

1 Phase 1 Project Submission

Please fill out:

- Student name: Matt LeGro
- Student pace: self paced / part time / (full time)
- Scheduled project review date/time: Friday, May 7 @ 12:00pm MST
- Instructor name: Abhineet (Reviewing Instructor: Lindsey)

```
In [1]: import os
import pandas as pd
import numpy as np
import tmdbsimple as tmdb
import requests
import matplotlib.pyplot as plt
import seaborn as sns
import ast
executed in 834ms, finished 16:10:08 2021-05-06
```

Let's load in our super dataframe!

```
In [2]: movie_collection = pd.read_csv('./dataFiles/movie_collection.csv')
executed in 29ms, finished 16:10:09 2021-05-06
```

```
In [3]: movie_collection.head(2)
executed in 12ms, finished 16:10:09 2021-05-06
```

Out[3]:

	tconst	Title	Genres	runtime_minutes	vote_average	vote_count	budget_est	worldw
0	tt0892769	How to Train Your Dragon	['Fantasy', 'Adventure', 'Animation', 'Family']	98	8.095082	618909	165000000.0	49
1	tt1228705	Iron Man 2	['Adventure', 'Action', 'Science Fiction']	124	6.996308	670058	185000000.0	62

```
In [4]: movie_collection.columns
executed in 15ms, finished 16:10:09 2021-05-06
```

```
Out[4]: Index(['tconst', 'Title', 'Genres', 'runtime_minutes', 'vote_average',
       'vote_count', 'budget_est', 'worldwide_gross', 'release_week_day',
       'release_day', 'release_month', 'release_year'],
       dtype='object')
```

In [5]:

`movie_collection.info()`

executed in 15ms, finished 16:10:09 2021-05-06

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2373 entries, 0 to 2372
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   tconst          2373 non-null    object  
 1   Title            2373 non-null    object  
 2   Genres           2373 non-null    object  
 3   runtime_minutes  2373 non-null    int64  
 4   vote_average     2373 non-null    float64 
 5   vote_count       2373 non-null    int64  
 6   budget_est       2373 non-null    float64 
 7   worldwide_gross  2373 non-null    float64 
 8   release_week_day 2373 non-null    int64  
 9   release_day      2373 non-null    int64  
 10  release_month    2373 non-null    int64  
 11  release_year     2373 non-null    int64  
dtypes: float64(3), int64(6), object(3)
memory usage: 222.6+ KB
```

In [6]:

`movie_collection.describe()`

executed in 30ms, finished 16:10:09 2021-05-06

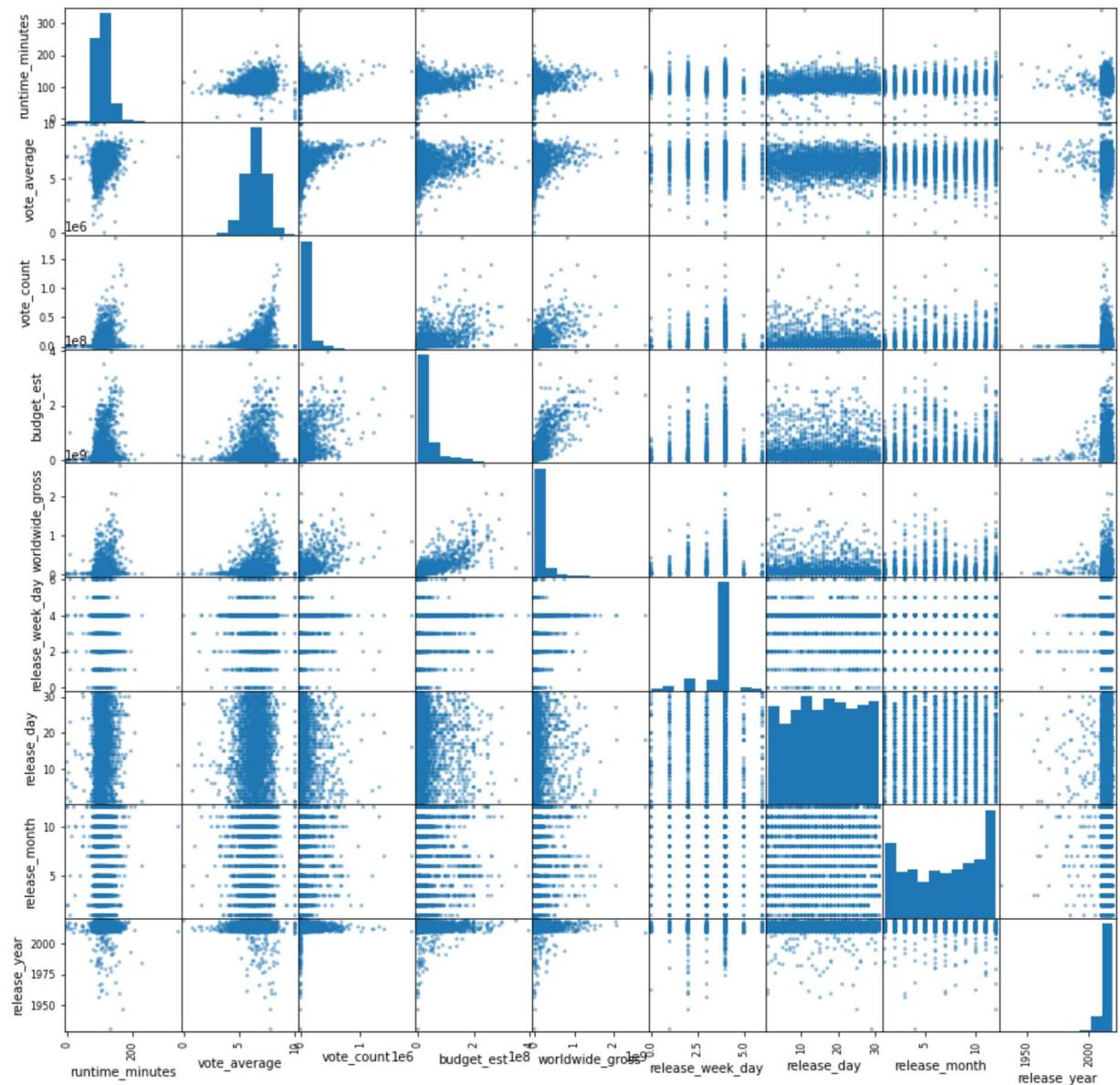
Out[6]:

	runtime_minutes	vote_average	vote_count	budget_est	worldwide_gross	release_week
count	2373.000000	2373.000000	2.373000e+03	2.373000e+03	2.373000e+03	2373.00
mean	108.500211	6.382529	7.379590e+04	3.580101e+07	1.128888e+08	3.63
std	20.781843	1.031946	1.438744e+05	4.941197e+07	2.176952e+08	0.98
min	0.000000	0.000000	1.000000e+00	1.000000e+00	1.000000e+00	0.00
25%	95.000000	5.800000	3.280000e+02	5.400000e+06	4.423508e+06	4.00
50%	106.000000	6.400000	8.665000e+03	1.700000e+07	3.029494e+07	4.00
75%	119.000000	7.093148	8.690500e+04	4.000000e+07	1.146292e+08	4.00
max	339.000000	10.000000	1.863252e+06	3.953000e+08	2.787965e+09	6.00

Okay, from here I am seeing: movies tend to be released more during the later months, we don't have data for years after 2019, there may be a record with one vote for a rating of zero. Similarly there is a movie with a budget of 1 dollar. Release day throughout the month seems fairly evenly distributed.

In [7]: `pd.plotting.scatter_matrix(movie_collection, figsize =(15,15));`

executed in 3.33s, finished 16:10:12 2021-05-06



Noteable Features: Most of the data is from after 2000, maybe even after 2010, depending on the binning here. Worlwide Gross seems to track slightly with budget, which is reasonable. Same with vote_average, though it is hard to quantify how you're movie will get voted on before it is released. Of course there may be some indirect relationships we can make. Slight improvements on vote_average with increasing movie length, for example. There may be something of a rightward skew with respect to worldwide_gross and runtime as well, but it could be because with increased runtime comes increased budget and increased gross. Another interesting feature is that there are

obvious months and days on which high-grossing movies are not released, though based on voter ratings, these movies aren't necessarily worse than their counterparts. Let's quickly add a ROI column to see if there are better defined relationships there.

```
In [8]: movie_collection['ROI_percent'] = (
    movie_collection.worldwide_gross - movie_collection.budget_est) / movie_collection.budget_est
```

executed in 15ms, finished 16:10:13 2021-05-06

```
In [9]: movie_collection.head(2)
```

executed in 15ms, finished 16:10:13 2021-05-06

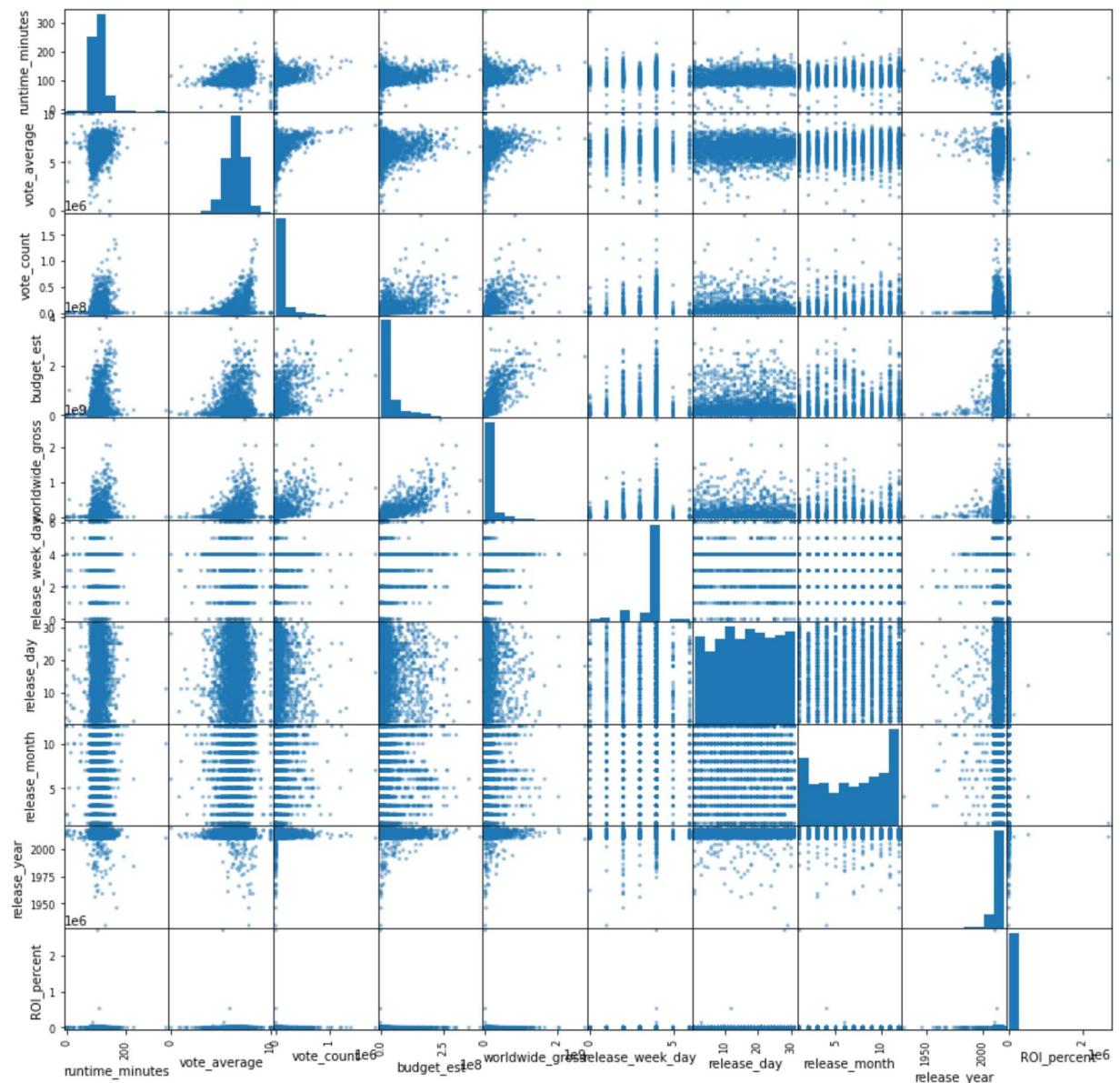
Out[9]:

	tconst	Title	Genres	runtime_minutes	vote_average	vote_count	budget_est	worldwide_gross
0	tt0892769	How to Train Your Dragon	['Fantasy', 'Adventure', 'Animation', 'Family']	98	8.095082	618909	165000000.0	49
1	tt1228705	Iron Man 2	['Adventure', 'Action', 'Science Fiction']	124	6.996308	670058	185000000.0	62



```
In [10]: pd.plotting.scatter_matrix(movie_collection, figsize =(15,15));
```

executed in 3.97s, finished 16:10:17 2021-05-06



Okay, there are some ROI outliers obscuring the data, lets see what they are.

In [11]: `movie_collection.sort_values(by='ROI_percent', ascending = False).head(50)`
executed in 30ms, finished 16:10:17 2021-05-06

Out[11]:

	tconst	Title	Genres	runtime_minutes	vote_average	vote_count	budget
1316	tt0893412	From Prada to Nada	['Comedy', 'Drama', 'Romance']	107	5.200000	163	
1502	tt1595656	To the Wonder	['Drama', 'Romance']	112	5.900000	266	
1337	tt1436559	Love, Wedding, Marriage	['Comedy', 'Romance']	90	5.300000	117	
750	tt2309260	The Gallows	['Horror', 'Thriller']	84	4.219320	18354	10
1466	tt2389344	The Mistress	['Romance', 'Drama']	125	6.400000	7	2
1594	tt2133234	Haunted	['Horror']	93	5.100000	5	
			['Action',				

Looking at the top 50 best performing by ROI percent, there are mostly legitimate movies / data, which is encouraging. The top three are definitely bad values; a quick google search can verify that. We'll have to do some plotting while excluding outliers or using specific budget ranges. Since I know there are some bad actors in this head, I'll remove them.

```
In [12]: remove_tconsts = ['tt0893412', 'tt1595656', 'tt1436559']
remove_indexes = movie_collection[ movie_collection['tconst'].isin(remove_tconsts)]
movie_collection = movie_collection.drop(remove_indexes)
movie_collection = movie_collection.reset_index()
movie_collection = movie_collection.drop(columns = 'index')
movie_collection.sort_values(by='ROI_percent', ascending = False).head(50)

executed in 31ms, finished 16:10:17 2021-05-06
```

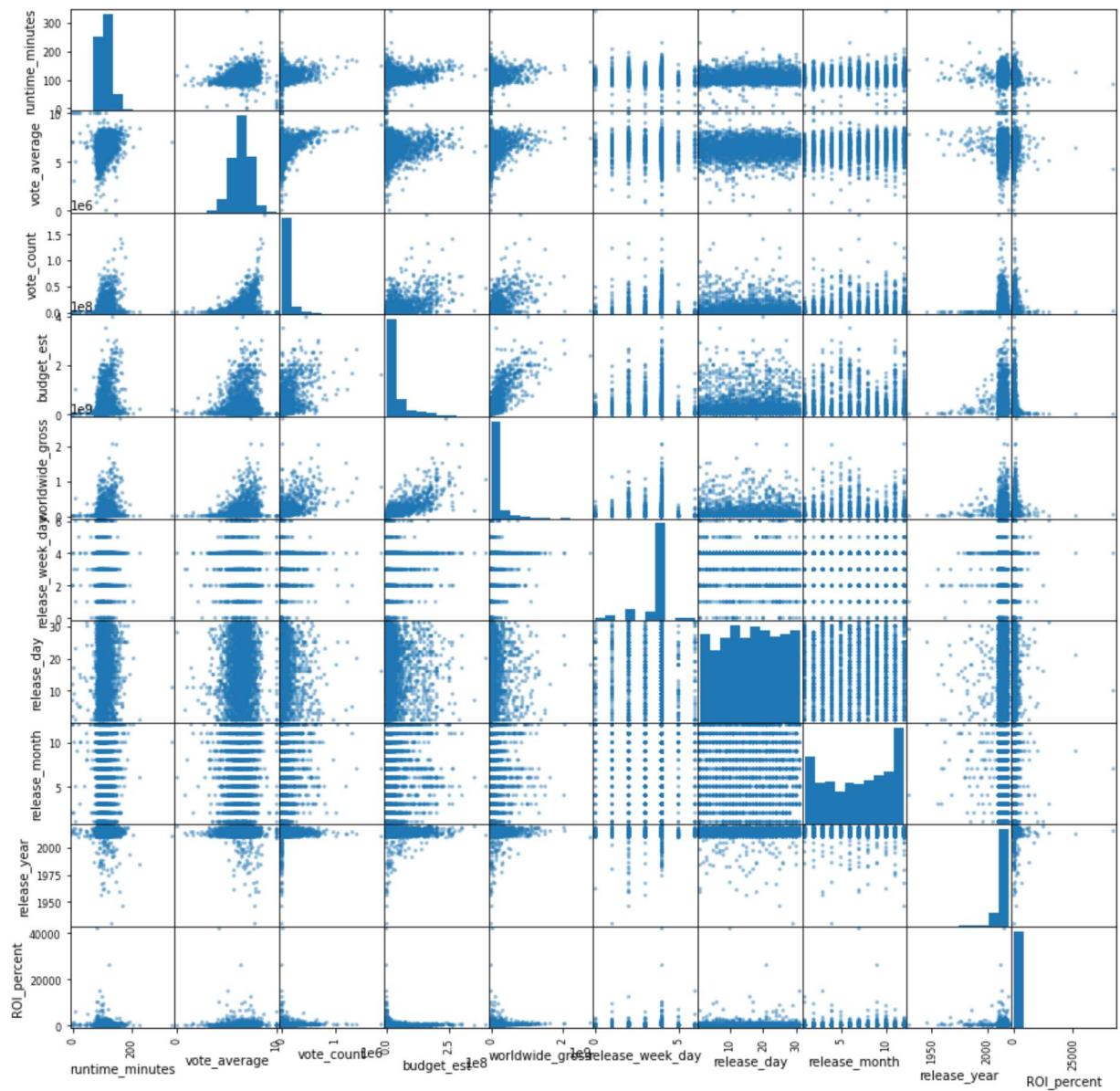
Out[12]:

	tconst	Title	Genres	runtime_minutes	vote_average	vote_count	budg
750	tt2309260	The Gallows	['Horror', 'Thriller']	84	4.219320	18354	10
1464	tt2389344	The Mistress	['Romance', 'Drama']	125	6.400000	7	2
1591	tt2133234	Haunted	['Horror']	93	5.100000	5	
2125	tt7961060	Dragon Ball Super: Broly	['Action', 'Animation', 'Fantasy', 'Adventure']...	101	7.400000	721	100
394	tt1560985	The Devil Inside	['Thriller', 'Horror']	83	4.206492	33963	100
1193	tt1584016	Catfish	['Documentary']	87	6.800000	284	3
1562	tt2459022	Lost in Thailand	['Comedy', 'Adventure']	105	6.500000	36	220

A lot of these will still be outliers, but it getting rid of those obviously wrong ones was good.

