Clustering in Python using an instructor evaluation data set

A Coding and Statistical Analysis Challenge

Kamal Babaei Sonbolabadi

Fall-2018

Introduction: In this independent Project, I have used statistical Methods and Machine Learning algorithms to Cluster a data set that is from a student evaluation in a school in Turkey.

Description of Data: This dataset has 5821 rows and 33 columns. The Attributes are as follows:

instr: Instructor's identifier; values taken from {1,2,3}

class: Course code (descriptor); values taken from {1-13}

nb.repeat: Number of times the student is taking this course; values taken from {0,1,2,3,...}

attendance: Code of the level of attendance; values from {0, 1, 2, 3, 4}

difficulty: Level of difficulty of the course as perceived by the student; values taken from {1,2,3,4,5}

Q1: The semester course content, teaching method and evaluation system were provided at the start.

Q2: The course aims and objectives were clearly stated at the beginning of the period.

Q3: The course was worth the amount of credit assigned to it.

Q4: The course was taught according to the syllabus announced on the first day of class.

.

Q28: The Instructor treated all students in a right and objective manner.

Q1-Q28 are all Likert-type, meaning that the values are taken from {1,2,3,4,5}

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q4	Q5	...	Q19	Q20
0	1	2	1	0	4	3	3	3	3	3	...	3	3
1	1	2	1	1	3	3	3	3	3	3	...	3	3
2	1	2	1	2	4	5	5	5	5	5	...	5	5
3	1	2	1	1	3	3	3	3	3	3	...	3	3
4	1	2	1	0	1	1	1	1	1	1	...	1	1

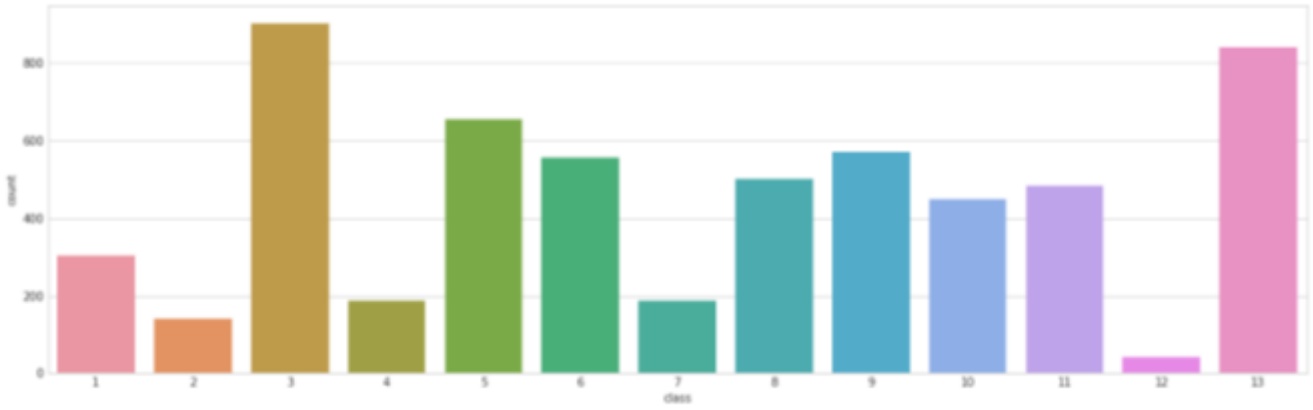
Step by step explanation: First, I have loaded data to get some descriptive details; for instance, for which course the students have given the most responses (and it is found to be Course 3 from below graph) or how the rating has been given by students for each question, just to name a few.

```
#Imports

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('whitegrid')

%matplotlib inline

dataset = pd.read_csv('C:/Users/Kamal/Desktop/turkiye-student-evaluation_generic.csv')

dataset.head()

dataset.describe()

plt.figure(figsize=(20, 6))

sns.countplot(x='class', data=dataset)

plt.figure(figsize=(20, 20))

sns.boxplot(data=dataset.iloc[:,5:31 ]);
```

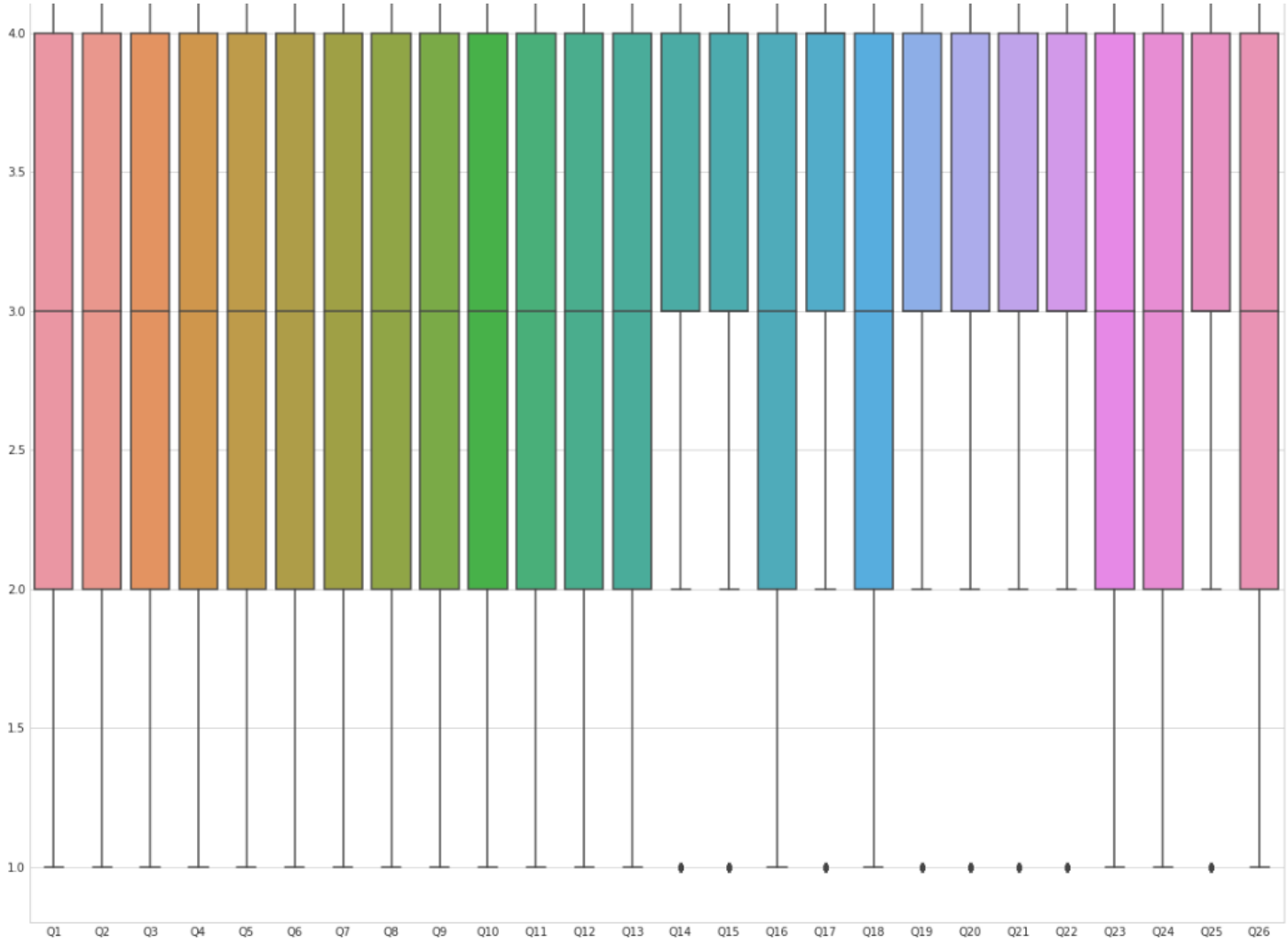
Discription:

	instr	class	nb.repeat	attendance	difficulty	Q1
count	5820.000000	5820.000000	5820.000000	5820.000000	5820.000000	5820.000000
mean	2.485567	7.276289	1.214089	1.675601	2.783505	2.929897
std	0.718473	3.688175	0.532376	1.474975	1.348987	1.341077
min	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000
25%	2.000000	4.000000	1.000000	0.000000	1.000000	2.000000
50%	3.000000	7.000000	1.000000	1.000000	3.000000	3.000000
75%	3.000000	10.000000	1.000000	3.000000	4.000000	4.000000
max	3.000000	13.000000	3.000000	4.000000	5.000000	5.000000

The following diagram shows that students have given the most responses to Course #3:



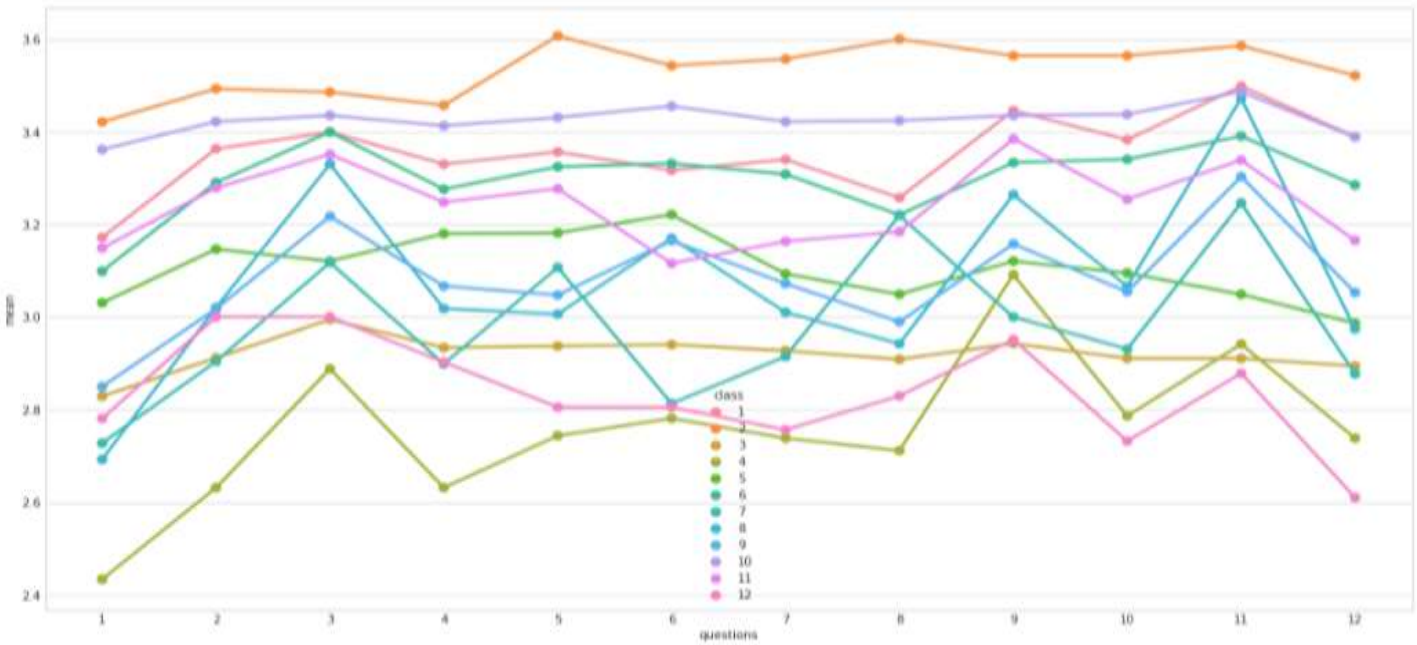
The following graph shows that a lot less students have given completely disagree (Rating 1) for Question Q14, Q15, Q17, Q19 - Q22, Q25.



Now let's see how the students have responded for the questions against classes:

```
# Calculate mean for each question response for all the classes.
questionmeans = []
classlist = []
questions = []
totalplotdata = pd.DataFrame(list(zip(classlist,questions,questionmeans))
                ,columns=['class','questions', 'mean'])
for class_num in range(1,13):
    class_data = dataset[(dataset["class"]==class_num)]
```

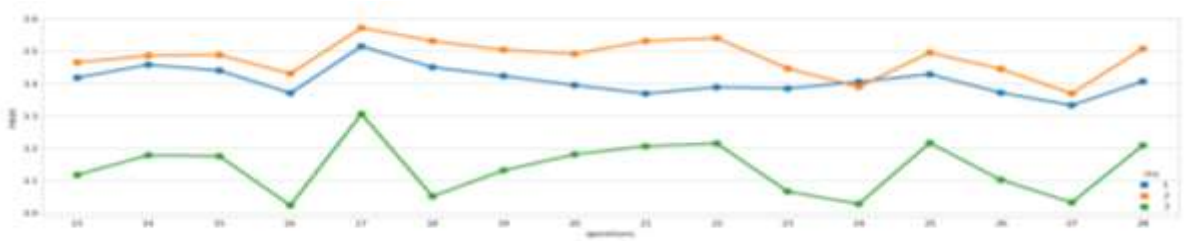
```
    questionmeans = []
    classlist = []
    questions = []
    for num in range(1,13):
        questions.append(num)
    #Class related questions are from Q1 to Q12
    for col in range(5,17):
        questionmeans.append(class_data.iloc[:,col].mean())
    classlist += 12 * [class_num]
    print(classlist)
    plotdata = pd.DataFrame(list(zip(classlist,questions,questionmeans))
                ,columns=['class','questions', 'mean'])
    totalplotdata = totalplotdata.append(plotdata, ignore_index=True)
plt.figure(figsize=(20, 10))
sns.pointplot(x="questions", y="mean", data=totalplotdata, hue="class")
```



The graph shows that we have best ratings from Class 2 and worst rating from class 4 students.

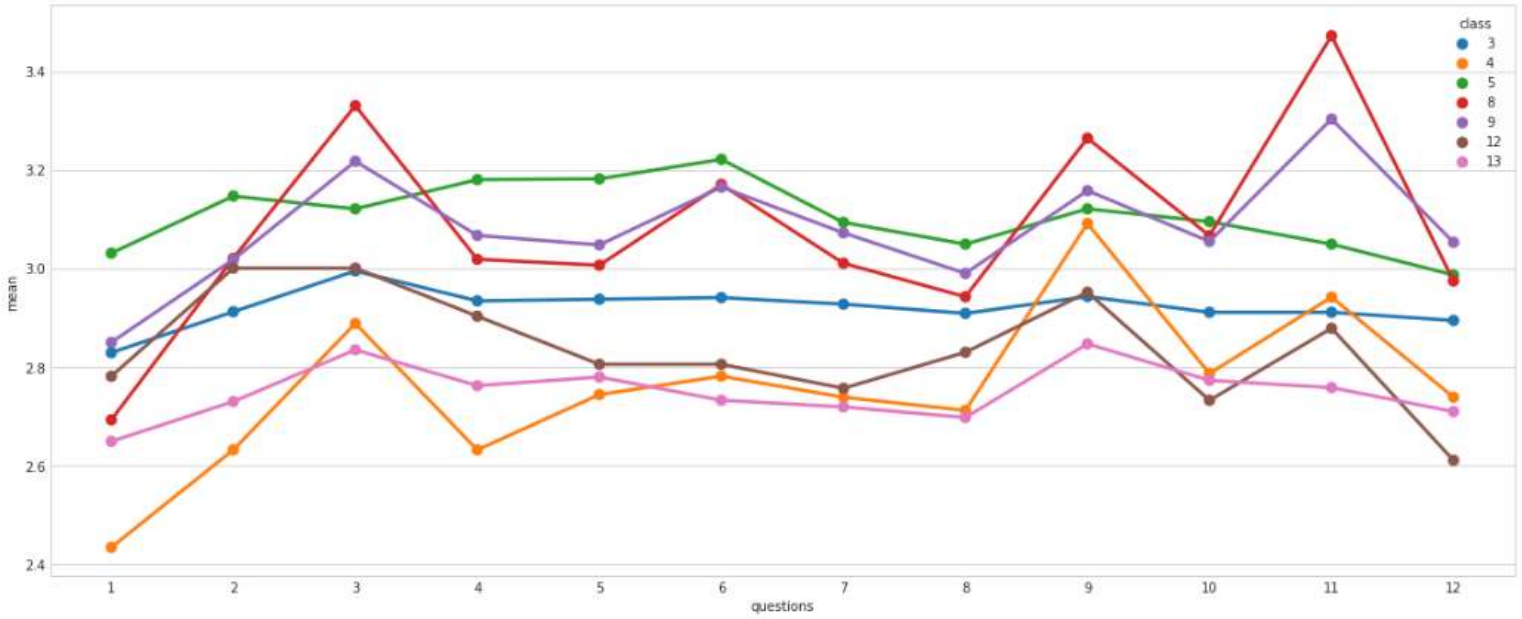
Now let's see how rating has been given against instructor wise.

```
# calculate mean for each question response for all the classes.
questionmeans = []
inslist = []
questions = []
totalplotdata = pd.DataFrame(list(zip(inslist,questions,questionmeans))
                ,columns=['ins','questions', 'mean'])
for ins_num in range(1,4):
    ins_data = dataset[(dataset["instr"]==ins_num)]
    questionmeans = []
    inslist = []
    questions = []
    for num in range(13,29):
        questions.append(num)
        for col in range(17,33):
        questionmeans.append(ins_data.iloc[:,col].mean())
    inslist += 16 * [ins_num]
    plotdata = pd.DataFrame(list(zip(inslist,questions,questionmeans))
                ,columns=['ins','questions', 'mean'])
    totalplotdata = totalplotdata.append(plotdata, ignore_index=True)
plt.figure(figsize=(20, 5))
sns.pointplot(x="questions", y="mean", data=totalplotdata, hue="ins")
```



Based on the provided graph we can see that According to the Student ratings Instructor 1 and 2 are performing well and got similar ratings, but Instructor 3 got less ratings. So we can further explore which course instructor 3 teaches and find out which course got least ratings.

```
# Calculate mean for each question response for all the classes for Instructor 3
dataset_inst3 = dataset[(dataset["instr"]==3)]
class_array_for_inst3 = dataset_inst3["class"].unique().tolist()
questionmeans = []
classlist = []
questions = []
totalplotdata = pd.DataFrame(list(zip(classlist,questions,questionmeans))
                ,columns=['class','questions', 'mean'])
for class_num in class_array_for_inst3:
    class_data = dataset_inst3[(dataset_inst3["class"]==class_num)]
        questionmeans = []
    classlist = []
    questions = []
    for num in range(1,13):
        questions.append(num)
    for col in range(5,17):
        questionmeans.append(class_data.iloc[:,col].mean())
    classlist += 12 * [class_num]
    plotdata = pd.DataFrame(list(zip(classlist,questions,questionmeans))
                ,columns=['class','questions', 'mean'])
    totalplotdata = totalplotdata.append(plotdata, ignore_index=True)
plt.figure(figsize=(20, 8))
sns.pointplot(x="questions", y="mean", data=totalplotdata, hue="class")
```



NOW we begin to cluster the students based on the questionnaire data.

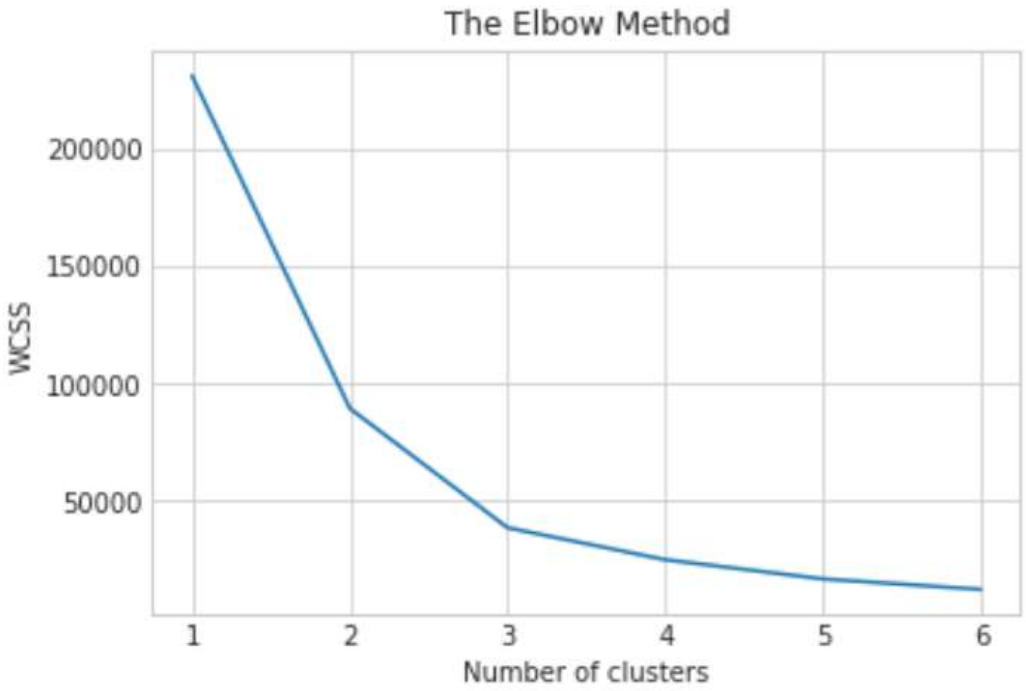
#Let's try to cluster all the students based on the Question responses data.

dataset_questions = dataset.iloc[:,5:33]

dataset_questions.head()

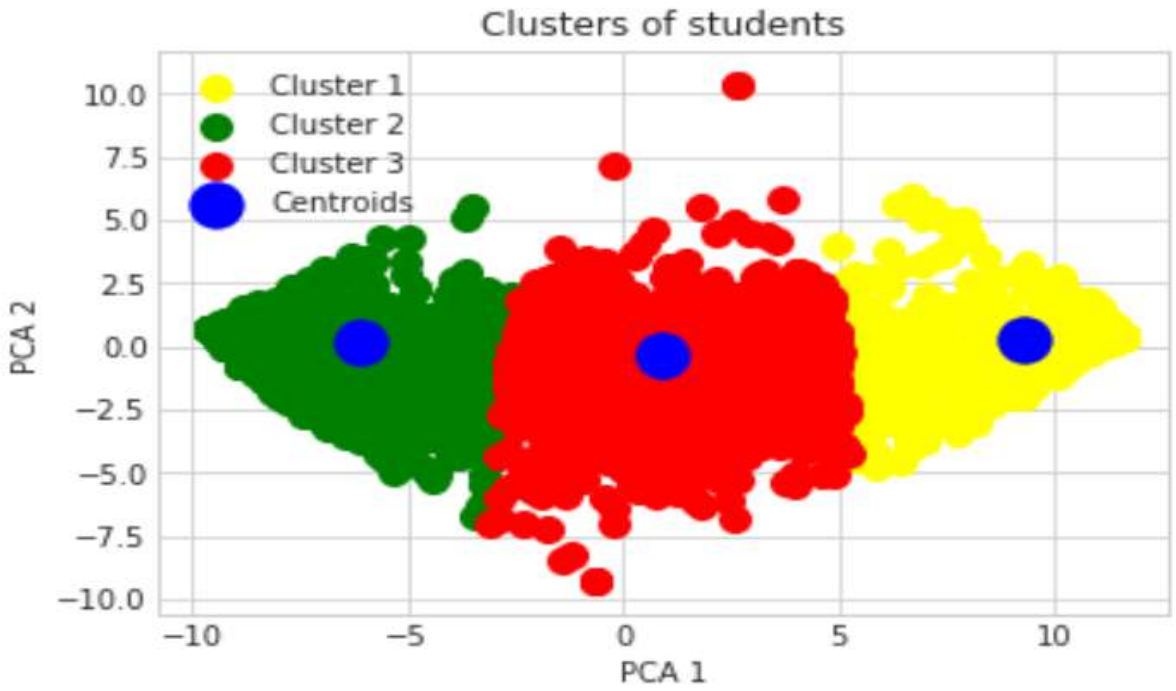
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	...	Q19	Q20	Q21	Q22	Q23
0	3	3	3	3	3	3	3	3	3	3	...	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3	...	3	3	3	3	3
2	5	5	5	5	5	5	5	5	5	5	...	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	...	3	3	3	3	3
4	1	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1

```
#lets do a PCA for feature dimensional reduction
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
dataset_questions_pca = pca.fit_transform(dataset_questions)
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 7):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(dataset_questions_pca)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 7), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



Based on the Elbow graph, we can go for 3 clusters.

```
kmeans = KMeans(n_clusters = 3, init = 'k-means++')

y_kmeans = kmeans.fit_predict(dataset_questions_pca)

# Visualising the clusters

plt.scatter(dataset_questions_pca[y_kmeans == 0, 0], dataset_questions_pca[y_kmeans == 0, 1], s = 100, c = 'yellow', label = 'Cluster 1')

plt.scatter(dataset_questions_pca[y_kmeans == 1, 0], dataset_questions_pca[y_kmeans == 1, 1], s = 100, c = 'green', label = 'Cluster 2')

plt.scatter(dataset_questions_pca[y_kmeans == 2, 0], dataset_questions_pca[y_kmeans == 2, 1], s = 100, c = 'red', label = 'Cluster 3')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'blue', label = 'Centroids')

plt.title('Clusters of students')

plt.xlabel('PCA 1')

plt.ylabel('PCA 2')

plt.legend()

plt.show()
```



Looking at the provided graph, we see that we have 3 clusters of students who have given “Negative”, “Neutral” and “Positive” feedback.

 # Now we check the count of students in each cluster

import collections

collections.Counter(y_kmeans)

***Output**: Counter({0: 1240, 1: 2222, 2: 2358})*

So we have 2358 students who have given negative ratings overall , 2222 students with positive ratings and 1240 students with neutral response.

Now to find the optimal number of clusters, we use dendrogram:

 # Using the dendrogram to find the optimal number of clusters

import scipy.cluster.hierarchy as sch

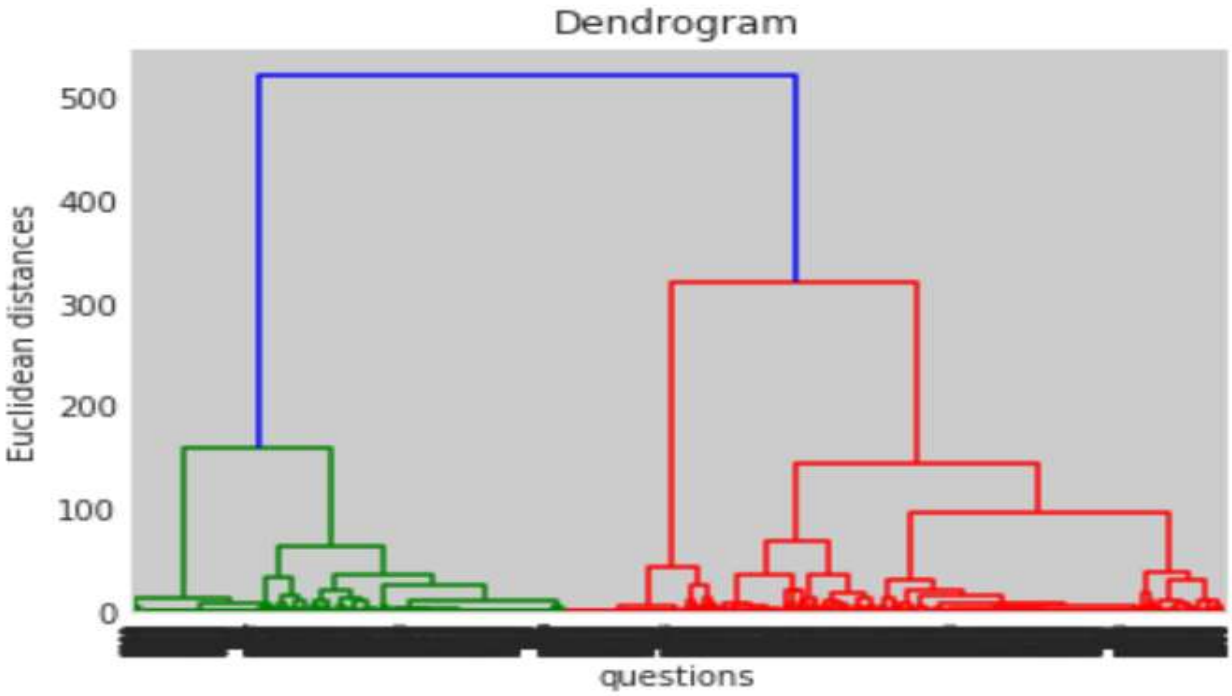
dendrogram = sch.dendrogram(sch.linkage(dataset_questions_pca, method = 'ward'))

plt.title('Dendrogram')

plt.xlabel('questions')

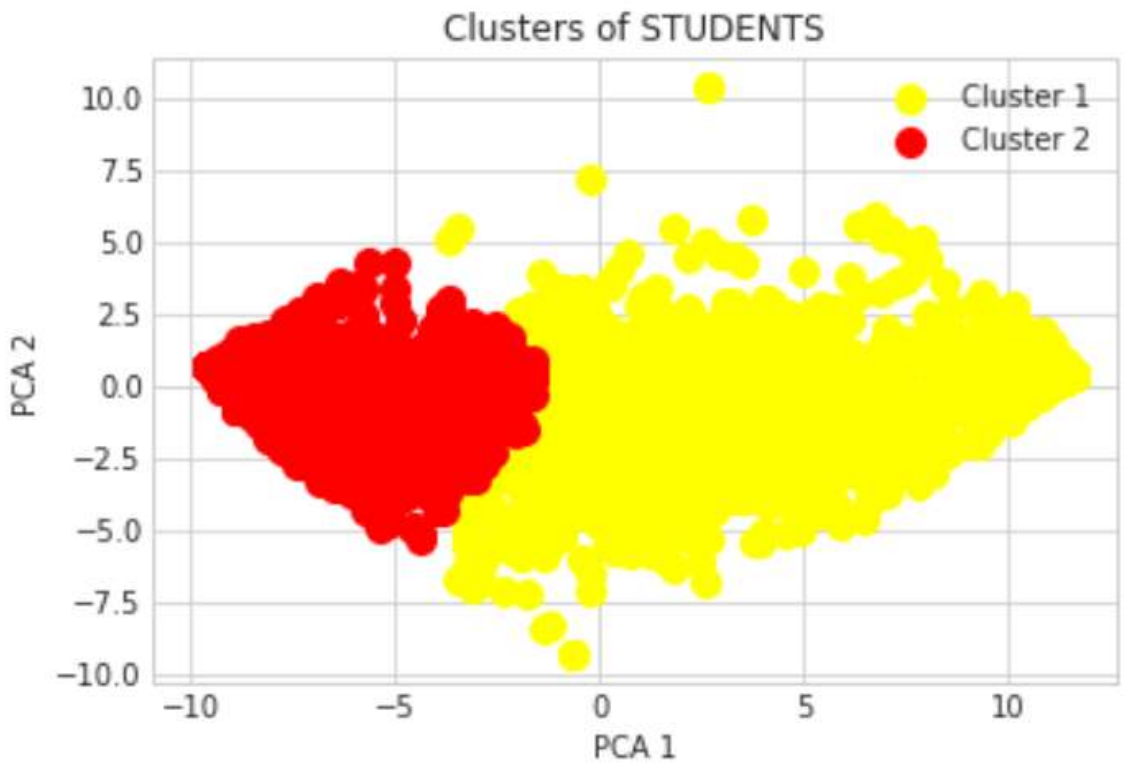
plt.ylabel('Euclidean distances')

plt.show()



Let’s try having 2 clusters:

```
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(dataset_questions_pca)
X = dataset_questions_pca
# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'yellow', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'red', label = 'Cluster 2')
plt.title('Clusters of STUDENTS')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend()
plt.show()
```



Now let's calculate the count of students in each cluster

```
# Let me check the count of students in each cluster

import collections

collections.Counter(y_hc)
```

Output:*counter({0: 3502, 1: 2318})*

Finally, if we compare the clusters of Kmeans and Hierarchical processes, we can see that the red cluster (Negative) is matching approximately.

Another method for doing this project would be factor analysis. I am currently working on that.

Below is the complete code from A to Z:

```
#Imports

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('whitegrid')

%matplotlib inline

dataset = pd.read_csv('C:/Users/Kamal/Desktop/turkiye-student-evaluation_generic.csv')

dataset.head()

dataset.describe()

plt.figure(figsize=(20, 6))

sns.countplot(x='class', data=dataset)

plt.figure(figsize=(20, 20))

sns.boxplot(data=dataset.iloc[:,5:31 ]);

# Calculate mean for each question response for all the classes.

questionmeans = []

classlist = []

questions = []
```

```
totalplotdata = pd.DataFrame(list(zip(classlist,questions,questionmeans))
                ,columns=['class','questions', 'mean'])
for class_num in range(1,13):
    class_data = dataset[(dataset["class"]==class_num)]

    questionmeans = []
    classlist = []
    questions = []

    for num in range(1,13):
        questions.append(num)
    #Class related questions are from Q1 to Q12
    for col in range(5,17):
        questionmeans.append(class_data.iloc[:,col].mean())
    classlist += 12 * [class_num]
    print(classlist)
    plotdata = pd.DataFrame(list(zip(classlist,questions,questionmeans))
                ,columns=['class','questions', 'mean'])
    totalplotdata = totalplotdata.append(plotdata, ignore_index=True)

plt.figure(figsize=(20, 10))
sns.pointplot(x="questions", y="mean", data=totalplotdata, hue="class")
# Calculate mean for each question response for all the classes.
questionmeans = []
inslist = []
questions = []
totalplotdata = pd.DataFrame(list(zip(inslist,questions,questionmeans))
```

```
                ,columns=['ins','questions', 'mean'])
for ins_num in range(1,4):
    ins_data = dataset[(dataset["instr"]==ins_num)]
    questionmeans = []
    inslist = []
    questions = []

    for num in range(13,29):
        questions.append(num)

    for col in range(17,33):
        questionmeans.append(ins_data.iloc[:,col].mean())
    inslist += 16 * [ins_num]
    plotdata = pd.DataFrame(list(zip(inslist,questions,questionmeans))
                ,columns=['ins','questions', 'mean'])
    totalplotdata = totalplotdata.append(plotdata, ignore_index=True)
plt.figure(figsize=(20, 5))
sns.pointplot(x="questions", y="mean", data=totalplotdata, hue="ins")
# Calculate mean for each question response for all the classes for Instructor 3
dataset_inst3 = dataset[(dataset["instr"]==3)]
class_array_for_inst3 = dataset_inst3["class"].unique().tolist()
questionmeans = []
classlist = []
questions = []
totalplotdata = pd.DataFrame(list(zip(classlist,questions,questionmeans))
                ,columns=['class','questions', 'mean'])
for class_num in class_array_for_inst3:
```

```
    class_data = dataset_inst3[(dataset_inst3["class"]==class_num)]

    questionmeans = []
    classlist = []
    questions = []

    for num in range(1,13):
        questions.append(num)

    for col in range(5,17):
        questionmeans.append(class_data.iloc[:,col].mean())
    classlist += 12 * [class_num]

    plotdata = pd.DataFrame(list(zip(classlist,questions,questionmeans))
                ,columns=['class','questions', 'mean'])
    totalplotdata = totalplotdata.append(plotdata, ignore_index=True)
plt.figure(figsize=(20, 8))
sns.pointplot(x="questions", y="mean", data=totalplotdata, hue="class")
#Lets try to cluster all the students based on the Question responses data.
dataset_questions = dataset.iloc[:,5:33]
#lets do a PCA for feature dimensional reduction
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
dataset_questions_pca = pca.fit_transform(dataset_questions)
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 7):
```

```
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(dataset_questions_pca)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 7), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters = 3, init = 'k-means++')
y_kmeans = kmeans.fit_predict(dataset_questions_pca)
# Visualising the clusters
plt.scatter(dataset_questions_pca[y_kmeans == 0, 0], dataset_questions_pca[y_kmeans == 0,
1], s = 100, c = 'yellow', label = 'Cluster 1')
plt.scatter(dataset_questions_pca[y_kmeans == 1, 0], dataset_questions_pca[y_kmeans == 1,
1], s = 100, c = 'green', label = 'Cluster 2')
plt.scatter(dataset_questions_pca[y_kmeans == 2, 0], dataset_questions_pca[y_kmeans == 2,
1], s = 100, c = 'red', label = 'Cluster 3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'blue', label
= 'Centroids')
plt.title('Clusters of students')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend()
plt.show()
# Let me check the count of students in each cluster
import collections
collections.Counter(y_kmeans)
# Using the dendrogram to find the optimal number of clusters
```

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(dataset_questions_pca, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('questions')
plt.ylabel('Euclidean distances')
plt.show()
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(dataset_questions_pca)
X = dataset_questions_pca
# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'yellow', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'red', label = 'Cluster 2')
plt.title('Clusters of STUDENTS')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend()
plt.show()
# Let me check the count of students in each cluster
import collections
collections.Counter(y_hc)
```