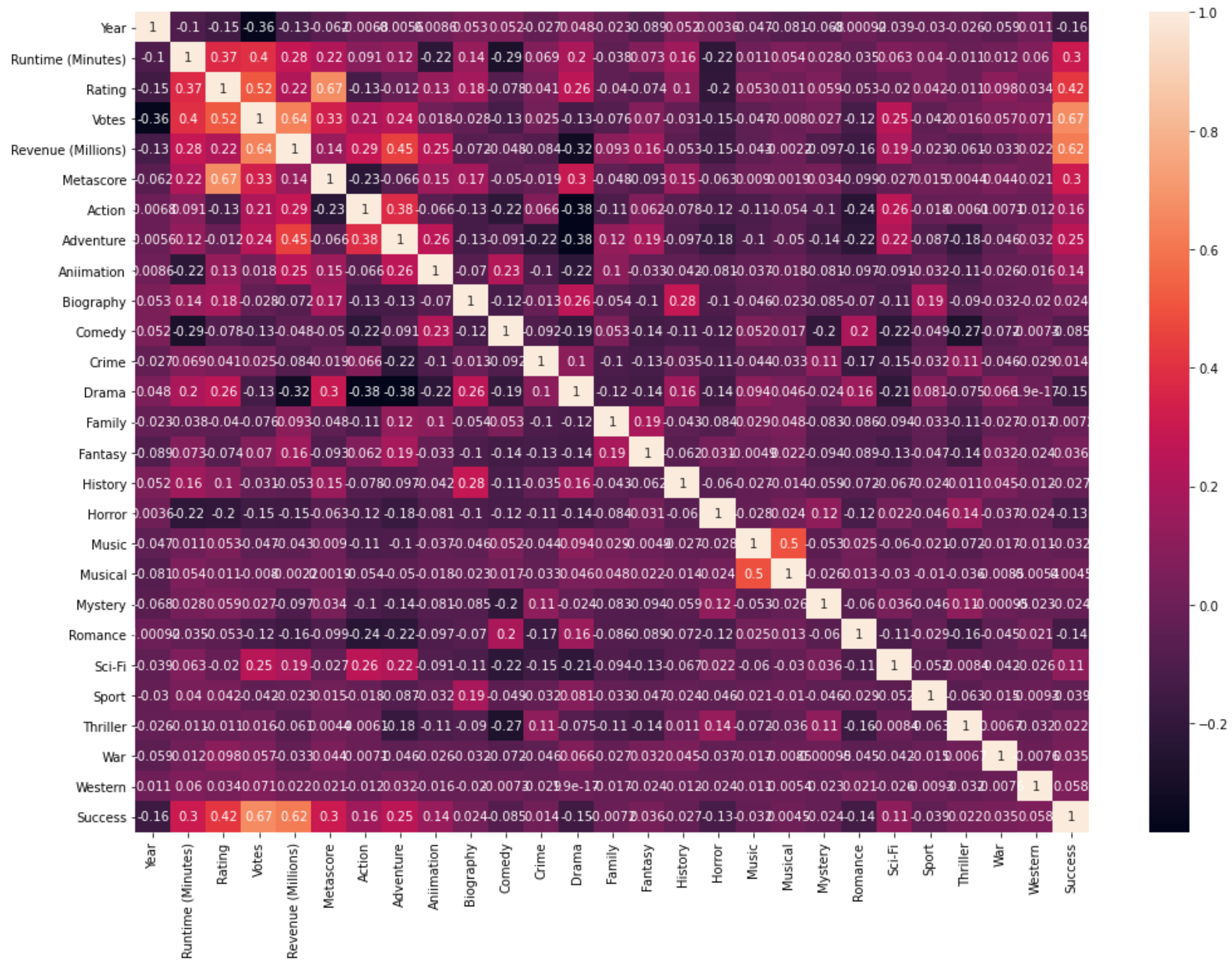


In [ ]:
```
plt.figure(figsize=(20,20))
count=1
for i in cat_col:
    plt.subplot(2,3,count)
    sns.countplot(df[i])
    count+=1
plt.show()
```

In [ ]:

In [24]:
```
plt.figure(figsize=(17,12))
sns.heatmap(df.corr(),annot=True)
```

Out[24]:
```
<AxesSubplot:>
```



Encoding

In [25]:
```
from sklearn.preprocessing import LabelEncoder
```

In [26]:
```
le=LabelEncoder()
```

In [27]:
```
for i in cat_col:
    df[i]=le.fit_transform(df[i])
```

In [28]:
```
df.head()
```

Out[28]:

