Udacity Data Analyst Nanodegree

02 Examine the TMDB Movie Database

DAVID LASSIG

2019-01-11

Contents

1 Project: Examine The TMDB Movie Database

1.1 Introduction

I've choosen the TMDB dataset as I'm a passionated movie watcher and in the past I had doubts whether the voting results of movie database services are representative for the real quality of some movies. For example it occurs that bombastic Blockbuster movies are rated much higher on a user driven platform IMBD then on a professional critic driven platform like Rotten Tomatoes. I've choosen these questions for further analysis:

- **Question 1:** Does movies have better ratings if they're voted more often?
- **Question 2:** Is there a correlation between specific actors and the associated movie rating or associated revenue?

Before doing any analysis I will import needed libraries and setting styles:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import json
plt.style.use("seaborn-dark")

%matplotlib inline
```

1.2 Data Wrangling

To answer my questions I have to look at two different datasources. **tmdb_5000_movies.csv** contains the movie title and all movie related values. Additionally I have to look at **tmdb_5000_credits.csv** which includes the complete cast and crew for every movie in the first source. As it includes the complete cast and crew it's size is much bigger than the first one.

1.2.1 General Properties

Movie source

```
movie_df = pd.read_csv('tmdb_5000_movies.csv')
movie_df.head(2)
```

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	production_countries	release_date	revenue	runtime	spoken_languages	status	tagline	title	vote_average	vote_count
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Ingenious Film Partners", "id": 289...	[{"iso_3166_1": "US", "name": "United States o...	2009-12-10	2787965087	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released	Enter the World of Pandora.	Avatar	7.2	11800
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt Disney Pictures", "id": 2}, {"...	[{"iso_3166_1": "US", "name": "United States o...	2007-05-19	961000000	169.0	[{"iso_639_1": "en", "name": "English"}]	Released	At the end of the world, the adventure begins.	Pirates of the Caribbean: At World's End	6.9	4500

Figure 1: movie_df.head(2)

```
movie_df.info()
```

```
Output:


    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 4803 entries, 0 to 4802
    Data columns (total 20 columns):
    budget                  4803 non-null int64
    genres                  4803 non-null object
    homepage                1712 non-null object
```

```
id                      4803 non-null int64
keywords                4803 non-null object
original_language       4803 non-null object
original_title          4803 non-null object
overview                4800 non-null object
popularity              4803 non-null float64
production_companies    4803 non-null object
production_countries    4803 non-null object
release_date            4802 non-null object
revenue                 4803 non-null int64
runtime                 4801 non-null float64
spoken_languages        4803 non-null object
status                  4803 non-null object
tagline                 3959 non-null object
title                   4803 non-null object
vote_average            4803 non-null float64
vote_count              4803 non-null int64
dtypes: float64(3), int64(4), object(13)
memory usage: 750.5+ KB
```

We can see many columns that have rows with NaN values. Regarding my research questions we can drop many columns and this will include every column that contains NaN values.

```
print(type(movie_df['genres'][0]))
```

```
Output:

    <class 'str'>
```

```
movie_df['genres'][0]
```

```
Output:

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

There are multiple fields with lists of dicts to describe multiple properties. There are different ways of solving it, like creating columns for every property if it's important for the own examination. I will create functions that can be applied if needed to extract the information on call.

Credits source

```
credits_df = pd.read_csv('tmdb_5000_credits.csv')
credits_df.head()
```

	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...

Figure 2: credits_df.head()

```
credits_df.info()
```

```
Output:

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 4803 entries, 0 to 4802
    Data columns (total 4 columns):
    movie_id    4803 non-null int64
    title       4803 non-null object
    cast        4803 non-null object
    crew        4803 non-null object
    dtypes: int64(1), object(3)
    memory usage: 150.2+ KB
```

This source has less columns but much more content in every row. I will cut it down to the desired data only.

1.2.2 Cleaning movie_df

```
movie_df.drop([ 'popularity',
        'overview',
        'tagline',
        'runtime',
        'keywords',
        'genres',
        'budget',
        'homepage',
        'original_language',
        'production_countries',
        'production_companies',
        'release_date',
        'status',
        'title',
        'spoken_languages',], axis=1,inplace=True)
movie_df.head()
```

	id	original_title	revenue	vote_average	vote_count
0	19995	Avatar	2787965087	7.2	11800
1	285	Pirates of the Caribbean: At World's End	961000000	6.9	4500
2	206647	Spectre	880674609	6.3	4466
3	49026	The Dark Knight Rises	1084939099	7.6	9106
4	49529	John Carter	284139100	6.1	2124

Figure 3: movie_df.head()

1.2.3 Cleaning credits_df

The file for this csv is much bigger although it has the same number of columns. The columns **cast** and **crew** are containing much more content. They include the complete cast and crew that have created a specific movie. As one question is regarding the cast I will extract only the top five of each movie and dropping the rest.

```
def extract_top_five_cast(val):
    mov_cast = json.loads(val)
    ex_cast = []
    for element in mov_cast[:5]:
        cast_id = element['id']
        name = element['name']
        ex_cast.append((cast_id,name))
    return ex_cast
```

```
credits_df['top_five'] = credits_df['cast'].apply(extract_top_five_cast)
cast_df = credits_df.drop(['title','cast','crew'],axis=1)
```

```
cast_df.rename(columns={'movie_id':'id'},inplace=True)
```

```
cast_df.head(1)
```

	id	top_five
0	19995	[(65731, Sam Worthington), (8691, Zoe Saldana)...

Figure 4: cast_df.head(1)

Now it's easy to merge or reference the movie dataframe with the cast dataframe by using the **id** key. Moreover I will extract all possible actor IDs for beeing able later on to query all existing actors. I will use a global variable inside the lambda function for collection all occurences of actors and the frequency:

```
actor_name = {}
actor_freq = {}


def extract_all_ids(row,s=actor_name,m=actor_freq):
    for i in range(0,5):
        try:
            temp_id = row[i][0]
            name = row[i][1]
            if temp_id not in s.keys():
                m[temp_id] = 0
                s[temp_id] = name
            else:
                m[temp_id] = m[temp_id] + 1
        except:
            pass
```

```
temp = cast_df['top_five'].apply(extract_all_ids)
```

Now I will create a Dataframe from the data I extracted. I pushed the data into dicts in the first place as the speed of execution was much higher.

```
actor_df = pd.DataFrame.from_dict(list(actor_name.items()))
```

```
actor_df['freq'] = actor_df[0].map(actor_freq)
```

```
actor_df = actor_df.rename(columns={0:'actor_id',1:'name'})
```

```
actor_df.head(1)
```

	actor_id	name	freq
0	65731	Sam Worthington	7

Figure 5: actor_df.head(1)

```
len(actor_df)
```

```
Output:

    9394
```

These are to many actors to examine. Especially actors with only a few or a single occurence in a movie could distort the statistical result in the end. Therefore I will extract only the actors with twenty or more occurences in movies.

```
actor_o20_df = actor_df[actor_df['freq'] >= 20]
actor_o20_df.info()
```

```
Output:

    <class 'pandas.core.frame.DataFrame'>
    Int64Index: 100 entries, 2 to 2128
    Data columns (total 3 columns):
    actor_id    100 non-null int64
    name        100 non-null object
    freq        100 non-null int64
    dtypes: int64(2), object(1)
    memory usage: 3.1+ KB
```

It's like a magic coincidence that the resulting Dataframe has exactly 100 rows. Sounds like a good result :)

1.3 Exploratory Data Analysis

1.3.1 Research Question 1: Does movies have better ratings if they're voted more often?

For comparing the **vote_count** with the **average_rating** it's necessary to normalize one of this values. I will choose the vote count to normalize as the rating has a defined range between zero and ten:

```
temp = 10*(movie_df['vote_count'] - movie_df['vote_count'].min())
movie_df['vote_count_normalized'] = temp / (movie_df['vote_count'].max() - movie_df['vote_count'].min())
```

For getting a more human friendly view into my visualisation I will order the resulting values by one of the examined data series. It will occur, that the least voted movies will have a normalized count equal zero.

```
movie_df_sc = movie_df.sort_values('vote_count')
```

```
movie_df_sc['vote_count_normalized'].head()
```

```
Output:

    4307    0.0
    4140    0.0
    4638    0.0
    4118    0.0
    4068    0.0
    Name: vote_count_normalized, dtype: float64
```

First Plot: Bar Plot with small data extract

I will use only a few values for the bar plot as it occured that adding more values will take very long to compute and the illustration will get worse:

```
# Use this, and more code cells, to explore your data. Don't forget to add
#   Markdown cells to document your observations and findings
fig = plt.figure(figsize=(10,10))

plt.bar(range(1,len(movie_df_sc['vote_average'][-200::])+1),movie_df_sc['vote_average'][-200::])
plt.bar( range(1,len(movie_df_sc['vote_average'][-200::])+1),movie_df_sc['vote_count_normalized'][-200::],alpha=0.7)
plt.title("Correlation between Count of Votings and the associated rating for 200 movies with highest vote count")
plt.xlabel("Index (Sorted by Vote Count)")
plt.ylabel("Rating between 0-10 and Vote Count normalized to same space")

plt.gca().legend(('Vote Average','Normalized Vote Count'))
```

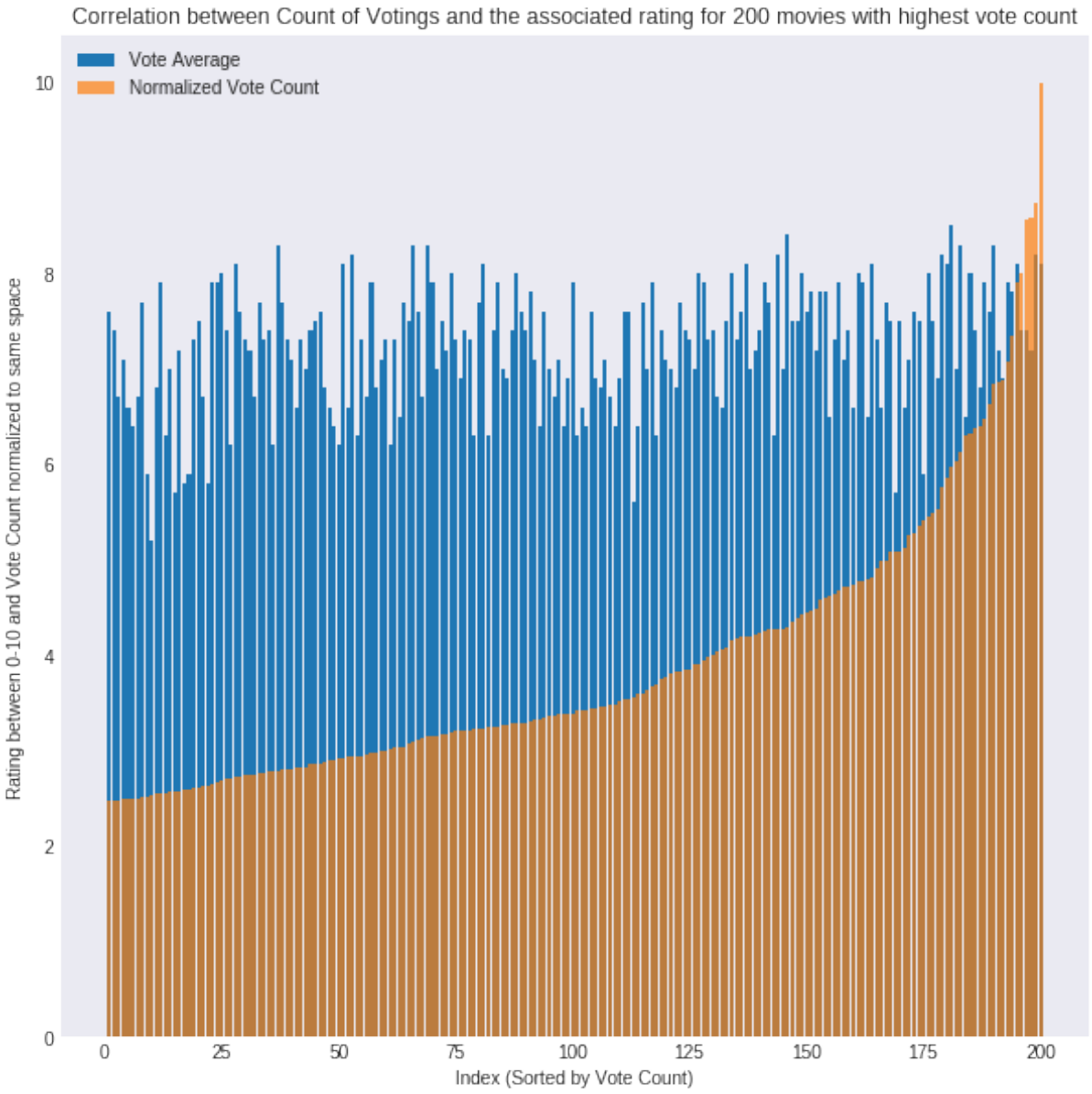

Figure 6: png

I cannot see any relationship between the two plotted series at all. I will calculate the correlation for emphasize or contradict this impression:

```
corr = movie_df_sc['vote_average'].corr(movie_df_sc['vote_count_normalized'])
corr
```

```
Output:

    0.31299740399576015
```

Second Plot: Default plot with complete data

This means there should be some kind of relationship. I will continue with a default plot and all values:

```
fig = plt.figure(figsize=(10,10))

plt.plot(range(1,len(movie_df_sc)+1),movie_df_sc['vote_average'],'.')
plt.plot(range(1,len(movie_df_sc)+1),movie_df_sc['vote_count_normalized'],'.')
plt.title("Correlation between Count of Votings and the associated rating")
plt.xlabel("Index (Sorted by Vote Count)")
plt.ylabel("Rating between 0-10 and Vote Count normalized to same space")

plt.gca().legend(('Vote Average','Normalized Vote Count'))
```

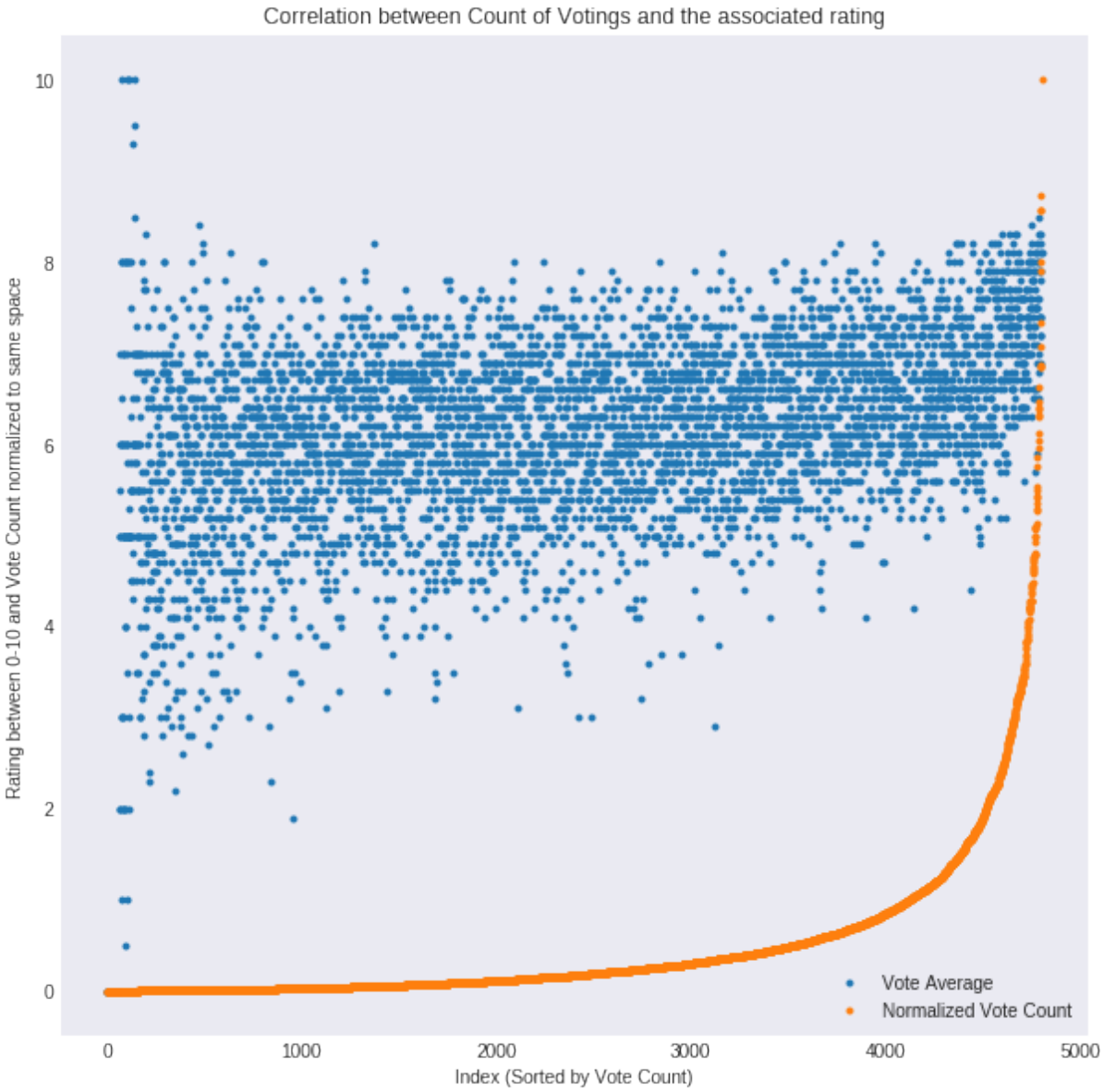

Figure 7: png

I would say there is some kind of correlation in this visualization. For improving this I will calculate the moving average for the **vote_average**

Third Plot: Default Plot with complete data and moving average

```
movie_df_sc['vote_average_rm']=movie_df_sc['vote_average'].rolling(window=10,min_periods=1).mean()
```

```
fig = plt.figure(figsize=(10,10))

plt.plot(range(1,len(movie_df_sc)+1),movie_df_sc['vote_average_rm'],'.')
plt.plot(range(1,len(movie_df_sc)+1),movie_df_sc['vote_count_normalized'],'.')
plt.title("Correlation between Count of Votings and the associated averaged rating")
plt.xlabel("Index (Sorted by Vote Count)")
plt.ylabel("Rating between 0-10 and Vote Count normalized to same space")

plt.grid(b=None, which='major', axis='both')

plt.gca().legend(('Vote Average with moving average over 10 values','Normalized Vote Count'))
```

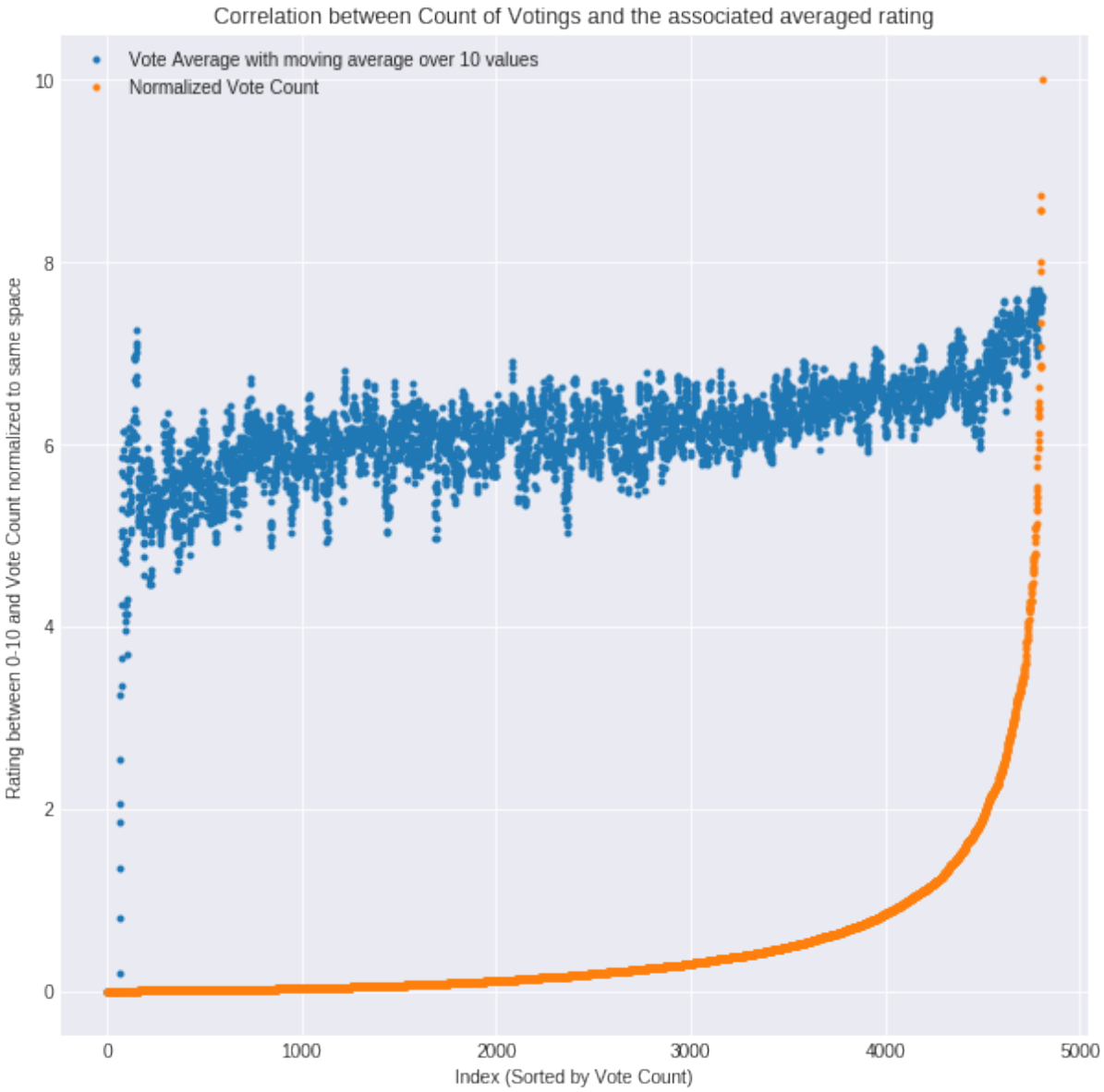

Figure 8: png

With data ordered by the count of votings, a **normalized vote count** and a moving average over the **vote_average** gives a clear positive correlation between the count of votings and the rating. Especially the top 300 movies with a **vote_count** above 2700 show a significant positive correlation.

1.3.2 Research Question 2: Is there a correlation between specific actors and the associated movie rating or associated revenue?

For achieving this answer I will collect all movie occurences for the top 100 actors I extracted before:

Extract all existing IDs

I will extract all movie occurences by selecting them from the previously created **cast_df** with a lambda function that will return **True** if a actor_id exists in the tuple of the "top_five" actors of a movie. This selection will be pushed onto a list of this selections **actor_frames** which will be processed afterwards:

```
type(moviecast_df['top_five'][0][0][0])
```

```
Output:

    int
```

```
def process_actor(df_row, actor_id):
    top_five = []
    for tupl in df_row['top_five']:
        top_five.append(tupl[0])

    if actor_id in top_five:
        return True
    else:
        return False
```

```
actor_frames = []

for actor_id in actor_o20_df['actor_id']:
    actor_frames.append(cast_df[cast_df.apply(process_actor,axis=1,actor_id=actor_id)]['id'])
```

```
actor_frames[0]
```

```
Output:

     0       19995
     562      6947
     740      8078
     838      8077
     1053      926
     1178     7461
     1309    10833
     1574     8780
     1804     9092
     2103     8326
     2138     1710
     2153    38303
     2244    77948
     2361    68924
     2391    75638
     2403      679
     2500    73935
     2778     9672
     3056    52067
     3105    25350
     3158      348
     4703    39141
     Name: id, dtype: int64
```

The previous step is very costly in terms of needed resources. Actually I'm hosting my Jupyter Notebook on a Raspberry Pi. But this calculation convinced me to change to my personal computer. I would like to know if there are possible measures to reduce this computational cost?

Now I will calculate the revenues and ratings over all occurences of a single actor by looping through the **actor_frames**. Important is to divide the sum of all values by the frequency of occurence of the actor:

```
actor_result_df = pd.DataFrame(columns=['name','average_rating','average_revenue'])

for i in range(0,len(actor_frames)):
        freq = actor_o20_df.iloc[i]['freq']
        all_ratings = 0
        all_revenues = 0
        for j in range(0,len(actor_frames[i])):
            all_ratings = all_ratings + float(movie_df[movie_df['id'] == actor_frames[i].iloc[j]]['vote_average'])
            all_revenues = all_revenues + float(movie_df[movie_df['id'] == actor_frames[i].iloc[j]]['revenue'])

        all_ratings = all_ratings / freq
        all_revenues = all_revenues / freq
```

```
        actor_result_df = actor_result_df.append({'name':actor_o20_df.iloc[i]['name'],
                          'average_rating':all_ratings,
                          'average_revenue':all_revenues},ignore_index=True)
```

```
actor_result_df.head(1)
```

	name	average_rating	average_revenue
0	Sigourney Weaver	6.542857	1.924631e+08

Figure 9: actor_result_df.head(1)

```
actor_result_df.sort_values(by='average_rating', ascending=False).iloc[0:3]
```

Figure 10: actor_result_df.sort_values(by="average_rating", ascending=False).iloc[0:3]

Top three actors with highest revenue

```
actor_result_df.sort_values(by='average_revenue', ascending=False).iloc[0:3]
```

	name	average_rating	average_revenue
16	Will Smith	6.760000	3.723642e+08
10	Robert Downey Jr.	6.756522	3.571872e+08
12	Scarlett Johansson	6.860000	3.063467e+08

Figure 11: actor_result_df.sort_values(by="average_revenue", ascending=False).iloc[0:3]

Visualisation of distribuation over all examined actors

Now we want to plot these two values next to each other for getting quick insights about the correlation between the **revenue** and the **ratings**. As the revenue is much higher than the ratings value, I have to normalize again. I will use the base 10 of the rating again:

```
temp = 10*(actor_result_df['average_revenue'] - actor_result_df['average_revenue'].min())
actor_result_df['revenue_normalized'] = temp  / (actor_result_df['average_revenue'].max() - actor_result_df['average_revenue'].min())
```

I would like to print multiple bars for one x-tick and add a label with the actors name on top of each bar. After some research, I found a good example here Multiple Bars. I modified the function and the plot for my purposes:

```
def autolabel(rects1,rects2,names):
    counter =0
    for rect1,rect2 in zip(rects1,rects2):
        h1 = rect1.get_height()
        h2 = rect2.get_height()
        if h2 > h1:
            h = h2
        else:
            h = h1
        ax.text(rect1.get_x()+rect1.get_width(), 1.05*h, '%s'%names[counter],
                ha='center', va='bottom', rotation='vertical',)
        counter = counter + 1
```

```
import numpy

ind = numpy.arange(50)

fig = plt.figure(figsize=(25,10))
width = 0.3

ax = fig.add_subplot(111)

yvals = actor_result_df['revenue_normalized'][0:50]
rects1 = ax.bar(ind,yvals,width)

zvals = actor_result_df['average_rating'][0:50]
rects2 = ax.bar(ind+width, zvals, width)

plt.tick_params(axis='x', which='both', bottom=False, top=False,
    labelbottom=False)

plt.title("Plotting normalized revenue against averaged ratings of top hundred most frequent actors")
plt.ylabel("Rating between 0-10 and Revenue normalized to same space")

plt.gca().legend(('Average Revenue Normalized','Average Rating'))

plt.ylim(0,12)


autolabel(rects1,rects2,actor_result_df['name'])
```

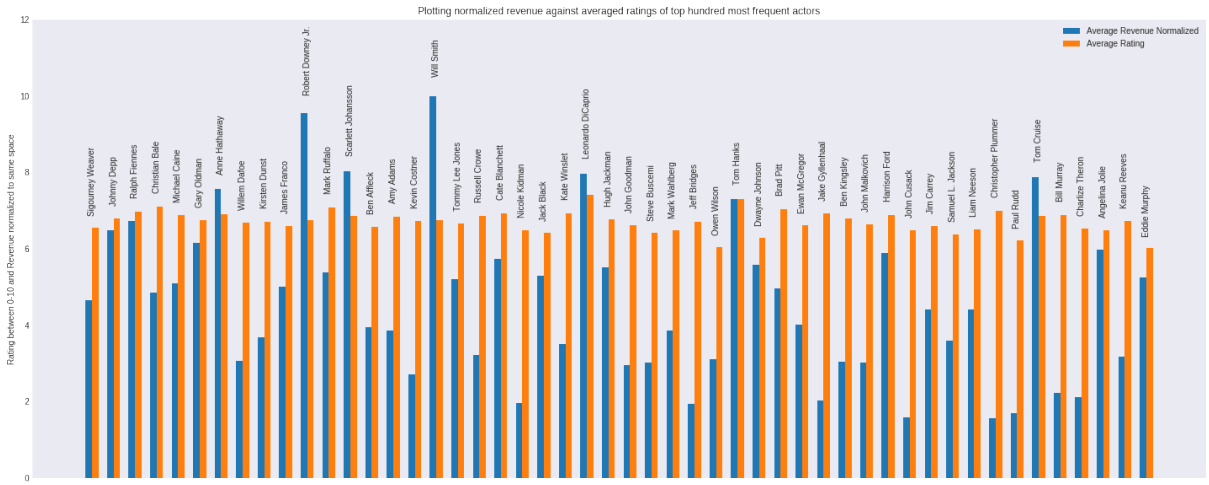

Figure 12: png

```
actor_result_df['average_rating'].corr(actor_result_df['revenue_normalized'])
```

```
Output:

    0.18375222363390781
```

My plot contains only first half of the examined actors for better visibility. We can expose the following results:

- The rating has a uniformly distribution over all actors
- The average revenue is very uneven distributed, there are actors with very high revenues and actors with very low ones
- The rating is barely correlated to the revenue. The very small positive correlation value emphasizes this impression.

1.4 Conclusions

After answering my research questions in full detail I got a much better insight into movie ratings associated to revenues and the involved actors.

My first research question did show a positive correlation between the count of votings and the rating for a movie. This could mean that a movie that gets more votings will get better ratins at all.

The second research question did highlight that there isn't barely a correlation between the rating and the revenue of a movie. Actually, it just figures for me but it's nice to emphasize this impression in this examination.