Assignment 2 Solution

Data and Setup

In [1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sps
```

In [2]:
```
movies = pd.read_csv('movies.dat', delimiter='\t', encoding='latin1', na_values=
movies.head()
```

Out[2]:

	id	title	imdbID	spanishTitle	imdbPictureURL	year	
0	1	Toy story	114709	Toy story (juguetes)	http://ia.media-imdb.com/images/M/MV5BMTMwNDU0...	1995	toy_s
1	2	Jumanji	113497	Jumanji	http://ia.media-imdb.com/images/M/MV5BMzM5NjE1...	1995	1068044-jum
2	3	Grumpy Old Men	107050	Dos viejos gruñones	http://ia.media-imdb.com/images/M/MV5BMTI5MTgy...	1993	grumpy_old_
3	4	Waiting to Exhale	114885	Esperando un respiro	http://ia.media-imdb.com/images/M/MV5BMTczMTMy...	1995	waiting_to_ex
4	5	Father of the Bride Part II	113041	Vuelve el padre de la novia (Ahora también abu...	http://ia.media-imdb.com/images/M/MV5BMTg1NDc2...	1995	father_of_the_bride_pa

5 rows × 21 columns

Understanding Missing Data

In [3]:
```
movies.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10197 entries, 0 to 10196
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      10197 non-null  int64
 1   title                   10197 non-null  object
 2   imdbID                  10197 non-null  int64
 3   spanishTitle            10197 non-null  object
 4   imdbPictureURL          10016 non-null  object
 5   year                    10197 non-null  int64
 6   rtID                    9886 non-null   object
 7   rtAllCriticsRating      9967 non-null   float64
 8   rtAllCriticsNumReviews  9967 non-null   float64
```

```
 9   rtAllCriticsNumFresh    9967 non-null   float64
 10  rtAllCriticsNumRotten   9967 non-null   float64
 11  rtAllCriticsScore       9967 non-null   float64
 12  rtTopCriticsRating      9967 non-null   float64
 13  rtTopCriticsNumReviews  9967 non-null   float64
 14  rtTopCriticsNumFresh    9967 non-null   float64
 15  rtTopCriticsNumRotten   9967 non-null   float64
 16  rtTopCriticsScore       9967 non-null   float64
 17  rtAudienceRating        9967 non-null   float64
 18  rtAudienceNumRatings    9967 non-null   float64
 19  rtAudienceScore         9967 non-null   float64
 20  rtPictureURL            9967 non-null   object
dtypes: float64(13), int64(3), object(5)
memory usage: 6.2 MB
```

Converting erroneous 0s to NAs

In [4]:
```
movies.loc[movies['rtAllCriticsRating'] == 0, 'rtAllCriticsRating'] = np.nan
movies.loc[movies['rtTopCriticsRating'] == 0, 'rtTopCriticsRating'] = np.nan
movies.loc[movies['rtAudienceRating'] == 0, 'rtAudienceRating'] = np.nan
#movies['rtAllCriticsRating'].describe()
```

In [5]:
```
movies[['rtAllCriticsRating', 'rtTopCriticsRating', 'rtAudienceRating']]
```

Out[5]:

	rtAllCriticsRating	rtTopCriticsRating	rtAudienceRating
0	9.0	8.5	3.7
1	5.6	5.8	3.2
2	5.9	7.0	3.2
3	5.6	5.5	3.3
4	5.3	5.4	3.0
...	...	...	...
10192	4.4	4.7	3.5
10193	7.0	NaN	3.7
10194	5.6	4.9	3.3
10195	6.7	6.9	3.5
10196	NaN	NaN	NaN

10197 rows × 3 columns

Size and Column of the Data Set

In [6]:
```
movies.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10197 entries, 0 to 10196
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      10197 non-null  int64
 1   title                   10197 non-null  object
```

```
 2   imdbID                  10197 non-null  int64
 3   spanishTitle            10197 non-null  object
 4   imdbPictureURL          10016 non-null  object
 5   year                    10197 non-null  int64
 6   rtID                    9886 non-null   object
 7   rtAllCriticsRating      8441 non-null   float64
 8   rtAllCriticsNumReviews  9967 non-null   float64
 9   rtAllCriticsNumFresh    9967 non-null   float64
 10  rtAllCriticsNumRotten   9967 non-null   float64
 11  rtAllCriticsScore       9967 non-null   float64
 12  rtTopCriticsRating      4662 non-null   float64
 13  rtTopCriticsNumReviews  9967 non-null   float64
 14  rtTopCriticsNumFresh    9967 non-null   float64
 15  rtTopCriticsNumRotten   9967 non-null   float64
 16  rtTopCriticsScore       9967 non-null   float64
 17  rtAudienceRating        7345 non-null   float64
 18  rtAudienceNumRatings    9967 non-null   float64
 19  rtAudienceScore         9967 non-null   float64
 20  rtPictureURL            9967 non-null   object
dtypes: float64(13), int64(3), object(5)
memory usage: 6.2 MB
```

In [7]:
```
movies.shape
```

Out[7]:
```
(10197, 21)
```

In [8]:
```
movies.columns
```

Out[8]:
```
Index(['id', 'title', 'imdbID', 'spanishTitle', 'imdbPictureURL', 'year',
       'rtID', 'rtAllCriticsRating', 'rtAllCriticsNumReviews',
       'rtAllCriticsNumFresh', 'rtAllCriticsNumRotten', 'rtAllCriticsScore',
       'rtTopCriticsRating', 'rtTopCriticsNumReviews', 'rtTopCriticsNumFresh',
       'rtTopCriticsNumRotten', 'rtTopCriticsScore', 'rtAudienceRating',
       'rtAudienceNumRatings', 'rtAudienceScore', 'rtPictureURL'],
      dtype='object')
```

In [9]:
```
movies.size
```

Out[9]:
```
214137
```

Comparing Ratings

Describing the distributions of the RottenTomatoes critic ratings (All Critics and Top Critics), the Audience Rating, and the mean rating given to a movie by MovieLens users, both numerically and graphically

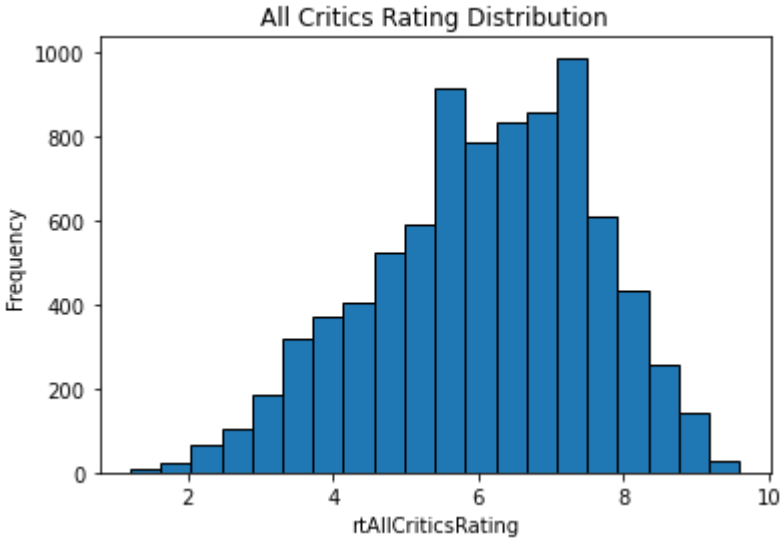
In [10]:
```
movies[['rtAllCriticsRating', 'rtTopCriticsRating', 'rtAudienceRating']].describ
```

Out[10]:

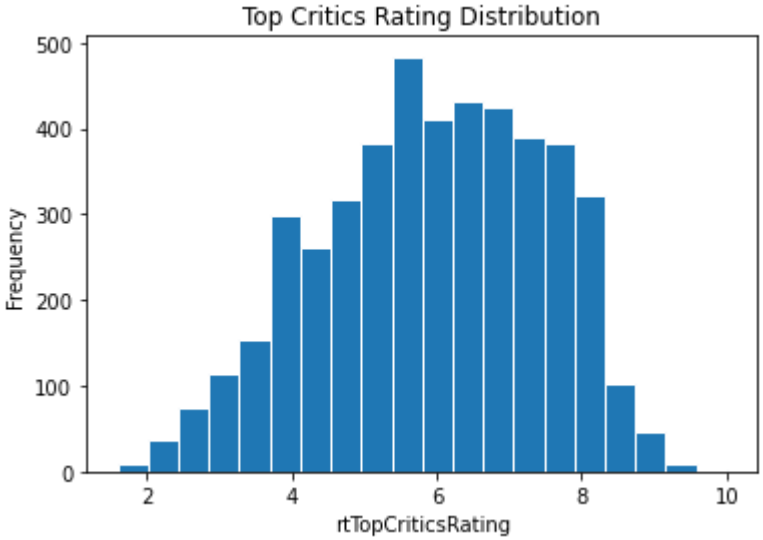
	rtAllCriticsRating	rtTopCriticsRating	rtAudienceRating
count	8441.000000	4662.000000	7345.000000
mean	6.068404	5.930330	3.389258
std	1.526898	1.534093	0.454034

	rtAllCriticsRating	rtTopCriticsRating	rtAudienceRating
min	1.200000	1.600000	1.500000
25%	5.000000	4.800000	3.100000
50%	6.200000	6.100000	3.400000
75%	7.200000	7.100000	3.700000
max	9.600000	10.000000	5.000000

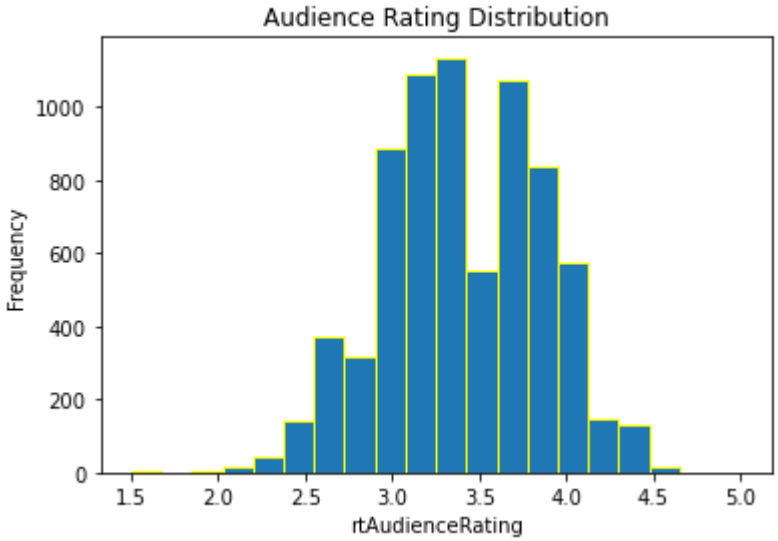
In [11]:
```
plt.hist(movies['rtAllCriticsRating'], edgecolor = 'black', bins = 20)
plt.title('All Critics Rating Distribution')
plt.ylabel('Frequency')
plt.xlabel('rtAllCriticsRating')
plt.show()
```



In [12]:
```
plt.hist(movies['rtTopCriticsRating'], edgecolor = 'white', bins = 20)
plt.title('Top Critics Rating Distribution')
plt.ylabel('Frequency')
plt.xlabel('rtTopCriticsRating')
plt.show()
```



In [13]:
```
plt.hist(movies['rtAudienceRating'], edgecolor = 'yellow', bins = 20)
plt.title('Audience Rating Distribution')
plt.ylabel('Frequency')
plt.xlabel('rtAudienceRating')
plt.show()
```



In [14]:
```
movie_lens = pd.read_csv('hetrec2011-movielens-2k-v2/user_ratedmovies.dat', deli
movie_lens.head()
```

Out[14]:

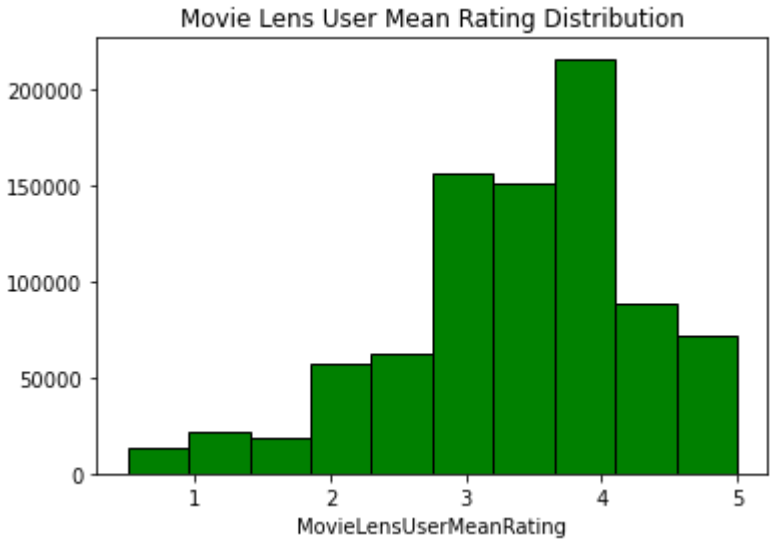
	userID	movieID	rating	date_day	date_month	date_year	date_hour	date_minute	date_second
0	75	3	1.0	29	10	2006	23	17	16
1	75	32	4.5	29	10	2006	23	23	44
2	75	110	4.0	29	10	2006	23	30	8
3	75	160	2.0	29	10	2006	23	16	52
4	75	163	4.0	29	10	2006	23	29	30

In [15]:
```
df1 = movie_lens.groupby('movieID').mean()[['rating']]
df1.head().T
```

Out[15]:

movieID	1	2	3	4	5
rating	3.735154	2.976471	2.873016	2.577778	2.753333

In [16]:
```
plt.hist(movie_lens['rating'], edgecolor = 'black', color = 'green')
plt.title('Movie Lens User Mean Rating Distribution')
plt.xlabel('MovieLensUserMeanRating')
plt.show()
```



Describing the distribution of the difference between the All Critics and Top Critics ratings for movies where both are defined, both numerically and graphically.

In [17]:
```
movies['difference'] = pd.Series.abs(movies['rtAllCriticsRating']-movies['rtTopC
movies['difference'].describe()
```

Out[17]:
```
count    4662.000000
mean        0.407979
std         0.380157
min         0.000000
25%         0.100000
50%         0.300000
75%         0.600000
max         3.200000
Name: difference, dtype: float64
```

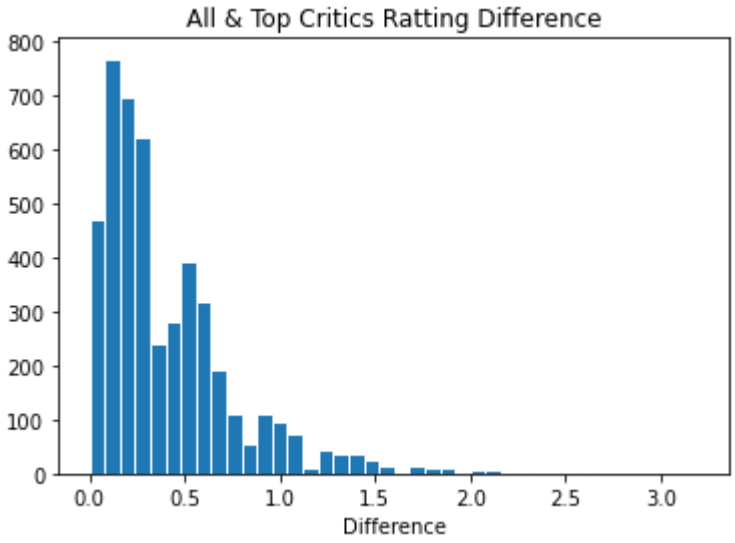
In [18]:
```
d_dist = movies[['rtAllCriticsRating', 'rtTopCriticsRating', 'difference']]
d_dist.T
```

Out[18]:

	0	1	2	3	4	5	6	7	8	9	...	10187	10188	10189	10190	1
rtAllCriticsRating	9.0	5.6	5.9	5.6	5.3	7.7	7.4	4.2	5.2	6.8	...	7.0	NaN	NaN	NaN	
rtTopCriticsRating	8.5	5.8	7.0	5.5	5.4	7.2	7.2	NaN	5.6	6.2	...	6.7	NaN	NaN	NaN	
difference	0.5	0.2	1.1	0.1	0.1	0.5	0.2	NaN	0.4	0.6	...	0.3	NaN	NaN	NaN	

3 rows × 10197 columns

In [19]:
```
plt.hist(d_dist['difference'], edgecolor = 'white', bins = 40)
plt.title('All & Top Critics Ratting Difference')
plt.xlabel('Difference')
plt.show()
```



Paired T-tests

In [20]:
```
sps.ttest_rel(movies['rtAllCriticsRating'], movies['rtTopCriticsRating'], nan_pc
```

Out[20]:
```
Ttest_relResult(statistic=11.691646881769836, pvalue=3.8130588929989856e-31)
```

Since the pvalue is less that 0.05, so it shows a signigicant difference between all and top critics rating.

Audience Rating Average

In [21]:
```
df2 = movies[['id','rtAudienceRating']].set_index('id')
df2.head()
```

Out[21]:

id	rtAudienceRating
1	3.7
2	3.2
3	3.2
4	3.3
5	3.0

In [22]:
```
df3 = df1.join(df2)
df3.head()
```

Out[22]:

movieID	rating	rtAudienceRating
1	3.735154	3.7
2	2.976471	3.2

movieID	rating	rtAudienceRating
3	2.873016	3.2
4	2.577778	3.3
5	2.753333	3.0

In [23]:
```
sps.ttest_rel(df3['rtAudienceRating'], df3['rating'], nan_policy= 'omit')
```

Out[23]:
```
Ttest_relResult(statistic=27.76689581170543, pvalue=2.038842597477622e-161)
```

Since the pvalue is 2.038842597477622e-161 which is very small and less that 0.05, so it shows a signigicant difference between he average audience rating RottenTomatoes users give to a movie and the mean rating MovieLens users give to it.

Here paired t-test is the appropriate since we are comparing the means of the same group-rating under two separate variables i.e. for the second part one variable is audience rating and the another variable is movie lens user's rating to check if their mean rating difference is zero.

Confidence Intervals

In [24]:
```
def mean_ci95(xs):
    mean = np.mean(xs)
    err = sps.sem(xs)
    width = 1.96 * err
    return mean - width, mean + width
```

Computing the mean and a 95% confidence interval for the all-critic ratings

In [25]:
```
movies_genres = pd.read_csv('hetrec2011-movielens-2k-v2/movie_genres.dat', delim
movies_genres.head()
```

Out[25]:

	movieID	genre
0	1	Adventure
1	1	Animation
2	1	Children
3	1	Comedy
4	1	Fantasy

In [26]:
```
df4 = movies[['id','title','rtAllCriticsRating']]
df4 = df4.rename(columns = {'id': 'movieID'})
df4.head()
```

Out[26]:

	movieID	title	rtAllCriticsRating

	movieID	title	rtAllCriticsRating
0	1	Toy story	9.0
1	2	Jumanji	5.6
2	3	Grumpy Old Men	5.9
3	4	Waiting to Exhale	5.6
4	5	Father of the Bride Part II	5.3

In [27]:
```
df5 = movies_genres.merge(df4, on='movieID')
df5.head()
```

Out[27]:

	movieID	genre	title	rtAllCriticsRating
0	1	Adventure	Toy story	9.0
1	1	Animation	Toy story	9.0
2	1	Children	Toy story	9.0
3	1	Comedy	Toy story	9.0
4	1	Fantasy	Toy story	9.0

In [28]:
```
df6 = df5.groupby('genre')['rtAllCriticsRating'].agg(['mean','count','std','sem'
df6.head()
```

Out[28]:

genre	mean	count	std	sem
Film-Noir	7.253543	127	1.273527	0.113007
Documentary	7.129641	334	0.979147	0.053577
IMAX	6.950000	16	0.747440	0.186860
War	6.753351	388	1.354775	0.068778
Western	6.613472	193	1.394007	0.100343

Finding the confidence Interval

In [29]:
```
df6['ci_width'] = df6['sem']*1.96
df6['ci_low'] = df6['mean']- (df6['sem'] * 1.96)
df6['ci_high'] = df6['mean']+ (df6['sem'] * 1.96)
df6
```

Out[29]:

genre	mean	count	std	sem	ci_width	ci_low	ci_high
Film-Noir	7.253543	127	1.273527	0.113007	0.221494	7.032049	7.475038
Documentary	7.129641	334	0.979147	0.053577	0.105010	7.024631	7.234651
IMAX	6.950000	16	0.747440	0.186860	0.366246	6.583754	7.316246

genre	mean	count	std	sem	ci_width	ci_low	ci_high
War	6.753351	388	1.354775	0.068778	0.134805	6.618545	6.888156
Western	6.613472	193	1.394007	0.100343	0.196672	6.416800	6.810143
Musical	6.483573	347	1.319328	0.070825	0.138817	6.344756	6.622391
Drama	6.462657	4306	1.337138	0.020377	0.039939	6.422718	6.502596
Animation	6.343404	235	1.393417	0.090897	0.178157	6.165247	6.521561
Mystery	6.228571	434	1.488513	0.071451	0.140044	6.088528	6.368615
Romance	6.194744	1427	1.432974	0.037934	0.074350	6.120394	6.269094
Crime	6.161612	943	1.494892	0.048680	0.095414	6.066198	6.257025
Fantasy	6.023362	458	1.577800	0.073726	0.144502	5.878860	6.167865
Adventure	5.952876	817	1.519157	0.053149	0.104171	5.848705	6.057048
Thriller	5.856680	1452	1.503450	0.039455	0.077332	5.779348	5.934013
Children	5.779864	442	1.532010	0.072870	0.142826	5.637039	5.922690
Comedy	5.732409	3030	1.546115	0.028088	0.055052	5.677357	5.787462
Action	5.574497	1192	1.560583	0.045201	0.088594	5.485903	5.663091
Sci-Fi	5.567601	571	1.554942	0.065072	0.127542	5.440059	5.695142
Horror	5.471046	784	1.601864	0.057209	0.112130	5.358915	5.583176

Yes it shows that there is a slight difference of mean critic ratings between the top two genres. We can see that from the table- Film-Noir(7.032049, 7.475038) and Documentary(7.032049, 7.475038) genres have overlap confidence interval. So, we need to perform additional t-test for ensuring if their mean difference is statistically significant or not.

Yes there is difference of mean critic ratings between the top and the bottom genres. ilm-Noir(7.032049, 7.475038) and Horror(5.358915, 5.583176) genres do not have overlap confidence interval.

Computing the mean and a 95% bootstrapped confidence interval for the mean all-critic score rating

In [30]:
```
rng = np.random.default_rng(20200913)
```

In [31]:
```
def boot_mean_estimate(vals, nboot=10000):
    obs = vals.dropna()  # ignore missing values
    mean = obs.mean()
    n = obs.count()

    boot_means = [np.mean(rng.choice(obs, size=n)) for i in range(nboot)]
    ci_low, ci_high = np.quantile(boot_means, [0.025, 0.975])
    return pd.Series({
        'mean': mean,
        'count': n,
        'ci_low': ci_low,
```

```
        'ci_high': ci_high
    })
```

In [32]:
```
df5.groupby('genre')['rtAllCriticsRating'].apply(boot_mean_estimate).unstack().s
```

```
/home/nahidanwar/anaconda3/lib/python3.8/site-packages/numpy/core/fromnumeric.p
y:3440: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/home/nahidanwar/anaconda3/lib/python3.8/site-packages/numpy/core/_methods.py:18
9: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

Out[32]:

genre	mean	count	ci_low	ci_high
Film-Noir	7.253543	127.0	7.029921	7.467717
Documentary	7.129641	334.0	7.020951	7.235030
IMAX	6.950000	16.0	6.606250	7.306250
War	6.753351	388.0	6.618557	6.887113
Western	6.613472	193.0	6.412435	6.805699
Musical	6.483573	347.0	6.345814	6.621909
Drama	6.462657	4306.0	6.422734	6.502323
Animation	6.343404	235.0	6.167660	6.519574
Mystery	6.228571	434.0	6.088249	6.367972
Romance	6.194744	1427.0	6.119690	6.270922
Crime	6.161612	943.0	6.067017	6.256734
Fantasy	6.023362	458.0	5.879907	6.170311
Adventure	5.952876	817.0	5.847858	6.056429
Thriller	5.856680	1452.0	5.779475	5.934160
Children	5.779864	442.0	5.635294	5.919457
Comedy	5.732409	3030.0	5.677855	5.786272
Action	5.574497	1192.0	5.487408	5.662334
Sci-Fi	5.567601	571.0	5.438529	5.694225
Horror	5.471046	784.0	5.356626	5.583039

Using the above information, we can say that confidence interval by standard error and bootstraped are approximately similar which means in some genres there is a slight difference.

Popularity and Bootstraps

Computing the number of MovieLens users who have rated each movie

In [33]:
```
df7 = movie_lens.merge(df4, on='movieID')
df7.head()
```

Out[33]:

	userID	movieID	rating	date_day	date_month	date_year	date_hour	date_minute	date_second
0	75	3	1.0	29	10	2006	23	17	16
1	783	3	2.0	2	11	2006	18	33	55
2	788	3	3.5	31	8	2007	14	49	41
3	1160	3	4.0	15	7	2008	14	38	42
4	1174	3	2.5	23	3	2005	6	23	20

In [34]:
```
df7[['title']].value_counts().to_frame().head()
```

Out[34]:

		0
	title	
	Kill Bill: Vol. 2	2406
	Spider-Man	2206
	Die Hard	1703
The Lord of the Rings: The Return of the King		1682
	The Matrix	1670

Testing null hypothesis by bootstraped pvalue for the rating of movielens user

In [35]:
```
df8 = movies_genres.merge(df7[['movieID', 'title']], on='movieID')
df8.head()
```

Out[35]:

	movieID	genre	title
0	1	Adventure	Toy story
1	1	Adventure	Toy story
2	1	Adventure	Toy story
3	1	Adventure	Toy story
4	1	Adventure	Toy story

In [36]:
```
action = df8.loc[df8['genre'] == 'Action']
s1 = action[['title']].value_counts()
```

In [37]:
```
documentary = df8.loc[df8['genre'] == 'Documentary']
s2 = documentary[['title']].value_counts()
```

In [38]:
```
def boot_ind(s1, s2, nboot=10000):
    ## we will ignore NAs here
    obs1 = s1.dropna()
    obs2 = s2.dropna()
    n1 = len(obs1)
    n2 = len(obs2)

    ## pool the observations together
    pool = pd.concat([obs1, obs2])
    ## grab the observed mean
    md = np.median(s1) - np.median(s2)

    ## compute our bootstrap samples of the mean under H0
    b1 = np.array([np.median(rng.choice(pool, size=n1)) for i in range(nboot)])
    b2 = np.array([np.median(rng.choice(pool, size=n2)) for i in range(nboot)])

    ## the P-value is the probability that we observe a difference as large
    ## as we did in the raw data, if the null hypothesis were true
    return md, np.mean(np.abs(b1 - b2) >= np.abs(md))
```

In [39]:
```
boot_ind(s1, s2)
```

Out[39]:
```
(45.0, 0.0)
```

Since the resulting bootstraped pvalue is 0.0 which is less than 0.05, so we reject the null hypothesis.

In [40]:
```
s1.median()
```

Out[40]:
```
57.0
```

In [41]:
```
s2.median() # Documentary
```

Out[41]:
```
12.0
```

In [42]:
```
print(f"Median number of ratings for movies in Action and Documentary genres are
```

```
Median number of ratings for movies in Action and Documentary genres are 57.0 an
d 12.0 respectively
```

Testing null hypothesis by bootstraped pvalue for the audience ratings from RottenTomatoes

In [43]:
```
df9 = movies[['id','title','rtAudienceRating']]
df9 = df9.rename(columns = {'id': 'movieID'})
df9.head()
```

Out[43]:

	movieID	title	rtAudienceRating

	movieID	title	rtAudienceRating
0	1	Toy story	3.7
1	2	Jumanji	3.2
2	3	Grumpy Old Men	3.2
3	4	Waiting to Exhale	3.3
4	5	Father of the Bride Part II	3.0

In [44]:
```
df10 = movie_lens.merge(df9, on='movieID')
df10.head()
```

Out[44]:

	userID	movieID	rating	date_day	date_month	date_year	date_hour	date_minute	date_second
0	75	3	1.0	29	10	2006	23	17	16
1	783	3	2.0	2	11	2006	18	33	55
2	788	3	3.5	31	8	2007	14	49	41
3	1160	3	4.0	15	7	2008	14	38	42
4	1174	3	2.5	23	3	2005	6	23	20

In [45]:
```
df10[['title']].value_counts().to_frame().head()
```

Out[45]:

		0
	title	
	Kill Bill: Vol. 2	2406
	Spider-Man	2206
	Die Hard	1703
	The Lord of the Rings: The Return of the King	1682
	The Matrix	1670

In [46]:
```
df11 = movies_genres.merge(df10[['movieID', 'title']], on='movieID')
df11.head()
```

Out[46]:

	movieID	genre	title
0	1	Adventure	Toy story
1	1	Adventure	Toy story

	movieID	genre	title
2	1	Adventure	Toy story
3	1	Adventure	Toy story
4	1	Adventure	Toy story

In [47]:
```
action1 = df11.loc[df11['genre'] == 'Action']
s3 = action1[['title']].value_counts()
```

In [48]:
```
documentary1 = df11.loc[df11['genre'] == 'Documentary']
s4 = documentary1[['title']].value_counts()
```

In [49]:
```
boot_ind(s3, s4)
```

Out[49]: (45.0, 0.0)

As like the rating of the movie lens user, the resulting bootstraped pvalue is also 0.0 which is less than 0.05, so we reject the null hypothesis.

In [50]:
```
s3.median()
```

Out[50]: 57.0

In [51]:
```
s4.median()
```

Out[51]: 12.0

In [52]:
```
print(f"Median number of ratings for movies in Action and Documentary genres are
```

```
Median number of ratings for movies in Action and Documentary genres are 57.0 an
d 12.0 respectively
```

Comparing the mean of the critic ratings (All Critics ratings from Rotten Tomatoes) between action and documentary movies.

Testing the difference with the bootstrap

In [53]:
```
def boot_ind(s1, s2, nboot=10000):
    ## we will ignore NAs here
    obs1 = s1.dropna()
    obs2 = s2.dropna()
    n1 = len(obs1)
    n2 = len(obs2)

    ## pool the observations together
    pool = pd.concat([obs1, obs2])
    ## grab the observed mean
    md = np.mean(s1) - np.mean(s2)
```

```
    ## compute our bootstrap samples of the mean under H0
    b1 = np.array([np.mean(rng.choice(pool, size=n1)) for i in range(nboot)])
    b2 = np.array([np.mean(rng.choice(pool, size=n2)) for i in range(nboot)])

    ## the P-value is the probability that we observe a difference as large
    ## as we did in the raw data, if the null hypothesis were true
    return md, np.mean(np.abs(b1 - b2) >= np.abs(md))
```

In [54]:
```
action_df = df5.loc[df5['genre'] == 'Action']
documentary_df = df5.loc[df5['genre'] == 'Documentary']
```

In [55]:
```
boot_ind(action_df['rtAllCriticsRating'],documentary_df['rtAllCriticsRating'])
```

Out[55]:
```
(-1.5551440742675675, 0.0)
```

Here pvalue = 0.0 < 0.05, so we reject the null hypothesis which means there is difference of mean of the Rotten Tomamtoes All Critics Rating for the action and documentary genres.

Testing the difference with the appropriate (independant) t-test.

In [56]:
```
sps.ttest_ind(action_df['rtAllCriticsRating'],documentary_df['rtAllCriticsRating
```

Out[56]:
```
Ttest_indResult(statistic=-17.28144197127851, pvalue=3.024636237600687e-61)
```

Here also pvalue is very small < 0.05, so we reject the null hypothesis. There is difference between the action and documentary genres of the Rotten Tomamtoes All Critics Rating.

Independent sample t-test is commonly used to find out the statistical differences between the means of two groups, that is why we used it here.

Reflection

From this assignment, I learned many things about statistics and its implementation. At first I learned how to handle missing data in a dataset using the python library. Also I can set up the environment to load and do operations on the data. I learned how to use the data dictionary/readme file to understand the data clearly. From there selecting appropriate variables to represent it numerically and graphically. Since there are three types of t-test, I learned which t-test to use in which scenario and its implementation. In addition, I learned about the bootstrap method which is basically a kind of sampling technique with replacement when the sample size is small and how to implement it. I also learned about the process of calculating standard error to find out the confidence interval.

Along with this I learned about the null hypothesis and in which scenario we can accept or reject the hypothesis. So I think as a whole I learned important aspects of statistics i.e. inference and t-test etc. which can be used to process data, maintain and analyze in an efficient way.