	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	...	Music	Musical	Mystery	Romance	Sci-Fi	Sport	Thriller	War	We
0	236	11	88	208	156	2014	121	8.1	757074	333.13	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
1	477	79	445	417	612	2012	124	7.0	485820	126.46	...	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
2	552	181	731	317	350	2016	117	7.3	157606	138.12	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
3	535	86	502	85	548	2016	108	7.2	60545	270.32	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	569	7	182	109	812	2016	123	6.2	393727	325.02	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 32 columns

In [ ]:

In [ ]:

In [ ]:

Feature Selection

In [29]:
```
x=df.drop('Success',axis=1)
y=df['Success']
```

In [30]:
```
x.head()
```

Out[30]:

	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	...	Horror	Music	Musical	Mystery	Romance	Sci-Fi	Sport	Thriller
0	236	11	88	208	156	2014	121	8.1	757074	333.13	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1	477	79	445	417	612	2012	124	7.0	485820	126.46	...	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0
2	552	181	731	317	350	2016	117	7.3	157606	138.12	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	535	86	502	85	548	2016	108	7.2	60545	270.32	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	569	7	182	109	812	2016	123	6.2	393727	325.02	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 31 columns

In [31]:
```
y
```

Out[31]:
```
0      1.0
1      1.0
2      0.0
3      0.0
4      0.0
      ...
834    0.0
835    0.0
836    0.0
837    0.0
838    0.0
Name: Success, Length: 839, dtype: float64
```

Split Data into Train Test Data

In [32]:
```
from sklearn.model_selection import train_test_split
```

In [33]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=111)
```

In [34]:
```
x_train.shape
```

Out[34]:
```
(671, 31)
```

In [35]:
```
x_test.shape
```

Out[35]:
```
(168, 31)
```

In [36]:
```
y_train.shape
```

Out[36]:
```
(671,)
```

In [37]:
```
y_test.shape
```

Out[37]:
```
(168,)
```

In [38]:
```
from sklearn.preprocessing import StandardScaler
```

In [39]:
```
sc=StandardScaler()
```

In [40]:
```
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

Logistic Regression Model

In [41]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

In [42]:
```
def my_model(clf):
    clf.fit(x_train,y_train)
    y_train_pred=clf.predict(x_train)
    y_test_pred=clf.predict(x_test)
    print('Train Data')
    print(classification_report(y_train,y_train_pred))
    print('Test Data')
    print(classification_report(y_test,y_test_pred))
```

In [43]:
```
lr=LogisticRegression()
```

In [44]:
```
my_model(lr)
```

```
Train Data
              precision    recall  f1-score   support

         0.0       0.96      0.98      0.97       557
         1.0       0.89      0.78      0.83       114

    accuracy                           0.95       671
   macro avg       0.92      0.88      0.90       671
weighted avg       0.94      0.95      0.94       671

Test Data
              precision    recall  f1-score   support

         0.0       0.95      0.97      0.96       133
         1.0       0.88      0.80      0.84        35

    accuracy                           0.93       168
   macro avg       0.91      0.88      0.90       168
weighted avg       0.93      0.93      0.93       168
```

Decision Tree Model

In [45]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [46]:
```
dt=DecisionTreeClassifier()
```

In [47]:
```
dt
```

Out[47]:
```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [48]:
```
my_model(dt)
```

```
Train Data
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       557
         1.0       1.00      1.00      1.00       114

    accuracy                           1.00       671
   macro avg       1.00      1.00      1.00       671
weighted avg       1.00      1.00      1.00       671

Test Data
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       133
         1.0       0.95      1.00      0.97        35

    accuracy                           0.99       168
   macro avg       0.97      0.99      0.98       168
weighted avg       0.99      0.99      0.99       168
```

In [49]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [50]:
```
param_grid={
    'criterion':['gini','entropy'],
    'class_weight':[None,'balanced'],
    'max_depth':np.arange(2,50),
    'min_samples_split':np.arange(2,50,2),
    'min_samples_leaf':np.arange(2,50)
}
```

In [80]:
```
dt_rcv=RandomizedSearchCV(dt,param_distributions=param_grid,n_iter=10,scoring='f1',n_jobs=-1)
```

In [81]:
```
dt_rcv.fit(x_train,y_train)
```

Out[81]:
```
RandomizedSearchCV(estimator=DecisionTreeClassifier(), n_jobs=-1,
                   param_distributions={'class_weight': [None, 'balanced'],
                                        'criterion': ['gini', 'entropy'],
                                        'max_depth': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),
                                        'min_samples_leaf': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30,
32, 34,
       36, 38, 40, 42, 44, 46, 48])},
                   scoring='f1')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [82]:
```
dt_rcv.best_params_
```

Out[82]:
```
{'min_samples_split': 30,
 'min_samples_leaf': 18,
 'max_depth': 28,
 'criterion': 'entropy',
 'class_weight': 'balanced'}
```

In [83]:
```
dt1=DecisionTreeClassifier(criterion='entropy',class_weight='balanced',max_depth=28,min_samples_split=30,min_samples_leaf=18)
```

In [84]:
```
my_model(dt1)
```

```
Train Data
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       557
         1.0       1.00      1.00      1.00       114

    accuracy                           1.00       671
   macro avg       1.00      1.00      1.00       671
weighted avg       1.00      1.00      1.00       671

Test Data
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       133
         1.0       0.95      1.00      0.97        35

    accuracy                           0.99       168
   macro avg       0.97      0.99      0.98       168
weighted avg       0.99      0.99      0.99       168
```

Random Forest Model

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [57]:
```
rf=RandomForestClassifier()
```

In [58]:
```
my_model(rf)
```

```
Train Data
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       557
         1.0       1.00      1.00      1.00       114

    accuracy                           1.00       671
   macro avg       1.00      1.00      1.00       671
weighted avg       1.00      1.00      1.00       671

Test Data
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       133
         1.0       0.95      1.00      0.97        35

    accuracy                           0.99       168
   macro avg       0.97      0.99      0.98       168
weighted avg       0.99      0.99      0.99       168
```

In [59]:
```
rf_rcv=RandomizedSearchCV(rf,param_distributions=param_grid,n_iter=10,scoring='f1',n_jobs=-1)
```

In [60]:
```
rf_rcv.fit(x_train,y_train)
```

Out[60]:
```
RandomizedSearchCV(estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'class_weight': [None, 'balanced'],
                                        'criterion': ['gini', 'entropy'],
                                        'max_depth': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),
                                        'min_samples_leaf': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30,
32, 34,
       36, 38, 40, 42, 44, 46, 48])},
                   scoring='f1')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:
```
rf_rcv.best_params_
```

Out[61]:
```
{'min_samples_split': 46,
 'min_samples_leaf': 10,
 'max_depth': 7,
 'criterion': 'entropy',
 'class_weight': 'balanced'}
```

In [85]:
```
rf1=RandomForestClassifier(criterion='entropy',class_weight='balanced',max_depth=7,min_samples_split=46,min_samples_leaf=10)
```

In [86]:
```
my_model(rf1)
```

```
Train Data
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       557
         1.0       1.00      1.00      1.00       114

    accuracy                           1.00       671
   macro avg       1.00      1.00      1.00       671
weighted avg       1.00      1.00      1.00       671

Test Data
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       133
         1.0       0.95      1.00      0.97        35

    accuracy                           0.99       168
   macro avg       0.97      0.99      0.98       168
weighted avg       0.99      0.99      0.99       168
```

In [ ]:

AdaBoost Model

In [64]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [65]:
```
adb=AdaBoostClassifier(n_estimators=450)
```

In [66]:
```
my_model(adb)
```

```
Train Data
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       557
         1.0       1.00      1.00      1.00       114

    accuracy                           1.00       671
   macro avg       1.00      1.00      1.00       671
weighted avg       1.00      1.00      1.00       671

Test Data
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       133
         1.0       0.95      1.00      0.97        35

    accuracy                           0.99       168
   macro avg       0.97      0.99      0.98       168
weighted avg       0.99      0.99      0.99       168
```

In [71]:
```
param_grid_ada={
    'learning_rate':[0.1,0.01,1,2,3],
    'n_estimators':[50,100,150]
}
```

In [75]:
```
adb_rcv=RandomizedSearchCV(adb,param_distributions=param_grid_ada,n_iter=10,scoring='f1',n_jobs=-1)
```

In [76]:
```
adb_rcv.fit(x_train,y_train)
```

Out[76]:
```
RandomizedSearchCV(estimator=AdaBoostClassifier(n_estimators=450), n_jobs=-1,
                   param_distributions={'learning_rate': [0.1, 0.01, 1, 2, 3],
                                        'n_estimators': [50, 100, 150]},
                   scoring='f1')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [77]:
```
adb_rcv.best_params_
```

Out[77]:
```
{'n_estimators': 50, 'learning_rate': 1}
```

In [78]:
```
adb1=AdaBoostClassifier(n_estimators=50,learning_rate=1)
```

In [79]:
```
my_model(adb1)
```

```
Train Data
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       557
         1.0       1.00      1.00      1.00       114

    accuracy                           1.00       671
   macro avg       1.00      1.00      1.00       671
weighted avg       1.00      1.00      1.00       671

Test Data
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       133
         1.0       0.95      1.00      0.97        35

    accuracy                           0.99       168
   macro avg       0.97      0.99      0.98       168
weighted avg       0.99      0.99      0.99       168
```

Best Prediction is Given By Logistic Regression Model

```
In [ ]:
```