```
In [13]: pd.plotting.scatter_matrix(movie_collection, figsize =(15,15));
```

executed in 3.88s, finished 16:10:21 2021-05-06



It isn't immediately obvious how I could use the Genres column as is, so let's implement what we did in the study group and make a boolean column for each unique genre in our collection.

```
In [14]: all_genres = set()
for genres in movie_collection['Genres']:
    all_genres.update(ast.literal_eval(genres))
```

executed in 29ms, finished 16:10:21 2021-05-06

```
In [15]: all_genres
```

executed in 15ms, finished 16:10:21 2021-05-06

```
Out[15]: {'Action',
          'Adventure',
          'Animation',
          'Biography',
          'Comedy',
          'Crime',
          'Documentary',
          'Drama',
          'Family',
          'Fantasy',
          'History',
          'Horror',
          'Music',
          'Musical',
          'Mystery',
          'Romance',
          'Sci-Fi',
          'Science Fiction',
          'Sport',
          'TV Movie',
          'Thriller',
          'War',
          'Western'}
```

```
In [16]: for genre in all_genres:
    movie_collection[genre] = np.zeros(shape=movie_collection.shape[0])
for index, row in movie_collection.iterrows():
    for genre in ast.literal_eval(row['Genres']):
        movie_collection.loc[index, genre] = 1
```

executed in 1.22s, finished 16:10:22 2021-05-06

```
In [17]: genre_count = {}
for col in movie_collection.loc[:, movie_collection.columns.isin(all_genres)].columns:
    count = (movie_collection[col]).sum()
    genre_count.update( {col:count} )
executed in 14ms, finished 16:10:22 2021-05-06
```

```
In [18]: genre_count
executed in 16ms, finished 16:10:23 2021-05-06
```

```
Out[18]: {'Mystery': 188.0,
          'Fantasy': 242.0,
          'Family': 236.0,
          'Drama': 1160.0,
          'Animation': 165.0,
          'Musical': 4.0,
          'Sci-Fi': 12.0,
          'Sport': 12.0,
          'History': 110.0,
          'Documentary': 55.0,
          'TV Movie': 2.0,
          'Western': 26.0,
          'Adventure': 404.0,
          'Comedy': 775.0,
          'Thriller': 608.0,
          'Science Fiction': 240.0,
          'War': 79.0,
          'Action': 624.0,
          'Biography': 20.0,
          'Romance': 353.0,
          'Crime': 322.0,
          'Music': 68.0,
          'Horror': 260.0}
```

Okay, there are a few equivalent genres. In our mapping to a primary_genre next, we take care of this. I would like to establish a primary genre for each movie to be able to more easily group by genre. I will accomplish this by choosing, from the list of genres for each movie, the genre that is the most common as an identifier. Therefore, with the four most common genre designations as ['Action','Comedy','Drama','Thriller'], all movies will be mapped to one of those. If they have one or more in the list, it chooses the one with the highest total count of movies of that genre in the dataset. If it does not have any of those, it will be mapped to one using the following dictionary:

```
general_genre_dict = {'Adventure':'Action', 'Horror':'Thriller', 'Romance':'Drama', 'Fantasy':'Action',
                      'Animation':'Comedy', 'Documentary':'Drama', 'Family':'Comedy', 'Musical':'Drama',
                      'History':'Drama', 'Music':'Drama', 'Crime':'Thriller', 'Western':'Action', 'Sci-Fi':'Action', 'Science Fiction':'Action'}
```

In [19]: `movie_collection.loc[movie_collection.Genres == '[]']`

executed in 29ms, finished 16:10:23 2021-05-06

Out[19]:

	tconst	Title	Genres	runtime_minutes	vote_average	vote_count	budget_est	worldv
1461	tt1765729	Legatee	[]	102	2.7	12	6500000.0	
1472	tt2243629	The Thought Exchange	[]	92	9.0	2	212.0	
1956	tt4732072	The Girl on the Mat	[]	27	10.0	2	4000.0	

3 rows × 36 columns

A couple movies with no genres causing problems, lets eliminate them.

In [20]: `remove_tconsts = ['tt1765729', 'tt2243629', 'tt4732072']
remove_indexes = movie_collection[movie_collection['tconst'].isin(remove_tconsts)]
movie_collection = movie_collection.drop(remove_indexes)
movie_collection = movie_collection.reset_index()
movie_collection = movie_collection.drop(columns = 'index')`

executed in 14ms, finished 16:10:23 2021-05-06

In [21]: `movie_collection['primary_genre'] = np.zeros(shape=movie_collection.shape[0])`

executed in 15ms, finished 16:10:23 2021-05-06

```
In [22]: general_genre_dict = {'Adventure':'Action',
                             'Horror':'Thriller',
                             'Romance':'Drama',
                             'Fantasy':'Action',
                             'Animation':'Comedy',
                             'Documentary':'Drama',
                             'Family':'Comedy',
                             'Musical':'Drama',
                             'History':'Drama',
                             'Music':'Drama',
                             'Crime':'Thriller',
                             'Western':'Action',
                             'Sci-Fi':'Action',
                             'Science Fiction':'Action'}
for index, row in movie_collection.iterrows():
    highest_count_genre = None
    for genre in ast.literal_eval(row['Genres']):
        if not highest_count_genre or genre_count[genre] > genre_count[highest_count_genre]:
            highest_count_genre = genre
    if highest_count_genre not in ['Action', 'Comedy', 'Drama', 'Thriller']:
        highest_count_genre = general_genre_dict[highest_count_genre]
    movie_collection.loc[index, 'primary_genre'] = highest_count_genre
```

executed in 751ms, finished 16:10:24 2021-05-06

In [23]: movie_collection.primary_genre.value_counts()

executed in 15ms, finished 16:10:24 2021-05-06

Out[23]:

primary_genre	count
Drama	1214
Comedy	529
Action	401
Thriller	223

Name: primary_genre, dtype: int64

Seems to have worked. 'Drama' is a lame category to deal with but oh well. Let's add a few columns to enable easier grouping by budget and ROI in the future.

In [24]: movie_collection['budget_category'] = movie_collection.budget_est.apply(`lambda x:`

executed in 14ms, finished 16:10:24 2021-05-06

In [25]: movie_collection['ROI_category'] = movie_collection.ROI_percent.apply(`lambda x:`

executed in 15ms, finished 16:10:24 2021-05-06

Seems important to establish a relationship between budget and ROI.

In [26]:

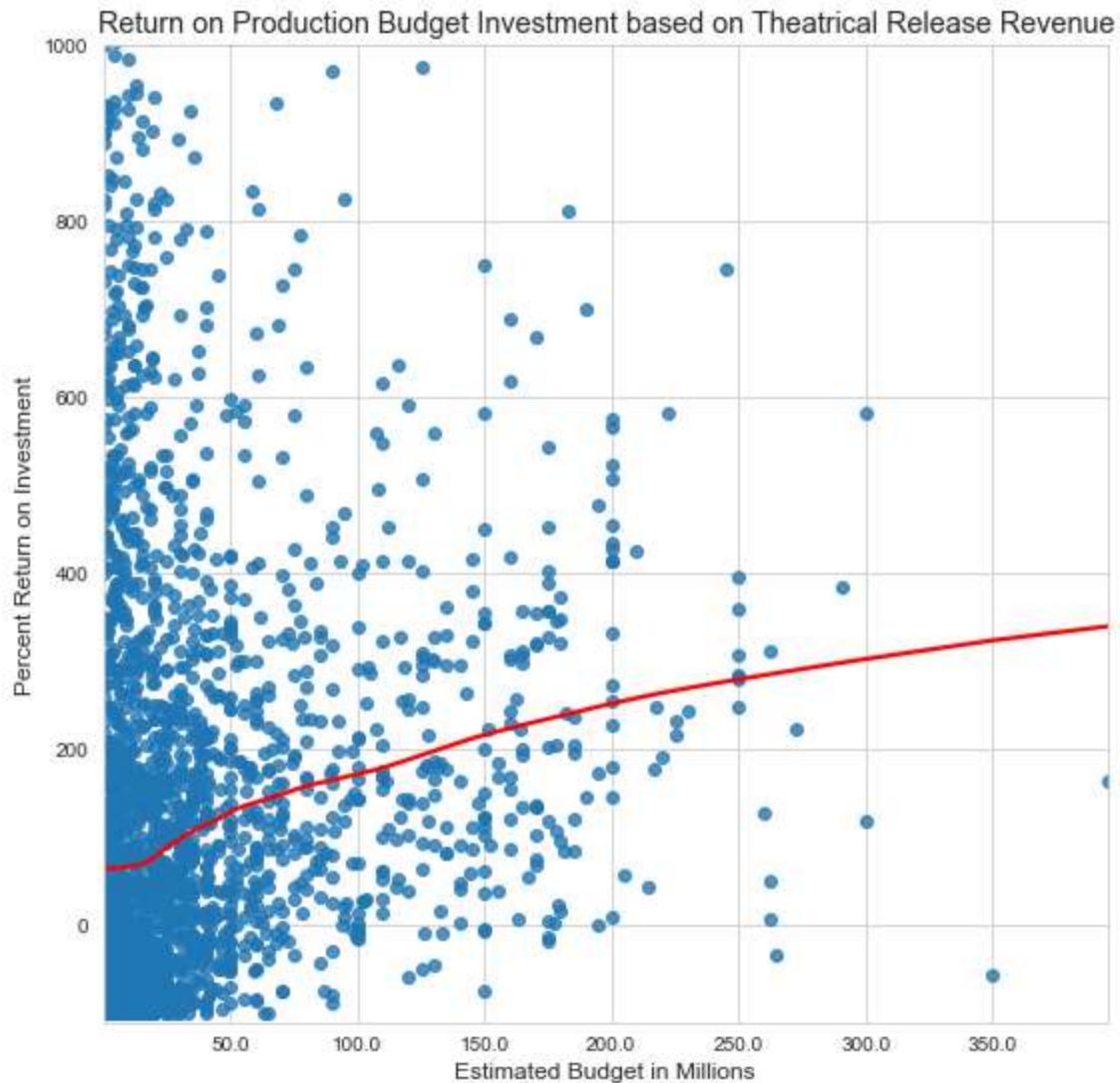
```

sns.set_style('whitegrid')
fig1, ax1 = plt.subplots(figsize=(9,9))
sns.regplot(data=movie_collection,x='budget_est',y='ROI_percent',
             truncate=False,lowess = True,line_kws = {'color':'red'})
plt.ylim(-110,1000)
plt.title('Return on Production Budget Investment based on Theatrical Release Revenue')
plt.xlabel('Estimated Budget in Millions',fontsize='large');
plt.ylabel('Percent Return on Investment',fontsize='large');
ax1.xaxis.get_major_formatter().set_scientific(False);
xlabels = [x for x in ax1.get_xticks()/1000000]
ax1.set_xticklabels(xlabels);

```

executed in 389ms, finished 16:10:25 2021-05-06

<ipython-input-26-f1947252cc71>:11: UserWarning: FixedFormatter should only be used together with FixedLocator
 ax1.set_xticklabels(xlabels);



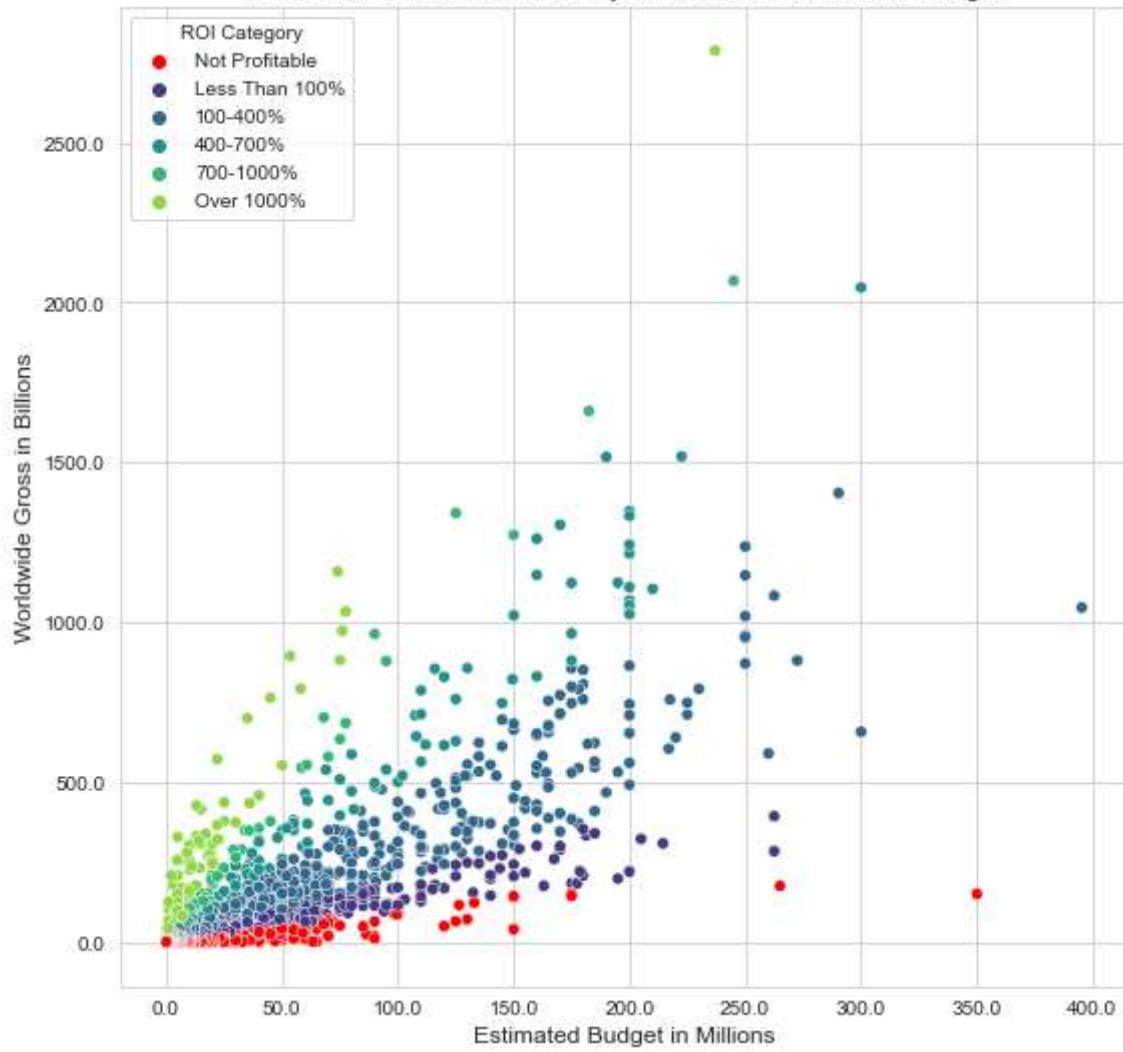
Let's also look at a scatter of our three most important categories.

```
In [27]: pal = sns.color_palette('viridis',5)
rgb_tuples = [(1,0,0)]
for color in pal:
    rgb_tuples.append(color)
new_pal = sns.color_palette(rgb_tuples)
fig2, ax2 = plt.subplots(figsize=(9,9))
sns.scatterplot(x='budget_est',y='worldwide_gross',data=movie_collection,
                 hue='ROI_category',legend=True,
                 hue_order = ['Not Profitable','Less Than 100%','100-400%','400-700%','700+'],
                 palette=new_pal);
plt.title('Worldwide Gross Revenue by Estimated Production Budget',fontsize='x-large');
plt.xlabel('Estimated Budget in Millions',fontsize='large');
plt.ylabel('Worldwide Gross in Billions',fontsize='large');
ax2.xaxis.get_major_formatter().set_scientific(False);
xlabels = [x for x in ax2.get_xticks()/1000000]
ax2.set_xticklabels(xlabels);
ax2.yaxis.get_major_formatter().set_scientific(False);
ylabels = [y for y in ax2.get_yticks()/1000000]
ax2.set_yticklabels(ylabels);
plt.legend(loc='upper left',title='ROI Category');
```

executed in 327ms, finished 16:10:25 2021-05-06

```
<ipython-input-27-da544b42f0e4>:16: UserWarning: FixedFormatter should only be
used together with FixedLocator
    ax2.set_xticklabels(xlabels);
<ipython-input-27-da544b42f0e4>:19: UserWarning: FixedFormatter should only be
used together with FixedLocator
    ax2.set_yticklabels(ylabels);
```

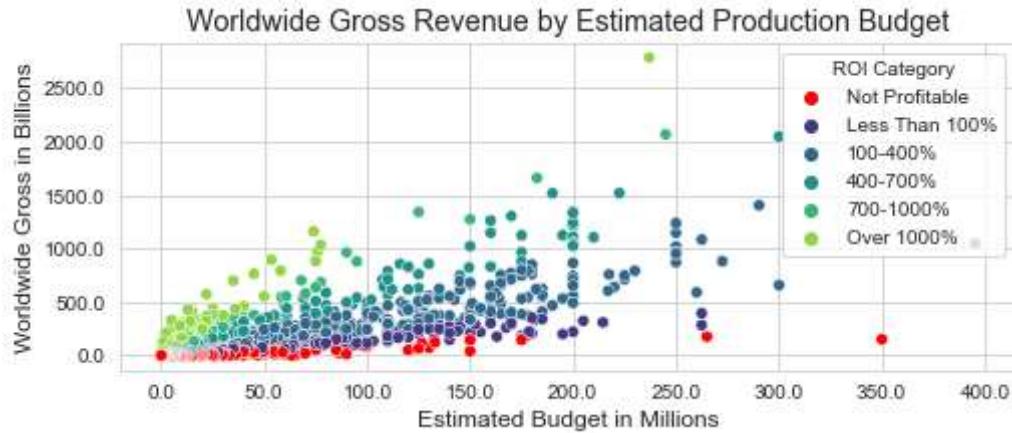
Worldwide Gross Revenue by Estimated Production Budget



```
In [28]: fig11, ax11 = plt.subplots(figsize=(8,3))
sns.scatterplot(x='budget_est',y='worldwide_gross',data=movie_collection,
                 hue='ROI_category',
                 hue_order = ['Not Profitable','Less Than 100%','100-400%','400-700%','700-1000%','Over 1000%'],
                 palette=new_pal);
plt.title('Worldwide Gross Revenue by Estimated Production Budget',fontsize='x-large');
plt.xlabel('Estimated Budget in Millions',fontsize='large');
plt.ylabel('Worldwide Gross in Billions',fontsize='large');
ax11.xaxis.get_major_formatter().set_scientific(False);
xlabels = [x for x in ax11.get_xticks()/1000000]
ax11.set_xticklabels(xlabels);
ax11.yaxis.get_major_formatter().set_scientific(False);
ylabels = [y for y in ax11.get_yticks()/1000000]
ax11.set_yticklabels(ylabels);
plt.legend(loc='upper right',title='ROI Category');
```

executed in 311ms, finished 16:10:25 2021-05-06

```
<ipython-input-28-e840cb44107f>:11: UserWarning: FixedFormatter should only be
used together with FixedLocator
    ax11.set_xticklabels(xlabels);
<ipython-input-28-e840cb44107f>:14: UserWarning: FixedFormatter should only be
used together with FixedLocator
    ax11.set_yticklabels(ylabels);
```

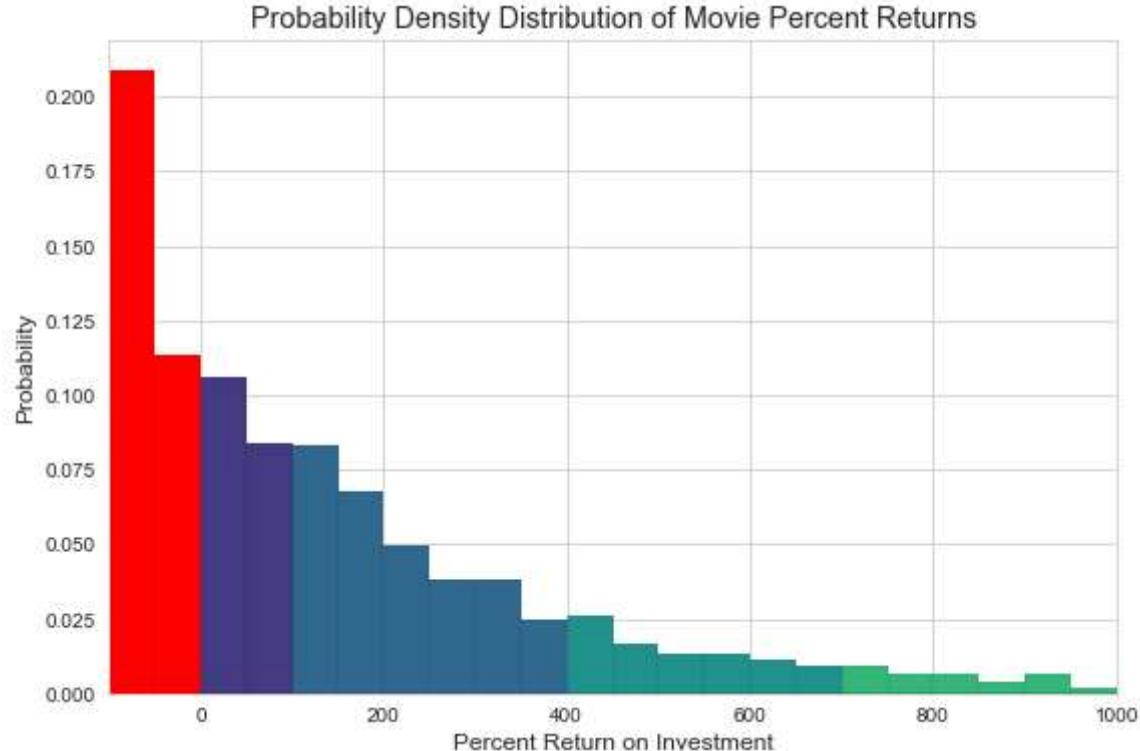


Let's get a better look at the distribution since the points are very overlapping.

```
In [29]: fig3,ax3 = plt.subplots(figsize=(9,6))

sns.histplot(x='ROI_percent',data=movie_collection, ax=ax3, binwidth = 50,
             stat = 'probability', hue='ROI_category',alpha=1,edgecolor='black',
             hue_order = ['Not Profitable','Less Than 100%','100-400%','400-700%'],
             palette=new_pal,legend=False);
plt.xlim((-100,1000))
plt.title('Probability Density Distribution of Movie Percent Returns',fontsize='large')
plt.xlabel('Percent Return on Investment',fontsize='large');
plt.ylabel('Probability',fontsize='large');
```

executed in 5.28s, finished 16:10:31 2021-05-06



I'm seeing that there are a decent number of movies in the collection that did not make money. It may be useful to look at characteristics of these movies, but lets look at the set of profitable movies for now.

```
In [30]: profitable_collection = movie_collection.loc[movie_collection.ROI_percent > 0]
profitable_collection = profitable_collection.reset_index()
profitable_collection = profitable_collection.drop(columns = 'index')
executed in 15ms, finished 16:10:31 2021-05-06
```

```
In [31]: profitable_collection.head()
executed in 29ms, finished 16:10:31 2021-05-06
```

Out[31]:

	tconst	Title	Genres	runtime_minutes	vote_average	vote_count	budget_est	wi
0	tt0892769	How to Train Your Dragon	['Fantasy', 'Adventure', 'Animation', 'Family']	98	8.095082	618909	1650000000.0	
1	tt1228705	Iron Man 2	['Adventure', 'Action', 'Science Fiction']	124	6.996308	670058	1850000000.0	
2	tt1375666	Inception	['Action', 'Science Fiction', 'Adventure']	148	8.794046	1863252	1600000000.0	
3	tt0814255	Percy Jackson & the Olympians: The Lightning T...	['Adventure', 'Fantasy', 'Family']	118	5.904995	169337	950000000.0	
4	tt0435761	Toy Story 3	['Animation', 'Family', 'Comedy']	103	8.292754	690558	2000000000.0	

5 rows × 39 columns

In [32]: `profitable_collection.info()`

executed in 14ms, finished 16:10:31 2021-05-06

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1605 entries, 0 to 1604
Data columns (total 39 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   tconst          1605 non-null    object  
 1   Title            1605 non-null    object  
 2   Genres           1605 non-null    object  
 3   runtime_minutes  1605 non-null    int64  
 4   vote_average     1605 non-null    float64 
 5   vote_count       1605 non-null    int64  
 6   budget_est       1605 non-null    float64 
 7   worldwide_gross  1605 non-null    float64 
 8   release_week_day 1605 non-null    int64  
 9   release_day      1605 non-null    int64  
 10  release_month    1605 non-null    int64  
 11  release_year     1605 non-null    int64  
 12  ROI_percent      1605 non-null    float64 
 13  Mystery          1605 non-null    float64 
 14  Fantasy          1605 non-null    float64 
 15  Family            1605 non-null    float64 
 16  Drama             1605 non-null    float64 
 17  Animation         1605 non-null    float64 
 18  Musical           1605 non-null    float64 
 19  Sci-Fi            1605 non-null    float64 
 20  Sport              1605 non-null    float64 
 21  History           1605 non-null    float64 
 22  Documentary        1605 non-null    float64 
 23  TV Movie          1605 non-null    float64 
 24  Western            1605 non-null    float64 
 25  Adventure          1605 non-null    float64 
 26  Comedy             1605 non-null    float64 
 27  Thriller           1605 non-null    float64 
 28  Science Fiction   1605 non-null    float64 
 29  War                1605 non-null    float64 
 30  Action              1605 non-null    float64 
 31  Biography          1605 non-null    float64 
 32  Romance            1605 non-null    float64 
 33  Crime              1605 non-null    float64 
 34  Music              1605 non-null    float64 
 35  Horror              1605 non-null    float64 
 36  primary_genre      1605 non-null    object  
 37  budget_category    1605 non-null    object  
 38  ROI_category       1605 non-null    object  
dtypes: float64(27), int64(6), object(6)
memory usage: 489.1+ KB
```

In [33]: `profitable_collection.groupby('ROI_category').count()`

executed in 15ms, finished 16:10:32 2021-05-06

Out[33]:

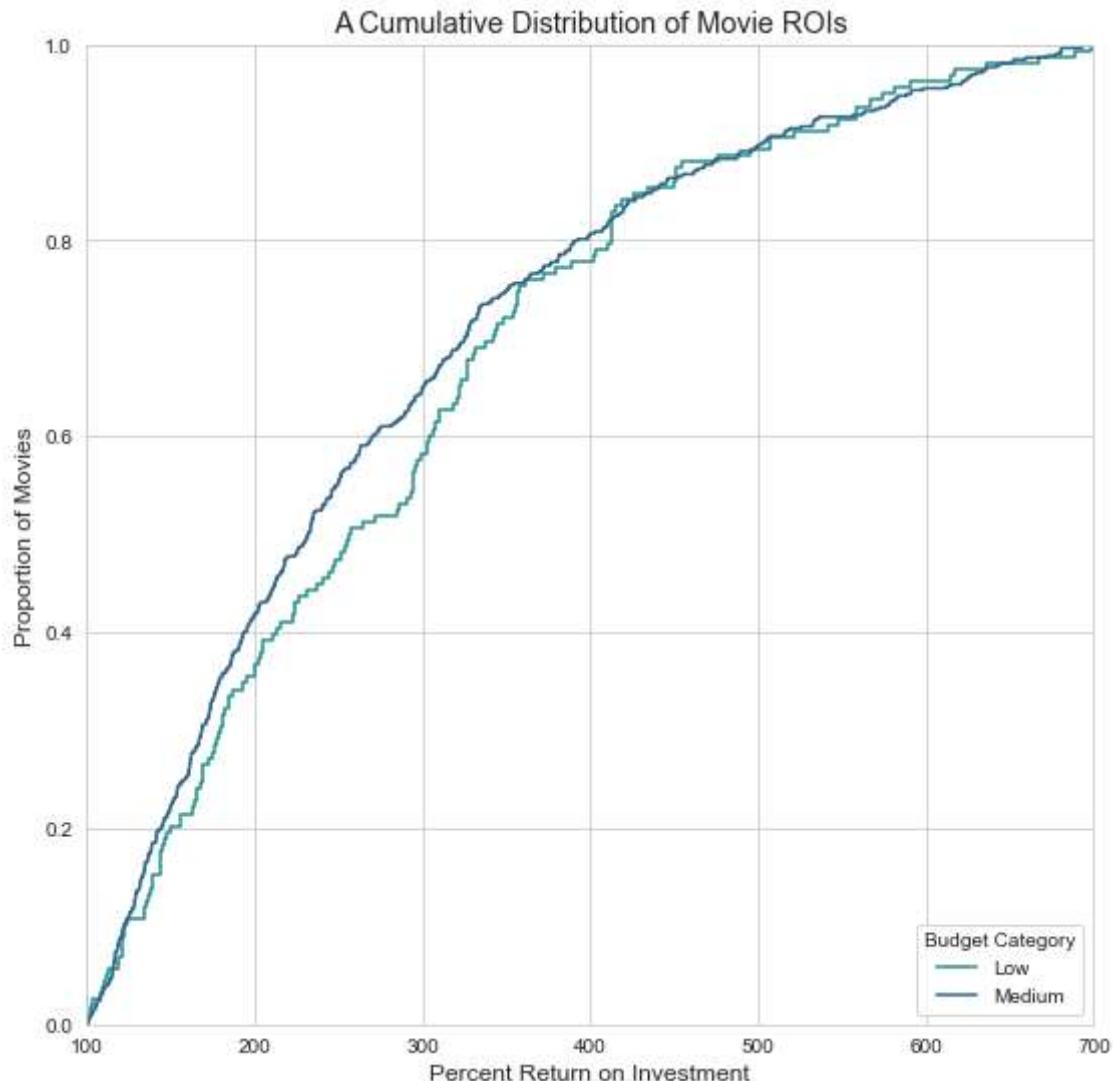
	tconst	Title	Genres	runtime_minutes	vote_average	vote_count	budget_est	world
ROI_category								
100-400%	719	719	719		719	719	719	719
400-700%	215	215	215		215	215	215	215
700-1000%	82	82	82		82	82	82	82
Less Than 100%	449	449	449		449	449	449	449
Over 1000%	140	140	140		140	140	140	140

5 rows × 38 columns

So, based on the above plots, there are a lot of movies that didn't do well, and it tails toward the successful end. From here onwards, I have decided to throw out the highest and lowest budget categories. The first, because it is a small sample size, and because Microsoft probably doesn't want to jump in with blockbuster budgets. The latter, because, while it is tempting seeing all the movies listed there with 1000% plus returns, there are equally as many that didn't make any money and many that lost much. Second, a movie in the lowest budget category needs to make ridiculous returns to come close to a higher budget movie making more standard returns. So, since we are interested in making movies in which he have reasonable confidence will be profitable, I want to look at the subset of the data that is in the low and medium budget range, and also the low and medium ROI range, since this is the biggest sample size of confidently successful movies we can achieve. Just to be safe, let's see if there is any difference in the distribution of ROI_percent for these two middle budget categories.

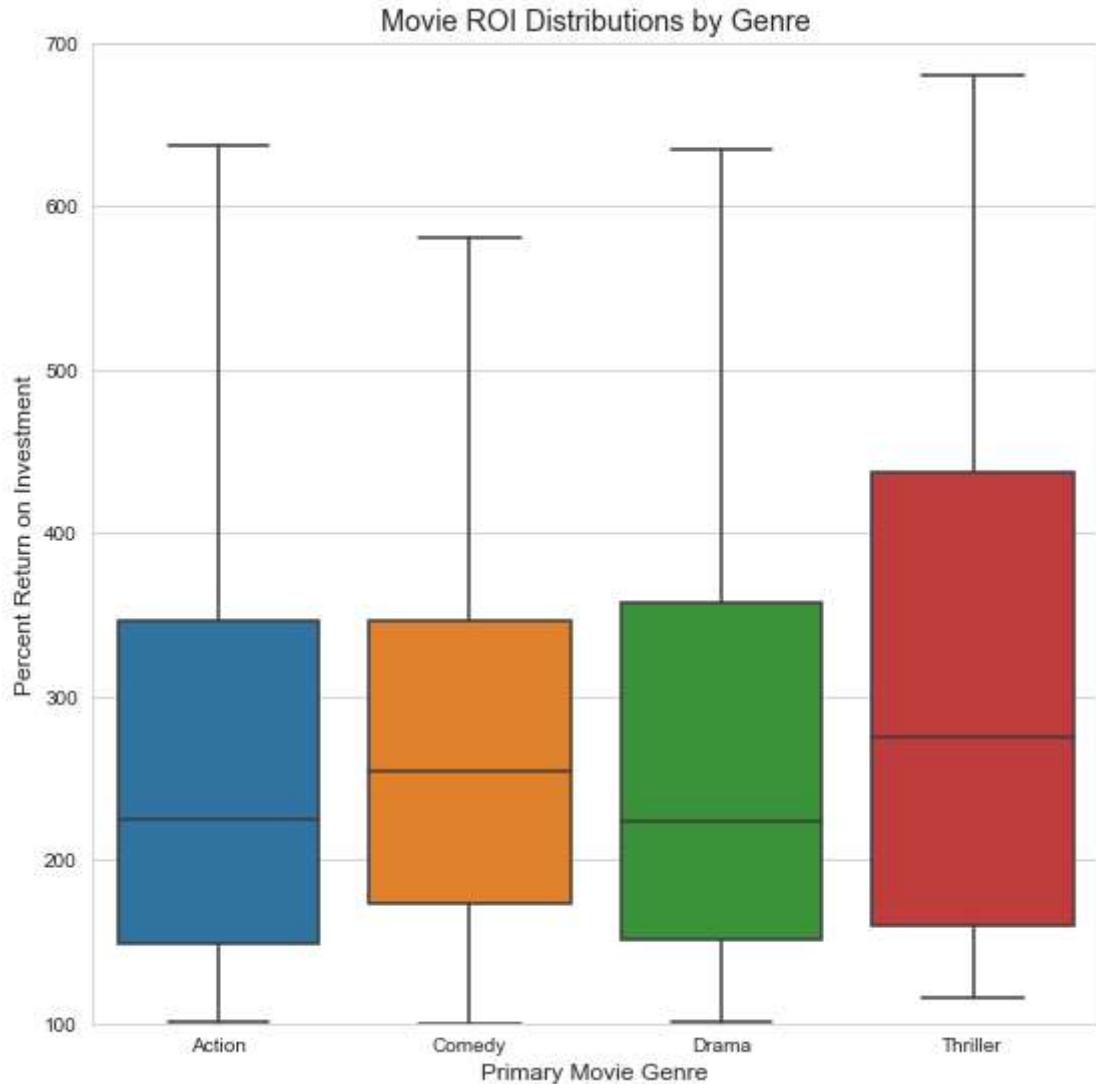
```
In [34]: fig4, ax4 = plt.subplots(figsize=(9,9))
sns.ecdfplot(data=profitable_collection.loc[((profitable_collection.budget_category == 'Low') & (profitable_collection.ROI_category == 'High')) | ((profitable_collection.budget_category == 'Medium') & (profitable_collection.ROI_category == 'High'))], x='ROI_percent',hue = 'budget_category',legend=False, hue_order = ['Low','Medium'],palette=new_pal[2:4])
ax4.set_xlim(100,700);
plt.legend(title='Budget Category',loc = 'lower right',labels = ['Low', 'Medium'])
plt.title('A Cumulative Distribution of Movie ROIs',fontsize='x-large');
plt.xlabel('Percent Return on Investment',fontsize='large');
plt.ylabel('Proportion of Movies',fontsize='large');
```

executed in 189ms, finished 16:10:32 2021-05-06



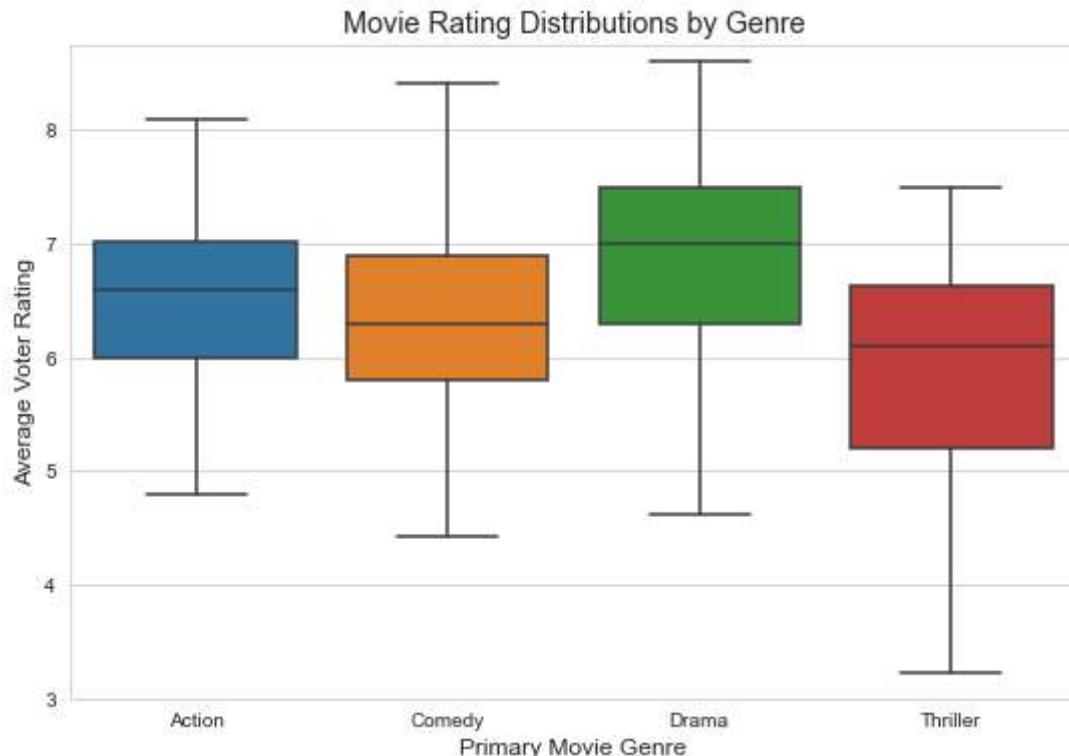
They are basically the same. So, all plots moving forward will be from the budget categories 'Medium' and 'Low' and the ROI_percent categories '100-400%' and '400-700%'. Let's start looking at the other categorical variables.

```
In [35]: fig5, ax5 = plt.subplots(figsize=(9,9))
sns.boxplot(data=profitable_collection.loc[((profitable_collection.budget_category == 'High') & (profitable_collection.ROI_category == 'High')) | ((profitable_collection.budget_category == 'Medium') & (profitable_collection.ROI_category == 'Medium')) | ((profitable_collection.budget_category == 'Low') & (profitable_collection.ROI_category == 'Low'))], x='primary_genre',y='ROI_percent',palette='tab10',showfliers=False, order = ['Action','Comedy','Drama','Thriller'])
plt.title('Movie ROI Distributions by Genre',fontsize='x-large');
plt.ylabel('Percent Return on Investment',fontsize='large');
plt.xlabel('Primary Movie Genre',fontsize='large');
plt.ylim(100,700);
executed in 141ms, finished 16:10:32 2021-05-06
```



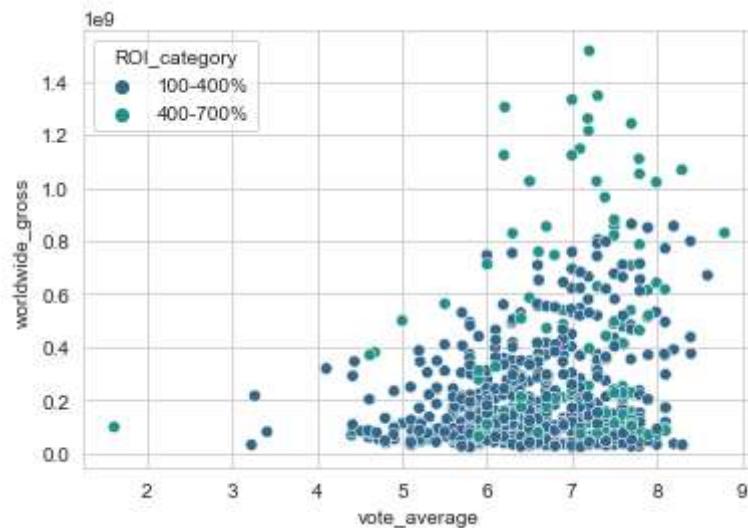
Genre seems to matter slightly, could make a recommendation here.

```
In [36]: fig6, ax6 = plt.subplots(figsize=(9,6))
sns.boxplot(data=profitable_collection.loc[((profitable_collection.budget_category == 'High') & (profitable_collection.ROI_category == 'High')) | ((profitable_collection.budget_category == 'Medium') & (profitable_collection.ROI_category == 'Medium')) | ((profitable_collection.budget_category == 'Low') & (profitable_collection.ROI_category == 'Low'))], x='primary_genre',y='vote_average',palette='tab10',showfliers=False, order = ['Action','Comedy','Drama','Thriller'])
plt.title('Movie Rating Distributions by Genre',fontsize='x-large');
plt.ylabel('Average Voter Rating',fontsize='large');
plt.xlabel('Primary Movie Genre',fontsize='large');
plt.ylim(3,8.75);
executed in 142ms, finished 16:10:33 2021-05-06
```



```
In [37]: sns.scatterplot(data=profitable_collection.loc[((profitable_collection.budget_cat >= 100) & (profitable_collection.budget_category <= 700)) | ((profitable_collection.ROI_category >= 100) & (profitable_collection.ROI_category <= 700))],  
                      x='vote_average',y='worldwide_gross',  
                      hue='ROI_category',legend=True,  
                      hue_order = ['100-400%', '400-700%'],  
                      palette=new_pal[2:4]);
```

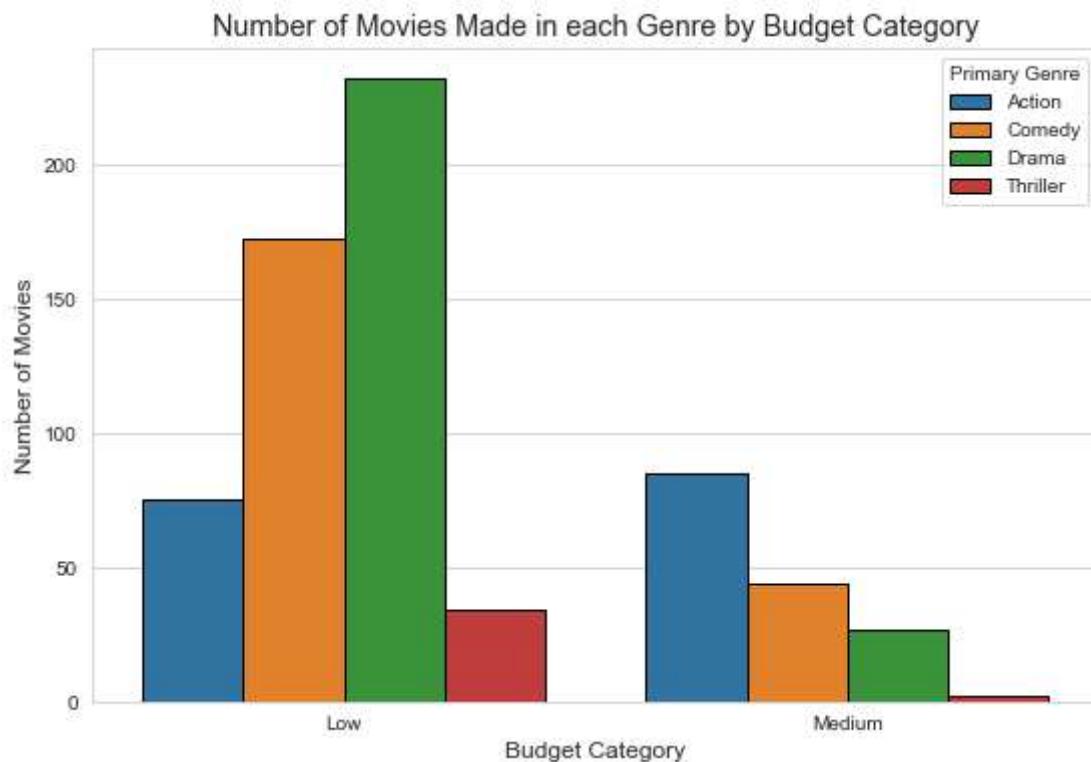
executed in 218ms, finished 16:10:33 2021-05-06



Excellent, doesn't even need to be a good movie. Higher rated movies do slightly better (maybe), but we're already looking at profitable categories here.

```
In [38]: fig7, ax7 = plt.subplots(figsize=(9,6))
sns.countplot(data=profitable_collection.loc[((profitable_collection.budget_category == 'Low') & (profitable_collection.ROI_category == 'High')) | ((profitable_collection.budget_category == 'Medium') & (profitable_collection.ROI_category == 'High')) | ((profitable_collection.budget_category == 'High') & (profitable_collection.ROI_category == 'High'))], x='budget_category', hue='primary_genre', hue_order = ['Action', 'Comedy', 'Drama', 'Thriller'], order = ['Low', 'Medium'], edgecolor='black')
plt.title('Number of Movies Made in each Genre by Budget Category', fontsize='x-large');
plt.xlabel('Budget Category', fontsize='large');
plt.ylabel('Number of Movies', fontsize='large');
plt.legend(title='Primary Genre');
```

executed in 158ms, finished 16:10:33 2021-05-06

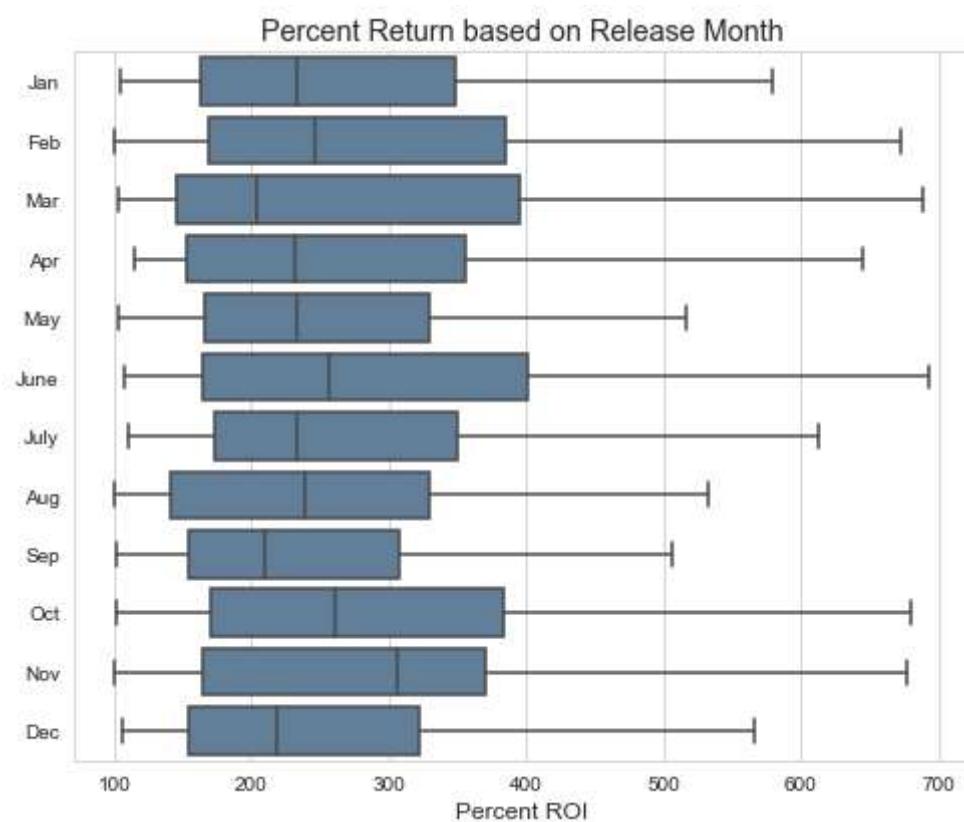


And there aren't that many of them being produced! I noticed in the scatter matrices that there could be something to release day, month, year, etc. Lets look at release window.

Okay, release day I suppose is Monday!

```
In [39]: fig8, ax8 = plt.subplots(figsize=(8,6.5))
sns.boxplot(data=profitable_collection.loc[((profitable_collection.budget_category == 'High') & (profitable_collection.ROI_category == 'High')) | ((profitable_collection.budget_category == 'Medium') & (profitable_collection.ROI_category == 'Medium')) | ((profitable_collection.budget_category == 'Low') & (profitable_collection.ROI_category == 'Low'))], y='release_month',x='ROI_percent',showfliers = False, color='steelblue',saturation = .5,orient='h');
plt.title('Percent Return based on Release Month',fontsize='x-large');
plt.ylabel('');
plt.yticks(ticks = [0,1,2,3,4,5,6,7,8,9,10,11], labels=['Jan','Feb','Mar','Apr','May','June','July','Aug','Sep','Oct','Nov','Dec']);
plt.xlabel('Percent ROI',fontsize='large');
```

executed in 235ms, finished 16:10:34 2021-05-06

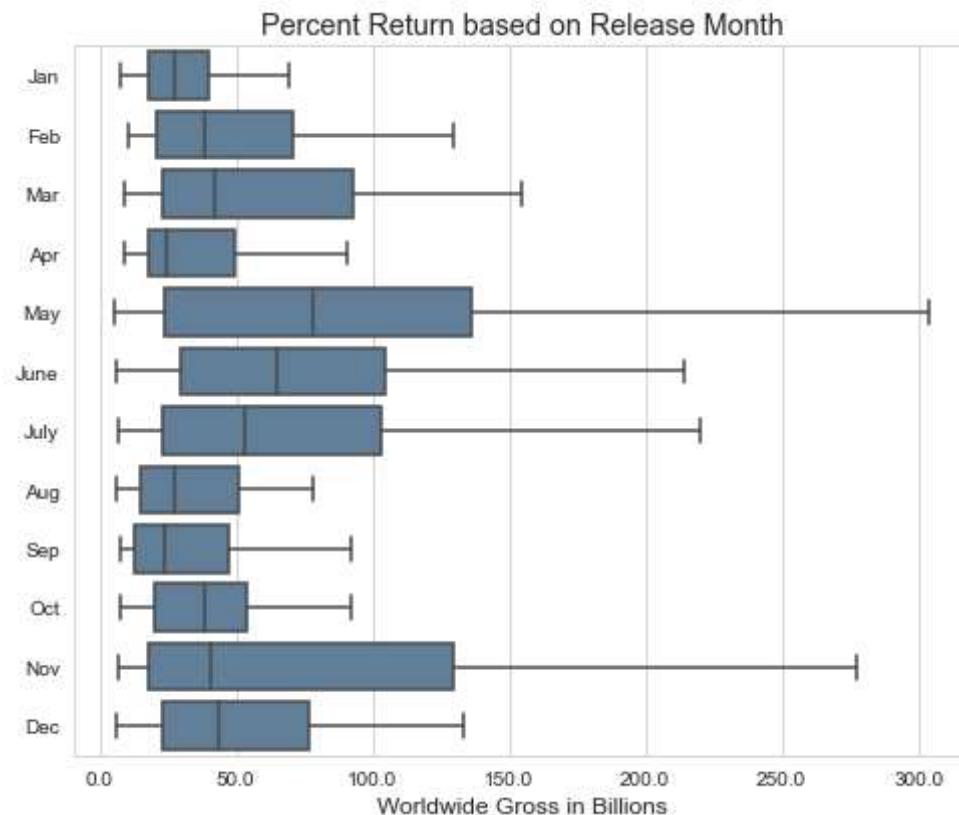


```
In [40]: fig9,ax9 = plt.subplots(figsize = (8,6.5))

sns.boxplot(data=profitable_collection.loc[((profitable_collection.budget_category == 'High') & ((profitable_collection.ROI_category == 'High') | (profitable_collection.ROI_category == 'Medium'))),y='release_month',x='worldwide_gross',showfliers = False,
             color='steelblue',saturation = .5,orient='h');
plt.title('Percent Return based on Release Month',fontsize='x-large');
plt.ylabel('');
plt.yticks(ticks = [0,1,2,3,4,5,6,7,8,9,10,11],
           labels=['Jan','Feb','Mar','Apr','May','June','July','Aug','Sep','Oct','Nov','Dec']);
plt.xlabel('Worldwide Gross in Billions',fontsize='large');
ax9.xaxis.get_major_formatter().set_scientific(False);
xlabels = [x for x in ax2.get_xticks()/1000000]
ax9.set_xticklabels(xlabels);
```

executed in 236ms, finished 16:10:34 2021-05-06

```
<ipython-input-40-1e649c5a952b>:16: UserWarning: FixedFormatter should only be
used together with FixedLocator
    ax9.set_xticklabels(xlabels);
```



This one I think is worth showing both, to highlight the peaks in release windows in terms of release month.

```
In [41]: fig10, ax10 = plt.subplots(figsize=(11,7))
sns.boxplot(data=profitable_collection.loc[((profitable_collection.budget_category == 'High') & ((profitable_collection.ROI_category == 'Low') | (profitable_collection.ROI_category == 'Medium'))), y='release_week_day',x='ROI_percent',showfliers = False, color='mediumpurple',saturation = .5,orient='h');
plt.title('Percent Return based on Release Week Day',fontsize='x-large');
plt.ylabel('');
plt.yticks(ticks = [0,1,2,3,4,5,6], labels=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']);
plt.xlabel('Percent ROI',fontsize='large');
plt.xlim(100,700);
```

executed in 187ms, finished 16:10:35 2021-05-06

