Tomisin Adeyemi

Introduction to Data Science Capstone Project

I. Introduction: How I Handled Dimension Reduction, Data Cleaning & Transformation

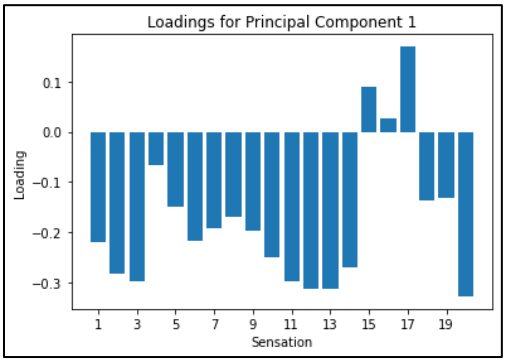
a. Dimension Reduction: For each question that involved analyzing user characteristics with one or more variables, a principal component analysis (PCA) was done to extract independent factors from the characteristics. The Keiser criterion was used to determine the number of independent factors. Usually, the first and second principal components were used solely to allow for more comprehensible analysis.

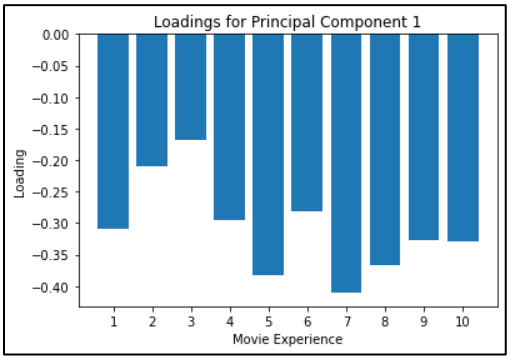
b. Data Cleaning: For each question, relevant data was extracted from the dataset and row wise removal of Nans was done (if needed) on this subset to make the data compatible with different Scikit-learn modules. (Each row the contained a NaN value was removed).

c. Data Transformation: Data was z-scored to normalize it before computing the PCA, then rotated in the new coordinate system of the relevant principal components.

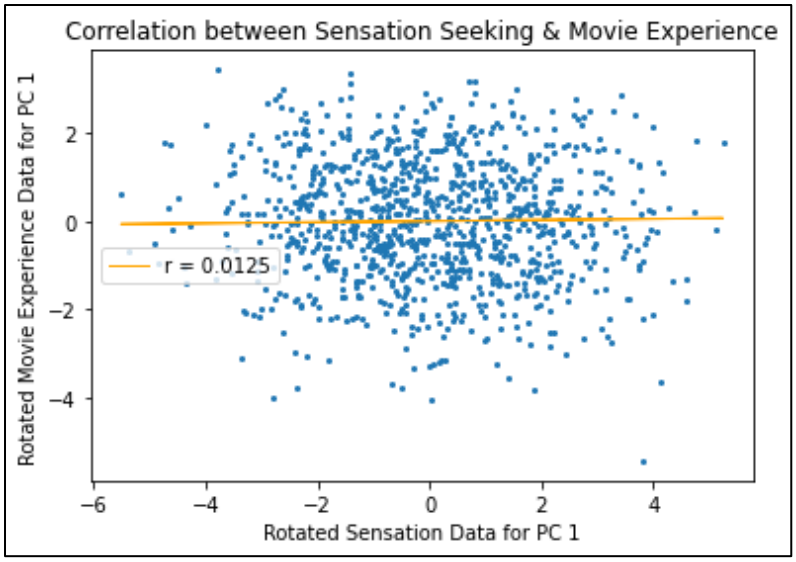
1. "What is the relationship between sensation seeking and movie experience?"

First, I extracted the "sensation seeking" and "movie experience" columns from the dataset, then combined them into a new subset and removed the NaN values row wise. 2 separate PCAs were done: one on sensation seeking, then one movie experience. Then I extracted the rotated data in the first Principal Component for both categories. The rotated data was found by performing a fit transform on the z-scored data, then multiplying by -1 because the direction is arbitrary. The Loadings matrix for the first principal component of sensation, and of movie experience are below:

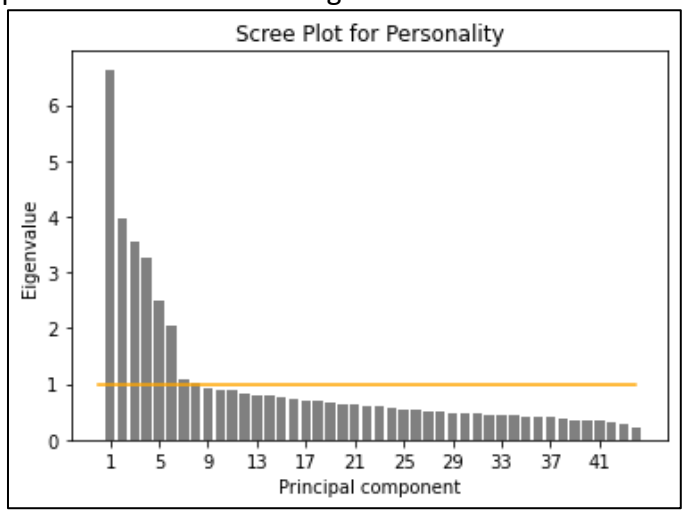




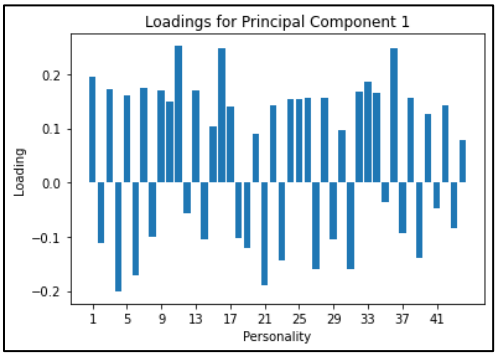
The correlation between sensation and movie experience was found by correlating first principal components of both sensation and movie experience in the new coordinate axis. According to this correlation, the relationship between sensation and movie experience is **0.01245** meaning that there is little to no correlation. The graph of this is below:

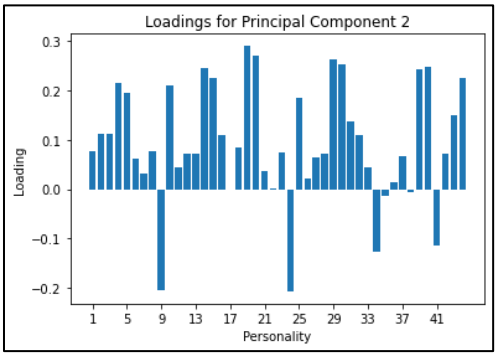


2. <u>"Is there evidence of personality types based on the data of these research participants? If so, characterize these types both quantitatively and narratively."</u>
   The personality columns were extracted from the data, and a PCA was performed resulting in 8 independent factors according to the Keiser criterion.



   As 8 factors will be cumbersome to deal with, the first 2 principal components are used. The Loadings for both are showed below:



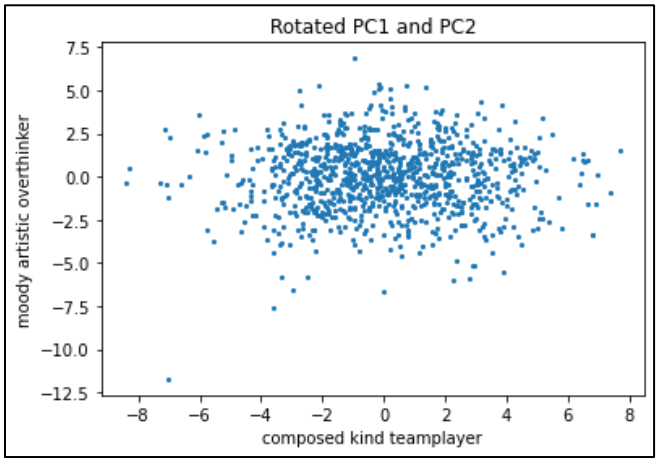


A threshold of eigenvectors greater than 0.15 was used to interpret the meaning of the first and second components. Based on this, Principal Component 1 and Principal Component 2 describe a person with the following characteristics:

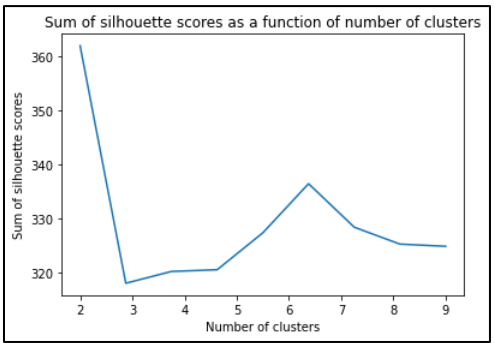
```
In [38]: PC1array
Out[38]:
['Is talkative',
 'Does a thorough job',
 'Is original/comes up with new ideas',
 'Is helpful and unselfish with others',
 'Is relaxed/handles stress well',
 'Is full of energy',
 'Is a reliable worker',
 'Generates a lot of Enthusiasm',
 'Is emotionally stable/not easily upset',
 'is inventive',
 'Has an assertive personality',
 'Perseveres until the task is finished',
 'Is considerate and kind to almost everyone',
 'Does things efficiently',
 'Remains calm in tense situations',
 'is outgoing/sociable',
 'Makes plans and follows through with them']

In [39]:
```

```
In [41]: PC2array
Out[41]:
['Is depressed/Blue',
 'Is original/comes up with new ideas',
 'Is curious about many different things',
 'Can be tense',
 'Is ingenious/a deep thinker',
 'Worries a lot',
 'Has an active imagination',
 'is inventive',
 'Can be moody',
 'Values artistic/aesthetic experiences',
 'Gets nervous easily',
 'Likes to reflect/play with ideas',
 'Is sophisticated in art or music or literature']
```

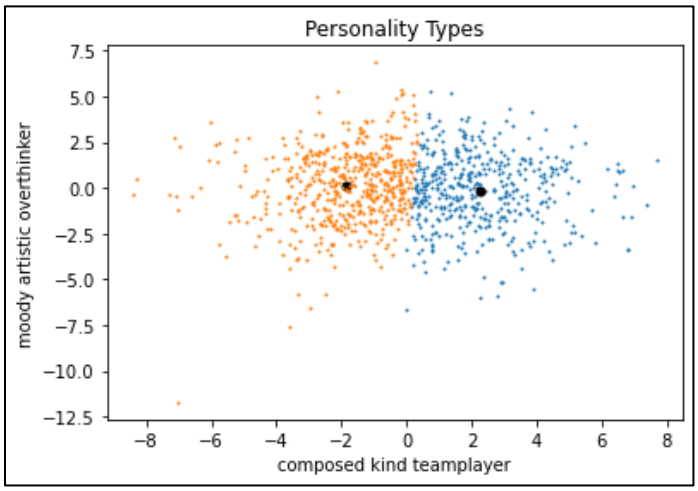
Based on these descriptions, principal component 1 describes the “composed kind teamplayer”: that one person that everyone likes, and principal component 2 describes the “moody artistic overthinker”: they can seem kind of reserved when you first meet them, but will say things that boggle your mind when you get close to them. Here is the graph of the rotated data in these axes:



To classify the personality types, KMeans Clustering was used, and the optimal k was found to be 2:

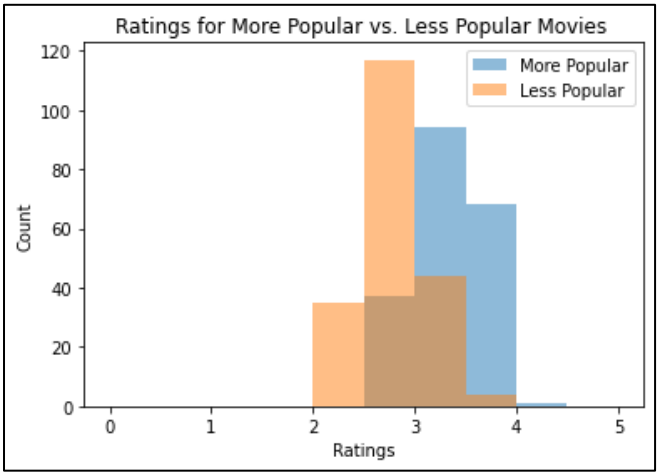


The clustered Personality types are as follows:



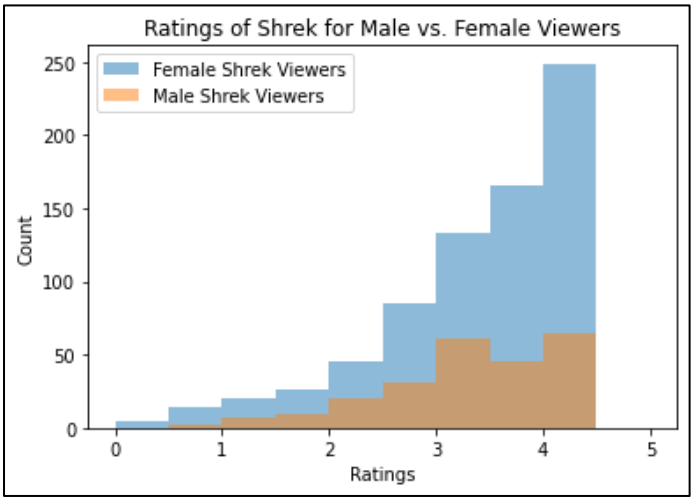
Based on this graph, people who observed these movies are of 2 types: the “composed kind teamplayer” and the people who are not.

3. “Are movies that are more popular rated higher than movies that are less popular? You can operationalize the popularity of a movie by how many ratings it has received”
   The movie ratings were extracted from the dataset, along with their counts and medians, they were then sorted by count. There was no NaN removal to not mess up the counts of each movie. The pandas median function automatically ignored NaNs. A median split was done as follows: the threshold for “popularity” was defined as the median of the “count” column: 197.5. A new column, “popularity” was added to the DataFrame, with values 0 and 1: 0 for a count below the median (less popular), and 1 for a count above the median(more popular). The data was then split into 2 groups: more popular and less popular. A histogram of these 2 groups is graphed below:

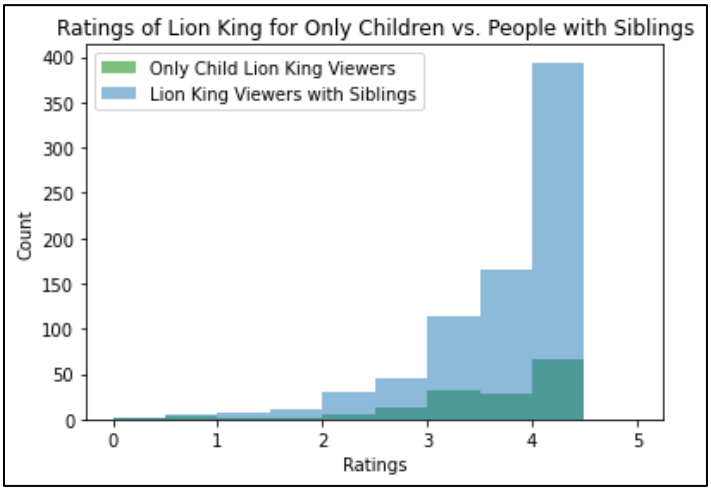


   A right-tailed mann-whitney u test was used to see if more popular movies were rated particularly *higher* than the less popular ones. This yielded a p-value of **9.9293e-35**, meaning that the difference in ratings between more popular and less popular (more popular movies being rated higher) cannot be due to chance alone, hence there is evidence that more popular movies are rated higher than the less popular ones.

4. <u>"Is enjoyment of 'Shrek (2001)' gendered, i.e. do male and female viewers rate it differently?"</u>
   The approach is similar as to before, Shrek and gender identity columns are extracted from the dataset and row wise removal of NaNs is done. Because the question asks for male and female viewers, removal of self-described people was also done (don't cancel me I did not really have a choice). The data was then split into male and female Shrek viewers, as shown by the histogram below:

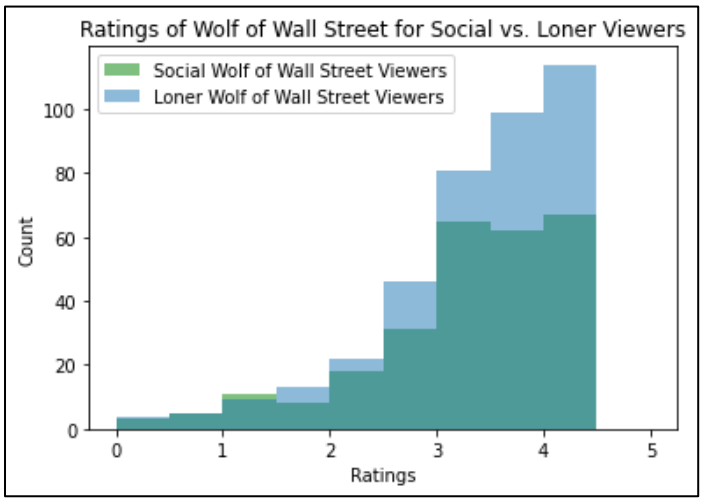


   A 2 tailed mann-whitney U test was done to find out if this difference in rankings was due to chance alone, yielding a p-value of **0.0505**, meaning that this difference from Male and Female is expected with chance, and there is no evidence that enjoyment of Shrek is gendered.
5. <u>Do people who are only children enjoy 'The Lion King (1994)' more than people with siblings?</u>
   Again, a similar approach is used to the questions above, the lion king and siblings data are extracted from the data and row wise removal of NaNs is done. There was also row wise removal of "-1" values, indicating no response for the siblings data. The data was then split into Lion King viewers with siblings and Lion Kings viewers that are only children:



A right tailed mann-whitney u test was done to see if only children enjoy lion king *more* than people with siblings, yielding a p-value of **0.97842**, meaning there is no evidence that people who are only children enjoy Lion King more than people with siblings.

6. <u>Do people who like to watch movies socially enjoy 'The Wolf of Wall Street (2013)' more than those who prefer to watch them alone?</u>
   A similar approach as above, wolf of wall street data and social preference data were extracted from the original dataset, then row wise removal of NaNs and "-1" values was done. The data was then split into social and loner viewers of the Wolf of Wall Street as shown below:



   A right tailed mann-whitney u test was done to examine whether people who like to watch movies socially enjoy The Wolf of Wall Street *more* than those who prefer to watch them alone, yielding a p-value of **0.9437**, meaning there is no evidence that people who like to watch movies socially enjoy The Wolf of Wall Street more than those who prefer to watch them alone.

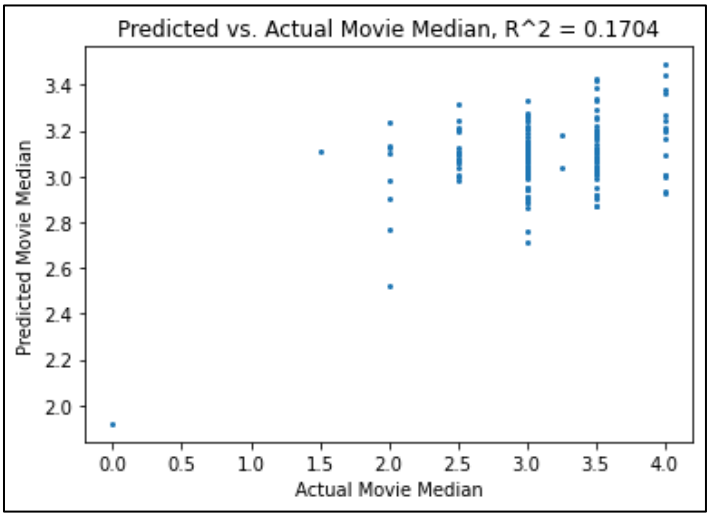
7. <u>There are ratings on movies from several franchises (['Star Wars', 'Harry Potter', 'The Matrix', 'Indiana Jones', 'Jurassic Park', 'Pirates of the Caribbean', 'Toy Story', 'Batman']) in this dataset. How many of these are of inconsistent quality, as experienced by viewers?</u>
   For each movie franchise, the movie columns were extracted from the data set, then a Kruskal Wallis test was done to see if there was inconsistency in the movies. If the p-value was less than 0.05, then the difference in the movie ratings was unlikely to just be due to chance, so the franchise was marked as of inconsistent quality. The result of this for each franchise is shown below:

ConsistencyOfFranchises - DataFrame

Index	Franchises	P-Values	tent with Cl
0	Star Wars	6.94016e-40	No
1	Harry Potter	0.117906	Yes
2	The Matrix	1.75373e-09	No
3	Indiana Jones	1.02012e-11	No
4	Jurassic Park	1.84923e-11	No
5	Pirates of the Caribbean	0.0357927	No
6	Toy Story	7.90223e-06	No
7	Batman	4.13805e-19	No

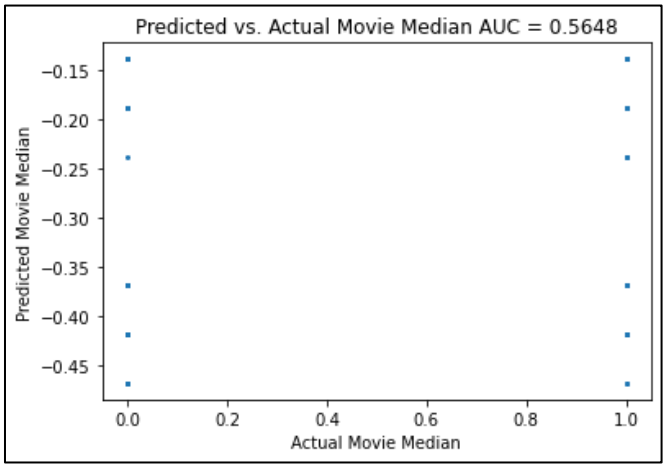
This shows that there is evidence that 7 out of the 8 movie franchises are of inconsistent quality according to viewers.

8. <u>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.</u>
The variable I decided to predict is the median rating each of person's entire movie viewing collection. I chose the median because it is a better measure of central tendency than the mean for ordinal data. I computed the median rating of each person's movie viewership history (python function used automatically accounts for NaNs), and combined this with the personality data. A PCA was done on the personality data, yielding 8 independent factors according to the Keiser criterion. The rotated data in these axes will be used as a predictor for a multiple linear regression model. To make sure that the model does not overfit, the data was split into train and test data sets using the train_test_split() method in scikit-learn. The test data size was 15% of the original data. The model was trained on train data and tested on the test data. Based on this, the RMSE was 0.5120, meaning on average predictions are off by 0.51 units. The R^2 was 0.17034, meaning 17.04% of variance in median ratings was explained by personality. A graph of the predicted vs. actual movie ratings are as follows.



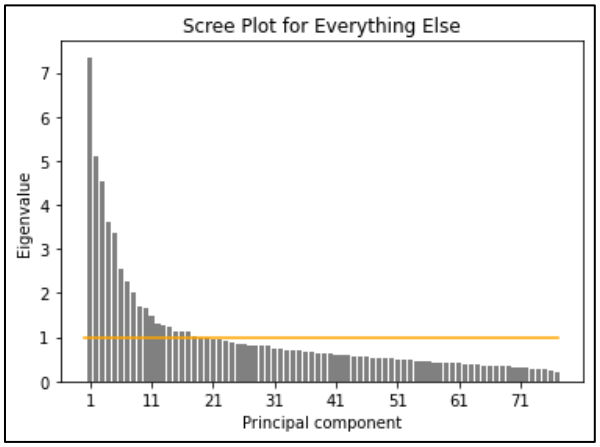
9. Build a prediction model of your choice (regression or supervised learning) to predict movie ratings (for all 400 movies) from gender identity, sibship status and social viewing preferences (columns 475-477) only. Make sure to use cross-validation methods to avoid overfitting and characterize the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

   I extracted the relevant predictors from the original data set and concatenated them with the median movie ratings per person (as Q8 above). There was row-wise removal of NaNs and of 3, -1, and -1 from the gender identity, sibship status and social viewing status respectively. I then sorted the array by the median movie ratings and then found the median of this column: 3.0. To allow for an easier median split, I set the median threshold to 3.15. The outcome is now whether each median rating is above (1) or below (0) the median of the *median* ratings. Like above, I split the data into train and test data sets. The AUC of this model is **0.5648**.

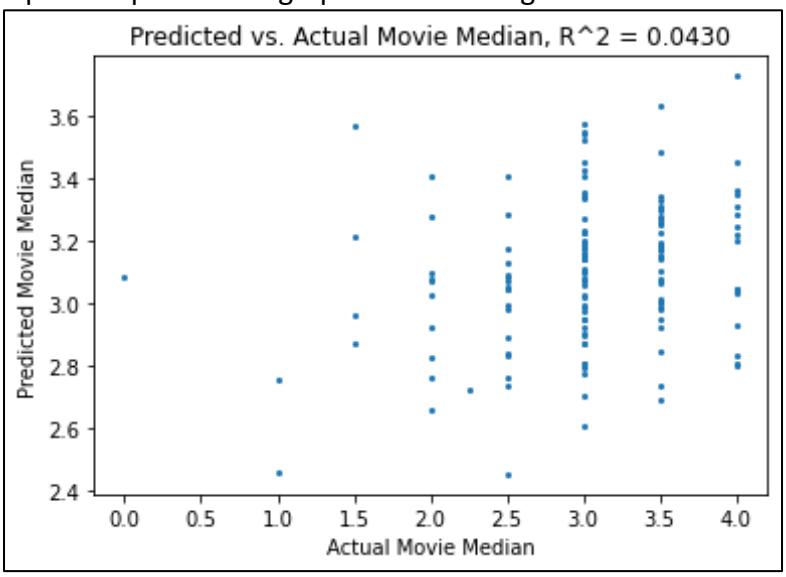


10. Build a prediction model of your choice (regression or supervised learning) to predict movie ratings (for all 400 movies) from all available factors that are not movie ratings (columns 401-477). Make sure to use cross-validation methods to avoid overfitting and characterize the accuracy of your model.

    Like before I am predicting the median movie ratings per person. I extracted "everything else": all the data that is not movie ratings, concatenated it with the median movie ratings and did row wise removal of NaN. I did a PCA on "everything else" and found there were 18 independent factors according to the Keiser criterion:



I extracted the rotated data for these 18 factors, which will serve as the predictor for the median movie ratings. Like before, I split the data into train and test data sets. The RMSE is 0.6696, meaning on average the predictions are off by 0.67 units. R^2 is 0.04296, meaning 4.3% of variance in median ratings is explained by the rotated data for the 18 Principal Components. A graph of the findings is below:



Code

```
#%% Loader
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from scipy.special import expit
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataPD = pd.read_csv("movieReplicationSet - Copy.csv", header=0)
dataColumns = dataPD.columns
dataNP = np.genfromtxt("movieReplicationSet - Copy.csv", delimiter=",",
skip_header=1)
# for refrence later
pcaList = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8',
'PC9', 'PC10']

#%% My functions
# should take in np array
def rowWiseRemovalOfNans(data):
    temp = np.isnan(data)
    temp2 = temp * 1
    sumOfEachRow = np.sum(temp2, axis=1)
    missingData = list(np.where(sumOfEachRow > 0)[0])
    output =  np.delete(data, obj=missingData, axis=0)
    return output

# data should have no nans
def computePCA(data, characteristicLength, characteristicName, step):
    zScoredData = stats.zscore(data)
    pca =  PCA().fit(zScoredData)
    eigVals = pca.explained_variance_
    loadings = pca.components_
    #rotated = pca.fit_transform(zScoredData)
    #varExplained = eigVals/sum(eigVals) * 100

    keiserFits = list(np.where(eigVals > 1)[0])

    x = np.linspace(1, characteristicLength, characteristicLength)
    plt.bar(x, eigVals, color='gray')
    plt.plot([0,characteristicLength],[1,1],color='orange')
    plt.xlabel('Principal component')
```

```
    plt.xticks(np.arange(1, characteristicLength+1, step=step))
    plt.ylabel('Eigenvalue')
    plt.title("Scree Plot for " + characteristicName)
    plt.show()

    for index in keiserFits:
        plt.bar(x,loadings[index,:]*-1)
        plt.xlabel(characteristicName)
        plt.xticks(np.arange(1, characteristicLength+1, step=step))
        plt.ylabel('Loading')
        plt.title("Loadings for Principal Component " + str(index + 1))
        plt.show()
    return pca
    #sensationPC1 =
newSensationPCA.fit_transform(stats.zscore(newSensation))

def tabularizeLoadingsForKeiserPC(pca, characteristicColumns):
    loadings = -1 * pca.components_
    keiserFits = list(np.where(pca.explained_variance_ > 1)[0])
    correspondingCharacteristics = {}
    for index in keiserFits:
        correspondingCharacteristics[index + 1] =
[characteristicColumns[ii] for ii in np.where(loadings[index] > 0)]
    return correspondingCharacteristics

def comprehendPCA():
    pass

# too high for all personality characteristics
def ninetyPercentVariance(pca):
    threshold = 90
    eigVals = pca.explained_variance_
    varExplained = eigVals/sum(eigVals)*100
    eigSum = np.cumsum(varExplained)
    return np.count_nonzero(eigSum < threshold) + 1

# make note of python indexing
def plotCorrelationMatrix(corr, title, step):
    plt.imshow(corr)
    plt.xlabel(title)
    plt.xticks(np.arange(0, len(corr), step=step))
    plt.ylabel(title)
    plt.yticks(np.arange(0, len(corr), step=step))
    plt.colorbar()
    plt.title("Correlation Matrix For " + title)
    plt.show()

def getFranchise(name):
    output = []
    for ii in dataColumns:
```

```
        if name in ii:
            output.append(ii)
    return sorted(output)

def generateFranchise(names):
    df = pd.DataFrame()
    for name in names:
        df = pd.concat([df, dataPD[name]], axis=1)
    return df

def kruskal(name):
    names = getFranchise(name)
    data = generateFranchise(names)
    dataNoNans = rowWiseRemovalOfNans(data.to_numpy()) # 1097 --> 260
    return dataNoNans, len(names)
    # h, p = stats.kruskal(matrixNoNans[:, 0], matrixNoNans[:, 1],
matrixNoNans[:, 2])

#%% Question 1
# What is the relationship between sensation seeking and movie
experience?

sensationSeeking = dataPD.iloc[:, 400:420] #20
movieExperience = dataPD.iloc[:, 464:474] #10

sensationMovieExperience =  pd.concat([sensationSeeking,
movieExperience], axis=1)
sensationMovieExperienceNoNan =
rowWiseRemovalOfNans(sensationMovieExperience.to_numpy()) # 1097 -->
1029

newSensation = sensationMovieExperienceNoNan[:, :20]
sensationLength = len(newSensation[0, :])
newMovieExperience = sensationMovieExperienceNoNan[:, 20:]
movieExperienceLength = len(newMovieExperience[0, :])

sensationCorrelationMatrix = np.corrcoef(newSensation, rowvar=False)
plotCorrelationMatrix(sensationCorrelationMatrix, "Sensation", 2)
sensationPCA = computePCA(newSensation, sensationLength, "Sensation",
2)
sensationGroupedFactors = tabularizeLoadingsForKeiserPC(sensationPCA,
sensationSeeking.columns)
sensationRelevantFactors = len(sensationGroupedFactors)
rotatedSensation = -1 *
sensationPCA.fit_transform(stats.zscore(newSensation))[:,
:sensationRelevantFactors]

movieExperienceCorrelationMatrix = np.corrcoef(newMovieExperience,
rowvar=False)
```

```
plotCorrelationMatrix(movieExperienceCorrelationMatrix, "Movie
Experience", 1)
movieExperiencePCA = computePCA(newMovieExperience,
movieExperienceLength, "Movie Experience", 1)
movieExperienceGroupedFactors =
tabularizeLoadingsForKeiserPC(movieExperiencePCA,
movieExperience.columns)
movieExperienceRelevantFactors = len(movieExperienceGroupedFactors)
rotatedMovieExperience = -1 *
movieExperiencePCA.fit_transform(stats.zscore(newMovieExperience))[:,
:movieExperienceRelevantFactors]

# correlate PC1 of sensation seeking and movie experience
# talk about multiplying by -1 in preamble
# plot scatter plot of both with line
correlationSensationMovie = np.corrcoef(rotatedSensation[:, 0],
rotatedMovieExperience[:, 0])[0][1]

yHatSensationMovie = correlationSensationMovie * rotatedSensation[:, 0]
plt.plot(rotatedSensation[:, 0], rotatedMovieExperience[:, 1], 'o',
markersize=2)
# 'Rotated Sensation Data for PC 1'
plt.xlabel("Rotated Sensation Data for PC 1")
# 'Rotated Movie Experience Data for PC 1'
plt.ylabel('Rotated Movie Experience Data for PC 1')
plt.plot(rotatedSensation[:,0],yHatSensationMovie,color='orange',linewi
dth=1, label="r = {:.4f}".format(correlationSensationMovie))
plt.legend(loc="center left")
plt.title("Correlation between Sensation Seeking & Movie Experience")
plt.show()
#%% Question 2
# Is there evidence of personality types based on the data of these
research participants?
# If so, characterize these types both quantitatively and narratively.
personality = dataPD.iloc[:, 420:464]
personalityDescriptions = list(personality.columns)
personalityNoNan = rowWiseRemovalOfNans(personality.to_numpy())
personalityLength = len(personalityNoNan[0, :])

personalityCorrelationMatrix = np.corrcoef(personalityNoNan,
rowvar=False)
plotCorrelationMatrix(personalityCorrelationMatrix, "Personality", 4)
personalityPCA = computePCA(personalityNoNan, personalityLength,
"Personality", 4)
personalityGroupedFactors =
tabularizeLoadingsForKeiserPC(personalityPCA, personality.columns)
personalityRelevantFactors = len(personalityGroupedFactors)
rotatedPersonality = -1 *
personalityPCA.fit_transform(stats.zscore(personalityNoNan))[:,
:personalityRelevantFactors]
```

```
# from 44 categories to 8 factors
personalityLoadings = -1 *
personalityPCA.components_[:personalityRelevantFactors]
personalityLoadingsTranspose = personalityLoadings.T

personalityLoadingsLabeled = pd.DataFrame(personalityLoadingsTranspose,
columns=pcaList[:8],
                                          index = np.arange(0, 44))

# 0.15 instead of 0 to make it more interpretable
personalityLoadingsThreshold = 0.15# changed from 0.15 to 0.2 to
account for more original categories in groupings
personalityAboveThreshold = np.empty([44, 8])
personalityAboveThreshold[:] = np.NaN
for column, PC in enumerate(personalityLoadingsLabeled.columns):
    for row, value in enumerate(personalityLoadingsLabeled[PC]):
        if value > personalityLoadingsThreshold:
            personalityAboveThreshold[row, column] = value

personalityAllPCs = pd.DataFrame(personalityAboveThreshold[:, :8],
index=personalityDescriptions, columns=pcaList[:8])

"""# to check rows not accounted for by threshold
for row in range(0, len(personalityAboveThreshold)):
    if np.nansum(personalityAboveThreshold[row]) == 0:
        print(row) # only 17 (0 indexing), "tends to be disorganized"""

# check number of rows not accounted for
count = 0
for row in range(0, 44):
    if np.isnan(personalityAboveThreshold[:, :2][row][0]) and
np.isnan(personalityAboveThreshold[:, :2][row][1]):
        count += 1
        # 16 not accounted for w/ 0.15; 28 accounted for

# to comprehend PC1 and PC2
personalityPC1PC2 = pd.DataFrame(personalityAboveThreshold[:, :2],
index=personalityDescriptions, columns=pcaList[:2])
# On first look (show arrays that represent them):
PC1criteria = np.where(np.isnan(personalityPC1PC2['PC1']) == False)[0]
PC1array = []
for ii in PC1criteria:
    PC1array.append(personalityPC1PC2.index[ii])
PC2criteria = np.where(np.isnan(personalityPC1PC2['PC2']) == False)[0]
PC2array = []
for ii in PC2criteria:
    PC2array.append(personalityPC1PC2.index[ii])
# PC1: composed kind teamplayer  "everyone likes"
# PC2: moody artistic overthinker "usually quiet but boggles minds when
they speak"
```

```
plt.plot(rotatedPersonality[:, 0], rotatedPersonality[:, 1], 'o',
markersize=2)
plt.xlabel("composed kind teamplayer")
plt.ylabel("moody artistic overthinker")
plt.title("Rotated PC1 and PC2")
plt.show()

# CLUSTERING
# should be x by 2
# transformed data are predictores
rotatedPersonalityPredictors =
np.column_stack((rotatedPersonality[:,0],rotatedPersonality[:,1]))
numClusters = 9
sSum = np.empty([numClusters,1])*np.NaN

# Compute kMeans for each k:
for ii in range(2, numClusters+2):
    kMeans = KMeans(n_clusters =
int(ii)).fit(rotatedPersonalityPredictors)
    cId = kMeans.labels_
    cCoords = kMeans.cluster_centers_
    s = silhouette_samples(rotatedPersonalityPredictors,cId)
    sSum[ii-2] = sum(s)
    # Plot data:
    plt.subplot(3,3,ii-1)
    plt.hist(s,bins=20)
    plt.xlim(-0.2,1)
    plt.ylim(0,150)
    plt.xlabel('Silhouette score')
    plt.ylabel('Count')
    plt.title('Sum: {}'.format(int(sSum[ii-2])) + " , k:
{}".format(ii))
    plt.tight_layout()

# Plot the sum of the silhouette scores as a function of the number of
clusters
plt.plot(np.linspace(2,numClusters,9),sSum)
plt.title("Sum of silhouette scores as a function of number of
clusters")
plt.xlabel('Number of clusters')
plt.ylabel('Sum of silhouette scores')
plt.show()

# 2 has highest sum so 2 clusters

numClusters = 2
kMeans = KMeans(n_clusters =
numClusters).fit(rotatedPersonalityPredictors)
cId = kMeans.labels_
```

```
cCoords = kMeans.cluster_centers_

# Plot the color-coded data:
for ii in range(numClusters):
    plotIndex = np.argwhere(cId == int(ii))

plt.plot(rotatedPersonalityPredictors[plotIndex,0],rotatedPersonalityPr
edictors[plotIndex,1],'o',markersize=1)
    plt.plot(cCoords[int(ii-1),0],cCoords[int(ii-
1),1],'o',markersize=5,color='black')
    plt.xlabel("composed kind teamplayer")
    plt.ylabel("moody artistic overthinker")
    plt.title("Personality Types")
# qualitatively describe these categories

#%% Question 3
# Are movies that are more popular rated higher than movies that are
less popular?
# You can operationalize the popularity of a movie by how many ratings
it has received.
movieRatings = dataPD.iloc[:, :400]
movieRatingsCount = movieRatings.describe().iloc[0, :]
movieRatingsMedian = movieRatings.median()
movieRatingsCountMedian = pd.concat([movieRatingsCount,
movieRatingsMedian], axis=1)
movieRatingsCountMedian.columns = ['count', 'median']
movieRatingsCountMedian = movieRatingsCountMedian.sort_values("count")
# 1 : above median
# 0 : below median
popularity = []

popularityThreshold = movieRatingsCountMedian['count'].median()

for ii in range(0, 400):
    popularity.append(1) if movieRatingsCountMedian['count'][ii] >
popularityThreshold else popularity.append(0)

movieRatingsCountMedian['popularity'] = popularity
popularityBelowAverage = movieRatingsCountMedian.loc[:'Knight and Day
(2010)', :]['median']
popularityAboveAverage = movieRatingsCountMedian.loc['Anaconda
(1997)':, :]['median']

uPopularity, pPopularity =
stats.mannwhitneyu(popularityAboveAverage.to_numpy(),
popularityBelowAverage.to_numpy(), alternative='greater')

# visual inspection seems like it
# p value after mann whitney u = 9.929258851707232e-35
```

```
# according to ... there is evidence that movies that are more popular
rated higher than movies that are less popular
# nan removal not necessary as using median and count

# graph
bins = np.linspace(0, 5, 11)
plt.hist(popularityAboveAverage, bins, alpha=0.5, label='More Popular')
plt.hist(popularityBelowAverage, bins, alpha=0.5, label='Less Popular')
plt.xlabel('Ratings')
plt.ylabel("Count")
plt.title("Ratings for More Popular vs. Less Popular Movies ")
plt.legend(loc='upper right')
plt.show()

#%% Question 4
# Is enjoyment of ‘Shrek (2001)’ gendered, i.e. do male and female
viewers rate it differently?
shrek = dataPD['Shrek (2001)']
genderIdentity = dataPD['Gender identity (1 = female; 2 = male; 3 =
self-described)']
shrekGenderIdentity = pd.concat([shrek, genderIdentity], axis=1)
shrekGenderIdentity = shrekGenderIdentity.sort_values('Gender identity
(1 = female; 2 = male; 3 = self-described)')
shrekGenderIdentityNoNans =
rowWiseRemovalOfNans(shrekGenderIdentity.to_numpy())

# need to remove this cause question asks that, don't cancel me
selfDescribeStart = np.where(shrekGenderIdentityNoNans[:, 1] ==
3)[0][0]
shrekGenderIdentityNoNans =
shrekGenderIdentityNoNans[:selfDescribeStart]
# femaleEnd = np.where(shrekGenderIdentityNoNans[:, 1] == 1)[0][-1]
maleStart = np.where(shrekGenderIdentityNoNans[:, 1] == 2)[0][0]

femaleShrek = shrekGenderIdentityNoNans[:maleStart][:, 0] #743
maleShrek = shrekGenderIdentityNoNans[maleStart:][:, 0] #241

uShrekGender, pShrekGender = stats.mannwhitneyu(femaleShrek, maleShrek)

# 2 tailed mann whitney u
# p value after mann whitney u : 0.050536625925559006
# according to ... there is no evidence that enjoyment of ‘Shrek
(2001)’ gendered
# graph

bins = np.linspace(0, 5, 11)
plt.hist(femaleShrek, bins, alpha=0.5, label='Female Shrek Viewers')
plt.hist(maleShrek, bins, alpha=0.5, label='Male Shrek Viewers')
plt.xlabel('Ratings')
plt.ylabel("Count")
```

```
plt.title("Ratings of Shrek for Male vs. Female Viewers")
plt.legend(loc='upper left')
plt.show()

# though it seems like that visually, taking into account the number???
# research on number

#%% Question 5
# Do people who are only children enjoy ‘The Lion King (1994)’ more
than people with siblings?
lionKing = dataPD['The Lion King (1994)']
siblingStatus = dataPD['Are you an only child? (1: Yes; 0: No; -1: Did
not respond)']
lionKingSiblingStatus = pd.concat([lionKing, siblingStatus], axis=1)
lionKingSiblingStatus = lionKingSiblingStatus.sort_values('Are you an
only child? (1: Yes; 0: No; -1: Did not respond)')

lionKingSiblingStatusNoNans =
rowWiseRemovalOfNans(lionKingSiblingStatus.to_numpy())

lastNoResponseKing = np.where(lionKingSiblingStatusNoNans[:, 1] == -
1)[0][-1]
lionKingSiblingStatusNoNans =
lionKingSiblingStatusNoNans[lastNoResponseKing+1: ] # 927

onlyChildStart = np.where(lionKingSiblingStatusNoNans[:, 1] == 1)[0][0]

siblingsLionKing = lionKingSiblingStatusNoNans[:onlyChildStart][:, 0]
onlyChildLionKing = lionKingSiblingStatusNoNans[onlyChildStart:][:, 0]

# right tailed mann whitney u test to see if only child specifically >
siblings
# alternative = greater parameter in python
uLionKingSiblings, pLionKingSiblings =
stats.mannwhitneyu(onlyChildLionKing, siblingsLionKing,
alternative='greater')

# p value: 0.978419092554931
# according to ... there is no evidence that people who are only
children enjoy ‘The Lion King (1994)’ more than people with siblings

bins = np.linspace(0, 5, 11)
plt.hist(onlyChildLionKing, bins, alpha=0.5, label='Only Child Lion
King Viewers', color='green')
plt.hist(siblingsLionKing, bins, alpha=0.5, label='Lion King Viewers
with Siblings')
plt.xlabel('Ratings')
plt.ylabel("Count")
plt.title("Ratings of Lion King for Only Children vs. People with
Siblings")
```

```
plt.legend(loc='upper left')
plt.show()

#%% Question 6
# Do people who like to watch movies socially enjoy ‘The Wolf of Wall
Street (2013)’ more than
# those who prefer to watch them alone?
wolf = dataPD['The Wolf of Wall Street (2013)']
socialPreference = dataPD['Movies are best enjoyed alone (1: Yes; 0:
No; -1: Did not respond)']
wolfSocialPreference = pd.concat([wolf, socialPreference], axis=1)
wolfSocialPreference = wolfSocialPreference.sort_values('Movies are
best enjoyed alone (1: Yes; 0: No; -1: Did not respond)')

wolfSocialPreferenceNoNans =
rowWiseRemovalOfNans(wolfSocialPreference.to_numpy())
lastNoResponseWolf = np.where(wolfSocialPreferenceNoNans[:, 1] == -
1)[0][-1]
wolfSocialPreferenceNoNans =
wolfSocialPreferenceNoNans[lastNoResponseWolf+1:]

aloneStart = np.where(wolfSocialPreferenceNoNans[:, 1] == 1)[0][0]

socialWolf = wolfSocialPreferenceNoNans[:aloneStart][:, 0] # 0
aloneWolf = wolfSocialPreferenceNoNans[aloneStart:][:, 0] # 1

# social > alone ?? right tailed mann whitney u test
# like question 5
uWolfSocial, pWolfSocial = stats.mannwhitneyu(socialWolf, aloneWolf,
alternative='greater')

# p value: 0.9436657996253056
# according to ... there is no evidence that people who like to watch
movies socially enjoy ‘The Wolf of Wall Street (2013)’
# more than those who prefer to watch them alone

bins = np.linspace(0, 5, 11)
plt.hist(socialWolf, bins, alpha=0.5, label='Social Wolf of Wall Street
Viewers', color='green')
plt.hist(aloneWolf, bins, alpha=0.5, label='Loner Wolf of Wall Street
Viewers')
plt.xlabel('Ratings')
plt.ylabel("Count")
plt.title("Ratings of Wolf of Wall Street for Social vs. Loner
Viewers")
plt.legend(loc='upper left')
plt.show()

#%% Question 7
# 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?

# Kruskal Wallis
# row wise removal: advantages and disadvantages, possibilites if they
did not watch
# If smaller than alpha: Significant --> Movies are of inconsistent
quality
# difference unlikely by chance

# 1) STAR WARS
# star wars 3 missing
starWarsNoNans, starWarsLength = kruskal("Star Wars") # 1097 --> 333
# 6 movies in star Wars
hStarWars, pStarWars = stats.kruskal(starWarsNoNans[:, 0],
starWarsNoNans[:, 1], starWarsNoNans[:, 2], starWarsNoNans[:, 3],
                                     starWarsNoNans[:, 4],
starWarsNoNans[:, 5])
# p : 6.940162236984522e-40, there is evidence that movies are of
inconsistent quality

# 2) HARRY POTTER
harryPotterNoNans, harryPotterLength = kruskal("Harry Potter") # 1097 -
-> 710
# 4 movies in harry potter
hHarryPotter, pHarryPotter = stats.kruskal(harryPotterNoNans[:, 0],
harryPotterNoNans[:, 1], harryPotterNoNans[:, 2],
                                           harryPotterNoNans[:, 3])
# p: 0.11790622831256074, there is no evidence that movies are of
inconsistent quality

# 3) THE MATRIX
matrixNoNans, matrixLength = kruskal("Matrix") # 1097 --> 260
# 3 movies in the matrix
hMatrix, pMatrix = stats.kruskal(matrixNoNans[:, 0], matrixNoNans[:,
1], matrixNoNans[:, 2])
# p: 1.7537323830838066e-09, evidence that movies are of inconsistent
quality

# 4) INDIANA JONES
indianaJonesNoNans, indianLength = kruskal("Indiana Jones") # 1097 -->
244
# 4 movies in indiana jones
hIndiana, pIndiana = stats.kruskal(indianaJonesNoNans[:, 0],
indianaJonesNoNans[:, 1], indianaJonesNoNans[:, 2],
                                   indianaJonesNoNans[:, 3])
# p: 1.020118354785894e-11, evidence that movies are of incosistent
quality
```

```
# 5) JURASSIC PARK
jurassicParkNoNans, jurassicLength = kruskal("Jurassic") # 1097 --> 398
# 3 mvoies jurassic
hJurassic, pJurassic = stats.kruskal(jurassicParkNoNans[:, 0],
jurassicParkNoNans[:, 1], jurassicParkNoNans[:, 2])
# p: 1.8492328391686058e-11, evidence that movies are of incosistent
quality

# 6) PIRATES OF THE CARIBBEAN
piratesNoNans, piratesLength = kruskal("Pirates")
hPirates, pPirates = stats.kruskal(piratesNoNans[:, 0],
piratesNoNans[:, 1], piratesNoNans[:, 2])
# p: 0.035792727694248905, evidence that movies are of inconsistent
quality

# 7) TOY STORY
toyNoNans, toyLength = kruskal("Toy")
hToy, pToy = stats.kruskal(toyNoNans[:, 0], toyNoNans[:, 1],
toyNoNans[:, 2])
# p: 7.902234665149812e-06, evidence that movies are of incosistent
quality

# 8) BATMAN
batManNoNans, batManLength = kruskal("Batman")
hBatMan, pBatMan = stats.kruskal(batManNoNans[:, 0], batManNoNans[:,
1], batManNoNans[:, 2])
# p: 4.1380499020034183e-19, evidence that movies are of incosistent
quality

# table to show values
# Franchise, p-value, incosistent w/quality (0.05)

ConsistencyOfFranchises = pd.DataFrame()

ConsistencyOfFranchises['Franchises'] = ['Star Wars', 'Harry Potter',
'The Matrix', 'Indiana Jones',
                                         'Jurassic Park', 'Pirates of
the Caribbean', 'Toy Story', 'Batman']

ConsistencyOfFranchises['P-Values'] = [6.940162236984522e-40,
0.11790622831256074, 1.7537323830838066e-09, 1.020118354785894e-11,
                                       1.8492328391686058e-11,
0.035792727694248905, 7.902234665149812e-06, 4.1380499020034183e-19]

consistencyValues = []
for ii in range(0, 8):
    if ConsistencyOfFranchises['P-Values'][ii] < 0.05:
        consistencyValues.append("No")
    else:
```

```
        consistencyValues.append("Yes")

ConsistencyOfFranchises['Consistent with Chance?'] = consistencyValues

# graph: ConsistencyOfFranchises (maybe in jupyter)
#%% Question 8
# 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.

# personality: (1097, 44)
# movies     : (1097, 400)
# median of each person's entire movie collection (explain why it makes
sense)
medianMovieRatingsPerPerson = movieRatings.median(axis=1)
medianRatingsPersonality = pd.concat([medianMovieRatingsPerPerson,
personality], axis=1)
medianRatingsPersonality.rename(columns={0: 'Median Movie Ratings'},
inplace=True)
medianRatingsPersonalityNoNans =
rowWiseRemovalOfNans(medianRatingsPersonality.to_numpy()) # 1097 -->
1000

outcomeMedianMovie = medianRatingsPersonalityNoNans[:, 0]
personalityMoviePCA = computePCA(medianRatingsPersonalityNoNans[:, 1:],
44, "Personality", 4)
personalityMovieGroupedFactors =
tabularizeLoadingsForKeiserPC(personalityMoviePCA, personality.columns)
personalityMovieRelevantFactors = len(personalityMovieGroupedFactors)
# X / Predictors
rotatedPersonalityMovie = -1 *
personalityPCA.fit_transform(stats.zscore(medianRatingsPersonalityNoNan
s[:, 1:]))[:, :personalityMovieRelevantFactors]
# from 44 categories to 8 factors
X1_train, X1_test, y1_train, y1_test =
train_test_split(rotatedPersonalityMovie, outcomeMedianMovie,
test_size=0.15, random_state=1234)
# multiple linear regression, explain parameters
M1 = LinearRegression().fit(X1_train, y1_train)
interceptM1, coefficientM1 = M1.intercept_, M1.coef_

# using train values on test dataset
yHatM1Test = interceptM1
for ii in range(0, len(coefficientM1)):
    yHatM1Test += (coefficientM1[ii]*X1_test[:, ii])

rmseM1 = mean_squared_error(y1_test, yHatM1Test, squared=False) #
0.5120119115856722, on average predictions are off by 0.51 units
```

```
r2M1 = r2_score(y1_test, yHatM1Test) #  0.17038962353698317 17.04% of
variance in median ratings explained by personality

# Graphhh
plt.plot(y1_test, yHatM1Test, 'o', linewidth=1, markersize=2)
plt.xlabel("Actual Movie Median")
plt.ylabel("Predicted Movie Median")
plt.title("Predicted vs. Actual Movie Median, R^2 =
{:.4f}".format(r2M1))


#%% Question 9
# 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.
predictorsOther = dataPD.iloc[:, 474:477]
moviesOther = pd.concat([medianMovieRatingsPerPerson, predictorsOther],
axis=1)
moviesOther.rename(columns={0: 'Median Movie Ratings'}, inplace=True)

# remove 3, -1, -1 respectively from each column
moviesOtherNoNans = rowWiseRemovalOfNans(moviesOther.to_numpy()) # 1097
--> 1073
deletionCriteria1 = np.where(moviesOtherNoNans[:, 1] == 3)[0]
deletionCriteria2 = np.where(moviesOtherNoNans[:, 2] == -1)[0]
deletionCriteria3 = np.where(moviesOtherNoNans[:, 3] == -1)[0]
allDeletionCriteria = np.concatenate((deletionCriteria1,
deletionCriteria2, deletionCriteria3))
allDeletionCriteria = sorted(list(allDeletionCriteria))
moviesOtherNoNans = np.delete(moviesOtherNoNans, allDeletionCriteria,
axis=0) # 1073 --> 1058
moviesOtherNoNans = moviesOtherNoNans[moviesOtherNoNans[:,
0].argsort()]


yM2 = np.empty((1058, 1))
medianThreshold = np.median(moviesOtherNoNans[:, 0])
# intially 3 but mad 3.15 for easier splitting
medianThreshold = 3.15
for index, value in enumerate(moviesOtherNoNans[:, 0]):
    yM2[index] = 0 if moviesOtherNoNans[:, 0][index] < medianThreshold
else 1


XM2 = moviesOtherNoNans[:, 1:]
#yM2 = moviesOtherNoNans[:, 0]
```

```
X2_train, X2_test, y2_train, y2_test = train_test_split(XM2, yM2,
test_size=0.15, random_state=1234)

M2 = LogisticRegression().fit(X2_train, y2_train)
interceptM2 = M2.intercept_[0]
coefficientM2 = M2.coef_.T

yHatM2Test = interceptM2
for ii in range(0, len(coefficientM2)):
    yHatM2Test += coefficientM2[ii]*X2_test[:, ii]

# rmseM2 = mean_squared_error(y2_test, yHatM2Test, squared=False) #
0.8413796654863762, on average predictions are off by 0.56 units
aucM2 = roc_auc_score(y2_test, yHatM2Test) # 0.5648148148148149 could
be better

# Plot:
plt.plot(y2_test, yHatM2Test, 'o', linewidth=1, markersize=2)
plt.xlabel("Actual Movie Median")
plt.ylabel("Predicted Movie Median")
plt.title("Predicted vs. Actual Movie Median AUC =
{:.4f}".format(aucM2))

"""M2 = LinearRegression().fit(X2_train, y2_train)
interceptM2, coefficientM2 = M2.intercept_, M2.coef_

yHatM2Test = interceptM2
for ii in range(0, len(coefficientM2)):
    yHatM2Test += (coefficientM2[ii]*X2_test[:, ii])

rmseM2 = mean_squared_error(y2_test, yHatM2Test, squared=False) #
0.5593165060851649, on average predictions are off by 0.56 units
r2M2 = r2_score(y2_test, yHatM2Test) #  -3.546456990544122e-05 not a
good modeal

plt.plot(y2_test, yHatM2Test, 'o', linewidth=1, markersize=2)
plt.xlabel("Actual Movie Median")
plt.ylabel("Predicted Movie Median")
plt.title("Predicted vs. Actual Movie Median")"""

#%% Question 10
# 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.

# get all that data
notMovieRatings = dataPD.iloc[:, 400:]
```

```
movieFromEverythingElse= pd.concat([medianMovieRatingsPerPerson,
notMovieRatings] ,axis=1)
movieFromEverythingElse.rename(columns={0: 'Median Movie Ratings'},
inplace=True)

movieFromEverythingElseNoNans =
rowWiseRemovalOfNans(movieFromEverythingElse.to_numpy()) # 1097 --> 946
everythingElse = movieFromEverythingElseNoNans[:, 1:]
lastMovie = movieFromEverythingElseNoNans[:, 0]

# PCA everythingelse
everythingElsePCA = computePCA(everythingElse, 77, "Everything Else",
10)
rotatedEverythingElse = -1 *
everythingElsePCA.fit_transform(stats.zscore(everythingElse))[:, :18]
# then put in model
X3_train, X3_test, y3_train, y3_test =
train_test_split(rotatedEverythingElse, lastMovie, test_size=0.15,
random_state=1234)
# multiple linear regression, explain parameters
M3 = LinearRegression().fit(X3_train, y3_train)
interceptM3, coefficientM3 = M3.intercept_, M3.coef_

# using train values on test dataset
yHatM3Test = interceptM3
for ii in range(0, len(coefficientM3)):
    yHatM3Test += (coefficientM3[ii]*X3_test[:, ii])

rmseM3 = mean_squared_error(y3_test, yHatM3Test, squared=False) #
0.6696055957234062, on average predictions are off by 0.67 units
r2M3 = r2_score(y3_test, yHatM3Test) #  0.042961207528030565 4.3% of
variance in median ratings explained by all factors
# given personality alone does that maybe ...

# Graphhh
plt.plot(y3_test, yHatM3Test, 'o', linewidth=1, markersize=2)
plt.xlabel("Actual Movie Median")
plt.ylabel("Predicted Movie Median")
plt.title("Predicted vs. Actual Movie Median, R^2 =
{:.4f}".format(r2M3))
```