Problem Statement:-

1) Perform Principal component analysis and perform clustering using first 3 principal component scores (both heirarchial and k mean clustering(scree plot or elbow curve).

2) 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.

1. Import Necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from mlxtend.cluster.kmeans import Kmeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
```

2. Import Data

In [2]:
```
Wine_data = pd.read_csv('wine.csv')
Wine_data
```

Out[2]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proar
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

3. Data Understanding

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

Out[3]:
```
(178, 14)
```

In [4]:
```
Wine_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Type             178 non-null    int64  
 1   Alcohol          178 non-null    float64
 2   Malic            178 non-null    float64
 3   Ash              178 non-null    float64
 4   Alcalinity       178 non-null    float64
 5   Magnesium        178 non-null    int64  
 6   Phenols          178 non-null    float64
 7   Flavanoids       178 non-null    float64
 8   Nonflavanoids    178 non-null    float64
 9   Proanthocyanins  178 non-null    float64
 10  Color            178 non-null    float64
 11  Hue              178 non-null    float64
 12  Dilution         178 non-null    float64
 13  Proline          178 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

In [5]:
```
Wine_data.dtypes
```

Out[5]:
```
Type                 int64
Alcohol            float64
Malic              float64
Ash                float64
Alcalinity         float64
Magnesium            int64
Phenols            float64
Flavanoids         float64
Nonflavanoids      float64
Proanthocyanins    float64
Color              float64
Hue                float64
Dilution           float64
Proline              int64
dtype: object
```

In [6]:
```
Wine_data.describe(include = 'all')
```

Out[6]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flava
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.0
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.02
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.9
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.34
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.2
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.1
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.8
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.0

In [7]:
```
Wine_data.describe(include = 'all').nunique()
```

Out[7]:
```
Type               6
Alcohol            8
Malic              8
Ash                8
Alcalinity         8
Magnesium          8
Phenols            8
Flavanoids         8
Nonflavanoids      8
Proanthocyanins    8
Color              8
Hue                8
Dilution           8
Proline            8
dtype: int64
```

In [8]:
```
Wine_data.isna().sum()
```

Out[8]:
```
Type               0
Alcohol            0
Malic              0
Ash                0
Alcalinity         0
Magnesium          0
Phenols            0
Flavanoids         0
Nonflavanoids      0
Proanthocyanins    0
Color              0
Hue                0
Dilution           0
Proline            0
dtype: int64
```

In [9]:
```
Wine_data.head(20)
```

Out[9]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proant
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	
5	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	
6	1	14.39	1.87	2.45	14.6	96	2.50	2.52	0.30	
7	1	14.06	2.15	2.61	17.6	121	2.60	2.51	0.31	
8	1	14.83	1.64	2.17	14.0	97	2.80	2.98	0.29	
9	1	13.86	1.35	2.27	16.0	98	2.98	3.15	0.22	
10	1	14.10	2.16	2.30	18.0	105	2.95	3.32	0.22	
11	1	14.12	1.48	2.32	16.8	95	2.20	2.43	0.26	
12	1	13.75	1.73	2.41	16.0	89	2.60	2.76	0.29	
13	1	14.75	1.73	2.39	11.4	91	3.10	3.69	0.43	
14	1	14.38	1.87	2.38	12.0	102	3.30	3.64	0.29	
15	1	13.63	1.81	2.70	17.2	112	2.85	2.91	0.30	
16	1	14.30	1.92	2.72	20.0	120	2.80	3.14	0.33	
17	1	13.83	1.57	2.62	20.0	115	2.95	3.40	0.40	
18	1	14.19	1.59	2.48	16.5	108	3.30	3.93	0.32	
19	1	13.64	3.10	2.56	15.2	116	2.70	3.03	0.17	

In [10]:
```
Wine_data['Alcohol'].unique()
```

Out[10]:
```
array([14.23, 13.2 , 13.16, 14.37, 13.24, 14.2 , 14.39, 14.06, 14.83,
       13.86, 14.1 , 14.12, 13.75, 14.75, 14.38, 13.63, 14.3 , 13.83,
       14.19, 13.64, 12.93, 13.71, 12.85, 13.5 , 13.05, 13.39, 13.3 ,
       13.87, 14.02, 13.73, 13.58, 13.68, 13.76, 13.51, 13.48, 13.28,
       13.07, 14.22, 13.56, 13.41, 13.88, 14.21, 13.9 , 13.94, 13.82,
       13.77, 13.74, 13.29, 13.72, 12.37, 12.33, 12.64, 13.67, 12.17,
       13.11, 13.34, 12.21, 12.29, 13.49, 12.99, 11.96, 11.66, 13.03,
       11.84, 12.7 , 12.  , 12.72, 12.08, 12.67, 12.16, 11.65, 11.64,
       12.69, 11.62, 12.47, 11.81, 12.6 , 12.34, 11.82, 12.51, 12.42,
       12.25, 12.22, 11.61, 11.46, 12.52, 11.76, 11.41, 11.03, 12.77,
       11.45, 11.56, 11.87, 12.07, 12.43, 11.79, 12.04, 12.86, 12.88,
       12.81, 12.53, 12.84, 13.36, 13.52, 13.62, 12.87, 13.32, 13.08,
       12.79, 13.23, 12.58, 13.17, 13.84, 12.45, 14.34, 12.36, 13.69,
       12.96, 13.78, 13.45, 12.82, 13.4 , 12.2 , 14.16, 13.27, 14.13])
```

In [11]:
```
Wine_data.nunique()
```

Out[11]:
```
Type                 3
Alcohol            126
Malic              133
Ash                 79
Alcalinity          63
Magnesium           53
Phenols             97
Flavanoids         132
Nonflavanoids       39
Proanthocyanins    101
Color              132
Hue                 78
Dilution           122
Proline            121
dtype: int64
```

4. Data Preparation

In [12]:
```
del Wine_data['Type']
```

In [13]:
```
Wine_data
```

Out[13]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocya
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 [14]:
```
#Converting data to numpy array
Wine = Wine_data.values
Wine
```

Out[14]:
```
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 [15]:
```
# Normalizing the nuemarical data
Wine_data_norm =scale(Wine_data)
Wine_data_norm
```

Out[15]:
```
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]])
```

In [16]:
```
Wine_data_1 = pd.DataFrame(Wine_data_norm)
Wine_data_1
```

Out[16]:

	0	1	2	3	4	5	6	7	
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034819	-0.659563	1.22488
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	0.733629	-0.820719	-0.54472
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	1.215533	-0.498407	2.13596
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	1.466525	-0.981875	1.03215
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.663351	0.226796	0.40140
...	...	...	...	...	...	...	...	...	.
173	0.876275	2.974543	0.305159	0.301803	-0.332922	-0.985614	-1.424900	1.274310	-0.93017
174	0.493343	1.412609	0.414820	1.052516	0.158572	-0.793334	-1.284344	0.549108	-0.31695
175	0.332758	1.744744	-0.389355	0.151661	1.422412	-1.129824	-1.344582	0.549108	-0.42207
176	0.209232	0.227694	0.012732	0.151661	1.422412	-1.033684	-1.354622	1.354888	-0.22934
177	1.395086	1.583165	1.365208	1.502943	-0.262708	-0.392751	-1.274305	1.596623	-0.42207

178 rows × 13 columns

5. Model Building / PCA Implementation

In [17]:
```
# Applying PCA fit transform to dataset
pca  = PCA(n_components=3)
Wine_pca = pca.fit_transform(Wine_data_norm)
Wine_pca
```

```
       [-1.60991228e+00, -2.40663816e+00,  5.48559697e-01],
       [-3.14313097e+00, -7.38161044e-01, -9.09987239e-02],
       [-2.24015690e+00, -1.17546529e+00, -1.01376932e-01],
       [-2.84767378e+00, -5.56043966e-01,  8.04215218e-01],
       [-2.59749706e+00, -6.97965537e-01, -8.84939521e-01],
       [-2.94929937e+00, -1.55530896e+00, -9.83400727e-01],

       [-3.53003227e+00, -8.82526796e-01, -4.66029128e-01],
       [-2.40611054e+00, -2.59235618e+00,  4.28226211e-01],
       [-2.92908473e+00, -1.27444695e+00, -1.21335827e+00],
       [-2.18141278e+00, -2.07753731e+00,  7.63782552e-01],
       [-2.38092779e+00, -2.58866743e+00,  1.41804403e+00],
       [-3.21161722e+00,  2.51249104e-01, -8.47129152e-01],
       [-3.67791872e+00, -8.47747844e-01, -1.33942023e+00],
       [-2.46555580e+00, -2.19379830e+00, -9.18780960e-01],
       [-3.37052415e+00, -2.21628914e+00, -3.42569512e-01],
       [-2.60195585e+00, -1.75722935e+00,  2.07581355e-01],
       [-2.67783946e+00, -2.76089913e+00, -9.40941877e-01],
       [-2.38701709e+00, -2.29734668e+00, -5.50696197e-01],
       [-3.20875816e+00, -2.76891957e+00,  1.01391366e+00]])
```

In [18]:
```
pca.components_
```

Out[18]:
```
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]])
```

Amount of variance for each PCA

In [19]:
```
Var = pca.explained_variance_ratio_
Var
```

Out[19]:
```
array([0.36198848, 0.1920749 , 0.11123631])
```

Commulative variance of each pca

In [20]:
```
Var1 = np.cumsum(np.round(Var, 4)*100)
Var1
```

Out[20]:
```
array([36.2 , 55.41, 66.53])
```

In [64]:
```
pca_data=pd.DataFrame(Wine_pca)
pca_data.columns=['PC1', 'PC2', 'PC3']
pca_data
```

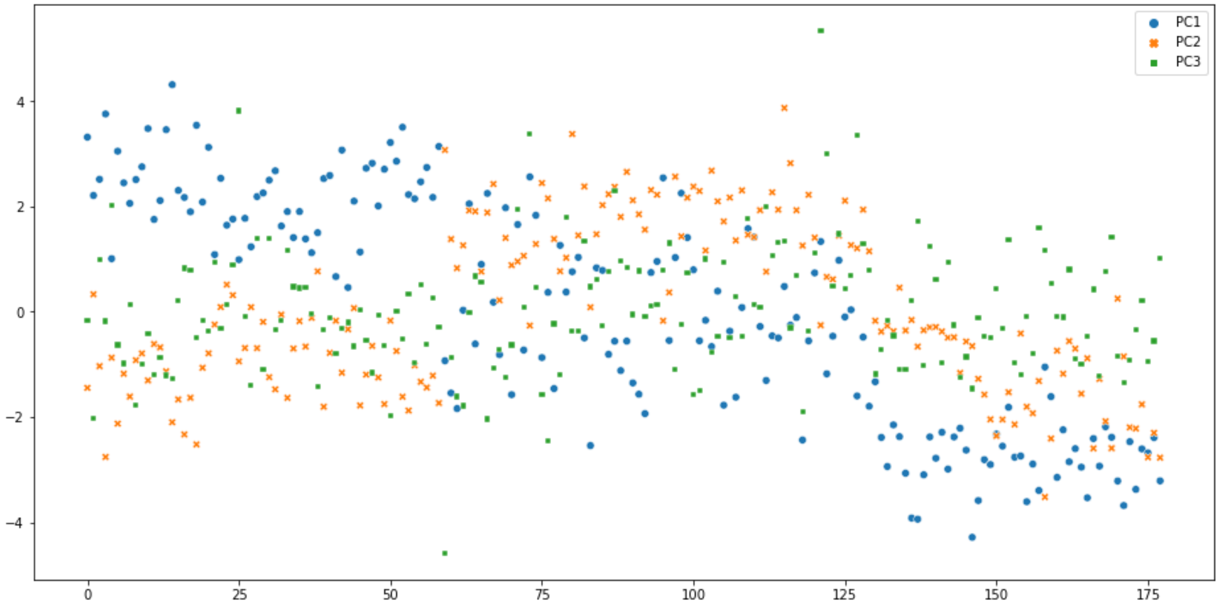
Out[64]:

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

178 rows × 3 columns

Note: - Here we got the 3 principal componets now we build the clustering through these 3 points

In [65]:
```
plt.figure(figsize=(16,8))
sns.scatterplot(data = pca_data)
plt.show()
```



Exploring Clustering Methods

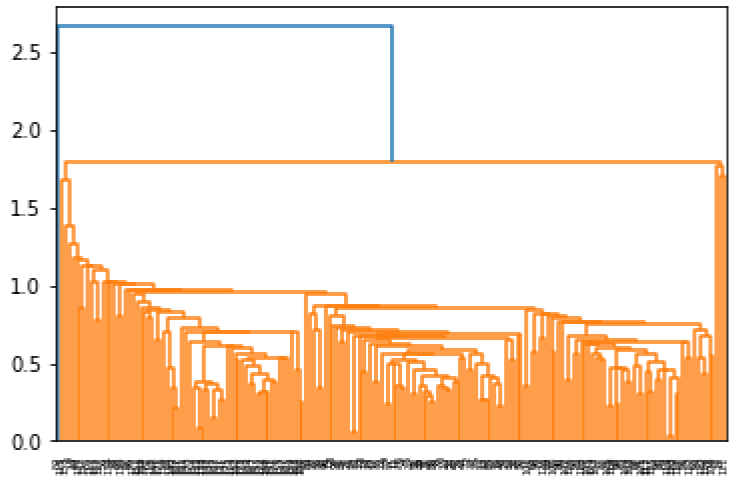
1. hierarchical Clustering, 2.KMeans Clustering, 3.DBSCAN

1. Hierarchical Clustering

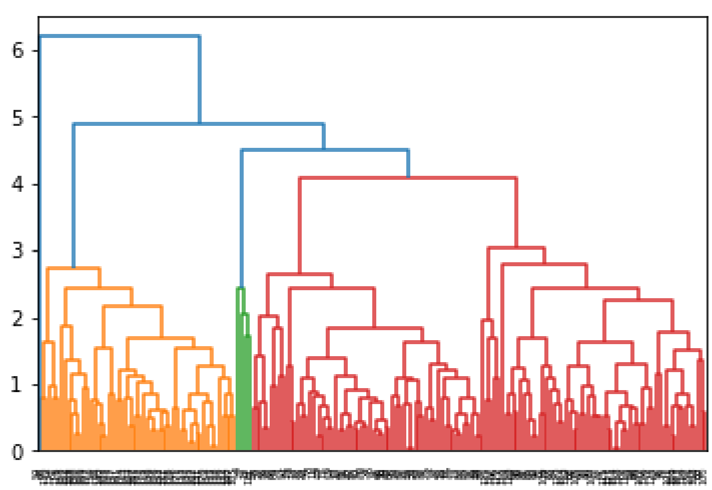
In [23]:
```
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
```

Creating Dendrogram for all the linkages

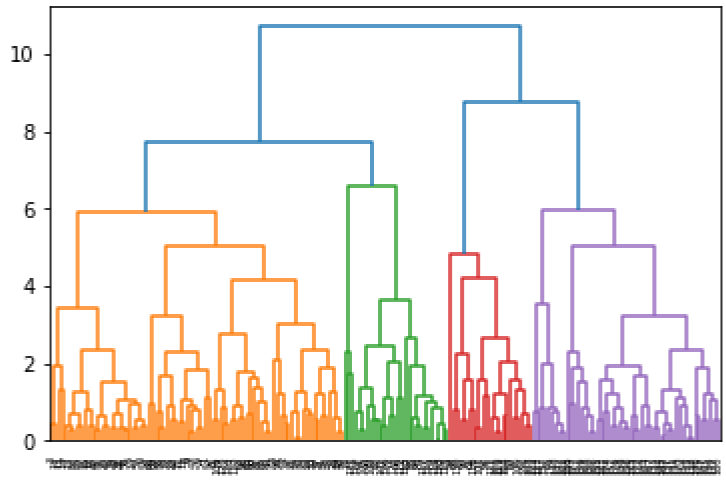
In [24]:
```
dendrogram = sch.dendrogram(sch.linkage( pca_data, method='single'))
```



In [25]:
```
dendrogram = sch.dendrogram(sch.linkage( pca_data, method='average'))
```



In [26]:
```python
dendrogram = sch.dendrogram(sch.linkage( pca_data, method='complete'))
```



In [27]:
```python
HC = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='single'
HC
```

Out[27]:
```
AgglomerativeClustering(linkage='single', n_clusters=3)
```

In [29]:
```python
#Building hierarchical clustering
y_predict = HC.fit_predict(pca_data)
y_predict
```

Out[29]:
```
array([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, 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, 1, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 0, 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, 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], dtype=int64)
```

In [38]:
```
y_predict
```

Out[38]:
```
array([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, 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, 1, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 0, 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, 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], dtype=int64)
```

In [39]:
```
clusters = pd.DataFrame(y_Hc, columns = ['clusters'])
```

In [40]:
```
clusters
```

Out[40]:

	clusters
0	2
1	2
2	2
3	2
4	2
...	...
173	2
174	2
175	2
176	2
177	2

178 rows × 1 columns

In [41]:
```
Wine_data['clusters']=clusters
```

In [42]:
```
Wine_data
```

Out[42]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proantho
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	

1. Hierarchical Clustering with cluster '0'

In [43]:
```
Wine_data[Wine_data['clusters']==0]
```

Out[43]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocya
25	13.05	2.05	3.22	25.0	124	2.63	2.68	0.47	
73	12.99	1.67	2.60	30.0	139	3.30	2.89	0.21	
121	11.56	2.05	3.23	28.5	119	3.18	5.08	0.47	

2. Hierarchical Clustering with cluster '1'

In [44]:
```
Wine_data[Wine_data['clusters']==1]
```

Out[44]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyan
59	12.37	0.94	1.36	10.6	88	1.98	0.57	0.28	0

3. Hierarchical Clustering with cluster '2'

In [45]:
```
Wine_data[Wine_data['clusters']==2]
```

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

174 rows × 14 columns

2. KMeans Clustering

In [48]:
```
from sklearn.cluster import KMeans
```

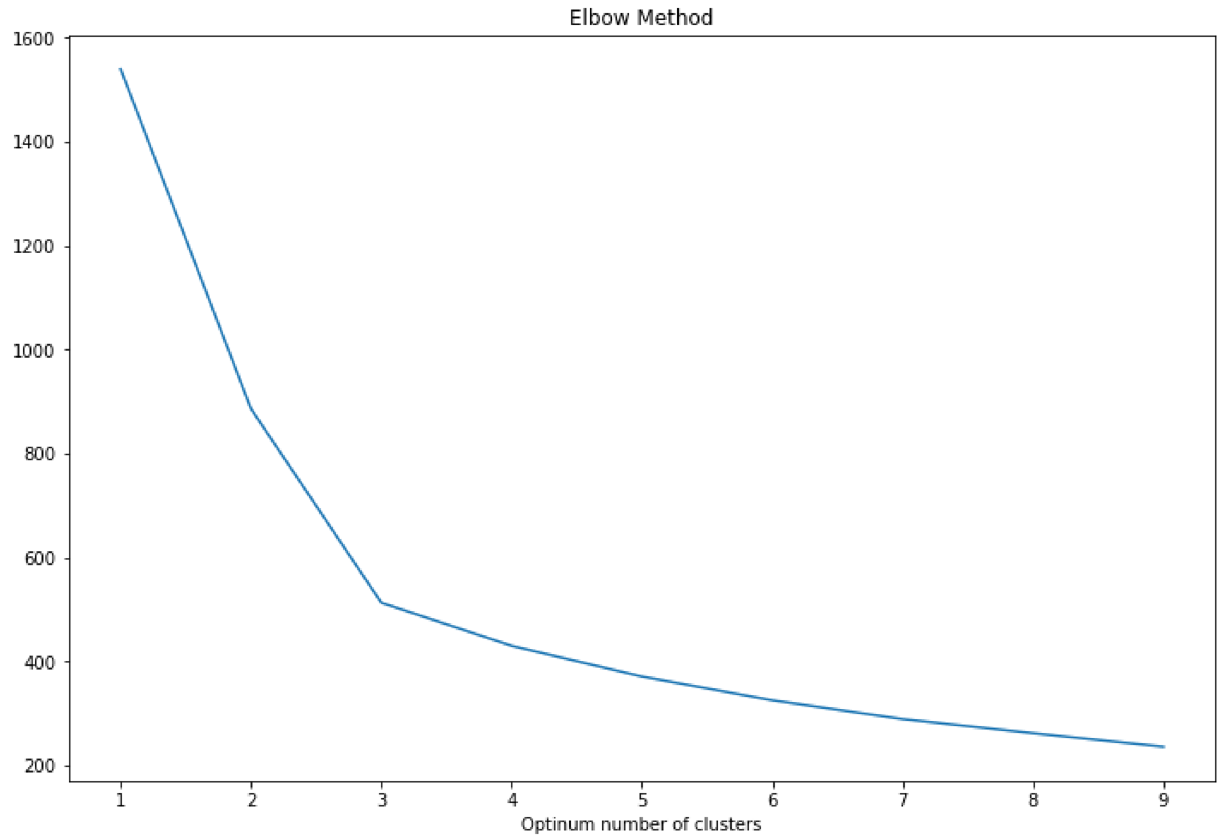
In [54]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [55]:
```
WCSS = []
for i in range(1,10):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(pca_data)
    WCSS.append(kmeans.inertia_)
    print(i, WCSS)
```

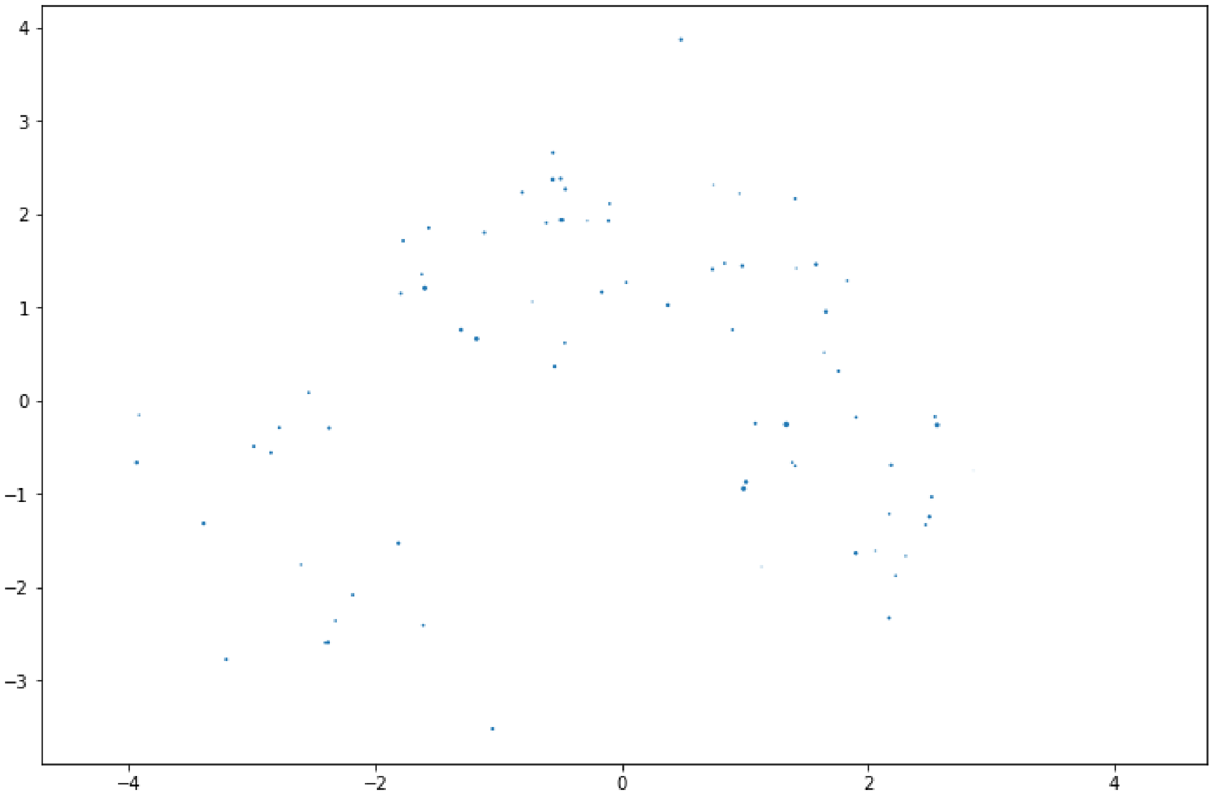
```
1 [1539.503480188306]
2 [1539.503480188306, 886.1611364823497]
3 [1539.503480188306, 886.1611364823497, 512.9995067661513]
4 [1539.503480188306, 886.1611364823497, 512.9995067661513, 430.0342519830439]
5 [1539.503480188306, 886.1611364823497, 512.9995067661513, 430.0342519830439,
370.88883220672295]
6 [1539.503480188306, 886.1611364823497, 512.9995067661513, 430.0342519830439,
370.88883220672295, 325.19703481069564]
7 [1539.503480188306, 886.1611364823497, 512.9995067661513, 430.0342519830439,
370.88883220672295, 325.19703481069564, 289.18443839814023]
8 [1539.503480188306, 886.1611364823497, 512.9995067661513, 430.0342519830439,
370.88883220672295, 325.19703481069564, 289.18443839814023, 262.0102072749322]
9 [1539.503480188306, 886.1611364823497, 512.9995067661513, 430.0342519830439,
370.88883220672295, 325.19703481069564, 289.18443839814023, 262.0102072749322,
236.08731811259065]
```

Exploration with Elbow method

In [59]:
```
plt.figure(figsize=(12,8))
plt.plot(range(1,10), WCSS)
plt.title('Elbow Method')
plt.xlabel('Optinum number of clusters')
plt.show()
```



In [70]:
```
plt.figure(figsize=(12,8))
plt.scatter(pca_data['PC1'], pca_data['PC2'], pca_data['PC3'], cmap=plt.cm.Accent
plt.show()
```



Optimum number of k is 3

Model Building

In [71]:
```
K_Means = KMeans( n_clusters=3, algorithm='auto',  max_iter=500)
K_Means
```

Out[71]:
```
KMeans(max_iter=500, n_clusters=3)
```

In [72]:
```
K_Means.fit(pca_data)
```

Out[72]:
```
KMeans(max_iter=500, n_clusters=3)
```

In [74]:
```
clusters = K_Means.predict(pca_data)
clusters
```

Out[74]:
```
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, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 1, 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])
```

In [75]:
```
pca_data['cluster']=clusters
```

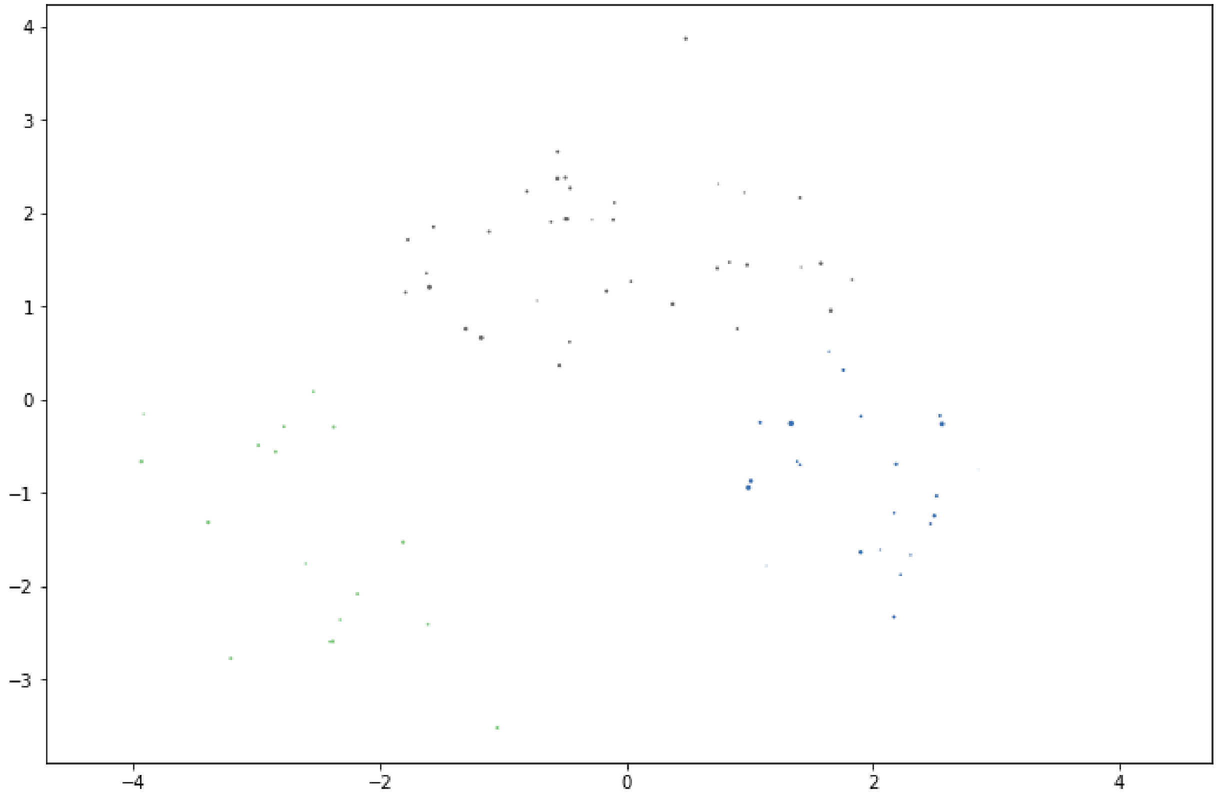
In [77]:
```
pca_data
```

Out[77]:

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

178 rows × 4 columns

In [83]:
```
plt.figure(figsize=(12,8))
plt.scatter(pca_data['PC1'], pca_data['PC2'], pca_data['PC3'], c=pca_data['cluste
plt.show()
```



1. KMeans Clustering with clusters '0'

In [84]:
```
pca_data[pca_data['cluster']==0]
```

Out[84]:

	PC1	PC2	PC3	cluster
61	-1.836250	0.829984	-1.605702	0
83	-2.538977	0.087443	0.474251	0
118	-2.433013	1.257141	-1.903027	0
130	-1.327102	-0.170389	-1.180013	0
131	-2.384501	-0.374583	-0.723823	0
132	-2.936940	-0.263862	-0.167640	0
133	-2.146811	-0.368255	-0.453301	0
134	-2.369869	0.459635	-1.101400	0
135	-3.063842	-0.353413	-1.099124	0
136	-3.915754	-0.154583	0.221828	0
137	-3.936463	-0.659687	1.712215	0
138	-3.094276	-0.348843	-1.026831	0
139	-2.374472	-0.291980	1.241914	0
140	-2.778813	-0.286805	0.609670	0
141	-2.286561	-0.372508	-0.971643	0
142	-2.985633	-0.489218	0.946953	0
143	-2.375195	-0.482334	-0.252884	0
144	-2.209866	-1.160053	-1.245125	0
145	-2.625621	-0.563161	-0.855961	0
146	-4.280639	-0.649671	-1.458197	0
147	-3.582641	-1.272703	-0.110784	0
148	-2.807064	-1.570534	-0.472528	0
149	-2.899659	-2.041057	-0.495960	0
150	-2.320737	-2.356366	0.437682	0
151	-2.549831	-2.045283	-0.312268	0
152	-1.812541	-1.527646	1.362590	0
153	-2.760145	-2.138932	-0.964629	0
154	-2.737151	-0.409886	-1.190405	0
155	-3.604869	-1.802384	-0.094037	0
156	-2.889826	-1.925219	-0.782323	0
157	-3.392156	-1.311876	1.602026	0
158	-1.048182	-3.515090	1.160039	0
159	-1.609912	-2.406638	0.548560	0
160	-3.143131	-0.738161	-0.090999	0

	PC1	PC2	PC3	cluster
161	-2.240157	-1.175465	-0.101377	0
162	-2.847674	-0.556044	0.804215	0
163	-2.597497	-0.697966	-0.884940	0
164	-2.949299	-1.555309	-0.983401	0
165	-3.530032	-0.882527	-0.466029	0
166	-2.406111	-2.592356	0.428226	0
167	-2.929085	-1.274447	-1.213358	0
168	-2.181413	-2.077537	0.763783	0
169	-2.380928	-2.588667	1.418044	0
170	-3.211617	0.251249	-0.847129	0
171	-3.677919	-0.847748	-1.339420	0
172	-2.465556	-2.193798	-0.918781	0
173	-3.370524	-2.216289	-0.342570	0
174	-2.601956	-1.757229	0.207581	0
175	-2.677839	-2.760899	-0.940942	0
176	-2.387017	-2.297347	-0.550696	0
177	-3.208758	-2.768920	1.013914	0

2. KMeans Clustering with clusters '1'

In [85]:
```
pca_data[pca_data['cluster']==1]
```

Out[85]:

	PC1	PC2	PC3	cluster
0	3.316751	-1.443463	-0.165739	1
1	2.209465	0.333393	-2.026457	1
2	2.516740	-1.031151	0.982819	1
3	3.757066	-2.756372	-0.176192	1
4	1.008908	-0.869831	2.026688	1
...	...	...	...	...
57	2.173741	-1.212200	0.261780	1
58	3.139380	-1.731579	-0.285661	1
73	2.562227	-0.260199	3.374394	1
95	2.543865	-0.169274	0.788697	1
121	1.336322	-0.253337	5.345388	1

62 rows × 4 columns

3. KMeans Clustering with clusters '2'

In [86]:
```
pca_data[pca_data['cluster']==2]
```

Out[86]:

	PC1	PC2	PC3	cluster
59	-0.928582	3.073486	-4.585064	2
60	-1.542480	1.381444	-0.874683	2
62	0.030607	1.262786	-1.784408	2
63	2.050262	1.925033	-0.007369	2
64	-0.609681	1.908059	0.679358	2
...	...	...	...	...
125	-0.096810	2.109998	0.434826	2
126	0.038487	1.266762	0.687578	2
127	-1.597159	1.208144	3.361176	2
128	-0.479565	1.938841	1.296508	2
129	-1.792833	1.150288	0.782800	2

65 rows × 4 columns

4.KMeans Clustering with clusters '3'

In [87]:
```
pca_data[pca_data['cluster']==3]
```

Out[87]:

PC1	PC2	PC3	cluster

Checking whether we have obtained same number of clusters with the original data

In [88]:
```
Wine_data_norm
```

Out[88]:
```
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]])
```

In [89]:
```
norm_df = pd.DataFrame(Wine_data_norm)
norm_df
```

Out[89]:

	0	1	2	3	4	5	6	7	
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034819	-0.659563	1.22488
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	0.733629	-0.820719	-0.54472
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	1.215533	-0.498407	2.13596
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	1.466525	-0.981875	1.03215
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.663351	0.226796	0.40140
...	...	...	...	...	...	...	...	...	.
173	0.876275	2.974543	0.305159	0.301803	-0.332922	-0.985614	-1.424900	1.274310	-0.93017
174	0.493343	1.412609	0.414820	1.052516	0.158572	-0.793334	-1.284344	0.549108	-0.31695
175	0.332758	1.744744	-0.389355	0.151661	1.422412	-1.129824	-1.344582	0.549108	-0.42207
176	0.209232	0.227694	0.012732	0.151661	1.422412	-1.033684	-1.354622	1.354888	-0.22934
177	1.395086	1.583165	1.365208	1.502943	-0.262708	-0.392751	-1.274305	1.596623	-0.42207

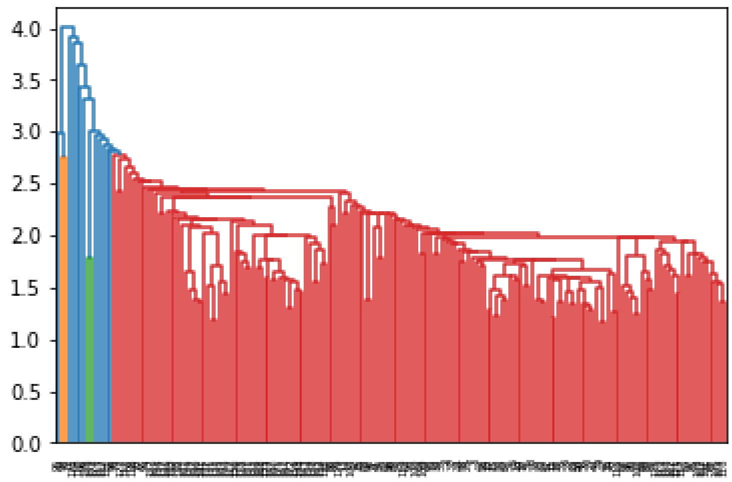
178 rows × 13 columns

1. Checking with original data using Hierarchical clustering

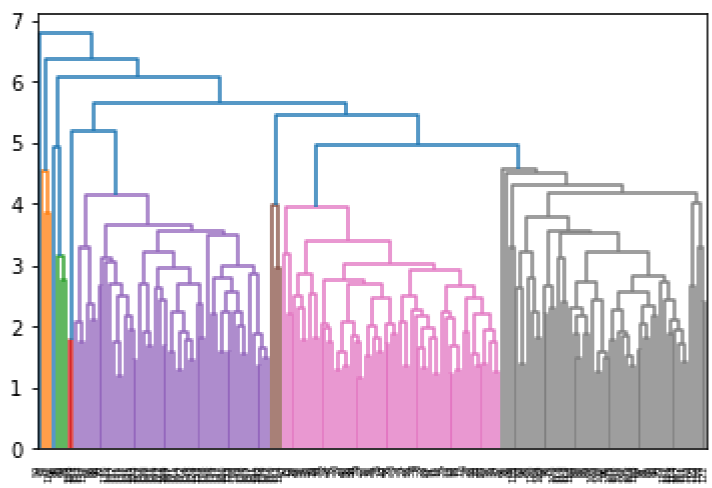
In [90]:
```
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
```

Creating Dendrogram for all the linkages

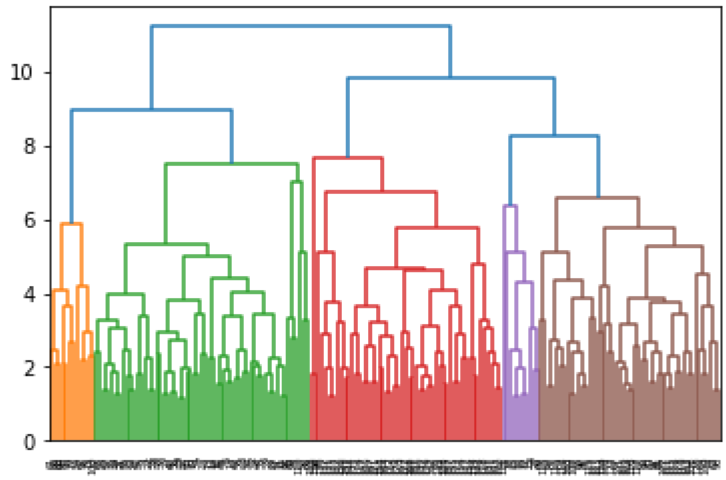
In [91]:
```
dendrogram = sch.dendrogram(sch.linkage( norm_df, method='single'))
```



In [92]:
```
dendrogram = sch.dendrogram(sch.linkage(norm_df, method='average'))
```



In [93]:
```
dendrogram = sch.dendrogram(sch.linkage( norm_df, method='complete'))
```



In [94]:
```
HC = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='single'
HC
```

Out[94]:
```
AgglomerativeClustering(linkage='single', n_clusters=3)
```

In [96]:
```
#Building hierarchical clustering
y_predict = HC.fit_predict(norm_df)
y_predict
```

Out[96]:
```
array([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, 0,
       0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

In [98]:
```
y_predict
```

Out[98]:
```
array([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, 0,
       0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

In [100]:
```
clusters = pd.DataFrame(y_Hc, columns = ['clusters'])
```

In [102]:
```
clusters
```

Out[102]:

	clusters
0	2
1	2
2	2
3	2
4	2
...	...
173	2
174	2
175	2
176	2
177	2

178 rows × 1 columns

In [104]:
```
Wine_data['clusters_original_data']=clusters
```

In [105]:
```
Wine_data
```

Out[105]:

m	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline	clusters	clu
27	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065	2	
)0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	2	
)1	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	2	
13	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	2	
18	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	2	
...	...	...	...	...	...	...	...	...	...	
)5	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740	2	
)2	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750	2	
20	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835	2	
20	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840	2	
)6	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560	2	

1. Hierarchical Clustering with cluster '0'

In [107]:
```
Wine_data[Wine_data['clusters_original_data']==0]
```

Out[107]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocya
25	13.05	2.05	3.22	25.0	124	2.63	2.68	0.47	
73	12.99	1.67	2.60	30.0	139	3.30	2.89	0.21	
121	11.56	2.05	3.23	28.5	119	3.18	5.08	0.47	

2. Hierarchical Clustering with cluster '1'

In [108]:
```
Wine_data[Wine_data['clusters_original_data']==1]
```

Out[108]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyan
59	12.37	0.94	1.36	10.6	88	1.98	0.57	0.28	(

3. Hierarchical Clustering with cluster '2'

In [109]:
```
Wine_data[Wine_data['clusters_original_data']==2]
```

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

174 rows × 15 columns

2. KMeans Clustering

In [ ]:
```
```

2. Checking with original data using KMeans Clustering

In [110]:
```
from sklearn.cluster import KMeans
```

In [111]:
```
import warnings
warnings.filterwarnings('ignore')
```

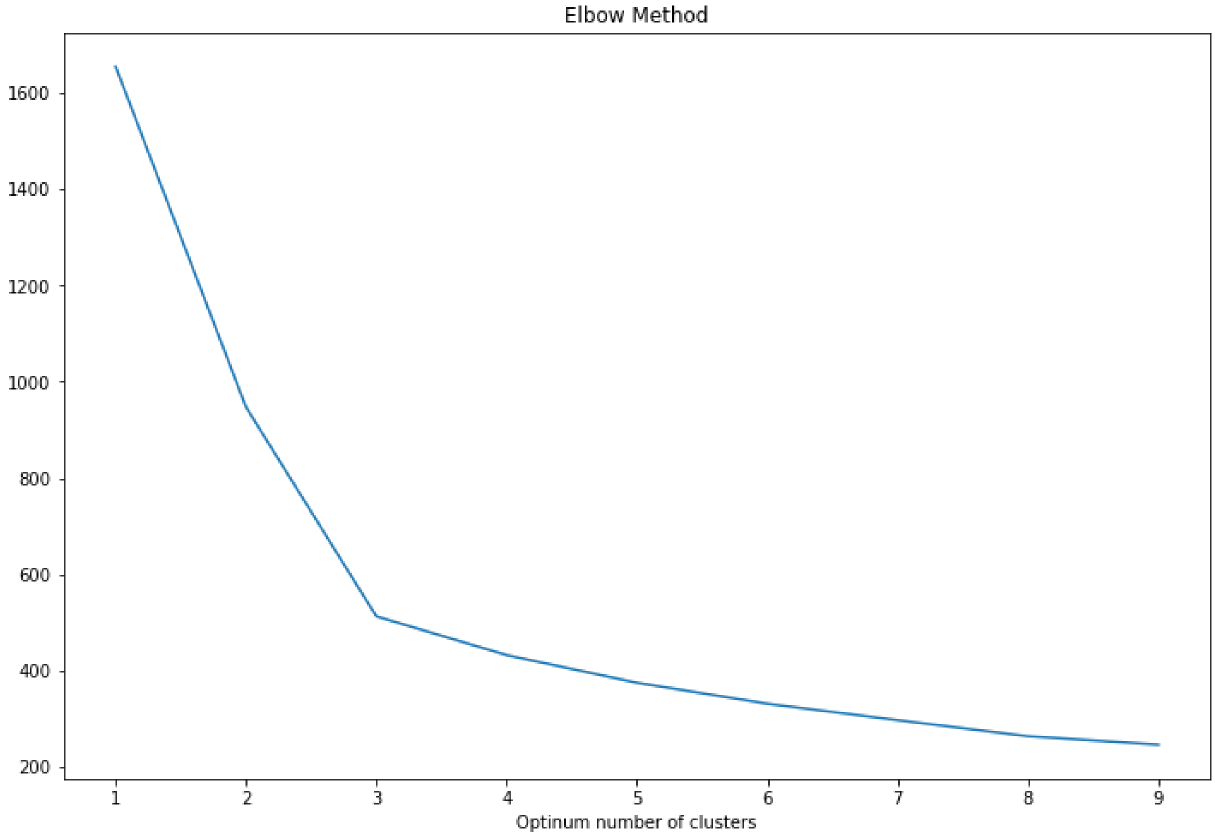
In [112]:
```
WCSS = []
for i in range(1,10):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(pca_data)
    WCSS.append(kmeans.inertia_)
    print(i, WCSS)
```

```
1 [1654.4023565928]
2 [1654.4023565928, 947.7236386265981]
3 [1654.4023565928, 947.7236386265981, 513.0564645910765]
4 [1654.4023565928, 947.7236386265981, 513.0564645910765, 432.48396554058843]
5 [1654.4023565928, 947.7236386265981, 513.0564645910765, 432.48396554058843, 3
75.0055350382643]
6 [1654.4023565928, 947.7236386265981, 513.0564645910765, 432.48396554058843, 3
75.0055350382643, 331.5691363865433]
7 [1654.4023565928, 947.7236386265981, 513.0564645910765, 432.48396554058843, 3
75.0055350382643, 331.5691363865433, 297.4707312377495]
8 [1654.4023565928, 947.7236386265981, 513.0564645910765, 432.48396554058843, 3
75.0055350382643, 331.5691363865433, 297.4707312377495, 264.1165550269689]
9 [1654.4023565928, 947.7236386265981, 513.0564645910765, 432.48396554058843, 3
75.0055350382643, 331.5691363865433, 297.4707312377495, 264.1165550269689, 246.
6750090775065]
```

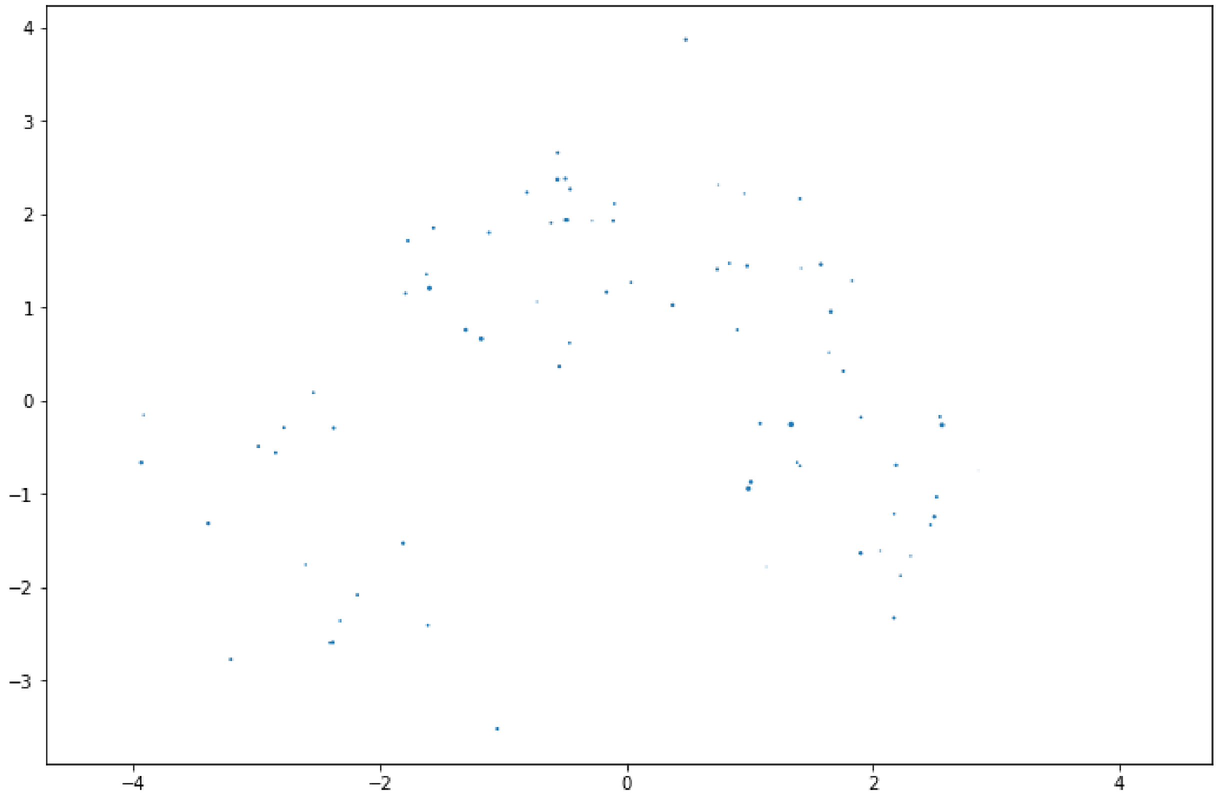
Exploration with Elbow method

In [113]:

```
plt.figure(figsize=(12,8))
plt.plot(range(1,10), WCSS)
plt.title('Elbow Method')
plt.xlabel('Optinum number of clusters')
plt.show()
```



In [114]:
```
plt.figure(figsize=(12,8))
plt.scatter(pca_data['PC1'], pca_data['PC2'], pca_data['PC3'], cmap=plt.cm.Accent
plt.show()
```



Optimum number of k is 3

Model Building

In [115]:
```
K_Means = KMeans( n_clusters=3, algorithm='auto',  max_iter=500)
K_Means
```

Out[115]: KMeans(max_iter=500, n_clusters=3)

In [116]:
```
K_Means.fit(norm_df)
```

Out[116]: KMeans(max_iter=500, n_clusters=3)

In [118]:
```
clusters = K_Means.predict(norm_df)
clusters
```

Out[118]:
```
array([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, 1, 1, 2, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 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])
```

In [126]:
```
pca_data['cluster_original_data_1']=clusters
```

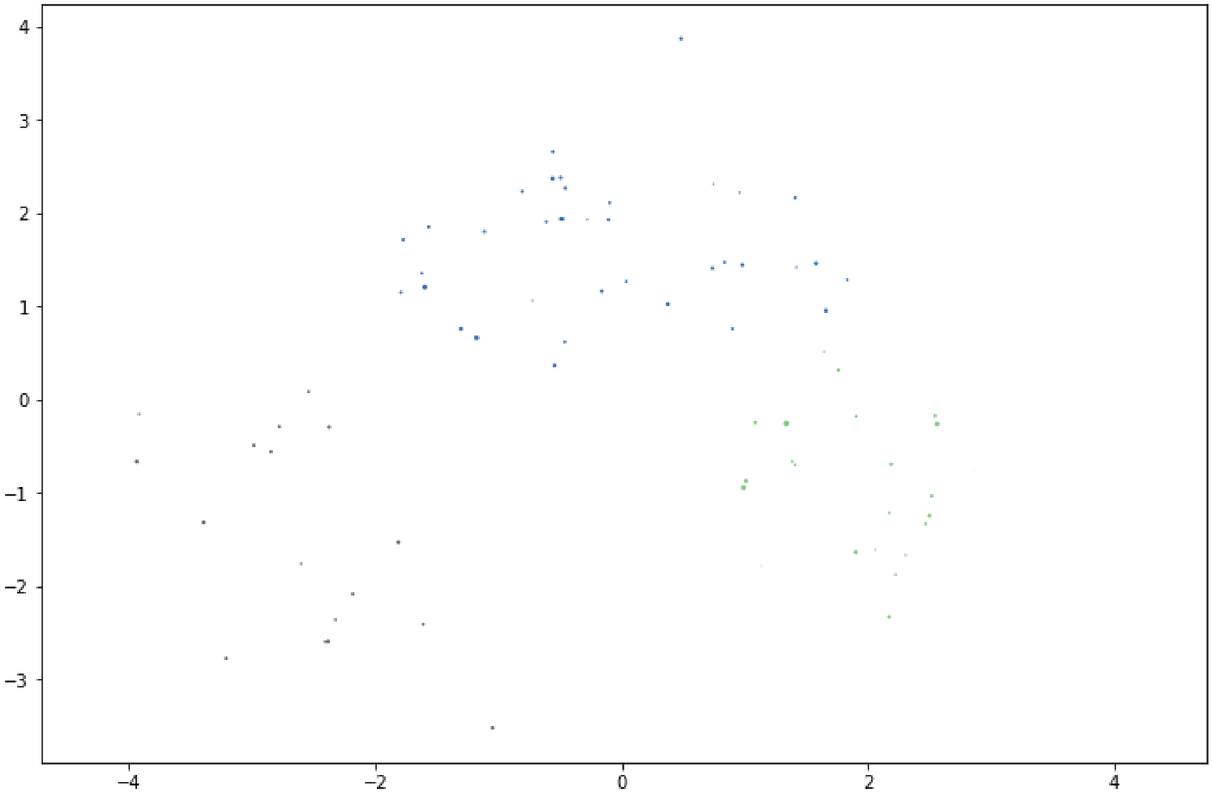
In [127]:
```
pca_data
```

Out[127]:

	PC1	PC2	PC3	cluster	cluster_original_data	cluster_original_data_1
0	3.316751	-1.443463	-0.165739	1	0	0
1	2.209465	0.333393	-2.026457	1	0	0
2	2.516740	-1.031151	0.982819	1	0	0
3	3.757066	-2.756372	-0.176192	1	0	0
4	1.008908	-0.869831	2.026688	1	0	0
...	...	...	...	...	...	...
173	-3.370524	-2.216289	-0.342570	0	2	2
174	-2.601956	-1.757229	0.207581	0	2	2
175	-2.677839	-2.760899	-0.940942	0	2	2
176	-2.387017	-2.297347	-0.550696	0	2	2
177	-3.208758	-2.768920	1.013914	0	2	2

178 rows × 6 columns

In [128]:
```
plt.figure(figsize=(12,8))
plt.scatter(pca_data['PC1'], pca_data['PC2'], pca_data['PC3'], c=pca_data['cluste
plt.show()
```



1. KMeans Clustering with clusters '0'

In [129]:
```
pca_data[pca_data['cluster_original_data_1']==0]
```

Out[129]:

	PC1	PC2	PC3	cluster	cluster_original_data	cluster_original_data_1
0	3.316751	-1.443463	-0.165739	1	0	0
1	2.209465	0.333393	-2.026457	1	0	0
2	2.516740	-1.031151	0.982819	1	0	0
3	3.757066	-2.756372	-0.176192	1	0	0
4	1.008908	-0.869831	2.026688	1	0	0
...	...	...	...	...	...	...
57	2.173741	-1.212200	0.261780	1	0	0
58	3.139380	-1.731579	-0.285661	1	0	0
73	2.562227	-0.260199	3.374394	1	0	0
95	2.543865	-0.169274	0.788697	1	0	0
121	1.336322	-0.253337	5.345388	1	0	0

62 rows × 6 columns

2. KMeans Clustering with clusters '1'

In [130]:
```
pca_data[pca_data['cluster_original_data_1']==1]
```

Out[130]:

	PC1	PC2	PC3	cluster	cluster_original_data	cluster_original_data_1
59	-0.928582	3.073486	-4.585064	2	1	1
60	-1.542480	1.381444	-0.874683	2	1	1
62	0.030607	1.262786	-1.784408	2	1	1
63	2.050262	1.925033	-0.007369	2	1	1
64	-0.609681	1.908059	0.679358	2	1	1
...	...	...	...	...	...	...
125	-0.096810	2.109998	0.434826	2	1	1
126	0.038487	1.266762	0.687578	2	1	1
127	-1.597159	1.208144	3.361176	2	1	1
128	-0.479565	1.938841	1.296508	2	1	1
129	-1.792833	1.150288	0.782800	2	1	1

65 rows × 6 columns

3. KMeans Clustering with clusters '2'

In [131]:
```
pca_data[pca_data['cluster_original_data_1']==2]
```

165	-3.530032	-0.882527	-0.466029	0	2	2
166	-2.406111	-2.592356	0.428226	0	2	2
167	-2.929085	-1.274447	-1.213358	0	2	2
168	-2.181413	-2.077537	0.763783	0	2	2
169	-2.380928	-2.588667	1.418044	0	2	2
170	-3.211617	0.251249	-0.847129	0	2	2
171	-3.677919	-0.847748	-1.339420	0	2	2
172	-2.465556	-2.193798	-0.918781	0	2	2
173	-3.370524	-2.216289	-0.342570	0	2	2
174	-2.601956	-1.757229	0.207581	0	2	2
175	-2.677839	-2.760899	-0.940942	0	2	2
176	-2.387017	-2.297347	-0.550696	0	2	2
177	-3.208758	-2.768920	1.013914	0	2	2

4.KMeans Clustering with clusters '3'

In [125]:
```
pca_data[pca_data['cluster_original_data']==3]
```

Out[125]:

PC1	PC2	PC3	cluster	cluster_original_data

Checking same number of clusters

Hierarchical Clustering

In [136]:
```
Wine_data['clusters'].value_counts()
```

Out[136]:
```
2    174
0      3
1      1
Name: clusters, dtype: int64
```

In [133]:
```
pca_data['cluster_original_data'].value_counts()
```

Out[133]:
```
1    65
0    62
2    51
Name: cluster_original_data, dtype: int64
```

KMeans Clustering

In [134]:
```
pca_data['cluster_original_data_1'].value_counts()
```

Out[134]:
```
1    65
0    62
2    51
Name: cluster_original_data_1, dtype: int64
```

In [137]:
```
pca_data['cluster'].value_counts()
```

Out[137]:
```
2    65
1    62
0    51
Name: cluster, dtype: int64
```

Inference :-

-In hierarchical clustering didn't get same number of clusters with original data

-**In KMeans clustering got same number of clusters with original data**

THE END