In [1]:
```
import warnings

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.stats as stats
import seaborn as sns
```

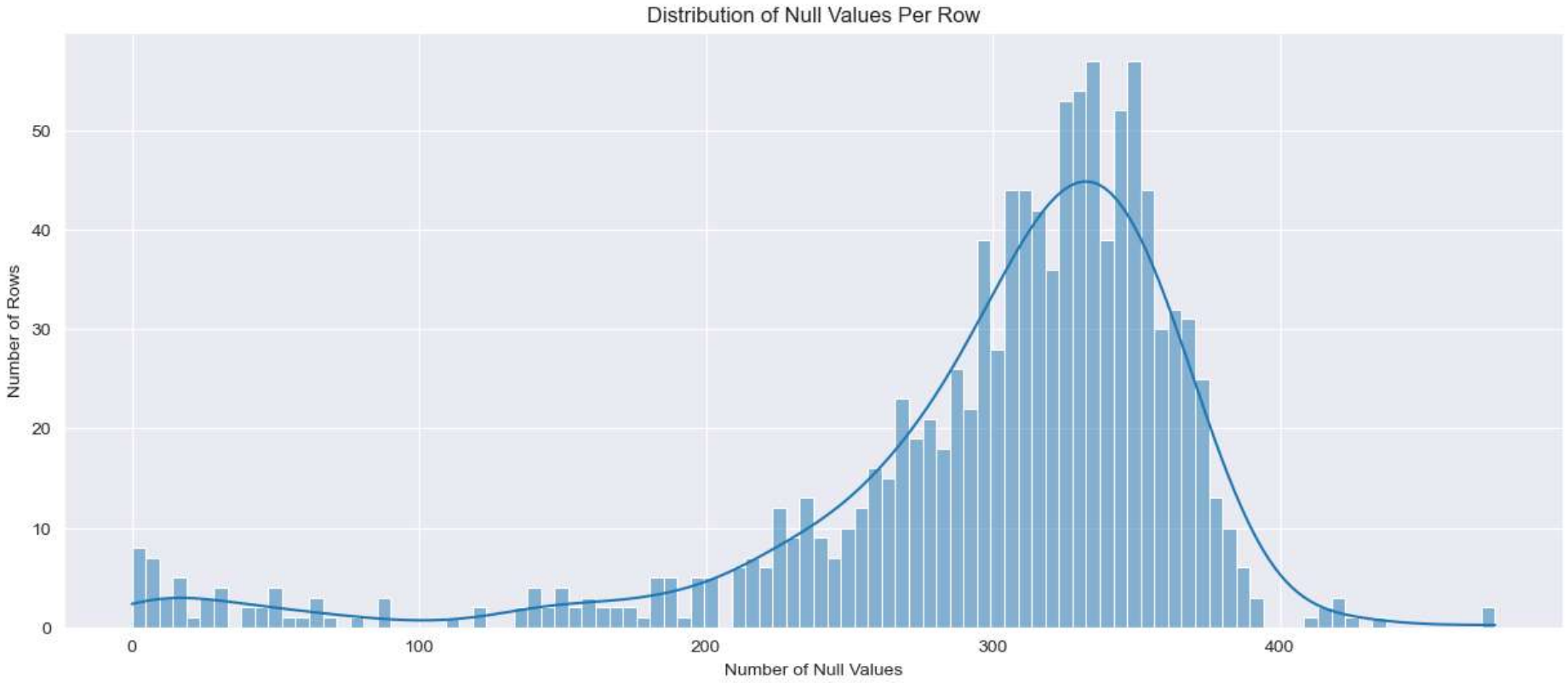
In [2]:
```
data = pd.read_csv('movieReplicationSet.csv')
num_rows = data.shape[1]
num_columns = data.shape[0]
print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_columns}")
```

```
Number of rows: 477
Number of columns: 1097
```

In [3]:
```
# null value analyze
null_counts_per_row = data.isnull().sum(axis=1)

# Analyze and print the number of rows with greater than a specific number of null values
thresholds = [100, 200, 300, 400]
for threshold in thresholds:
    count = len(null_counts_per_row[null_counts_per_row > threshold])
    print(f"Number of rows with more than {threshold} null values: {count}")

# Draw distribution chart
plt.figure(figsize=(15, 6))
sns.histplot(null_counts_per_row, kde=True, bins=100)
plt.title('Distribution of Null Values Per Row')
plt.xlabel('Number of Null Values')
plt.ylabel('Number of Rows')
plt.show()
```

```
Number of rows with more than 100 null values: 1048
Number of rows with more than 200 null values: 1003
Number of rows with more than 300 null values: 705
Number of rows with more than 400 null values: 10
```

```
C:\Users\YHD\.conda\envs\scientificProject\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\YHD\.conda\envs\scientificProject\lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [4]:
```
# 1) Are movies that are more popular (operationalized as having more ratings) rated higher than movies that are less popular? [Hin

# First calculate the number of non-missing ratings and average rating for each movie
movie_ratings = data.iloc[:, 0:400]
movie_popularity = movie_ratings.count()
movie_avg_ratings = movie_ratings.mean()

# Then perform a median split based on popularity
median_popularity = movie_popularity.median()
high_popularity_movies = movie_avg_ratings[movie_popularity > median_popularity]
median_popularity_movies = movie_avg_ratings[movie_popularity == median_popularity]
```

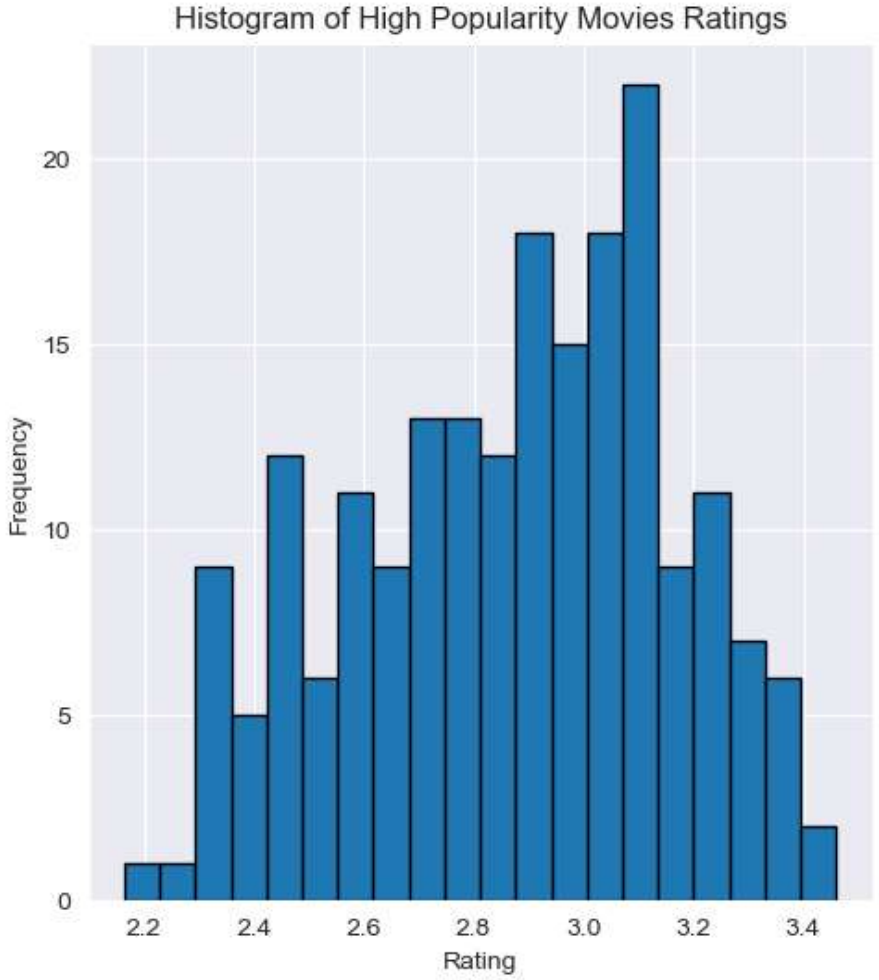
```
low_popularity_movies = movie_avg_ratings[movie_popularity < median_popularity]

# List the number of movies of each type
number_of_high_popularity_movies = len(high_popularity_movies)
number_of_median_popularity_movies = len(median_popularity_movies)
number_of_low_popularity_movies = len(low_popularity_movies)

print(
    f"Num of movie rate in high:{number_of_high_popularity_movies}"
    f", median: {number_of_median_popularity_movies}"
    f", low: {number_of_low_popularity_movies}")
```

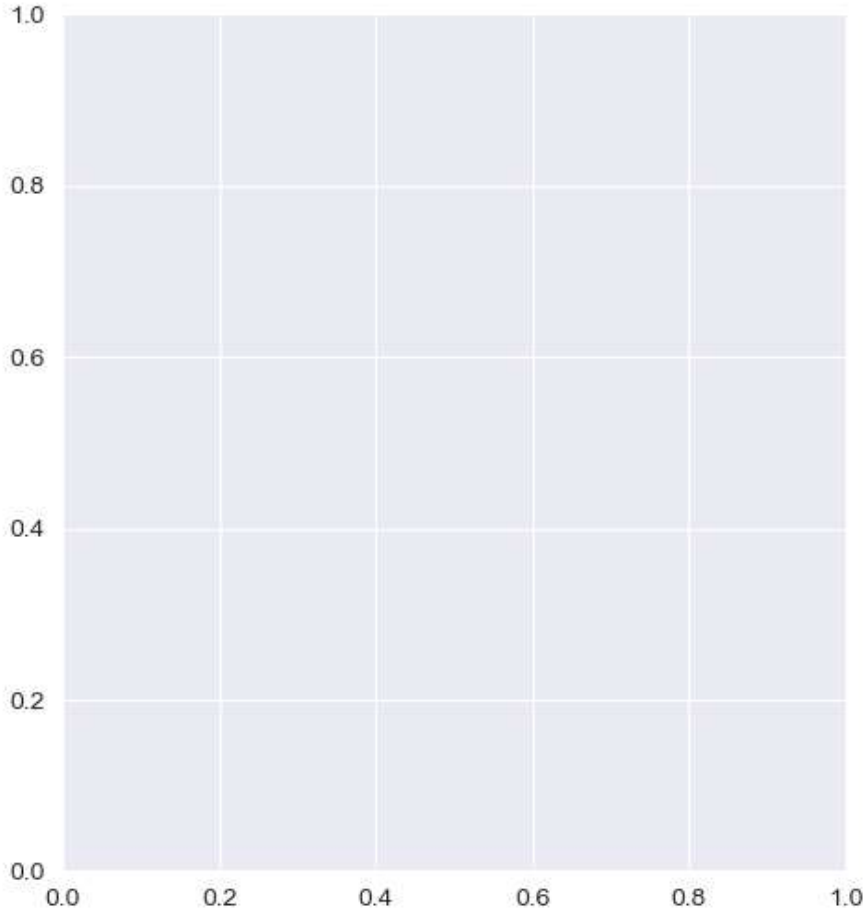
```
Num of movie rate in high:200, median: 0, low: 200
```

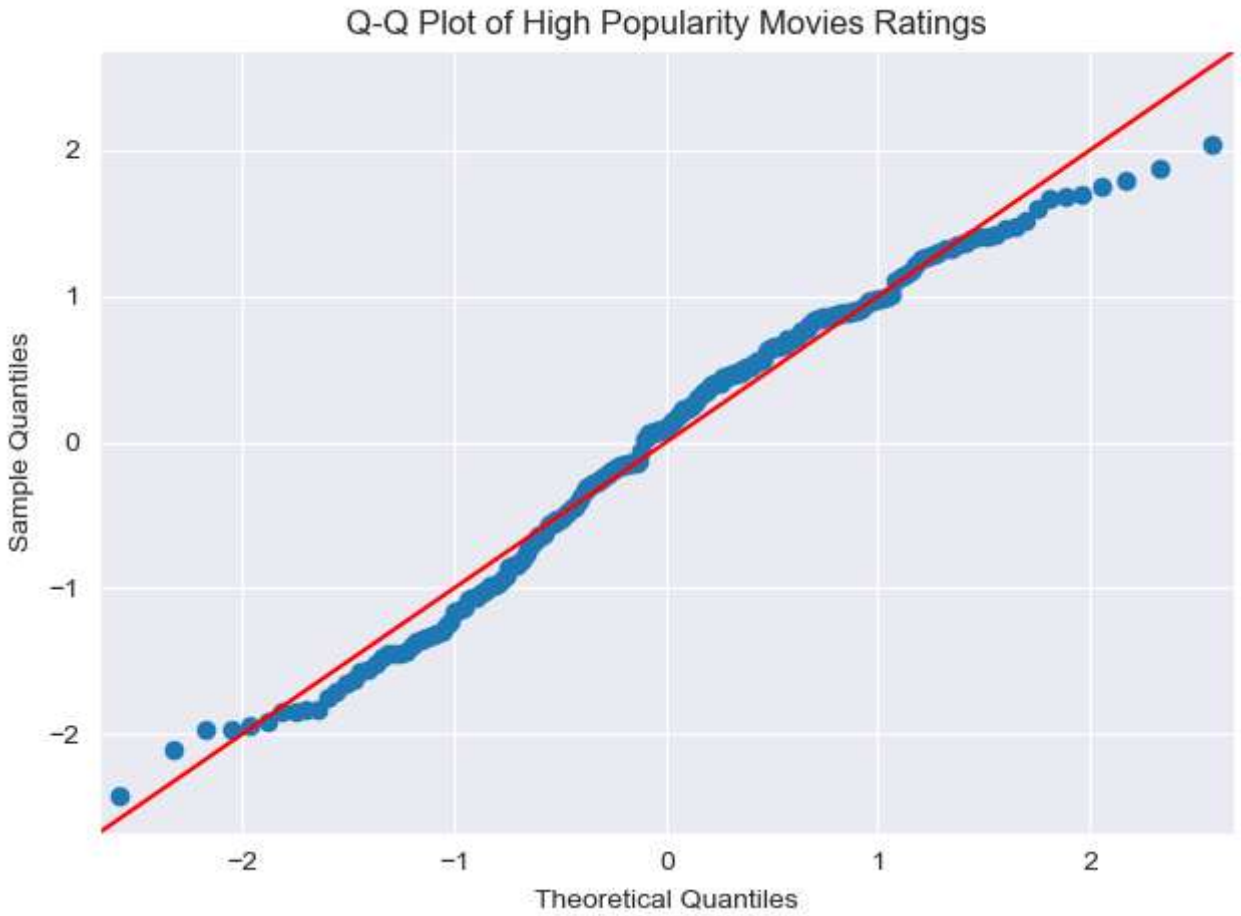
In [5]:

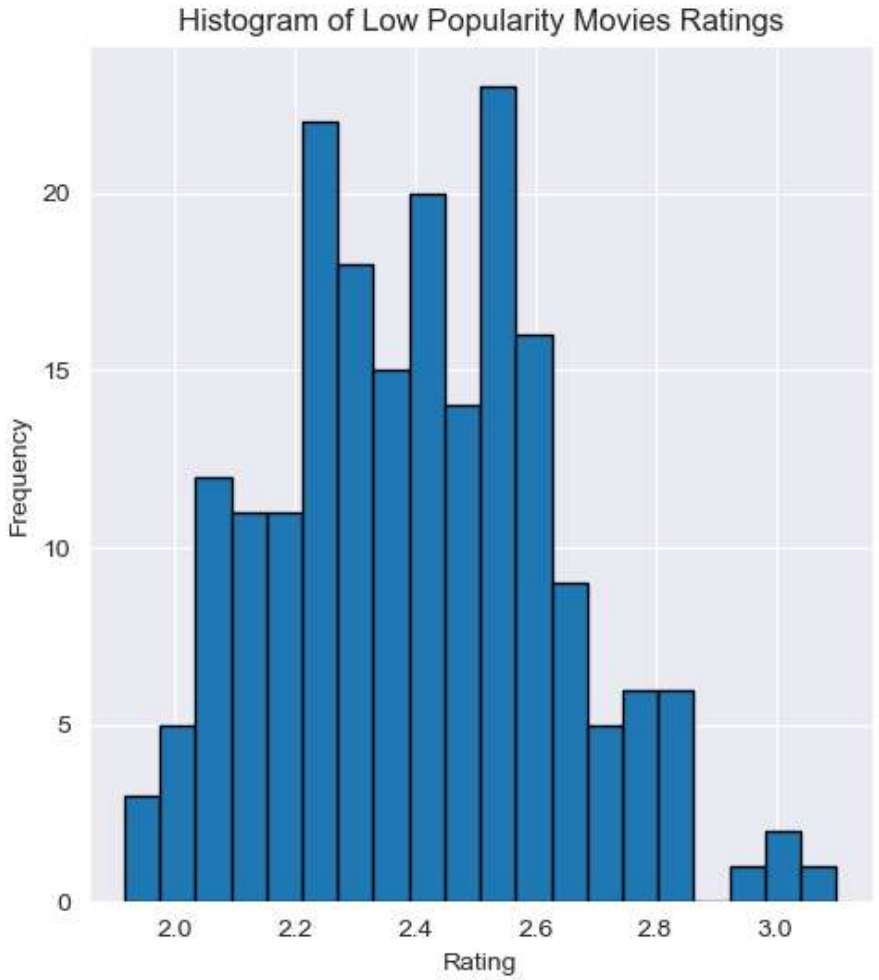
```
# Normal distribution image test
import statsmodels.api as sm

# Histograms and Q-Q plots of high-rating movies
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(high_popularity_movies, bins=20, edgecolor='k')
plt.title('Histogram of High Popularity Movies Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sm.qqplot(high_popularity_movies, line='45', fit=True)
plt.title('Q-Q Plot of High Popularity Movies Ratings')

plt.tight_layout()
plt.show()

# Histogram and Q-Q plot of low-rating movies
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(low_popularity_movies.dropna(), bins=20, edgecolor='k')
plt.title('Histogram of Low Popularity Movies Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sm.qqplot(low_popularity_movies.dropna(), line='45', fit=True)
```

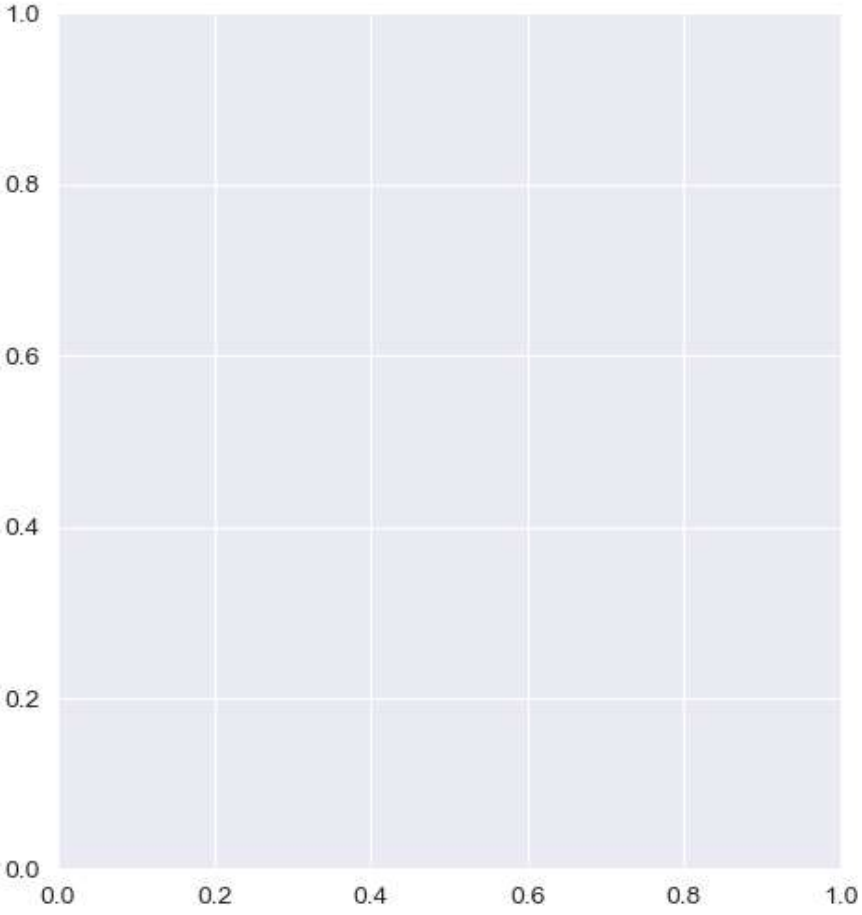
```
plt.title('Q-Q Plot of Low Popularity Movies Ratings')

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

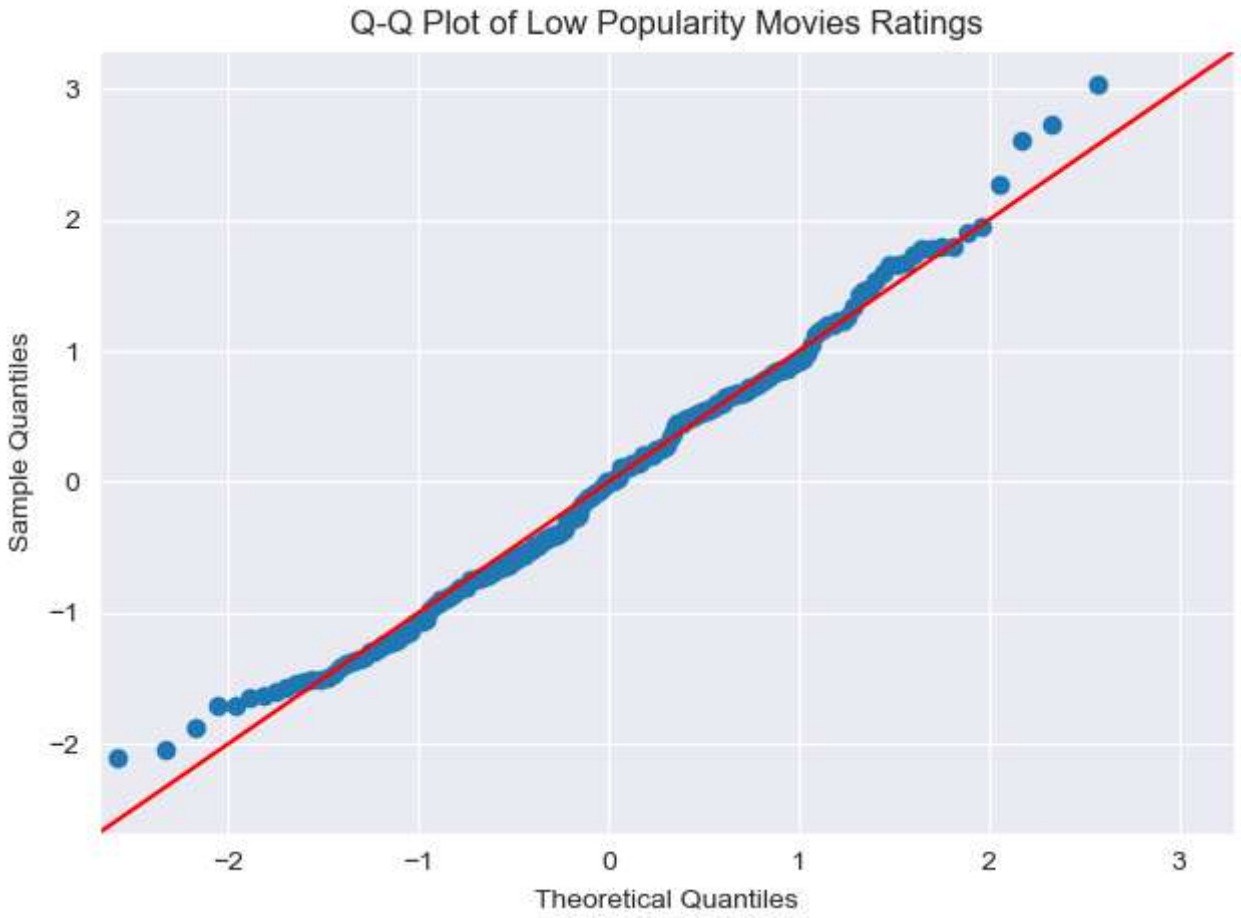





Q-Q Plot of High Popularity Movies Ratings
Sample Quantiles
Theoretical Quantiles
2
1
0
−1
−2
−2
−1
0
1
2


Histogram of Low Popularity Movies Ratings
Frequency
Rating
0
5
10
15
20
2.0
2.2
2.4
2.6
2.8
3.0


0.0
0.2
0.4
0.6
0.8
1.0



In [6]:
```
# Test whether the ratings of high-rated movies conform to the normal distribution
from scipy.stats import kstest

stat_high, p_high = kstest(high_popularity_movies, 'norm', args=(np.mean(high_popularity_movies), np.std(high_popularity_movies)))
if p_high > 0.005:
    print("High popularity movies' ratings appear to be normally distributed according to KS test.")
else:
    print("High popularity movies' ratings do not appear to be normally distributed according to KS test.")

# Test whether the ratings of low-rated movies follow a normal distribution
stat_low, p_low = kstest(low_popularity_movies.dropna(), 'norm', args=(np.mean(low_popularity_movies), np.std(low_popularity_movies
if p_low > 0.005:
    print("Low popularity movies' ratings appear to be normally distributed according to KS test.")
else:
```

```
    print("Low popularity movies' ratings do not appear to be normally distributed according to KS test.")

print("KS test p-value for high popularity movies: ", p_high)
print("KS test p-value for low popularity movies: ", p_low)
```

```
High popularity movies' ratings appear to be normally distributed according to KS test.
Low popularity movies' ratings appear to be normally distributed according to KS test.
KS test p-value for high popularity movies:  0.23863797760351713
KS test p-value for low popularity movies:  0.5713483383717397
```

In [28]:
```
# Variance calculation
variance_high = np.var(high_popularity_movies, ddof=1)
variance_low = np.var(low_popularity_movies, ddof=1)

print("High-pop variance:", variance_high)
print("Low-pop variance:", variance_low)
```

```
High-pop variance: 0.08500243213090868
Low-pop variance: 0.05357798836736422
```

In [8]:
```
# Welch t-test
from scipy.stats import ttest_ind

t_statistic, p_value = ttest_ind(high_popularity_movies, low_popularity_movies, equal_var=False)

print(f"Welch's T-test p-value: {p_value}")
print(f"Welch's T-test statistic: {t_statistic}")
```

```
Welch's T-test p-value: 9.536936590853455e-52
Welch's T-test statistic: 17.7560492698737
```

In [9]:
```
# Mann-whitney U test
from scipy.stats import mannwhitneyu

U, p_value = mannwhitneyu(high_popularity_movies, low_popularity_movies.dropna())

print(f"Mann-Whitney U test p-value: {p_value}")
print(f"Mann-Whitney U test U value: {U}")
```

```
Mann-Whitney U test p-value: 1.6971433120157929e-40
Mann-Whitney U test U value: 35404.0
```

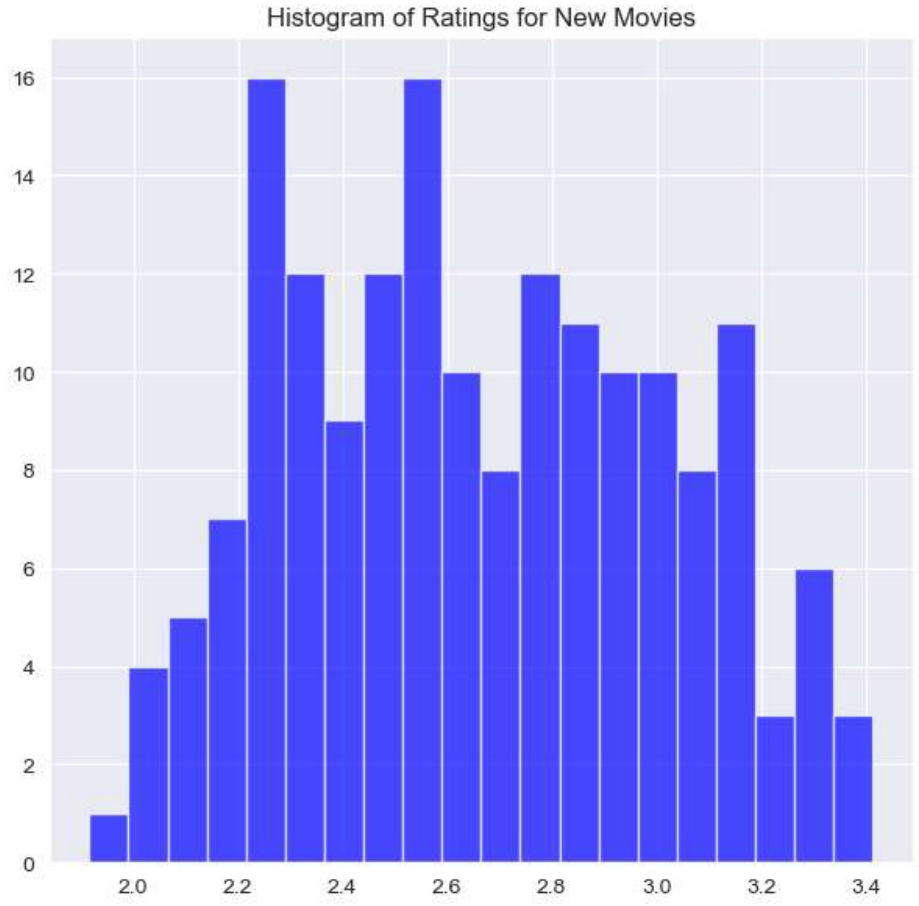
In [10]:
```
#-------------------------------------------------------------------------------------------
```

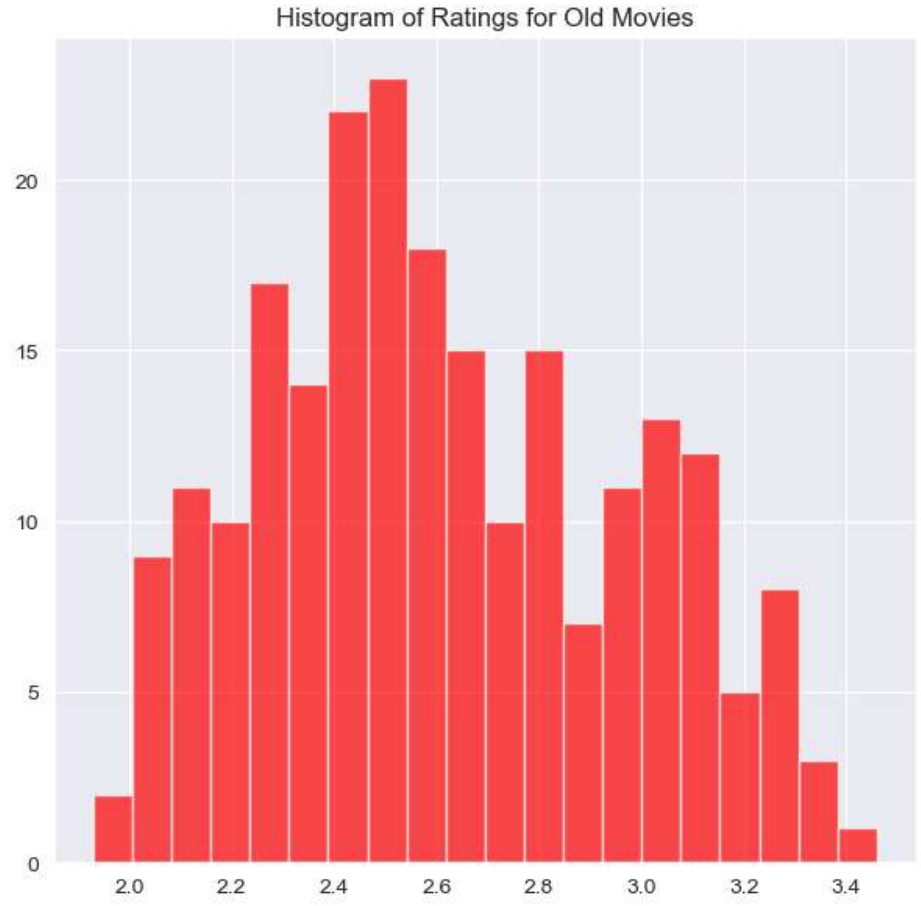
In [11]:
```
# 2) Are movies that are newer rated differently than movies that are older? [Hint: Do a median split of year of release to contras

# Extract the year from the movie title
years = data.columns[0:400].str.extract(r'\((\d{4})\)')[0].astype(int)

# Calculate the average rating of each movie
movie_ratings = data.iloc[:, 0:400]
movie_avg_ratings = movie_ratings.mean()

# Use the median of the years to divide movies into new and old movies
median_year = years.median()
new_movies = movie_avg_ratings[years.values > median_year]
old_movies = movie_avg_ratings[years.values <= median_year]
print(f"median year:{median_year}")
print(f"number of new movies:{len(new_movies)}")
print(f"number of old movies:{len(old_movies)}")
```

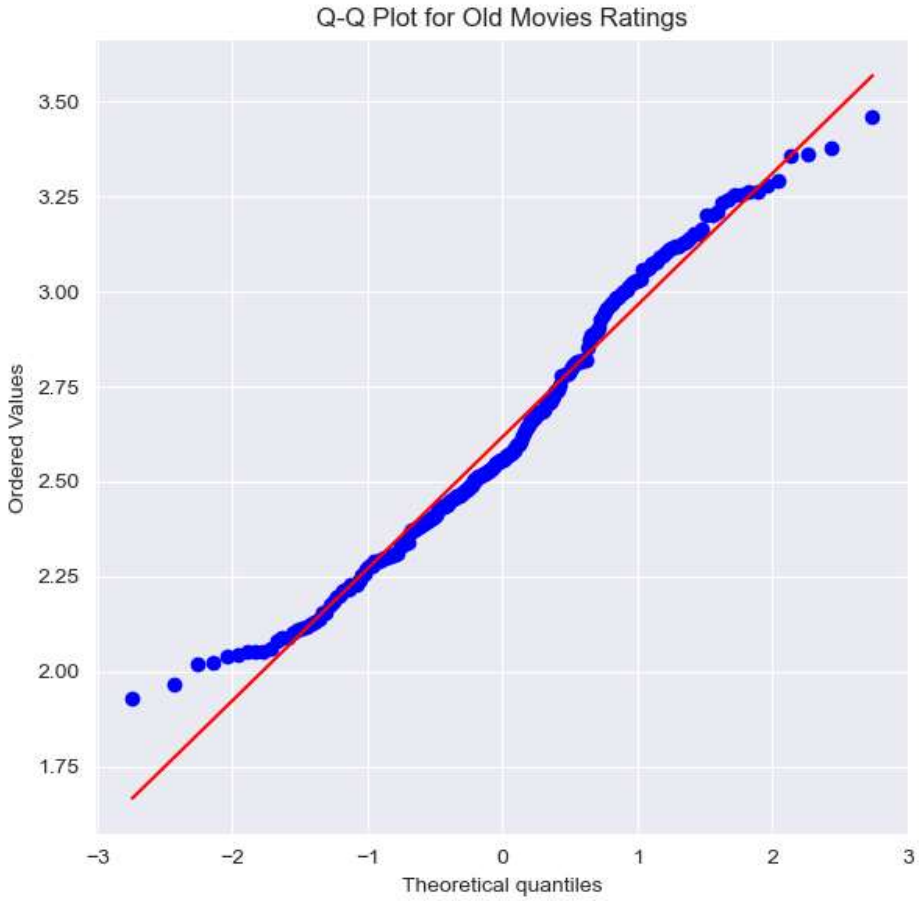
```
median year:1999.0
number of new movies:174
number of old movies:226
```

In [12]:
```
# Normal distribution image test

# Get all ratings for old and new movies
ratings_new_movies = movie_ratings.loc[:, years.values > median_year]
ratings_old_movies = movie_ratings.loc[:, years.values <= median_year]

# Convert the rating data of old and new movies into a 1D array and remove NaN values
new_ratings = ratings_new_movies.values.ravel()
new_ratings = new_ratings[~np.isnan(new_ratings)]

old_ratings = ratings_old_movies.values.ravel()
old_ratings = old_ratings[~np.isnan(old_ratings)]

# 1. Histogram
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(new_movies, bins=20, color='blue', alpha=0.7)
plt.title('Histogram of Ratings for New Movies')

plt.subplot(1, 2, 2)
plt.hist(old_movies, bins=20, color='red', alpha=0.7)
```

```
plt.title('Histogram of Ratings for Old Movies')

plt.tight_layout()
plt.show()

# 2. Q-Q plot
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
stats.probplot(new_movies, plot=plt)
plt.title('Q-Q Plot for New Movies Ratings')

plt.subplot(1, 2, 2)
stats.probplot(old_movies, plot=plt)
plt.title('Q-Q Plot for Old Movies Ratings')

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


Histogram of Ratings for New Movies
16
14
12
10
8
6
4
2
0
2.0
2.2
2.4
2.6
2.8
3.0
3.2
3.4


Histogram of Ratings for Old Movies
20
15
10
5
0
2.0
2.2
2.4
2.6
2.8
3.0
3.2
3.4



In [13]:
```
# Test whether the new movie's ratings conform to the normal distribution
stat_high, p_high = kstest(new_movies, 'norm', args=(np.mean(new_movies), np.std(new_movies)))
if p_high > 0.005:
    print("New movies' ratings appear to be normally distributed according to KS test.")
else:
    print("New movies' ratings do not appear to be normally distributed according to KS test.")

# Test whether the ratings of old movies conform to the normal distribution
stat_low, p_low = kstest(old_movies, 'norm', args=(np.mean(old_movies), np.std(old_movies)))
if p_low > 0.005:
    print("Old movies' ratings appear to be normally distributed according to KS test.")
else:
    print("Old movies' ratings do not appear to be normally distributed according to KS test.")
```

```
print("KS test p-value for high popularity movies: ", p_high)
print("KS test p-value for low popularity movies: ", p_low)
```

```
New movies' ratings appear to be normally distributed according to KS test.
Old movies' ratings appear to be normally distributed according to KS test.
KS test p-value for high popularity movies:  0.34658959977828174
KS test p-value for low popularity movies:  0.1240371505207325
```

In [14]:
```
# Variance calculation
variance_new = np.var(new_movies, ddof=1)
variance_old = np.var(old_movies, ddof=1)

print("New movies' ratings sample variance:", variance_new)
print("Old movies' ratings sample variance:", variance_old)
```

```
New movies' ratings sample variance: 0.127208880767958
Old movies' ratings sample variance: 0.121154119383467
```

In [15]:
```
# Measure skewness
from scipy.stats import skew

# Calculate the skewness of old and new movies
skew_new_movies = skew(new_movies)
skew_old_movies = skew(old_movies)

print(f"Skewness for newer movies ratings: {skew_new_movies}")
print(f"Skewness for older movies ratings: {skew_old_movies}")
```

```
Skewness for newer movies ratings: 0.13419152300817333
Skewness for older movies ratings: 0.2879637333714247
```

In [16]:
```
# t-test
from scipy.stats import ttest_ind

# Down sampling t-test
num_iterations = 100000
p_values = []

for i in range(num_iterations):
    sampled_old_movies = np.random.choice(old_movies, size=len(new_movies), replace=False)
    t_stat, p = ttest_ind(new_movies, sampled_old_movies, equal_var=True)
    p_values.append(p)

# Calculate the average of p-values
avg_p = np.mean(p_values)
```

```
print(f"Average P-value after {num_iterations} iterations: {avg_p}")

# Determine significance
if avg_p < 0.005:
    print("There is a significant difference in ratings between newer and randomly sampled older movies.")
else:
    print("There is no significant difference in ratings between newer and randomly sampled older movies.")
```

```
Average P-value after 100000 iterations: 0.3180344833316252
There is no significant difference in ratings between newer and randomly sampled older movies.
```

In [17]:
```
# Perform a down sampling Mann-Whitney U test
num_iterations = 100000
p_values = []

for _ in range(num_iterations):
    sampled_old_movies = np.random.choice(old_movies, size=len(new_movies), replace=False)
    _, p = mannwhitneyu(new_movies, sampled_old_movies, alternative='two-sided')
    p_values.append(p)

# Calculate the average of p-values
avg_p = np.mean(p_values)
print(f"Average P-value after {num_iterations} iterations: {avg_p}")

# Determine significance
if avg_p < 0.005:
    print("There is a significant difference in ratings between newer and randomly sampled older movies.")
else:
    print("There is no significant difference in ratings between newer and randomly sampled older movies.")
```

```
Average P-value after 100000 iterations: 0.3153290160792952
There is no significant difference in ratings between newer and randomly sampled older movies.
```

In [18]:
```
#-----------------------------------------------------------------------------------------------
```

In [19]:
```
# Preliminary analysis of the third question
# Count the number of values 1 in column C475
count_1 = (data['Gender identity (1 = female; 2 = male; 3 = self-described)'] == 1).sum()

# Count the number of values 2 in column C475
count_2 = (data['Gender identity (1 = female; 2 = male; 3 = self-described)'] == 2).sum()

print(f'In column C475, there are {count_1} of data with a value of 1, and {count_2} of data with a value of 2.')
```

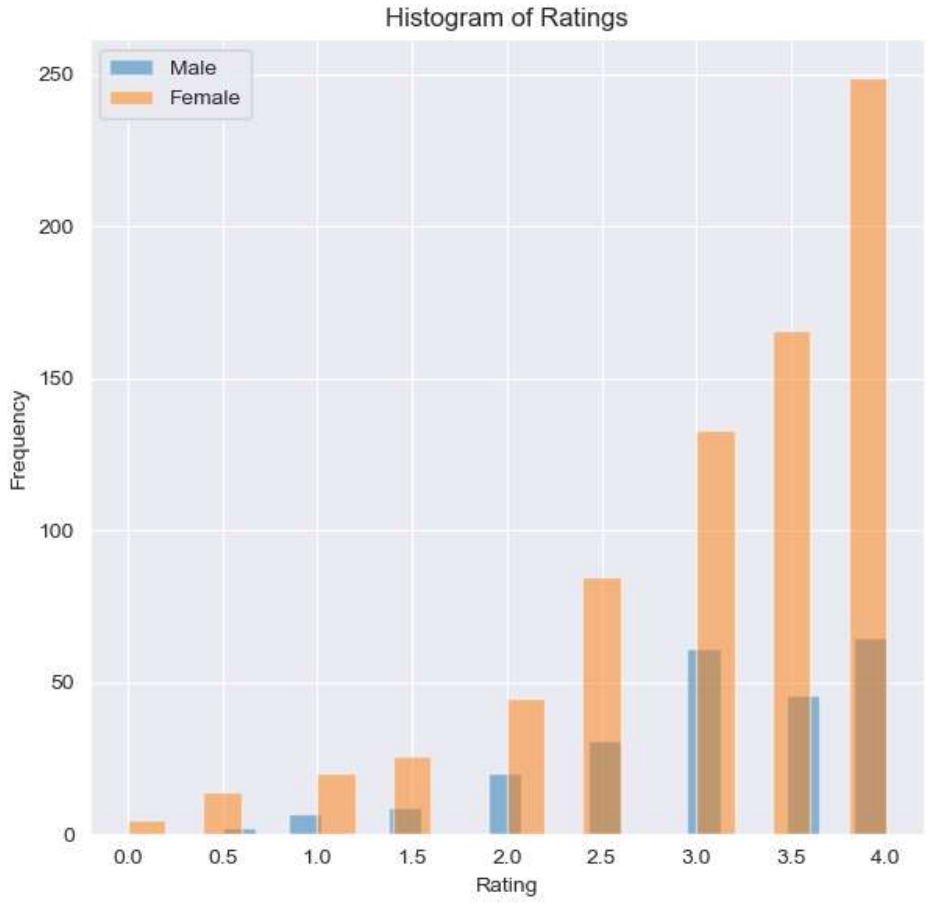
In column C475, there are 807 of data with a value of 1, and 260 of data with a value of 2.

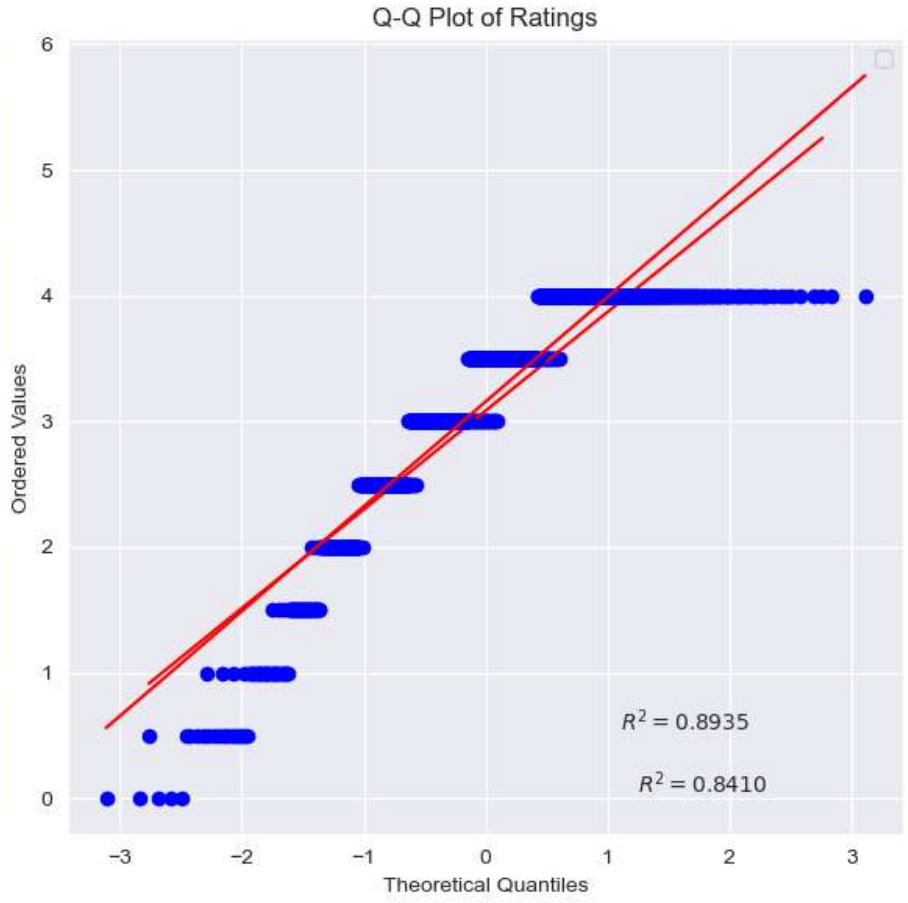
In [20]:
```
# 3) Is enjoyment of 'Shrek (2001)' gendered, i.e. do male and female viewers rate it differently?
# Delete rows with null values for gender
filtered_data = data.dropna(subset=['Gender identity (1 = female; 2 = male; 3 = self-described)', 'Shrek (2001)'])

# Extract the ratings of Shrek (2001)
shrek_ratings = filtered_data['Shrek (2001)']

# Divide ratings based on gender
male_ratings = shrek_ratings[filtered_data['Gender identity (1 = female; 2 = male; 3 = self-described)'] == 2]
female_ratings = shrek_ratings[filtered_data['Gender identity (1 = female; 2 = male; 3 = self-described)'] == 1]
print(f"male: {len(male_ratings)},female:{len(female_ratings)}")
```

```
male: 241,female:743
```

In [21]:
```
# Normality graphical analysis
# Histogram
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(male_ratings, bins=20, alpha=0.5, label='Male')
plt.hist(female_ratings, bins=20, alpha=0.5, label='Female')
plt.title('Histogram of Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.legend()

# Q-Q plot
plt.subplot(1, 2, 2)
stats.probplot(male_ratings, dist="norm", plot=plt, rvalue=True)
stats.probplot(female_ratings, dist="norm", plot=plt, rvalue=True)
plt.title('Q-Q Plot of Ratings')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Ordered Values')
plt.legend()
plt.tight_layout()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
```





In [22]:
```
# 2. Kolmogorov-Smirnov test
male_stat, male_p = stats.kstest(male_ratings, 'norm', args=(np.mean(male_ratings), np.std(male_ratings)))
female_stat, female_p = stats.kstest(female_ratings, 'norm', args=(np.mean(female_ratings), np.std(female_ratings)))

print(f"Kolmogorov-Smirnov Test for Male Ratings: Statistic:{male_stat}, p-value:{male_p}")
print(f"Kolmogorov-Smirnov Test for Female Ratings: Statistic:{female_stat}, p-value:{female_p}")
```

```
Kolmogorov-Smirnov Test for Male Ratings: Statistic:0.17354623877694658, p-value:8.103899310077524e-07
Kolmogorov-Smirnov Test for Female Ratings: Statistic:0.2066938896741966, p-value:2.5678033507710075e-28
```

In [23]:
```
# Down sampling and perform Mann-whitney U test
from scipy.stats import mannwhitneyu

iterations = 100000
p_values = []
```

```
for _ in range(iterations):
    # Down sampling
    sampled_female_ratings = filtered_data[
        filtered_data['Gender identity (1 = female; 2 = male; 3 = self-described)'] == 1].sample(n=241)
    male_ratings_data = filtered_data[filtered_data['Gender identity (1 = female; 2 = male; 3 = self-described)'] == 2]

    male_ratings = male_ratings_data['Shrek (2001)']
    female_ratings = sampled_female_ratings['Shrek (2001)']
    # Mann-Whitney U test
    _, p = mannwhitneyu(male_ratings, female_ratings, alternative='two-sided')
    p_values.append(p)

# 计算平均p值
average_p_value = np.mean(p_values)

print(f"Average p-value after {iterations} iterations: {average_p_value}")
```

```
Average p-value after 100000 iterations: 0.17349935743209838
```

In [24]:
```
#-------------------------------------------------------------------------------------------------
```

In [27]:
```
# 4) What proportion of movies are rated differently by male and female viewers?
# CAUTION: running this cell with 10000 iterations will take you approximate 25 minutes
import random

alpha = 0.005

movies = list(data.columns)[0:400]

# Variable for counting
has_diff_counter = 0
no_diff_counter = 0
detailed_results = {}

# Analyze each movie
for movie in movies:
    filtered_data = data.dropna(subset=['Gender identity (1 = female; 2 = male; 3 = self-described)', movie])

    # Extract the rating of the movie
    ratings = filtered_data[movie]

    # Divide ratings based on gender
    male_ratings = ratings[filtered_data['Gender identity (1 = female; 2 = male; 3 = self-described)'] == 2]
```

```
    female_ratings = ratings[filtered_data['Gender identity (1 = female; 2 = male; 3 = self-described)'] == 1]

    # 2. Kolmogorov-Smirnov test
    male_stat, male_p = stats.kstest(male_ratings, 'norm', args=(np.mean(male_ratings), np.std(male_ratings)))
    female_stat, female_p = stats.kstest(female_ratings, 'norm', args=(np.mean(female_ratings), np.std(female_ratings)))

    # Make sure the two sets of samples are equal in size
    if len(male_ratings) > len(female_ratings):
        sampled_male_ratings = random.sample(list(male_ratings), len(female_ratings))
    else:
        sampled_female_ratings = random.sample(list(female_ratings), len(male_ratings))

    # 3. Select the inspection method and perform inspection
    p_values = []
    if male_p > alpha and female_p > alpha:
        # If both sets of data conform to normal distribution, use t test
        for _ in range(10000):
            if len(male_ratings) > len(female_ratings):
                _, p_value = stats.ttest_ind(sampled_male_ratings, female_ratings, equal_var=False)
            else:
                _, p_value = stats.ttest_ind(male_ratings, sampled_female_ratings, equal_var=False)
            p_values.append(p_value)
    else:
        # If the data does not conform to the normal distribution, use the Mann-Whitney U test
        for _ in range(10000):
            if len(male_ratings) > len(female_ratings):
                _, p_value = stats.mannwhitneyu(sampled_male_ratings, female_ratings, alternative='two-sided')
            else:
                _, p_value = stats.mannwhitneyu(male_ratings, sampled_female_ratings, alternative='two-sided')
            p_values.append(p_value)

    # Calculate the average p-value
    average_p_value = np.mean(p_values)

    # Determine whether there is a significant difference
    if average_p_value < alpha:
        has_diff_counter += 1
        detailed_results[movie] = 'Significant Difference'
    else:
        no_diff_counter += 1
        detailed_results[movie] = 'No Significant Difference'

# 打印结果
print(f"Movies with significant gender differences in ratings: {has_diff_counter}")
```

```
print(f"Movies without significant gender differences in ratings: {no_diff_counter}")
print("Detailed results:")
# for movie, result in detailed_results.items():
    # print(f"{movie}: {result}")
```

```
Movies with significant gender differences in ratings: 34
Movies without significant gender differences in ratings: 366
Detailed results:
```

In [26]:
```
#----------------------------------------------------------------------------------------
```