Victoria (Xinyue) Shi
Pascal Wallisch
Introduction to Data Science (DS-UA 112)
21 Aug 2022

Capstone Project

Dimension reduction:

For certain analyses involving information such as sensation seeking and personality traits, relevant data is first separated into new datasets (storing in Pandas DataFrames), then z-scored, followed by dimensionality reductions using PCA. Screeplots are plotted and analyzed, and I choose the Kaiser criterion to keep factors with eigenvalues larger than 1.

Data cleaning:

The .csv file is read by using Pandas, and all the information is stored in a DataFrame named “data”. Subsequently, useful information is drawn from *data* and converted into NumPy arrays for the ease of use. Because the dataset contains a large amount of missing data (NaNs), row-wise removals are done during the process. However, in order to keep the complete dataset untouched and to keep as much useful data as possible, subsets, or smaller arrays, are created from *data*, and then row-wise removals of NaNs are conducted accordingly. Also, imputation by means is performed in certain cases, and will be explicitly mentioned in the analysis.

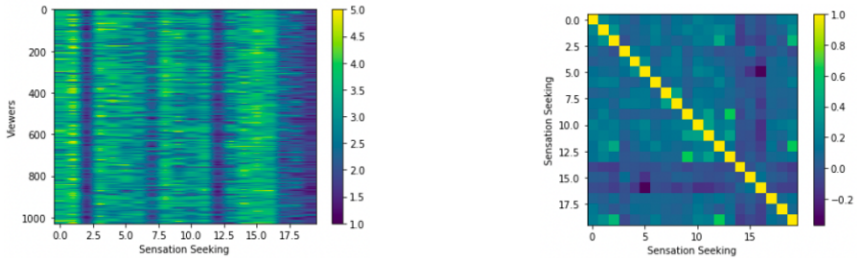
Data transformation and processing:

In order to perform PCA, certain data transformations are done by z-scoring the data. Afterwards, the transformed data is rotated, and the old data is graphed in a new coordinate system. Several libraries and packages are imported and used throughout the project:

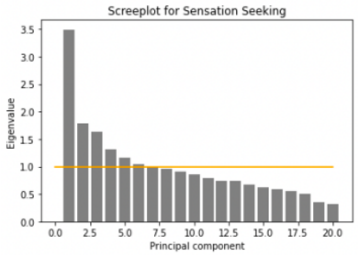
```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
```

Question 1:

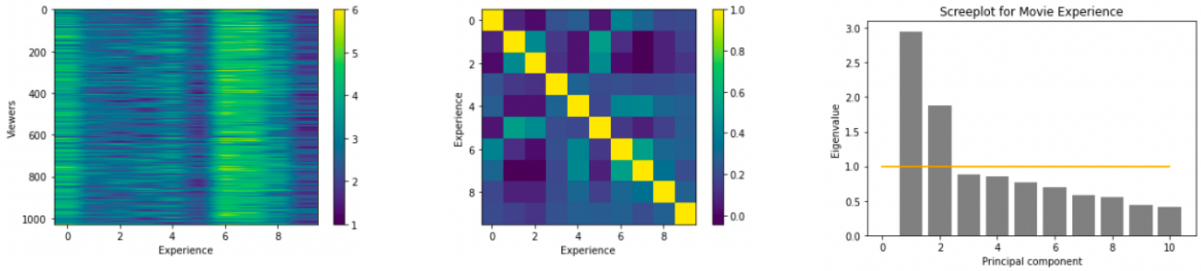
Since sensation seeking and movie experience both contain many columns and it'll be hard to interpret data with such complicated dimensions, PCA is used for both variables. Row-wise removal of missing data is performed on the DataFrame containing data of sensation seeking and movie experience.



Above are the heatmap of the sensation seeking data and correlation data between sensation seeking questions. Then the data is z-scored by using *stats.zscore()* and a PCA is done by using *PCA().fit()*. The variance explained for each factor is also calculated by using *pca.explained_variance_*. Below is the screeplot for sensation seeking, and according to the Kaiser criterion, 6 factors which have eigenvalues larger than 1 are kept for analysis.

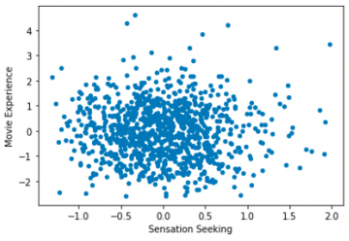


Similarly, the PCA of movie experience is done, and 2 factors are retained.



Hence, the 6 principal factors of sensation seeking and 2 principal factors of movie experience are extracted and used for further analysis. In order to decide the relationship, the means of the variables are calculated, and then a correlation is calculated by using np.corrcoef(). Below are the correlation matrix as well as the scatterplot of the two variables.

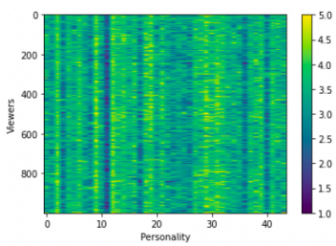
```
...: corrSE = np.corrcoef(sensationMean,experienceMean)
...: print(corrSE)
[[ 1.         -0.02757175]
 [-0.02757175  1.        ]]
```

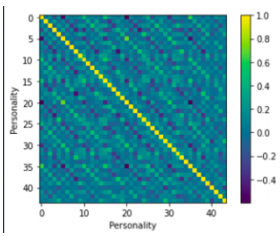


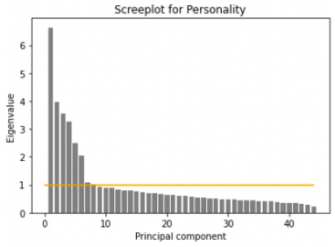
As we can see from the correlation coefficient which is approximately zero as well as the scatterplot which shows no meaningful pattern, the data seems to be all over the place and doesn't seem to be strongly correlated. Consequently, my conclusion is that there isn't a notable relationship between sensation seeking and movie experience.

Question 2:

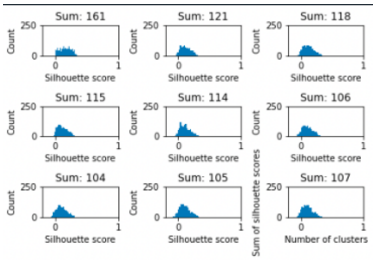
Again, since the data of personality includes many columns so is of high complexity, a PCA is performed. By the Kaiser criterion, 8 factors are kept for further analysis.





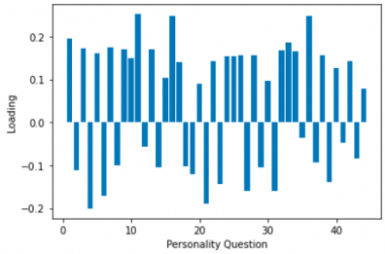


In order to classify personalities, I first use the silhouette method to find the optimal number of clusters. According to the silhouette scores for each possible number of clusters, the score is highest when the cluster number is 2. Hence, quantitatively speaking, there should be 2 types of personalities.

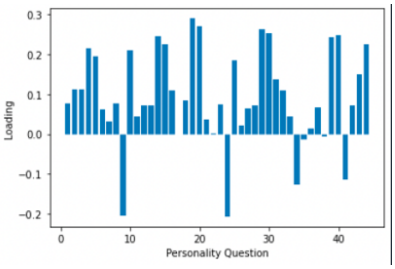


Therefore, an investigation of the two principal factors of the personality variable is done. By looking at the loading of the first principal component, question 11, 16 and 36 seem to be

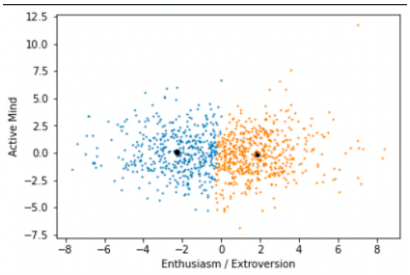
contributing to the factor most notably, and these questions are “Is full of energy”, “Generates a lot of Enthusiasm”, and “is outgoing/sociable”. Hence, I would describe the first factor as “enthusiasm / extroversion”.



Similarly, the loading of the second factor is investigated, and questions 19 and 20 stand out, which correspond to “Worries a lot” and “Has an active imagination”, so I would describe this second personality factor as “active mind”.



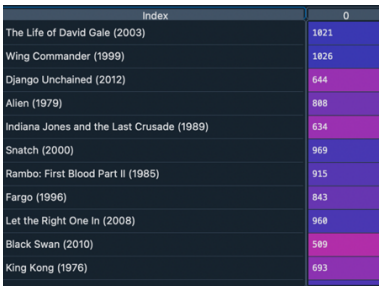
Then, the kMeans method is used, and “enthusiasm / extroversion” is plotted on the x-axis while “active mind” is plotted on the y-axis. According to the graph, one group of people is less enthusiastic, or, in other words, more introverted, whereas the other group has high enthusiasm, so can be said to be more out-going. This classification of personalities makes sense.



Question 3:

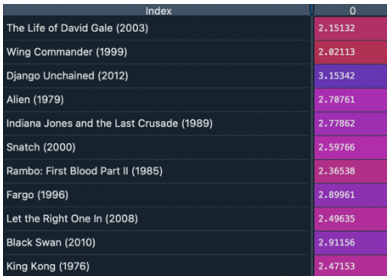
The popularity of each movie can be indicated by the number of participants who rate it, so the movie which contains the most missing values (NaNs) in ratings is the one being the least popular. Hence, the number of NaNs in the ratings of each movie is found by using

the *.isnull().sum(axis = 0)* function of Pandas. The figure below is an overview of the resulting Series variable, showing the corresponding number of missing values of each movie:



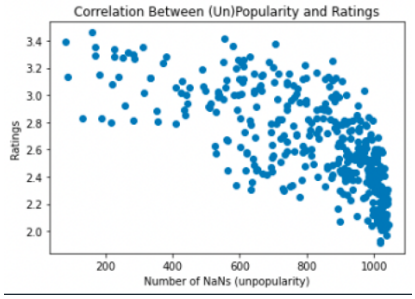
Index	0
The Life of David Gale (2003)	1021
Wing Commander (1999)	1026
Django Unchained (2012)	644
Alien (1979)	808
Indiana Jones and the Last Crusade (1989)	634
Snatch (2000)	969
Rambo: First Blood Part II (1985)	915
Fargo (1996)	843
Let the Right One In (2008)	960
Black Swan (2010)	589
King Kong (1976)	693

Then, the mean ratings of all 400 movies are calculated by using *.mean (axis = 0)*, being stored in another Series variable, as shown in the figure:



Index	0
The Life of David Gale (2003)	2.15132
Wing Commander (1999)	2.02113
Django Unchained (2012)	3.15342
Alien (1979)	2.70761
Indiana Jones and the Last Crusade (1989)	2.77862
Snatch (2000)	2.59766
Rambo: First Blood Part II (1985)	2.36538
Fargo (1996)	2.89961
Let the Right One In (2008)	2.49635
Black Swan (2010)	2.91156
King Kong (1976)	2.47153

A scatterplot of popularity (measured in terms of number of NaNs) and mean ratings is generated by using Matplotlib:



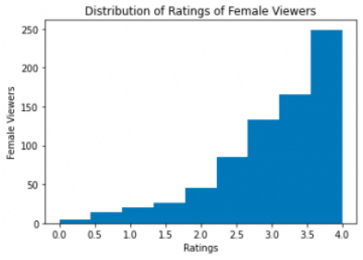
Since the relationship between the two variables is obviously not linear and seems reasonably monotonic, a Spearman's rank correlation rho is calculated by using the *stats.spearmanr()* function of SciPy, and the result is -0.7608226731098053. Therefore, it's suggested that there's a relatively strong negative nonlinear relationship between unpopularity and ratings of movies. In other words, there's a relatively strong positive relationship between popularity and ratings: movies that are more popular indeed tend to be rated higher.

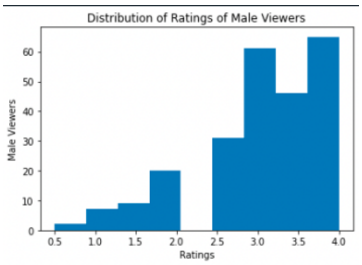
A hypothesis testing is also conducted, with the null hypothesis that movies which are more popular are rated equally as movies which are less popular. A paired t-test is used and the p-value

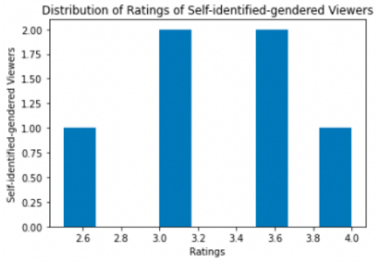
is 8.936835083345294e-129, much smaller than the alpha level of 0.05, so we can as well reject the null hypothesis and conclude that more popular movies are rated higher.

Question 4:

A hypothesis testing is conducted, and the null hypothesis is that male and female viewers rate 'Shrek (2001)' equally. In preparation for the testing, the columns in *data* corresponding to the information of gender and the ratings of 'Shrek (2001)' are drawn out and concatenated to form a DataFrame object. In order to process the data, rows containing NaNs are dropped using the *.dropna()* method of Pandas. For the ease of use, the DataFrame object is converted into a Numpy array. Since comparisons between genders are the aim, I separate the array into three smaller ones, corresponding to ratings of female, male, and self-identified viewers. The figures below show the distribution of the ratings of all three groups.







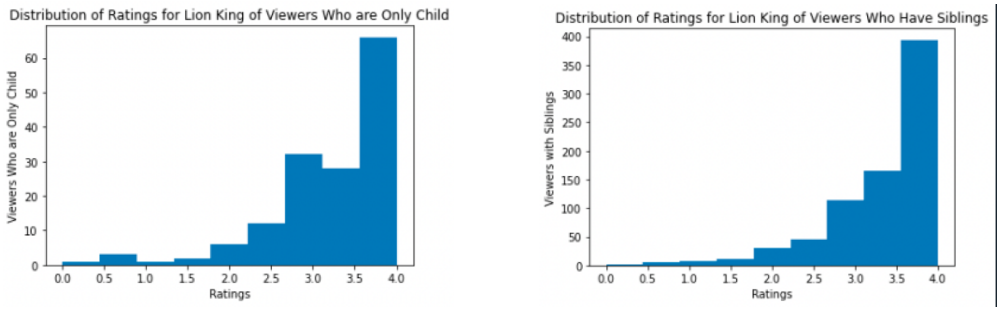
From the plots, it's obvious that the data isn't distributed normally, and since the "psychological difference" between different levels of ratings is not consistent, a non-parametric test seems more appropriate. Hence, three Mann-Whitney U tests are conducted by using the *stats.mannwhitneyu()* function of SciPy: one between the female group and male group, one between the female group and self-identified-gendered group, and one between the male group and self-identified-gendered group. The figure below show the test statistic U and p-values for all three tests.

```
Test statistic U for Mann-Whitney U test of female and male ratings: 96830.5
p-value for Mann-Whitney U test of female and male ratings: 0.050536625925559006

Test statistic U for Mann-Whitney U test of female and self-described gender ratings: 2342.0
p-value for Mann-Whitney U test of female and self-described gender ratings: 0.826358951526508

Test statistic U for Mann-Whitney U test of male and self-described gender ratings: 679.0
p-value for Mann-Whitney U test of male and self-described gender ratings: 0.7967165733788198
```

The p-value of the Mann-Whitney U test between female and male viewers is slightly larger, so we're on the verge of rejecting the null hypothesis, but still fail to reject the null hypothesis, indicating that there might not be an obvious gendered preference in terms of watching 'Shrek (2001)'. As for the other two Mann-Whitney U tests (between female and self-described gender, and between male and self-described gender), the p-value is much larger than the alpha level of 0.05, suggesting a failure towards rejecting the null.

Question 5:

Again, a hypothesis test is conducted, and the null hypothesis is that there's no difference between the ratings for 'The Lion King (1994)' between people who are only child and people who have siblings. A smaller DataFrame containing ratings for 'The Lion King (1994)' and sibship information of viewers are created and then converted into a NumPy array for analysis. Row-wise removal of NaNs are involved, and rows containing -1 for sibship status are also dropped. Then two separated sub-arrays are drawn, corresponding to the group of people who're only child and the viewers who have siblings respectively.



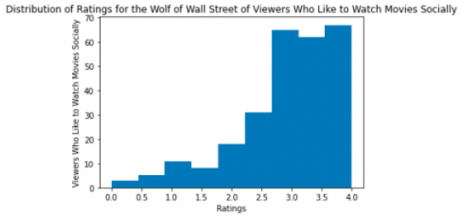
A non-parametric Mann-Whitney U test is done because the data isn't normally distributed, and the figure below shows the test statistics and p-value:

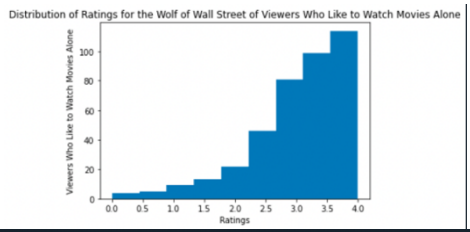
```
- Test statistic U for Mann-Whitney U test of ratings of viewers who are only child and viewers
who have siblings: 52929.0
- p-value for Mann-Whitney U test of ratings of viewers who are only child and viewers who have
siblings: 0.04319872995682849
```

The p-value is approximately 0.04, which is smaller than the alpha level of 0.05, which means that the observed pattern is unlikely to happen by chance. In other words, people who're only children indeed tend to rate 'The Lion King (1994)' differently than people who have siblings.

Question 6:

Similarly, a non-parametric Mann-Whitney U test is conducted, and the null hypothesis is that there's no difference between the ratings for 'The Wolf of Wall Street (2013)' of people who like to watch movies socially and people who prefer watching movies alone.






```
- Test statistic U for Mann-Whitney U test of ratings of viewers who prefer watching movies
socially and viewers who prefer watching movies alone: 49303.5
- p-value for Mann-Whitney U test of social viewers and viewers alone ratings:
0.1127642933222891
```

Since the p-value is approximately 0.11, which is much greater than the alpha level of 0.05, we consequently fail to reject the null. It's likely that the observed pattern in our dataset is completely due to chance. In other words, people who like to watch movies socially seem to enjoy 'The Wolf of Wall Street (2013)' equally as people who prefer watching movies alone.

Questions 7:

Starting with the 'Star Wars' franchise which consists of 6 movies as in the dataset, I first put all ratings of this series into a NumPy array. Because the data is not independent in this case, row-wise removal of missing values is performed. Then I calculate some descriptive statistics to get a general picture of the data. The columns in the figure below represent the means, standard deviations, numbers of viewers, and standard error of the mean respectively, and the rows represent 'Star Wars: Episode 1 - The Phantom Menace (1999)', 'Star Wars: Episode II - Attack of the Clones (2002)', 'Star Wars: Episode IV - A New Hope (1977)', 'Star Wars: Episode V - The Empire Strikes Back (1980)', 'Star Wars: Episode VI - The Return of the Jedi (1983)', and 'Star Wars: Episode VII - The Force Awakens (2015)'.

```
[[2.51051051e+00 1.12267539e+00 3.33000000e+02 6.15222326e-02]
 [2.57657658e+00 1.02491089e+00 3.33000000e+02 5.61647708e-02]
 [3.22522523e+00 9.19603172e-01 3.33000000e+02 5.03939434e-02]
 [3.23123123e+00 9.21376427e-01 3.33000000e+02 5.04911172e-02]
 [3.14264264e+00 9.51487652e-01 3.33000000e+02 5.21412021e-02]
 [3.16216216e+00 9.30373756e-01 3.33000000e+02 5.09841677e-02]]
```

Figure: Descriptive Statistics of the Star Wars Franchise

It seems that there are indeed some fluctuations between the ratings of these 6 movies of the franchise. An ANOVA test is conducted since comparisons between more than two sample means are involved, and the null hypothesis is that there're no differences between the ratings of the six movies of the Star Wars franchise, or, in other words, the fluctuations are due to chance alone. The resulting test statistic *f* and p-value are shown in the figure below:

```
- f-statistic for ANOVA of Star Wars:  39.029939613074056
- p-value for ANOVA of Star Wars:  2.3995951635329 92e-38
```

Figure: f-value and p-value of ANOVA for the Star Wars Franchise

The p-value is extremely small, much smaller than the alpha level of 0.05, suggesting that the observed pattern is unlikely due to chance alone, which means the quality of the Star Wars franchise is indeed inconsistent.

The same process is repeated for all the other franchises, including 'Harry Potter', 'The Matrix', 'Indiana Jones', 'Jurassic Park', 'Pirates of the Caribbean', 'Toy Story', and 'Batman'.

```
[[3.41901408e+00 7.90429826e-01 7.10000000e+02 2.96643028e-02]
 [3.33098592e+00 8.50615610e-01 7.10000000e+02 3.19230350e-02]
 [3.35915493e+00 7.96484788e-01 7.10000000e+02 2.98915415e-02]
 [3.37323944e+00 8.16288016e-01 7.10000000e+02 3.06347434e-02]]
```

Figure: Descriptive Statistics of the Harry Potter Franchise

```
- f-statistic for ANOVA of Harry Potter:  1.4456904473285563
- p-value for ANOVA of Harry Potter:  0.2275340290918136
```

Figure: f-value and p-value of ANOVA for the Harry Potter Franchise

```
[[3.25384615e+00 8.40664009e-01 2.60000000e+02 5.21357686e-02]
 [2.84615385e+00 9.35098008e-01 2.60000000e+02 5.79923166e-02]
 [2.80576923e+00 9.93132856e-01 2.60000000e+02 6.15914851e-02]]
```

Figure: Descriptive Statistics of the Matrix Franchise

```
- f-statistic for ANOVA of Matrix:  18.592815113038 09
- p-value for ANOVA of Matrix:  1.2957225925356723e-08
```

Figure: f-value and p-value of ANOVA for the Matrix Franchise

```
[[3.05532787e+00 8.89379674e-01 2.44000000e+02 5.69366992e-02]
 [2.82991803e+00 8.81716839e-01 2.44000000e+02 5.64461365e-02]
 [2.92418033e+00 8.76251648e-01 2.44000000e+02 5.60962635e-02]
 [2.45286885e+00 1.04058620e+00 2.44000000e+02 6.66167051e-02]]
```

Figure: Descriptive Statistics of the Indiana Jones Franchise

```
- f-statistic for ANOVA of Indiana Jones:  19.050958699528884
- p-value for ANOVA of Indiana Jones:  5.204254257621 15e-12
```

Figure: f-value and p-value of ANOVA for the Indiana Jones Franchise

```
[[3.13316583e+00 8.78186532e-01 3.98000000e+02 4.40195133e-02]
 [2.95728643e+00 8.67147274e-01 3.98000000e+02 4.34661653e-02]
 [2.71231156e+00 9.36836430e-01 3.98000000e+02 4.69593670e-02]]
```

Figure: Descriptive Statistics of the Jurassic Park Franchise

```
- f-statistic for ANOVA of Jurassic Park:  22.163615231952214
- p-value for ANOVA of Jurassic Park:  3.542127514286409e-10
```

Figure: f-value and p-value of ANOVA for the Jurassic Park Franchise

```
[[3.03565062e+00 9.01856856e-01 5.61000000e+02 3.80764263e-02]
 [2.98484848e+00 8.69487947e-01 5.61000000e+02 3.67098099e-02]
 [2.89483066e+00 9.53738314e-01 5.61000000e+02 4.02668632e-02]]
```

Figure: Descriptive Statistics of the Pirates of the Caribbean Franchise

```
- f-statistic for ANOVA of Pirates of the Caribbean:  3.4465950041304327
- p-value for ANOVA of Pirates of the Caribbean:  0.03207932803269902
```

Figure: f-value and p-value of ANOVA for the Pirates of the Caribbean Franchise

```
[[3.35865258e+00 8.24713893e-01 7.57000000e+02 2.99747363e-02]
 [3.19682959e+00 8.39632862e-01 7.57000000e+02 3.05169754e-02]
 [3.30713342e+00 8.10956576e-01 7.57000000e+02 2.94747181e-02]]
```

Figure: Descriptive Statistics of the Toy Story Franchise

```
- f-statistic for ANOVA of Toy Story:  7.5881445425788 29
- p-value for ANOVA of Toy Story:  0.0005193828629536134
```

Figure: f-value and p-value of ANOVA for the Toy Story Franchise

```
[[2.83789954e+00 9.67497912e-01 2.19000000e+02 6.53774502e-02]
 [2.32420091e+00 1.11748604e+00 2.19000000e+02 7.55127086e-02]
 [3.22831050e+00 9.47441821e-01 2.19000000e+02 6.40221852e-02]]
```

Figure: Descriptive Statistics of the Batman Franchise

```
- f-statistic for ANOVA of Batman:  43.625878915167 57
- p-value for ANOVA of Batman:  1.6410731510652519e-18
```

Figure: f-value and p-value of ANOVA for the Batman Franchise

As can be concluded from all the p-values of the ANOVA tests, only the p-value of the Harry Potter franchise is larger than 0.05, suggesting a failure to reject the null hypothesis, which in turn means the four movies of the Harry Potter franchise are likely to have no notable differences in ratings. All the other p-values are much lower than the alpha level of 0.05, which indicates that viewers have unfortunately experienced inconsistent quality of those franchises.

Question 8:

For the prediction model, I choose the linear regression. Here when cleaning data, an imputation by means is conducted since row-wise removal of NaNs will result in a significant loss of data. In order to build the model with personality factors as the independent variable and ratings as the dependent variable, I first use cross-validation to separate the data into test and training sets (50% for test and 50% for training). Then, the linear regression model is generated by fitting the train sets, and the R-squared as well as the RMSE are calculated by using the test sets. The figure below shows the predictions for movies, R-squared and RMSE for the prediction model. Since R-squared is negative, it basically means the prediction model cannot explain any proportion of the variance, so I think it's reasonable to conclude that the linear regression model fits worse than just plotting a horizontal line.

```
[[2.10993237 1.97965655 3.19359403 ... 2.4361825  2.15073386 2.3548983 ]
 [2.06233086 2.09929618 2.96089676 ... 2.352495   2.12840686 2.2350712 ]
 [2.2150603  2.10261215 3.0898562  ... 2.4211975  2.12958059 2.32070941]
 ...
 [2.226744   1.944087   3.29151623 ... 2.46201136 2.00789742 2.63500803]
 [2.10872182 1.93529861 3.24896956 ... 2.34718149 2.04463017 2.11234482]
 [2.07918468 1.99067365 2.95902149 ... 2.52842634 2.22734468 2.42146602]]
R-squared of the prediction model is: -0.0744879235951027
RMSE of the prediction model is: 0.5403772551053222
```

Question 9:

Still, a linear regression is used for the prediction. Missing data is handled by using imputation by means. Cross-validation is utilized to separate 50% of data in to training sets and the other 50% into test sets. Again, the R-squared is smaller than 0, so the movie ratings cannot be reasonably predicted by the linear regression model.

```
[[2.14064598 2.02917453 3.16597051 ... 2.37824251 2.10299473 2.33799113]
 [2.14064598 2.02917453 3.16597051 ... 2.37824251 2.10299473 2.33799113]
 [2.14064598 2.02917453 3.16597051 ... 2.37824251 2.10299473 2.33799113]
 ...
 [2.14064598 2.02917453 3.16597051 ... 2.37824251 2.10299473 2.33799113]
 [2.1280152  2.03157935 2.98817924 ... 2.33388722 2.12757226 2.3416177 ]
 [2.14064598 2.02917453 3.16597051 ... 2.37824251 2.10299473 2.33799113]]
R-squared of the prediction model is: -0.005239048901056348
RMSE of the prediction model is: 0.5088213456461093
```

Question 10:

The data is cleaned by imputing by the means, and then cross-validation is performed to separate data into training and test sets. Afterwards, the data is fit into a linear regression model

which turns out to have extremely small R-squared. Hence, similar to previous situations, the model doesn't predict ratings well.

```
[[2.14995112 1.97126216 2.85492578 ... 2.32926226 2.08659474 2.26547399]
 [2.17595282 2.15994897 3.33516378 ... 2.59463918 2.25348184 2.37691194]
 [2.38242966 2.09664993 3.34980367 ... 2.52234468 2.36625987 2.39868077]
 ...
 [2.15553575 2.0618429  3.21061604 ... 2.33423679 2.17950728 2.53615989]
 [2.18760365 2.20854967 3.27673684 ... 2.39177007 2.26464938 2.33832284]
 [2.25633574 2.20764378 3.15560984 ... 2.56189709 2.28487576 2.51276694]]
R-squared of the prediction model is: -0.13900090589548647
RMSE of the prediction model is: 0.5294626018602467
```

Extra credit:

1. From questions 8-10 above, one insight which can be drawn is that it seems unreasonable to predict specific movie ratings from factors such as personalities or gender identity by using a linear regression model.
2. Some movies are considered to have breakthrough effects on the discussion of gender equality, and "Bend It Like Beckham (2002)" is one of such, exploring the world of female soccer. I perform a hypothesis testing to see if female viewers and male viewers indeed have different opinions regarding this movie, and the null hypothesis is that both gender rate the movie equally. Data cleaning is done by row-wise removal of missing data, and then a Mann-Whitney test is performed. The p-value turns out to be much smaller than 0.05, which suggests a success of rejecting the null hypothesis. Hence, I now conclude that female and male viewers rate "Bend It Like Beckham (2002)" differently, and this may be considered as evidence that this movie indeed generates some heated discussion upon the topic of gender equality.

```
Test statistic U for Mann-Whitney U test of female and male ratings: 15340.0
p-value for Mann-Whitney U test of female and male ratings: 3.207869539880227e-06
```

Appendix (codes):

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer


data = pd.read_csv('/Users/victoriashi/Documents/Courses/Intro to
DS/Data Analysis Reports/movieReplicationSet.csv')

#%% Q1: What is the relationship between sensation seeking and
movie experience?

#stack sensation seeking data and movie experience data
sensation = data.iloc[:, 400:420]
experience = data.iloc[:, 464:474]
sensationAndExperience = pd.concat([sensation,experience], axis = 1)
sensationAndExperience.dropna(inplace = True)

nonnanSensation = sensationAndExperience.iloc[:,0:20]
nonnanExperience = sensationAndExperience.iloc[:,20:]


#PCA: dimensionality reduction for sensation seeking
arrSensation = nonnanSensation.to_numpy()
print(arrSensation.shape)
plt.imshow(arrSensation, aspect='auto')
plt.xlabel('Sensation Seeking')
plt.ylabel('Viewers')
plt.colorbar()
plt.show()

corrMatrixSensation = np.corrcoef(arrSensation,rowvar=False)
plt.imshow(corrMatrixSensation)
plt.xlabel('Sensation Seeking')
plt.ylabel('Sensation Seeking')
plt.colorbar()
plt.show()

zscoredSensation = stats.zscore(arrSensation)
pca = PCA().fit(zscoredSensation)
eigValsSensation = pca.explained_variance_
loadingsSensation = pca.components_
rotatedSensation = pca.fit_transform(zscoredSensation)

varExplainedSensation = eigValsSensation/sum(eigValsSensation)*100
for ii in range(len(varExplainedSensation)):
    print(varExplainedSensation[ii].round(3))

numSensation = len(sensation.columns)
x = np.linspace(1,numSensation,numSensation)
plt.bar(x, eigValsSensation, color='gray')
plt.plot([0,numSensation],[1,1],color='orange') # Orange Kaiser
criterion line for the fox
plt.xlabel('Principal component')
plt.ylabel('Eigenvalue')
plt.title('Screeplot for Sensation Seeking')
plt.show()

kaiserThreshold = 1
print('Number of factors selected by Kaiser criterion:',
np.count_nonzero(eigValsSensation > kaiserThreshold))

dfSensationQuestions = pd.DataFrame(list(nonnanSensation.columns))
whichPrincipalComponent = 1 # Select and look at one factor at a time
plt.bar(x,loadingsSensation[whichPrincipalComponent,:]*-1) # note:
eigVecs multiplied by -1 because the direction is arbitrary
plt.xlabel('Sensation Seeking Question')
plt.ylabel('Loading')
plt.show() # Show bar plot
print(dfSensationQuestions.values)
origDataNewCoordinates = pca.fit_transform(zscoredSensation)*-1
sensationX =
np.column_stack((origDataNewCoordinates[:,0],origDataNewCoordina
tes[:,1]))


#PCA: dimensionality reduction for movie experience

arrExperience = nonnanExperience.to_numpy()
print(arrExperience.shape)
plt.imshow(arrExperience, aspect='auto')
plt.xlabel('Experience')
plt.ylabel('Viewers')
plt.colorbar()
plt.show()

corrMatrixExperience = np.corrcoef(arrExperience,rowvar=False)
plt.imshow(corrMatrixExperience)
plt.xlabel('Experience')
plt.ylabel('Experience')
plt.colorbar()
plt.show()

zscoredExperience = stats.zscore(arrExperience)
pca = PCA().fit(zscoredExperience)
eigValsExperience = pca.explained_variance_
loadingsExperience = pca.components_
rotatedExperience = pca.fit_transform(zscoredExperience)

varExplainedExperience =
eigValsExperience/sum(eigValsExperience)*100
for ii in range(len(varExplainedExperience)):
    print(varExplainedExperience[ii].round(3))

numExperience = len(experience.columns)
x = np.linspace(1,numExperience,numExperience)
plt.bar(x, eigValsExperience, color='gray')
plt.plot([0,numExperience],[1,1],color='orange') # Orange Kaiser
criterion line for the fox
plt.xlabel('Principal component')
plt.ylabel('Eigenvalue')
plt.title('Screeplot for Movie Experience')
plt.show()

kaiserThreshold = 1
print('Number of factors selected by Kaiser criterion:',
np.count_nonzero(eigValsExperience > kaiserThreshold))

dfExperienceQuestions =
pd.DataFrame(list(nonnanExperience.columns))
whichPrincipalComponent = 3 # Select and look at one factor at a time
plt.bar(x,loadingsExperience[whichPrincipalComponent,:]*-1) # note:
eigVecs multiplied by -1 because the direction is arbitrary
plt.xlabel('Experience Question')
plt.ylabel('Loading')
plt.show() # Show bar plot
print(dfExperienceQuestions.values)
```

```
sensationPrincipal = rotatedSensation[:,:6]
experiencePrincipal = rotatedExperience[:,:2]

sensationMean = np.mean(sensationPrincipal,axis=1)
experienceMean = np.mean(experiencePrincipal,axis=1)

corrSE = np.corrcoef(sensationMean,experienceMean)
print(corrSE)

plt.plot(sensationMean,experienceMean, 'o', markersize=4)
plt.xlabel('Sensation Seeking')
plt.ylabel('Movie Experience')
plt.show()


#%% Q2: Is there evidence of personality types based on the data of
these research participants? If so, characterize these types both
quantitatively and narratively.
#PCA: personality
personality = data.iloc[:, 420:464]
personality.dropna(inplace = True)
arrPersonality = personality.to_numpy()
print(arrPersonality.shape)
plt.imshow(arrPersonality, aspect='auto')
plt.xlabel('Personality')
plt.ylabel('Viewers')
plt.colorbar()
plt.show()

corrMatrixPersonality = np.corrcoef(arrPersonality,rowvar=False)
plt.imshow(corrMatrixPersonality)
plt.xlabel('Personality')
plt.ylabel('Personality')
plt.colorbar()
plt.show()

zscoredPersonality = stats.zscore(arrPersonality)
pca = PCA().fit(zscoredPersonality)
eigValsPersonality = pca.explained_variance_
loadingsPersonality = pca.components_
rotatedPersonality = pca.fit_transform(zscoredPersonality)

varExplainedPersonality =
eigValsPersonality/sum(eigValsPersonality)*100
for ii in range(len(varExplainedPersonality)):
   print(varExplainedPersonality[ii].round(3))

numPersonality = len(personality.columns)
x = np.linspace(1,numPersonality,numPersonality)
plt.bar(x, eigValsPersonality, color='gray')
plt.plot([0,numPersonality],[1,1],color='orange') # Orange Kaiser
criterion line for the fox
plt.xlabel('Principal component')
plt.ylabel('Eigenvalue')
plt.title('Screeplot for Personality')
plt.show()

kaiserThreshold = 1
print('Number of factors selected by Kaiser criterion:',
np.count_nonzero(eigValsPersonality > kaiserThreshold))

dfPersonalityQuestions = pd.DataFrame(list(personality.columns))
whichPrincipalComponent = 1 # Select and look at one factor at a time
plt.bar(x,loadingsPersonality[whichPrincipalComponent,:]*-1) # note:
eigVecs multiplied by -1 because the direction is arbitrary
plt.xlabel('Personality Question')
plt.ylabel('Loading')
plt.show() # Show bar plot
print(dfPersonalityQuestions.values)
#factor1: enthusiasm / extroversion
#factor2: active mind

personalityX = rotatedPersonality[:,:8]

numClusters = 9
sSum = np.empty([numClusters,1])*np.NaN
for ii in range(2, numClusters+2): # Loop through each cluster (from 2
to 10)
   kMeans = KMeans(n_clusters = int(ii)).fit(personalityX) # compute
kmeans using scikit
   cId = kMeans.labels_ # vector of cluster IDs that the row belongs to
   cCoords = kMeans.cluster_centers_ # coordinate location for center
of each cluster
   s = silhouette_samples(personalityX,cId) # compute the mean
silhouette coefficient of all samples
   sSum[ii-2] = sum(s) # take the sum
   # Plot data:
   plt.subplot(3,3,ii-1)
   plt.hist(s,bins=20)
   plt.xlim(-0.2,1)
   plt.ylim(0,250)
   plt.xlabel('Silhouette score')
   plt.ylabel('Count')
   plt.title('Sum: {}'.format(int(sSum[ii-2]))) # sum rounded to nearest
integer
   plt.tight_layout() # adjusts subplot

plt.plot(np.linspace(2,10,9),sSum)
plt.xlabel('Number of clusters')
plt.ylabel('Sum of silhouette scores')
plt.show()

numClusters = 2
kMeans = KMeans(n_clusters = numClusters).fit(personalityX)
cId = kMeans.labels_
cCoords = kMeans.cluster_centers_
for ii in range (numClusters):
   plotIndex = np.argwhere(cId == int(ii))
   plt.plot(personalityX[plotIndex,0],
personalityX[plotIndex,1],'o',markersize=1)
   plt.plot(cCoords[int(ii-1),0],cCoords[int(ii-
1),1],'o',markersize=5,color='black')
   plt.xlabel('Enthusiasm / Extroversion')
   plt.ylabel('Active Mind')


#%% Q3: Are movies that are more popular rated higher than movies
that are less popular?

rating = data.iloc[:,0:400]

nans=rating.isnull().sum(axis=0)
means=rating.mean(axis=0)

plt.plot(nans,means,'o')
plt.xlabel('Number of NaNs (unpopularity)')
plt.ylabel('Ratings')
plt.title('Correlation Between (Un)Popularity and Ratings')

rho = stats.spearmanr(nans,means)
print(rho.correlation)
actualRho = rho.correlation

plt.hist(nans) #the data is left-skewed
plt.hist(nans**3)
plt.hist(means)
#paired (dependent) t-test
arrNansCube = (nans.to_numpy())**3
arrMeans = means.to_numpy()
t1,p1 = stats.ttest_rel(arrNansCube,arrMeans)
```

```
print("p-value for dependent t-test of popularity (number of nans) and ratings (mean):", p1)

#%% Q4: Is enjoyment of ‘Shrek (2001)’ gendered, i.e. do male and female viewers rate it differently?
gender = data.iloc[:, 474]
dfGender = gender.to_frame()
shrekList = ['Shrek (2001)']
shrekRating = rating[shrekList]
shrekRatingGender = pd.concat([shrekRating, dfGender],axis=1)
shrekRatingGender.dropna(inplace = True)

#sort the dataframe according to gender
shrekRatingGender.sort_values(by=['Gender identity (1 = female; 2 = male; 3 = self-described)'],inplace=True)
arrShrekRatingGender = shrekRatingGender.to_numpy()

#put different genders into different arrays
femaleShrek= arrShrekRatingGender[0:743,0]
maleShrek = arrShrekRatingGender[743:984,0]
otherShrek = arrShrekRatingGender[984:,0]

#null hypothesis: the ratings have no difference (all genders rate Shrek equally)
#first take a look at the distributions of ratings for different genders
plt.hist(femaleShrek[:],bins=9)
plt.title('Distribution of Ratings of Female Viewers')
plt.xlabel('Ratings')
plt.ylabel('Female Viewers')
plt.hist(maleShrek[:],bins=9)
plt.title('Distribution of Ratings of Male Viewers')
plt.xlabel('Ratings')
plt.ylabel('Male Viewers')
plt.hist(otherShrek[:],bins=9)
plt.title('Distribution of Ratings of Self-identified-gendered Viewers')
plt.xlabel('Ratings')
plt.ylabel('Self-identified-gendered Viewers')
#clearly, the data isn't normally distributed, so a non-parametric test may be more appropriate
u1,up1 = stats.mannwhitneyu(femaleShrek[:],maleShrek[:]) #Mann-Whitney U test: female and male
print("Test statistic U for Mann-Whitney U test of female and male ratings:", u1)
print("p-value for Mann-Whitney U test of female and male ratings:", up1)
print()
u2,up2 = stats.mannwhitneyu(femaleShrek[:],otherShrek[:]) #Mann-Whitney U test: female and self-described
print("Test statistic U for Mann-Whitney U test of female and self-described gender ratings:", u2)
print("p-value for Mann-Whitney U test of female and self-described gender ratings:", up2)
print()
u3,up3 = stats.mannwhitneyu(maleShrek[:],otherShrek[:]) #Mann-Whitney U test: male and self-described
print("Test statistic U for Mann-Whitney U test of male and self-described gender ratings:", u3)
print("p-value for Mann-Whitney U test of male and self-described gender ratings:", up3)
print()

#%% Q5: Do people who are only children enjoy ‘The Lion King (1994)’ more than people with siblings?
onlychild = data.iloc[:, 475]
dfChild = onlychild.to_frame()
lionKingList = ['The Lion King (1994)']
lionKingRating = rating[lionKingList]
lionKingRatingChild = pd.concat([lionKingRating, dfChild], axis = 1)
lionKingRatingChild.dropna(inplace = True)
lionKingRatingChild.sort_values(by=['Are you an only child? (1: Yes; 0: No; -1: Did not respond)'],inplace=True)
arrlionKingRatingChild = lionKingRatingChild.to_numpy()

# separate the participants who have siblings from those who are only children
siblingLionKing = arrlionKingRatingChild[10:786,0]
onlyLionKing = arrlionKingRatingChild[786:,0]

#null hypothesis: the ratings have no difference no matter people are only children or not
plt.hist(siblingLionKing[:],bins=9)
plt.title('Distribution of Ratings for Lion King of Viewers Who Have Siblings')
plt.xlabel('Ratings')
plt.ylabel('Viewers with Siblings')

plt.hist(onlyLionKing[:],bins=9)
plt.title('Distribution of Ratings for Lion King of Viewers Who are Only Child')
plt.xlabel('Ratings')
plt.ylabel('Viewers Who are Only Child')

u4, up4 = stats.mannwhitneyu(onlyLionKing[:],siblingLionKing[:])
#Mann-Whitney U test: only children and with siblings
print("- Test statistic U for Mann-Whitney U test of ratings of viewers who are only child and viewers who have siblings:", u4)
print("- p-value for Mann-Whitney U test of ratings of viewers who are only child and viewers who have siblings:", up4)

#%% Q6: Do people who like to watch movies socially enjoy ‘The Wolf of Wall Street (2013)’ more than those who prefer to watch them alone?
socialAlone = data.iloc[:, 476]
dfSocialAlone = socialAlone.to_frame()
WWSList = ['The Wolf of Wall Street (2013)']
WWSRating = rating[WWSList]
WWSRatingSocial = pd.concat([WWSRating, dfSocialAlone], axis = 1)
WWSRatingSocial.dropna(inplace = True)
WWSRatingSocial.sort_values(by=['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)'],inplace=True)
arrWWSRatingSocial = WWSRatingSocial.to_numpy()

socialWWS = arrWWSRatingSocial[4:274,0]
aloneWWS = arrWWSRatingSocial[274:,0]

plt.hist(socialWWS[:],bins=9)
plt.title('Distribution of Ratings for the Wolf of Wall Street of Viewers Who Like to Watch Movies Socially')
plt.xlabel('Ratings')
plt.ylabel('Viewers Who Like to Watch Movies Socially')

plt.hist(aloneWWS[:],bins=9)
plt.title('Distribution of Ratings for the Wolf of Wall Street of Viewers Who Like to Watch Movies Alone')
plt.xlabel('Ratings')
plt.ylabel('Viewers Who Like to Watch Movies Alone')

u5, up5 = stats.mannwhitneyu(socialWWS[:],aloneWWS[:]) #Mann-Whitney U test: watch movies socially (with others) and alone
print("- Test statistic U for Mann-Whitney U test of ratings of viewers who prefer watching movies socially and viewers who prefer watching movies alone:", u5)
print("- p-value for Mann-Whitney U test of social viewers and viewers alone ratings:", up5)

#%% Q7:There are ratings on movies from several franchises ([‘Star Wars’, ‘Harry Potter’, ‘The Matrix’, ‘Indiana Jones’, ‘Jurassic Park’, #‘Pirates of the Caribbean’, ‘Toy Story’, ‘Batman’]) in this dataset. How many of these are of inconsistent quality, as experienced by viewers?
```

```
#star wars:
starWars_movie_list = ['Star Wars: Episode 1 - The Phantom Menace
(1999)',
                'Star Wars: Episode II - Attack of the Clones (2002)',
                'Star Wars: Episode IV - A New Hope (1977)',
                'Star Wars: Episode V - The Empire Strikes Back (1980)',
                'Star Wars: Episode VI - The Return of the Jedi (1983)',
                'Star Wars: Episode VII - The Force Awakens (2015)']
starWars = rating[starWars_movie_list]
starWars.dropna(inplace = True)
arrStarWars = starWars.to_numpy()
arrSW1 = arrStarWars[:,0]
arrSW2 = arrStarWars[:,1]
arrSW4 = arrStarWars[:,2]
arrSW5 = arrStarWars[:,3]
arrSW6 = arrStarWars[:,4]
arrSW7 = arrStarWars[:,5]

#first we can look at some descriptive statistics to get an overview of
the data
numMovies = 6
descriptivesContainer = np.empty([numMovies,4]) #Initialize as empty
descriptivesContainer[:] = np.NaN  #Filling them with nans to begin
with

for ii in range(numMovies):
   descriptivesContainer[ii,0] = np.mean(arrStarWars[:,ii]) # Mean
   descriptivesContainer[ii,1] = np.std(arrStarWars[:,ii]) # SD
   descriptivesContainer[ii,2] = len(arrStarWars[:,ii]) # n
   descriptivesContainer[ii,3] =
descriptivesContainer[ii,1]/np.sqrt(descriptivesContainer[ii,2]) # sem

print(descriptivesContainer)

swf,swp =
stats.f_oneway(arrStarWars[:,0],arrStarWars[:,1],arrStarWars[:,2],arrSt
arWars[:,3],arrStarWars[:,4],arrStarWars[:,5])
print("- f-statistic for ANOVA of Star Wars: ", swf)
print("- p-value for ANOVA of Star Wars: ", swp)

#harry potter:
harry_movie_list = ['Harry Potter and the Sorcerer\'s Stone (2001)',
                'Harry Potter and the Chamber of Secrets (2002)',
                'Harry Potter and the Goblet of Fire (2005)',
                'Harry Potter and the Deathly Hallows: Part 2 (2011)']

harry = rating[harry_movie_list]
harry.dropna(inplace = True)
arrHarry = harry.to_numpy()
arrHP1 = arrHarry[:,0]
arrHP2 = arrHarry[:,1]
arrHP3 = arrHarry[:,2]
arrHP4 = arrHarry[:,3]

#first we can look at some descriptive statistics to get an overview of
the data
numMovies = 4
descriptivesContainer = np.empty([numMovies,4]) #Initialize as empty
descriptivesContainer[:] = np.NaN  #Filling them with nans to begin
with

for ii in range(numMovies):
   descriptivesContainer[ii,0] = np.mean(arrHarry[:,ii]) # Mean
   descriptivesContainer[ii,1] = np.std(arrHarry[:,ii]) # SD
   descriptivesContainer[ii,2] = len(arrHarry[:,ii]) # n
   descriptivesContainer[ii,3] =
descriptivesContainer[ii,1]/np.sqrt(descriptivesContainer[ii,2]) # sem

print(descriptivesContainer)

hpf,hpp =
stats.f_oneway(arrHarry[:,0],arrHarry[:,1],arrHarry[:,2],arrHarry[:,3])
print("- f-statistic for ANOVA of Harry Potter: ", hpf)
print("- p-value for ANOVA of Harry Potter: ", hpp)


#the matrix:
matrix_list = ['The Matrix (1999)',
          'The Matrix Reloaded (2003)',
          'The Matrix Revolutions (2003)']

matrix = rating[matrix_list]
matrix.dropna(inplace = True)
arrMatrix = matrix.to_numpy()
arrM1 = arrMatrix[:,0]
arrM2 = arrMatrix[:,1]
arrM3 = arrMatrix[:,2]

#first we can look at some descriptive statistics to get an overview of
the data
numMovies = 3
descriptivesContainer = np.empty([numMovies,4]) #Initialize as empty
descriptivesContainer[:] = np.NaN  #Filling them with nans to begin
with

for ii in range(numMovies):
   descriptivesContainer[ii,0] = np.mean(arrMatrix[:,ii]) # Mean
   descriptivesContainer[ii,1] = np.std(arrMatrix[:,ii]) # SD
   descriptivesContainer[ii,2] = len(arrMatrix[:,ii]) # n
   descriptivesContainer[ii,3] =
descriptivesContainer[ii,1]/np.sqrt(descriptivesContainer[ii,2]) # sem

print(descriptivesContainer)

mf,mp = stats.f_oneway(arrMatrix[:,0],arrMatrix[:,1],arrMatrix[:,2])
print("- f-statistic for ANOVA of Matrix: ", mf)
print("- p-value for ANOVA of Matrix: ", mp)


#Indiana Jones
IJ_list = ['Indiana Jones and the Raiders of the Lost Ark (1981)',
        'Indiana Jones and the Temple of Doom (1984)',
        'Indiana Jones and the Last Crusade (1989)',
        'Indiana Jones and the Kingdom of the Crystal Skull (2008)']

IJ = rating[IJ_list]
IJ.dropna(inplace = True)
arrIJ = IJ.to_numpy()
arrIJ1 = arrIJ[:,0]
arrIJ2 = arrIJ[:,1]
arrIJ3 = arrIJ[:,2]
arrIJ4 = arrIJ[:,3]

#first we can look at some descriptive statistics to get an overview of
the data
numMovies = 4
descriptivesContainer = np.empty([numMovies,4]) #Initialize as empty
descriptivesContainer[:] = np.NaN  #Filling them with nans to begin
with

for ii in range(numMovies):
   descriptivesContainer[ii,0] = np.mean(arrIJ[:,ii]) # Mean
   descriptivesContainer[ii,1] = np.std(arrIJ[:,ii]) # SD
   descriptivesContainer[ii,2] = len(arrIJ[:,ii]) # n
   descriptivesContainer[ii,3] =
descriptivesContainer[ii,1]/np.sqrt(descriptivesContainer[ii,2]) # sem

print(descriptivesContainer)

ijf,ijp = stats.f_oneway(arrIJ[:,0],arrIJ[:,1],arrIJ[:,2],arrIJ[:,3])
```

```
print("- f-statistic for ANOVA of Indiana Jones: ", ijf)
print("- p-value for ANOVA of Indiana Jones: ", ijp)


#Jurassic park
JP_list = ['Jurassic Park (1993)',
        'The Lost World: Jurassic Park (1997)',
        'Jurassic Park III (2001)']

JP = rating[JP_list]
JP.dropna(inplace = True)
arrJP = JP.to_numpy()
arrJP1 = arrJP[:,0]
arrJP2 = arrJP[:,1]
arrJP3 = arrJP[:,2]

#first we can look at some descriptive statistics to get an overview of
the data
numMovies = 3
descriptivesContainer = np.empty([numMovies,4]) #Initialize as empty
descriptivesContainer[:] = np.NaN  #Filling them with nans to begin
with

for ii in range(numMovies):
   descriptivesContainer[ii,0] = np.mean(arrJP[:,ii]) # Mean
   descriptivesContainer[ii,1] = np.std(arrJP[:,ii]) # SD
   descriptivesContainer[ii,2] = len(arrJP[:,ii]) # n
   descriptivesContainer[ii,3] =
descriptivesContainer[ii,1]/np.sqrt(descriptivesContainer[ii,2]) # sem

print(descriptivesContainer)

jpf,jpp = stats.f_oneway(arrJP[:,0],arrJP[:,1],arrJP[:,2])
print("- f-statistic for ANOVA of Jurassic Park: ", jpf)
print("- p-value for ANOVA of Jurassic Park: ", jpp)


#pirates of the caribbean
caribbean_list = ['Pirates of the Caribbean: The Curse of the Black Pearl
(2003)',
            'Pirates of the Caribbean: Dead Man\'s Chest (2006)',
            'Pirates of the Caribbean: At World\'s End (2007)']

caribbean = rating[caribbean_list]
caribbean.dropna(inplace = True)
arrCaribbean = caribbean.to_numpy()
arrPC1 = arrCaribbean[:,0]
arrPC2 = arrCaribbean[:,1]
arrPC3 = arrCaribbean[:,2]

#first we can look at some descriptive statistics to get an overview of
the data
numMovies = 3
descriptivesContainer = np.empty([numMovies,4]) #Initialize as empty
descriptivesContainer[:] = np.NaN  #Filling them with nans to begin
with

for ii in range(numMovies):
   descriptivesContainer[ii,0] = np.mean(arrCaribbean[:,ii]) # Mean
   descriptivesContainer[ii,1] = np.std(arrCaribbean[:,ii]) # SD
   descriptivesContainer[ii,2] = len(arrCaribbean[:,ii]) # n
   descriptivesContainer[ii,3] =
descriptivesContainer[ii,1]/np.sqrt(descriptivesContainer[ii,2]) # sem

print(descriptivesContainer)

pcf,pcp =
stats.f_oneway(arrCaribbean[:,0],arrCaribbean[:,1],arrCaribbean[:,2])
print("- f-statistic for ANOVA of Pirates of the Caribbean: ", pcf)
print("- p-value for ANOVA of Pirates of the Caribbean: ", pcp)


#Toy Story
TS_list = ['Toy Story (1995)',
        'Toy Story 2 (1999)',
        'Toy Story 3 (2010)']

TS = rating[TS_list]
TS.dropna(inplace = True)
arrTS = TS.to_numpy()
arrTS1 = arrTS[:,0]
arrTS2 = arrTS[:,1]
arrTS3 = arrTS[:,2]

#first we can look at some descriptive statistics to get an overview of
the data
numMovies = 3
descriptivesContainer = np.empty([numMovies,4]) #Initialize as empty
descriptivesContainer[:] = np.NaN  #Filling them with nans to begin
with

for ii in range(numMovies):
   descriptivesContainer[ii,0] = np.mean(arrTS[:,ii]) # Mean
   descriptivesContainer[ii,1] = np.std(arrTS[:,ii]) # SD
   descriptivesContainer[ii,2] = len(arrTS[:,ii]) # n
   descriptivesContainer[ii,3] =
descriptivesContainer[ii,1]/np.sqrt(descriptivesContainer[ii,2]) # sem

print(descriptivesContainer)

tsf,tsp = stats.f_oneway(arrTS[:,0],arrTS[:,1],arrTS[:,2])
print("- f-statistic for ANOVA of Toy Story: ", tsf)
print("- p-value for ANOVA of Toy Story: ", tsp)


#Batman
Batman_list = ['Batman (1989)',
            'Batman & Robin (1997)',
            'Batman: The Dark Knight (2008)']

Batman = rating[Batman_list]
Batman.dropna(inplace = True)
arrBatman = Batman.to_numpy()
arrBatman1 = arrBatman[:,0]
arrBatman2 = arrBatman[:,1]
arrBatman3 = arrBatman[:,2]

#first we can look at some descriptive statistics to get an overview of
the data
numMovies = 3
descriptivesContainer = np.empty([numMovies,4]) #Initialize as empty
descriptivesContainer[:] = np.NaN  #Filling them with nans to begin
with

for ii in range(numMovies):
   descriptivesContainer[ii,0] = np.mean(arrBatman[:,ii]) # Mean
   descriptivesContainer[ii,1] = np.std(arrBatman[:,ii]) # SD
   descriptivesContainer[ii,2] = len(arrBatman[:,ii]) # n
   descriptivesContainer[ii,3] =
descriptivesContainer[ii,1]/np.sqrt(descriptivesContainer[ii,2]) # sem

print(descriptivesContainer)

batmanf,batmanp =
stats.f_oneway(arrBatman[:,0],arrBatman[:,1],arrBatman[:,2])
print("- f-statistic for ANOVA of Batman: ", batmanf)
print("- p-value for ANOVA of Batman: ", batmanp)
```

```
#%% Q8:Build a prediction model of your choice (regression or supervised learning) to predict movie
#ratings (for all 400 movies) from personality factors only. Make sure to use cross-validation
#methods to avoid overfitting and characterize the accuracy of your model.


y_Rating = rating.to_numpy()

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(data.iloc[:, 420:464])
imputed_personality = imputer.transform(data.iloc[:, 420:464].values)

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(rating)
imputed_rating = imputer.transform(rating.values)


personality_train, personality_test, rating_train, rating_test = train_test_split(imputed_personality, imputed_rating, test_size = 0.5)

regressor = LinearRegression()
regressor.fit(personality_train, rating_train)

rating_predict = regressor.predict(personality_test)

print(rating_predict)

print("R-squared of the prediction model is:", metrics.r2_score(rating_test, rating_predict))
print("RMSE of the prediction model is:", np.sqrt(metrics.mean_squared_error(rating_test, rating_predict)))

#%% Q9:Build a prediction model of your choice (regression or supervised learning) to predict movie
#ratings (for all 400 movies) from gender identity, sibship status and social viewing preferences (columns 475-477)
#only. Make sure to use cross-validation methods to avoid overfitting and characterize the accuracy of your model.

#data cleaning (pruning)
#generate an array/ dataframe with 3 columns: gender, sibship, social viewing
#these are our X (predictors)
gender_sibship_social = data.iloc[:, 474:477]

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(gender_sibship_social)
imputed_gss = imputer.transform(gender_sibship_social.values)

gss_train, gss_test, rating_train_gss, rating_test_gss = train_test_split(imputed_gss, imputed_rating, test_size = 0.5)

regressor = LinearRegression()
regressor.fit(gss_train, rating_train_gss)

rating_predict_gss = regressor.predict(gss_test)
print(rating_predict_gss)

print("R-squared of the prediction model is:", metrics.r2_score(rating_test_gss, rating_predict_gss))
print("RMSE of the prediction model is:", np.sqrt(metrics.mean_squared_error(rating_test_gss, rating_predict_gss)))


#%% Q10: Build a prediction model of your choice (regression or supervised learning) to predict movie ratings (for all 400 movies) from all available factors that are not movie ratings (columns 401- 477). Make sure to use cross-validation methods to avoid overfitting and characterize the accuracy of your model.

allFactors = data.iloc[:, 401:477]

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(allFactors)
imputed_af = imputer.transform(allFactors.values)

af_train, af_test, rating_train_af, rating_test_af = train_test_split(imputed_af, imputed_rating, test_size = 0.5)

regressor = LinearRegression()
regressor.fit(af_train, rating_train_af)

rating_predict_af = regressor.predict(af_test)
print(rating_predict_af)

print("R-squared of the prediction model is:", metrics.r2_score(rating_test_af, rating_predict_af))
print("RMSE of the prediction model is:", np.sqrt(metrics.mean_squared_error(rating_test_af, rating_predict_af)))


#%% Extra credit

gender = data.iloc[:, 474]
dfGender = gender.to_frame()
bendList = ['Bend it Like Beckham (2002)']
bendRating = rating[bendList]
bendRatingGender = pd.concat([bendRating, dfGender],axis=1)
bendRatingGender.dropna(inplace = True)

#sort the dataframe according to gender
bendRatingGender.sort_values(by=['Gender identity (1 = female; 2 = male; 3 = self-described)'],inplace=True)
arrBendRatingGender = bendRatingGender.to_numpy()

#put different genders into different arrays
femaleBend= arrBendRatingGender[0:294,0]
maleBend = arrBendRatingGender[294:372,0]


#null hypothesis: the ratings have no difference (all genders rate Shrek equally)
#first take a look at the distributions of ratings for different genders
plt.hist(femaleShrek[:],bins=9)
plt.title('Distribution of Ratings of Female Viewers')
plt.xlabel('Ratings')
plt.ylabel('Female Viewers')
plt.hist(maleShrek[:],bins=9)
plt.title('Distribution of Ratings of Male Viewers')
plt.xlabel('Ratings')
plt.ylabel('Male Viewers')
plt.hist(otherShrek[:],bins=9)
plt.title('Distribution of Ratings of Self-identified-gendered Viewers')
plt.xlabel('Ratings')
plt.ylabel('Self-identified-gendered Viewers')
#clearly, the data isn't normally distributed, so a non-parametric test may be more appropriate
ub,upb = stats.mannwhitneyu(femaleBend[:],maleBend[:]) #Mann-Whitney U test: female and male
print("Test statistic U for Mann-Whitney U test of female and male ratings:", ub)
print("p-value for Mann-Whitney U test of female and male ratings:", upb)
```