

In [12]: `import pandas as pd`

In [13]: `file_path = 'movieReplicationSet.csv'`
`df = pd.read_csv(file_path)`
`df`

Out[13]:

The Life of David Gale (2003)	Wing Commander (1999)	Django Unchained (2012)	Alien (1979)	Indiana Jones and the Last Crusade (1989)	Snatch (2000)	Rambo: First Blood Part II (1985)	Fargo (1996)	Let the Right One In (2008)	Black Swan (2010)	...
-------------------------------	-----------------------	-------------------------	--------------	---	---------------	-----------------------------------	--------------	-----------------------------	-------------------	-----

0	NaN	NaN	4.0	NaN	3.0	NaN	NaN	NaN	NaN	NaN	...
1	NaN	NaN	1.5	NaN	...						
2	NaN	...									
3	NaN	NaN	2.0	NaN	3.0	NaN	NaN	NaN	NaN	4.0	...
4	NaN	NaN	3.5	NaN	0.5	NaN	0.5	1.0	NaN	0.0	...
...
1092	NaN	NaN	NaN	NaN	3.5	NaN	NaN	NaN	NaN	NaN	...
1093	3.0	4.0	NaN	NaN	4.0	4.0	2.5	NaN	3.5	3.5	...
1094	NaN	3.5	NaN	NaN	...						
1095	NaN	...									
1096	NaN	NaN	4.0	NaN	2.5	NaN	NaN	3.0	NaN	3.5	...

1097 rows × 477 columns

In [14]: `#9`

```
In [15]: # Extract ratings for 'Home Alone (1990)' and 'Finding Nemo (2003)'  
home_alone_ratings = df['Home Alone (1990)'].dropna()  
finding_nemo_ratings = df['Finding Nemo (2003)'].dropna()  
  
# Display basic statistics and the first few ratings of each movie  
home_alone_ratings.describe(), home_alone_ratings.head(), finding_nemo_r
```

```
Out[15]: (count    857.000000  
          mean     3.130105  
          std      0.909287  
          min      0.000000  
          25%     2.500000  
          50%     3.500000  
          75%     4.000000  
          max     4.000000  
          Name: Home Alone (1990), dtype: float64,  
          0      4.0  
          1      4.0  
          2      4.0  
          3      1.5  
          4      2.0  
          Name: Home Alone (1990), dtype: float64,  
          count   1014.000000  
          mean     3.388067  
          std      0.788331  
          min      0.000000  
          25%     3.000000  
          50%     3.500000  
          75%     4.000000  
          max     4.000000  
          Name: Finding Nemo (2003), dtype: float64,  
          0      3.5  
          1      4.0  
          2      3.5  
          3      2.5  
          4      2.5  
          Name: Finding Nemo (2003), dtype: float64)
```

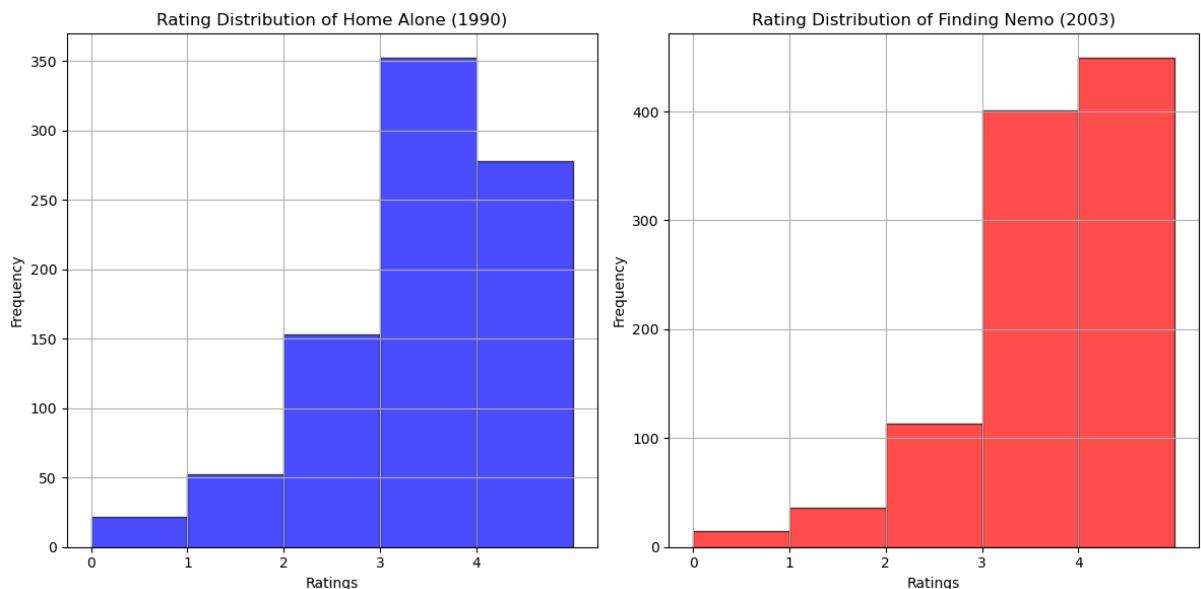
```
In [16]: import matplotlib.pyplot as plt

# Create histograms for the ratings of 'Home Alone (1990)' and 'Finding Nemo (2003)'
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(home_alone_ratings, bins=range(6), alpha=0.7, color='blue', edgecolor='black')
plt.title('Rating Distribution of Home Alone (1990)')
plt.xlabel('Ratings')
plt.ylabel('Frequency')
plt.xticks(range(5))
plt.grid(True)

plt.subplot(1, 2, 2)
plt.hist(finding_nemo_ratings, bins=range(6), alpha=0.7, color='red', edgecolor='black')
plt.title('Rating Distribution of Finding Nemo (2003)')
plt.xlabel('Ratings')
plt.ylabel('Frequency')
plt.xticks(range(5))
plt.grid(True)

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



```
In [17]: home_alone_mean = home_alone_ratings.mean()
home_alone_std = home_alone_ratings.std()
finding_nemo_mean = finding_nemo_ratings.mean()
finding_nemo_std = finding_nemo_ratings.std()

home_alone_mean, home_alone_std, finding_nemo_mean, finding_nemo_std
```

Out[17]: (3.1301050175029173, 0.9092870336253511, 3.388067061143984, 0.7883314952083654)

```
In [18]: from scipy.stats import ks_2samp

# Conducting the KS test
ks_statistic, p_value = ks_2samp(home_alone_ratings, finding_nemo_rating)

ks_statistic, p_value
```

```
Out[18]: (0.15269080020897632, 6.379397182836346e-10)
```

```
In [19]: from scipy.stats import chi2_contingency

# Creating a contingency table
contingency_table = pd.crosstab(index=[home_alone_ratings, finding_nemo_
                                         columns='count', margins=True, margins_name='Total')

# Calculating the chi-square statistic
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Displaying the contingency table, chi-square result, p-value, and degrees of freedom
contingency_table, chi2, p, dof
```

```
Out[19]: (col_0
           Home Alone (1990) Finding Nemo (2003)
           0.0          0.0              2      2
                         0.5              1      1
                         2.0              1      1
                         3.0              1      1
                         3.5              1      1
...
           4.0          2.5              4      4
                         3.0             21     21
                         3.5             51     51
                         4.0            171    171
           Total                  810    810
[69 rows x 2 columns],
0.0,
1.0,
68)
```

```
In [26]: # Categorizing the ratings into bins
home_alone_ratings_binned = pd.cut(home_alone_ratings, bins=[-0.1, 1, 2,
finding_nemo_ratings_binned = pd.cut(finding_nemo_ratings, bins=[-0.1, 1

# Creating the contingency table
contingency_table_binned = pd.DataFrame({
    'Home Alone (1990)': home_alone_ratings_binned.value_counts(sort=False),
    'Finding Nemo (2003)': finding_nemo_ratings_binned.value_counts(sort=False)
})

# Displaying the binned contingency table
contingency_table_binned
```

Out[26]:

	Home Alone (1990)	Finding Nemo (2003)
0-1	44	31
1.5-2	89	57
2.5-3	270	234
3.5-4	454	692

```
In [30]: # Performing the Chi-square test on the binned contingency table
chi2_stat_binned, p_value_binned, dof_binned, expected_freq_binned = chi2_contingency(contingency_table_binned)

# Displaying the results of the Chi-square test on the binned contingency table
chi2_stat_binned, p_value_binned, dof_binned, expected_freq_binned
```

Out[30]: (48.432825506789584,
1.722502856649771e-10,
3,
array([[34.35328701, 40.64671299],
[66.87439872, 79.12560128],
[230.85408872, 273.14591128],
[524.91822555, 621.08177445]]))

```
In [29]: # Creating a contingency table with the frequency of each rating for both movies
contingency_table = pd.crosstab(index=[home_alone_ratings], columns=[finding_nemo_ratings])
contingency_table_observed = contingency_table.iloc[:-1, :-1]

# Performing the chi-square test
chi2, p, dof, expected = chi2_contingency(contingency_table_observed)

contingency_table, chi2, p, dof
```

```
Out[29]: (Finding Nemo (2003)  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  Total
   Home Alone (1990)
   0.0               2    1    0    0    1    0    1    1    1    3
   0.5               0    0    0    1    2    2    1    4    1    1
   1.0               0    1    3    0    4    2    4    2    6    2
   1.5               0    0    1    2    1    4    8    6    6    2
   2.0               0    1    1    1    3    8   13   11   16   5
   2.5               0    1    2    3    5   12   14   24   29   9
   3.0               1    1    3    5    3   15   36   45   61  17
   3.5               0    1    1    1    2   13   24   50   75  16
   4.0               2    3    2    2    3    4   21   51  171  25
   Total             5    9   13   15   24   60  122  194  368  81
   ,
   251.3516891163737,
   5.046751157619788e-24,
   64)
```

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

```
In [ ]: # Selecting the specified columns
columns_to_show = list(df.columns[464:475])
# Displaying the selected columns
df[columns_to_show].head()
```

```
In [16]: # Define the keywords for the franchises
franchises_keywords = ['Star Wars', 'Harry Potter', 'The Matrix', 'Indiana Jones', 'Jurassic Park', 'Pirates of the Caribbean', 'Toy Story', 'Batman']

# Filter the columns based on the franchises keywords
filtered_columns = [col for col in df.columns if any(keyword in col for keyword in franchises_keywords)]

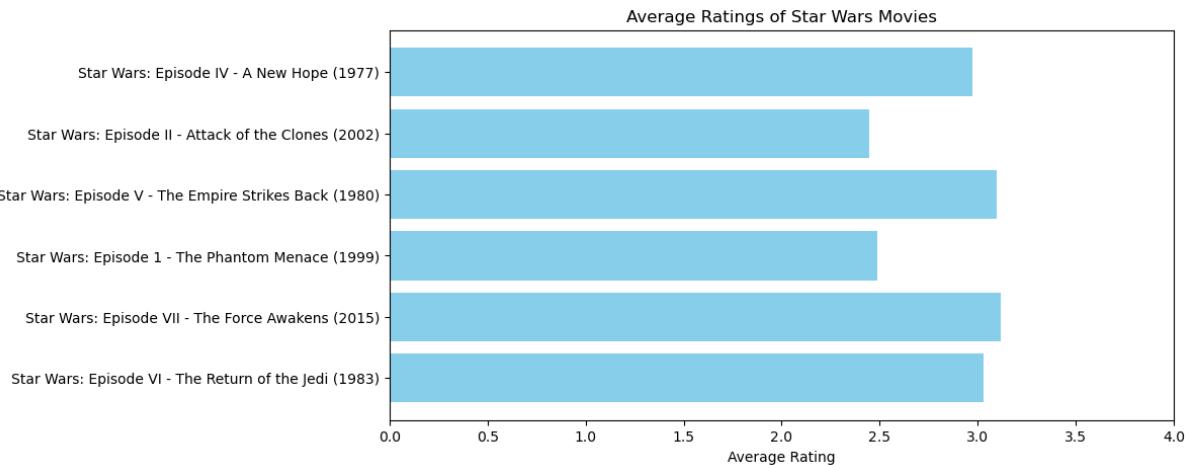
# Create a new dataframe with only the filtered columns
df_franchises = df[filtered_columns]

# Define the franchises and their movies
franchises_movies = {
    'Star Wars': [col for col in filtered_columns if 'Star Wars' in col],
    'Harry Potter': [col for col in filtered_columns if 'Harry Potter' in col],
    'The Matrix': [col for col in filtered_columns if 'The Matrix' in col],
    'Indiana Jones': [col for col in filtered_columns if 'Indiana Jones' in col],
    'Jurassic Park': [col for col in filtered_columns if 'Jurassic Park' in col],
    'Pirates of the Caribbean': [col for col in filtered_columns if 'Pirates of the Caribbean' in col],
    'Toy Story': [col for col in filtered_columns if 'Toy Story' in col],
    'Batman': [col for col in filtered_columns if 'Batman' in col],
}

# Function to plot the bar charts for each franchise
def plot_franchise_ratings(franchise, movies):
    # Calculate the mean ratings
    mean_ratings = [df_franchises[movie].mean() for movie in movies]

    # Create the bar chart
    plt.figure(figsize=(10, 5))
    plt.barh(movies, mean_ratings, color='skyblue')
    plt.xlabel('Average Rating')
    plt.title(f'Average Ratings of {franchise} Movies')
    plt.xlim(0, 4) # As ratings are from 0 to 4
    plt.gca().invert_yaxis() # Invert y-axis to have the highest rating at the top
    plt.show()

# Plotting the bar charts for each franchise
for franchise, movies in franchises_movies.items():
    plot_franchise_ratings(franchise, movies)
```



```
In [30]: from scipy.stats import f_oneway

# Function to perform ANOVA and return the result
def perform_anova(franchise, movies):
    # Extracting the movie ratings and dropping NaN values
    movie_ratings = [df[movie].dropna() for movie in movies]
    # Performing ANOVA
    f_stat, p_value = f_oneway(*movie_ratings)
    return f_stat, p_value

# Performing ANOVA for each franchise and storing the results
anova_results = {franchise: perform_anova(franchise, movies)
                 for franchise, movies in franchises_movies.items()}
# Display the ANOVA results (F-statistic and p-value) for each franchise
anova_results
```

```
Out[30]: {'Star Wars': (45.645133146545426, 1.5252665421920376e-45),
          'Harry Potter': (0.7729869142003183, 0.5089850679527991),
          'The Matrix': (25.07705929547676, 2.1374604702645577e-11),
          'Indiana Jones': (14.566513091640147, 2.2613291345737005e-09),
          'Jurassic Park': (22.716093347500227, 1.8386566379299737e-10),
          'Pirates of the Caribbean': (9.672049744576752, 6.582065023243829e-05),
          'Toy Story': (7.6709305226197095, 0.0004763856415114884),
          'Batman': (108.26045119136043, 1.538339512760035e-44)}
```

```
In [21]: movie_ratings = df[movie_columns]
# Calculating the standard deviation for each movie in the franchises
std_deviations = {}
for franchise, movies in franchises_movies.items():
    std_deviations[franchise] = {}
    for movie in movies:
        std_deviations[franchise][movie] = movie_ratings[movie].std()

std_deviations
```

```
Out[21]: {'Star Wars': {'Star Wars: Episode IV – A New Hope (1977)': 1.0236032197259226,
 'Star Wars: Episode II – Attack of the Clones (2002)': 1.0549107155636193,
 'Star Wars: Episode V – The Empire Strikes Back (1980)': 0.9386756209067411,
 'Star Wars: Episode 1 – The Phantom Menace (1999)': 1.1037779280765565,
 'Star Wars: Episode VII – The Force Awakens (2015)': 0.947812870483516,
 'Star Wars: Episode VI – The Return of the Jedi (1983)': 0.9763210724689464},
 'Harry Potter': {"Harry Potter and the Sorcerer's Stone (2001)": 0.842393377300038,
 'Harry Potter and the Deathly Hallows: Part 2 (2011)': 0.8612575956349633,
 'Harry Potter and the Goblet of Fire (2005)': 0.8357480028722679,
 'Harry Potter and the Chamber of Secrets (2002)': 0.8753347870447454},
 'The Matrix': {'The Matrix Revolutions (2003)': 1.0391402580050462,
 'The Matrix Reloaded (2003)': 1.0079596157905715,
 'The Matrix (1999)': 0.8689258940699982},
 'Indiana Jones': {'Indiana Jones and the Last Crusade (1989)': 0.906993410218155,
 'Indiana Jones and the Temple of Doom (1984)': 0.9080375677978448,
 'Indiana Jones and the Raiders of the Lost Ark (1981)': 0.9214539331184456,
 'Indiana Jones and the Kingdom of the Crystal Skull (2008)': 1.0139638557339272},
 'Jurassic Park': {'The Lost World: Jurassic Park (1997)': 0.8752658148953424,
 'Jurassic Park III (2001)': 0.9615426331928156,
 'Jurassic Park (1993)': 0.8804049778989719},
 'Pirates of the Caribbean': {"Pirates of the Caribbean: Dead Man's Chest (2006)": 0.9558539491134094,
 "Pirates of the Caribbean: At World's End (2007)": 0.9783845406924503,
 'Pirates of the Caribbean: The Curse of the Black Pearl (2003)': 0.8994336017067975},
 'Toy Story': {'Toy Story 2 (1999)': 0.8561758474090498,
 'Toy Story 3 (2010)': 0.8479545279716212,
 'Toy Story (1995)': 0.8575467532153276},
 'Batman': {'Batman & Robin (1997)': 1.0985331383547257,
 'Batman (1989)': 0.9440192305509598,
 'Batman: The Dark Knight (2008)': 0.86644376787407}}
```

```
In [22]: # Calculating the mean standard deviation across all movies in the specific franchise
all_std_deviations = [std for franchise_std in std_deviations.values() if franchise_std != 0]
mean_std_deviation = sum(all_std_deviations) / len(all_std_deviations)

# Identifying movies with inconsistent quality (std_dev > mean_std_deviation)
inconsistent_movies = {}
for franchise, movies_std in std_deviations.items():
    inconsistent_movies[franchise] = [movie for movie, std in movies_std.items() if std > mean_std_deviation]

# Counting the number of inconsistent movies in each franchise
inconsistent_count = {franchise: len(movies) for franchise, movies in inconsistent_movies.items()}

mean_std_deviation, inconsistent_count
```

```
Out[22]: (0.9361333968167878,
{'Star Wars': 6,
'Harry Potter': 0,
'The Matrix': 2,
'Indiana Jones': 1,
'Jurassic Park': 1,
'Pirates of the Caribbean': 2,
'Toy Story': 0,
'Batman': 2})
```

```
In [ ]: #People with higher empathy for movies rate movies higher.
#Also compared to all 400 movies listed, franchises movies have a lower rating.
```

```
In [8]: for column in df.columns:
    if "when watching a movie i feel like the things on the screen are having an effect on my experience" in column:
        experience_column = column
```

```
In [9]: # Define the franchises and movies
franchises = ['Star Wars', 'Harry Potter', 'The Matrix', 'Indiana Jones',
              'Jurassic Park', 'Pirates of the Caribbean', 'Toy Story',

# Dictionary to store the correlation results
correlation_results = {}

# For each franchise, find the related movies and calculate the correlation
for franchise in franchises:
    for column in df.columns:
        if franchise.lower() in column.lower():
            # Dropping rows where either of the columns has a missing value
            df_filtered = df[[column, experience_column]].dropna()

            # Calculating the Pearson correlation coefficient
            correlation = df_filtered[column].corr(df_filtered[experience_column])

            # Storing the results
            correlation_results[column] = {
                'correlation': correlation,
                'non_missing_pairs': len(df_filtered)
            }

correlation_results
```

```
Out[9]: {'Star Wars: Episode IV – A New Hope (1977)': {'correlation': 0.03164284776718563, 'non_missing_pairs': 536}, 'Star Wars: Episode II – Attack of the Clones (2002)': {'correlation': 0.10482922994201317, 'non_missing_pairs': 518}, 'Star Wars: Episode V – The Empire Strikes Back (1980)': {'correlation': 0.027563853779713297, 'non_missing_pairs': 486}, 'Star Wars: Episode I – The Phantom Menace (1999)': {'correlation': 0.10149636026196124, 'non_missing_pairs': 449}, 'Star Wars: Episode VII – The Force Awakens (2015)': {'correlation': 0.047789070561897753, 'non_missing_pairs': 484}, 'Star Wars: Episode VI – The Return of the Jedi (1983)': {'correlation': 0.04530440130016709, 'non_missing_pairs': 452}, "Harry Potter and the Sorcerer's Stone (2001)": {'correlation': 0.07454912798649307, 'non_missing_pairs': 863}, 'Harry Potter and the Deathly Hallows: Part 2 (2011)': {'correlation': 0.0690789911960064, 'non_missing_pairs': 822}, 'Harry Potter and the Goblet of Fire (2005)': {'correlation': 0.048730064935550675, 'non_missing_pairs': 810}, 'Harry Potter and the Chamber of Secrets (2002)': {'correlation': 0.06305574095499364, 'non_missing_pairs': 841}, 'The Matrix Revolutions (2003)': {'correlation': 0.003557158889092187, 'non_missing_pairs': 361}, 'The Matrix Reloaded (2003)': {'correlation': 0.04504018170863612, 'non_missing_pairs': 332}, 'The Matrix (1999)': {'correlation': 0.11000510993188362, 'non_missing_pairs': 494}, 'Indiana Jones and the Last Crusade (1989)': {'correlation': -0.008069492119042154, 'non_missing_pairs': 451}, 'Indiana Jones and the Temple of Doom (1984)': {'correlation': -0.003970999662757587, 'non_missing_pairs': 465}, 'Indiana Jones and the Raiders of the Lost Ark (1981)': {'correlation': -0.03540697875310553, 'non_missing_pairs': 332}, 'Indiana Jones and the Kingdom of the Crystal Skull (2008)': {'correlation': 0.07525955310250403, 'non_missing_pairs': 450}, 'The Lost World: Jurassic Park (1997)': {'correlation': 0.08646878208147771, 'non_missing_pairs': 555}, 'Jurassic Park III (2001)': {'correlation': 0.04432550622252206, 'non_missing_pairs': 524}, 'Jurassic Park (1993)': {'correlation': 0.12528749975908868, 'non_missing_pairs': 596}, "Pirates of the Caribbean: Dead Man's Chest (2006)": {'correlation': 0.0954196026983239, 'non_missing_pairs': 555}}
```

```
'non_missing_pairs': 797},
"Pirates of the Caribbean: At World's End (2007)": {'correlation': 0.1
373557469739529,
    'non_missing_pairs': 652},
'Pirates of the Caribbean: The Curse of the Black Pearl (2003)': {'cor
relation': 0.07475484653493199,
    'non_missing_pairs': 667},
'Toy Story 2 (1999)': {'correlation': 0.04837976360676105,
    'non_missing_pairs': 903},
'Toy Story 3 (2010)': {'correlation': 0.05686782476297887,
    'non_missing_pairs': 860},
'Toy Story (1995)': {'correlation': 0.04389427146559562,
    'non_missing_pairs': 917},
'Batman & Robin (1997)': {'correlation': 0.20421507132422873,
    'non_missing_pairs': 333},
'Batman (1989)': {'correlation': 0.12718662531006908,
    'non_missing_pairs': 377},
'Batman: The Dark Knight (2008)': {'correlation': -0.00433598776395710
5,
    'non_missing_pairs': 720}}
```

```
In [10]: # Isolating the first 400 columns (movie ratings) and the column of interest
movie_columns = df.columns[:400]

# Dictionary to store the correlation results for each movie
all_correlations = {}

# Calculating the correlation for each movie
for movie in movie_columns:
    # Dropping rows where either of the columns has a missing value
    df_filtered = df[[movie, experience_column]].dropna()

    # Calculating the Pearson correlation coefficient
    correlation = df_filtered[movie].corr(df_filtered[experience_column])

    # Storing the results
    all_correlations[movie] = {
        'correlation': correlation,
        'non_missing_pairs': len(df_filtered)
    }

# Displaying some of the correlation results to avoid too much output
list(all_correlations.items())[:10] # Displaying the first 10 results
```

```
Out[10]: [('The Life of David Gale (2003)', {'correlation': 0.06320829544992806, 'non_missing_pairs': 72}),
('Wing Commander (1999)', {'correlation': 0.18994180619532078, 'non_missing_pairs': 70}),
('Django Unchained (2012)', {'correlation': 0.05551910726842503, 'non_missing_pairs': 441}),
('Alien (1979)', {'correlation': 0.04900105342850054, 'non_missing_pairs': 283}),
('Indiana Jones and the Last Crusade (1989)', {'correlation': -0.008069492119042154, 'non_missing_pairs': 451}),
('Snatch (2000)', {'correlation': -0.0605387806675502, 'non_missing_pairs': 124}),
('Rambo: First Blood Part II (1985)', {'correlation': 0.06590240946098086, 'non_missing_pairs': 177}),
('Fargo (1996)', {'correlation': -0.01337097380488035, 'non_missing_pairs': 247}),
('Let the Right One In (2008)', {'correlation': 0.07702801422067912, 'non_missing_pairs': 131}),
('Black Swan (2010)', {'correlation': 0.024554156182422692, 'non_missing_pairs': 574})]
```

```
In [12]: # Creating a color map to distinguish different franchises
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'orange']

# Creating a scatter plot
plt.figure(figsize=(15, 7))

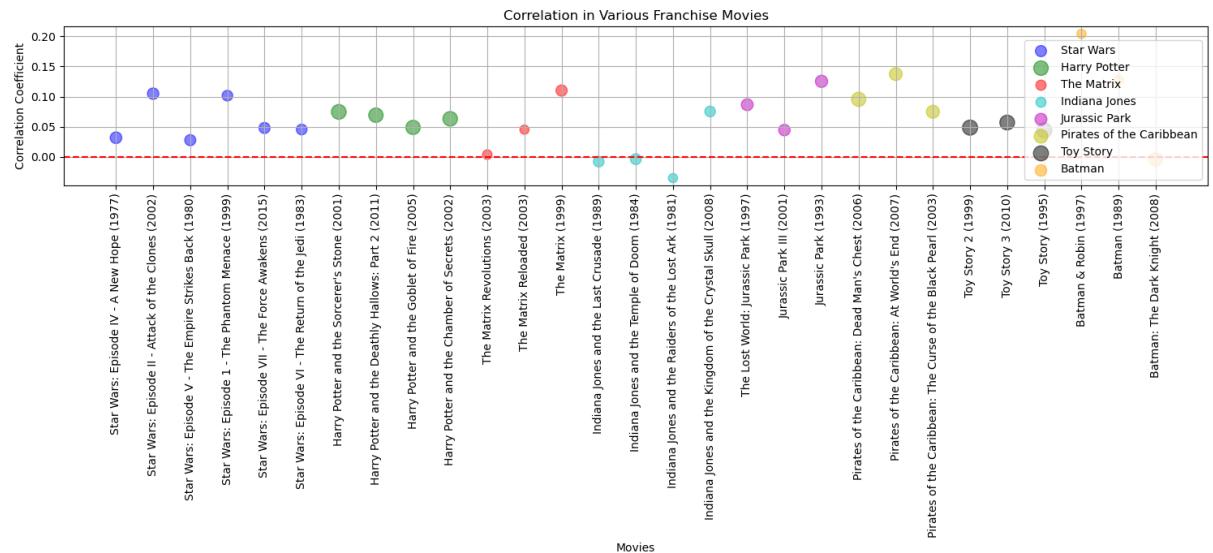
# Adding each franchise's movies to the plot
for franchise, color in zip(franchises, colors):
    # Finding the movies belonging to the franchise
    franchise_movies = [movie for movie in all_correlations.keys() if frachaince == movie]

    # Extracting correlations and non_missing_pairs
    correlations = [all_correlations[movie]['correlation'] for movie in franchise_movies]
    non_missing_pairs = [all_correlations[movie]['non_missing_pairs'] for movie in franchise_movies]

    # Creating the scatter plot
    plt.scatter(franchise_movies, correlations, s=np.array(non_missing_pairs) * 10)

# Adding labels and titles
plt.xlabel('Movies')
plt.ylabel('Correlation Coefficient')
plt.title('Correlation in Various Franchise Movies')
plt.xticks(rotation=90)
plt.axhline(y=0, color='r', linestyle='--') # Adding a horizontal line at zero
plt.legend()
plt.grid(True)

# Displaying the plot
plt.tight_layout()
plt.show()
```

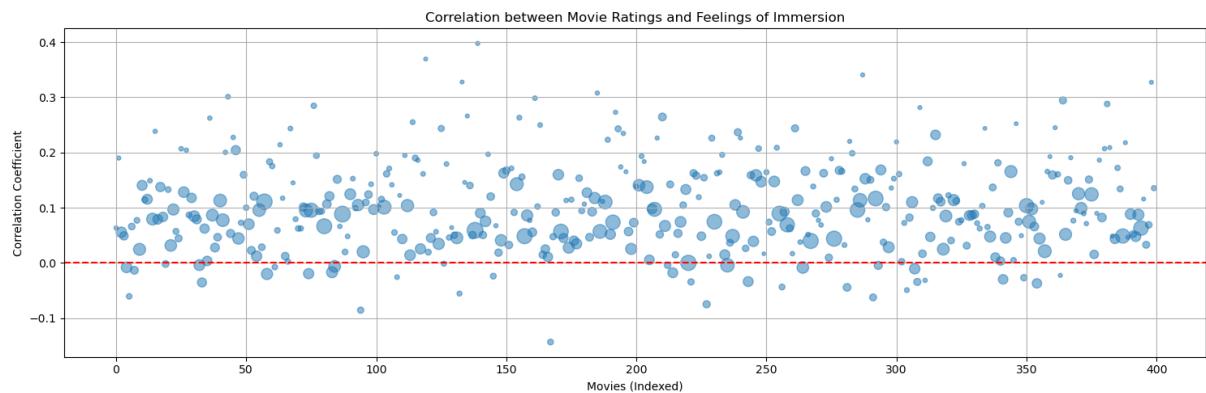


```
In [13]: # Extracting movie titles, correlations, and non-missing pair counts
movies = list(range(len(all_correlations))) # Using index numbers instead
correlations = [all_correlations[movie]['correlation'] for movie in all_movies]
non_missing_pairs = [all_correlations[movie]['non_missing_pairs'] for movie in all_movies]

# Creating a scatter plot
plt.figure(figsize=(15, 5))
plt.scatter(movies, correlations, s=np.array(non_missing_pairs)/5, alpha=0.5)

# Adding labels and titles
plt.xlabel('Movies (Indexed)')
plt.ylabel('Correlation Coefficient')
plt.title('Correlation between Movie Ratings and Feelings of Immersion')
plt.axhline(y=0, color='r', linestyle='--') # Adding a horizontal line at y=0
plt.grid(True)

# Displaying the plot
plt.tight_layout()
plt.show()
```



```
In [14]: # Calculating the average correlation for all 400 movies
all_movies_correlations = [all_correlations[movie]['correlation'] for movie in all_movies]
average_correlation_all_movies = np.mean(all_movies_correlations)

# Calculating the average correlation for specified franchises
franchise_movies_correlations = [all_correlations[movie]['correlation'] for movie in franchise_movies if any(franchise.lower() in movie.lower() for franchise in franchises)]
average_correlation_franchises = np.mean(franchise_movies_correlations)

average_correlation_all_movies, average_correlation_franchises
```

Out[14]: (0.10070145866454729, 0.06345771637100574)

```
In [27]: from scipy.stats import ttest_ind  
  
# Conducting the independent two-sample t-test  
t_stat, t_p_value = ttest_ind(all_movies_correlations, franchise_movies_  
    t_stat, t_p_value
```

```
Out[27]: (3.613577427252558, 0.0008432947415034776)
```

```
In [ ]:
```