Importing the libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from collections import Counter as count
```

Importing the dataset

In [2]:
```
ds=pd.read_csv("Data/turkiye-student-evaluation_generic.csv")
```

visualizing the dataset

In [3]:
```
ds.shape
```

Out[3]: (5820, 33)

In [4]:
```
ds.head()
```

Out[4]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q4	Q5	...	Q19	Q20	Q21	Q22
0	1	2	1	0	4	3	3	3	3	3	...	3	3	3	3
1	1	2	1	1	3	3	3	3	3	3	...	3	3	3	3
2	1	2	1	2	4	5	5	5	5	5	...	5	5	5	5
3	1	2	1	1	3	3	3	3	3	3	...	3	3	3	3
4	1	2	1	0	1	1	1	1	1	1	...	1	1	1	1

5 rows × 33 columns

In [5]:
```
ds.describe()
```

Out[5]:

	instr	class	nb.repeat	attendance	difficulty	Q1	
count	5820.000000	5820.000000	5820.000000	5820.000000	5820.000000	5820.000000	5820.0000
mean	2.485567	7.276289	1.214089	1.675601	2.783505	2.929897	3.0738
std	0.718473	3.688175	0.532376	1.474975	1.348987	1.341077	1.2852
min	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.0000
25%	2.000000	4.000000	1.000000	0.000000	1.000000	2.000000	2.0000
50%	3.000000	7.000000	1.000000	1.000000	3.000000	3.000000	3.0000
75%	3.000000	10.000000	1.000000	3.000000	4.000000	4.000000	4.0000
max	3.000000	13.000000	3.000000	4.000000	5.000000	5.000000	5.0000

8 rows × 33 columns

In [6]:
```
ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5820 entries, 0 to 5819
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   instr       5820 non-null   int64
 1   class       5820 non-null   int64
 2   nb.repeat   5820 non-null   int64
 3   attendance  5820 non-null   int64
 4   difficulty  5820 non-null   int64
 5   Q1          5820 non-null   int64
 6   Q2          5820 non-null   int64
 7   Q3          5820 non-null   int64
 8   Q4          5820 non-null   int64
 9   Q5          5820 non-null   int64
 10  Q6          5820 non-null   int64
 11  Q7          5820 non-null   int64
 12  Q8          5820 non-null   int64
 13  Q9          5820 non-null   int64
 14  Q10         5820 non-null   int64
 15  Q11         5820 non-null   int64
 16  Q12         5820 non-null   int64
 17  Q13         5820 non-null   int64
 18  Q14         5820 non-null   int64
 19  Q15         5820 non-null   int64
 20  Q16         5820 non-null   int64
 21  Q17         5820 non-null   int64
 22  Q18         5820 non-null   int64
 23  Q19         5820 non-null   int64
 24  Q20         5820 non-null   int64
 25  Q21         5820 non-null   int64
 26  Q22         5820 non-null   int64
 27  Q23         5820 non-null   int64
 28  Q24         5820 non-null   int64
 29  Q25         5820 non-null   int64
 30  Q26         5820 non-null   int64
 31  Q27         5820 non-null   int64
 32  Q28         5820 non-null   int64
dtypes: int64(33)
memory usage: 1.5 MB
```

In [7]:
```
ds.columns
```

Out[7]:
```
Index(['instr', 'class', 'nb.repeat', 'attendance', 'difficulty', 'Q1', 'Q2',
       'Q3', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10', 'Q11', 'Q12', 'Q13',
       'Q14', 'Q15', 'Q16', 'Q17', 'Q18', 'Q19', 'Q20', 'Q21', 'Q22', 'Q23',
       'Q24', 'Q25', 'Q26', 'Q27', 'Q28'],
      dtype='object')
```

In [8]:
```
ds.shape
```

Out[8]:
```
(5820, 33)
```

In [9]:
```
ds.isnull()
```

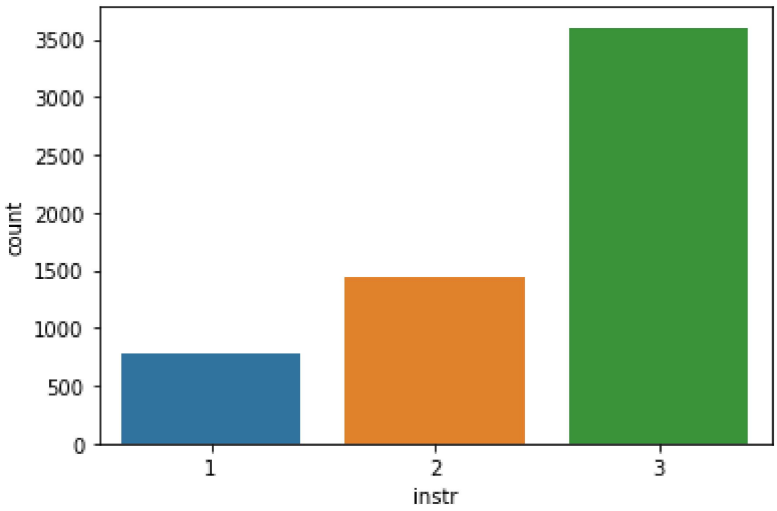
Out[9]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q4	Q5	...	Q19
0	False	False	False	False	False	False	False	False	False	False	...	False
1	False	False	False	False	False	False	False	False	False	False	...	False
2	False	False	False	False	False	False	False	False	False	False	...	False
3	False	False	False	False	False	False	False	False	False	False	...	False
4	False	False	False	False	False	False	False	False	False	False	...	False
...	...	...	...	...	...	...	...	...	...	...	...	...
5815	False	False	False	False	False	False	False	False	False	False	...	False
5816	False	False	False	False	False	False	False	False	False	False	...	False
5817	False	False	False	False	False	False	False	False	False	False	...	False
5818	False	False	False	False	False	False	False	False	False	False	...	False
5819	False	False	False	False	False	False	False	False	False	False	...	False

5820 rows × 33 columns

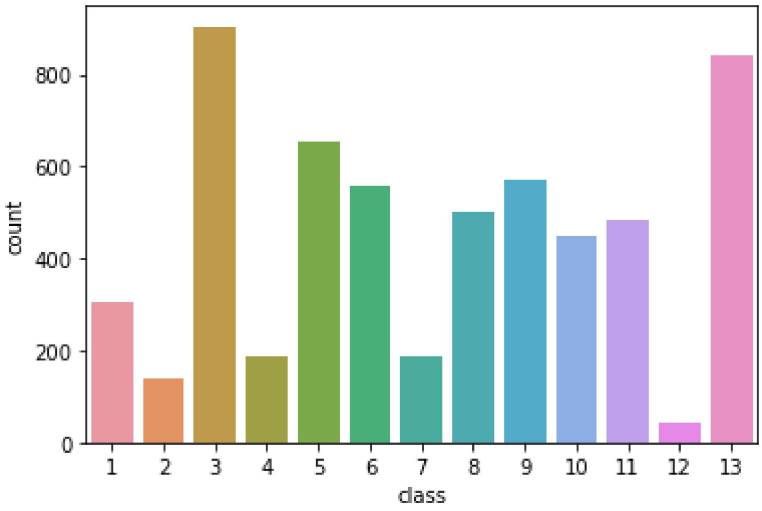
In [10]:
```
sns.countplot(x='instr',data=ds)
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f4ab625708>
```

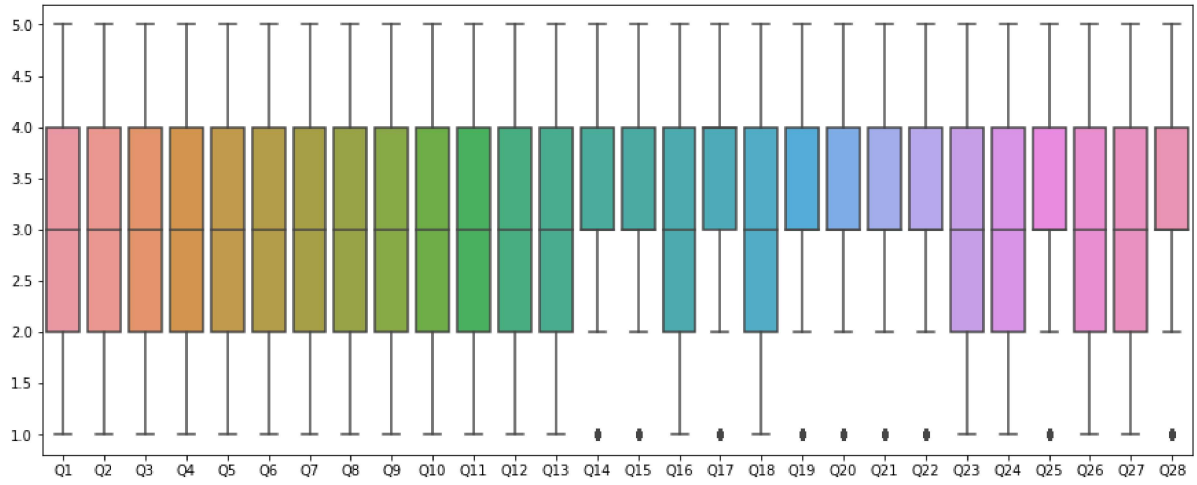


In [11]:
```
sns.countplot(x='class',data=ds)
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f4ab9977c8>
```



In [12]:
```
plt.figure(figsize=(15,6))
sns.boxplot(data=ds.loc[:,'Q1':'Q28'])
plt.show()
```



Scaling the data

In [13]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
ds=pd.DataFrame(sc.fit_transform(ds),columns=ds.columns)
ds
```

Out[13]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	
0	-2.06785	-1.430719	-0.402174	-1.136118	0.901862	0.052278	-0.057490	-0.142561	-0.0
1	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.057490	-0.142561	-0.0
2	-2.06785	-1.430719	-0.402174	0.219954	0.901862	1.543745	1.498760	1.453023	1.4
3	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.057490	-0.142561	-0.0
4	-2.06785	-1.430719	-0.402174	-1.136118	-1.322221	-1.439189	-1.613740	-1.738145	-1.6
...	...	...	...	...	...	...	...	...	
5815	0.71607	1.552042	-0.402174	-1.136118	-1.322221	-1.439189	-1.613740	-1.738145	-1.6
5816	0.71607	1.552042	-0.402174	0.897990	0.901862	0.798012	0.720635	0.655231	0.7
5817	0.71607	1.552042	-0.402174	-1.136118	0.901862	1.543745	1.498760	1.453023	1.4
5818	0.71607	1.552042	-0.402174	-0.458082	-0.580860	-1.439189	-1.613740	-1.738145	-1.6
5819	0.71607	1.552042	-0.402174	-0.458082	-0.580860	-1.439189	-1.613740	-1.738145	-1.6

5820 rows × 33 columns

Implementation of k-means algorithm

In [14]:
```
from sklearn.cluster import KMeans
```

In [15]:
```
dataset=ds.iloc[:,5:33]
```

In [16]:
```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
datasetpca=pca.fit_transform(dataset)
```

In [17]:
```
pca.explained_variance_ratio_.cumsum()[1]
```

Out[17]:
```
0.8676364447297639
```

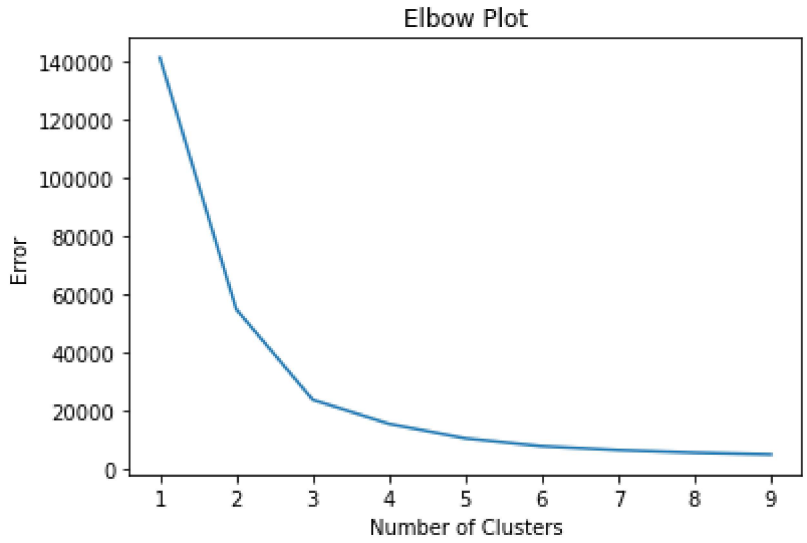
In [18]:
```
clusterrange=range(1,10)
clustererror=[]
for i in clusterrange:
    clusters=KMeans(i,n_init=10,max_iter=100)
    clusters.fit(datasetpca)
    clustererror.append(clusters.inertia_)
pd.DataFrame({'number of clusters :':clusterrange, 'Error' : clustererror})
```

Out[18]:

	number of clusters :	Error
0	1	141390.035033
1	2	54785.323341
2	3	23705.255472
3	4	15359.457586
4	5	10365.065716
5	6	7665.463804
6	7	6311.465133
7	8	5435.260018
8	9	4822.905291

Checking the number of clusters can be made using Elbow method

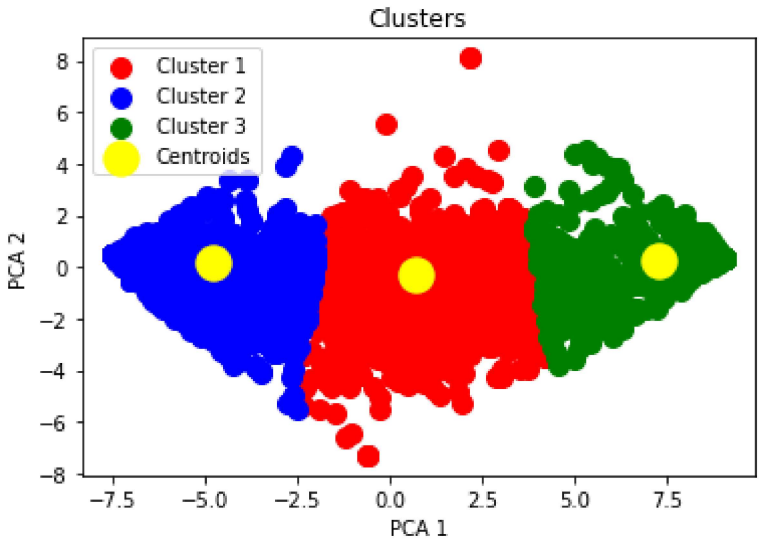
In [19]:
```
plt.plot(clusterrange, clustererror )
plt.title('Elbow Plot')
plt.xlabel('Number of Clusters')
plt.ylabel('Error')
plt.show()
```



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

In [20]:
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++')
y_kmeans = kmeans.fit_predict(datasetpca)
```

In [21]:
```
plt.scatter(datasetpca[y_kmeans == 0, 0], datasetpca[y_kmeans == 0, 1], s = 10
0, c = 'red', label = 'Cluster 1')
plt.scatter(datasetpca[y_kmeans == 1, 0], datasetpca[y_kmeans == 1, 1], s = 10
0, c = 'blue', label = 'Cluster 2')
plt.scatter(datasetpca[y_kmeans == 2, 0], datasetpca[y_kmeans == 2, 1], s = 10
0, c = 'green', label = 'Cluster 3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =
300, c = 'yellow', label = 'Centroids')
plt.title('Clusters')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend()
plt.show()
```



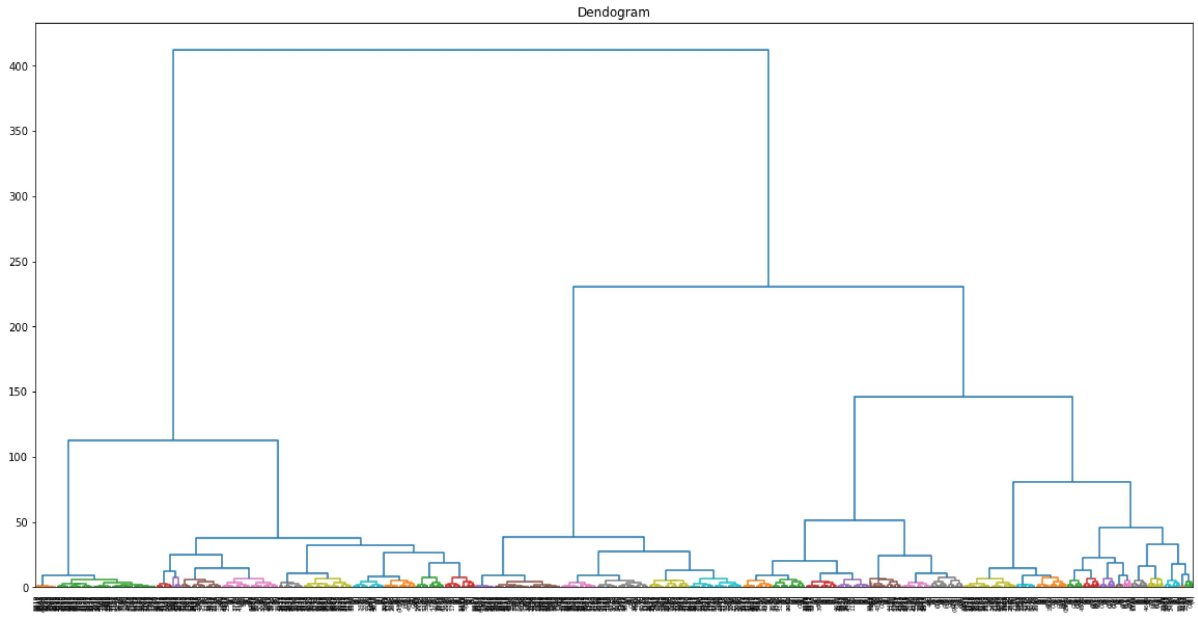
Result of kmeans

In [22]:
```
count(y_kmeans)
```

Out[22]:
```
Counter({0: 2362, 1: 2229, 2: 1229})
```

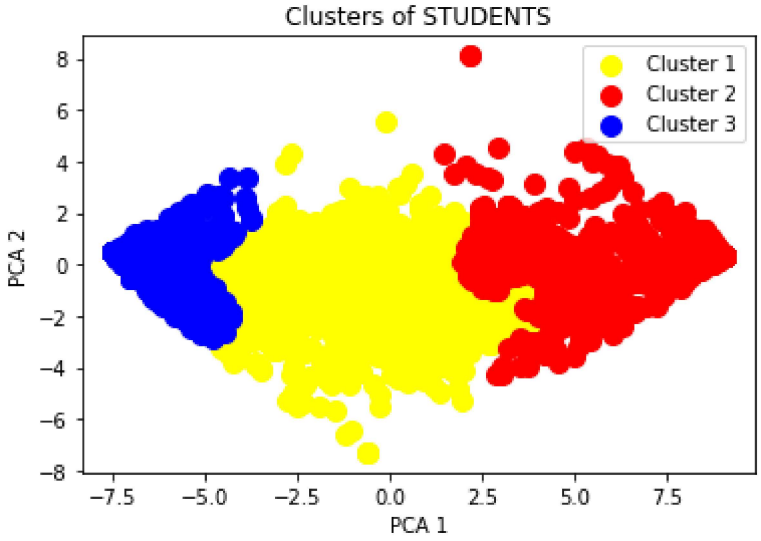
Plotting the dendrogram

In [23]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
plt.figure(figsize=(20,10))
Z = linkage(datasetpca, method='ward')
dendrogram(Z, leaf_rotation=90, p=10, truncate_mode='level', leaf_font_size=6,
color_threshold=8)
plt.title('Dendogram')
plt.show()
```



Implementation of Hierarchical clustering

In [24]:
```
from sklearn.cluster import AgglomerativeClustering
hc=AgglomerativeClustering(n_clusters=3,affinity='euclidean',linkage='ward')
ac=hc.fit_predict(datasetpca)
X=datasetpca
#visualizing the cluster
plt.scatter(X[ac == 0, 0], X[ac == 0, 1], s = 100, c = 'yellow', label = 'Clus
ter 1')
plt.scatter(X[ac == 1, 0], X[ac == 1, 1], s = 100, c = 'red', label = 'Cluster
2')
plt.scatter(X[ac == 2, 0], X[ac == 2, 1], s = 100, c = 'blue', label = 'Cluste
r 3')
plt.title('Clusters of STUDENTS')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend()
plt.show()
```



In [25]:
```
hc.labels_
```

Out[25]:
```
array([0, 0, 2, ..., 2, 1, 1], dtype=int64)
```

In [26]:
```
ac=hc.fit_predict(datasetpca)
```

Result of Hierarchical

In [27]:
```
count(ac)
```

Out[27]:
```
Counter({0: 3476, 2: 924, 1: 1420})
```

Comparing the kmeans and hierarchical clustering

In [28]:
```
first0=[2229,3476]
second1=[1229,1420]
third2=[2362,924]
clusters=['Kmeans','Agglm Cluster']
d=pd.DataFrame({'Clusters':clusters,'FirstC':first0,'SecondC':second1,'ThirdC'
:third2})
d
```

Out[28]:

	Clusters	FirstC	SecondC	ThirdC
0	Kmeans	2229	1229	2362
1	Agglm Cluster	3476	1420	924

kmeans

In [29]:
```
df=ds.copy()
df.head()
```

Out[29]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q
0	-2.06785	-1.430719	-0.402174	-1.136118	0.901862	0.052278	-0.05749	-0.142561	-0.06420
1	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.05749	-0.142561	-0.06420
2	-2.06785	-1.430719	-0.402174	0.219954	0.901862	1.543745	1.49876	1.453023	1.49283
3	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.05749	-0.142561	-0.06420
4	-2.06785	-1.430719	-0.402174	-1.136118	-1.322221	-1.439189	-1.61374	-1.738145	-1.62125

5 rows × 33 columns

In [30]:
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++')
kmeans.fit(datasetpca)
df['label'] = kmeans.labels_
df.head()
```

Out[30]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q
0	-2.06785	-1.430719	-0.402174	-1.136118	0.901862	0.052278	-0.05749	-0.142561	-0.06420
1	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.05749	-0.142561	-0.06420
2	-2.06785	-1.430719	-0.402174	0.219954	0.901862	1.543745	1.49876	1.453023	1.49283
3	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.05749	-0.142561	-0.06420
4	-2.06785	-1.430719	-0.402174	-1.136118	-1.322221	-1.439189	-1.61374	-1.738145	-1.62125

5 rows × 34 columns

In [31]:
```
df['label'].value_counts()
```

Out[31]:
```
2    2362
0    2228
1    1230
Name: label, dtype: int64
```

In [32]:
```
X=df.drop(columns='label')
y=df['label']
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3, random_st
ate=1)

print(Xtrain.shape)
print(Xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(4074, 33)
(1746, 33)
(4074,)
(1746,)
```

In [33]:
```
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(Xtrain, ytrain)
```

Out[33]:
```
LogisticRegression()
```

In [34]:
```
ypredict=lr.predict(Xtest)
accuracy=(metrics.accuracy_score(ytest,ypredict))
accuracy
```

Out[34]:
```
0.995418098510882
```

hierarchical

In [35]:
```
df2=ds.copy()
df2.head()
```

Out[35]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q
0	-2.06785	-1.430719	-0.402174	-1.136118	0.901862	0.052278	-0.05749	-0.142561	-0.06420
1	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.05749	-0.142561	-0.06420
2	-2.06785	-1.430719	-0.402174	0.219954	0.901862	1.543745	1.49876	1.453023	1.49283
3	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.05749	-0.142561	-0.06420
4	-2.06785	-1.430719	-0.402174	-1.136118	-1.322221	-1.439189	-1.61374	-1.738145	-1.62125

5 rows × 33 columns

In [36]:
```
hirar = AgglomerativeClustering(n_clusters=3, affinity='euclidean',  linkage=
'ward')
hirar.fit(df2)
```

Out[36]:
```
AgglomerativeClustering(n_clusters=3)
```

In [37]:
```
df2['label'] = hirar.labels_
df2.head()
```

Out[37]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q
0	-2.06785	-1.430719	-0.402174	-1.136118	0.901862	0.052278	-0.05749	-0.142561	-0.06420
1	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.05749	-0.142561	-0.06420
2	-2.06785	-1.430719	-0.402174	0.219954	0.901862	1.543745	1.49876	1.453023	1.49283
3	-2.06785	-1.430719	-0.402174	-0.458082	0.160501	0.052278	-0.05749	-0.142561	-0.06420
4	-2.06785	-1.430719	-0.402174	-1.136118	-1.322221	-1.439189	-1.61374	-1.738145	-1.62125

5 rows × 34 columns

In [38]:
```
X2=df2.drop(columns='label')
y2=df2['label']
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X2, y2, test_size=0.3, random_
state=1)

print(Xtrain.shape)
print(Xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(4074, 33)
(1746, 33)
(4074,)
(1746,)
```

In [39]:
```
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(Xtrain, ytrain)
```

Out[39]:
```
LogisticRegression()
```

In [40]:
```
ypredict=lr.predict(Xtest)
accuracy=(metrics.accuracy_score(ytest,ypredict))
accuracy
```

Out[40]:
```
0.9513172966781214
```

we got 99% accuracy on applying kmeans and 95% accuracy on appyling hierarchical clustering so we conclude that k-means gives the best clustering groups.