

# Assignment-08-PCA (Wine)

Perform Principal component analysis and perform clustering using first 3 principal component scores (both heirarchial and k mean clustering(scree plot or elbow curve) and obtain optimum number of clusters and check whether we have obtained same number of clusters with the original data (class column we have ignored at the begining who shows it has 3 clusters)df

In [1]:

```
# Import Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
```

In [2]:

```
# import dataset
wine=pd.read_csv("C:/Users/LENOVO/Documents/Custom Office Templates/wine.csv")
wine
```

Out[2]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	P
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	
...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	

178 rows × 14 columns



In [3]:

```
wine['Type'].value_counts()
```

Out[3]:

```
2    71
1    59
3    48
Name: Type, dtype: int64
```

In [4]:

```
wine2=wine.iloc[:,1:]
wine2
```

Out[4]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthoc
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	
...	...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	

178 rows × 13 columns

In [5]:

```
wine2.shape
```

Out[5]:

```
(178, 13)
```

In [6]:

```
wine2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   Alcohol               178 non-null    float64
 1   Malic                 178 non-null    float64
 2   Ash                  178 non-null    float64
 3   Alcalinity           178 non-null    float64
 4   Magnesium            178 non-null    int64
 5   Phenols              178 non-null    float64
 6   Flavanoids           178 non-null    float64
 7   Nonflavanoids        178 non-null    float64
 8   Proanthocyanins      178 non-null    float64
 9   Color                178 non-null    float64
10   Hue                  178 non-null    float64
11   Dilution            178 non-null    float64
12   Proline              178 non-null    int64
dtypes: float64(11), int64(2)
memory usage: 18.2 KB
```

In [7]:

```
# Converting data to numpy array
wine_ary=wine2.values
wine_ary
```

Out[7]:

```
array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
        1.185e+03],
       ...,
       [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
        8.350e+02],
       [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
        8.400e+02],
       [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
        5.600e+02]])
```

In [8]:

```
# Normalizing the numerical data
wine_norm=scale(wine_ary)
wine_norm
```

Out[8]:

```
array([[ 1.51861254, -0.5622498 ,  0.23205254, ...,  0.36217728,
         1.84791957,  1.01300893],
       [ 0.24628963, -0.49941338, -0.82799632, ...,  0.40605066,
         1.1134493 ,  0.96524152],
       [ 0.19687903,  0.02123125,  1.10933436, ...,  0.31830389,
         0.78858745,  1.39514818],
       ...,
       [ 0.33275817,  1.74474449, -0.38935541, ..., -1.61212515,
        -1.48544548,  0.28057537],
       [ 0.20923168,  0.22769377,  0.01273209, ..., -1.56825176,
        -1.40069891,  0.29649784],
       [ 1.39508604,  1.58316512,  1.36520822, ..., -1.52437837,
        -1.42894777, -0.59516041]])
```

## PCA Implementation

In [9]:

```
# Applying PCA Fit Transformation to dataset
pca=PCA(n_components=13)

wine_pca=pca.fit_transform(wine_norm)
wine_pca
```

Out[9]:

```
array([[ 3.31675081e+00, -1.44346263e+00, -1.65739045e-01, ...,
        -4.51563395e-01,  5.40810414e-01, -6.62386309e-02],
       [ 2.20946492e+00,  3.33392887e-01, -2.02645737e+00, ...,
        -1.42657306e-01,  3.88237741e-01,  3.63650247e-03],
       [ 2.51674015e+00, -1.03115130e+00,  9.82818670e-01, ...,
        -2.86672847e-01,  5.83573183e-04,  2.17165104e-02],
       ...,
       [-2.67783946e+00, -2.76089913e+00, -9.40941877e-01, ...,
         5.12492025e-01,  6.98766451e-01,  7.20776948e-02],
       [-2.38701709e+00, -2.29734668e+00, -5.50696197e-01, ...,
         2.99821968e-01,  3.39820654e-01, -2.18657605e-02],
       [-3.20875816e+00, -2.76891957e+00,  1.01391366e+00, ...,
        -2.29964331e-01, -1.88787963e-01, -3.23964720e-01]])
```

In [10]:

```
# PCA Components matrix or covariance matrix
pca.components_
```

Out[10]:

```
array([[ 0.1443294 , -0.24518758, -0.00205106, -0.23932041,  0.14199204,
         0.39466085,  0.4229343 , -0.2985331 ,  0.31342949, -0.0886167 ,
         0.29671456,  0.37616741,  0.28675223],
       [-0.48365155, -0.22493093, -0.31606881,  0.0105905 , -0.299634 ,
        -0.06503951,  0.00335981, -0.02877949, -0.03930172, -0.52999567,
         0.27923515,  0.16449619, -0.36490283],
       [-0.20738262,  0.08901289,  0.6262239 ,  0.61208035,  0.13075693,
         0.14617896,  0.1506819 ,  0.17036816,  0.14945431, -0.13730621,
         0.08522192,  0.16600459, -0.12674592],
       [-0.0178563 ,  0.53689028, -0.21417556,  0.06085941, -0.35179658,
         0.19806835,  0.15229479, -0.20330102,  0.39905653,  0.06592568,
        -0.42777141,  0.18412074, -0.23207086],
       [-0.26566365,  0.03521363, -0.14302547,  0.06610294,  0.72704851,
        -0.14931841, -0.10902584, -0.50070298,  0.13685982, -0.07643678,
        -0.17361452, -0.10116099, -0.1578688 ],
       [-0.21353865, -0.53681385, -0.15447466,  0.10082451, -0.03814394,
         0.0841223 ,  0.01892002,  0.25859401,  0.53379539,  0.41864414,
        -0.10598274, -0.26585107, -0.11972557],
       [-0.05639636,  0.42052391, -0.14917061, -0.28696914,  0.3228833 ,
        -0.02792498, -0.06068521,  0.59544729,  0.37213935, -0.22771214,
         0.23207564, -0.0447637 ,  0.0768045 ],
       [-0.39613926, -0.06582674,  0.17026002, -0.42797018,  0.15636143,
         0.40593409,  0.18724536,  0.23328465, -0.36822675,  0.03379692,
        -0.43662362,  0.07810789, -0.12002267],
       [ 0.50861912, -0.07528304, -0.30769445,  0.20044931,  0.27140257,
         0.28603452,  0.04957849,  0.19550132, -0.20914487,  0.05621752,
         0.08582839,  0.1372269 , -0.57578611],
       [ 0.21160473, -0.30907994, -0.02712539,  0.05279942,  0.06787022,
        -0.32013135, -0.16315051,  0.21553507,  0.1341839 , -0.29077518,
        -0.52239889,  0.52370587,  0.162116 ],
       [-0.22591696,  0.07648554, -0.49869142,  0.47931378,  0.07128891,
         0.30434119, -0.02569409,  0.11689586, -0.23736257,  0.0318388 ,
        -0.04821201,  0.0464233 ,  0.53926983],
       [-0.26628645,  0.12169604, -0.04962237, -0.05574287,  0.06222011,
        -0.30388245, -0.04289883,  0.04235219, -0.09555303,  0.60422163,
         0.259214 ,  0.60095872, -0.07940162],
       [ 0.01496997,  0.02596375, -0.14121803,  0.09168285,  0.05677422,
        -0.46390791,  0.83225706,  0.11403985, -0.11691707, -0.0119928 ,
        -0.08988884, -0.15671813,  0.01444734]])
```

In [11]:

```
# The amount of variance that each PCA has
var=pca.explained_variance_ratio_
var
```

Out[11]:

```
array([0.36198848, 0.1920749 , 0.11123631, 0.0706903 , 0.06563294,
        0.04935823, 0.04238679, 0.02680749, 0.02222153, 0.01930019,
        0.01736836, 0.01298233, 0.00795215])
```

In [12]:

```
# Cumulative variance of each PCA  
var1=np.cumsum(np.round(var,4)*100)  
var1
```

Out[12]:

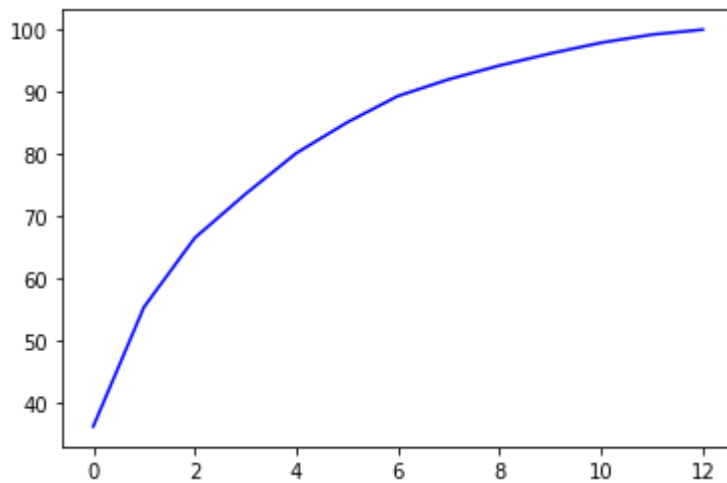
```
array([ 36.2 ,  55.41,  66.53,  73.6 ,  80.16,  85.1 ,  89.34,  92.02,  
       94.24,  96.17,  97.91,  99.21, 100.01])
```

In [13]:

```
# Variance plot for PCA components obtained  
plt.plot(var1,color='blue')
```

Out[13]:

[<matplotlib.lines.Line2D at 0xa592cff7f0>]



In [14]:

```
# Final DataFrame
final_df=pd.concat([wine['Type'],pd.DataFrame(wine_pca[:,0:3],columns=['PC1','PC2','PC3'])])
final_df
```

Out[14]:

	Type	PC1	PC2	PC3
0	1	3.316751	-1.443463	-0.165739
1	1	2.209465	0.333393	-2.026457
2	1	2.516740	-1.031151	0.982819
3	1	3.757066	-2.756372	-0.176192
4	1	1.008908	-0.869831	2.026688
...	...	...	...	...
173	3	-3.370524	-2.216289	-0.342570
174	3	-2.601956	-1.757229	0.207581
175	3	-2.677839	-2.760899	-0.940942
176	3	-2.387017	-2.297347	-0.550696
177	3	-3.208758	-2.768920	1.013914

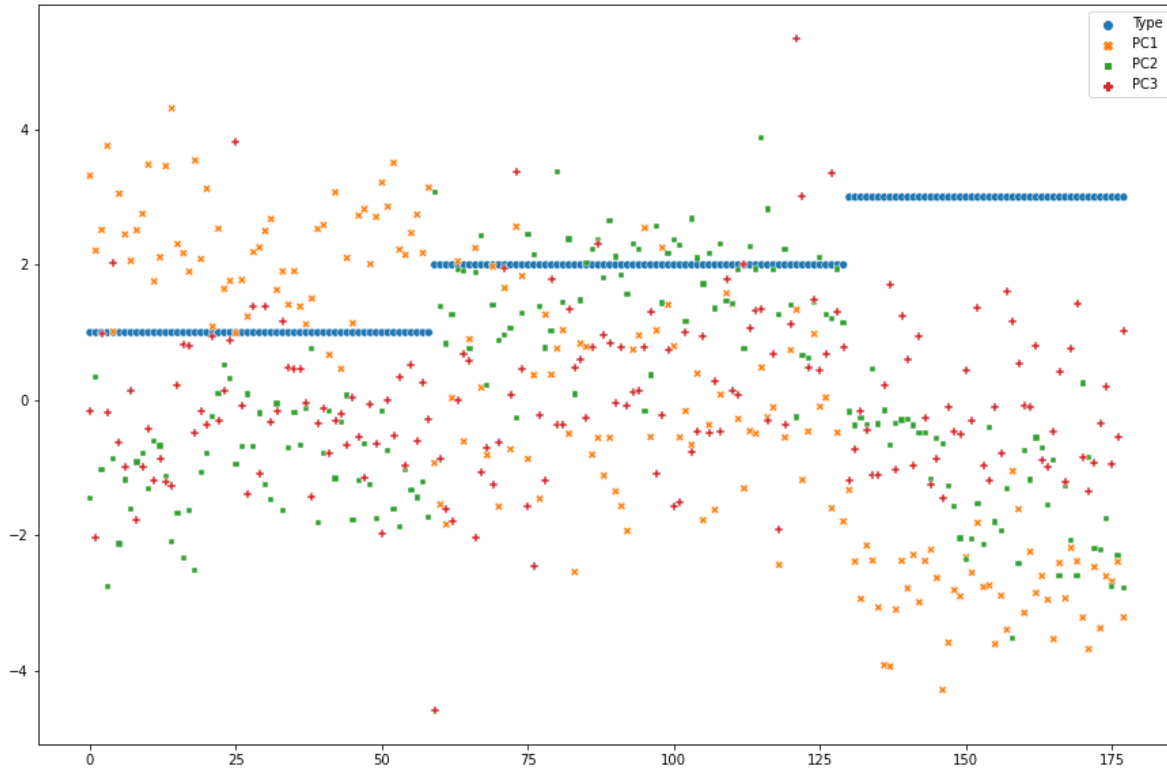
178 rows × 4 columns

In [15]:

```
# Visualization of PCAs
fig=plt.figure(figsize=(15,10))
sns.scatterplot(data=final_df)
```

Out[15]:

<AxesSubplot:>



## Checking with other Clustering Algorithms

### 1. Hierarchical Clustering

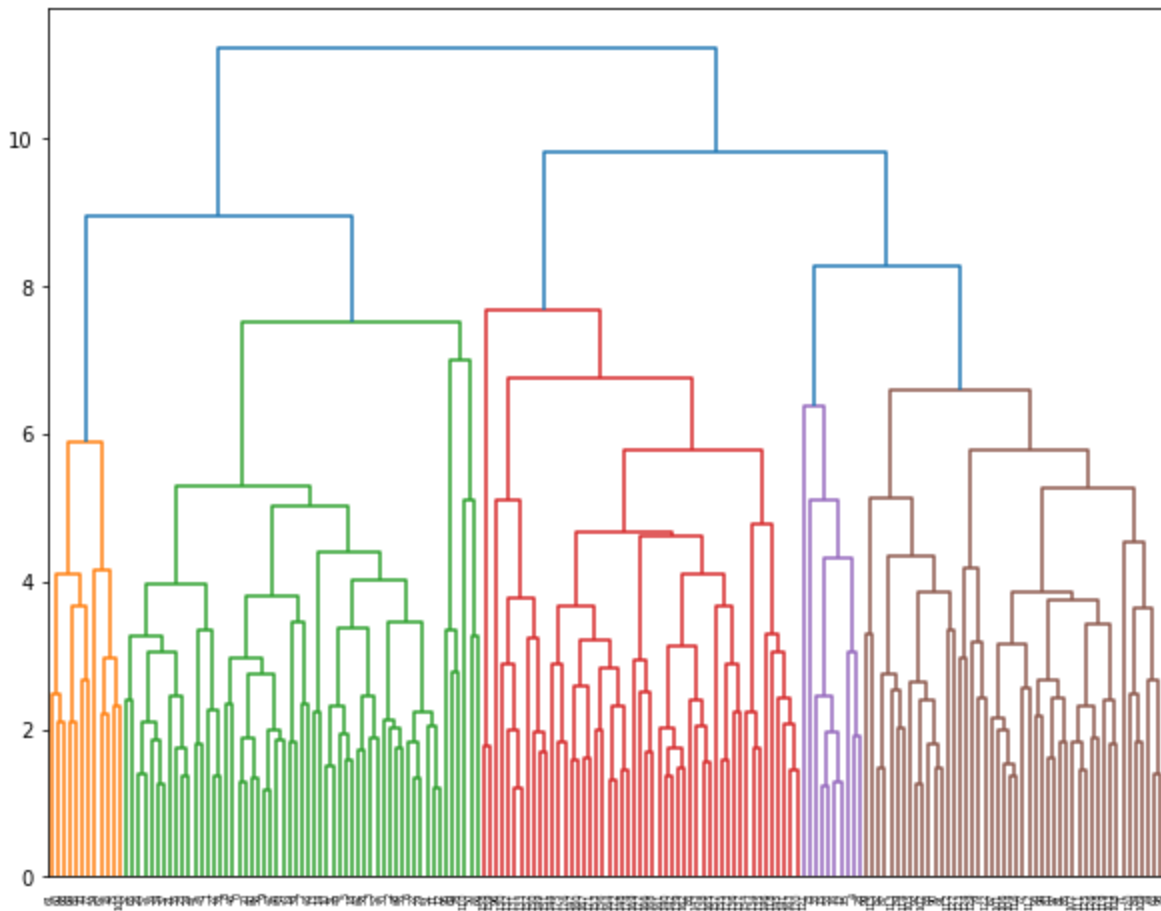
In [16]:

```
# Import Libraries
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import normalize
```



In [17]:

```
# As we already have normalized data, create Dendrograms  
plt.figure(figsize=(10,8))  
dendrogram=sch.dendrogram(sch.linkage(wine_norm,'complete'))
```



In [18]:

```
# Create Clusters (y)  
hclusters=AgglomerativeClustering(n_clusters=3,affinity='euclidean', linkage='ward')  
hclusters
```

Out[18]:

```
AgglomerativeClustering(n_clusters=3)
```

In [19]:

```
y=pd.DataFrame(hclusters.fit_predict(wine_norm),columns=['clusytersid'])
y['clusytersid'].value_counts()
```

Out[19]:

```
2    64
0    58
1    56
Name: clusytersid, dtype: int64
```

In [20]:

```
# Adding clusters to dataset
wine3=wine.copy()
wine3['clustersid']=hclusters.labels_
wine3
```

Out[20]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	P
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	
...	...	...	...	...	...	...	...	...	...	
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	

178 rows × 15 columns

## 2. K-Means Clustering

In [21]:

```
# Import Libraries
from sklearn.cluster import KMeans
```

In [22]:

```
# As we already have normalized data
# Use Elbow Graph to find optimum number of clusters (K value) from K values range
# The K-Means algorithm aims to choose centroids that minimise the inertia, or within-cluster
# random state can be anything from 0-42, but the same number to be used everytime, so that
```

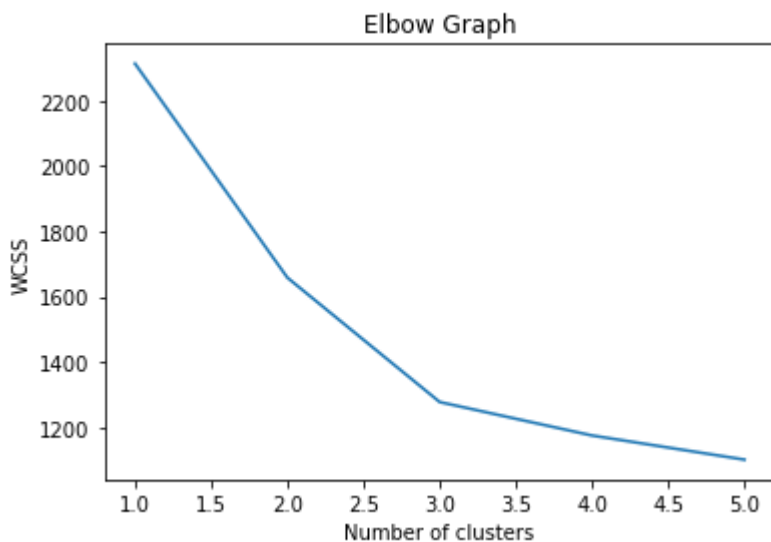
In [23]:

```
# within-cluster sum-of-squares criterion
wcss=[]
for i in range(1,6):
    kmeans=KMeans(n_clusters=i, random_state=2)
    kmeans.fit(wine_norm)
    wcss.append(kmeans.inertia_)
```

C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\cluster\\_kmeans.py:881:  
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when  
there are less chunks than available threads. You can avoid it by setting the  
environment variable OMP\_NUM\_THREADS=1.  
warnings.warn(

In [24]:

```
# Plot K values range vs WCSS to get Elbow graph for choosing K (no. of clusters)
plt.plot(range(1,6),wcss)
plt.title('Elbow Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



## Build Cluster algorithm using K=3

In [25]:

```
# cluster algorithm using K=4
clusters3=KMeans(3,random_state=30).fit(wine_norm)
clusters3
```

Out[25]:

KMeans(n\_clusters=3, random\_state=30)

In [26]:

```
clusters3.labels_
```

Out[26]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0])
```

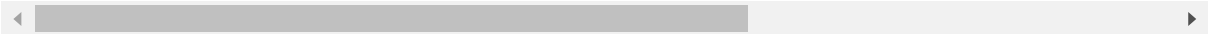
In [27]:

```
# Assign clusters to the data set
wine4=wine.copy()
wine4['clusters3id']=clusters3.labels_
wine4
```

Out[27]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	P
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	
...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	

178 rows × 15 columns



In [28]:

```
wine4['clusters3id'].value_counts()
```

Out[28]:

```
2    65
```

```
1    62
```

```
0    51
```

```
Name: clusters3id, dtype: int64
```

In [ ]: