In [1]:
```
# Supress Warnings
import warnings
warnings.filterwarnings('ignore')
```

#Importing the Libraries

In [2]:
```
# lets import the basic Libraries
import numpy as np
import pandas as pd

# for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# for jupyter notebook widgets
#import ipywidgets as widgets
#from ipywidgets import interact
#from ipywidgets import interact_manual

# for Interactive Shells
from IPython.display import display

# setting up the chart size and background
plt.rcParams['figure.figsize'] = (16, 8)
plt.style.use('fivethirtyeight')
```

#Loading the Data Set

In [3]:
```
# lets read the dataset
data = pd.read_csv('movie_metadata.csv')
```

#Shape of the Data

In [4]:
```
# lets check the shape
print(data.shape)
```

```
(5043, 28)
```

#Information about the Data Set

In [5]:
```python
# lets check the column wise info

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5043 entries, 0 to 5042
Data columns (total 28 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   color                      5024 non-null   object 
 1   director_name              4939 non-null   object 
 2   num_critic_for_reviews     4993 non-null   float64
 3   duration                   5028 non-null   float64
 4   director_facebook_likes    4939 non-null   float64
 5   actor_3_facebook_likes     5020 non-null   float64
 6   actor_2_name               5030 non-null   object 
 7   actor_1_facebook_likes     5036 non-null   float64
 8   gross                      4159 non-null   float64
 9   genres                     5043 non-null   object 
 10  actor_1_name               5036 non-null   object 
 11  movie_title                5043 non-null   object 
 12  num_voted_users            5043 non-null   int64  
 13  cast_total_facebook_likes  5043 non-null   int64  
 14  actor_3_name               5020 non-null   object 
 15  facenumber_in_poster       5030 non-null   float64
 16  plot_keywords              4890 non-null   object 
 17  movie_imdb_link            5043 non-null   object 
 18  num_user_for_reviews       5022 non-null   float64
 19  language                   5029 non-null   object 
 20  country                    5038 non-null   object 
 21  content_rating             4740 non-null   object 
 22  budget                     4551 non-null   float64
 23  title_year                 4935 non-null   float64
 24  actor_2_facebook_likes     5030 non-null   float64
 25  imdb_score                 5043 non-null   float64
 26  aspect_ratio               4714 non-null   float64
 27  movie_facebook_likes       5043 non-null   int64  
dtypes: float64(13), int64(3), object(12)
memory usage: 1.1+ MB
```

In [6]:
```
# lets remove unnecassary columns from the dataset

# Use the 'drop()' function to drop the unnecessary columns

data = data.drop(['color',
                  'director_facebook_likes',
                  'actor_3_facebook_likes',
                  'actor_1_facebook_likes',
                  'cast_total_facebook_likes',
                  'actor_2_facebook_likes',
                  'facenumber_in_poster',
                  'content_rating',
                  'country',
                  'movie_imdb_link',
                  'aspect_ratio',
                  'plot_keywords',
                  ],
                   axis = 1)
data.columns
```

Out[6]:
```
Index(['director_name', 'num_critic_for_reviews', 'duration', 'actor_2_nam
e',
       'gross', 'genres', 'actor_1_name', 'movie_title', 'num_voted_user
s',
       'actor_3_name', 'num_user_for_reviews', 'language', 'budget',
       'title_year', 'imdb_score', 'movie_facebook_likes'],
      dtype='object')
```

#Missing Values Imputation

In [7]:
```
# lets check the rows having high percentage of missing values in the datase

round(100*(data.isnull().sum()/len(data.index)), 2)
```

Out[7]:
```
director_name              2.06
num_critic_for_reviews     0.99
duration                   0.30
actor_2_name               0.26
gross                     17.53
genres                     0.00
actor_1_name               0.14
movie_title                0.00
num_voted_users            0.00
actor_3_name               0.46
num_user_for_reviews       0.42
language                   0.28
budget                     9.76
title_year                 2.14
imdb_score                 0.00
movie_facebook_likes       0.00
dtype: float64
```

In [8]:
```
# Since 'gross' and 'budget' columns have large number of NaN values, drop c
# 'isnan' function of NumPy alongwith a negation '~'

data = data[~np.isnan(data['gross'])]
data = data[~np.isnan(data['budget'])]

# Now lets again check the Missing Values column wise
data.isnull().sum()
```

Out[8]:
```
director_name              0
num_critic_for_reviews     1
duration                   1
actor_2_name               5
gross                      0
genres                     0
actor_1_name               3
movie_title                0
num_voted_users            0
actor_3_name              10
num_user_for_reviews       0
language                   4
budget                     0
title_year                 0
imdb_score                 0
movie_facebook_likes       0
dtype: int64
```

In [9]:
```
# The rows for which the sum of Null is less than two are retained

data = data[data.isnull().sum(axis=1) <= 2]
data.isnull().sum()
```

Out[9]:
```
director_name             0
num_critic_for_reviews    1
duration                  1
actor_2_name              2
gross                     0
genres                    0
actor_1_name              0
movie_title               0
num_voted_users           0
actor_3_name              7
num_user_for_reviews      0
language                  4
budget                    0
title_year                0
imdb_score                0
movie_facebook_likes      0
dtype: int64
```

In [10]:
```
# lets impute the missing values

# using mean for numerical columns
data['num_critic_for_reviews'].fillna(data['num_critic_for_reviews'].mean()
data['duration'].fillna(data['duration'].mean(), inplace = True)

# using mode for categorical column
data['language'].fillna(data['language'].mode()[0], inplace = True)

# As we know that We cannot use statistical values for imputing the missing
# actor names with "Unknown Actor"

data['actor_2_name'].fillna('Unknown Actor', inplace = True)
data['actor_3_name'].fillna('Unknown Actor', inplace = True)

# as we imputed all the missing values lets check the no. of total missing
data.isnull().sum().sum()
```

Out[10]: 0

#Feature Engineering

In [11]:
```
# Lets convert the gross and budget from $ to Million $ to make our analysis

data['gross'] = data['gross']/1000000
data['budget'] = data['budget']/1000000
```

In [12]:
```
# lets create a Profit column using the Budget and Gross

data['Profit'] = data['gross'] - data['budget']

# lets also check the name of Top 10 Profitable Movies
data[['Profit','movie_title']].sort_values(by = 'Profit', ascending  = Fals
```

Out[12]:

	Profit	movie_title
0	523.505847	Avatar
29	502.177271	Jurassic World
26	458.672302	Titanic
3024	449.935665	Star Wars: Episode IV - A New Hope
3080	424.449459	E.T. the Extra-Terrestrial
794	403.279547	The Avengers
17	403.279547	The Avengers
509	377.783777	The Lion King
240	359.544677	Star Wars: Episode I - The Phantom Menace
66	348.316061	The Dark Knight

In [13]:
```
# By looking at the above result we can easily analyze that there are some d

# lets print the no. of rows before removing Duplicates
print("No. of Rows Before Removing Duplicates: ",data.shape[0])

# so lets remove all the duplicates from the data
data.drop_duplicates(subset = None, keep = 'first', inplace = True)

# lets print the no. of rows after removing Duplicates
print("No. of Rows After Removing Duplicates: ",data.shape[0])
```

```
No. of Rows Before Removing Duplicates:  3888
No. of Rows After Removing Duplicates:  3853
```

#Top 10 Movies with Highest profit

In [14]:
```
# Lets check the Top 10 Profitable Movies Again
data[['movie_title','Profit']].sort_values(by = 'Profit', ascending  = False
```

Out[14]:

	movie_title	Profit
0	Avatar	523.505847
29	Jurassic World	502.177271
26	Titanic	458.672302
3024	Star Wars: Episode IV - A New Hope	449.935665
3080	E.T. the Extra-Terrestrial	424.449459
17	The Avengers	403.279547
509	The Lion King	377.783777
240	Star Wars: Episode I - The Phantom Menace	359.544677
66	The Dark Knight	348.316061
439	The Hunger Games	329.999255

#Manupulating the Duration and Language Collumn

In [15]:
```python
# lets check the values in the language column
data['language'].value_counts()
```

Out[15]:
```
language
English       3674
French          37
Spanish         26
Mandarin        14
German          13
Japanese        12
Hindi           10
Cantonese        8
Italian          7
Portuguese       5
Korean           5
Norwegian        4
Thai             3
Hebrew           3
Persian          3
Danish           3
Dutch            3
Dari             2
Indonesian       2
Aboriginal       2
Arabic           1
Russian          1
Vietnamese       1
Dzongkha         1
Romanian         1
Zulu             1
Bosnian          1
Czech            1
Icelandic        1
Hungarian        1
Mongolian        1
Aramaic          1
Telugu           1
Kazakh           1
Maya             1
Filipino         1
Swedish          1
Name: count, dtype: int64
```

In [16]:
```
# Looking at the above output we can easily observe that out of 3,500 movies

# so it is better to keep only two languages that is English and Foreign
def language(x):
    if x == 'English':
        return 'English'
    else:
        return 'Foreign'

# lets apply the function on the language column
data['language'] = data['language'].apply(language)

# lets check the values again
data['language'].value_counts()
```

Out[16]:
```
language
English    3674
Foreign     179
Name: count, dtype: int64
```

In [17]:
```
# The Duration of Movies is not varying a lot but we know that most of the u
# duration movies. we can categorize the movies in two part i.e., short and
# lets define a function for categorizing Duration of Movies
def duration(x):
    if x <= 120:
        return 'Short'
    else:
        return 'Long'

# lets apply this function on the duration column
data['duration'] = data['duration'].apply(duration)

# lets check the values of Duration column
data['duration'].value_counts()
```

Out[17]:
```
duration
Short    2936
Long      917
Name: count, dtype: int64
```

In [18]:
```
# lets also check the values in the Genres Column

data['genres'].value_counts()
```

Out[18]:
```
genres
Drama                                   153
Comedy|Drama|Romance                    151
Comedy|Drama                            147
Comedy                                  145
Comedy|Romance                          135
                                       ...
Action|Crime|Drama|Thriller|War           1
Adventure|Comedy|Family|Musical           1
Action|Adventure|Family|Fantasy|Sci-Fi    1
Action|Drama|Mystery|Thriller|War         1
Comedy|Crime|Horror                       1
Name: count, Length: 762, dtype: int64
```

In [19]:
```
data['genres'].str.split('|')[0]
```

Out[19]:
```
['Action', 'Adventure', 'Fantasy', 'Sci-Fi']
```

In [20]:
```
# we can see from the above output that most of the movies are having a lot
# also, a movie can have so many genres so lets keep four genres

data['Moviegenres'] = data['genres'].str.split('|')
data['Genre1'] = data['Moviegenres'].apply(lambda x: x[0])

# Some of the movies have only one genre. In such cases, assign the same gen
data['Genre2'] = data['Moviegenres'].apply(lambda x: x[1] if len(x) > 1 els
data['Genre3'] = data['Moviegenres'].apply(lambda x: x[2] if len(x) > 2 els
data['Genre4'] = data['Moviegenres'].apply(lambda x: x[3] if len(x) > 3 els

# lets check the head of the  data
data[['genres','Genre1','Genre2','Genre3','Genre4']].head(5)
```

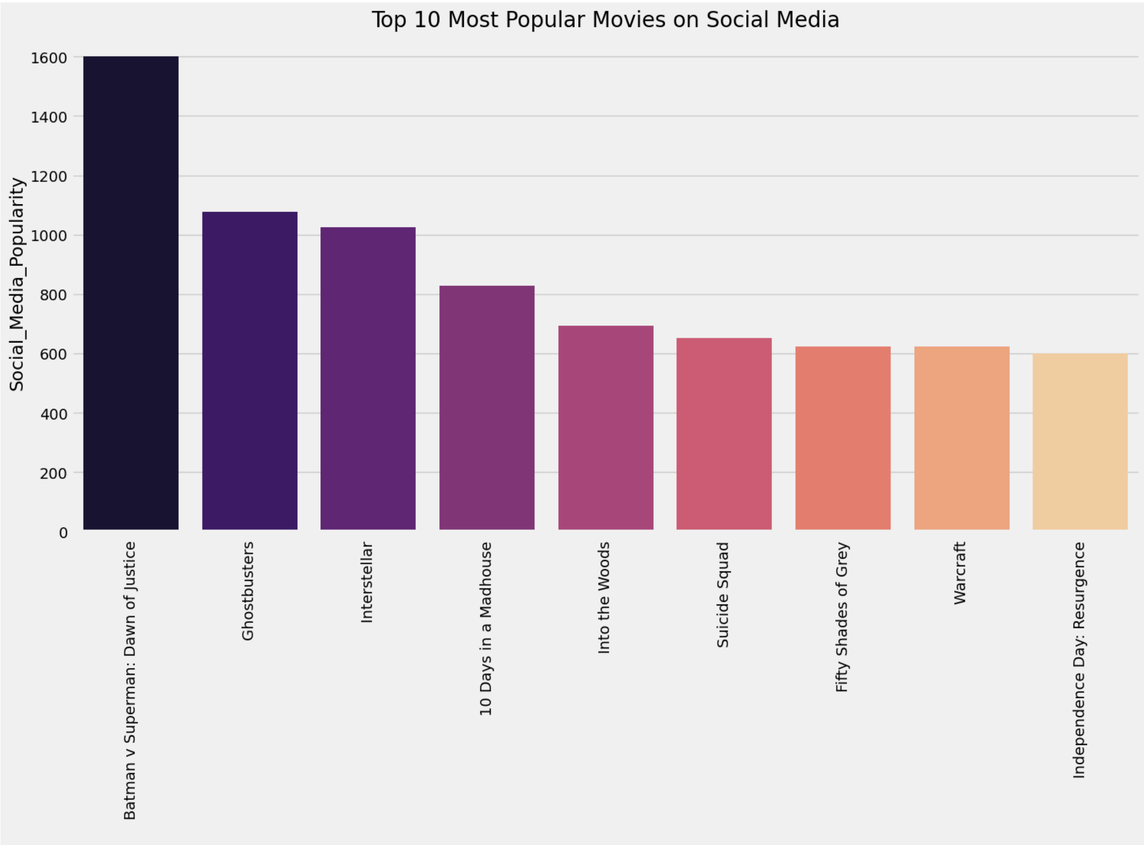
Out[20]:

	genres	Genre1	Genre2	Genre3	Genre4
0	Action|Adventure|Fantasy|Sci-Fi	Action	Adventure	Fantasy	Sci-Fi
1	Action|Adventure|Fantasy	Action	Adventure	Fantasy	Action
2	Action|Adventure|Thriller	Action	Adventure	Thriller	Action
3	Action|Thriller	Action	Thriller	Action	Action
5	Action|Adventure|Sci-Fi	Action	Adventure	Sci-Fi	Action

#Data Visualisation

In [21]:
```
# lets also calculate the Social Media Popularity of a Movie

# to calculate popularity of a movie, we can aggregate No. of voted users, N
data['Social_Media_Popularity'] = (data['num_user_for_reviews']/
                                   data['num_voted_users'])*data['movie_fac

# lets also check the Top 10 Most Popular Movies on Social Media
x = data[['movie_title','Social_Media_Popularity']].sort_values(by = 'Socia
                                                            ascending =
print(x)

sns.barplot(x=x['movie_title'], y=x['Social_Media_Popularity'], palette = 'm
plt.title('Top 10 Most Popular Movies on Social Media', fontsize = 20)
plt.xticks(rotation = 90, fontsize = 14)
plt.xlabel(' ')
plt.show()
```

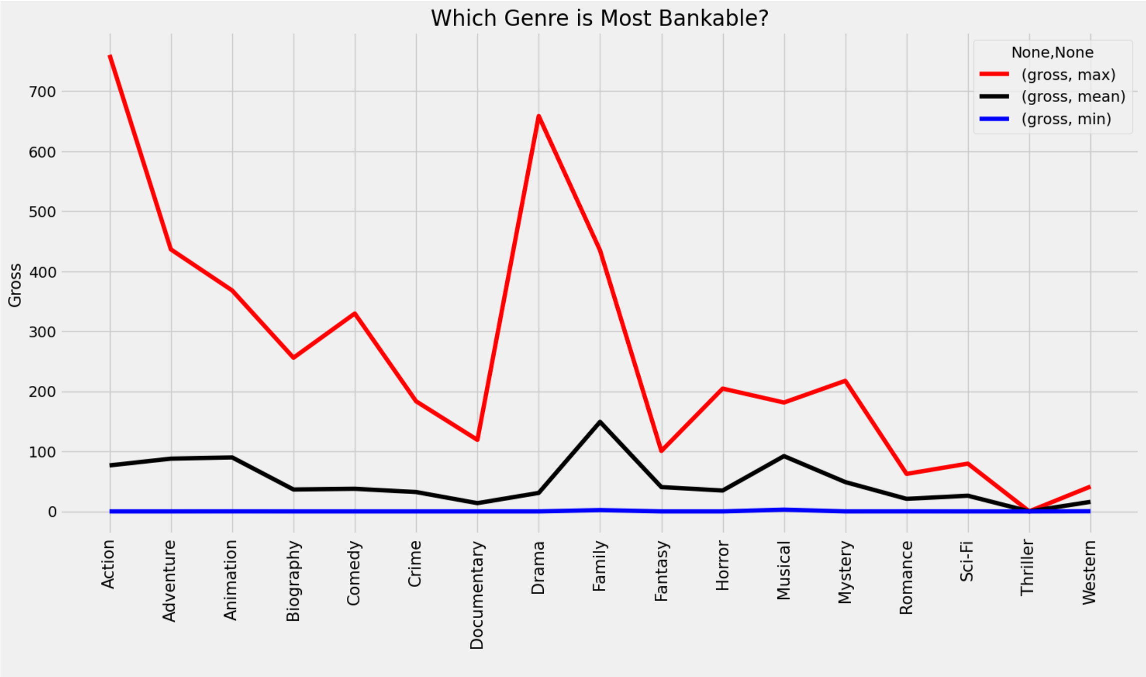
```
   index                         movie_title  Social_Media_Popularity
0     10  Batman v Superman: Dawn of Justice              1599.794424
1    150                        Ghostbusters              1076.336425
2   1582                        Ghostbusters              1075.827482
3     96                        Interstellar              1024.560802
4   3015                 10 Days in a Madhouse              828.025478
5    945                       Into the Woods              692.937200
6     73                        Suicide Squad              652.816996
7   1190                  Fifty Shades of Grey              624.306881
8    108                            Warcraft              622.790277
9     92         Independence Day: Resurgence              599.274128
```



In [22]:

```
# Lets compare the Gross with Genres

# first group the genres and get max, min, and avg gross of the movies of th
display(data[['Genre1','gross',]].groupby(['Genre1']).agg(['max','mean','mi

# lets plot these values using lineplot
data[['Genre1','gross',]].groupby(['Genre1']).agg(['max','mean','min']).plo
plt.title('Which Genre is Most Bankable?', fontsize = 20)
plt.xticks(np.arange(17), ['Action', 'Adventure', 'Animation', 'Biography',
       'Documentary', 'Drama', 'Family', 'Fantasy', 'Horror', 'Musical',
       'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'Western'], rotation = 9
plt.ylabel('Gross', fontsize = 15)
plt.xlabel(' ',)
plt.show()

print('The Most Profitable Movie from each Genre')
display(data.loc[data.groupby(data['Genre1'])['Profit'].idxmax()][['Genre1'
                                    'movie_title','gross']].style.backgroun
```

	gross		
	max	mean	min
Genre1			
Action	760.505847	76.584686	0.000162
Adventure	436.471036	87.827145	0.004091
Animation	368.049635	89.873480	0.071442
Biography	255.950375	36.431983	0.012836
Comedy	329.691196	37.611935	0.000703
Crime	183.405771	32.223226	0.001111
Documentary	119.078393	13.704278	0.005858
Drama	658.672302	30.778967	0.002580
Family	434.949459	149.160478	2.119994
Fantasy	100.614858	40.483059	0.003478
Horror	204.565000	34.737117	0.005725
Musical	181.360000	92.084000	2.808000
Mystery	217.536138	48.822296	0.016066
Romance	62.453315	20.886339	0.076382
Sci-Fi	79.568000	26.071841	0.018195
Thriller	0.070071	0.040513	0.002468
Western	41.400000	15.914589	0.243768




```
The Most Profitable Movie from each Genre
```

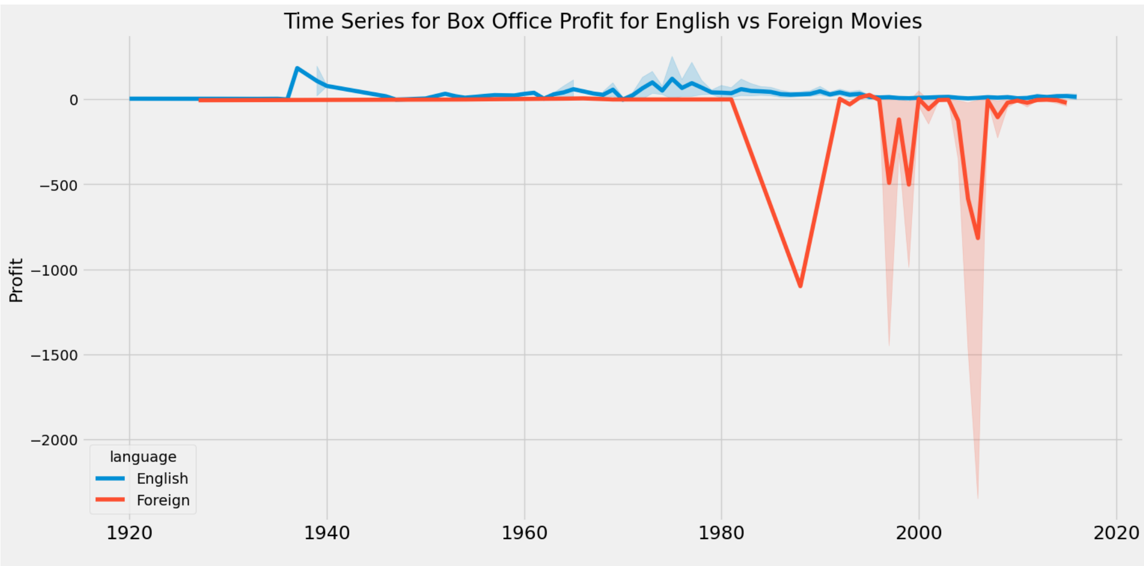
	Genre1	movie_title	gross
0	Action	Avatar	760.505847
509	Adventure	The Lion King	422.783777
521	Animation	Despicable Me 2	368.049635
1403	Biography	The Blind Side	255.950375
836	Comedy	Forrest Gump	329.691196
3466	Crime	The Godfather	134.821952
3583	Documentary	Fahrenheit 9/11	119.078393
26	Drama	Titanic	658.672302
3080	Family	E.T. the Extra-Terrestrial	434.949459
2485	Fantasy	The Others	96.471845
2916	Horror	The Exorcist	204.565000
3581	Musical	Grease	181.360000
208	Mystery	The Da Vinci Code	217.536138
928	Romance	The Adjustment Bureau	62.453315
2925	Sci-Fi	WarGames	79.568000
5034	Thriller	Cavite	0.070071
3540	Western	Pale Rider	41.400000

In [23]:

```
# lets convert year into integer
data['title_year'] = data['title_year'].astype('int')

print('Most Profitable Years in Box Office')
display(data[['title_year','language','Profit']].groupby(['language',
                                    'title_year']).agg('sum').sort_values(by
                                     ascending = False).head(10).style.back

# lets plot them
sns.lineplot(x=data['title_year'], y=data['Profit'], hue = data['language']
plt.title('Time Series for Box Office Profit for English vs Foreign Movies'
plt.xticks(fontsize = 18)
plt.xlabel(' ')
plt.show()

print("Movies that Made Huge Losses")
display(data[data['Profit'] < -2000][['movie_title',
                        'language','Profit']].style.background_gradient(cma
```

Most Profitable Years in Box Office

language	title_year	Profit
English	2014	2729.797944
	2012	2701.504634
	2015	2364.554417
	2002	2268.274235
	2009	2133.449256
	2013	2080.782304
	2003	1924.411513
	2007	1754.855579
	2001	1666.984435
	1994	1600.413059



Movies that Made Huge Losses

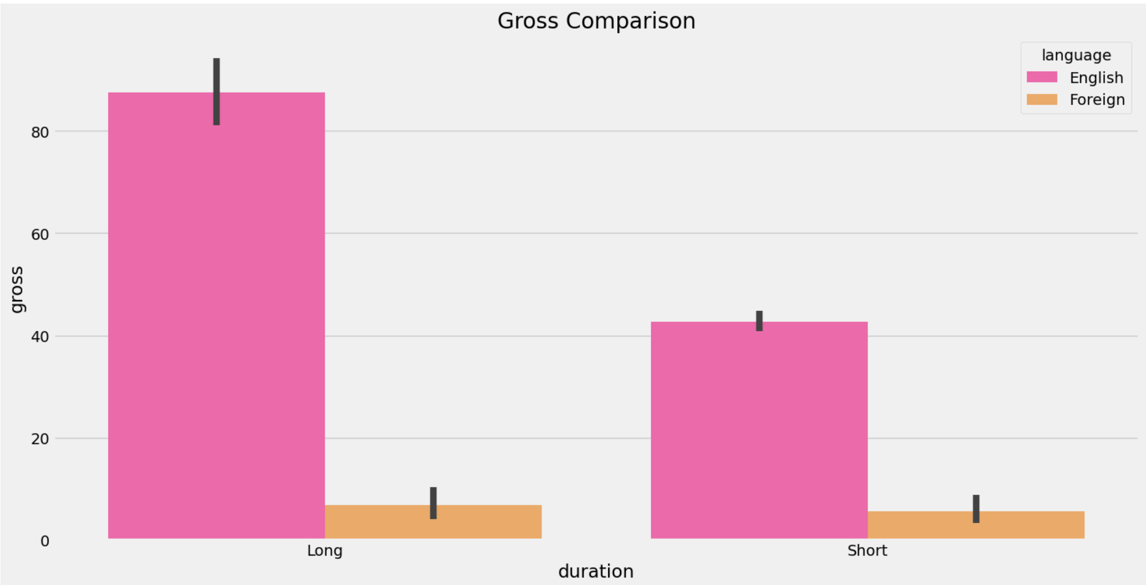
	movie_title	language	Profit
2323	Princess Mononoke	Foreign	-2397.701809
2334	Steamboy	Foreign	-2127.109510
2988	The Host	Foreign	-12213.298588
3005	Fateless	Foreign	-2499.804112
3859	Lady Vengeance	Foreign	-4199.788333

In [24]:
```
display(data[data['duration'] == 'Long'][['movie_title', 'duration', 'gross
                    'Profit']].sort_values(by = 'Profit',ascending = False)

display(data[data['duration'] == 'Short'][['movie_title', 'duration', 'gros
                    'Profit']].sort_values(by = 'Profit',ascending = False)

sns.barplot(x=data['duration'], y=data['gross'], hue = data['language'], pa
plt.title('Gross Comparison')
```

	movie_title	duration	gross	Profit
0	Avatar	Long	760.505847	523.505847
29	Jurassic World	Long	652.177271	502.177271
26	Titanic	Long	658.672302	458.672302
3024	Star Wars: Episode IV - A New Hope	Long	460.935665	449.935665
17	The Avengers	Long	623.279547	403.279547

	movie_title	duration	gross	Profit
3080	E.T. the Extra-Terrestrial	Short	434.949459	424.449459
509	The Lion King	Short	422.783777	377.783777
812	Deadpool	Short	363.024263	305.024263
521	Despicable Me 2	Short	368.049635	292.049635
338	Finding Nemo	Short	380.838870	286.838870

Out[24]: `Text(0.5, 1.0, 'Gross Comparison')`



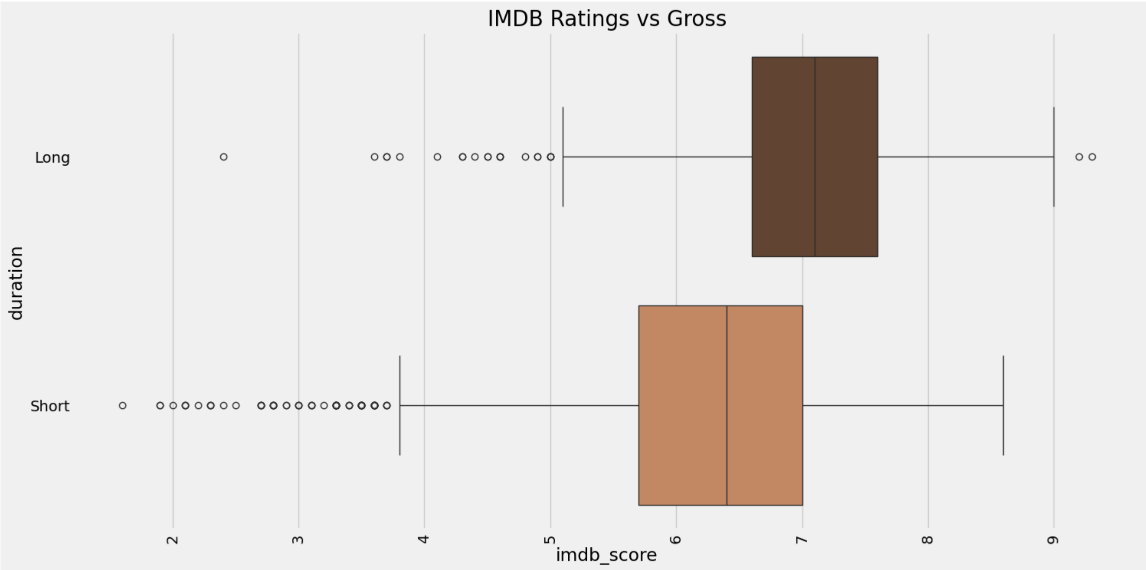
In [25]:
```
print("Average IMDB Score for Long Duration Movies is {0:.2f}".format(data[
print("Average IMDB Score for Short Duration Movies is {0:.2f}".format(data
```

```
Average IMDB Score for Long Duration Movies is 7.06
Average IMDB Score for Short Duration Movies is 6.28
```

In [26]:
```
print("\nHighest Rated Long Duration Movie\n",
    data[data['duration'] == 'Long'][['movie_title','imdb_score']].sort_valu
print("\nHighest Rated Short Duration Movie\n",
    data[data['duration'] == 'Short'][['movie_title','imdb_score']].sort_va
```

```

Highest Rated Long Duration Movie
                      movie_title  imdb_score
1937  The Shawshank Redemption          9.3

Highest Rated Short Duration Movie
                movie_title  imdb_score
3592  The Usual Suspects          8.6
```

In [27]:
```
sns.boxplot(x=data['imdb_score'], y=data['duration'], palette = 'copper')
plt.title('IMDB Ratings vs Gross', fontsize = 20)
plt.xticks(rotation = 90)
plt.show()
```



In [28]:
```python
def query_actors(x):
    # Create a boolean mask for each actor's column
    mask_a = data['actor_1_name'] == x
    mask_b = data['actor_2_name'] == x
    mask_c = data['actor_3_name'] == x

    # Combine the masks to get rows where any of the three actors match
    combined_mask = mask_a | mask_b | mask_c

    # Use the combined mask to filter the data
    result = data[combined_mask]

    # Select only the desired columns
    result = result[['movie_title', 'budget', 'gross', 'title_year', 'genre

    return result
```

In [29]:
```
#  usage of the function to query actor-specific data
actor_name = "Leonardo DiCaprio"  # Replace with the actor's name you want 
result = query_actors(actor_name)
print(result)
```

```
                          movie_title  budget       gross  title_year  \
26                            Titanic   200.0  658.672302        1997   
50                    The Great Gatsby   105.0  144.812796        2013   
97                          Inception   160.0  292.568851        2010   
179                       The Revenant   135.0  183.635922        2015   
257                        The Aviator   110.0  102.608827        2004   
296                   Django Unchained   100.0  162.804648        2012   
307                      Blood Diamond   100.0   57.366262        2006   
308            The Wolf of Wall Street   100.0  116.866727        2013   
326                 Gangs of New York   100.0   77.679638        2002   
361                       The Departed    90.0  132.373442        2006   
452                     Shutter Island    80.0  127.968405        2010   
641                       Body of Lies    70.0   39.380442        2008   
911                Catch Me If You Can    52.0  164.435221        2002   
990                          The Beach    50.0   39.778599        2000   
1114               Revolutionary Road    35.0   22.877808        2008   
1422           The Man in the Iron Mask    35.0   56.876365        1998   
1453                          J. Edgar    35.0   37.304950        2011   
1560            The Quick and the Dead    32.0   18.636537        1995   
2067                      Marvin's Room    23.0   12.782508        1996   
2757                     Romeo + Juliet    14.5   46.338728        1996   
3058      What's Eating Gilbert Grape    11.0    9.170214        1993   
3476                  The Great Gatsby   105.0  144.812796        2013   

                                  genres language  imdb_score  
26                         Drama|Romance  English         7.7  
50                         Drama|Romance  English         7.3  
97      Action|Adventure|Sci-Fi|Thriller  English         8.8  
179    Adventure|Drama|Thriller|Western  English         8.1  
257                      Biography|Drama  English         7.5  
296                        Drama|Western  English         8.5  
307             Adventure|Drama|Thriller  English         8.0  
308         Biography|Comedy|Crime|Drama  English         8.2  
326                          Crime|Drama  English         7.5  
361                 Crime|Drama|Thriller  English         8.5  
452                     Mystery|Thriller  English         8.1  
641                Action|Drama|Thriller  English         7.1  
911                Biography|Crime|Drama  English         8.0  
990             Adventure|Drama|Thriller  English         6.6  
1114                       Drama|Romance  English         7.3  
1422                    Action|Adventure  English         6.4  
1453               Biography|Crime|Drama  English         6.6  
1560             Action|Thriller|Western  English         6.4  
2067                               Drama  English         6.7  
2757                       Drama|Romance  English         6.8  
3058                       Drama|Romance  English         7.8  
3476                       Drama|Romance  English         7.3  
```

In [30]:
```python
def actors_report(x):
    a = data[data['actor_1_name'] == x]
    b = data[data['actor_2_name'] == x]
    c = data[data['actor_3_name'] == x]

    # Concatenate the dataframes vertically
    frames = [a, b, c]
    y = pd.concat(frames, axis=0)

    print("Time:", y['title_year'].min(), y['title_year'].max())
    print("Max Gross : {0:.2f} Millions".format(y['gross'].max()))
    print("Avg Gross : {0:.2f} Millions".format(y['gross'].mean()))
    print("Min Gross : {0:.2f} Millions".format(y['gross'].min()))
    print("Number of 100 Million Movies :", y[y['gross'] > 100].shape[0])
    print("Avg IMDB Score : {0:.2f}".format(y['imdb_score'].mean()))
    print("Most Common Genres:\n", y['Genre1'].value_counts().head())

# usage of the function to generate a report for the actor
actors_report('Leonardo DiCaprio')
```

```
Time: 1993 2015
Max Gross : 658.67 Millions
Avg Gross : 120.44 Millions
Min Gross : 9.17 Millions
Number of 100 Million Movies : 11
Avg IMDB Score : 7.51
Most Common Genres:
 Genre1
Drama        8
Action       4
Biography    4
Adventure    3
Crime        2
Name: count, dtype: int64
```

In [31]:
```
# Lets compare Brad Pitt, Leonardo Caprio and Tom Cruise

def critically_acclaimed_actors(m):
    a = data[data['actor_1_name'] == m]
    b = data[data['actor_2_name'] == m]
    c = data[data['actor_3_name'] == m]

    # Concatenate the dataframes vertically
    frames = [a, b, c]
    y = pd.concat(frames, axis=0)

    return y['num_critic_for_reviews'].sum().astype('int')

print("Number of Critics Reviews for Brad Pitt")
print(critically_acclaimed_actors('Brad Pitt'))

print("Number of Critics Reviews for Leonardo DiCaprio")
print(critically_acclaimed_actors('Leonardo DiCaprio'))

print("Number of Critics Reviews for Tom Cruise")
print(critically_acclaimed_actors('Tom Cruise'))
```

```
Number of Critics Reviews for Brad Pitt
7814
Number of Critics Reviews for Leonardo DiCaprio
7014
Number of Critics Reviews for Tom Cruise
6740
```

In [32]:
```
!pip install ipywidgets
```

```
Requirement already satisfied: ipywidgets in c:\users\ishan\appdata\local
\programs\python\python311\lib\site-packages (8.1.1)
Requirement already satisfied: comm>=0.1.3 in c:\users\ishan\appdata\local
\programs\python\python311\lib\site-packages (from ipywidgets) (0.1.3)
Requirement already satisfied: ipython>=6.1.0 in c:\users\ishan\appdata\lo
cal\programs\python\python311\lib\site-packages (from ipywidgets) (8.14.0)
Requirement already satisfied: traitlets>=4.3.1 in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from ipywidgets) (5.9.
0)
Requirement already satisfied: widgetsnbextension~=4.0.9 in c:\users\ishan
\appdata\local\programs\python\python311\lib\site-packages (from ipywidget
s) (4.0.9)
Requirement already satisfied: jupyterlab-widgets~=3.0.9 in c:\users\ishan
\appdata\local\programs\python\python311\lib\site-packages (from ipywidget
s) (3.0.9)
Requirement already satisfied: backcall in c:\users\ishan\appdata\local\pr
ograms\python\python311\lib\site-packages (from ipython>=6.1.0->ipywidget
s) (0.2.0)
Requirement already satisfied: decorator in c:\users\ishan\appdata\local\p
rograms\python\python311\lib\site-packages (from ipython>=6.1.0->ipywidget
s) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\ishan\appdata\local
\programs\python\python311\lib\site-packages (from ipython>=6.1.0->ipywidg
ets) (0.18.2)
Requirement already satisfied: matplotlib-inline in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from ipython>=6.1.0->i
pywidgets) (0.1.6)
Requirement already satisfied: pickleshare in c:\users\ishan\appdata\local
\programs\python\python311\lib\site-packages (from ipython>=6.1.0->ipywidg
ets) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30 in
c:\users\ishan\appdata\local\programs\python\python311\lib\site-packages
(from ipython>=6.1.0->ipywidgets) (3.0.38)
Requirement already satisfied: pygments>=2.4.0 in c:\users\ishan\appdata\l
ocal\programs\python\python311\lib\site-packages (from ipython>=6.1.0->ipy
widgets) (2.15.1)
Requirement already satisfied: stack-data in c:\users\ishan\appdata\local
\programs\python\python311\lib\site-packages (from ipython>=6.1.0->ipywidg
ets) (0.6.2)
Requirement already satisfied: colorama in c:\users\ishan\appdata\local\pr
ograms\python\python311\lib\site-packages (from ipython>=6.1.0->ipywidget
s) (0.4.6)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in c:\users\ishan\appda
ta\local\programs\python\python311\lib\site-packages (from jedi>=0.16->ipy
thon>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: wcwidth in c:\users\ishan\appdata\local\pro
grams\python\python311\lib\site-packages (from prompt-toolkit!=3.0.37,<3.
1.0,>=3.0.30->ipython>=6.1.0->ipywidgets) (0.2.6)
Requirement already satisfied: executing>=1.2.0 in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from stack-data->ipyth
on>=6.1.0->ipywidgets) (1.2.0)
Requirement already satisfied: asttokens>=2.1.0 in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from stack-data->ipyth
on>=6.1.0->ipywidgets) (2.2.1)
Requirement already satisfied: pure-eval in c:\users\ishan\appdata\local\p
rograms\python\python311\lib\site-packages (from stack-data->ipython>=6.1.
0->ipywidgets) (0.2.2)
Requirement already satisfied: six in c:\users\ishan\appdata\local\program
s\python\python311\lib\site-packages (from asttokens>=2.1.0->stack-data->i
python>=6.1.0->ipywidgets) (1.16.0)
```

In [33]:

```
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

In [34]:

```
@interact
def show_movies_more_than(column='imdb_score', score=9.0):
    x = data.loc[data[column] > score][['title_year', 'movie_title', 'direc
    x = x.sort_values(by='imdb_score', ascending=False)
    x = x.drop_duplicates(keep='first')
    return x
```

column imdb_score

score 9.00

	title_year	movie_title	director_name	actor_1_name	actor_2_name	actor_3_name	
1937	1994	The Shawshank Redemption	Frank Darabont	Morgan Freeman	Jeffrey DeMunn	Bob Gunton	
3466	1972	The Godfather	Francis Ford Coppola	Al Pacino	Marlon Brando	Robert Duvall	1:

In [35]:

```
@interact
def show_articles_more_than(column=['budget','gross'], x=1000):
    return data.loc[data[column] > x][['movie_title','duration','gross','Pr
```

column budget

x 1000

	movie_title	duration	gross	Profit	imdb_score
2323	Princess Mononoke	Long	2.298191	-2397.701809	8.4
2334	Steamboy	Short	0.410388	-2127.109510	6.9
2988	The Host	Short	2.201412	-12213.298588	7.0
3005	Fateless	Long	0.195888	-2499.804112	7.1
3423	Akira	Long	0.439162	-1099.560838	8.1
3859	Lady Vengeance	Short	0.211667	-4199.788333	7.7

#Recommending Movies based on Languages

In [36]:

```
def recommend_lang(x):
    y = data[['language','movie_title','imdb_score']][data['language'] == x
    y = y.sort_values(by = 'imdb_score', ascending = False)
    return y.head(15)
```

In [37]:
```
recommended_movies = recommend_lang('English')
print(recommended_movies)
```

```
     language                                         movie_title  imdb_sco
re
1937  English                            The Shawshank Redemption
9.3
3466  English                                       The Godfather
9.2
66    English                                     The Dark Knight
9.0
2837  English                              The Godfather: Part II
9.0
339   English      The Lord of the Rings: The Return of the King
8.9
1874  English                                    Schindler's List
8.9
3355  English                                        Pulp Fiction
8.9
97    English                                           Inception
8.8
836   English                                        Forrest Gump
8.8
2051  English     Star Wars: Episode V - The Empire Strikes Back
8.8
270   English  The Lord of the Rings: The Fellowship of the R...
8.8
683   English                                          Fight Club
8.8
3867  English                     One Flew Over the Cuckoo's Nest
8.7
3024  English                  Star Wars: Episode IV - A New Hope
8.7
1903  English                                          Goodfellas
8.7
```

#Recommending Movies Based on Actors

In [38]:
```
import pandas as pd

def recommend_movies_on_actors(x):
    a = data[['movie_title', 'imdb_score']][data['actor_1_name'] == x]
    b = data[['movie_title', 'imdb_score']][data['actor_2_name'] == x]
    c = data[['movie_title', 'imdb_score']][data['actor_3_name'] == x]

    # Concatenate the dataframes vertically
    frames = [a, b, c]
    a = pd.concat(frames)

    a = a.sort_values(by='imdb_score', ascending=False)
    return a.head(15)
```

In [39]:
```python
recommended_movies = recommend_movies_on_actors('Tom Cruise')
print(recommended_movies)
```

```
                                           movie_title  imdb_score
1868                                         Rain Man         8.0
75                                   Edge of Tomorrow         7.9
284                                    Minority Report        7.7
158                                   The Last Samurai        7.7
736                                         Collateral        7.6
1524                                     A Few Good Men        7.6
940   Interview with the Vampire: The Vampire Chroni...        7.6
155               Mission: Impossible - Ghost Protocol        7.4
135                 Mission: Impossible - Rogue Nation        7.4
671                                    Eyes Wide Shut        7.3
930                                     Jerry Maguire        7.3
3128                                    The Outsiders        7.2
2768                       Born on the Fourth of July        7.2
370                                          Valkyrie        7.1
438                               Mission: Impossible        7.1
```

#Recommending similar Genres

In [40]:
```
!pip install mlxtend
```

```
Requirement already satisfied: mlxtend in c:\users\ishan\appdata\local\pro
grams\python\python311\lib\site-packages (0.23.0)
Requirement already satisfied: scipy>=1.2.1 in c:\users\ishan\appdata\loca
l\programs\python\python311\lib\site-packages (from mlxtend) (1.11.0)
Requirement already satisfied: numpy>=1.16.2 in c:\users\ishan\appdata\loc
al\programs\python\python311\lib\site-packages (from mlxtend) (1.25.0)
Requirement already satisfied: pandas>=0.24.2 in c:\users\ishan\appdata\lo
cal\programs\python\python311\lib\site-packages (from mlxtend) (2.0.2)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\ishan\appda
ta\local\programs\python\python311\lib\site-packages (from mlxtend) (1.2.
2)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from mlxtend) (3.7.1)
Requirement already satisfied: joblib>=0.13.2 in c:\users\ishan\appdata\lo
cal\programs\python\python311\lib\site-packages (from mlxtend) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from matplotlib>=3.0.0
->mlxtend) (1.1.0)
Requirement already satisfied: cycler>=0.10 in c:\users\ishan\appdata\loca
l\programs\python\python311\lib\site-packages (from matplotlib>=3.0.0->mlx
tend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from matplotlib>=3.0.0
->mlxtend) (4.40.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from matplotlib>=3.0.0
->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\ishan\appdata\l
ocal\programs\python\python311\lib\site-packages (from matplotlib>=3.0.0->
mlxtend) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\ishan\appdata\loc
al\programs\python\python311\lib\site-packages (from matplotlib>=3.0.0->ml
xtend) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\ishan\appdata
\local\programs\python\python311\lib\site-packages (from matplotlib>=3.0.0
->mlxtend) (3.1.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ishan\appd
ata\local\programs\python\python311\lib\site-packages (from matplotlib>=3.
0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\ishan\appdata\loca
l\programs\python\python311\lib\site-packages (from pandas>=0.24.2->mlxten
d) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in c:\users\ishan\appdata\lo
cal\programs\python\python311\lib\site-packages (from pandas>=0.24.2->mlxt
end) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\ishan\appd
ata\local\programs\python\python311\lib\site-packages (from scikit-learn>=
1.0.2->mlxtend) (3.1.0)
Requirement already satisfied: six>=1.5 in c:\users\ishan\appdata\local\pr
ograms\python\python311\lib\site-packages (from python-dateutil>=2.7->matp
lotlib>=3.0.0->mlxtend) (1.16.0)
```

In [41]:
```
from mlxtend.preprocessing import TransactionEncoder

x = data['genres'].str.split('|')
te = TransactionEncoder()
x = te.fit_transform(x)
x = pd.DataFrame(x, columns = te.columns_)

# lets check the head of x
x.head()
```

Out[41]:

	Action	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drama	Family
0	True	True	False	False	False	False	False	False	False
1	True	True	False	False	False	False	False	False	False
2	True	True	False	False	False	False	False	False	False
3	True	False	False	False	False	False	False	False	False
4	True	True	False	False	False	False	False	False	False

5 rows × 23 columns

In [42]:
```
# lets convert this data into boolean so that we can perform calculations
genres = x.astype('int')

genres.head()
```

Out[42]:

	Action	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drama	Family
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	1	1	0	0	0	0	0	0	0

5 rows × 23 columns

In [43]:
```python
# now, lets insert the movie titles in the first column, so that we can bett
genres.insert(0, 'movie_title', data['movie_title'])

genres.head()
```

Out[43]:

	movie_title	Action	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drar
0	Avatar	1	1	0	0	0	0	0	
1	Pirates of the Caribbean: At World's End	1	1	0	0	0	0	0	
2	Spectre	1	1	0	0	0	0	0	
3	The Dark Knight Rises	1	0	0	0	0	0	0	
4	NaN	1	1	0	0	0	0	0	

5 rows × 24 columns

In [44]:
```python
# lets set these movie titles as index of the data
genres = genres.set_index('movie_title')
genres.head()
```

Out[44]:

nimation	Biography	Comedy	Crime	Documentary	Drama	Family	Fantasy	...	Music	Musi
0	0	0	0	0	0	0	1	...	0	
0	0	0	0	0	0	0	1	...	0	
0	0	0	0	0	0	0	0	...	0	
0	0	0	0	0	0	0	0	...	0	
0	0	0	0	0	0	0	0	...	0	

In [45]:
```python
# making a recommendation engine for getting similar genres

def recommendation_genres(gen):
    gen = genres[gen]
    similar_genres = genres.corrwith(gen)
    similar_genres = similar_genres.sort_values(ascending=False)
    similar_genres = similar_genres.iloc[1:]
    return similar_genres.head(3)
```

In [46]:
```python
recommendation_genres('Action')
```

Out[46]:
```
Adventure    0.320532
Thriller     0.303708
Sci-Fi       0.295018
dtype: float64
```

#Recommending similar Movies

In [47]:
```python
# lets make a sparse matrix to recommend the movies

x = genres.transpose()
x.head()
```

Out[47]:

movie_title	Avatar	Pirates of the Caribbean: At World's End	Spectre	The Dark Knight Rises	NaN	John Carter	Spider-Man 3	Tangled	Avengers: Age of Ultron
Action	1	1	1	1	1	1	0	1	0
Adventure	1	1	1	0	1	1	1	1	1
Animation	0	0	0	0	0	0	1	0	0
Biography	0	0	0	0	0	0	0	0	0
Comedy	0	0	0	0	0	0	1	0	0

5 rows × 3853 columns

In [48]:
```python
# making a recommendation engine for getting similar movies

def recommendation_movie(movie):
    movie = x[movie+'\xa0']
    similar_movies = x.corrwith(movie)
    similar_movies = similar_movies.sort_values(ascending=False)
    similar_movies = similar_movies.iloc[1:]
    return similar_movies.head(20)
```

In [49]:
```python
# lets test on some results
recommendation_movie('The Avengers')
```

Out[49]:
```
movie_title
The Avengers                                    1.0
Transformers: Dark of the Moon                  1.0
Rapa Nui                                        1.0
American Reunion                                1.0
Lost in Space                                   1.0
NaN                                             1.0
Risen                                           1.0
The Girl with the Dragon Tattoo                 1.0
NaN                                             1.0
The Flowers of War                              1.0
American Sniper                                 1.0
Pirates of the Caribbean: At World's End        1.0
Edge of Darkness                                1.0
The Sorcerer's Apprentice                       1.0
The Hobbit: The Battle of the Five Armies       1.0
Legally Blonde                                  1.0
Quantum of Solace                               1.0
Australia                                       1.0
X-Men Origins: Wolverine                        1.0
R.I.P.D.                                        1.0
dtype: float64
```

In [ ]: