Final Project Submission

Please fill out:

- Student name: OMONDI IMMANUEL OCHIENG
- Student pace: part time
- Scheduled project review date/time: 16/04/2023
- Instructor name: Noah kandie
- Blog post URL:

ANALYSIS TO FIND OUT POTENTIAL MOVIES FOR MICROSOFT NEWS STUDIO.

BUSINESS UNDERSTANDING

Microsoft had been observing how major companies had been creating movie content and decided to come up with their own movie studio. To do this however, they required an in-depth analysis of which movies had been performing well in the box office.

This notebook will bring all the data sources together on movies for further analysis.

DATA UNDERSTANDING

In order for there to be accurate, easier and proper data analysis, for this project i will make use of database files and some csv files stored on my local computer. My reason for using csv and database file is that they are easier to read and open using inbuilt python libraries. They are also not as cumbersome as json files.

DATA SOURCES

- bom.movie_gross.csv.gz
- rt.movie_info.tsv.gz
- rt.reviews.tsv.gz
- tmdb.movies.csv.gz
- tn.movie_budgets.csv.gz

DATA PREPARATION

The steps i would follow during data gathering and preparation are as follows:

In [48]:

```
# importing of necessary libraries in order to read the data files.
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import style
import warnings
```

In [75]:

```
# importing of pandas will enable me read directly from the csv files using pandas read c
# importing of sqlite3 will enable me establish direct connectivity to the file of intere
# importing of matplotlib will aid in illustrating of my data findings. Matplotlib comes
# i also include matplotlib inline to enable inline plotting on the notebook and make my
```

The next step of my data preparation process would be to open and read the necessary files and assign them to variables.

tmdb.movies.csv.gz

In [471]:

```
tmdb= pd.read_csv('tmdb.movies.csv.gz')
tmdb.head(10)
# This code works by reading files using the pandas attribute read_csv to access the info
# tmdb.head() function then displays the first 8 rows of the dataset.
```

Out[471]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Po H
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	Tra
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iro
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	To
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	In
5	5	[12, 14, 10751]	32657	en	Percy Jackson & the Olympians: The Lightning T...	26.691	2010-02-11	Ja Oly L
6	6	[28, 12, 14, 878]	19995	en	Avatar	26.526	2009-12-18	
7	7	[16, 10751, 35]	10193	en	Toy Story 3	24.445	2010-06-17	Toy
8	8	[16, 10751, 35]	20352	en	Despicable Me	23.673	2010-07-09	Des
9	9	[16, 28, 35, 10751, 878]	38055	en	Megamind	22.855	2010-11-04	Me

In [97]:

```
tmdb.info()
# this code gives us a general overview of the structure of the dataset.
# It is a pandasdataframe with 9 columns by 26517 rows
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         26517 non-null  int64  
 1   genre_ids          26517 non-null  object 
 2   id                 26517 non-null  int64  
 3   original_language  26517 non-null  object 
 4   original_title     26517 non-null  object 
 5   popularity         26517 non-null  float64
 6   release_date       26517 non-null  object 
 7   title              26517 non-null  object 
 8   vote_average       26517 non-null  float64
 9   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 2.0+ MB
```

In [98]:

```
tmdb.columns
#Shows all the columns of the dataset.
```

Out[98]:

```
Index(['Unnamed: 0', 'genre_ids', 'id', 'original_language', 'original_tit
le',
       'popularity', 'release_date', 'title', 'vote_average', 'vote_coun
t'],
      dtype='object')
```

In [99]:

```
tmdb.tail()
#This command gives back the last 5 rows of the dataset
# We can now see the dataset has 26516 rows
```

Out[99]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_
26512	26512	[27, 18]	488143	en	Laboratory Conditions	0.6	2018-
26513	26513	[18, 53]	485975	en	_EXHIBIT_84xxx_	0.6	2018-
26514	26514	[14, 28, 12]	381231	en	The Last One	0.6	2018-
26515	26515	[10751, 12, 28]	366854	en	Trailer Made	0.6	2018-
26516	26516	[53, 27]	309885	en	The Church	0.6	2018-

data cleaning

The next step is to perform some data cleaning. The data_set has some null values in some columns and some rows are displaying unreal values.

In [101]:

```
tmdb.isnull()
# This code identifies null values in the dataset
#lets display for the first top rows
tmdb.isnull()
```

Out[101]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
26512	False	False	False	False	False	False	False
26513	False	False	False	False	False	False	False
26514	False	False	False	False	False	False	False
26515	False	False	False	False	False	False	False
26516	False	False	False	False	False	False	False

26517 rows × 10 columns

In [102]:

```
tmdb.shape
#checks the shape of the dataset, it has 26517 columns by 9 rows.
```

Out[102]:

```
(26517, 10)
```

In [103]:

```
tmdb.dropna(how ='all').shape
#This code checks for null values in all rows, since the shape remains the same, it prove
```

Out[103]:

```
(26517, 10)
```

In [104]:

```
tmdb.dropna(how= 'any').shape
```

Out[104]:

```
(26517, 10)
```

In [105]:

```
# The two codes run above show that the dataset has no null values since the shape of the
```

In [106]:

```
tmdb.isnull().sum()
#Here we sum all the total values in each row, the output diplayed below further shows th
```

Out[106]:

```
Unnamed: 0          0
genre_ids           0
id                  0
original_language   0
original_title      0
popularity          0
release_date        0
title               0
vote_average        0
vote_count          0
dtype: int64
```

In [107]:

```
tmdb= tmdb[tmdb['vote_average'] != 0.0 ]
tmdb.head(10)
# In this code we dropped rows where ('vote_average') was equal to 0 to get rid of empty
```

Out[107]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Po H
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	Tra
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iro
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	To
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	In
5	5	[12, 14, 10751]	32657	en	Percy Jackson & the Olympians: The Lightning T...	26.691	2010-02-11	Ja Oly L
6	6	[28, 12, 14, 878]	19995	en	Avatar	26.526	2009-12-18	
7	7	[16, 10751, 35]	10193	en	Toy Story 3	24.445	2010-06-17	Toy
8	8	[16, 10751, 35]	20352	en	Despicable Me	23.673	2010-07-09	Des
9	9	[16, 28, 35, 10751, 878]	38055	en	Megamind	22.855	2010-11-04	Me

In [108]:

```
tmdb.shape
# this code shows there has been a reduction in number of rows showing they have been dro
```

Out[108]:

```
(26381, 10)
```

In [109]:

```
tmdb= tmdb.loc[tmdb['vote_count'] > 10000]
tmdb.head()
# in this cell we selected the rows whose (VOTE_COUNT) was greater than 10,000 and droppe
```

Out[109]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	I ar De Ha I
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iror
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Ince
6	6	[28, 12, 14, 878]	19995	en	Avatar	26.526	2009-12-18	A

In [110]:

```
tmdb.shape
#judging by this output, we can see definitely some rows were dropped.
```

Out[110]:

```
(72, 10)
```

In [111]:

```
tmdb= tmdb['vote_count'].sort_values(ascending= True)
tmdb.head()
# This code sorts all the values in the VOTE_COUNT column in ascending order
```

Out[111]:

```
11068    10019
20688    10028
17443    10028
8        10057
11022    10062
Name: vote_count, dtype: int64
```

In [112]:

```
 tmdb.describe()
# This code gives a return of mathematical operations performed in the dataset
```

Out[112]:

```
count       72.000000
mean     12347.972222
std       2605.554014
min      10019.000000
25%      10393.750000
50%      11970.000000
75%      12709.250000
max      22186.000000
Name: vote_count, dtype: float64
```

In [214]:

```
tmdb = tmdb.loc[tmdb['vote_average'] > 7.7 ]
tmdb.head()
# in this code we are selecting only the rows with a vote_average greater than 7.7
```

Out[214]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	
19	[18, 53, 9648]	11324	en	Shutter Island	18.060	2010-02-18	Shutter Island	
43	[35, 10749]	239	en	Some Like It Hot	14.200	1959-03-18	Some Like It Hot	
58	[18]	705	en	All About Eve	13.163	2000-10-06	All About Eve	

In [213]:

```
#lets now drop the unwanted column
tmdb.drop(tmdb.columns[[0]], axis= 1)
```

Out[213]:

	id	original_language	original_title	popularity	release_date	title	vote_avera
0	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	
1	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	
2	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	
3	862	en	Toy Story	28.005	1995-11-22	Toy Story	
4	27205	en	Inception	27.920	2010-07-16	Inception	
...	...	...	...	...	...	...	
24268	490	sv	Det sjunde inseglet	8.693	1958-10-13	The Seventh Seal	
24275	301804	en	Before I Wake	8.631	2018-01-05	Before I Wake	
24287	451480	en	The Guernsey Literary & Potato Peel Pie Society	8.402	2018-08-10	The Guernsey Literary & Potato Peel Pie Society	
24309	401371	en	Mute	8.180	2018-02-23	Mute	
24462	503314	ja	ドラゴンボール超スーパー ブロリー	6.868	2019-01-16	Dragon Ball Super: Broly	

1687 rows × 8 columns

In [ ]:

```
# The number of rows has greatly reduced because we have further dropped rows with vote a
```

In [212]:

```
# lets now select the rows containing vote_count > 500
tmdb= tmdb.loc[tmdb['vote_count'] > 500 ]
tmdb.head(8)
```

Out[212]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vot
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	
5	[12, 14, 10751]	32657	en	Percy Jackson & the Olympians: The Lightning T...	26.691	2010-02-11	Percy Jackson & the Olympians: The Lightning T...	
6	[28, 12, 14, 878]	19995	en	Avatar	26.526	2009-12-18	Avatar	
7	[16, 10751, 35]	10193	en	Toy Story 3	24.445	2010-06-17	Toy Story 3	

In [118]:

```
#lets now get rid of the unnamed column
tmdb= pd.read_csv('tmdb.movies.csv.gz', index_col= 0)
```

In [119]:

```
tmdb.head()
```

Out[119]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	

data analysis

Next we will plot a histogram using vote_count columns and popularity columns to compare which movies are most popular.

In [120]:

```
votes= tmdb['vote_average']
votes.head()
```

Out[120]:

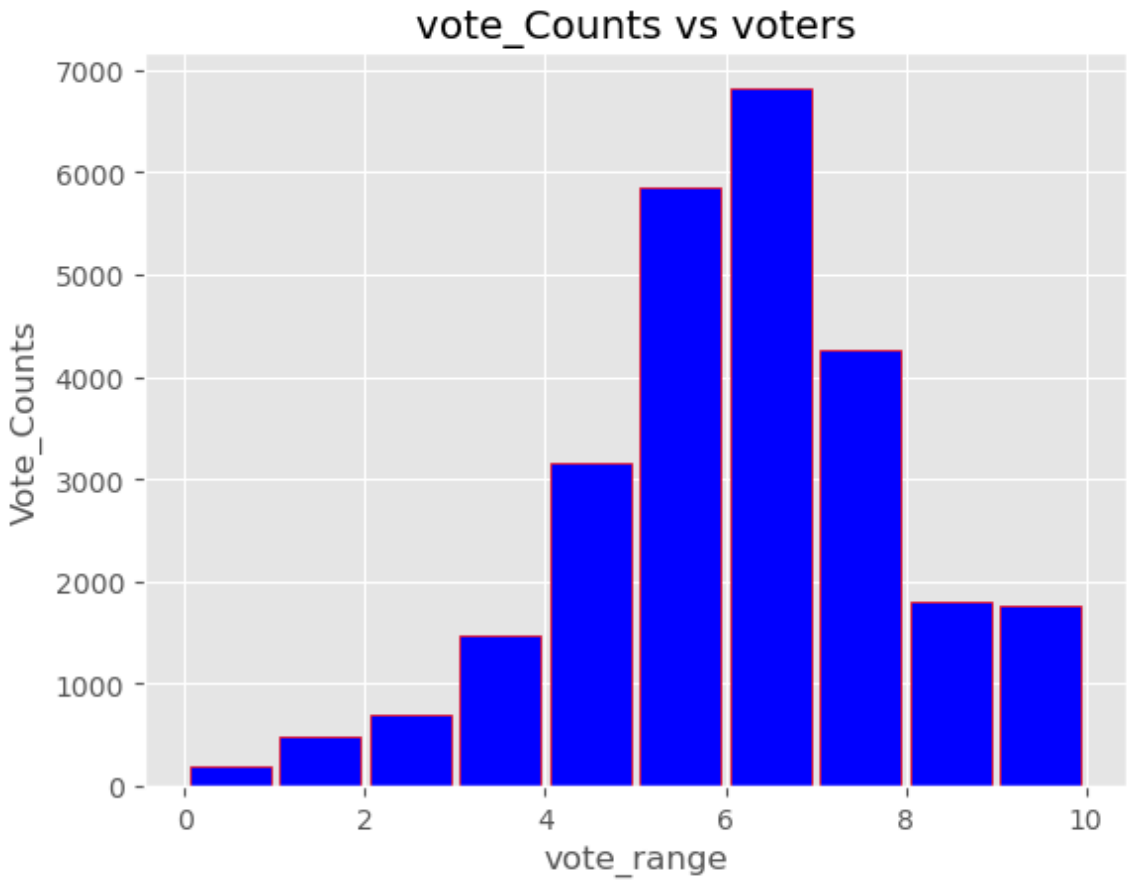
```
0    7.7
1    7.7
2    6.8
3    7.9
4    8.3
Name: vote_average, dtype: float64
```

In [121]:

```
style.use('ggplot')
plt.hist(votes, bins= 10, edgecolor= 'red', color= 'blue', rwidth= 0.9)
plt.title('vote_Counts vs voters')
plt.xlabel('vote_range')
plt.ylabel('Vote_Counts')
```

Out[121]:

```
Text(0, 0.5, 'Vote_Counts')
```



In [122]:

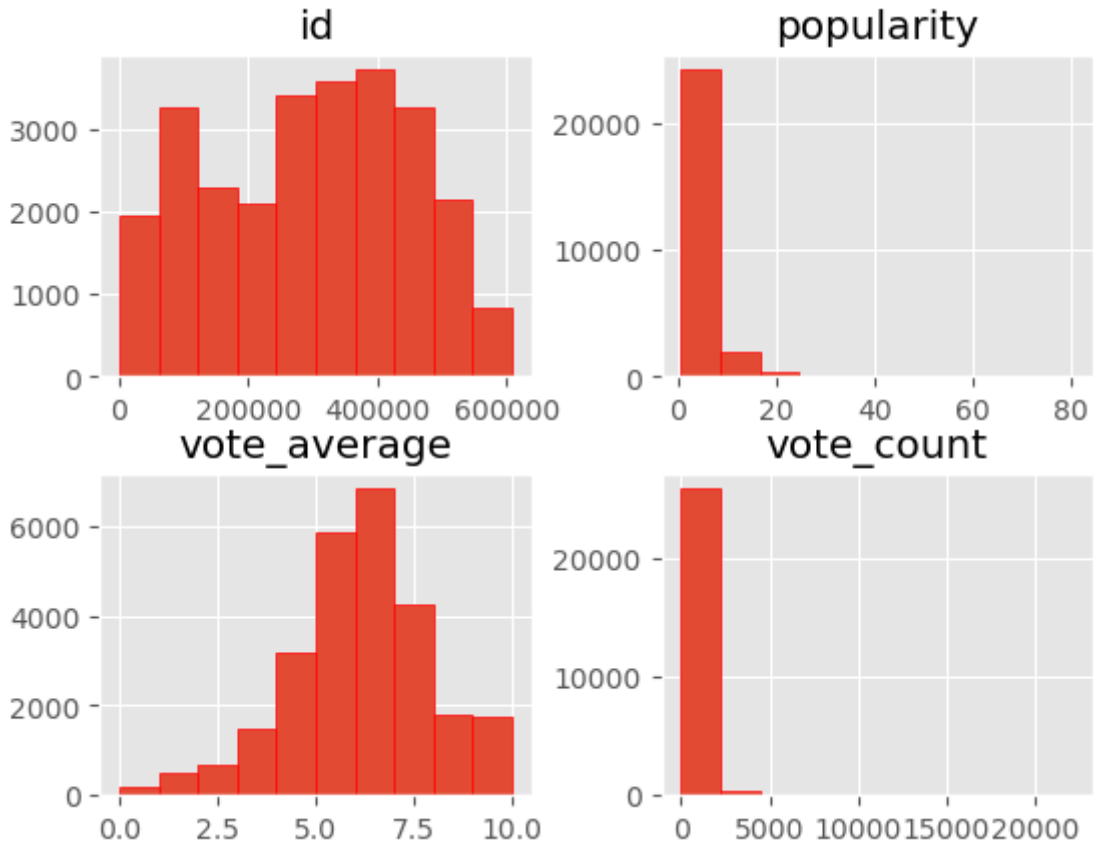
```
# From the above visualization we can see majority of the votes came from a vote range of
```

In [123]:

```
tmdb.hist(bins= 10, edgecolor= 'red')
```

Out[123]:

```
array([[<AxesSubplot:title={'center':'id'}>,
        <AxesSubplot:title={'center':'popularity'}>],
       [<AxesSubplot:title={'center':'vote_average'}>,
        <AxesSubplot:title={'center':'vote_count'}>]], dtype=object)
```



In [124]:

```
#The above visualization shows the distibution of various columns
```

In [125]:

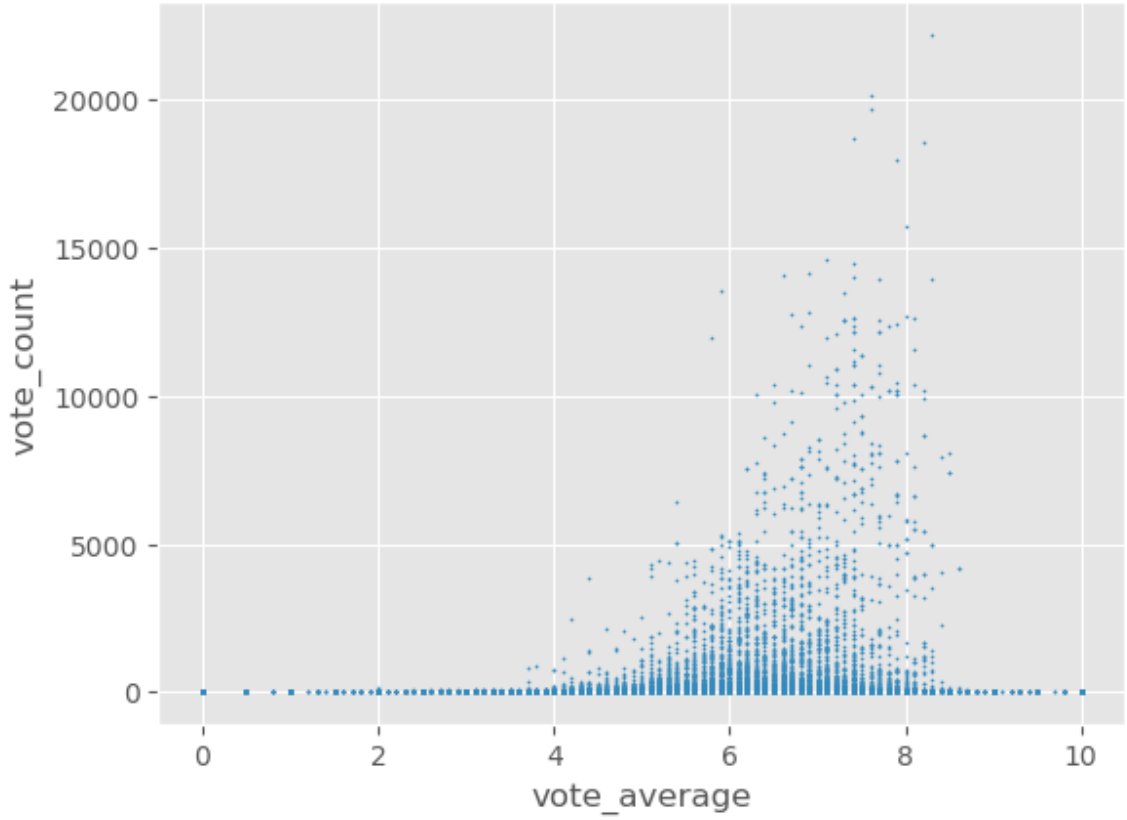
```
tmdb= pd.read_csv('tmdb.movies.csv.gz', index_col= 0)
```

In [126]:

```
tmdb.plot.scatter(x= 'vote_average',
                  y= 'vote_count',
                  s= 0.5)
#
```

Out[126]:

```
<AxesSubplot:xlabel='vote_average', ylabel='vote_count'>
```



In [127]:

```
#From the above scatterplot, majority of the vote counts are below 5000
```

tn.movie_budgets.csv.gz

#lets try and compare movie data for a second dataframe

In [797]:

```
rt= pd.read_csv('tn.movie_budgets.csv.gz', index_col= 0)
rt.head(10)
# Here we are working with another dataset
# We open it using the pd.read function.
```

Out[797]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747
6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	$306,000,000	$936,662,225	$2,053,311,220
7	Apr 27, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,048,134,200
8	May 24, 2007	Pirates of the Caribbean: At Worldâ s End	$300,000,000	$309,420,425	$963,420,425
9	Nov 17, 2017	Justice League	$300,000,000	$229,024,295	$655,945,209
10	Nov 6, 2015	Spectre	$300,000,000	$200,074,175	$879,620,923

In [798]:

```
rt.info()
# Checking the dataframe for general info
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5782 entries, 1 to 82
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       5782 non-null   object
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   object
 3   domestic_gross     5782 non-null   object
 4   worldwide_gross    5782 non-null   object
dtypes: object(5)
memory usage: 271.0+ KB
```

In [799]:

```
rt.columns
#getting the columns from the dataset
```

Out[799]:

```
Index(['release_date', 'movie', 'production_budget', 'domestic_gross',
       'worldwide_gross'],
      dtype='object')
```

In [800]:

```
rt.tail()
# checking last few columns of the dataset
```

Out[800]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
78	Dec 31, 2018	Red 11	$7,000	$0	$0
79	Apr 2, 1999	Following	$6,000	$48,482	$240,495
80	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338
81	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0
82	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041

In [801]:

```
rt.isnull().head()
#checking for null values
```

Out[801]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False

In [802]:

```
rt.describe()
#getting all the numeric counts for the dataset
```

Out[802]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross
count	5782	5782	5782	5782	5782
unique	2418	5698	509	5164	5356
top	Dec 31, 2014	Halloween	$20,000,000	$0	$0
freq	24	3	231	548	367

data cleaning

In [803]:

```
#lets start by converting the columns with dollar signs to integers
rt['production_budget']= rt['production_budget'].str.replace('$','').str.replace(',','')
rt['production_budget']= pd.to_numeric(rt['production_budget'])
```

```
C:\Users\Emmanuel Omondi\AppData\Local\Temp\ipykernel_15820\1058954517.py:
2: FutureWarning: The default value of regex will change from True to Fals
e in a future version. In addition, single character regular expressions w
ill *not* be treated as literal strings when regex=True.
  rt['production_budget']= rt['production_budget'].str.replace('$','').st
r.replace(',','')
```

In [804]:

```
rt['production_budget'].head()
#from this code we can see column(production_budget) does not have dollar signs
```

Out[804]:

```
id
1    425000000
2    410600000
3    350000000
4    330600000
5    317000000
Name: production_budget, dtype: int64
```

In [805]:

```
#lets do the same for the remainig columns
rt['domestic_gross']= rt['domestic_gross'].str.replace('$','').str.replace(',','')
rt['domestic_gross']= pd.to_numeric(rt['domestic_gross'])
```

```
C:\Users\Emmanuel Omondi\AppData\Local\Temp\ipykernel_15820\2198233976.py:
2: FutureWarning: The default value of regex will change from True to Fals
e in a future version. In addition, single character regular expressions w
ill *not* be treated as literal strings when regex=True.
  rt['domestic_gross']= rt['domestic_gross'].str.replace('$','').str.repla
ce(',','')
```

In [806]:

```
rt['domestic_gross'].head()
```

Out[806]:

```
id
1    760507625
2    241063875
3     42762350
4    459005868
5    620181382
Name: domestic_gross, dtype: int64
```

In [807]:

```
rt['worldwide_gross']= rt['worldwide_gross'].str.replace('$','').str.replace(',','')
rt['worldwide_gross']= pd.to_numeric(rt['worldwide_gross'])
```

```
C:\Users\Emmanuel Omondi\AppData\Local\Temp\ipykernel_15820\331787404.py:
1: FutureWarning: The default value of regex will change from True to Fals
e in a future version. In addition, single character regular expressions w
ill *not* be treated as literal strings when regex=True.
  rt['worldwide_gross']= rt['worldwide_gross'].str.replace('$','').str.rep
lace(',','')
```

In [808]:

```
rt['worldwide_gross'].head()
```

Out[808]:

```
id
1    2776345279
2    1045663875
3     149762350
4    1403013963
5    1316721747
Name: worldwide_gross, dtype: int64
```

In [809]:

```
rt.head(8)
#from the output below we can see we got rid of the dollar signs
#we can now proceed to check which values to keep and release
```

Out[809]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350
4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747
6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220
7	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200
8	May 24, 2007	Pirates of the Caribbean: At Worldâ s End	300000000	309420425	963420425

In [810]:

```
#getting rid of rows where production budget is not equal to 0
rt= rt[rt['production_budget'] != 0]
rt.head()
```

Out[810]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350
4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747

In [811]:

```
#getting rid of rows where domestic_gross is not eaqual to 0
rt= rt[rt['domestic_gross']!= 0 ]
rt.head()
```

Out[811]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350
4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747

In [812]:

```
#getting rid of rows where worldwide gross is not equal to 0
rt= rt[rt['worldwide_gross'] != 0]
rt.head()
```

Out[812]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350
4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747

In [813]:

```
rt.isnull().value_counts()
```

Out[813]:

```
release_date  movie  production_budget  domestic_gross  worldwide_gross
False         False  False              False           False
5234
dtype: int64
```

In [814]:

```
# the above code shows there are no null values per column
```

In [815]:

```
#Lets now sort the rows by id
rt= rt.sort_values( by=['id'])
rt.head(5)
```

Out[815]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
1	Jun 13, 1962	Lolita	2000000	9250000	9250000
1	Jul 16, 1999	Lake Placid	27000000	31770413	31770413
1	Sep 19, 1990	Goodfellas	25000000	46743809	46777347
1	Feb 7, 1974	Blazing Saddles	2600000	119500000	119500000

In [816]:

```
# Lets now select only rows where production_budget is greater than/equal to 500000
rt= rt[rt['production_budget'] >= 500000 ]
rt
#Lets do the same for domestic_gross and worldwide_gross columns.
```

Out[816]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
1	Jun 13, 1962	Lolita	2000000	9250000	9250000
1	Jul 16, 1999	Lake Placid	27000000	31770413	31770413
1	Sep 19, 1990	Goodfellas	25000000	46743809	46777347
1	Feb 7, 1974	Blazing Saddles	2600000	119500000	119500000
...	...	...	...	...	...
100	Dec 18, 2009	Nine	80000000	19676965	53508858
100	Oct 19, 2012	Alex Cross	35000000	25888412	35426759
100	Apr 8, 2016	Hardcore Henry	2000000	9252038	17187434
100	Dec 1, 2017	The Disaster Artist	10000000	21120616	28717667
100	Oct 21, 2005	The Work and the Glory: American Zion	6500000	2025032	2025032

5041 rows × 5 columns

In [817]:

```
rt= rt[rt['domestic_gross'] >= 500000]
rt
```

Out[817]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
1	Jun 13, 1962	Lolita	2000000	9250000	9250000
1	Jul 16, 1999	Lake Placid	27000000	31770413	31770413
1	Sep 19, 1990	Goodfellas	25000000	46743809	46777347
1	Feb 7, 1974	Blazing Saddles	2600000	119500000	119500000
...	...	...	...	...	...
100	Dec 18, 2009	Nine	80000000	19676965	53508858
100	Oct 19, 2012	Alex Cross	35000000	25888412	35426759
100	Apr 8, 2016	Hardcore Henry	2000000	9252038	17187434
100	Dec 1, 2017	The Disaster Artist	10000000	21120616	28717667
100	Oct 21, 2005	The Work and the Glory: American Zion	6500000	2025032	2025032

4522 rows × 5 columns

In [818]:

```
rt= rt[rt['worldwide_gross'] >= 500000]
rt.head(8)
```

Out[818]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
1	Jun 13, 1962	Lolita	2000000	9250000	9250000
1	Jul 16, 1999	Lake Placid	27000000	31770413	31770413
1	Sep 19, 1990	Goodfellas	25000000	46743809	46777347
1	Feb 7, 1974	Blazing Saddles	2600000	119500000	119500000
1	Dec 7, 2005	The World's Fastest Indian	25000000	5128124	18991288
1	May 29, 2009	Up	175000000	293004164	731463377
1	Jun 24, 1987	Spaceballs	22700000	38119483	38119483

In [819]:

```
# from the above output we can clearly see the reduction in rows.
```

data visualization

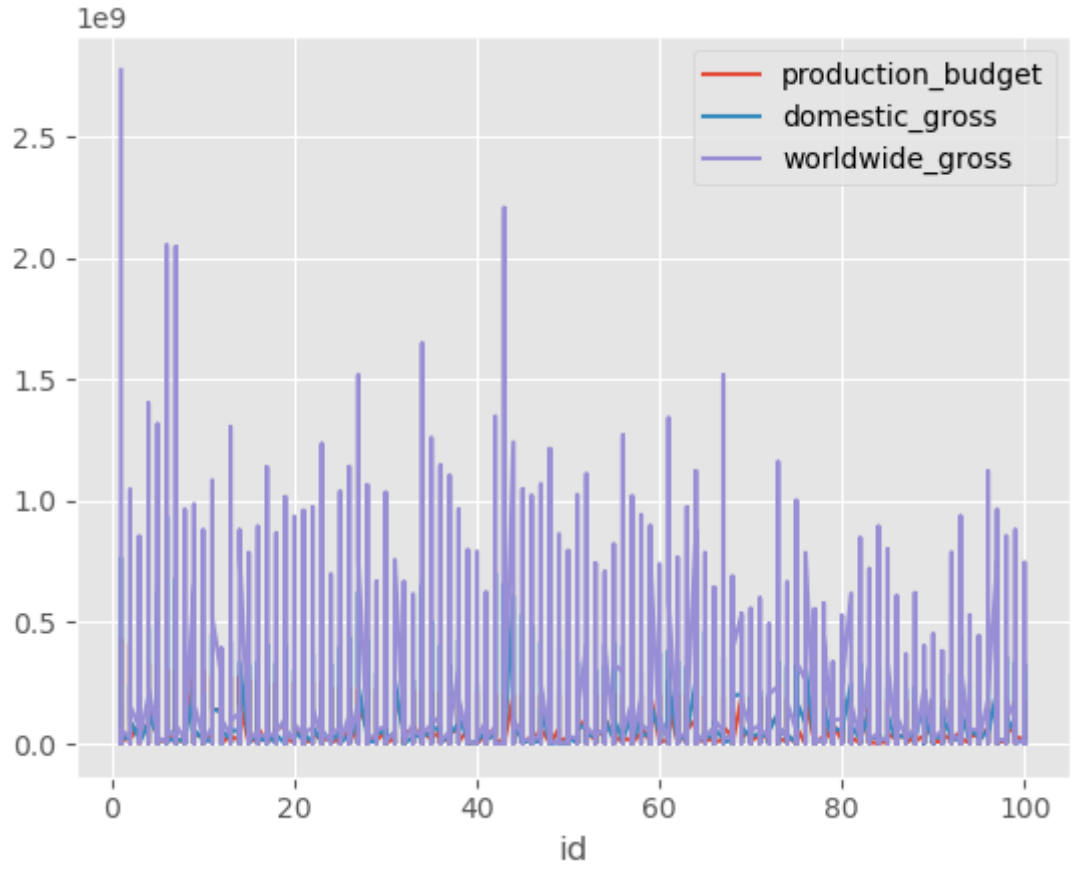
in these section we are going to visualize the data and performe various comparisons

In [820]:

```
rt.plot()
```

Out[820]:

```
<AxesSubplot:xlabel='id'>
```



In [821]:

```
rt.plot(title= 'comparison', subplots= True,  figsize= (10,15))
```

Out[821]:

```
array([<AxesSubplot:xlabel='id'>, <AxesSubplot:xlabel='id'>,
       <AxesSubplot:xlabel='id'>], dtype=object)
```

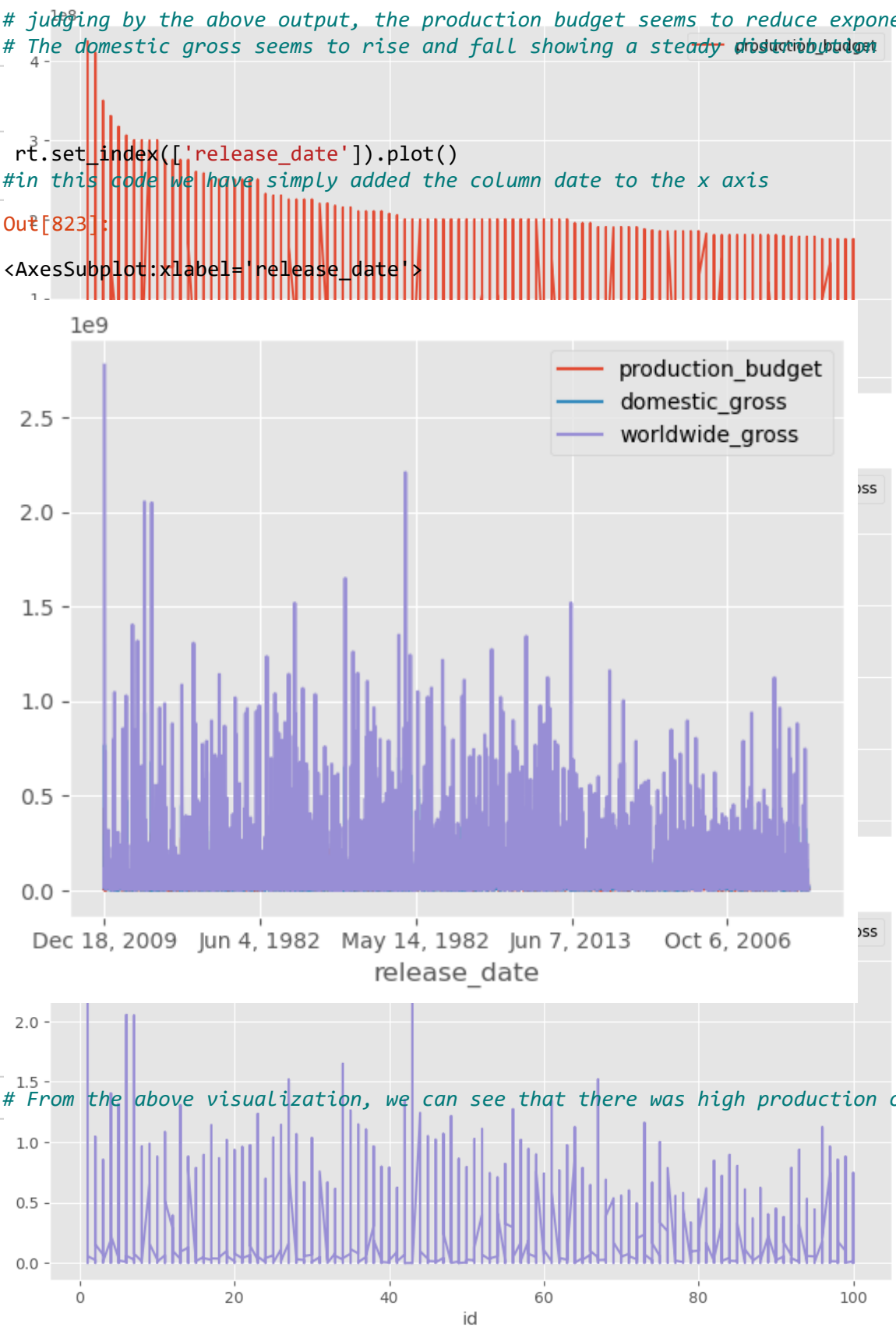
comparison

```
# judging by the above output, the production budget seems to reduce exponentially before
# The domestic gross seems to rise and fall showing a steady production budget
```

```
rt.set_index(['release_date']).plot()
#in this code we have simply added the column date to the x axis
```

Out[823]:

<AxesSubplot:xlabel='release_date'>



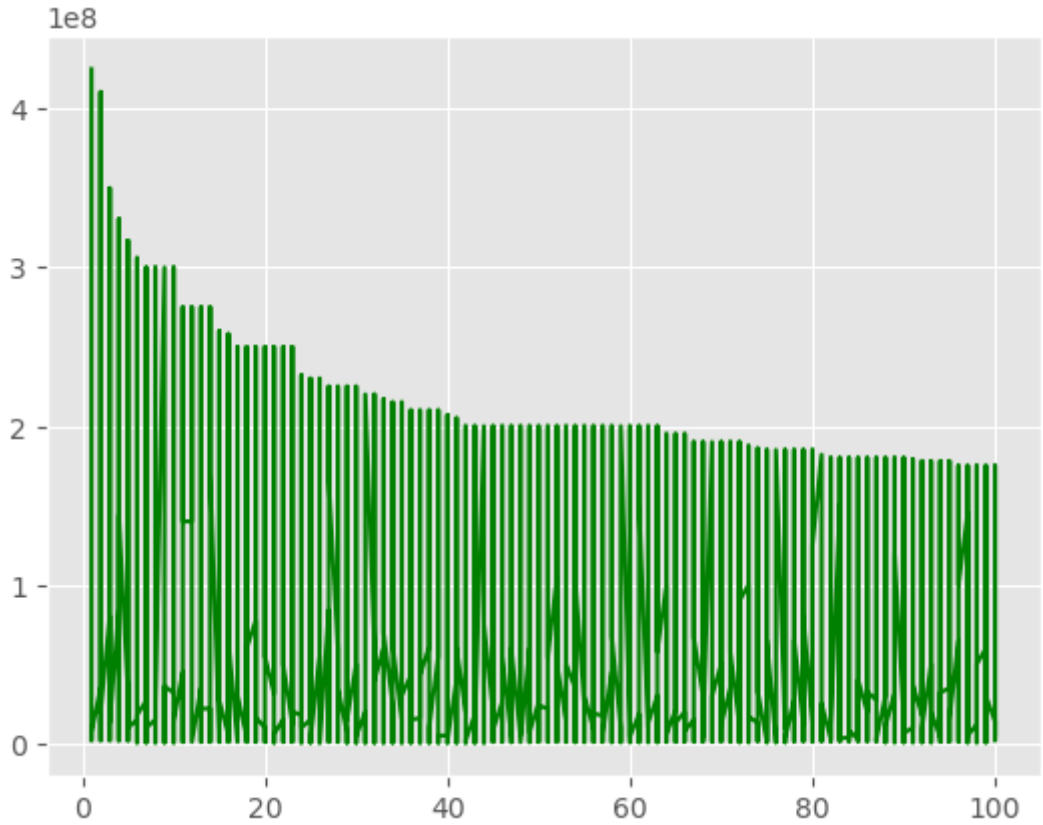

```
# From the above visualization, we can see that there was high production of movies in 20
```

In [828]:

```
plt.plot(rt['production_budget', ], 'g')
```

Out[828]:

```
[<matplotlib.lines.Line2D at 0x1f6dd186a90>]
```



In [ ]:

```
# production budget for the movies generally seems to be reducing
```

In [518]:

```
#lets now work with another dataset
```

bom.movie_gross.csv.gz

In [831]:

```
bm= pd.read_csv('bom.movie_gross.csv.gz', index_col= 0)
bm.head()
#opening the dataset
```

Out[831]:

title	studio	domestic_gross	foreign_gross	year
Toy Story 3	BV	415000000.0	652000000	2010
Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
Inception	WB	292600000.0	535700000	2010
Shrek Forever After	P/DW	238700000.0	513900000	2010

In [832]:

```
bm.dtypes
# checking the dataset for datatypes
```

Out[832]:

```
studio             object
domestic_gross    float64
foreign_gross      object
year                int64
dtype: object
```

In [833]:

```
bm.info
#getting summarized info in the dataset
```

Out[833]:

```
<bound method DataFrame.info of
studio  domestic_gross  \
title
Toy Story 3                                          BV     415000000.0
Alice in Wonderland (2010)                           BV     334200000.0
Harry Potter and the Deathly Hallows Part 1          WB     296000000.0
Inception                                            WB     292600000.0
Shrek Forever After                                P/DW     238700000.0
...                                                 ...             ...
The Quake                                         Magn.          6200.0
Edward II (2018 re-release)                          FM          4800.0
El Pacto                                           Sony          2500.0
The Swan                                     Synergetic          2400.0
An Actor Prepares                                 Grav.          1700.0

                                           foreign_gross  year
title
Toy Story 3                                    652000000  2010
Alice in Wonderland (2010)                     691300000  2010
Harry Potter and the Deathly Hallows Part 1    664300000  2010
Inception                                      535700000  2010
Shrek Forever After                            513900000  2010
...                                                  ...   ...
The Quake                                            NaN  2018
Edward II (2018 re-release)                          NaN  2018
El Pacto                                             NaN  2018
The Swan                                             NaN  2018
An Actor Prepares                                    NaN  2018

[3387 rows x 4 columns]>
```

In [834]:

```
bm.columns
#checking the dataset columns.
```

Out[834]:

```
Index(['studio', 'domestic_gross', 'foreign_gross', 'year'], dtype='objec
t')
```

In [835]:

```
bm.tail()
#checking the last few rows of the dataset
```

Out[835]:

title	studio	domestic_gross	foreign_gross	year
The Quake	Magn.	6200.0	NaN	2018
Edward II (2018 re-release)	FM	4800.0	NaN	2018
El Pacto	Sony	2500.0	NaN	2018
The Swan	Synergetic	2400.0	NaN	2018
An Actor Prepares	Grav.	1700.0	NaN	2018

In [836]:

```
bm.describe()
```

Out[836]:

	domestic_gross	year
count	3.359000e+03	3387.000000
mean	2.874585e+07	2013.958075
std	6.698250e+07	2.478141
min	1.000000e+02	2010.000000
25%	1.200000e+05	2012.000000
50%	1.400000e+06	2014.000000
75%	2.790000e+07	2016.000000
max	9.367000e+08	2018.000000

data cleaning

In [837]:

```

# lets start by checking null values in the dataset
```

In [838]:

```
bm.isnull()
#the output below shows we have some null values indicated by the string (true)
```

Out[838]:

title	studio	domestic_gross	foreign_gross	year
Toy Story 3	False	False	False	False
Alice in Wonderland (2010)	False	False	False	False
Harry Potter and the Deathly Hallows Part 1	False	False	False	False
Inception	False	False	False	False
Shrek Forever After	False	False	False	False
...	...	...	...	...
The Quake	False	False	True	False
Edward II (2018 re-release)	False	False	True	False
El Pacto	False	False	True	False
The Swan	False	False	True	False
An Actor Prepares	False	False	True	False

3387 rows × 4 columns

In [839]:

```
#lets count the number of these null values
bm.isnull().sum()
```

Out[839]:

```
studio              5
domestic_gross     28
foreign_gross    1350
year                0
dtype: int64
```

In [840]:

```
# The output above shows we have null values in 3 columns
#lets proceed to drop them
bm= bm.dropna()
bm.head()
```

Out[840]:

title	studio	domestic_gross	foreign_gross	year
Toy Story 3	BV	415000000.0	652000000	2010
Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
Inception	WB	292600000.0	535700000	2010
Shrek Forever After	P/DW	238700000.0	513900000	2010

In [841]:

```
bm.isnull().sum()
```

Out[841]:

```
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

In [842]:

```
# from the output above we no longer have null values
```

In [859]:

```
#lets now sort the dataframe by  the column(years)
bm = bm.sort_values('year')
bm.head()
```

Out[859]:

title	studio	domestic_gross	foreign_gross	year
Toy Story 3	BV	415000000.0	652000000	2010
Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
Inception	WB	292600000.0	535700000	2010
Shrek Forever After	P/DW	238700000.0	513900000	2010

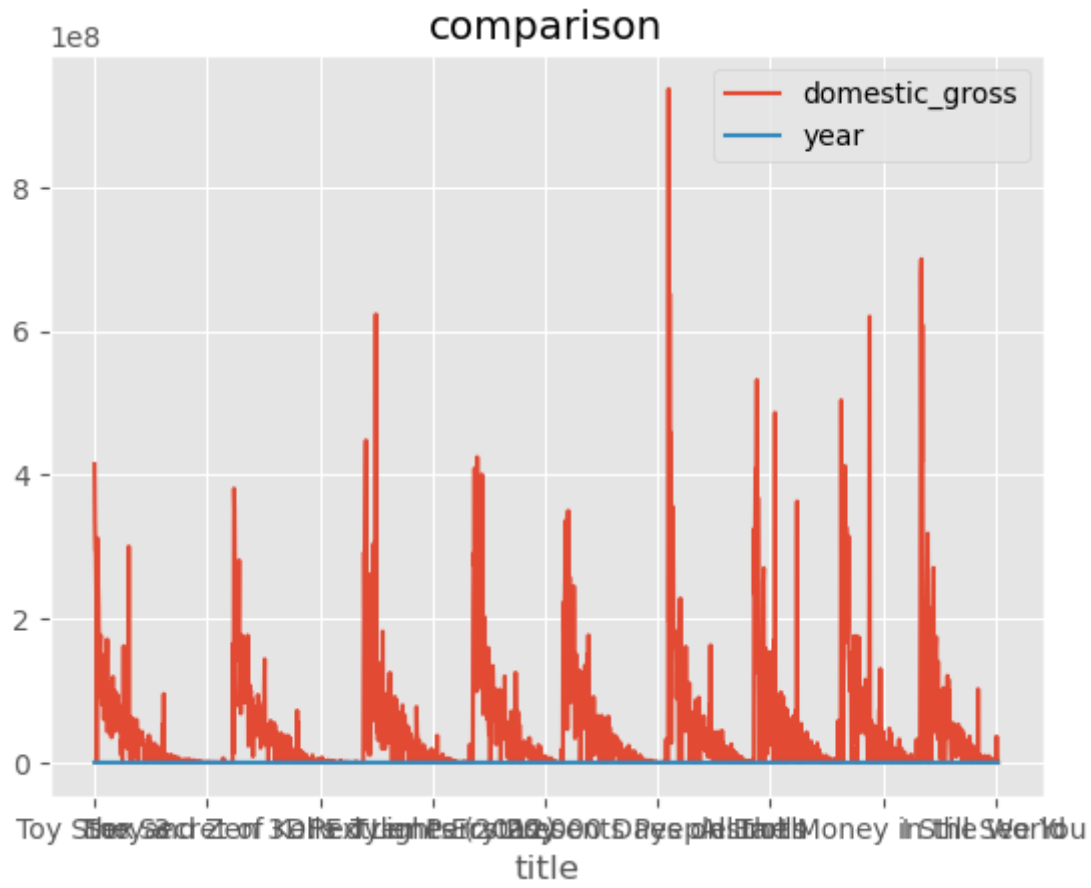
data visualization

In [860]:

```
bm.plot()
plt.title('comparison')
plt
```

Out[860]:

```
<module 'matplotlib.pyplot' from 'C:\\Users\\Emmanuel Omondi\\anaconda3\\a
naconda install\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



subplots

In [861]:

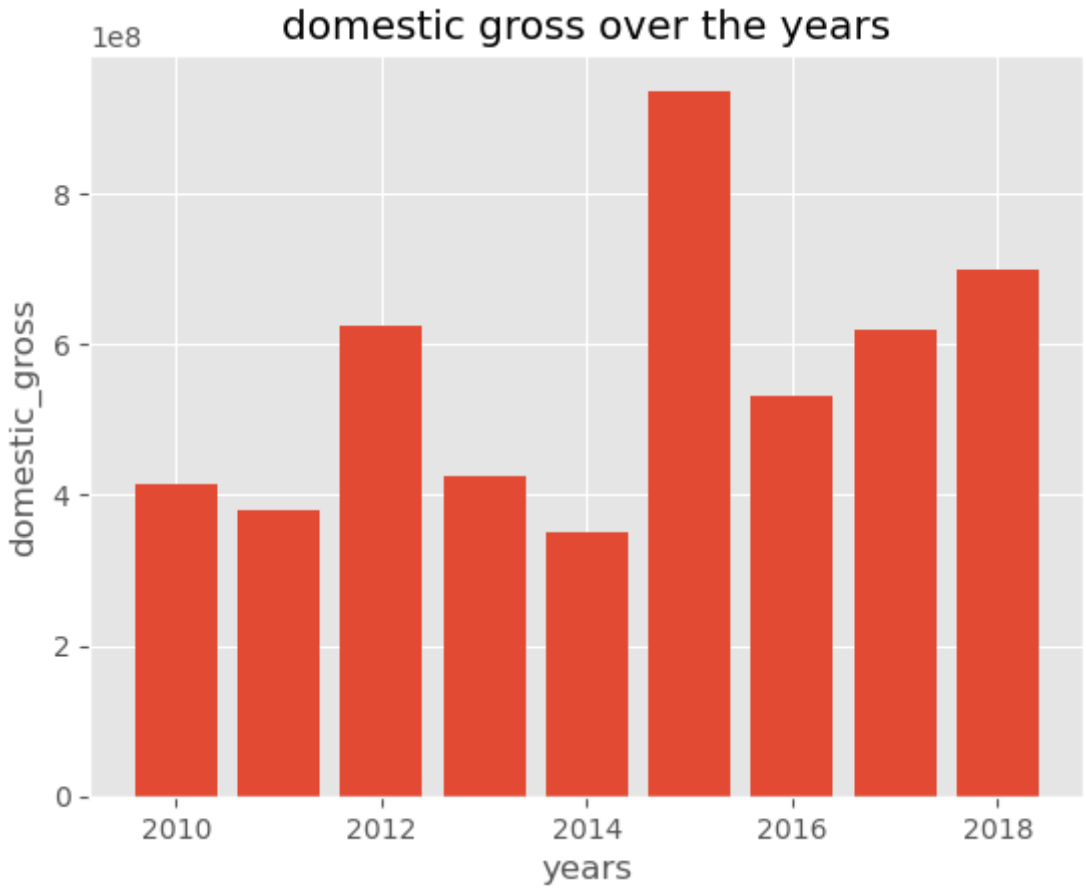
```
studio= bm['studio']
years= bm['year']
foreign = bm['foreign_gross']
domestic= bm['domestic_gross']
```

In [862]:

```
plt.bar(years, domestic)
plt.ylabel('domestic_gross')
plt.xlabel('years')
plt.title('domestic gross over the years')
```

Out[862]:

```
Text(0.5, 1.0, 'domestic gross over the years')
```

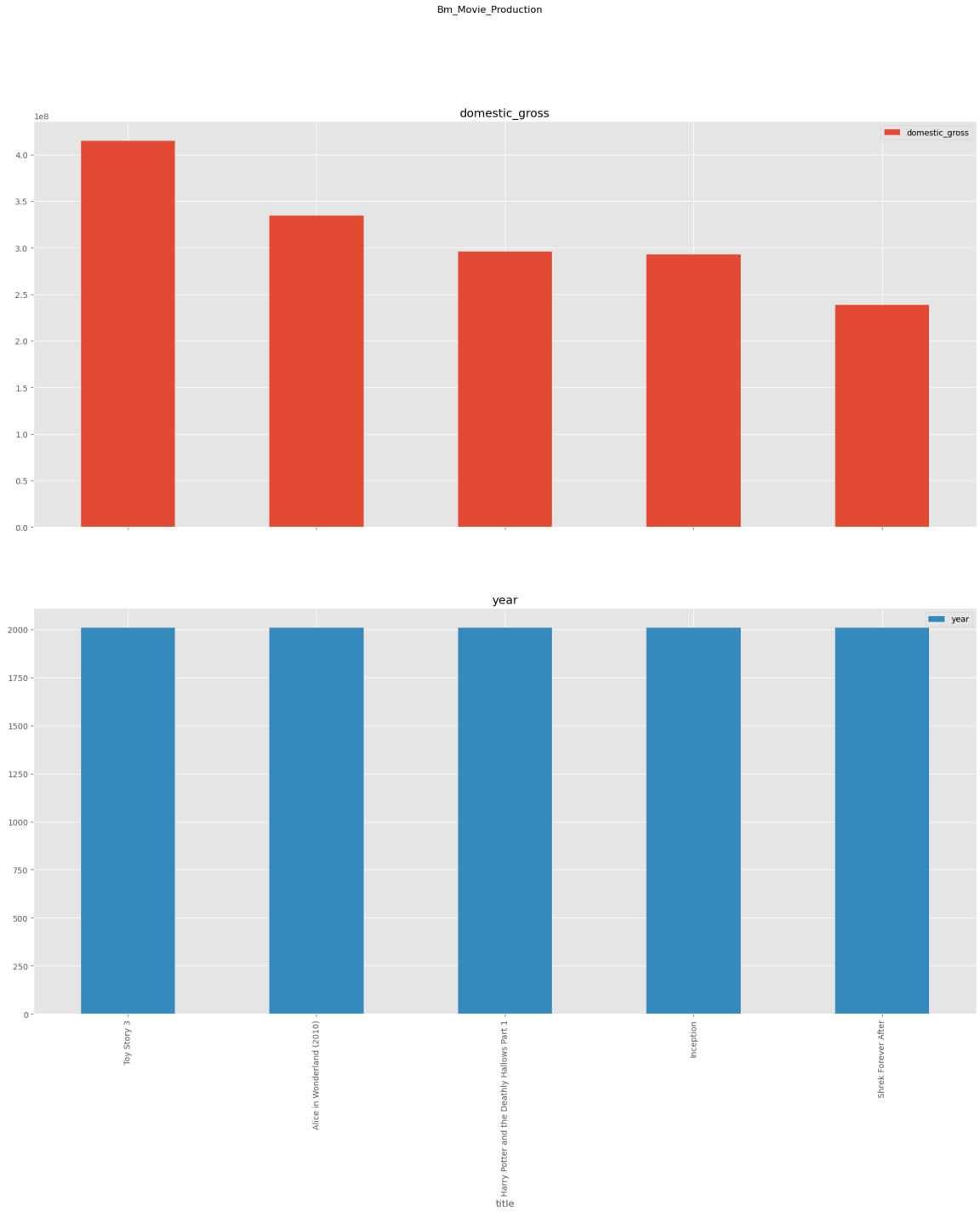


In [863]:

```
bm.head(5).plot.bar(title= 'Bm_Movie_Production', subplots= True, figsize= (21,20))
```

Out[863]:

```
array([<AxesSubplot:title={'center':'domestic_gross'}, xlabel='title'>,
       <AxesSubplot:title={'center':'year'}, xlabel='title'>],
      dtype=object)
```



In [865]:

```
bm.head()
```

Out[865]:

title	studio	domestic_gross	foreign_gross	year
Toy Story 3	BV	415000000.0	652000000	2010
Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
Inception	WB	292600000.0	535700000	2010
Shrek Forever After	P/DW	238700000.0	513900000	2010

In [871]:

```
bm.head()
```

Out[871]:

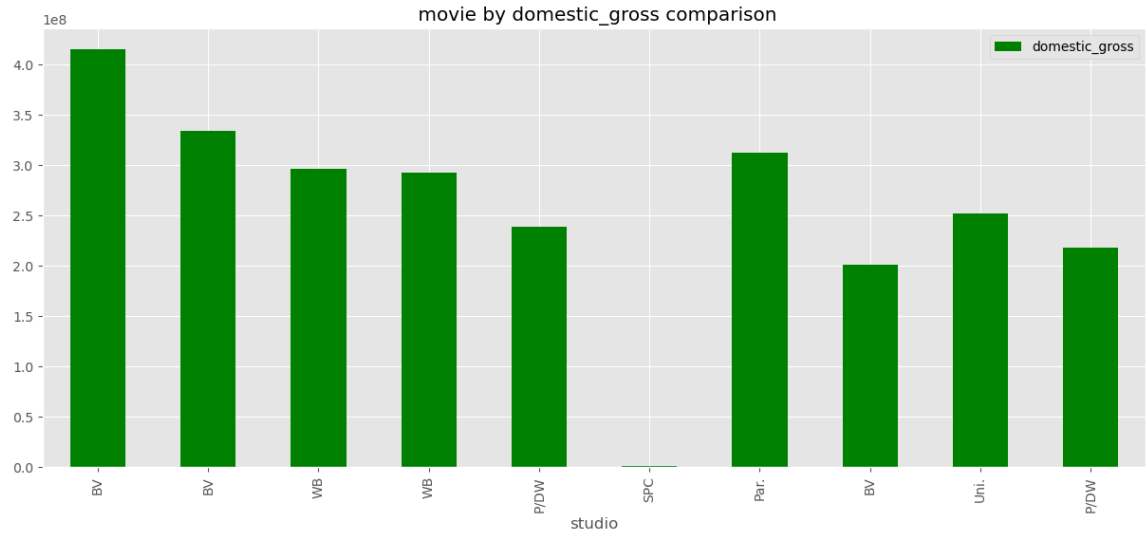
title	studio	domestic_gross	foreign_gross	year
Toy Story 3	BV	415000000.0	652000000	2010
Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
Inception	WB	292600000.0	535700000	2010
Shrek Forever After	P/DW	238700000.0	513900000	2010

In [882]:

```
bm.head(10).plot( 'studio', 'domestic_gross', kind='bar', color= 'g', figsize= (15,6))
plt.title(' movie by domestic_gross comparison')
```

Out[882]:

```
Text(0.5, 1.0, ' movie by domestic_gross comparison')
```



In [ ]:

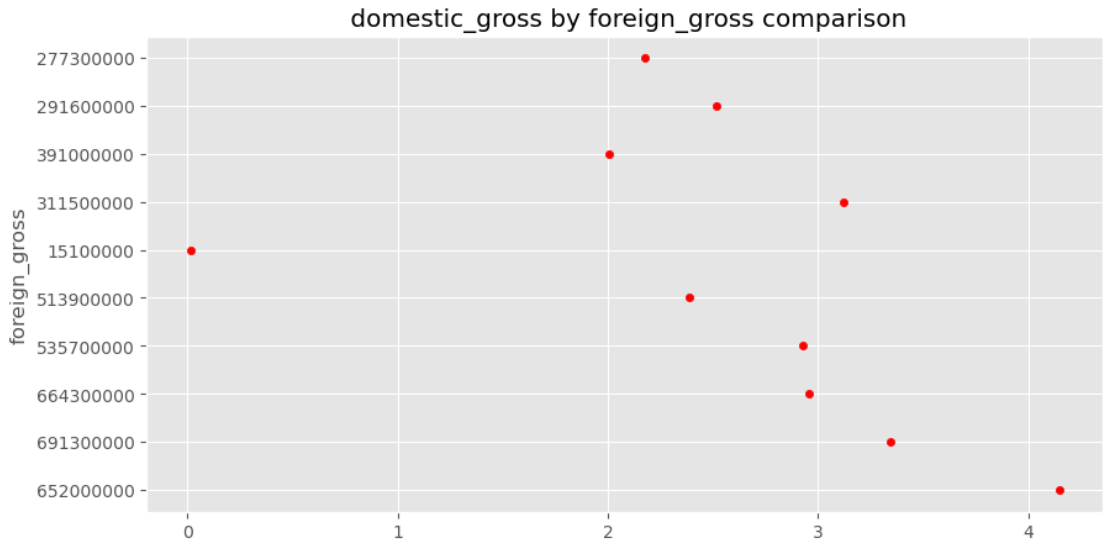
```
# The reading above shows studio bv has the highest domestic gross.
```

In [891]:

```
bm.head(10).plot( 'domestic_gross', 'foreign_gross', kind='scatter', color= 'r', figsize=
plt.title(' domestic_gross by foreign_gross comparison')
```

Out[891]:

```
Text(0.5, 1.0, ' domestic_gross by foreign_gross comparison')
```



In [ ]: