In [1]:

```
from pykalman import KalmanFilter
import numpy as np
import pandas as pd
import sys
import matplotlib
import matplotlib.pyplot as plt
from skimage.color import lab2rgb
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
import skimage
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from functools import reduce
import statsmodels.api as sm
lowess = sm.nonparametric.lowess
from scipy import stats
```

In [2]:

```
def to_timestamp(dateTime):
    return dateTime.timestamp()

def map_genre(row):
    result = []
    for genre_code in row:
        matches = genres[genres['wikidata_id'] == genre_code]['genre_label'].valu
        for match in matches:
            result.append(match)
    return result
```

In [3]:

```
wikidata = pd.read_json('movies/data/wikidata-movies.json.gz', orient='record', l
#wikidata = pd.read_json('movies/data/wikidata-movies.json.gz', orient='record',
genres = pd.read_json('movies/data/genres.json.gz', orient='record', lines=True,
```

In [4]:

```
wikidata = wikidata[wikidata['made_profit'].notnull()].reset_index(drop=True)
```

In [5]:

```
#movies = movies.copy() #from https://stackoverflow.com/questions/31468176/settin
#movies['genre_names'] = movies.apply(map_genre,axis=1)
wikidata['genre_names'] = wikidata['genre'].apply(map_genre)
wikidata['publication_timestamp'] = wikidata['publication_date'].apply(to_timesta
```

In [6]:

```
rotten_tomatoes = pd.read_json('movies/data/rotten-tomatoes.json.gz', orient='rec
```

In [7]:

```
#rotten_tomatoes
rotten_tomatoes.columns
```

Out[7]:

```
Index(['audience_average', 'audience_percent', 'audience_ratings',
       'critic_average', 'critic_percent', 'imdb_id', 'rotten_tomatoes_id'],
      dtype='object')
```

```
In [8]: omdb = pd.read_json('movies/data/omdb-data.json.gz', orient='record', lines=True)
```

```
In [9]: #omdb
```

```
In [10]: combined = wikidata.join(rotten_tomatoes.set_index('rotten_tomatoes_id'), on='rot
```

```
In [11]: combined = combined.join(omdb.set_index('imdb_id'), on='imdb_id')
```

```
In [12]: combined
```

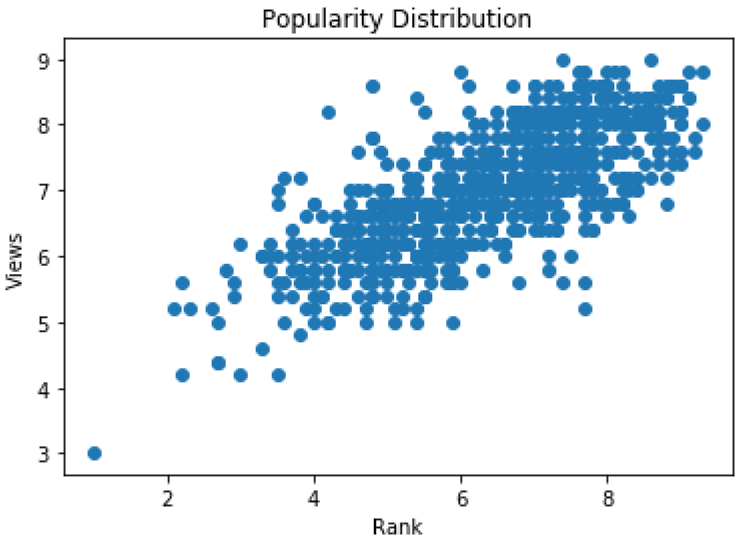
```
1.0   ...      1.278547e+09       4.2       91.0      568239.0      8.1      86

1.0   ...      1.358381e+09       3.4       55.0      143566.0      6.0      65

1.0   ...      6.946560e+07       4.4       98.0      731426.0      9.3      98

1.0   ...      1.325376e+09       4.2       90.0      207900.0      8.4      96
```

```
In [13]: plt.title('Popularity Distribution')
         plt.xlabel('Rank')
         plt.ylabel('Views')
         plt.scatter(combined['critic_average'], combined['audience_average'] * 2)
         plt.show()
```



In [14]:
```
test3 = combined[combined['audience_average'].notnull() & combined['critic_averag
print(stats.normaltest(test3['audience_average']).pvalue) #<0.05, therefore not n
print(stats.mannwhitneyu(test3['critic_average'], test3['audience_average'] * 2).
```

```
0.0012723373652919845
8.904724420410354e-15
```

In [15]:
```
chi2, p, dof, expected = stats.chi2_contingency([test3['critic_average'].values,
print(p) #>0.05, therefore one has no effect on the other?
print(expected)
```

```
1.0
[[5.5546799  7.87989474 7.16941243 ... 5.9422157  7.87989474 8.39660915]
 [3.0453201  4.32010526 3.93058757 ... 3.2577843  4.32010526 4.60339085]]
```

In [16]:
```
# chi2, p, dof, expected = stats.chi2_contingency([test3['critic_average'].values,
# print(p) #>0.05, therefore one has no effect on the other?
# print(expected)
```

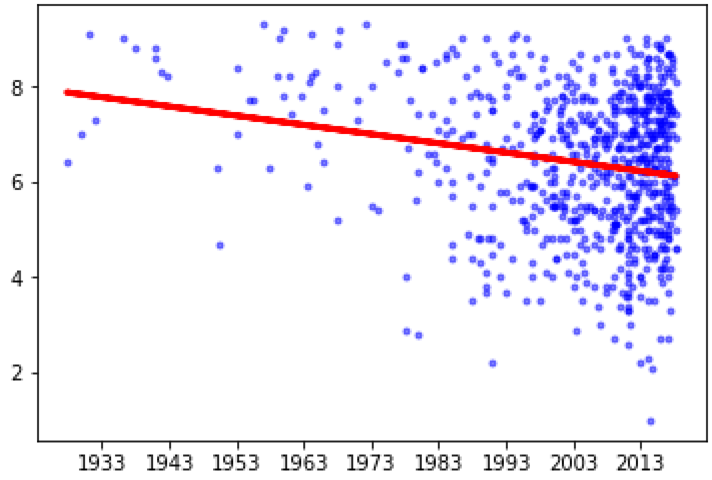
In [17]:
```
#combined.groupby('genre_names')
#pd.value_counts(combined.groupby('genre_names'), sort=False)
## TODO: Count distrbution of genres and graph on histogram
```

Have average ratings changed over time?

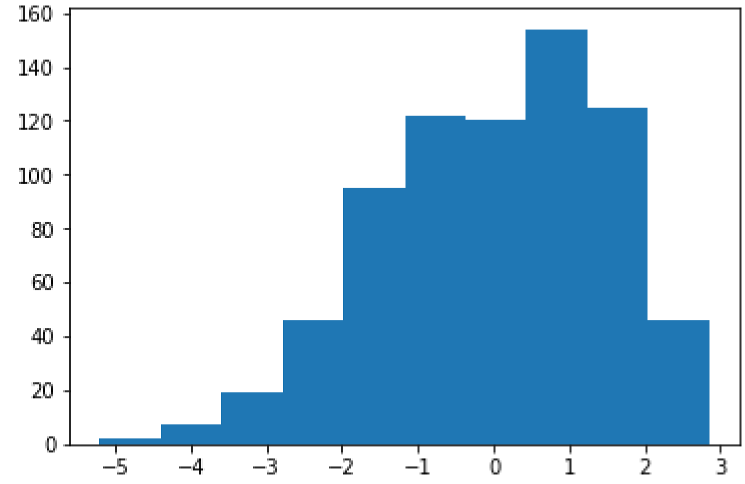
In [18]:
```
critic_average_test = combined[['publication_date','publication_timestamp','criti
fit = stats.linregress(critic_average_test['publication_timestamp'], critic_avera
critic_average_test['prediction'] = critic_average_test['publication_timestamp']*
print(fit.pvalue) #p < 0.05, therefore we can conclude that critic ratings are de
```

```
6.156831173958255e-08
```

In [19]:
```
plt.plot(critic_average_test['publication_date'], critic_average_test['critic_ave
plt.plot(critic_average_test['publication_date'], critic_average_test['prediction
plt.show()
```



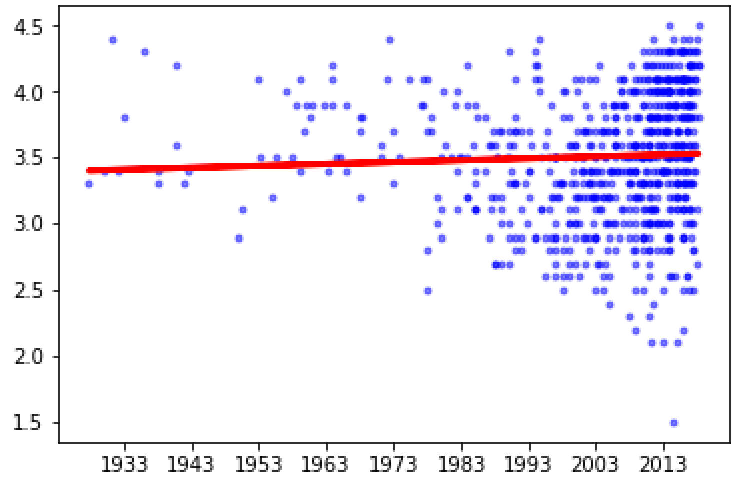
In [20]:
```
plt.hist(np.subtract(critic_average_test['critic_average'],critic_average_test['p
plt.show()
#This is close enough to being normal.
#We expect a greater decline on the high end because the average critic rating is
```



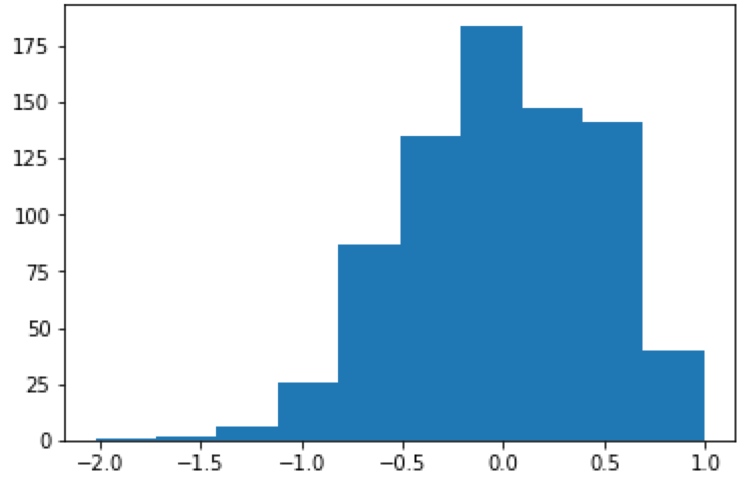
In [21]:
```
audience_average_test = combined[['publication_date','publication_timestamp','aud
fit = stats.linregress(audience_average_test['publication_timestamp'], audience_a
audience_average_test['prediction'] = audience_average_test['publication_timestam
print(fit.pvalue) #p > 0.05, therefore we cannot conclude that the audience ratin
```

```
0.20019655801512012
```

In [22]:
```
plt.plot(audience_average_test['publication_date'], audience_average_test['audien
plt.plot(audience_average_test['publication_date'], audience_average_test['predic
plt.show()
```



In [23]:
```
plt.hist(np.subtract(audience_average_test['audience_average'],audience_average_t
plt.show()
#This is close enough to being normal.
#We expect a greater decline on the high end because the average audience rating
```

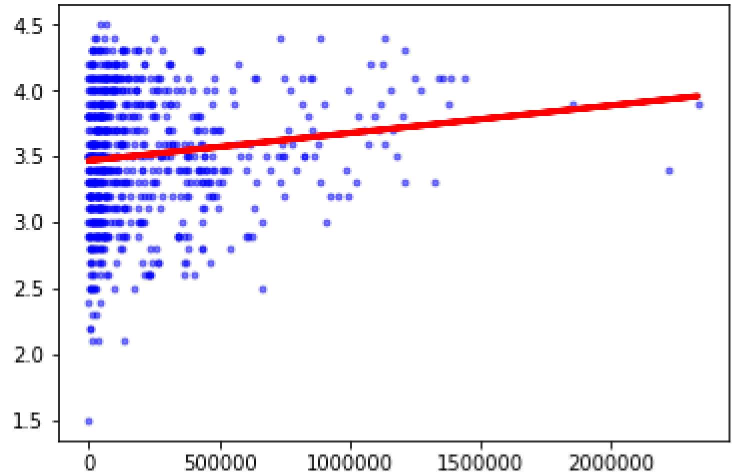


Do average audience ratings change based on its popularity?

In [24]:
```
audience_ratings_test = combined[['publication_date','publication_timestamp','aud
#Removing movies with n >= 10000000 ratings as they seem like outliers
audience_ratings_test = audience_ratings_test[audience_ratings_test['audience_rat
fit = stats.linregress(audience_ratings_test['audience_ratings'], audience_rating
audience_ratings_test['prediction'] = audience_ratings_test['audience_ratings']*f
print(fit.pvalue) #p < 0.05, therefore we can conclude that higher averages corre
```

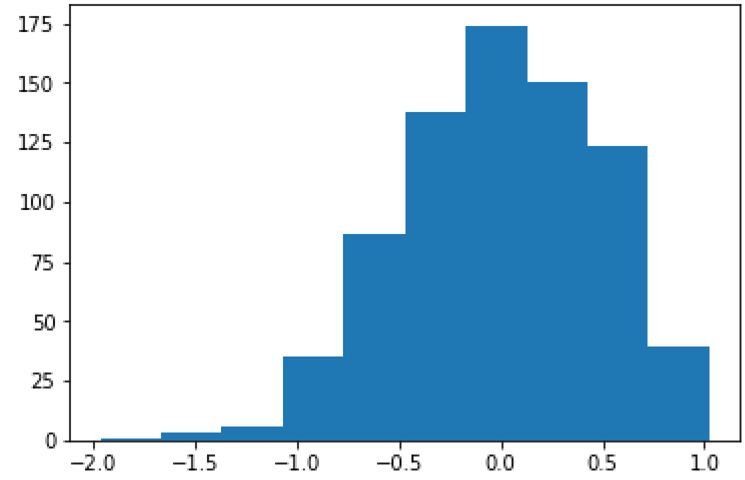
```
0.000308565324974134
```

In [25]:
```
plt.plot(audience_ratings_test['audience_ratings'], audience_ratings_test['audien
plt.plot(audience_ratings_test['audience_ratings'], audience_ratings_test['predic
plt.show()
```



In [26]:

```
plt.hist(np.subtract(audience_ratings_test['audience_average'],audience_ratings_t
plt.show()
#This is close enough to being normal.
#We expect a greater decline on the high end because the average audience rating
```



Does genre have an effect on profitability?

In [27]:

```
def genre_agg(combined_row):
    for genre_id in combined_row['genre']:
        genre_test.loc[genre_test['wikidata_id'] == genre_id,'total']+=1
        if (combined_row['made_profit'] == 1.0):
            genre_test.loc[genre_test['wikidata_id'] == genre_id,'profit']+=1
```

In [28]:

```
genre_test = genres
genre_test['profit'] = 0
genre_test['total'] = 0
combined.apply(genre_agg, axis=1)
genre_test = genre_test[genre_test['total'] > 0]
```

In [29]:

```
genre_test['loss'] = genre_test['total'] - genre_test['profit']
contingency = genre_test[['profit','loss']]
contingency = contingency[contingency['profit'] >= 5]
contingency = contingency[contingency['loss'] >= 5]
chi2, p, dof, expected = stats.chi2_contingency(contingency)
print(p) # p < 0.05, therefore genre has some effect on profitability
```

```
0.01956332775267009

C:\Users\User\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
```

In [ ]:

In [30]:

```
#NLP
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

In [31]:

```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(test3["omdb_plot"])
```

In [51]:

```
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

In [52]:

```
from sklearn.naive_bayes import MultinomialNB
y=test3["audience_average"]
y=y.astype('int')
clf = MultinomialNB().fit(X_train_tfidf, y)
```

In [62]:

```
docs_new = ['love', 'Fat']
X_new_counts = count_vect.transform(docs_new)
X_new_tfidf = tfidf_transformer.transform(X_new_counts)
predicted = clf.predict(X_new_tfidf)
```

In [63]:

```
for doc, category in zip(docs_new, predicted):
     print('%r => %s' % (doc, test3["omdb_plot"][category])) #not working?
```

```
'love' => Lt. John Dunbar is dubbed a hero after he accidentally leads Union tr
oops to a victory during the Civil War. He requests a position on the western f
rontier, but finds it deserted. He soon finds out he is not alone, but meets a
wolf he dubs "Two-socks" and a curious Indian tribe. Dunbar quickly makes frien
ds with the tribe, and discovers a white woman who was raised by the Indians. H
e gradually earns the respect of these native people, and sheds his white-man's
ways.
'Fat' => Lt. John Dunbar is dubbed a hero after he accidentally leads Union tro
ops to a victory during the Civil War. He requests a position on the western fr
ontier, but finds it deserted. He soon finds out he is not alone, but meets a w
olf he dubs "Two-socks" and a curious Indian tribe. Dunbar quickly makes friend
s with the tribe, and discovers a white woman who was raised by the Indians. He
gradually earns the respect of these native people, and sheds his white-man's w
ays.
```