Anita Ye
yy3557

Capstone Project

0) Data Cleaning

Before starting to answer the questions, we need to clean out the dataset, and split them into manageable parts to make it readable for us and the people who need to read our code. I imported the two datasets using dataframes, and convert them into numpy arrays when I need to use the methods in numpy. I first imported all the packages I need, and add to the group whenever I need just in case I will need to use them later as well. I then put down my N number for random.seed, especially when splitting data for train_test_split.

The next chunk of code was saved for reused data frames, such as the large rate and classical, etc. I believe that the splitting of the large dataset should always be the first step, and then we can consider the missing data. Later in the process of doing the questions, I dropped the nan in the dataset when it is meaningless to fill them with anything. It is possible to fill in with, for example, the median of the data, while for some categories, age, for example, does not make sense to take the median.

1) Is classical art more well-liked than modern art?

The question of comparing which type of art is more liked compared to the other is equivalent to determining the difference between their preference ratings, and therefore we can choose a representative feature, mean, for example, to compare, and then perform a t-test to see if the rates are different from each other.

I started by reading the CSV files and extracting the data. I split modern and classical ratings using iloc. The first thing that came up is to simply look at the mean of modern, and we can see that the mean of the mean of the classical rating is higher by around 0.5, which means that classical art is more well-liked than modern art. Next, we can conduct both a two-sided and greater t-test to compare the means, and here we can get the p-values. We can see here the p-value is lower than 0.05, which means that it is very unlikely to observe data under the null hypothesis. Therefore, we reject the null hypothesis and conclude that there is a difference.

```
print('mean of classical art: '+ str(classical.mean().mean()))
print('mean of modern art: '+ str(modern.mean().mean()))
```

```
mean of classical art: 4.722962962962963
mean of modern art: 4.210380952380954
```

```
ttest_ind(classical.mean(), modern.mean(), alternative = "two-sided")
```

```
Ttest_indResult(statistic=3.672293579314024, pvalue=0.0004702217237268629)
```

```
ttest_ind(classical.mean(), modern.mean(), alternative = "greater")
```

```
Ttest_indResult(statistic=3.672293579314024, pvalue=0.0002351108618634314)
```

2) Is there a difference in the preference ratings for modern art vs. non-human (animals and computers) generated art?

To compare the difference between the preference ratings for modern art vs. non-human generated art, we can compare the mean to get an intuitive sense, and again t-tests to evaluate the likelihood of data to happen. Compared to question 1, the mean is much farther away, as we can see the mean of modern is around 1 more than that of non-human art, which means that modern art is higher than that of non-human art. Again, in the t-test, we can see that the p-value for both the two-sided t-test and the greater t-test is very small, which means that it is very unlikely to observe the data under the null hypothesis. Therefore, we reject the null hypothesis and conclude that there is a difference in the preference ratings for modern vs. non-human arts.

```
print('mean of modern art: '+str(modern.mean().mean()))
print('mean of non-human art: '+str(nonhuman.mean().mean()))
```

```
mean of modern art: 4.210380952380954
mean of non-human art: 3.308095238095238
```

```
ttest_ind(modern.mean(), nonhuman.mean(), alternative = "two-sided")
```

```
Ttest_indResult(statistic=5.348006124028942, pvalue=1.8468344751323058e-0
6)
```

```
ttest_ind(modern.mean(), nonhuman.mean(), alternative = "greater")
```

```
Ttest_indResult(statistic=5.348006124028942, pvalue=9.234172375661529e-0
7)
```

3) Do women give higher art preference ratings than men?

We start by extracting the data of males and females, and conduct t-tests to see if they are different. Here, I conducted t-tests with and without using .mean() to observe the result with or without reducing the dimension and simplifying the dataset. Without using .mean(), the dataframe shows the statistics on the first row (0), and the p-value from the second row (1). Although the p-values diverge, we can see that they are relatively high, which means that we can not reject the null hypothesis. Using .mean(), the result is more clear that we have gotten a high p-value and we fail to reject the null hypothesis. Therefore, we have reached the conclusion that women do not give higher art preference ratings than men.

```
pd.DataFrame(ttest_ind(male,female,alternative = 'two-sided'))
```

	0	1	2	3	4	5	6	7	8	
0	-2.014529	-0.411456	-0.127988	-2.042704	2.097864	1.525626	-0.038003	2.227570	0.996111	1.7
1	0.044938	0.681062	0.898253	0.042046	0.036840	0.128265	0.969713	0.026727	0.320082	0.(

2 rows × 91 columns

```
pd.DataFrame(ttest_ind(male,female, alternative = 'greater'))
```

	0	1	2	3	4	5	6	7	8	
0	-2.014529	-0.411456	-0.127988	-2.042704	2.097864	1.525626	-0.038003	2.227570	0.996111	1.7
1	0.977531	0.659469	0.550873	0.978977	0.018420	0.064132	0.515144	0.013364	0.160041	0.(

2 rows × 91 columns

```
ttest_ind(male.mean(),female.mean(), alternative = 'two-sided')
```

```
Ttest_indResult(statistic=-0.09753731189464791, pvalue=0.922408227150896
9)
```

```
ttest_ind(male.mean(),female.mean(), alternative = 'greater')
```

```
Ttest_indResult(statistic=-0.09753731189464791, pvalue=0.538795886424551
5)
```

4) Is there a difference in the preference ratings of users with some art background (some art education) vs. none?

We first extract the data for having an art background and no art background. Again, we conduct t-tests with and without using .mean(). Although the p-values diverge, we can see that they are relatively high (>0.5), which means that we can not reject the null hypothesis. Using .mean(), the result is more clear that we have gotten a high p-value and we fail to reject the null hypothesis. Therefore, we have reached the conclusion that there is no difference in the preference rating of users with different art backgrounds.

```
pd.DataFrame(ttest_ind(art, noart, alternative='two-sided'))
```

	0	1	2	3	4	5	6	7	8	
0	-0.585859	-0.794131	-0.643341	-1.333318	-0.683641	0.308652	1.033184	-1.112333	-2.058822	-
1	0.558414	0.427751	0.520498	0.183446	0.494733	0.757802	0.302356	0.266891	0.040380	

2 rows × 91 columns

```
pd.DataFrame(ttest_ind(art, noart, alternative='greater'))
```

	0	1	2	3	4	5	6	7	8	
0	-0.585859	-0.794131	-0.643341	-1.333318	-0.683641	0.308652	1.033184	-1.112333	-2.058822	-
1	0.720793	0.786125	0.739751	0.908277	0.752633	0.378901	0.151178	0.866554	0.979810	

2 rows × 91 columns

```
ttest_ind(art.mean(), noart.mean(), alternative='two-sided')
```

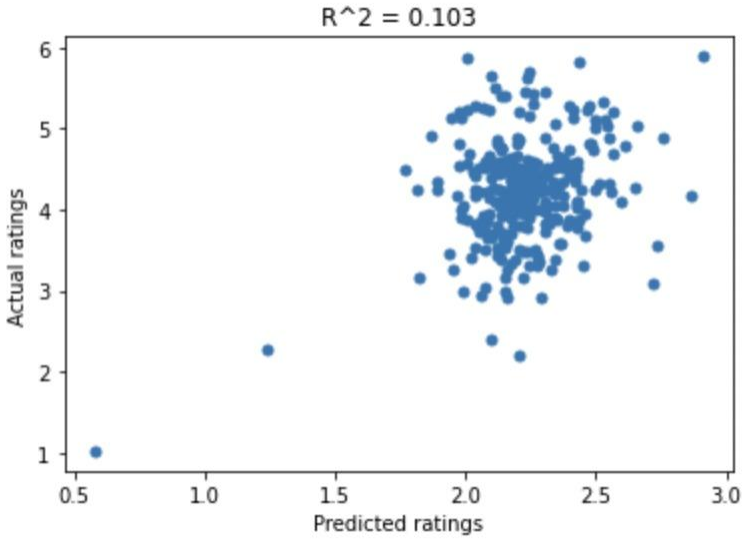
```
Ttest_indResult(statistic=-1.0063732409491997, pvalue=0.3155869939349934
5)
```

```
ttest_ind(art.mean(), noart.mean(), alternative='greater')
```

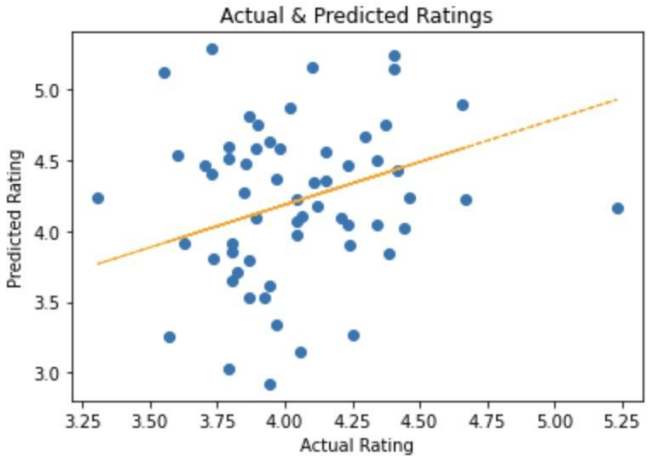
```
Ttest_indResult(statistic=-1.0063732409491997, pvalue=0.8422065030325032)
```

5) Build a regression model to predict art preference ratings from energy ratings only. Make sure to use cross-validation methods to avoid overfitting and characterize how well your model predicts art preference ratings.

I started by first extracting the data of rates and energy, and then plotting a simple scatter plot to see how the data points behave. I then use the LinearRegression to build a model and get the intercept and coefficient of the linear regression model. I used R^2 to evaluate the performance of the model. We can see here there are some outliers, but since the non-outlier points do not show very strong inclination of linear relation, it is ok to leave it here. Here's the scatter plot of the predicted and actual ratings, with a R-square of 0.103, which is fair.



To make sure, I also used train_test_split to build a simple regression model and model scores to test if my regression seemed to predict the data, and avoid cross-validation. I then plotted the test data points as a scatter plot and the linear regression line. I used the root of mean square error and r-squared to measure the error and look at the behavior of the model. We can see that the root of the mean squared error is around 0.55, which is not a great score, but still relatively acceptable looking at the graph.

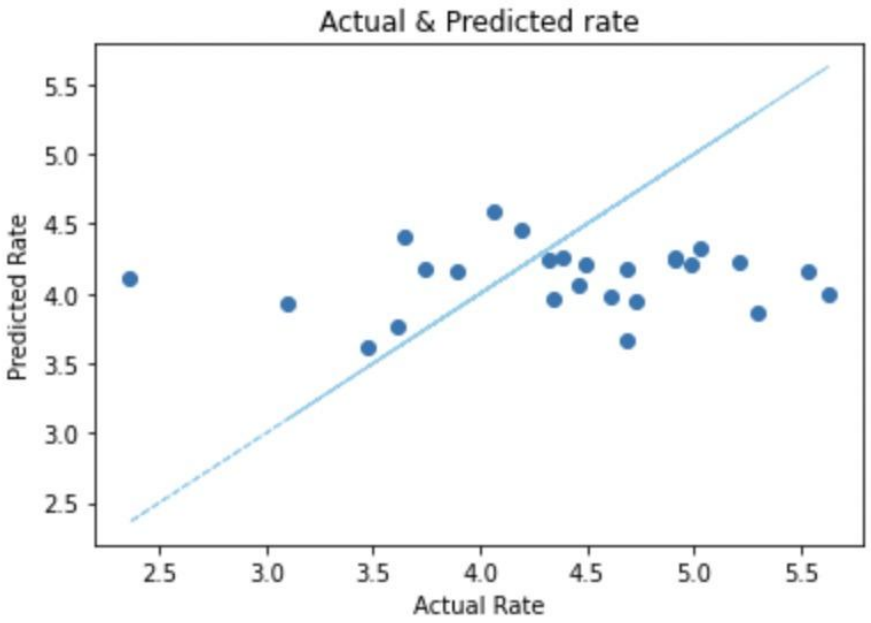



```
Root Mean Squared Error: 0.5532847578238594
R-squared -0.028111359369796363
```

6) Build a regression model to predict art preference ratings from energy ratings and demographic information. Make sure to use cross-validation methods to avoid overfitting and comment on how well your model predicts relative to the "energy ratings only" model.

I first extracted the data for demographic information. Notice that each column of demographic information represents different responses, which are not very representative and requires extra analysis. I started by using a simple linear regression built by train_split to use a cross-validation method to fit the data and get the intercept and coefficient. We can see the performance of the model by calculating the root mean square error and R^2 error. We can see that we are still getting a relatively high error, which means that the model was not behaving well in predicting the data. I also draw a matrix for demographic and energy to see their correlation. Since the dimension of x is high, we can not visually plot it on a 2D graph, so alternatively I plotted y only. This is obviously not a great model since the score is low, and this is caused by overfitting and messy data. Here is a plotted rate graph, and the intercept and coefficient I got:

```
Intercept [6.28556553]
Coefficient [[ 0.0814783  -0.09754177 -0.22277014 -0.05800916  0.07680
077  0.10353489
  -0.22117561]]
```




```
Root Mean Squared Error: 0.8143054289585965
R-squared -0.18243976940266426
```

7) Considering the 2D space of average preference ratings vs. average energy rating (that contains the 91 art pieces as elements), how many clusters can you – algorithmically - identify in this space? Make sure to comment on the identity of the clusters – do they correspond to particular types of art?

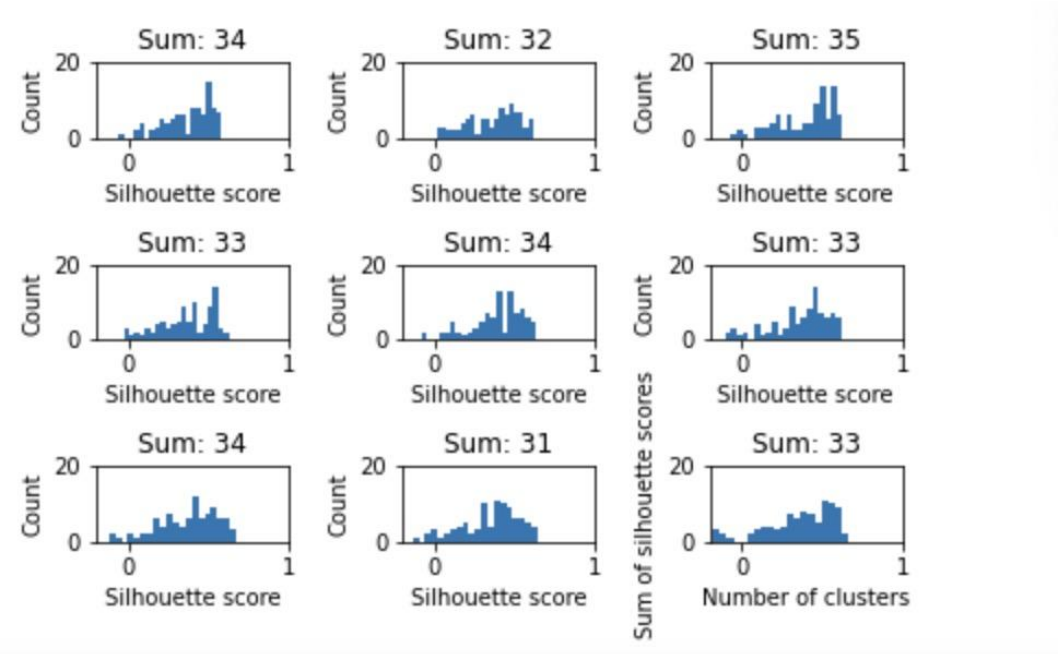
I first extracted the mean of energy and scaled them so that it is easier to fit later. I first determine the number of clusters using two ways: kneed elbow and by looping from 1 to 10, and then compute the mean silhouette and take the mean. Then we can conclude that the model is best achieved with four clusters.

Here, I used the kneed elbow method to determine the number of clusters, and get 4.

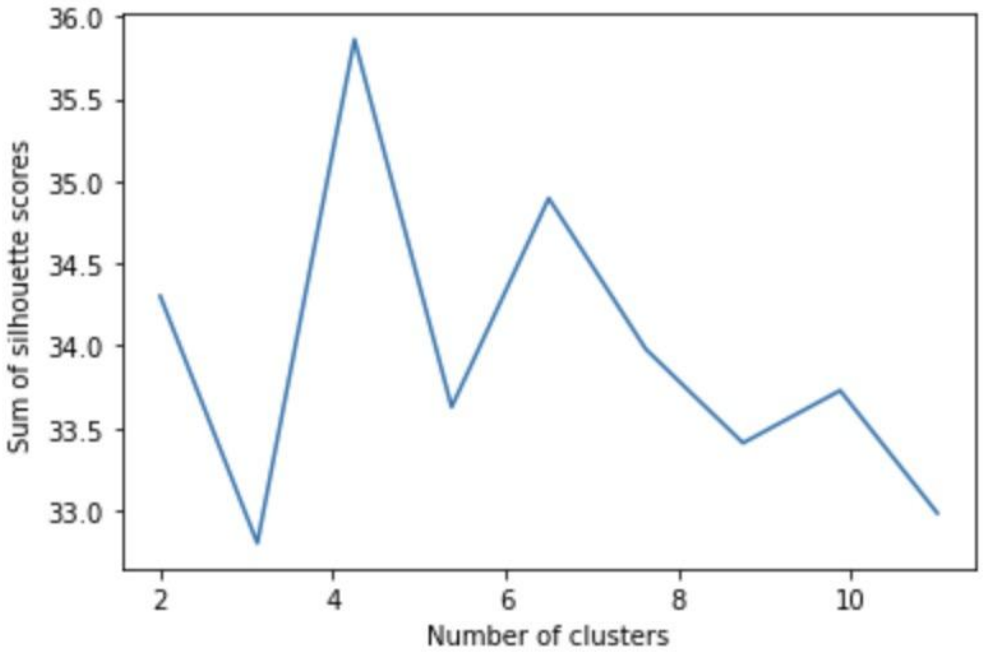
```
energy_mean1 = np.array(energy_mean).reshape(-1,1)
energy_mean = pd.DataFrame(energy_mean)
#energy_mean1 = np.array(np.mean(energy)).reshape(-1,1)
scaler = StandardScaler().fit(energy_mean1)
x_scaled = scaler.transform(energy_mean1)
df_scaled = pd.DataFrame(x_scaled, columns = energy_mean.columns)

error = list()  # to save error

for k in range(1,10):
    kmeans = KMeans(n_clusters=k) # init k-means object
    kmeans.fit(x_scaled)  # run k-means!

    error.append(kmeans.inertia_) # save sum of squared error (SSE)
kl = KneeLocator(range(1, 10), error,
                 curve='convex', direction='decreasing')

print('The number of cluster is: ' + str(kl.elbow))
```

```
The number of cluster is: 4
```

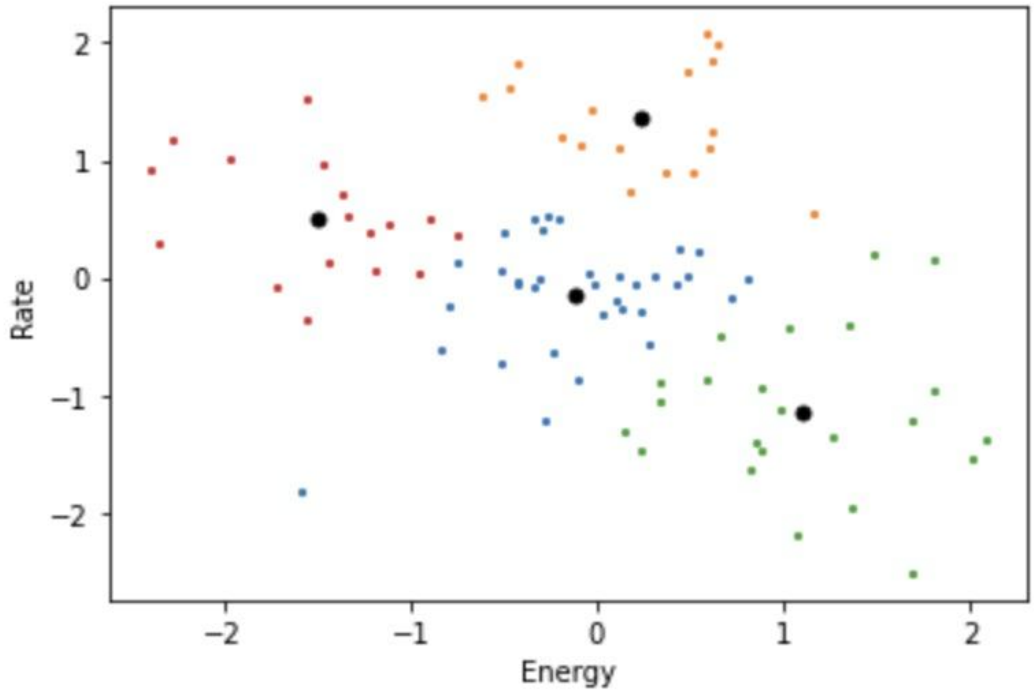
Then, I plot the sum of silhouette scores as a function of the number of clusters, in different numbers clusters, compare the sum and visually see how they behave differently.



Then, I plotted another diagram to visualize and compare the correlation more clearly. Here again, we can conclude that having four clusters would be the best choice.



Finally, I plot the cluster out using different colors to see if the plot makes sense.

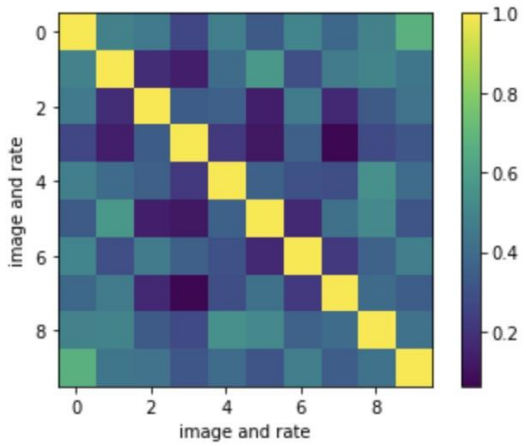


It seemed that the center of the clusters and the clusters themselves have been distributed fairly. The red cluster corresponds to modern art, the middle blue cluster is the mixture, the orange cluster is classical, and the green cluster is the non-human arts, due to their rate and energy distributions. Although without close-enough distributed data points, the cluster is relatively accurate in splitting the groups of data.

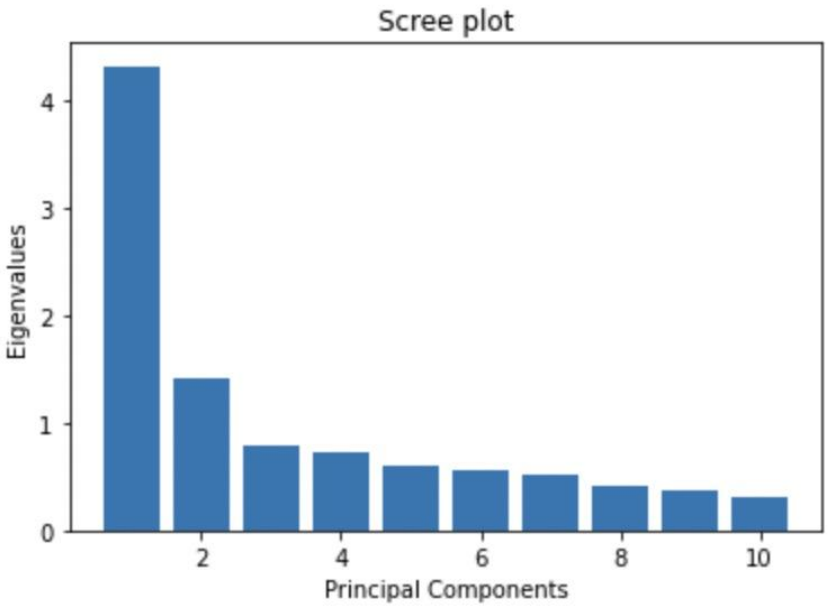
8) Considering only the first principal component of the self-image ratings as inputs to a regression model – how well can you predict art preference ratings from that factor alone?

I first cleaned the data by dropping the NaN values, since some data is missing in the image section. I chose not to fill it with median or mean because it does not make sense to do so with missing scores of self-image, as it is not reflexive enough.

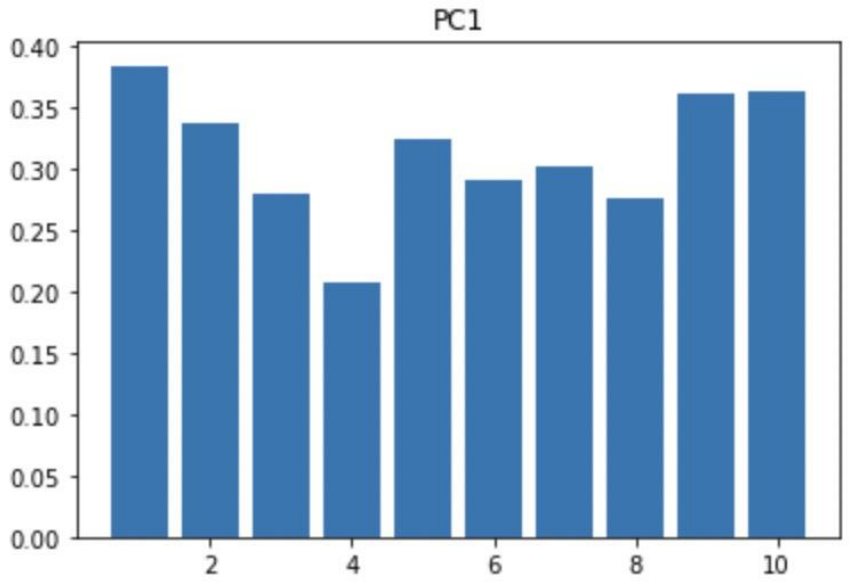
I plotted a correlation matrix for image and rate, and we can see that the correlation is relatively low still, which explains why we should conduct a PCA, as the dimension reduction helps with cleaning data to make the relations more clear.



Since we need to consider the first PCA, obviously we start by doing PCA test. I calculate the z-score, then uses the PCA and fit the z-score. Then we calculate the eigenvalues and eigenvectors (loadings), rotate the data, and transform it. We then plot the principle components:



We can see that the first factor is dominatingly meaningful, therefore choosing the first component is meaningful. I also plotted the first factor to see how it is effective in prediction:



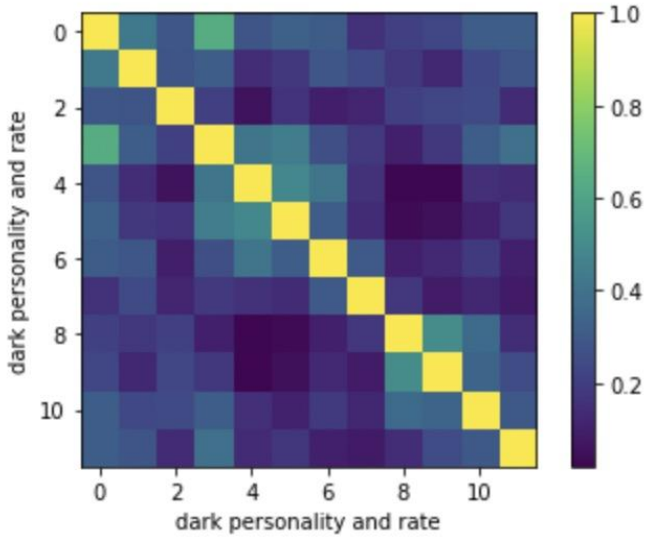
I then plug the x after PCA into a linear regression, and tested the RMSE of it. The result was not satisfying, which means that doing PCA with only the first factor does not seem to be enough. This could be because of a lack of other factors that are also influencing, and data normalization.

```
RMSE = mean_squared_error(y_train, model.predict(x_train_fin),
                          squared = False)
print('RMSE:'+str(RMSE))
```

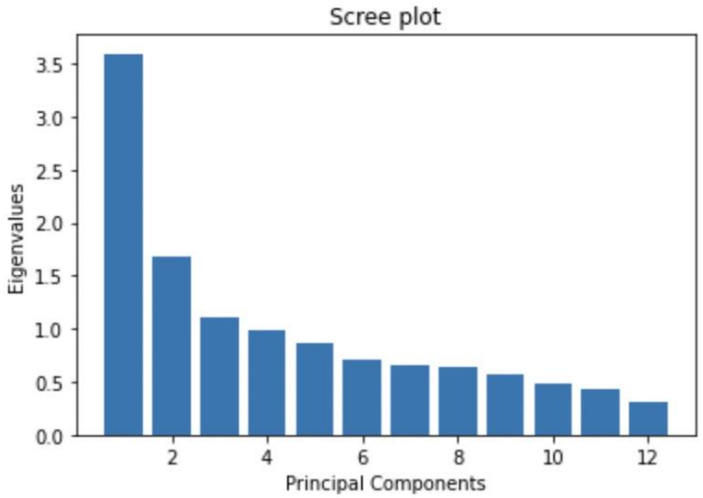
```
RMSE:0.6487406612566566
```

9) Consider the first 3 principal components of the "dark personality" traits – use these as inputs to a regression model to predict art preference ratings. Which of these components significantly predict art preference ratings? Comment on the likely identity of these factors (e.g. narcissism, manipulativeness, callousness, etc.).

I first extracted the data and plotted a correlation matrix, and we can see that they are not much correlated:

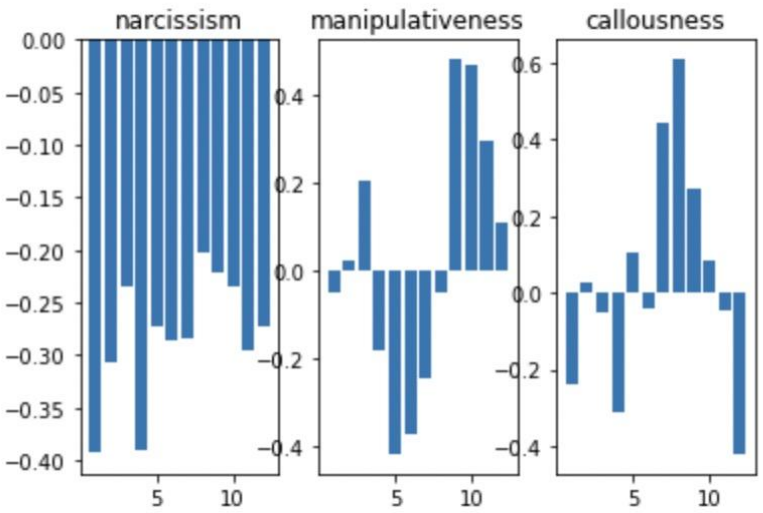


Compared to question 8, question 9 requires the first 3 instead of the first principal components. Similarly, I extracted the data from the dark personality trait, and conduct PCA.



We can see that the first factor is still dominating, yet the two after it is also fairly predictable. I then plotted the factor of all three factors to see their power. Notice that the first factor appeared to be negative in the first place, but here we can simply flip the graph by adding a negative sign to loading, which is the eigenvector. The negative value here simply means that it is negatively correlated, so the regression line has a negative slope, while the absolute value of it showed that it is effective still.

The first one is narcissism, the second is manipulativeness, and the third one is callousness. We can identify these by looking at the trends and the relationship, for example, narcissism is mainly negative, meaning that they all determine data and correlates in the same direction. For manipulativeness, we get high negative values in the middle, while for callousness we get high positive values in the middle.



From there, we can continue to perform a linear regression model to see if our PCA is working. I used the train_split and LinearRegression() model, then fit the training data in. I printed out all the intercepts, and then from RMSE we can get how accurate our model is.

```
Intercept 4.095911324257911
Coefficient [-0.02359469  0.04883483  0.06653795 -0.02195448 -0.00199663
0.05619765
 -0.02981349  0.14362701 -0.0368169  -0.09537507  0.01851114 -0.00539727]
```

```
RMSE = mean_squared_error(y_train, model.predict(x_train),
                          squared = False)
RMSE
```

```
0.5111765563999069
```

We can see from the RMSE is relatively large, yet sort of acceptable and predicts the data. PCA here seemed not much helpful in making a more accurate linear regression model.

I also built a linear regression of all three components, and printed out the RMSE and coefficient of each in order:

PC1:

```
RMSE: 0.5556023568032002
```

```
b0, b1
```

```
(4.0975555545224855, array([0.00549971]))
```

PC2:

```
RMSE: 0.5517490294974274
```

```
b0, b1
```

```
(4.094033312438195, array([0.04282172]))
```

PC3:

```
RMSE: 0.553254884171123
```

```
b0, b1
```

```
(4.10164831841852, array([0.03937008]))
```

We can see that the predicting power and errors are similar, and the difference is minor, although the scree plot seemed diverges. It is better to understand the data from other factors.

However, when testing out the data without PCA, the RMSE is much larger and unacceptable. This means that although still not satisfying enough, PCA did helped us reduce dimensions and clear out data for a better model.

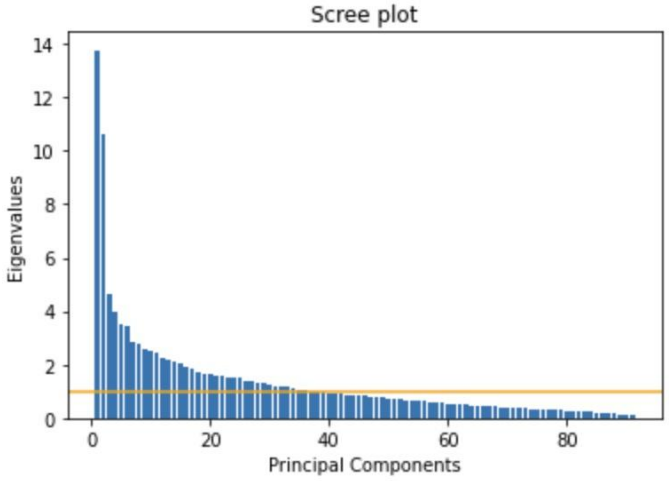
```
RMSE = mean_squared_error(y_train, model.predict(x_train),
                          squared = False)
RMSE
```

```
1.2405290935458382
```

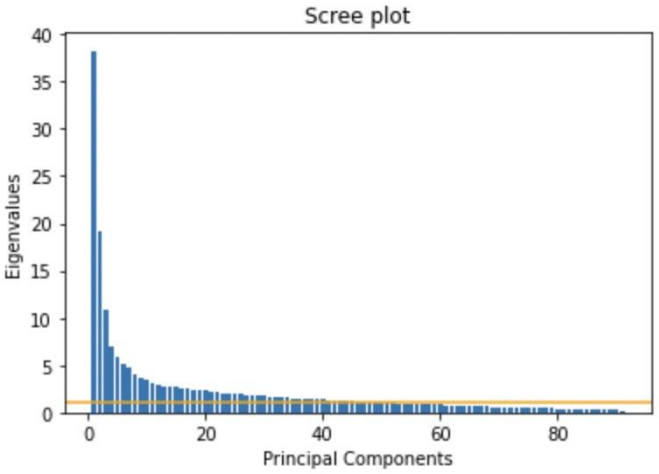
10) Can you determine the political orientation of the users (to simplify things and avoid gross class imbalance issues, you can consider just 2 classes: "left" (progressive & liberal) vs. "nonleft" (everyone else)) from all the other information available, using any classification model of your choice? Make sure to comment on the classification quality of this model.

I first extracted the data on political orientation and created another column to identify left with 1, and non-left by 0. Then I can identify each row with either left or non-left. Again, we conducted a PCA to reduce the dimensions, since this is going to be a huge test. We calculate the z-score and fit it with PCA, then we plot the scree plot. Here, we can see that here above the orange line, all components are valuable and has predicting power. However, the plot seemed to be a little extreme, since the first fews are much higher eigenvalues. We can solve this problem by normalizing and dropping some data, or modifying factors and their weights.

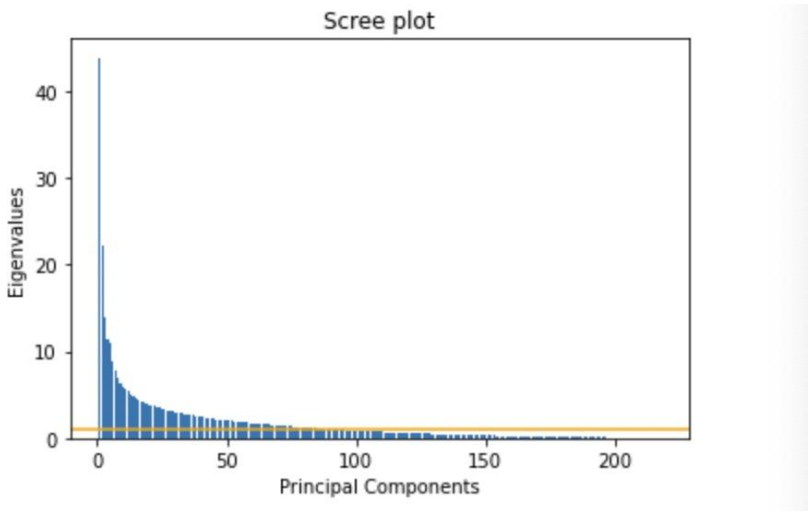
I started by extracting energy scores and seeing if it is a great predictor, and continues on to use all the other columns as predictors. Here is the plot for when energy score is the predictor for political orientation:



Here is the plot when I use the rates as the predictor:



Here is the plot when I use all columns as the predictor:



Then I split the data using train test split and used logistic regression to fit the model. Notice here linear regression would not be as effective since the amount of dimensions are adding up. I calculated the score of this model, and it is a rather high score, which means that it is relatively predictable of the political orientation.

```
model.score(x_train_fin,y_train)
```

```
0.6333333333333333
```

```
model.score(x_test_fin,y_test)
```

```
0.6222222222222222
```

Extra credit: Tell us something interesting about this dataset that is not trivial and not already part of an answer (implied or explicitly) to these enumerated questions.

- Inconsistent shape: I encounter this error a lot of times when not dealing with cleaning it properly. This is mainly because the location and numbers of missing data is different, and this made me need to be extra careful when cleaning it.
- Notice that the first principal component factor is all negative, and although it is the first one, it is not the strongest predictor, since it is all in the same direction. It is interesting as I have not seen a factor with only effects on one direction and especially with negative direction.
- It becomes hard to clean out data when we need to separate the entire dataset based on each category. Also, PCA tests are not always effective with not predictable enough data and not relatable columns of scores.

CapstoneProject

December 21, 2022

[1]:
```
import numpy as np
import scipy as sp
import pandas as pd
import statsmodels as sm
from scipy import stats
from scipy.stats import ttest_ind
import random
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from math import sqrt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn import preprocessing
from scipy.stats import zscore
from sklearn import metrics
from kneed import KneeLocator # for KMeans, elbow method
from sklearn.preprocessing import scale

from sklearn.metrics import silhouette_samples

from sklearn.linear_model import LogisticRegression
from scipy.special import expit

from sklearn.preprocessing import StandardScaler     # feature scaling
from sklearn import metrics                          # for evaluation metrics

from sklearn.neighbors import KNeighborsClassifier   # knn
from sklearn.cluster import KMeans                   # kmeans

# additional metrics
from kneed import KneeLocator # for KMeans, elbow method
#from yellowbrick.cluster import SilhouetteVisualizer # for KMeans, silhouette␣
 ↪scores
```

[2]:
```
seed = 10676823
```

[3]:
```
#1
art = pd.read_csv('theArt.csv')
data = pd.read_csv('theData.csv', header = None)
rate = data.iloc[:,0:91]
classical = rate.iloc[:,0:36]
modern = rate.iloc[:,36:71]

#2
#modern
nonhuman = rate.iloc[:,70:91]

#3
gender = pd.DataFrame(data.iloc[:,216])
rate_gender = pd.concat([rate,gender],axis=1)
male = rate_gender[rate_gender[216]==1].iloc[:,:-1]
female = rate_gender[rate_gender[216]==2].iloc[:,:-1]

#4
education = data.iloc[:,218]
rate_education = pd.concat([rate,education],axis=1)
noart = rate_education[rate_education.iloc[:,91] == 0].iloc[:,:-1]
art = rate_education[rate_education.iloc[:,91] != 0].iloc[:,:-1]

#5
energy1 = data.iloc[:,91:182]
energy = energy1.fillna(energy1.median())


#6
#It only makes sense to drop it instead of filling it with median or
#something since it does not make sense
demographic = data.iloc[:,215:221]
#notice I made rate to rate mean
rate_energy_demographic = pd.concat([np.mean(rate,axis=0),
                                     np.mean(energy, axis=1),
                                     demographic],axis=1).dropna()

#7
rate_mean = np.mean(rate, axis=1)
energy_mean = np.mean(energy,axis=1)
rate_energy_mean = pd.concat([np.mean(rate, axis=1),np.mean(energy,axis=1)],␣
 ↪axis=1)
rate_energy_mean = pd.DataFrame (rate_mean,columns =['rate'])
rare_energy_mean = pd.DataFrame(energy_mean,columns =['energy'])
```

```
#8
image = data.iloc[:,205:215].dropna()

#9
dark = data.iloc[:,182:194].dropna()

#10
political1 = data.iloc[:,217:218]
politicalnp = political1.to_numpy()
political2=[]
for i in range(300):
    if(politicalnp[i:i+1]==1.0 ):
        political2.append(1)
    elif(politicalnp[i:i+1]==2.0):
        political2.append(1)
    else:
        political2.append(0)
political = pd.DataFrame(political2)
rate_political = pd.concat([rate,political], axis=1)
#left = rate_political[rate_political.iloc[:,91:92]==1].iloc[:,:-2]
#temp=rate_political[rate_political.iloc[:,91:92]==1]
```

[4]:
```
# 1) Is classical art more well liked than modern art?
```

[5]:
```
print('mean of classical art: '+ str(classical.mean().mean()))
print('mean of modern art: '+ str(modern.mean().mean()))
```

```
mean of classical art: 4.722962962962963
mean of modern art: 4.210380952380954
```

[6]:
```
ttest_ind(classical.mean(), modern.mean(), alternative = "two-sided")
```

[6]:
```
Ttest_indResult(statistic=3.672293579314024, pvalue=0.0004702217237268629)
```

[7]:
```
ttest_ind(classical.mean(), modern.mean(), alternative = "greater")
```

[7]:
```
Ttest_indResult(statistic=3.672293579314024, pvalue=0.00023511086186343144)
```

[8]:
```
# 2) Is there a difference in the preference ratings for modern art vs.␣
 ↪non-human (animals and computers) generated art?
```

[9]:
```
print('mean of modern art: '+str(modern.mean().mean()))
print('mean of non-human art: '+str(nonhuman.mean().mean()))
```

```
mean of modern art: 4.210380952380954
mean of non-human art: 3.308095238095238
```

[10]:
```
ttest_ind(modern.mean(), nonhuman.mean(), alternative = "two-sided")
```

[10]:
```
Ttest_indResult(statistic=5.348006124028942, pvalue=1.8468344751323058e-06)
```

[11]:
```
ttest_ind(modern.mean(), nonhuman.mean(), alternative = "greater")
```

[11]:
```
Ttest_indResult(statistic=5.348006124028942, pvalue=9.234172375661529e-07)
```

[12]:
```
# 3) Do women give higher art preference ratings than men?
```

[13]:
```
pd.DataFrame(ttest_ind(male,female,alternative = 'two-sided'))
```

[13]:

	0	1	2	3	4	5	6	7	8	9	…	81	82	83	84	85	86	87	88	89	90
0	-2.014529	-0.411456	-0.127988	-2.042704	2.097864	1.525626	-0.038003	2.227570	0.996111	1.773157	…	1.574397	1.832576	1.057185	0.109336	0.592502	1.563676	0.939356	-2.288329	-0.461777	-0.099121
1	0.044938	0.681062	0.898253	0.042046	0.036840	0.128265	0.969713	0.026727	0.320082	0.077322	…	0.116558	0.067959	0.291365	0.913017	0.554007	0.119056	0.348382	0.022886	0.644610	0.921115

[2 rows x 91 columns]

[14]:
```
pd.DataFrame(ttest_ind(male,female, alternative = 'greater'))
```

[14]:

	0	1	2	3	4	5	6	7	8	9	…	81	82	83	84	85	86	87	88	89	90
0	-2.014529	-0.411456	-0.127988	-2.042704	2.097864	1.525626	-0.038003	2.227570	0.996111	1.773157	…	1.574397	1.832576	1.057185	0.109336	0.592502	1.563676	0.939356	-2.288329	-0.461777	-0.099121
1	0.977531	0.659469	0.550873	0.978977	0.018420	0.064132	0.515144	0.013364	0.160041	0.038661	…	0.058279	0.033979	0.145683	0.456508	0.277003	0.059528	0.174191	0.988557	0.677695	0.539442

[2 rows x 91 columns]

[15]:
```
ttest_ind(male.mean(),female.mean(), alternative = 'two-sided')
```

[15]:
```
Ttest_indResult(statistic=-0.09753731189464791, pvalue=0.9224082271508969)
```

[16]:
```
ttest_ind(male.mean(),female.mean(), alternative = 'greater')
```

[16]:
```
Ttest_indResult(statistic=-0.09753731189464791, pvalue=0.5387958864245515)
```

[17]:
```
# 4) Is there a difference in the preference ratings of users with some art␣
↪background (some art education) vs. none?
```

[18]:
```
pd.DataFrame(ttest_ind(art, noart, alternative='two-sided'))
```

[18]:

	0	1	2	3	4	5	6	\
0	-0.585859	-0.794131	-0.643341	-1.333318	-0.683641	0.308652	1.033184	
1	0.558414	0.427751	0.520498	0.183446	0.494733	0.757802	0.302356	

	7	8	9	…	81	82	83	84	\
0	-1.112333	-2.058822	-1.270792	…	-1.471701	-1.560366	-1.461679	-0.808435	
1	0.266891	0.040380	0.204794	…	0.142157	0.119735	0.144883	0.419485	

	85	86	87	88	89	90
0	-0.969079	-2.117416	-1.401603	-1.506832	-2.245935	-2.135719
1	0.333291	0.035053	0.162075	0.132913	0.025441	0.033517

[2 rows x 91 columns]

[19]:
```
pd.DataFrame(ttest_ind(art, noart, alternative='greater'))
```

[19]:

	0	1	2	3	4	5	6	\
0	-0.585859	-0.794131	-0.643341	-1.333318	-0.683641	0.308652	1.033184	
1	0.720793	0.786125	0.739751	0.908277	0.752633	0.378901	0.151178	

	7	8	9	…	81	82	83	84	\
0	-1.112333	-2.058822	-1.270792	…	-1.471701	-1.560366	-1.461679	-0.808435	
1	0.866554	0.979810	0.897603	…	0.928922	0.940133	0.927559	0.790257	

	85	86	87	88	89	90
0	-0.969079	-2.117416	-1.401603	-1.506832	-2.245935	-2.135719
1	0.833354	0.982474	0.918963	0.933543	0.987280	0.983241

[2 rows x 91 columns]

[20]:
```
ttest_ind(art.mean(), noart.mean(), alternative='two-sided')
```

[20]:
```
Ttest_indResult(statistic=-1.0063732409491997, pvalue=0.31558699393499345)
```

[21]:
```
ttest_ind(art.mean(), noart.mean(), alternative='greater')
```

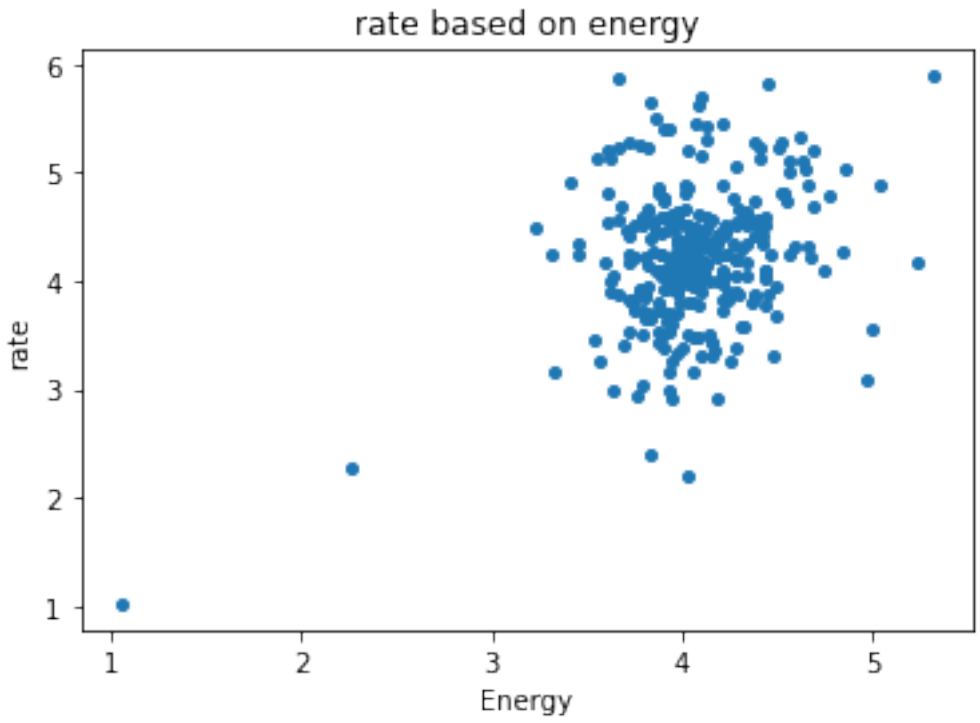
[21]:
```
Ttest_indResult(statistic=-1.0063732409491997, pvalue=0.8422065030325032)
```

[22]:
```

```

```
# 5) Build a regression model to predict art preference ratings from energy␣
↪ratings only. Make sure to use cross-validation methods to avoid overfitting␣
↪and characterize how well your model predicts art preference ratings.
```

[23]:
```
y = np.mean(rate, axis=1)
x = np.mean(energy, axis=1)
plt.plot(x,y,'o',markersize=4)

plt.xlabel('Energy')
plt.ylabel('rate')
plt.title('rate based on energy')
#plt.legend()
```

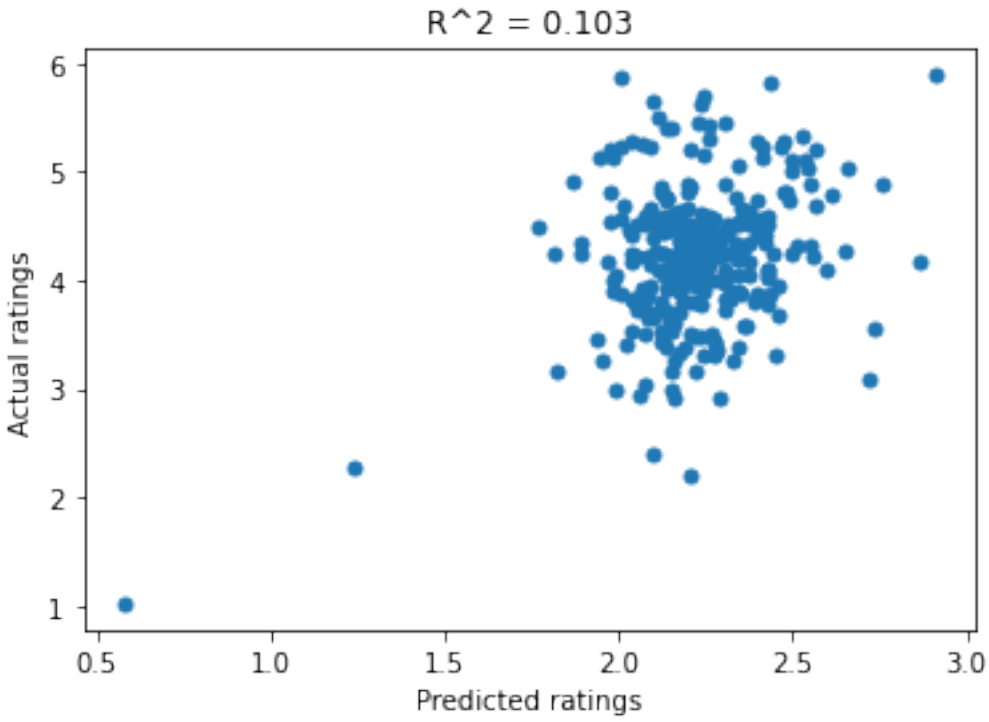
[23]: Text(0.5, 1.0, 'rate based on energy')



[24]:
```
x = np.array(x).reshape(-1,1)
model = LinearRegression().fit(x,y)
b0, b1 = model.intercept_, model.coef_
b0,b1
```

[24]: (1.9991485860138005, array([0.54749206]))

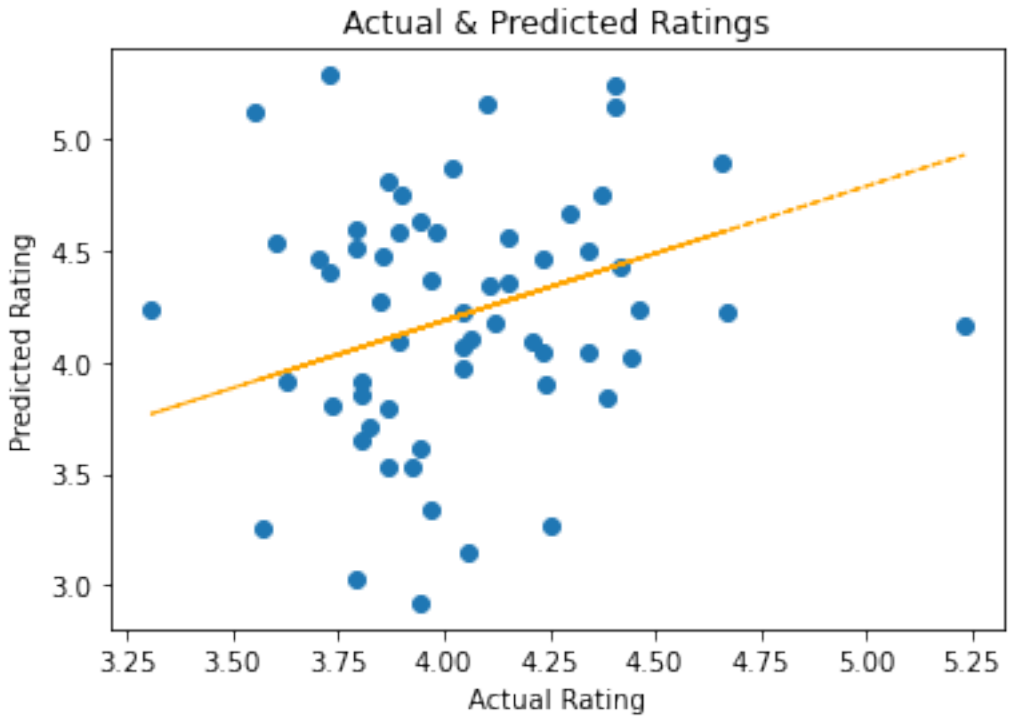
[25]:
```
yHat = b1[0]*x[:,0]
plt.plot(yHat,y,'o',markersize=5)
plt.xlabel('Predicted ratings')
plt.ylabel('Actual ratings')
rSqr = model.score(x,y)
plt.title('R^2 = {:.3f}'.format(rSqr))
plt.show()
```



[26]:
```
x_train, x_test, y_train, y_test = train_test_split(x,                  # X matrix
                                                    y,                  # Y vector
                                                    test_size=0.20,   # percent␣
 ↪of data in test
                                                    random_state=seed  # set␣
 ↪random state
                                                   )

regressor = LinearRegression().fit(x_train, y_train)

print('Intercept', regressor.intercept_)        # Print the intercept
print('Coefficient', regressor.coef_)           # Print the coefficient
```

```
Intercept 1.776355345678529
Coefficient [0.60233973]
```

[27]:
```
y_pred = regressor.predict(x_test)

#plt.scatter(y_test, y_pred)
#plt.plot(y_test, y_test, color = 'skyblue', linestyle='--', linewidth=1)

plt.scatter(x_test, y_test)
plt.plot(x_test, y_pred, color = 'orange', linestyle='--', linewidth=1)


plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')

plt.title('Actual & Predicted Ratings')
#plt.legend()
plt.show()

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_pred)))
print('R-squared', metrics.r2_score(y_test, y_pred))
```




```
Root Mean Squared Error: 0.5532847578238594
R-squared -0.028111359369796363
```

[28]:
```
# 6) Build a regression model to predict art preference ratings from energy␣
↪ratings and demographic information. Make sure to use cross-validation␣
↪methods to avoid overfitting and comment on how well your model predicts␣
↪relative to the "energy ratings only" model.
```

[29]:
```
x = rate_energy_demographic.iloc[:,1:].values
y = rate_energy_demographic.iloc[:,:1].values
model = LinearRegression().fit(x,y)
b0, b1 = model.intercept_, model.coef_
b0, b1
```

[29]:
```
(array([5.53728266]),
 array([[ 0.17810425, -0.09002415, -0.09178336, -0.04811588,  0.01579167,
          0.12976079, -0.16778859]]))
```

[30]:
```
#x = np.array(x).reshape(-1,1)
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.3,
                                                    random_state=seed)

regressor = LinearRegression()
regressor.fit(x_train, y_train)

print('Intercept', regressor.intercept_)
print('Coefficient', regressor.coef_)
```

```
Intercept [6.28556553]
Coefficient [[ 0.0814783  -0.09754177 -0.22277014 -0.05800916  0.07680077
0.10353489
  -0.22117561]]
```

[31]:
```
y_pred = regressor.predict(x_test)
model.score(x_test,y_pred)
```

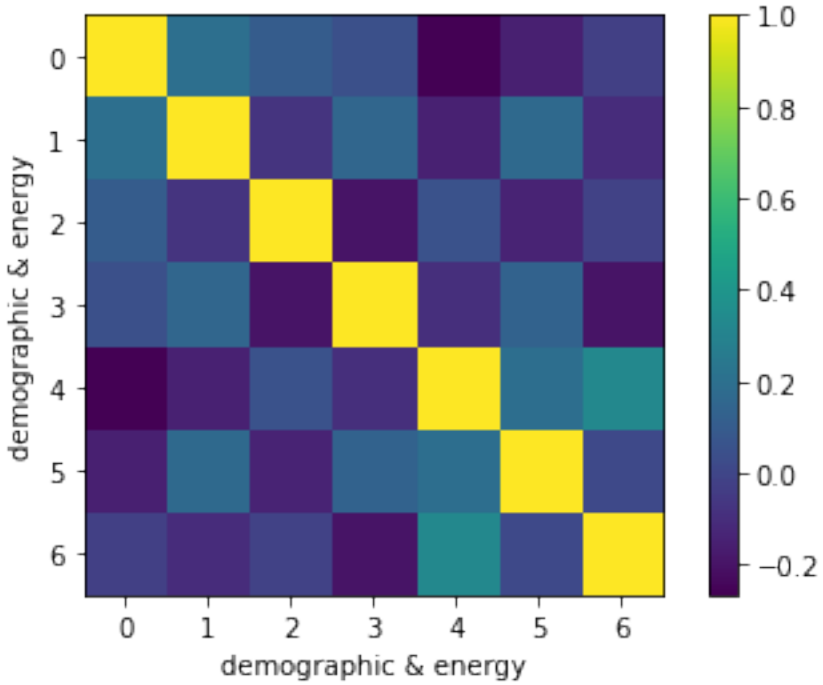
[31]:
```
0.5668405482979331
```

[32]:
```
model.score(x_test,y_test)
```

[32]:
```
-0.0036424327341577634
```

[33]:
```
corrMatrix1 = np.corrcoef(x, rowvar = False)
# column-wise is appropirate to know how similar painting were answered.
plt.imshow(corrMatrix1)
plt.xlabel('demographic & energy')
plt.ylabel('demographic & energy')
plt.colorbar()
```

```
plt.show()
```



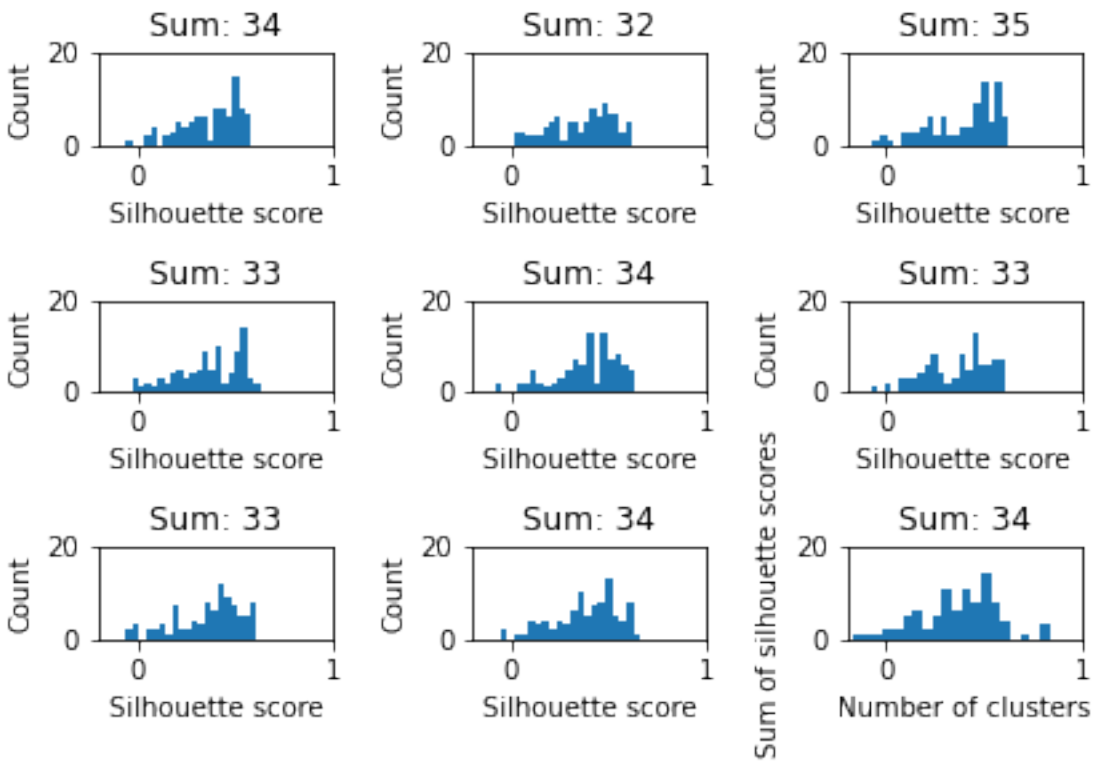
[34]:
```
# 7) Considering the 2D space of average preference ratings vs. average energy␣
↪rating (that contains the 91 art pieces as elements), how many clusters can␣
↪you - algorithmically - identify in this space? Make sure to comment on the␣
↪identity of the clusters - do they correspond to particular types of art?
```

[35]:
```
energy_mean1 = np.array(energy_mean).reshape(-1,1)
energy_mean = pd.DataFrame(energy_mean)
#energy_mean1 = np.array(np.mean(energy)).reshape(-1,1)
scaler = StandardScaler().fit(energy_mean1)
x_scaled = scaler.transform(energy_mean1)
df_scaled = pd.DataFrame(x_scaled, columns = energy_mean.columns)

error = list()  # to save error

for k in range(1,10):
    kmeans = KMeans(n_clusters=k) # init k-means object
    kmeans.fit(x_scaled)  # run k-means!

    error.append(kmeans.inertia_) # save sum of squared error (SSE)
kl = KneeLocator(range(1, 10), error,
                 curve='convex', direction='decreasing')
```

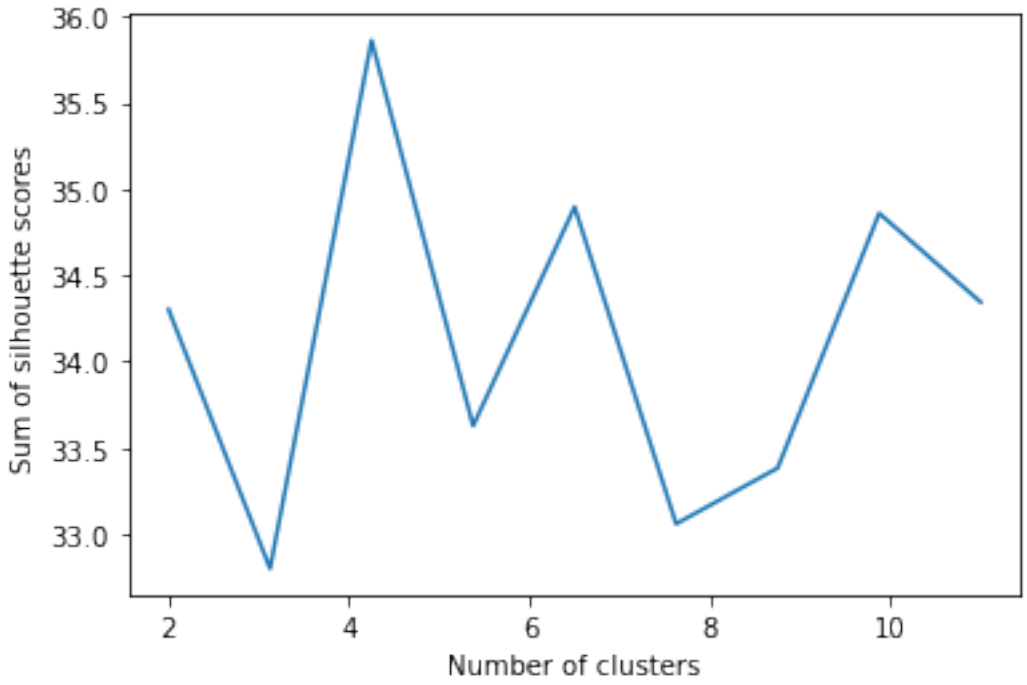
```
print('The number of cluster is: ' + str(kl.elbow))
```

```
The number of cluster is: 4
```

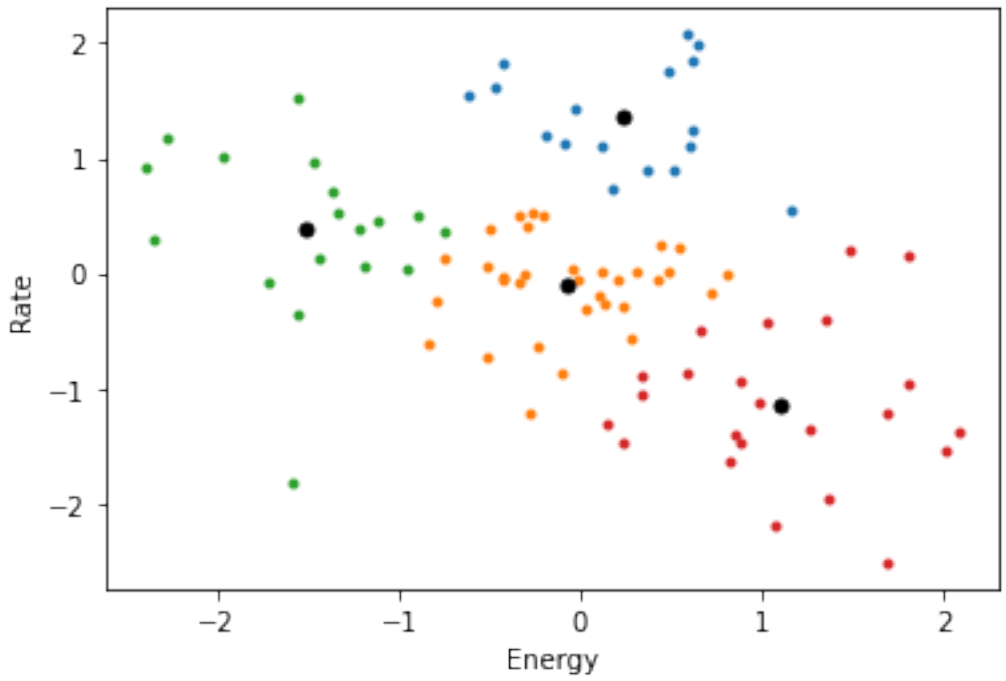
[36]:
```
#x=energy_mean.to_numpy().reshape(-1,1)
energy_mean = np.mean(energy, axis=0)
rate_mean = np.mean(rate, axis=0)
x = np.column_stack((scale(rate_mean),scale(energy_mean)))
numClusters = 9 # how many clusters are we looping over? (from 2 to 10)
sSum = np.empty([numClusters,1])*np.NaN # init container to store sums

# Compute kMeans for each k:
for ii in range(2, numClusters+2): # Loop through each cluster (from 2 to 10)
    kMeans = KMeans(n_clusters = int(ii)).fit(x) # 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(x,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(0,20)
    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

# Plot the sum of the silhouette scores as a function of the number of␣
 ↪clusters, to make it clearer what is going on
plt.plot(np.linspace(2,numClusters,9),sSum)
plt.xlabel('Number of clusters')
plt.ylabel('Sum of silhouette scores')
plt.show()
```



[37]:
```
#
x=x.reshape(-1,1)
numClusters = 9
Q = np.empty([numClusters,1])*np.NaN

# Compute kMeans:
for ii in range(2, 11):
    kMeans = KMeans(n_clusters = int(ii)).fit(x)
    cId = kMeans.labels_
    cCoords = kMeans.cluster_centers_
    s = silhouette_samples(x,cId)
    Q[ii-2] = sum(s)

plt.plot(np.linspace(2,11,9),sSum)
plt.xlabel('Number of clusters')
plt.ylabel('Sum of silhouette scores')
plt.show()
```



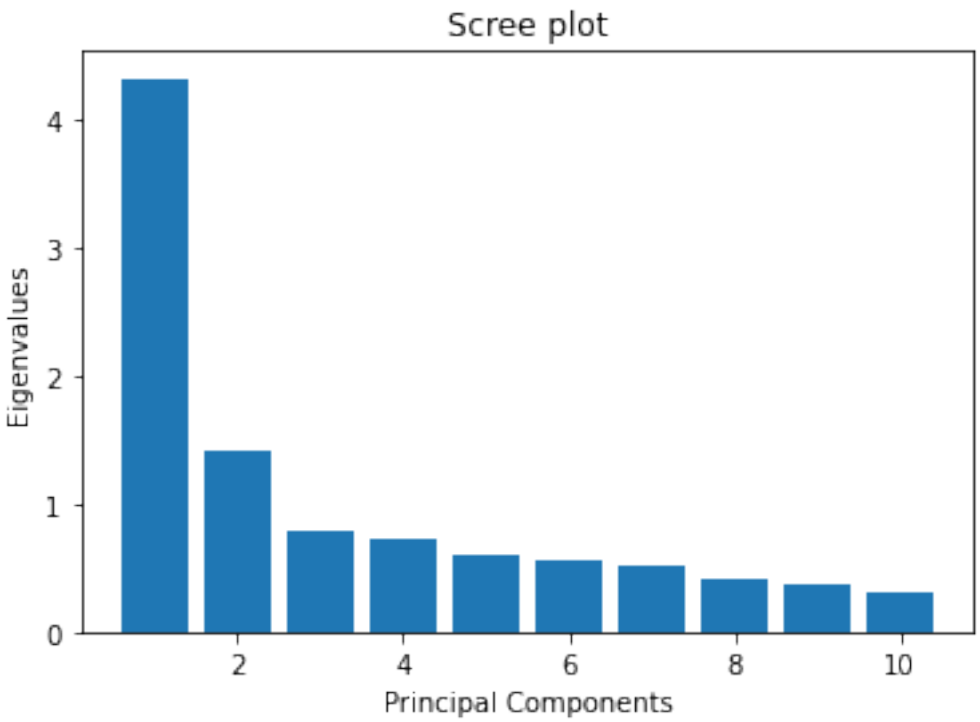
[38]:
```
# kMeans:
ls = []
x = np.column_stack((scale(np.mean(rate, axis=0)),scale(np.mean(energy,␣
 ↪axis=0))))
numClusters = 4
kMeans = KMeans(n_clusters = numClusters).fit(x)
cId = kMeans.labels_
cCoords = kMeans.cluster_centers_

# Plot the color-coded data:
for ii in range(numClusters):
    plotIndex = np.argwhere(cId == int(ii))
    ls.append(x[plotIndex,:])
    plt.plot(x[plotIndex,0],x[plotIndex,1],'o',markersize=3)
    plt.
 ↪plot(cCoords[int(ii-1),0],cCoords[int(ii-1),1],'o',markersize=5,color='black')
    plt.xlabel('Energy')
    plt.ylabel('Rate')
    #plt.legend()
```



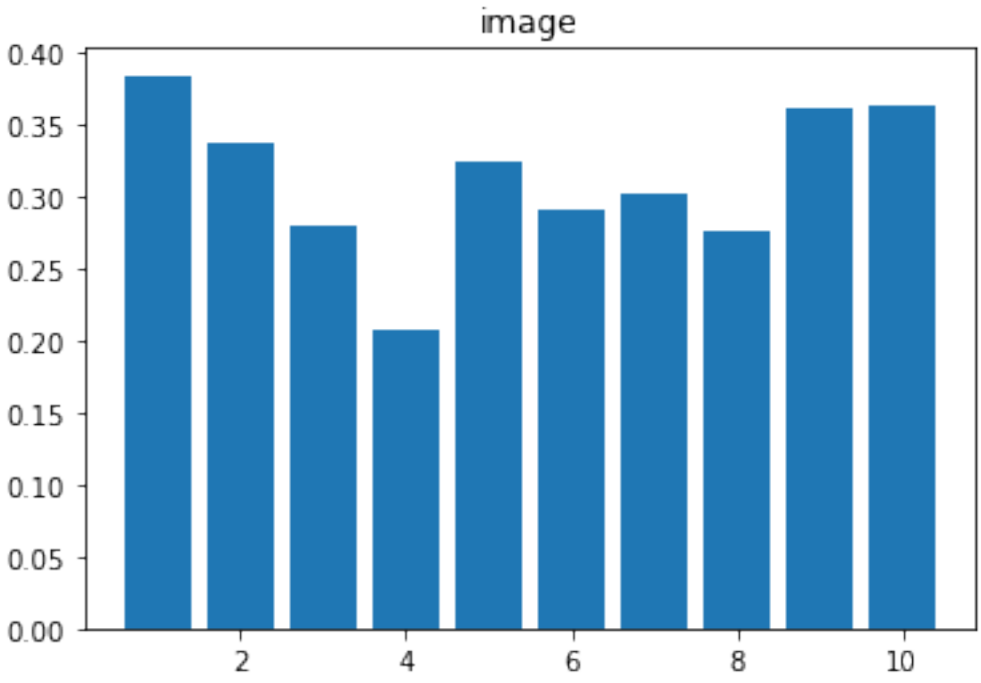
[39]:
```
# 8) Considering only the first principal component of the self-image ratings␣
↪as inputs to a regression model - how well can you predict art preference␣
↪ratings from that factor alone?
```

[40]:
```
rateimage = rate.join(image)
rateimage = rateimage.dropna()
rateimage = rateimage.iloc[:,0:91]
rateimage = rateimage.to_numpy()
```

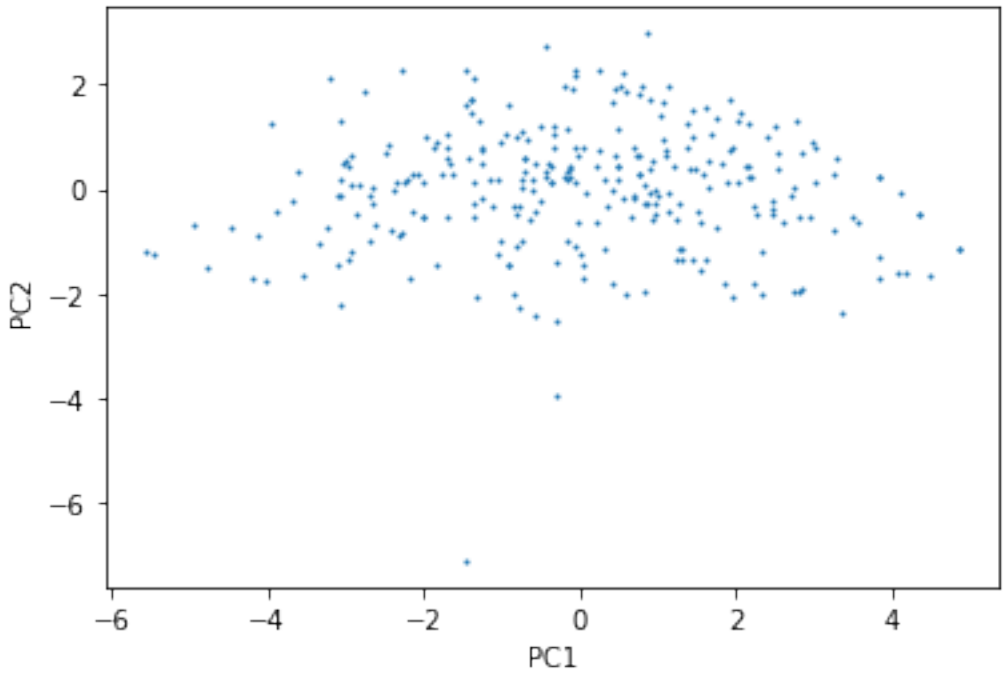
[41]:
```
zscoredData = stats.zscore(image)
pca = PCA().fit(zscoredData)
eigVals = pca.explained_variance_
loadings = pca.components_*-1
origDataNewCoordinates = pca.fit_transform(zscoredData)*-1

numPredictors = 10
plt.bar(np.linspace(1,numPredictors,numPredictors),eigVals)
plt.title('Scree plot')
plt.xlabel('Principal Components')
plt.ylabel('Eigenvalues')
plt.show()
```



[42]:
```
plt.bar(np.linspace(1,numPredictors,numPredictors),loadings[0,:])
plt.title('image')
plt.show()
```



[43]:
```
plt.plot(origDataNewCoordinates[:,0],origDataNewCoordinates[:
  ↪,1],'o',markersize=1)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

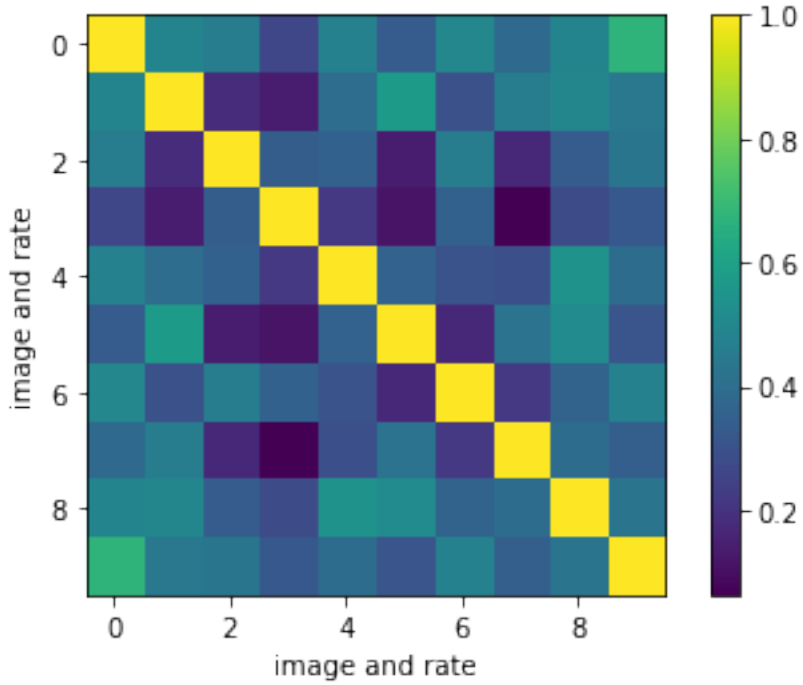


[44]:
```
x = pca.fit_transform(rateimage)*-1
y=np.mean(rateimage,axis=1)

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
  ↪3,random_state=seed)

scaler_x_train = StandardScaler().fit(x_train)
scaler_x_test = StandardScaler().fit(x_test)

x_train_trans = scaler_x_train.transform(x_train)
x_test_trans = scaler_x_test.transform(x_test)

x_train_fit = pca.fit_transform(x_train_trans)*-1
x_test_fit = pca.transform(x_test_trans)*-1
```

[45]:
```
x_train_inv = scaler_x_train.inverse_transform(x_train_fit)
x_train_fin = x_train_inv[:,:1]

x_test_inv = scaler_x_train.inverse_transform(x_test_fit)
x_test_fin = x_test_inv[:,:1]

model = LinearRegression().fit(x_train_fin, y_train)

RMSE = mean_squared_error(y_train, model.predict(x_train_fin),
                          squared = False)
print('RMSE:'+str(RMSE))
```

```
RMSE:0.6487406612566566
```

[46]:
```
corrMatrix1 = np.corrcoef(image, rowvar = False)
# column-wise is appropirate to know how similar painting were answered.
plt.imshow(corrMatrix1)
plt.xlabel('image and rate')
plt.ylabel('image and rate')
plt.colorbar()

plt.show()
```



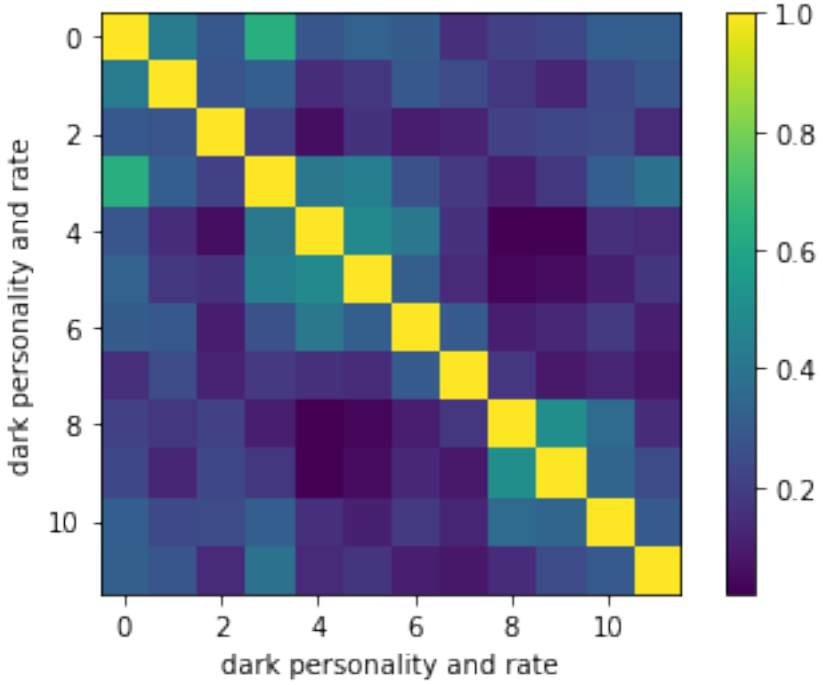
[47]:
```
model.score(x_test_fin,y_test)
```

[47]:
```
-0.05925153114333681
```

[48]:
```
# 9) Consider the first 3 principal components of the "dark personality" traits␣
↪- use these as inputs to a regression model to predict art preference␣
↪ratings. Which of these components significantly predict art preference␣
↪ratings? Comment on the likely identity of these factors (e.g. narcissism,␣
↪manipulativeness, callousness, etc.).
```

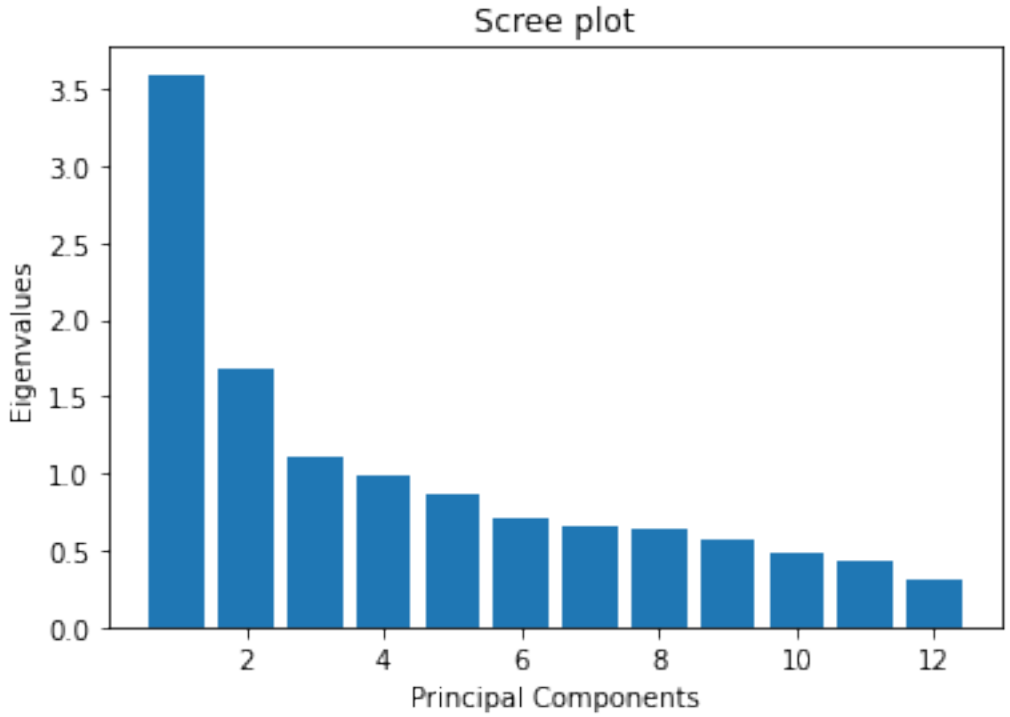
[49]:
```
dark = data.iloc[:,182:194].dropna()
```

[50]:
```
corrMatrix1 = np.corrcoef(dark, rowvar = False)
# column-wise is appropirate to know how similar painting were answered.
plt.imshow(corrMatrix1)
plt.xlabel('dark personality and rate')
plt.ylabel('dark personality and rate')
plt.colorbar()

plt.show()
```

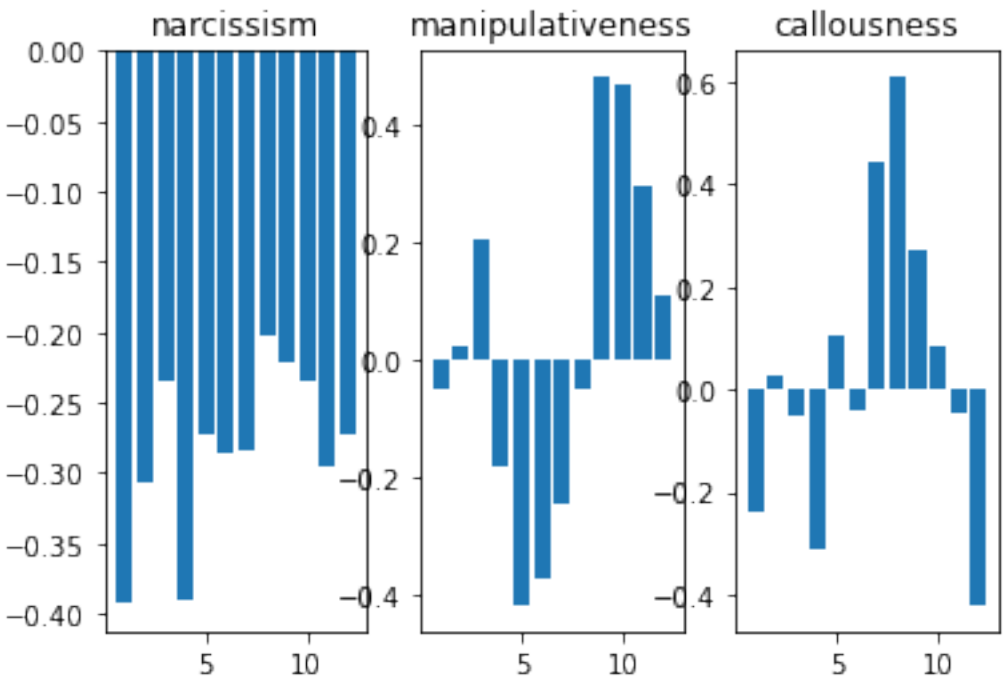


[51]:
```
zscoredData = stats.zscore(dark)
pca = PCA().fit(zscoredData)
eigVals = pca.explained_variance_
loadings = pca.components_*-1
origDataNewCoordinates = pca.fit_transform(zscoredData)*-1
```

```
numPredictors = 12
plt.bar(np.linspace(1,numPredictors,numPredictors),eigVals)
plt.title('Scree plot')
plt.xlabel('Principal Components')
plt.ylabel('Eigenvalues')
plt.show()
```



[52]:
```
plt.subplot(1,3,1) # Factor 1:
plt.bar(np.linspace(1,numPredictors,numPredictors),loadings[0,:]) # "Challenges"
plt.title('narcissism')
plt.subplot(1,3,2) # Factor 2:
plt.bar(np.linspace(1,numPredictors,numPredictors),loadings[1,:]) # "Support"
plt.title('manipulativeness')
plt.subplot(1,3,3)
plt.bar(np.linspace(1,numPredictors,numPredictors),loadings[2,:])
plt.title('callousness')
plt.show()
```



[53]:
```
x = pca.fit_transform(dark)*-1
rate_dark = pd.concat([rate,dark],axis=1).dropna()
y=np.mean(rate_dark,axis=1)
```

[54]:
```
x_train, x_test, y_train, y_test = train_test_split(x,                  # X matrix
                                                    y,                  # Y vector
                                                    test_size=0.20,   # percent␣
 ↪of data in test
                                                    random_state=seed   # set␣
 ↪random state
                                                   )

regressor = LinearRegression().fit(x_train, y_train)

print('Intercept', regressor.intercept_)        # Print the intercept
print('Coefficient', regressor.coef_)           # Print the coefficient
```

```
Intercept 4.095911324257911
Coefficient [-0.02359469  0.04883483  0.06653795 -0.02195448 -0.00199663
0.05619765
 -0.02981349  0.14362701 -0.0368169  -0.09537507  0.01851114 -0.00539727]
```

[55]:
```
#x = np.array(x).reshape(-1,1)
model = LinearRegression().fit(x,y)
```

```
b0, b1 = model.intercept_, model.coef_
b0, b1
```

[55]:
```
(4.076917817585122,
 array([-0.0273342 ,  0.052767  ,  0.07162473, -0.00579222,  0.02043085,
         0.00227953, -0.03697892,  0.14005256, -0.05197097, -0.05967979,
         0.03862776, -0.02621807]))
```

[56]:
```
RMSE = mean_squared_error(y_train, model.predict(x_train),
                          squared = False)
RMSE
```

[56]:
```
0.5111765563999069
```

[57]:
```
#PC1
x_train,x_test,y_train,y_test = train_test_split(x,
                                                 y,
                                                 test_size=0.3,
                                                 random_state=seed)

scaler_x_train = StandardScaler().fit(x_train)
scaler_x_test = StandardScaler().fit(x_test)

x_train_trans = scaler_x_train.transform(x_train)
x_test_trans = scaler_x_test.transform(x_test)

x_train_fit = pca.fit_transform(x_train_trans)*-1
x_test_fit = pca.transform(x_test_trans)*-1

x_train_inv = scaler_x_train.inverse_transform(x_train_fit)
x_train_fin = x_train_inv[:,0:1]

x_test_inv = scaler_x_train.inverse_transform(x_test_fit)
x_test_fin = x_test_inv[:,0:1]

model = LinearRegression().fit(x_train_fin, y_train)
b0, b1 = model.intercept_, model.coef_
RMSE = mean_squared_error(y_train, model.predict(x_train_fin),
                          squared = False)
print('RMSE: '+str(RMSE))
```

```
RMSE: 0.5556023568032002
```

[58]:
```
b0, b1
```

[58]:
```
(4.0975555545224855, array([0.00549971]))
```

[59]:
```
#PC2
x_train,x_test,y_train,y_test = train_test_split(x,
                                                 y,
                                                 test_size=0.3,
                                                 random_state=seed)

scaler_x_train = StandardScaler().fit(x_train)
scaler_x_test = StandardScaler().fit(x_test)

x_train_trans = scaler_x_train.transform(x_train)
x_test_trans = scaler_x_test.transform(x_test)

x_train_fit = pca.fit_transform(x_train_trans)*-1
x_test_fit = pca.transform(x_test_trans)*-1

x_train_inv = scaler_x_train.inverse_transform(x_train_fit)
x_train_fin = x_train_inv[:,1:2]

x_test_inv = scaler_x_train.inverse_transform(x_test_fit)
x_test_fin = x_test_inv[:,1:2]

model = LinearRegression().fit(x_train_fin, y_train)
b0, b1 = model.intercept_, model.coef_
RMSE = mean_squared_error(y_train, model.predict(x_train_fin),
                          squared = False)
print('RMSE: '+str(RMSE))
```

```
RMSE: 0.5517490294974274
```

[60]:
```
b0, b1
```

[60]:
```
(4.094033312438195, array([0.04282172]))
```

[61]:
```
#PC3
x_train,x_test,y_train,y_test = train_test_split(x,
                                                 y,
                                                 test_size=0.3,
                                                 random_state=seed)

scaler_x_train = StandardScaler().fit(x_train)
scaler_x_test = StandardScaler().fit(x_test)

x_train_trans = scaler_x_train.transform(x_train)
x_test_trans = scaler_x_test.transform(x_test)

x_train_fit = pca.fit_transform(x_train_trans)*-1
x_test_fit = pca.transform(x_test_trans)*-1
```

```
x_train_inv = scaler_x_train.inverse_transform(x_train_fit)
x_train_fin = x_train_inv[:,2:3]

x_test_inv = scaler_x_train.inverse_transform(x_test_fit)
x_test_fin = x_test_inv[:,2:3]

model = LinearRegression().fit(x_train_fin, y_train)
b0, b1 = model.intercept_, model.coef_
RMSE = mean_squared_error(y_train, model.predict(x_train_fin),
                          squared = False)
print('RMSE: '+str(RMSE))
```

```
RMSE: 0.553254884171123
```

[62]:
```
b0, b1
```

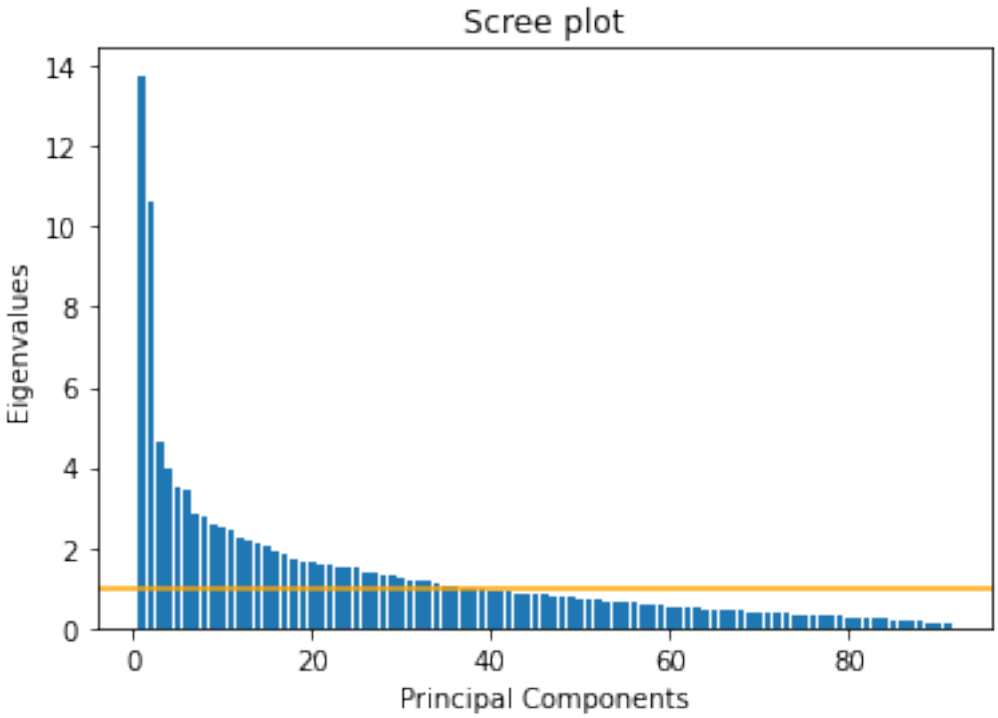
[62]:
```
(4.10164831841852, array([0.03937008]))
```

[63]:
```
x=dark
y=rate_dark


x_train, x_test, y_train, y_test = train_test_split(x,                # X matrix
                                                    y,                # Y vector
                                                    test_size=0.20,   # percent␣
 ↪of data in test
                                                    random_state=seed  # set␣
 ↪random state
                                                   )

regressor = LinearRegression().fit(x_train, y_train)

#print('Intercept', regressor.intercept_)# Print the intercept
#print('Coefficient', regressor.coef_)   # Print the coefficient


model = LinearRegression().fit(x,y)
b0, b1 = model.intercept_, model.coef_

RMSE = mean_squared_error(y_train, model.predict(x_train),
                          squared = False)
RMSE
```

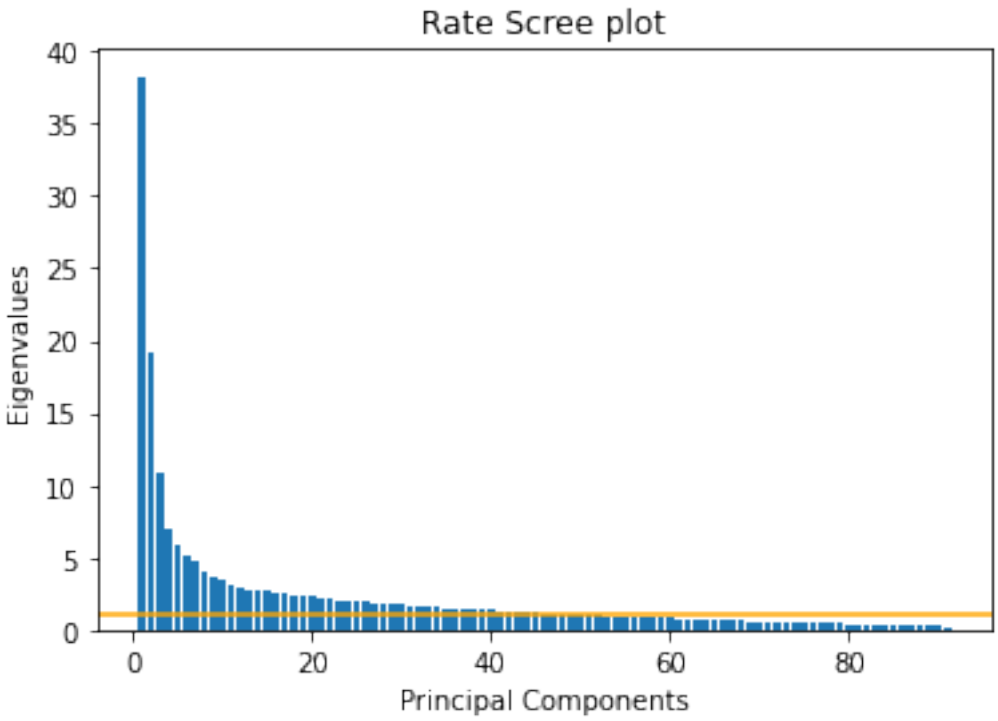
[63]:
```
1.2405290935458382
```

[64]:
```
# 10) Can you determine the political orientation of the users (to simplify␣
↪things and avoid gross class imbalance issues, you can consider just 2␣
↪classes: "left" (progressive & liberal) vs. "nonleft" (everyone else)) from␣
↪all the other information available, using any classification model of your␣
↪choice? Make sure to comment on the classification quality of this model.
```

[65]:
```
energy_predictor = pd.DataFrame(data.loc[:,91:181])
x = energy_predictor.to_numpy()
zscoredData = stats.zscore(energy_predictor)
pca = PCA().fit(energy_predictor)
eigVals = pca.explained_variance_

loadings = pca.components_*-1
origDataNewCoordinates = pca.fit_transform(energy_predictor)*-1
# Scree plot:
numPredictors = 91
plt.bar(np.linspace(1,numPredictors,numPredictors),eigVals)
plt.title('Scree plot')
plt.axhline(y = 1, color = 'orange', linestyle = '-')
plt.xlabel('Principal Components')
plt.ylabel('Eigenvalues')
plt.show()
```

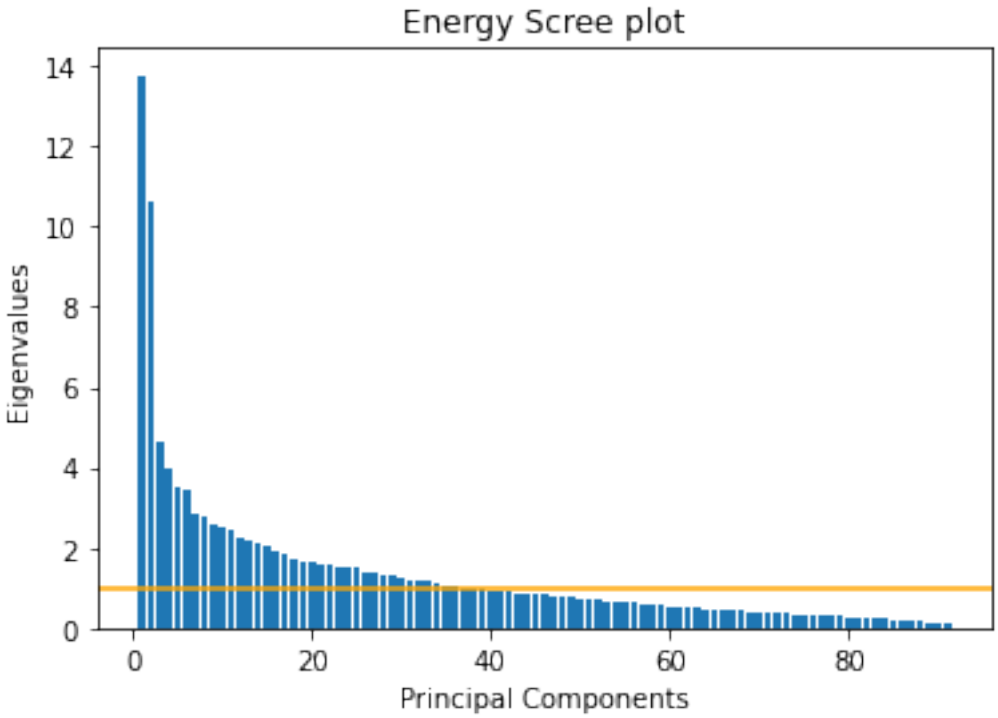


[66]:
```
rate_predictor = pd.DataFrame(rate.loc[:,0:90])
x = rate_predictor.to_numpy()
zscoredData = stats.zscore(rate_predictor)
pca = PCA().fit(rate_predictor)
eigVals = pca.explained_variance_

loadings = pca.components_*-1
origDataNewCoordinates = pca.fit_transform(rate_predictor)*-1
# Scree plot:
numPredictors = 91
plt.bar(np.linspace(1,numPredictors,numPredictors),eigVals)
plt.title('Rate Scree plot')
plt.axhline(y = 1, color = 'orange', linestyle = '-')
plt.xlabel('Principal Components')
plt.ylabel('Eigenvalues')
plt.show()
```



[67]:
```
energy_predictor = pd.DataFrame(data.loc[:,91:181])
x = energy_predictor.to_numpy()
zscoredData = stats.zscore(energy_predictor)
pca = PCA().fit(energy_predictor)
eigVals = pca.explained_variance_
```

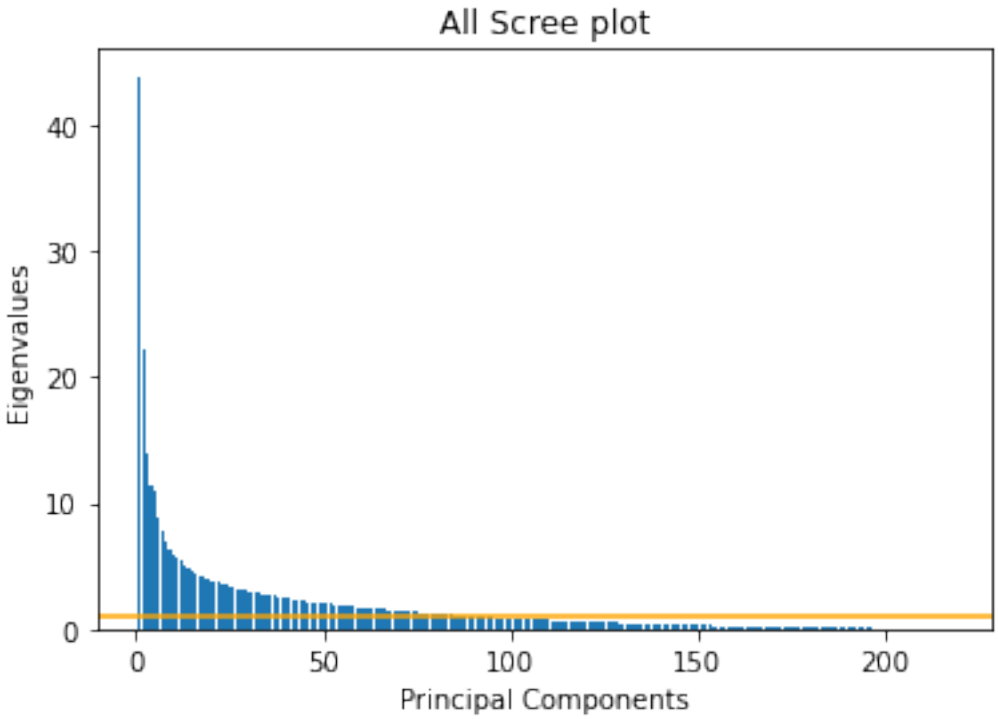
```
loadings = pca.components_*-1
origDataNewCoordinates = pca.fit_transform(energy_predictor)*-1
# Scree plot:
numPredictors = 91
plt.bar(np.linspace(1,numPredictors,numPredictors),eigVals)
plt.title('Energy Scree plot')
plt.axhline(y = 1, color = 'orange', linestyle = '-')
plt.xlabel('Principal Components')
plt.ylabel('Eigenvalues')
plt.show()
```



[68]:
```
data_drop = data.dropna()
data_drop = data_drop.iloc[:,0:217].to_numpy()
```

[69]:
```
zscoredData = stats.zscore(data_drop)
pca = PCA().fit(data_drop)
eigVals = pca.explained_variance_

loadings = pca.components_*-1
origDataNewCoordinates = pca.fit_transform(data_drop)*-1
#Scree plot:
numPredictors = 217
plt.bar(np.linspace(1,numPredictors,numPredictors),eigVals,align = 'center')
```

```
plt.title('All Scree plot')
plt.axhline(y = 1, color = 'orange', linestyle = '-')
plt.xlabel('Principal Components')
plt.ylabel('Eigenvalues')
plt.show()
```



[70]:
```
x = energy_predictor.to_numpy()
y = political2




x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
 ↪3,random_state=seed)

scaler_x_train = StandardScaler().fit(x_train)
scaler_x_test = StandardScaler().fit(x_test)

x_train_trans = scaler_x_train.transform(x_train)
x_test_trans = scaler_x_test.transform(x_test)

x_train_fit = pca.fit_transform(x_train_trans)*-1
```

```
x_test_fit = pca.transform(x_test_trans)*-1
```

[71]:
```
x_train_inv = scaler_x_train.inverse_transform(x_train_fit)
x_train_fin = x_train_inv[:,:20]

x_test_inv = scaler_x_train.inverse_transform(x_test_fit)
x_test_fin = x_test_inv[:,:20]
```

[72]:
```
model = LogisticRegression()
model.fit(x_train_fin, y_train)
```

[72]:
```
LogisticRegression()
```

[73]:
```
model.score(x_train_fin,y_train)
```

[73]:
```
0.6333333333333333
```

[74]:
```
model.score(x_test_fin,y_test)
```

[74]:
```
0.6222222222222222
```

[75]:
```
RMSE = mean_squared_error(y_train, model.predict(x_train_fin),
                          squared = False)
RMSE
```

[75]:
```
0.6055300708194983
```

[ ]:
```

```