Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
warnings.simplefilter('ignore')
```

In [2]:

```
movies = pd.read_csv('movies.csv')
credits = pd.read_csv('credits.csv')
```

In [3]:

```
movies.head()
```

Out[3]:

	budget	genres	homepage	id	keywords	original
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

In [4]:

```
credits.head()
```

Out[4]:

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

find out Common columns between movies and credits dataset so that we can merge them.

In [5]:

```
cm_cols = [col for col in movies if col in credits]
cm_cols
```

Out[5]:

```
['title']
```

In [6]:

```
original_df = pd.merge(movies , credits , on = 'title')
original_df.head()
```

Out[6]:

	budget	genres	homepage	id	keywords	original
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

5 rows × 23 columns

In [7]:

```
df = original_df.copy()
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4809 entries, 0 to 4808
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   budget                4809 non-null   int64  
 1   genres                4809 non-null   object 
 2   homepage              1713 non-null   object 
 3   id                    4809 non-null   int64  
 4   keywords              4809 non-null   object 
 5   original_language     4809 non-null   object 
 6   original_title        4809 non-null   object 
 7   overview              4806 non-null   object 
 8   popularity            4809 non-null   float64
 9   production_companies  4809 non-null   object 
 10  production_countries  4809 non-null   object 
 11  release_date          4808 non-null   object 
 12  revenue               4809 non-null   int64  
 13  runtime               4807 non-null   float64
 14  spoken_languages      4809 non-null   object 
 15  status                4809 non-null   object 
 16  tagline               3965 non-null   object 
 17  title                 4809 non-null   object 
 18  vote_average          4809 non-null   float64
 19  vote_count            4809 non-null   int64  
 20  movie_id              4809 non-null   int64  
 21  cast                  4809 non-null   object 
 22  crew                  4809 non-null   object 
dtypes: float64(3), int64(5), object(15)
memory usage: 901.7+ KB
```

In [9]:

```
movies.shape , credits.shape
```

Out[9]:

```
((4803, 20), (4803, 4))
```

In [10]:

```
df.shape
```

Out[10]:

```
(4809, 23)
```

Handling Json columns

In [11]:

```
json_cols = ['genres' , 'keywords' , 'production_companies' , 'production_countries' , 'spo
df[json_cols].head()
```

Out[11]:

	genres	keywords	production_companies	production_countries	spoken_languages	
0	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 1463, "name": "culture clash"}, {"id":...	[{"name": "Ingenious Film Partners", "id": 289...	[{"iso_3166_1": "US", "name": "United States o...	[{"iso_639_1": "en", "name": "English"}, {"iso...	[{"c "cha S
1	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	[{"name": "Walt Disney Pictures", "id": 2}, {"...	[{"iso_3166_1": "US", "name": "United States o...	[{"iso_639_1": "en", "name": "English"}]	[{"c "cha " Jac
2	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 470, "name": "spy"}, {"id": 818, "name...	[{"name": "Columbia Pictures", "id": 5}, {"nam...	[{"iso_3166_1": "GB", "name": "United Kingdom"...	[{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...	[{"c "cha
3	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	[{"id": 849, "name": "dc comics"}, {"id": 853,...	[{"name": "Legendary Pictures", "id": 923}, {"...	[{"iso_3166_1": "US", "name": "United States o...	[{"iso_639_1": "en", "name": "English"}]	[{"c "cha '
4	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 818, "name": "based on novel"}, {"id":...	[{"name": "Walt Disney Pictures", "id": 2}]	[{"iso_3166_1": "US", "name": "United States o...	[{"iso_639_1": "en", "name": "English"}]	[{"c "cha

In [12]:

```
import ast
```

In [13]:

```
df.genres[0]
```

Out[13]:

```
'[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14,
"name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]'
```

This fucntion will return a list of genres from genres columns.

In [14]:

```
def convert(data):
    genres_list = []
    for i in ast.literal_eval(data ):
        genres_list.append(i['name'])
    return genres_list
```

In [15]:

```
df['genres'] = df['genres'].apply(convert)
```

Previosuly genres:[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]

Now: ['Action', 'Adventure', 'Fantasy', 'Science Fiction']

In [16]:

```
df['genres'][0:5]
```

Out[16]:

```
0    [Action, Adventure, Fantasy, Science Fiction]
1                   [Adventure, Fantasy, Action]
2                     [Action, Adventure, Crime]
3               [Action, Crime, Drama, Thriller]
4           [Action, Adventure, Science Fiction]
Name: genres, dtype: object
```

In [17]:

```
df['keywords'] = df['keywords'].apply(convert)
df['production_companies'] =df['production_companies'].apply(convert)
df['production_countries'] =df['production_countries'].apply(convert)
```

from cast columns we will only select first 4 cast

In [18]:

```
def fetch_casts(data):
    cnt = 0
    genres_list = []
    for i in ast.literal_eval(data ):
        if cnt<4:
            genres_list.append(i['name'])
        cnt = cnt + 1
        if cnt>3:
            break


    return genres_list
```

In [19]:

```
df['cast'] = df['cast'].apply(fetch_casts)
```

In [20]:

```
df['cast'][0]
```

Out[20]:

```
['Sam Worthington', 'Zoe Saldana', 'Sigourney Weaver', 'Stephen Lang']
```

From crew columns extracting director name

In [21]:

```
def fetch_director(data):
    for i in ast.literal_eval(data):
        if i['job']=='Director':   #there are mulitple key in crew colulmns from those key
            return i['name']
```

In [22]:

```
df['Director' ] =  df['crew'].apply(fetch_director)
df['Director'].head()
```

Out[22]:

```
0        James Cameron
1       Gore Verbinski
2           Sam Mendes
3    Christopher Nolan
4       Andrew Stanton
Name: Director, dtype: object
```

In [23]:

```
df.head()
```

Out[23]:

	budget	genres	homepage	id	keywords	original_language	orig
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	http://www.avatarmovie.com/	19995	[culture clash, future, space war, space colon...	en	
1	300000000	[Adventure, Fantasy, Action]	http://disney.go.com/disneypictures/pirates/	285	[ocean, drug abuse, exotic island, east india ...	en	Pira C A

In [24]:

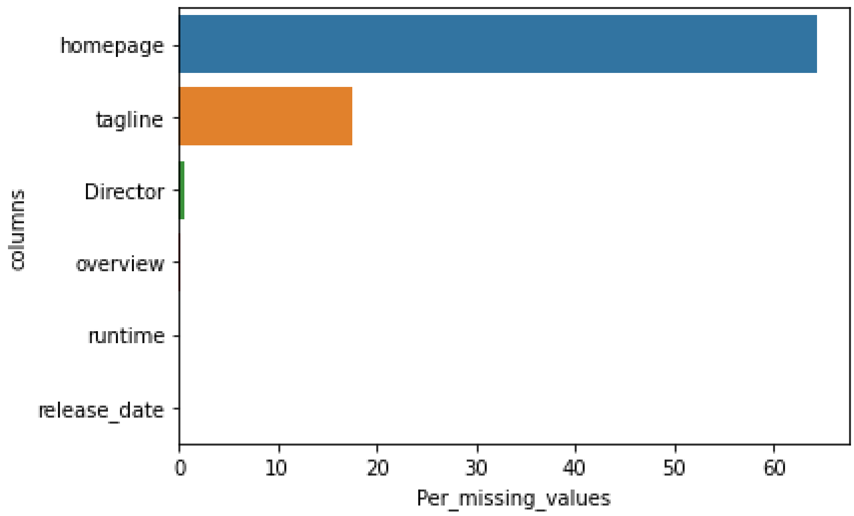
```
missing_values = ((df.isna().sum())*100 / len(df)).sort_values(ascending = False)
missing_values = missing_values[0:6].reset_index()
missing_values.columns = ['columns' , 'Per_missing_values']
missing_values
```

Out[24]:

	columns	Per_missing_values
0	homepage	64.379289
1	tagline	17.550426
2	Director	0.623830
3	overview	0.062383
4	runtime	0.041589
5	release_date	0.020794

In [25]:

```
sns.barplot(x = 'Per_missing_values' , y = 'columns' , data = missing_values)
plt.show()
```



In [26]:

```
df.drop(['homepage' , 'tagline'] , axis = 1 , inplace = True)
```

In [27]:

```
df.shape
```

Out[27]:

```
(4809, 22)
```

In [28]:

```
df.dropna(inplace = True)
```

In [29]:

```
df
```

Out[29]:

	budget	genres	id	keywords	original_language	original_title	overview
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	19995	[culture clash, future, space war, space colon...	en	Avatar	In the 22nd century, a paraplegic Marine is di...
1	300000000	[Adventure, Fantasy, Action]	285	[ocean, drug abuse, exotic island, east india ...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...
2	245000000	[Action, Adventure, Crime]	206647	[spy, based on novel, secret agent, sequel, mi...	en	Spectre	A cryptic message from Bond's past sends him o...
3	250000000	[Action, Crime, Drama, Thriller]	49026	[dc comics, crime fighter, terrorist, secret i...	en	The Dark Knight Rises	Following the death of District Attorney Harve...
4	260000000	[Action, Adventure, Science Fiction]	49529	[based on novel, mars, medallion, space travel...	en	John Carter	John Carter is a war-weary, former military ca...
...	...	...	...	...	...	...	...
4804	220000	[Action, Crime, Thriller]	9367	[united states–mexico barrier, legs, arms, pap...	es	El Mariachi	El Mariachi just wants to play his guitar and ...
4805	9000	[Comedy, Romance]	72766	[]	en	Newlyweds	A newlywed couple's honeymoon is upended by th...
4806	0	[Comedy, Drama, Romance, TV Movie]	231617	[date, love at first sight, narration, investi...	en	Signed, Sealed, Delivered	"Signed, Sealed, Delivered" introduces a dedic...
4807	0	[]	126186	[]	en	Shanghai Calling	When ambitious New York attorney Sam is sent t...

	budget	genres	id	keywords	original_language	original_title	overview
4808	0	[Documentary]	25975	[obsession, camcorder, crush, dream girl]	en	My Date with Drew	Ever since the second grade when he first saw ...

4776 rows × 22 columns

In [30]:

```
df.shape
```

Out[30]:

```
(4776, 22)
```

In [31]:

```
print('number of rows with 0 budget are: {}'.format(len(df[df['budget']==0])))
print('number of rows with 0 revnue are: {}'.format(len(df[df['revenue']==0])))
```

```
number of rows with 0 budget are: 1016
number of rows with 0 revnue are: 1399
```

Remove rows which has zero budget and zero revenue from our dataset

Top 10 Popular Movies based on budget

In [32]:

```
df = df[(df['revenue']!=0) & (df['budget']!=0)]
df.shape
```

Out[32]:

```
(3230, 22)
```

In [33]:

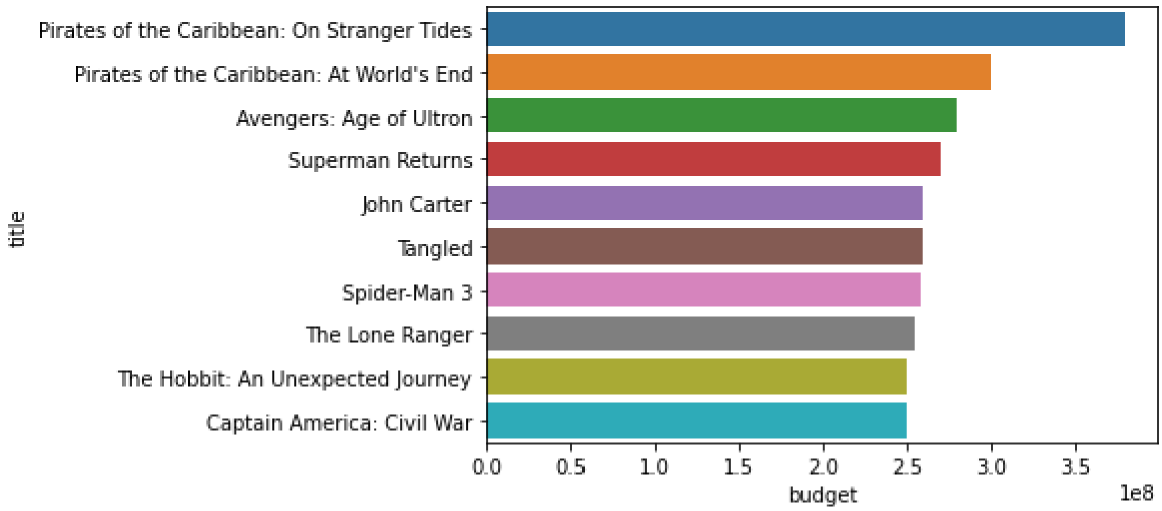
```
top10_budget_movies = df.sort_values(ascending = False , by = 'budget')[['title' , 'Directo
top10_budget_movies.reset_index(drop = True)
```

Out[33]:

	title	Director	budget
0	Pirates of the Caribbean: On Stranger Tides	Rob Marshall	380000000
1	Pirates of the Caribbean: At World's End	Gore Verbinski	300000000
2	Avengers: Age of Ultron	Joss Whedon	280000000
3	Superman Returns	Bryan Singer	270000000
4	John Carter	Andrew Stanton	260000000
5	Tangled	Byron Howard	260000000
6	Spider-Man 3	Sam Raimi	258000000
7	The Lone Ranger	Gore Verbinski	255000000
8	The Hobbit: An Unexpected Journey	Peter Jackson	250000000
9	Captain America: Civil War	Anthony Russo	250000000

In [34]:

```
sns.barplot(x = 'budget' , y = 'title' , data  = top10_budget_movies)
plt.show()
```



Top 10 Top 10 Popular Movies based on revenue

In [35]:

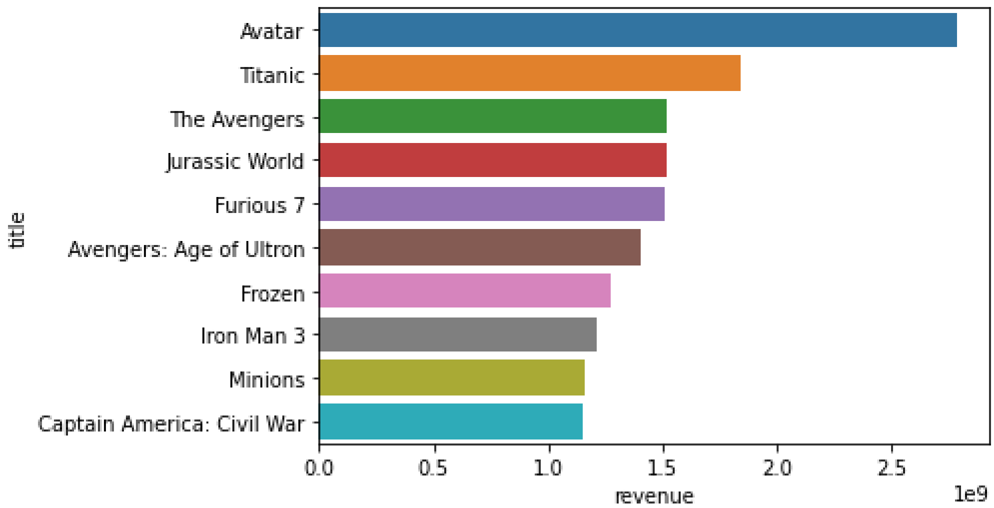
```
top10_gross_movies = df.sort_values(ascending = False , by = 'revenue')[['title' ,  'Direct
top10_gross_movies.reset_index(drop = True)
```

Out[35]:

	title	Director	revenue
0	Avatar	James Cameron	2787965087
1	Titanic	James Cameron	1845034188
2	The Avengers	Joss Whedon	1519557910
3	Jurassic World	Colin Trevorrow	1513528810
4	Furious 7	James Wan	1506249360
5	Avengers: Age of Ultron	Joss Whedon	1405403694
6	Frozen	Chris Buck	1274219009
7	Iron Man 3	Shane Black	1215439994
8	Minions	Kyle Balda	1156730962
9	Captain America: Civil War	Anthony Russo	1153304495

In [36]:

```
sns.barplot(x = 'revenue' , y = 'title' , data  = top10_gross_movies)
plt.show()
```



Top 10 Popular Movies based on Rating

In [37]:

```
top10_voted_movies = df.sort_values(ascending = False , by = 'vote_average')[['title' ,  'D
top10_voted_movies.reset_index(drop = True)
```

Out[37]:

	title	Director	vote_average
0	There Goes My Baby	Floyd Mutrux	8.5
1	The Shawshank Redemption	Frank Darabont	8.5
2	The Godfather	Francis Ford Coppola	8.4
3	Whiplash	Damien Chazelle	8.3
4	Pulp Fiction	Quentin Tarantino	8.3
5	Schindler's List	Steven Spielberg	8.3
6	Fight Club	David Fincher	8.3
7	The Godfather: Part II	Francis Ford Coppola	8.3
8	Spirited Away	Hayao Miyazaki	8.3
9	The Dark Knight	Christopher Nolan	8.2

Top 10 Popular Movies based on Populairty

In [38]:

```
top10_popularity_movies = df.sort_values(ascending = False , by = 'popularity')[['title' ,
top10_popularity_movies.reset_index(drop = True)
```

Out[38]:

	title	Director	popularity
0	Minions	Kyle Balda	875.581305
1	Interstellar	Christopher Nolan	724.247784
2	Deadpool	Tim Miller	514.569956
3	Guardians of the Galaxy	James Gunn	481.098624
4	Mad Max: Fury Road	George Miller	434.278564
5	Jurassic World	Colin Trevorrow	418.708552
6	Pirates of the Caribbean: The Curse of the Bla...	Gore Verbinski	271.972889
7	Dawn of the Planet of the Apes	Matt Reeves	243.791743
8	The Hunger Games: Mockingjay - Part 1	Francis Lawrence	206.227151
9	Big Hero 6	Chris Williams	203.734590

Top 10 Long length Movies based on Director

In [39]:

```
top10_runtime_movies = df.sort_values(ascending = False , by = 'runtime')[['title' ,  'Dire
top10_runtime_movies.reset_index(drop = True)
```

Out[39]:

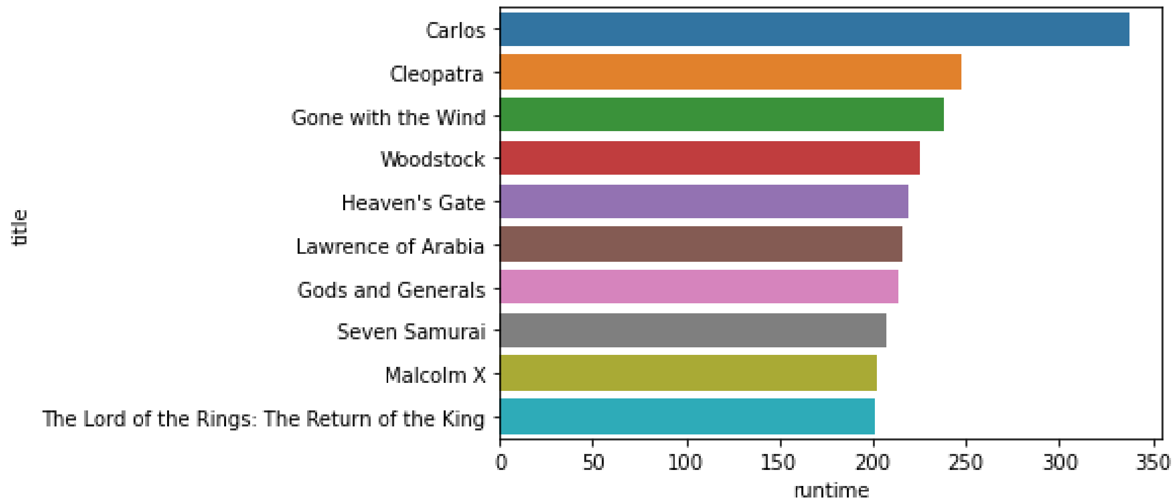
	title	Director	runtime
0	Carlos	Olivier Assayas	338.0
1	Cleopatra	Joseph L. Mankiewicz	248.0
2	Gone with the Wind	Victor Fleming	238.0
3	Woodstock	Michael Wadleigh	225.0
4	Heaven's Gate	Michael Cimino	219.0
5	Lawrence of Arabia	David Lean	216.0
6	Gods and Generals	Ronald F. Maxwell	214.0
7	Seven Samurai	Akira Kurosawa	207.0
8	Malcolm X	Spike Lee	202.0
9	The Lord of the Rings: The Return of the King	Peter Jackson	201.0

In [40]:

```
sns.barplot(x = 'runtime' , y = 'title' ,data = top10_runtime_movies)
```

Out[40]:

```
<AxesSubplot:xlabel='runtime', ylabel='title'>
```



Fearure Engineering

In [41]:

```
#converting object dtype to datetime so that we can extract some datetime features.
df['release_date'] = pd.to_datetime(df['release_date'])
```

this extracting_date_features will extract year , month , weekday columns from release_date columns

In [42]:

```
l = ['year' , 'month' , 'weekday']
def extracting_date_features(df_date):
    for i in l:
        df[i] = getattr(df_date['release_date'].dt , i).astype('int')
    return df_date
```

In [43]:

```
df = extracting_date_features(df)
df.head(2)
```

Out[43]:

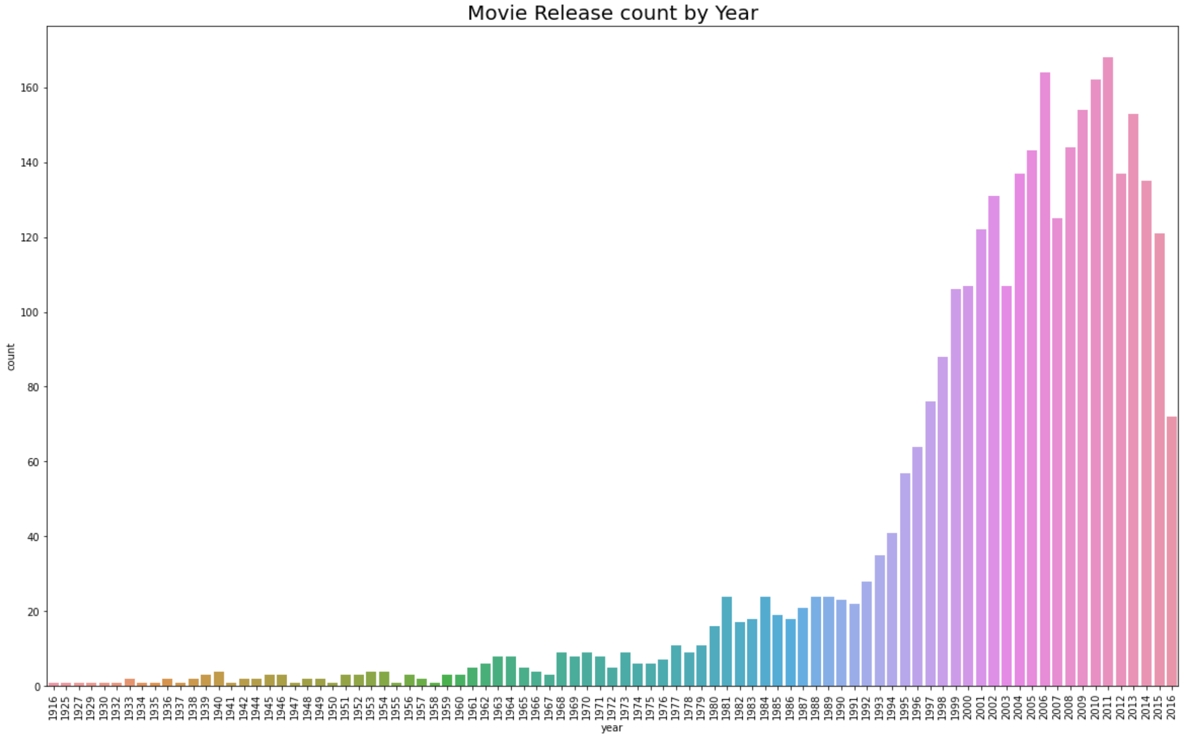
	budget	genres	id	keywords	original_language	original_title	overview	popula
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	19995	[culture clash, future, space war, space colon...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437
1	300000000	[Adventure, Fantasy, Action]	285	[ocean, drug abuse, exotic island, east india ...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082

2 rows × 25 columns

Movies released count per year

In [44]:

```
plt.figure(figsize = (20,12))
sns.countplot(x = 'year' , data = df )
plt.title("Movie Release count by Year",fontsize=20)
plt.xticks(rotation = 'vertical')
plt.show()
```



Extracting decades from year columns

In [45]:

```
def decade(x):
    if x>=1960 and x<=1969:
        return '60s'
    elif x>=1970 and x<=1979:
        return '70s'
    elif x>=1980 and x<=1989:
        return '80s'
    elif x>=1990 and x<=1999:
        return '90s'
    elif x>1999:
        return '21s'
    else:
        return 'movie between 1916 to 1960'
```

In [46]:

```
df['decade'] = df['year'].apply(decade)
```

In [47]:

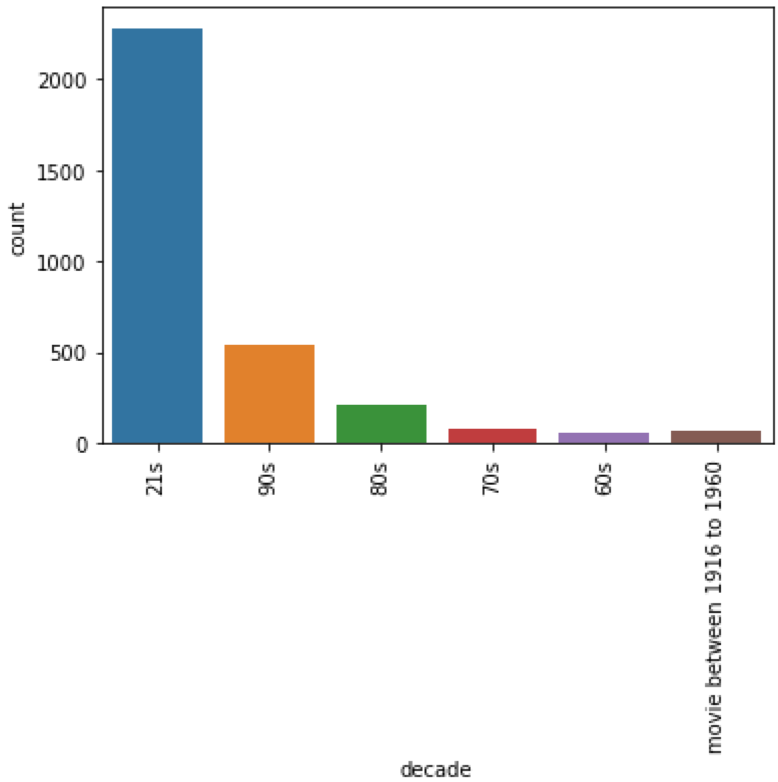
```
df['decade'].value_counts()
```

Out[47]:

```
21s                          2282
90s                           540
80s                           205
70s                            81
movie between 1916 to 1960     63
60s                            59
Name: decade, dtype: int64
```

In [48]:

```
sns.countplot(df['decade'])
plt.xticks(rotation = 'vertical')
plt.show()
```



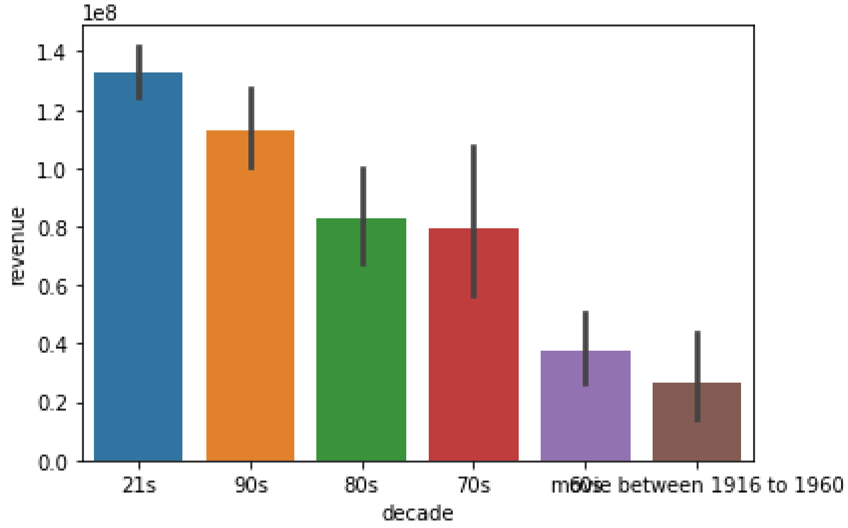
From the above chart movie released in 21s generated more revenue compared to old movies.

In [49]:

```
sns.barplot(x = 'decade' , y = 'revenue' , data = df )
```

Out[49]:

```
<AxesSubplot:xlabel='decade', ylabel='revenue'>
```



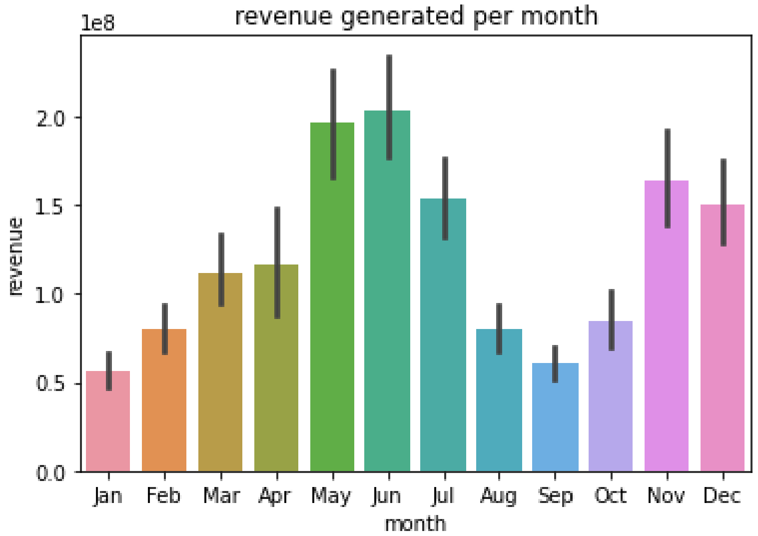
From this bar graph we can see that 21s century movie is generating more revenue compare to 70s , 80s and older movies.

In [50]:

```
sns.barplot(x = 'month' , y = 'revenue' , data = df )
#lets replace number by actual month name
loc , labels = plt.xticks()
loc, labels = loc, ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "
plt.xticks(loc, labels,fontsize=10)
plt.title('revenue generated per month')
```

Out[50]:

```
Text(0.5, 1.0, 'revenue generated per month')
```



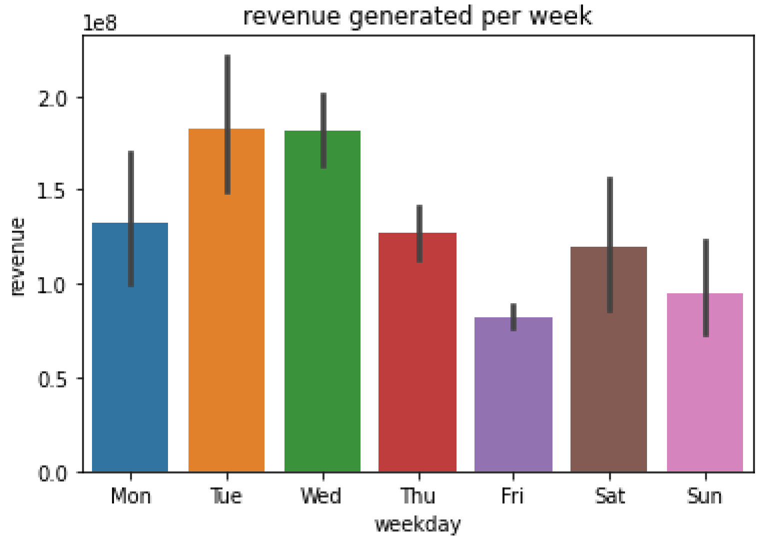
From the above bar graph we can see that movie released in jun month generating more revenue compared to other month

In [51]:

```
# plt.figure(figsize=(20,5));
sns.barplot(x = 'weekday' , y = 'revenue' , data = df)
loc , labels = plt.xticks()
loc , labels = loc ,['Mon' , 'Tue' , 'Wed' , 'Thu' , 'Fri' , 'Sat' , 'Sun']
plt.xticks(loc , labels,fontsize=10)
plt.title('revenue generated per week')
```

Out[51]:

```
Text(0.5, 1.0, 'revenue generated per week')
```



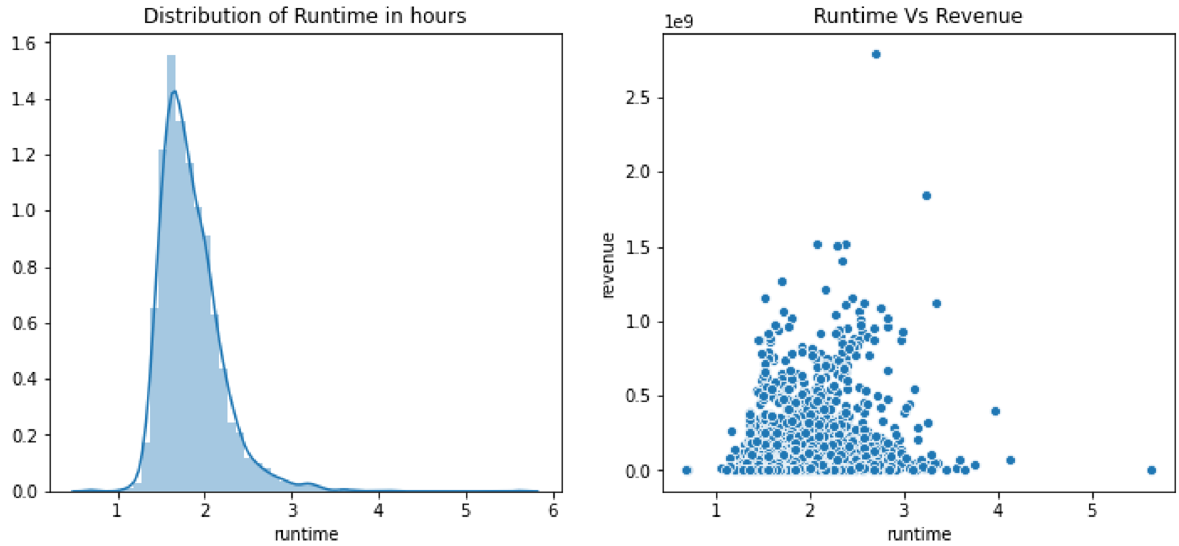
Revenue Vs runtime

In [52]:

```
plt.figure(figsize = (12,5))
plt.subplot(1,2,1)
plt.title('Distribution of Runtime in hours')
sns.distplot((df['runtime'] / 60 ) , kde = True )
plt.subplot(1,2,2)
plt.title('Runtime Vs Revenue')
sns.scatterplot(x = df['runtime']/60  , y = 'revenue' , data = df)
```

Out[52]:

```
<AxesSubplot:title={'center':'Runtime Vs Revenue'}, xlabel='runtime', ylabel
='revenue'>
```



from this distribution we can see that most of the movies runtime are 2hrs.

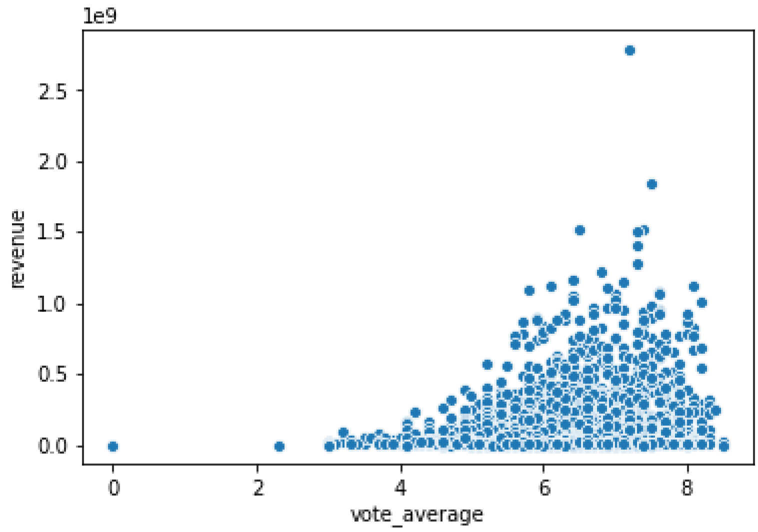
Rating Vs revenue

In [53]:

```
sns.scatterplot(x ='vote_average' , y = 'revenue', data = df)
```

Out[53]:

<AxesSubplot:xlabel='vote_average', ylabel='revenue'>



Most of the movies are rated between 6 to 8 . movies which are rated above 6 genrating more revenue compare to movies which are rated below 6.

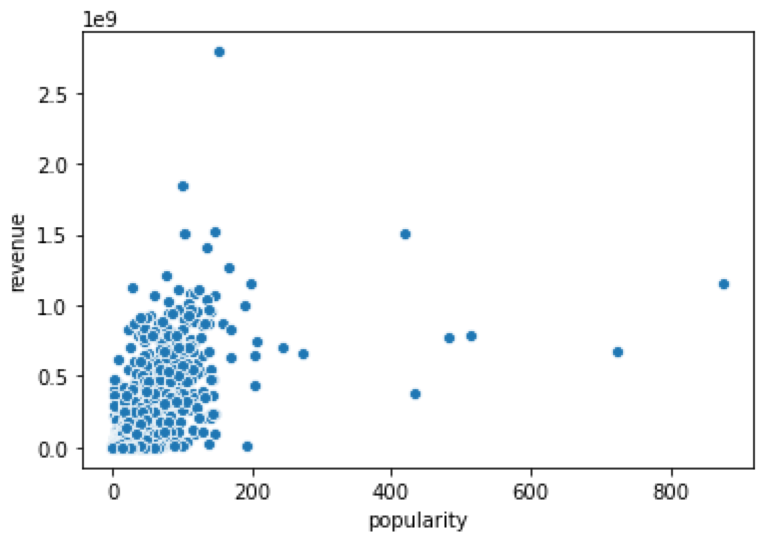
Relationship Popularity Vs Revenue

In [54]:

```
sns.scatterplot(x ='popularity' , y = 'revenue', data = df)
```

Out[54]:

<AxesSubplot:xlabel='popularity', ylabel='revenue'>



the above graph is showing relationship between popularity and revenue.m

Selectig only top 8 languages.

In [55]:

```
a = df['original_language'].value_counts()[0:8]
print(a)
ln_list = a.index.tolist()
print(ln_list)
#selecting index of top 8 languages
```

```
en    3102
fr      25
es      15
ja      13
zh      13
de       9
hi       7
ru       6
Name: original_language, dtype: int64
['en', 'fr', 'es', 'ja', 'zh', 'de', 'hi', 'ru']
```

Most of movies original language are english.

Bollywood movies List

In [56]:

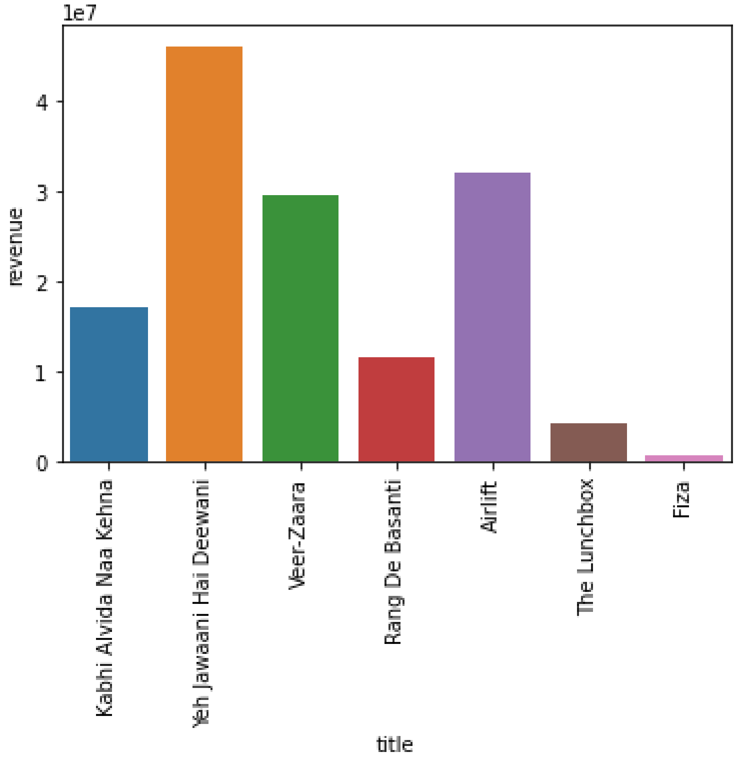
```
bolly_df = df[df['original_language'] == 'hi'][['budget' , 'title' , 'Director' , 'revenue'
bolly_df
```

Out[56]:

	budget	title	Director	revenue
2967	7400000	Kabhi Alvida Naa Kehna	Karan Johar	17000000
3233	7700000	Yeh Jawaani Hai Deewani	Ayan Mukherjee	46000000
3379	7000000	Veer-Zaara	Yash Chopra	29385320
3548	2200000	Rang De Basanti	Rakeysh Omprakash Mehra	11502151
3729	4500000	Airlift	Raja Menon	32000000
4205	1000000	The Lunchbox	Ritesh Batra	4235151
4377	1000000	Fiza	Khalid Mohammed	623791

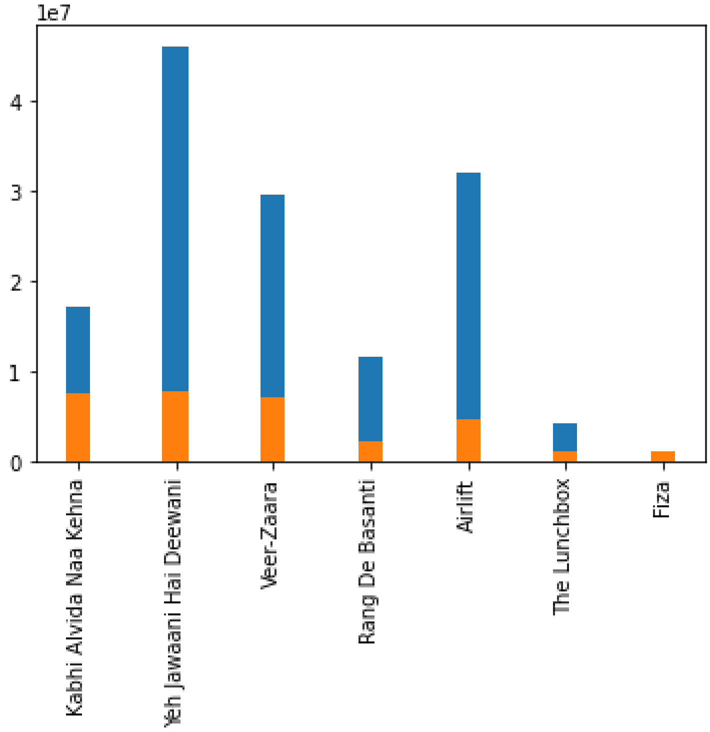
In [57]:

```
sns.barplot(x = 'title' , y = 'revenue' , data = bolly_df)
plt.xticks(rotation = 'vertical')
plt.show()
```



In [58]:

```
plt.bar(bolly_df['title'] , bolly_df['revenue'], width = 0.25)
plt.bar(bolly_df['title'] , bolly_df['budget'] , width = 0.25)
plt.xticks(rotation = 'vertical')
plt.show()
```



above graph showing profit of movie.

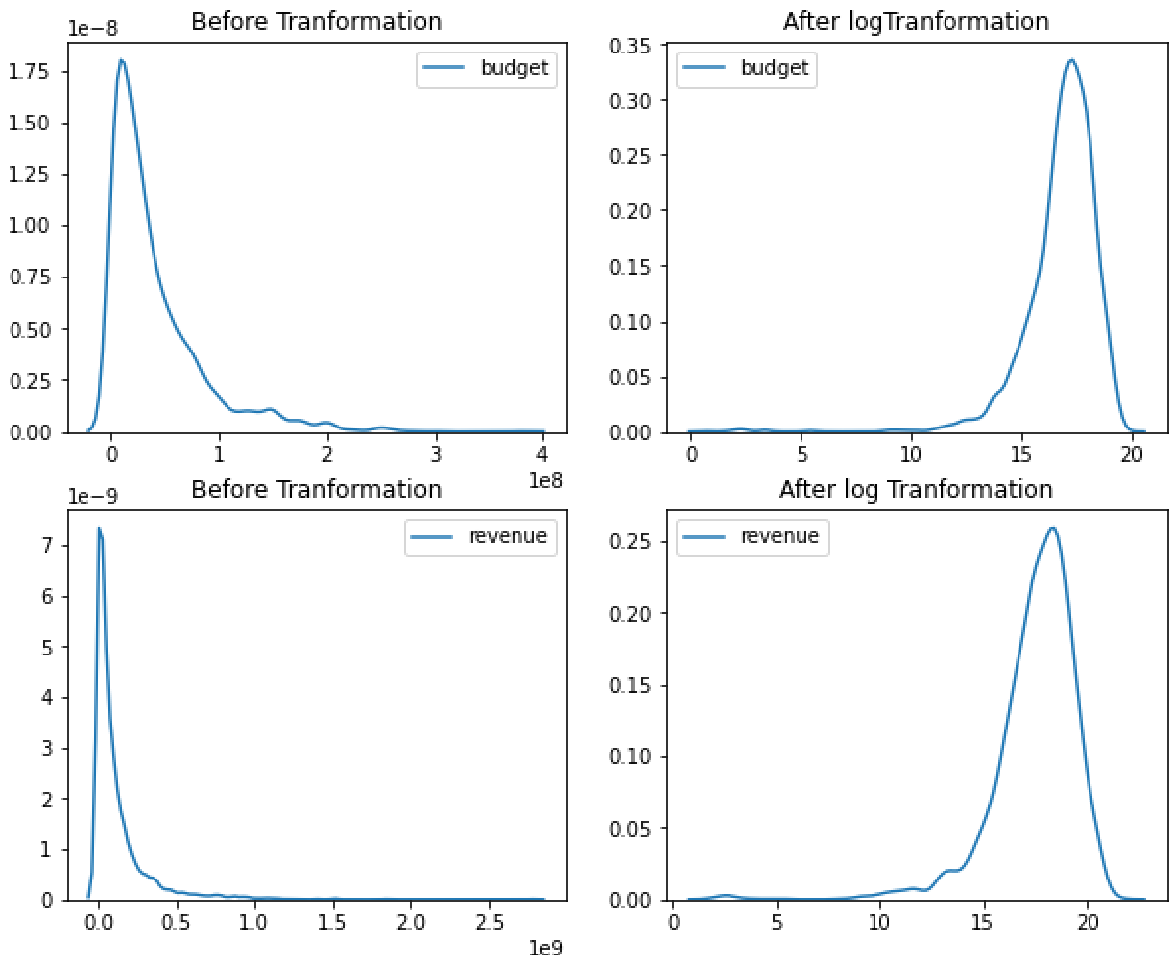
Distribution of budget and revenue before and after log transformation

In [59]:

```
fig , figpos  = plt.subplots(2 , 2 , figsize = (10 , 8)) #2 rows 2 cols suplots ,
#figpos basically tell about where we want to plot a particular graph in in 2x2 matric
sns.kdeplot(df['budget'] , ax = figpos[0][0]  )
figpos[0][0].set_title('Before Tranformation')
sns.kdeplot(np.log1p(df['budget']) , ax = figpos[0][1])
#we are not using log0 to avoid & and null value as there might be 0 value
figpos[0][1].set_title('After logTranformation')
sns.kdeplot(df['revenue'] , ax = figpos[1][0])
figpos[1][0].set_title('Before Tranformation')
sns.kdeplot(np.log1p(df['revenue']) , ax = figpos[1][1])
figpos[1][1].set_title('After log Tranformation')
```

Out[59]:

```
Text(0.5, 1.0, 'After log Tranformation')
```



We can see that this data is very skewed and therefore it is difficult to draw conclusion from this graph.we knew to normalise this data.

Introducing log

Why skewed data is not good fit for modeling in Linear Regression ?

1.Because they may act as an outlier ,and we know that outlier is not good for our model performance. 2.To linearize the fit as much as possible. Statistical test are usually based on the assumption of normality(normal distribution).

Log Transformation is popular method for handling skew data.

we can see that before transformation data was very skewed but after transformation its now nomalized.

Outliers

In [60]:

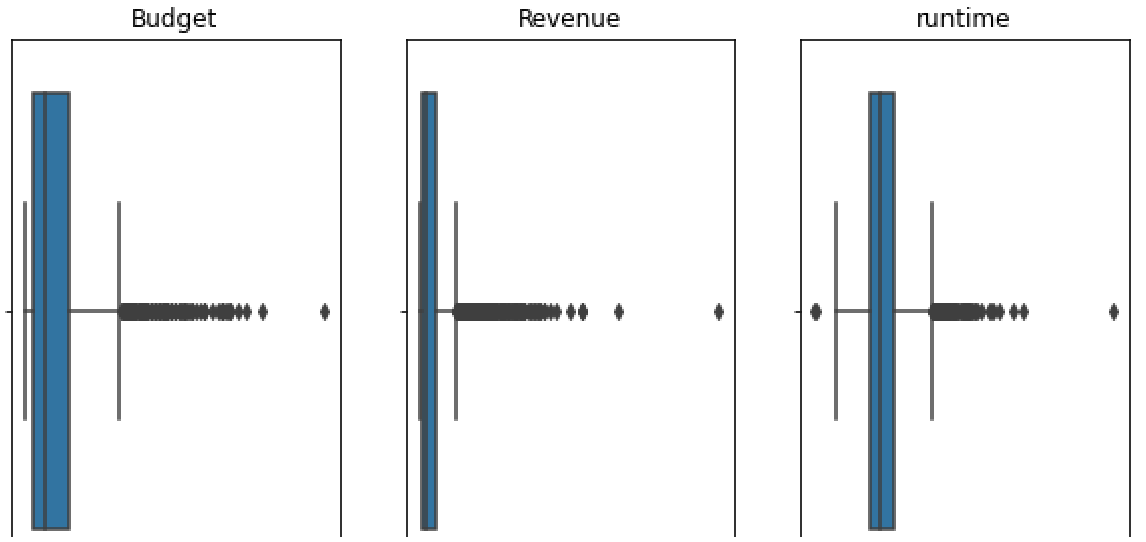
```
plt.figure(figsize = (10,5))
plt.subplot(1 ,3,1)
plt.title('Budget')
sns.boxplot(df['budget'])

plt.subplot(1,3,2)
plt.title('Revenue')
sns.boxplot(df['revenue'])

plt.subplot(1,3,3)
plt.title('runtime')
sns.boxplot(df['runtime'])
```

Out[60]:

```
<AxesSubplot:title={'center':'runtime'}, xlabel='runtime'>
```



In [61]:

```
# Finding the IQR For Budget columns
dict = {}
for col in ['budget' , 'revenue' , 'runtime']:
    percentile25 = df[col].quantile(0.25)
    percentile75 = df[col].quantile(0.75)
    IQR  = percentile75 - percentile25
    upper_limit = percentile75 + 1.5 * IQR
    lower_limit = percentile25 - 1.5 * IQR
    dict['upper_limit'+ '_' + col] = upper_limit
    dict['lower_limit'+ '_' + col] = lower_limit
```

In Above code cell i just created a dictionary to keep upper_limit and lower_limit of budget , revenue , runtime.

In [62]:

```
dict
```

Out[62]:

```
{'upper_limit_budget': 121750000.0,
 'lower_limit_budget': -56250000.0,
 'upper_limit_revenue': 340489618.5,
 'lower_limit_revenue': -176959149.5,
 'upper_limit_runtime': 158.5,
 'lower_limit_runtime': 58.5}
```

This scatter plot shows there is relationship exists between budget and revenue.

In [63]:

```
for col in ['budget' , 'revenue' , 'runtime']:
    print('There are total {} movies data which {} are less than lower limit.'.format(len(d
    print('There are total {} movies data which {} are more than upper limit.'.format(len(d
```

```
There are total 0 movies data which budget are less than lower limit.
There are total 216 movies data which budget are more than upper limit.
There are total 0 movies data which revenue are less than lower limit.
There are total 285 movies data which revenue are more than upper limit.
There are total 2 movies data which runtime are less than lower limit.
There are total 97 movies data which runtime are more than upper limit.
```

In [64]:

```
len(df[df['budget'] > dict['upper_limit_budget']])
```

Out[64]:

```
216
```

dict['upper_limit_' + col] == df['upper_limit_budget']

Capping Budget and Revenue with upper limit and lower limit.

np.where(condtion,true,false)

In [65]:

```
for col in ['budget' , 'revenue' , 'runtime']:
    df[col] = np.where(
        df[col] > dict['upper_limit_' + col],
        dict['upper_limit_' + col],
        np.where(
            df[col] < dict['lower_limit_' + col],
            dict['lower_limit_' + col],
            df[col]
        )
    )
```

In [66]:

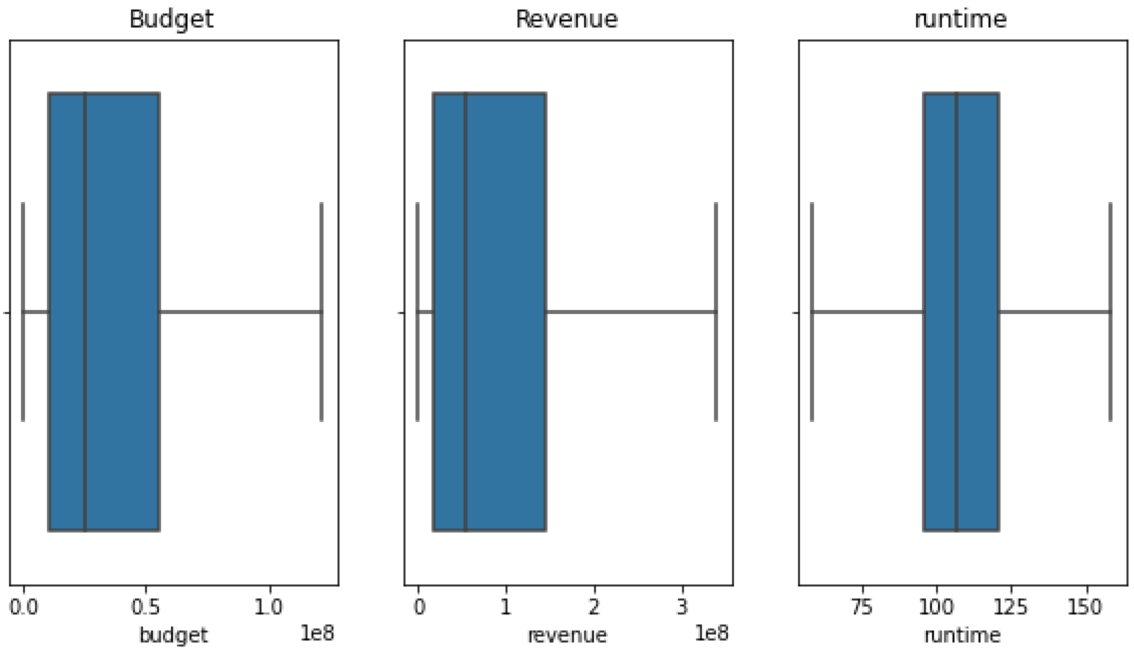
```
plt.figure(figsize = (10,5))
plt.subplot(1 ,3,1)
plt.title('Budget')
sns.boxplot(df['budget'])

plt.subplot(1,3,2)
plt.title('Revenue')
sns.boxplot(df['revenue'])

plt.subplot(1,3,3)
plt.title('runtime')
sns.boxplot(df['runtime'])
```

Out[66]:

```
<AxesSubplot:title={'center':'runtime'}, xlabel='runtime'>
```



Now you can see outliers are removed.

In [67]:

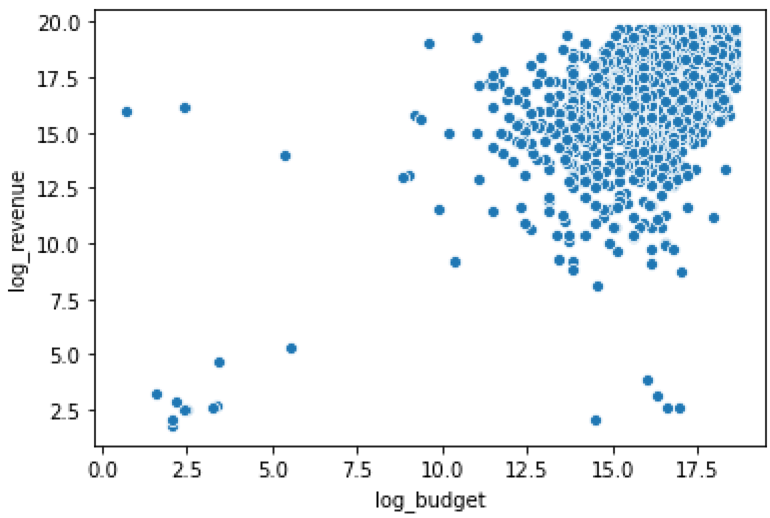
```
df['log_budget'] = np.log1p(df['budget'])
df['log_revenue'] = np.log1p(df['revenue'])
```

In [68]:

```
sns.scatterplot(x ='log_budget'  , y = 'log_revenue' , data = df)
```

Out[68]:

```
<AxesSubplot:xlabel='log_budget', ylabel='log_revenue'>
```

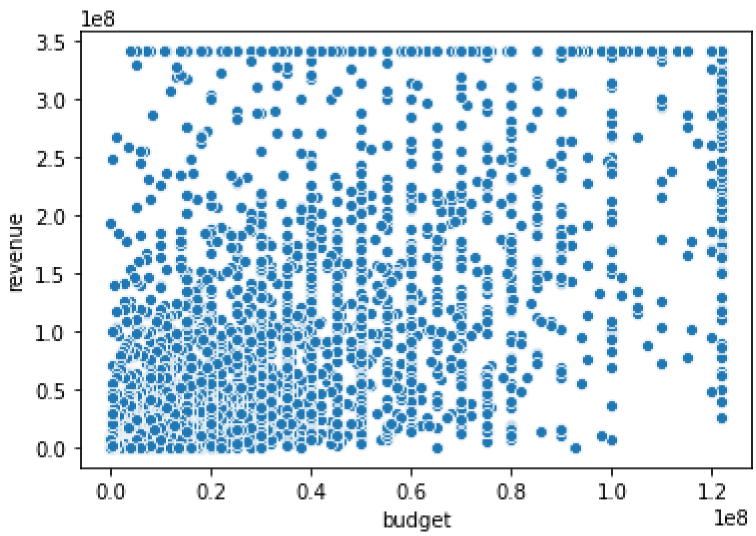


In [69]:

```
sns.scatterplot(x ='budget'  , y = 'revenue' , data = df   )
```

Out[69]:

```
<AxesSubplot:xlabel='budget', ylabel='revenue'>
```



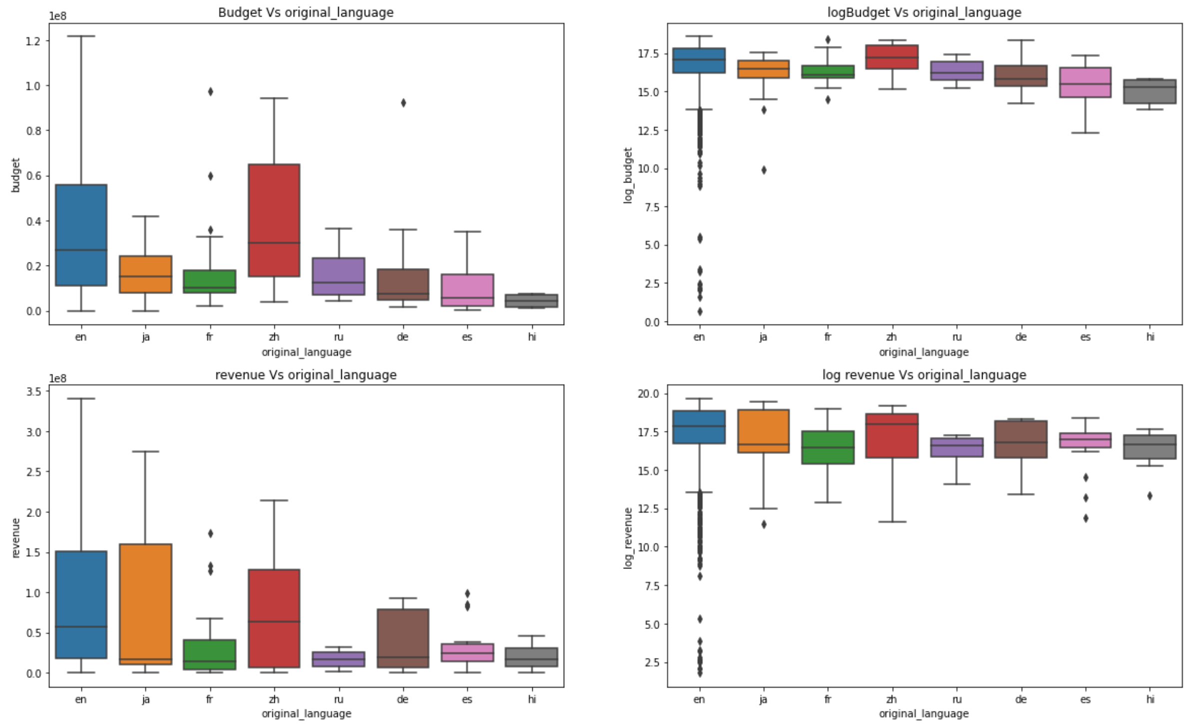
Budget , revenue with original_language

In [70]:

```
fig , figpos = plt.subplots(2 , 2 , figsize = (20 , 12))
sns.boxplot(x = 'original_language' , y = 'budget' , data = df[df['original_language'].isin
figpos[0][0].set_title('Budget Vs original_language')
sns.boxplot(x = 'original_language' , y = 'log_budget' , data = df[df['original_language'].
figpos[0][1].set_title('logBudget Vs original_language')
sns.boxplot(x = 'original_language' , y = 'revenue' , data = df[df['original_language'].isi
figpos[1][0].set_title('revenue Vs original_language')
sns.boxplot(x = 'original_language' , y = 'log_revenue' , data = df[df['original_language']
figpos[1][1].set_title('log revenue Vs original_language')
```

Out[70]:

```
Text(0.5, 1.0, 'log revenue Vs original_language')
```



- From the fig_1 we can see that x-axis indicated langaue plotted. We can see that english language has higher revenue by far margin compared to toher lagnguage. This graph also says us that english language over shadowed all other language in terms of revenue. This information may be quite incorrect and mis leading. Lets see fig_2 for more details
- From the fig_2 : We can see that orginal langauge vs log transformation of revenue and we can see that other language are also creating revenue near english language . How ever it's english language movie that is leading.

In [71]:

```
genres_df = df['genres'].apply(pd.Series)
genres_df.head()
```

Out[71]:

	0	1	2	3	4	5	6
0	Action	Adventure	Fantasy	Science Fiction	NaN	NaN	NaN
1	Adventure	Fantasy	Action	NaN	NaN	NaN	NaN
2	Action	Adventure	Crime	NaN	NaN	NaN	NaN
3	Action	Crime	Drama	Thriller	NaN	NaN	NaN
4	Action	Adventure	Science Fiction	NaN	NaN	NaN	NaN

So now, we have N number of columns (number of distinct genres) for each movie, and non null columns represent each of the movies' genres.

In [72]:

```
stacked_genres = genres_df.stack()
stacked_genres.head(10)
```

Out[72]:

```
0  0            Action
   1         Adventure
   2           Fantasy
   3    Science Fiction
1  0         Adventure
   1           Fantasy
   2            Action
2  0            Action
   1         Adventure
   2             Crime
dtype: object
```

So we converted our cleaned list `[Action, Adventure    Fantasy ,Science Fiction]` to 4 rows, each represents only 1 particular genre. We are now ready to convert this to dummies!

In [73]:

```
raw_dummies = pd.get_dummies(stacked_genres)
raw_dummies.head()
```

Out[73]:

		Action	Adventure	Animation	Comedy	Crime	Documentary	Drama	Family	Fantasy	F
0	**0**	1	0	0	0	0	0	0	0	0	
	1	0	1	0	0	0	0	0	0	0	
	2	0	0	0	0	0	0	0	0	1	
	3	0	0	0	0	0	0	0	0	0	
1	**0**	0	1	0	0	0	0	0	0	0	

In [74]:

```
raw_dummies[['Action' , 'Adventure'   , 'Fantasy' ,'Science Fiction' , 'Comedy']].head(10)
```

Out[74]:

		Action	Adventure	Fantasy	Science Fiction	Comedy
0	**0**	1	0	0	0	0
	1	0	1	0	0	0
	2	0	0	1	0	0
	3	0	0	0	1	0
1	**0**	0	1	0	0	0
	1	0	0	1	0	0
	2	1	0	0	0	0
2	**0**	1	0	0	0	0
	1	0	1	0	0	0
	2	0	0	0	0	0

Here's how the new DataFrame looks like, for simplicity I only selected a few columns. But in actual DataFrame, we have all the genres.

And if you focus on the first 4 rows, you see that Action, Adventure Fantasy ,Science Fiction columns are 1, and the rest are 0.

we need these dummy columns but we only need one row per movie. So how do we do that?

We just aggregate the newly created DataFrame by summing on index level.

In [75]:

```
genre_dummies = raw_dummies.sum(level=0)
genre_dummies.head(10)
```

Out[75]:

	Action	Adventure	Animation	Comedy	Crime	Documentary	Drama	Family	Fantasy	Fore
0	1	1	0	0	0	0	0	0	1	
1	1	1	0	0	0	0	0	0	1	
2	1	1	0	0	1	0	0	0	0	
3	1	0	0	0	1	0	1	0	0	
4	1	1	0	0	0	0	0	0	0	
5	1	1	0	0	0	0	0	0	1	
6	0	0	1	0	0	0	0	1	0	
7	1	1	0	0	0	0	0	0	0	
8	0	1	0	0	0	0	0	1	1	
9	1	1	0	0	0	0	0	0	1	

In [76]:

```
unique_genres = genre_dummies.columns.tolist()
print('Number of unique genres: {} \nGenres: {}'.format(len(unique_genres) ,unique_genres )
```

```
Number of unique genres: 19
Genres: ['Action', 'Adventure', 'Animation', 'Comedy', 'Crime', 'Documentar
y', 'Drama', 'Family', 'Fantasy', 'Foreign', 'History', 'Horror', 'Music',
'Mystery', 'Romance', 'Science Fiction', 'Thriller', 'War', 'Western']
```

In [77]:

```
popular_genres = genre_dummies.sum().sort_values(ascending = False).reset_index()
popular_genres.columns = ['genres' , 'count']
popular_genres
px.bar(data_frame = popular_genres , x = 'count' ,  y = 'genres'  , color = 'genres' )
```

We can see that , from above bar graph Drama is most popular and foreign is least popular.

Top 10 Production Companies

In [78]:

```
prod_compny_df = pd.get_dummies(df['production_companies'].apply(pd.Series)[[0,1,2]].stack(
prod_compny_df.columns = ['production_companies' , 'count']
prod_compny_df
```

Out[78]:

	production_companies	count
0	Universal Pictures	273
1	Paramount Pictures	245
2	Warner Bros.	223
3	Twentieth Century Fox Film Corporation	199
4	Columbia Pictures	167
5	New Line Cinema	142
6	Walt Disney Pictures	95
7	Columbia Pictures Corporation	86
8	Touchstone Pictures	75
9	Village Roadshow Pictures	73

In [79]:

```
px.bar(data_frame = prod_compny_df , x = 'count' , y ='production_companies' , color = 'pro
```

Universal Pictures and Paramount Pictures these two are big prodution companies.

Top 10 Production Countries

In [80]:

```
prod_countries_df = pd.get_dummies(df['production_countries'].apply(pd.Series)[[0,1,2]].sta
prod_countries_df.columns = ['production_countries' , 'count']
prod_countries_df
```

Out[80]:

	production_countries	count
0	United States of America	2857
1	United Kingdom	415
2	Germany	229
3	France	189
4	Canada	159
5	Australia	78
6	Spain	41
7	Japan	38
8	China	37
9	Italy	35

In [81]:

```
px.bar(data_frame = prod_countries_df , x = 'count' , y ='production_countries' , color = '
```

Most of the movies produced in United States of America

Model 1

Content Based Movie Recommender System

In [82]:

```
def remove_white_spaces(x):
    l= []
    for i in x:
        l.append(i.replace(" ",""))
    return l
```

Working of remove_white_spaces function:

remove_white_spaces([' Action' , ' Rakesh' , ' Kumar']) -->> return ['Action' , 'Rakesh' , 'Kumar']

In [83]:

```
df['cast'] = df['cast'].apply(remove_white_spaces)
df['Director'] = df['Director'].str.split(',').apply(remove_white_spaces)
df['genres'] = df['genres'].apply(remove_white_spaces)
df['keywords'] = df['keywords'].apply(remove_white_spaces)
```

In [84]:

```
df['overview'] = df['overview'].apply(lambda x:x.split())
```

In [85]:

```
df['info'] = df['overview'] + df['keywords'] + df['genres'] + df['cast'] + df['Director']
```

In [86]:

```
df['info'] = df['info'].apply(lambda x: " ".join(x))
```

In [87]:

```
df['info'][0:2]
```

Out[87]:

```
0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
Name: info, dtype: object
```

For movie recommendation system i will only select title , movie_id and info columns

In [88]:

```
movie_recommend_df = df[['movie_id' , 'title' , 'info']]
movie_recommend_df.head()
```

Out[88]:

	movie_id	title	info
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...
2	206647	Spectre	A cryptic message from Bond’s past sends him o...
3	49026	The Dark Knight Rises	Following the death of District Attorney Harve...
4	49529	John Carter	John Carter is a war-weary, former military ca...

In [89]:

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=5000,stop_words='english')
```

In [90]:

```
vector = cv.fit_transform(movie_recommend_df['info']).toarray()
```

In [91]:

```
vector.shape
```

Out[91]:

```
(3230, 5000)
```

In [92]:

```
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vector)
```

In [93]:

```
def recommend(movie):
    index = movie_recommend_df[movie_recommend_df['title'] == movie].index[0]
    distances = sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x: x[1]
    for i in distances[1:6]:
        print(movie_recommend_df.iloc[i[0]].title)
```

In [94]:

```
recommend('The Dark Knight Rises')
```

```
The Dark Knight
Batman
Batman Begins
Batman Returns
Batman
```

In [95]:

```
recommend('Spider-Man 3')
```

```
Spider-Man 2
Spider-Man
The Amazing Spider-Man 2
The Amazing Spider-Man
Arachnophobia
```

In [96]:

```
recommend("Pirates of the Caribbean: At World's End")
```

```
Pirates of the Caribbean: Dead Man's Chest
Pirates of the Caribbean: The Curse of the Black Pearl
Pirates of the Caribbean: On Stranger Tides
20,000 Leagues Under the Sea
Puss in Boots
```

Analsysis of above recommend function

step 1. `index =  movie_recommend_df[movie_recommend_df['title'] == movie_name].index[0]`

first line will give us index of movie title

once we get movie title we will try to find which simalarity scores are close to input movie name.

step 2. `list(enumerate(similarity[index]))` o\p: (0, 0.03382550457458692), (1, 0.07372097807744857), (2, 0.11058146711617285),

this code give us enumerating list which consists index of movie and simalarity score of that movie in unsorted order. but we need sorted list so that we can extract top 5 highest simalarity score

step 3. `sorted(list(enumerate(similarity[744])),reverse=True )`

this line of code will sort our list by index but we want sorted list by simalarity score

O/P of following code:
[(3229, 0.0), (3228, 0.0), (3227, 0.03256448045129181), (3226, 0.03806934938134405), (3225, 0.04662524041201569),

step 4. `sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x: x[1])`

for sorted by simalarity score we will pass a key by using lambda function O/p:

[(744, 1.0000000000000002), (952, 0.23372319715296228), (108, 0.18650096164806276), (1538, 0.17782168978975482), (2242, 0.17376201171422898), (638, 0.16051447078102563),

step 5:distances[1:6]

takiing only top 5 movies. slicing [1:6] because 1st movie is our input movie.

step 6: now we will extract movie title by using movie index

In [97]:

```
import pickle
```

In [98]:

```
pickle.dump(movie_recommend_df,open('movie_list.pkl','wb'))
pickle.dump(similarity,open('similarity.pkl','wb'))
```

Model 2

Movie Revenue Prediction Model

In [99]:

```
new_df = pd.concat([df, genre_dummies],axis=1, sort=False) #merging two data frame
new_df.head(5)
```

Out[99]:

	budget	genres	id	keywords	original_language	original_title	overvie
0	121750000.0	[Action, Adventure, Fantasy, ScienceFiction]	19995	[cultureclash, future, spacewar, spacecolony, ...	en	Avatar	[In, th 22n century,, paraplegi Marin
1	121750000.0	[Adventure, Fantasy, Action]	285	[ocean, drugabuse, exoticisland, eastindiatrad...	en	Pirates of the Caribbean: At World's End	[Captai Barbossa lon believe to, be, d
2	121750000.0	[Action, Adventure, Crime]	206647	[spy, basedonnovel, secretagent, sequel, mi6, ...	en	Spectre	[A, crypti messag fror Bond' pas send
3	121750000.0	[Action, Crime, Drama, Thriller]	49026	[dccomics, crimefighter, terrorist, secretiden...	en	The Dark Knight Rises	[Followin the, deat of, Distric Attorney
4	121750000.0	[Action, Adventure, ScienceFiction]	49529	[basedonnovel, mars, medallion, spacetravel, p...	en	John Carter	[Joh Carter, i a, wa weary forme mili

5 rows × 48 columns

In [100]:

```
new_df.columns
```

Out[100]:

```
Index(['budget', 'genres', 'id', 'keywords', 'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'title', 'vote_average', 'vote_count',
       'movie_id', 'cast', 'crew', 'Director', 'year', 'month', 'weekday',
       'decade', 'log_budget', 'log_revenue', 'info', 'Action', 'Adventure',
       'Animation', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Family',
       'Fantasy', 'Foreign', 'History', 'Horror', 'Music', 'Mystery',
       'Romance', 'Science Fiction', 'Thriller', 'War', 'Western'],
      dtype='object')
```

In [101]:

```
cols_drop = ['genres', 'id', 'keywords', 'original_language','original_title', 'overview',
'info' , 'production_countries', 'log_budget' , 'release_date','spoken_languages', 'status'
```

In [102]:

```
revenue_df = new_df.drop(cols_drop , axis =1 )
revenue_df.sample(5)
```

Out[102]:

	budget	popularity	revenue	runtime	vote_average	vote_count	year	month	we
3324	7500000.0	0.571663	3347647.0	110.0	6.1	7	2004	11	
1981	17000000.0	7.701041	16980098.0	95.0	5.5	160	2004	1	
1873	25000000.0	21.729434	59192128.0	101.0	5.5	444	2007	9	
2978	12000000.0	5.759545	9622846.0	145.0	6.4	35	2012	6	
3016	10000000.0	11.350382	25605015.0	100.0	6.1	122	1998	6	

5 rows × 29 columns

In [103]:

```
revenue_df.columns
```

Out[103]:

```
Index(['budget', 'popularity', 'revenue', 'runtime', 'vote_average',
       'vote_count', 'year', 'month', 'weekday', 'log_revenue', 'Action',
       'Adventure', 'Animation', 'Comedy', 'Crime', 'Documentary', 'Drama',
       'Family', 'Fantasy', 'Foreign', 'History', 'Horror', 'Music', 'Myster
y',
       'Romance', 'Science Fiction', 'Thriller', 'War', 'Western'],
      dtype='object')
```

In [104]:

```
X = revenue_df.drop(['revenue' , 'log_revenue'] , 1)
y = revenue_df['revenue']
```

In [105]:

```
X.columns
```

Out[105]:

```
Index(['budget', 'popularity', 'runtime', 'vote_average', 'vote_count', 'yea
r',
       'month', 'weekday', 'Action', 'Adventure', 'Animation', 'Comedy',
       'Crime', 'Documentary', 'Drama', 'Family', 'Fantasy', 'Foreign',
       'History', 'Horror', 'Music', 'Mystery', 'Romance', 'Science Fictio
n',
       'Thriller', 'War', 'Western'],
      dtype='object')
```

Train Test Split

In [106]:

```
from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(X , y , test_size = 0.2,  random_sta
```

In [107]:

```
X_train.shape , X_test.shape , y_train.shape , y_test.shape
```

Out[107]:

```
((2584, 27), (646, 27), (2584,), (646,))
```

In [108]:

```
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor,AdaBoostRegres
from sklearn.metrics import r2_score , mean_absolute_error  , mean_squared_error
```

In [109]:

```
RF_model = RandomForestRegressor(random_state =0, n_estimators=500, max_depth=10)
RF_model.fit(X_train, y_train)
```

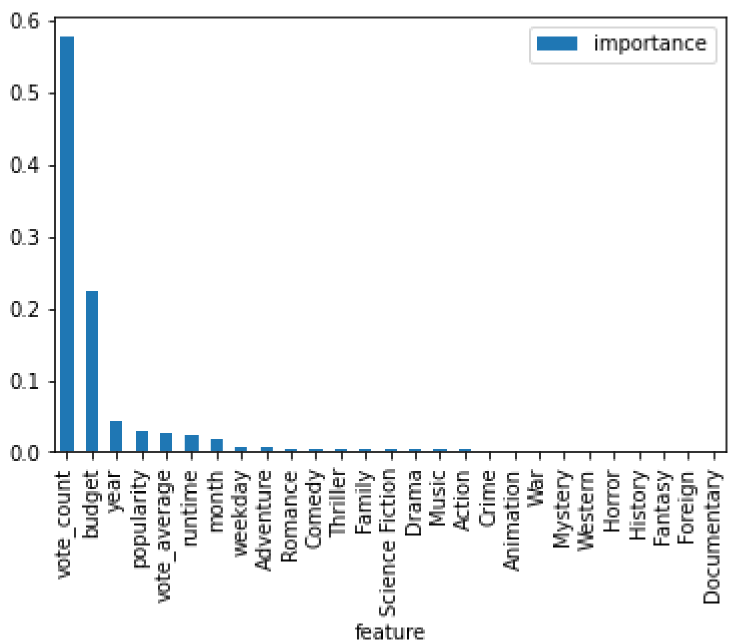
Out[109]:

```
RandomForestRegressor(max_depth=10, n_estimators=500, random_state=0)
```

In [110]:

```
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.round(RF_model.featur
importances = importances.sort_values('importance',ascending=False).set_index('feature');
print(importances)
importances.plot.bar();
```

```
                 importance
feature
vote_count            0.577
budget                0.223
year                  0.044
popularity            0.030
vote_average          0.025
runtime               0.024
month                 0.017
weekday               0.008
Adventure             0.006
Romance               0.005
Comedy                0.005
Thriller              0.005
Family                0.004
Science Fiction       0.004
Drama                 0.004
Music                 0.003
Action                0.003
Crime                 0.002
Animation             0.002
War                   0.001
Mystery               0.001
Western               0.001
Horror                0.001
History               0.001
Fantasy               0.001
Foreign               0.000
Documentary           0.000
```



In [111]:

```
def model_feature(model):
    model.fit(X_train , y_train)
    y_pred = model.predict(X_test)
    print(str(model)[0 : -2] + ' ' 'Model')
    print('r2_score:{}'.format(round(r2_score(y_test , y_pred) , 2)))
    print('MAE',round(mean_absolute_error(y_test , y_pred) , 2))
#     print('MAPE' ,  round(mean_absolute_percentage_error(y_test , y_pred) , 2))
    print('MSE' , round(mean_squared_error(y_test , y_pred) , 2))
```

In [112]:

```
from xgboost import XGBRegressor
import lightgbm as lgb
```

In [113]:

```
model_list = [LinearRegression() ,XGBRegressor() , Ridge() , Lasso() , KNeighborsRegressor(
model_list1 = []
R2_score = []
mae = []
score = []
mse = []

for model in model_list:
    model_list1.append(str(model)[0:-2])
    model.fit(X_train , y_train)
    y_pred = model.predict(X_test)
    R2_score.append(round(r2_score(y_test , y_pred) , 2))
    mae.append(round(mean_absolute_error(y_test , y_pred) , 2))
    mse.append(round(mean_squared_error(y_test , y_pred) , 2))
```

In [114]:

```
dict = {'Model':model_list1, 'R2_score':R2_score , 'MAE':mae , 'MSE':mse}
model_df = pd.DataFrame(dict).sort_values(ascending = False , by = 'R2_score')
model_df
```

Out[114]:

	Model	R2_score	MAE	MSE
1	XGBRegressor(base_score=None, booster=None, co...	0.77	35749553.06	2.756840e+15
10	LGBMRegressor	0.77	36032935.40	2.843900e+15
6	RandomForestRegressor	0.76	36863739.52	2.951209e+15
7	GradientBoostingRegressor	0.75	37648689.33	3.001657e+15
9	ExtraTreesRegressor	0.74	38558031.75	3.132600e+15
8	AdaBoostRegressor	0.67	48797684.25	4.033369e+15
0	LinearRegression	0.65	46785488.18	4.282781e+15
2	Ridge	0.65	46763721.34	4.280193e+15
3	Lasso	0.65	46785485.79	4.282781e+15
4	KNeighborsRegressor	0.63	46365038.40	4.551948e+15
5	DecisionTreeRegressor	0.56	46097310.97	5.345226e+15

In [ ]:

```
```