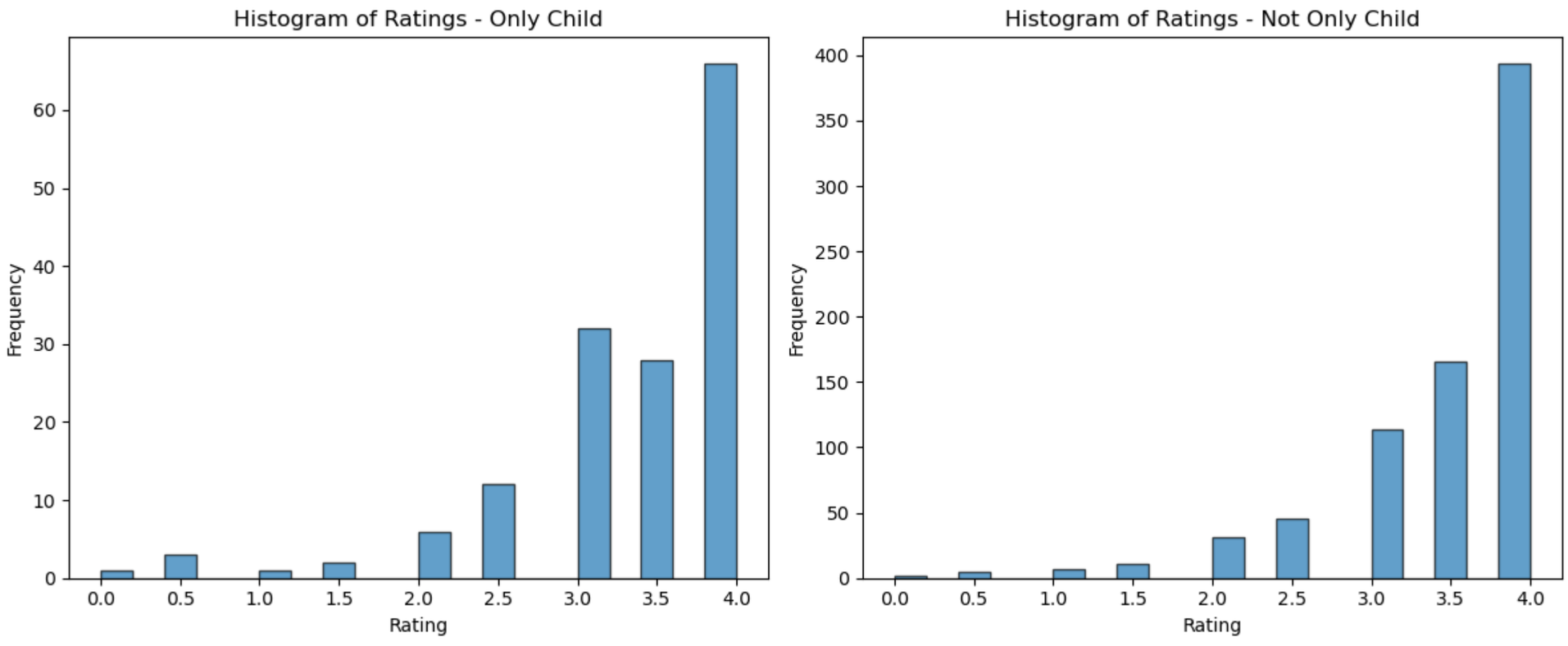
In [9]:
```
#5

import pandas as pd
import numpy as np
from scipy.stats import kstest

# Load the dataset
data = pd.read_csv('movieReplicationSet.csv')

# Handle missing data and non-responses
data = data.replace('N/A', pd.NA)
data = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] != -1]
data.dropna(subset=['The Lion King (1994)'], inplace=True)

# Separate ratings based on the 'only_child_column'
only_child_ratings = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 1]['The Lion King (1994)'].astype(float)
not_only_child_ratings = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 0]['The Lion King (1994)'].astype(float)

# Check normality using K-S test for only children's ratings
stat1, p_value1 = kstest(only_child_ratings, 'norm', args=(only_child_ratings.mean(), only_child_ratings.std()))

# Check normality using K-S test for those who are not the only child
stat2, p_value2 = kstest(not_only_child_ratings, 'norm', args=(not_only_child_ratings.mean(), not_only_child_ratings.std()))

# Interpret the result
alpha = 0.005

if p_value1 > alpha:
    print(f"The ratings from only children follow a normal distribution (p-value = {p_value1}).")
else:
    print(f"The ratings from only children do not follow a normal distribution (p-value = {p_value1}).")

if p_value2 > alpha:
    print(f"The ratings from those who are not the only child follow a normal distribution (p-value = {p_value2}).")
else:
    print(f"The ratings from those who are not the only child do not follow a normal distribution (p-value = {p_value2}).")
```

```
The ratings from only children do not follow a normal distribution (p-value = 3.412348324234531e-07).
The ratings from those who are not the only child do not follow a normal distribution (p-value = 2.358702819665285e-51).
```

In [11]:
```
# Plot histograms
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.hist(only_child_ratings, bins=20, edgecolor='k', alpha=0.7)
plt.title('Histogram of Ratings - Only Child')
plt.xlabel('Rating')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(not_only_child_ratings, bins=20, edgecolor='k', alpha=0.7)
plt.title('Histogram of Ratings - Not Only Child')
plt.xlabel('Rating')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



In [32]:
```
import pandas as pd
import numpy as np
from scipy.stats import mannwhitneyu

# Load the dataset
data = pd.read_csv('movieReplicationSet.csv')

# Filter the data to exclude non-responses for 'only child' and 'movie preference'
data = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] != -1]
data = data[data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] != -1]
data.dropna(subset=['The Lion King (1994)'], inplace=True)

def get_balanced_sample(group_data, sample_size_per_preference=50):
    enjoy_alone = group_data[group_data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 1]
    enjoy_socially = group_data[group_data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 0]

    return pd.concat([
        enjoy_alone.sample(sample_size_per_preference, replace=True),
        enjoy_socially.sample(sample_size_per_preference, replace=True)
    ])

# Bootstrap resampling with Mann-Whitney U test for pairwise comparison
num_iterations = 10000
p_values = []

only_child_data = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 1]
not_only_child_data = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 0]

for _ in range(num_iterations):
    only_child_sample = get_balanced_sample(only_child_data)
    not_only_child_sample = get_balanced_sample(not_only_child_data)

    _, p_value = mannwhitneyu(
        only_child_sample['The Lion King (1994)'].astype(float),
        not_only_child_sample['The Lion King (1994)'].astype(float),
        alternative='greater'
    )

    p_values.append(p_value)

avg_p_value = np.mean(p_values)

# Interpret the result
alpha = 0.005

if avg_p_value <= alpha:
    print(f"There is a statistically significant difference in ratings between only children and those with siblings (average p-value = {avg_p_value}).")
else:
    print(f"There is no statistically significant difference in ratings between only children and those with siblings (average p-value = {avg_p_value}).")
```

```
There is no statistically significant difference in ratings between only children and those with siblings (average p-value = 0.7991836133052952).
```

In [ ]:

In [ ]:
```
data = pd.read_csv('movieReplicationSet.csv')
for movie in data.columns[:400]:
    only_child_ratings = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 1][movie].astype(float)
    not_only_child_ratings = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 0][movie].astype(float)
    # remove nan
    only_child_ratings = only_child_ratings[~np.isnan(only_child_ratings)]
    not_only_child_ratings = not_only_child_ratings[~np.isnan(not_only_child_ratings)]
    print(not_only_child_ratings.shape, only_child_ratings.shape)
```

```
(55,) (17,)
(54,) (15,)
(361,) (77,)
(227,) (54,)
(372,) (76,)
(99,) (24,)
(140,) (35,)
(208,) (36,)
(101,) (28,)
(472,) (99,)
(321,) (72,)
(109,) (31,)
(330,) (58,)
(76,) (11,)
(453,) (107,)
(54,) (18,)
(312,) (60,)
(285,) (63,)
(280,) (76,)
```

In [ ]:
```python
# 6
import pandas as pd
import numpy as np
from scipy.stats import mannwhitneyu, ttest_ind, kstest
from tqdm import tqdm
from copy import deepcopy
data = pd.read_csv('movieReplicationSet.csv')
alpha = 0.005
movie_to_pvalue = dict()
pbar = tqdm(data.columns[:400])
for movie in pbar:
    movie_data = deepcopy(data)
    movie_data = movie_data[movie_data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] != -1]
    movie_data = movie_data[movie_data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] != -1]
    movie_data.dropna(subset=[movie], inplace=True)


    only_child_ratings = movie_data[movie_data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 1][movie].astype(float)
    # print('only_child_ratings', only_child_ratings.shape)
    not_only_child_ratings = movie_data[movie_data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 0][movie].astype(float)
    # print('not_only_child_ratings', not_only_child_ratings.shape)
    # raise RuntimeError

    stat1, p_value1 = kstest(only_child_ratings, 'norm', args=(only_child_ratings.mean(), only_child_ratings.std()))
    stat2, p_value2 = kstest(not_only_child_ratings, 'norm', args=(not_only_child_ratings.mean(), not_only_child_ratings.std()))

    # sample_size = int(min(only_child_ratings.shape[0], not_only_child_ratings.shape[0]) * 0.8)
    # because too less data
    # sample_size = 100
    sample_size = 200
    # print(only_child_ratings.shape[0], not_only_child_ratings.shape[0])
    if sample_size == 0:
        continue
    if p_value1 < alpha or p_value2 < alpha:
        # u test
        num_iterations = 1000
        p_values = []

        for _ in range(num_iterations):
            only_child_sample = only_child_ratings.sample(sample_size, replace=True)
            not_only_child_sample = not_only_child_ratings.sample(sample_size, replace=True)
            _, p_value = mannwhitneyu(
                only_child_sample.astype(float),
                not_only_child_sample.astype(float),
                alternative='greater'
            )

            p_values.append(p_value)

        avg_p_value = np.mean(p_values)
        # print(f'p-value = {avg_p_value}')
    else:
        # use t test
        num_iterations = 1000
        p_values = []

        for _ in range(num_iterations):
            only_child_sample = only_child_ratings.sample(sample_size, replace=True)
            not_only_child_sample = not_only_child_ratings.sample(sample_size, replace=True)
            _, p_value = ttest_ind(
                only_child_sample.astype(float),
```

```
                not_only_child_sample.astype(float),
                alternative='greater'
            )

            p_values.append(p_value)

        avg_p_value = np.mean(p_values)
        # print(f'p-value = {avg_p_value}')
    movie_to_pvalue[movie] = avg_p_value
    pbar.set_description(f'number of movies is significant: {len([p for p in movie_to_pvalue.values() if p < alpha])}')
```

```
number of movies is significant: 2: 100%|██████████| 400/400 [02:04<00:00,  3.22it/s]
```

In [ ]:
```
cnt = 0
for movie in movie_to_pvalue:
    if movie_to_pvalue[movie] < 0.005:
        print(movie)
        cnt += 1
print(f'portion: {cnt/400}')
# print(f'{400-len(movie_to_pvalue)} does not have enough data')
```

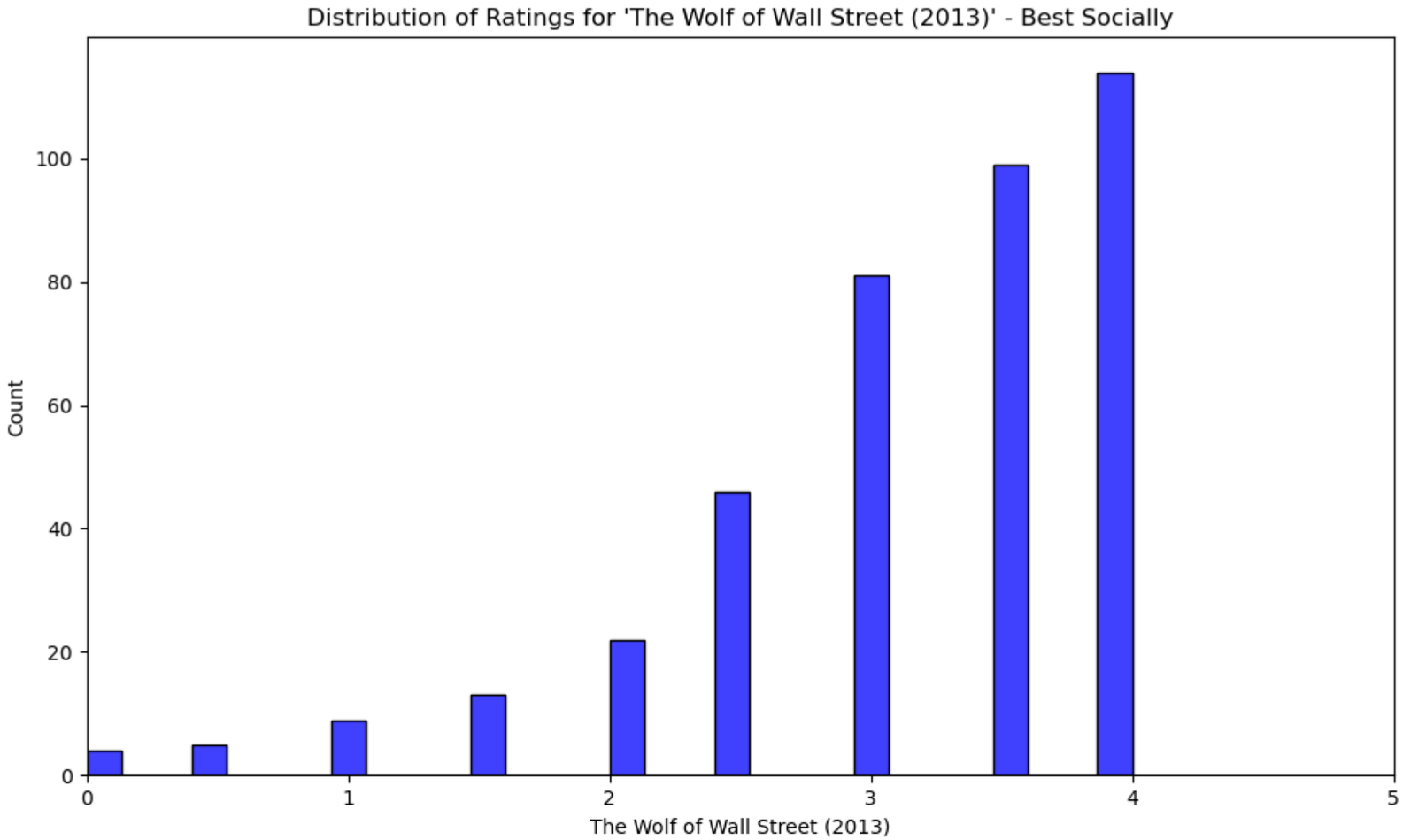
```
Leon (1994)
The Land That Time Forgot (1974)
portion: 0.005
0 does not have enough data
```

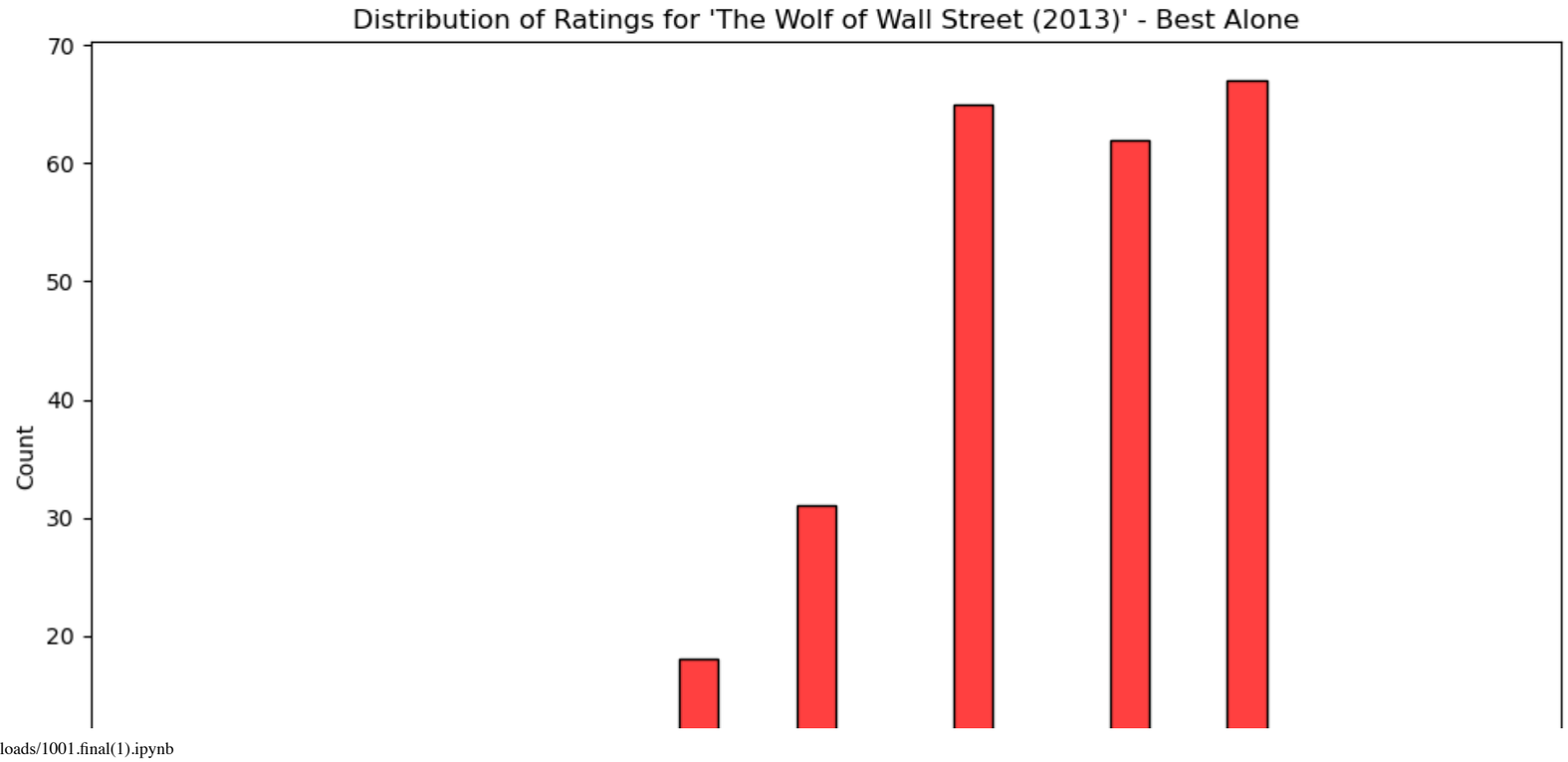
In [ ]:
```

```

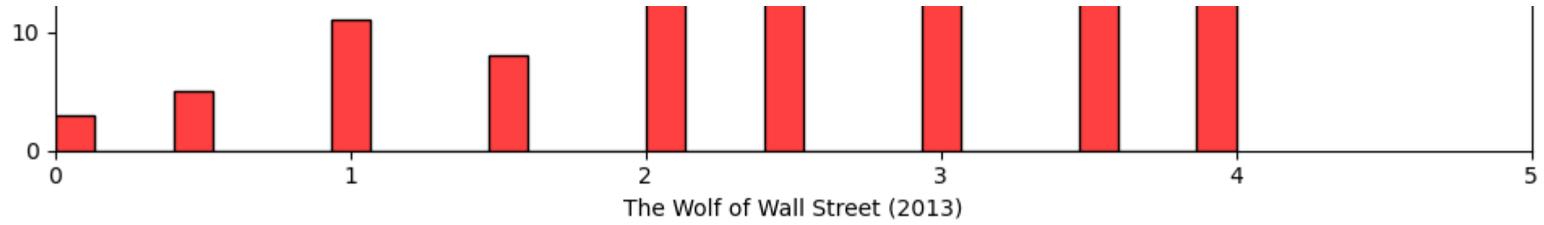
In [13]:
```
# 7
import pandas as pd
from scipy.stats import kstest

# Load the dataset
data = pd.read_csv('movieReplicationSet.csv')

# Assuming you have a column like 'Watch preference (1: Socially; 0: Alone)'
# Handle missing data and non-responses
data = data.replace('N/A', pd.NA)
data.dropna(subset=['The Wolf of Wall Street (2013)'], inplace=True)

# Separate ratings based on watch preference
social_watch_ratings = data[data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 1]['The Wolf of Wall Street (2013)'].astype(float)
alone_watch_ratings = data[data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 0]['The Wolf of Wall Street (2013)'].astype(float)

# Check normality using K-S test for social watchers' ratings
stat1, p_value1 = kstest(social_watch_ratings, 'norm', args=(social_watch_ratings.mean(), social_watch_ratings.std()))

# Check normality using K-S test for those who watch alone
stat2, p_value2 = kstest(alone_watch_ratings, 'norm', args=(alone_watch_ratings.mean(), alone_watch_ratings.std()))

# Interpret the result
alpha = 0.005

if p_value1 > alpha:
    print(f"The ratings from those who watch movies socially follow a normal distribution (p-value = {p_value1}).")
else:
    print(f"The ratings from those who watch movies socially do not follow a normal distribution (p-value = {p_value1}).")

if p_value2 > alpha:
    print(f"The ratings from those who watch movies alone follow a normal distribution (p-value = {p_value2}).")
else:
    print(f"The ratings from those who watch movies alone do not follow a normal distribution (p-value = {p_value2}).")
```

```
The ratings from those who watch movies socially do not follow a normal distribution (p-value = 2.216819709934105e-14).
The ratings from those who watch movies alone do not follow a normal distribution (p-value = 2.4501961699727183e-10).
```

In [22]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('movieReplicationSet.csv')

# Handle missing data and non-responses
data = data.replace('N/A', pd.NA)
data.dropna(subset=['The Wolf of Wall Street (2013)'], inplace=True)

# Filter out "-1" responses
data = data[data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] != -1]

# Separate data based on watch preference
social_watch_data = data[data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 1]['The Wolf of Wall Street (2013)']
alone_watch_data = data[data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 0]['The Wolf of Wall Street (2013)']

# Plot histograms separately
fig, axs = plt.subplots(2, 1, figsize=(10, 12))

# Social watchers' ratings
sns.histplot(social_watch_data, ax=axs[0], color='blue', bins=30, kde=False)
axs[0].set_title("Distribution of Ratings for 'The Wolf of Wall Street (2013)' - Best Socially")
axs[0].set_xlim(0, 5)

# Ratings for those who watch alone
sns.histplot(alone_watch_data, ax=axs[1], color='red', bins=30, kde=False)
axs[1].set_title("Distribution of Ratings for 'The Wolf of Wall Street (2013)' - Best Alone")
axs[1].set_xlim(0, 5)

plt.tight_layout()
plt.show()
```


Distribution of Ratings for 'The Wolf of Wall Street (2013)' - Best Socially
Count
0
20
40
60
80
100
0
1
2
3
4
5
The Wolf of Wall Street (2013)


Distribution of Ratings for 'The Wolf of Wall Street (2013)' - Best Alone
Count
20
30
40
50
60
70



In [37]:
```
import pandas as pd
import numpy as np
from scipy.stats import mannwhitneyu

# Load the dataset
data = pd.read_csv('movieReplicationSet.csv')

# Handle missing data and non-responses
data = data.replace('N/A', pd.NA)
data.dropna(subset=['The Wolf of Wall Street (2013)'], inplace=True)

def get_balanced_sample():
    only_child_sample = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 1].sample(50)
    not_only_child_sample = data[data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] == 0].sample(50)
    return pd.concat([only_child_sample, not_only_child_sample])

# Bootstrap resampling with Mann-Whitney U test
num_iterations = 10000
p_values = []

for _ in range(num_iterations):
    balanced_sample = get_balanced_sample()

    # Separate ratings based on watch preference
    social_watch_ratings = balanced_sample[balanced_sample['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 0]['The Wolf of Wall Street (201
    alone_watch_ratings = balanced_sample[balanced_sample['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 1]['The Wolf of Wall Street (2013

    # Perform the Mann-Whitney U test
    _, p_value = mannwhitneyu(social_watch_ratings, alone_watch_ratings, alternative='two-sided')
    p_values.append(p_value)

# Compute average p-value
avg_p_value = np.mean(p_values)

# Interpret the result
alpha = 0.005

if avg_p_value <= alpha:
    print(f"Based on the average p-value, there is a statistically significant difference in ratings for 'The Wolf of Wall Street (2013)' between those who enjoy wa
else:
    print(f"Based on the average p-value, there is no statistically significant difference in ratings for 'The Wolf of Wall Street (2013)' between those who enjoy w
```

```
Based on the average p-value, there is no statistically significant difference in ratings for 'The Wolf of Wall Street (2013)' between those who enjoy watching mov
ies alone and those who enjoy them socially (average p-value = 0.543898453046458).
```

In [ ]:

In [ ]:

problem8

```
import pandas as pd
import numpy as np
from scipy.stats import mannwhitneyu, ttest_ind, kstest
from tqdm import tqdm
from copy import deepcopy
data = pd.read_csv('movieReplicationSet.csv')
alpha = 0.005
movie_to_pvalue = dict()
pbar = tqdm(data.columns[:400])
for movie in pbar:
    movie_data = deepcopy(data)
    movie_data = movie_data[movie_data['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'] != -1]
    movie_data = movie_data[movie_data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] != -1]
    movie_data.dropna(subset=[movie], inplace=True)


    only_child_ratings = movie_data[movie_data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 1][movie].astype(float)
    # print('only_child_ratings', only_child_ratings.shape)
    not_only_child_ratings = movie_data[movie_data['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'] == 0][movie].astype(float)
    # print('not_only_child_ratings', not_only_child_ratings.shape)
    # raise RuntimeError

    stat1, p_value1 = kstest(only_child_ratings, 'norm', args=(only_child_ratings.mean(), only_child_ratings.std()))
    stat2, p_value2 = kstest(not_only_child_ratings, 'norm', args=(not_only_child_ratings.mean(), not_only_child_ratings.std()))

    # sample_size = int(min(only_child_ratings.shape[0], not_only_child_ratings.shape[0]) * 0.8)
    # because too less data
    # sample_size = 100
    sample_size = 200
    # print(only_child_ratings.shape[0], not_only_child_ratings.shape[0])
    if sample_size == 0:
        continue
    if p_value1 < alpha or p_value2 < alpha:
        # u test
        num_iterations = 1000
        p_values = []

        for _ in range(num_iterations):
            only_child_sample = only_child_ratings.sample(sample_size, replace=True)
            not_only_child_sample = not_only_child_ratings.sample(sample_size, replace=True)
            _, p_value = mannwhitneyu(
                only_child_sample.astype(float),
                not_only_child_sample.astype(float),
                alternative='greater'
            )

            p_values.append(p_value)

        avg_p_value = np.mean(p_values)
        # print(f'p-value = {avg_p_value}')
    else:
        # use t test
        num_iterations = 1000
        p_values = []

        for _ in range(num_iterations):
            only_child_sample = only_child_ratings.sample(sample_size, replace=True)
            not_only_child_sample = not_only_child_ratings.sample(sample_size, replace=True)
            _, p_value = ttest_ind(
                only_child_sample.astype(float),
                not_only_child_sample.astype(float),
```

```
                alternative='greater'
            )

            p_values.append(p_value)

        avg_p_value = np.mean(p_values)
        # print(f'p-value = {avg_p_value}')
    movie_to_pvalue[movie] = avg_p_value
    pbar.set_description(f'number of movies is significant: {len([p for p in movie_to_pvalue.values() if p < alpha])}')
```

```
number of movies is significant: 7: 100%|██████████| 400/400 [01:56<00:00,  3.44it/s]
```

In [ ]:
```
cnt = 0
for movie in movie_to_pvalue:
    if movie_to_pvalue[movie] < 0.005:
        print(movie)
        cnt += 1
print(f'portion: {cnt/400}')
# print(f'{400-len(movie_to_pvalue)} does not have enough data')
```

```
The Evil Dead (1981)
Donnie Darko (2001)
Mulholland Dr. (2001)
Apocalypse Now (1979)
American History X (1998)
Midnight Cowboy (1969)
Fatal Attraction (1987)
portion: 0.0175
0 does not have enough data
```