Principal Component Analysis

1. Standardize the data.
2. Use the standardised data to create a covariance matrix.
3. Use the resulting matrix to calculate eigen vectots(pc) and their coreesponding eigen values.
4. sort the components in desending orders by its eigen value.
5. Choose n_components which explain the most variance within the data.
6. Create the new matrix using the new components.

In [1]:

```python
# import the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```python
data = pd.read_csv("Wine.csv")
data
```

Out[2]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows × 14 columns

EDA

In [3]:

```python
data.info()   # no null values
```

```
<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 [4]:

```python
data.describe().round(2).style.background_gradient(cmap = 'Oranges')
```

Out[4]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.0

mean	1.940000	13.000000	2.340000	2.370000	19.490000	99.740000	2.300000	2.030000	0.360000	1.590000	5.0
std	0.780000	0.810000	1.120000	0.270000	3.340000	14.280000	0.630000	1.000000	0.120000	0.570000	2.3
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.2
25%	1.000000	12.360000	1.600000	2.210000	17.200000	88.000000	1.740000	1.200000	0.270000	1.250000	3.2
50%	2.000000	13.050000	1.870000	2.360000	19.500000	98.000000	2.360000	2.130000	0.340000	1.560000	4.6
75%	3.000000	13.680000	3.080000	2.560000	21.500000	107.000000	2.800000	2.880000	0.440000	1.950000	6.2
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.0

In [5]:

```
data.duplicated()
```

Out[5]:

```
0      False
1      False
2      False
3      False
4      False
       ...
173    False
174    False
175    False
176    False
177    False
Length: 178, dtype: bool
```

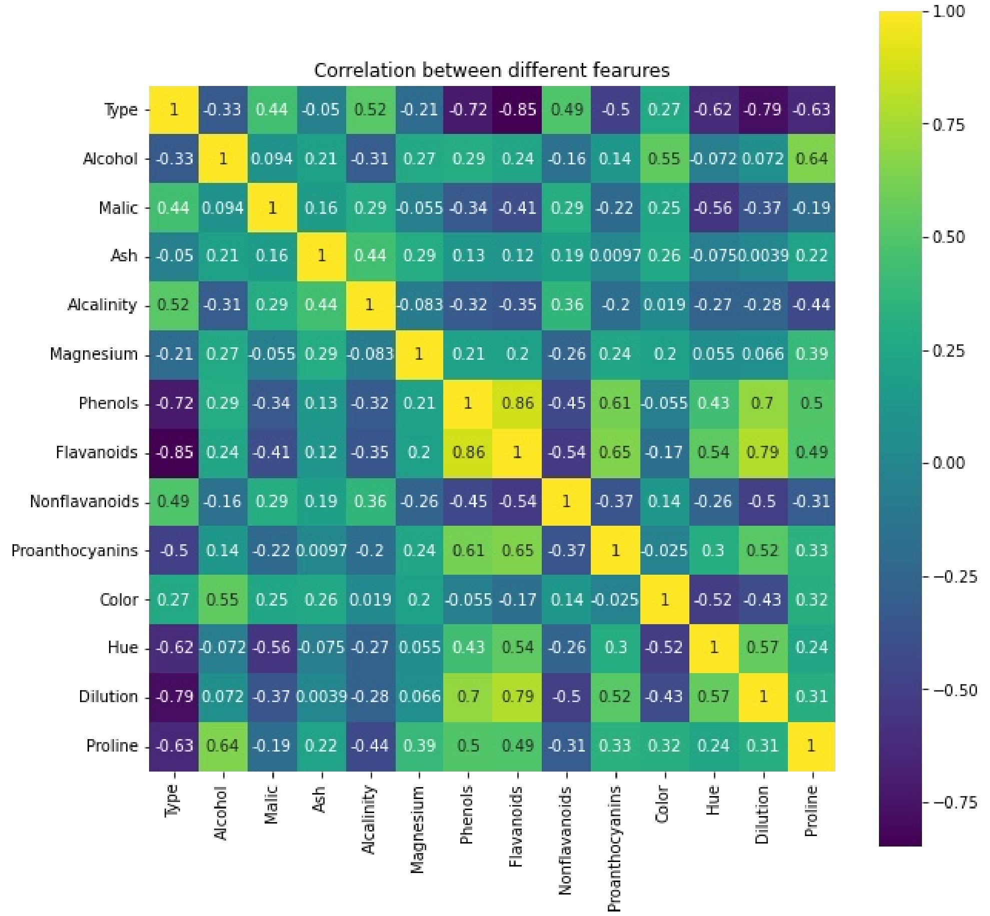
In [6]:

```
import seaborn as sns
correlation = data.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, vmax=1, square=True,annot=True,cmap='viridis')

plt.title('Correlation between different fearures')
```

Out[6]:

```
Text(0.5, 1.0, 'Correlation between different fearures')
```

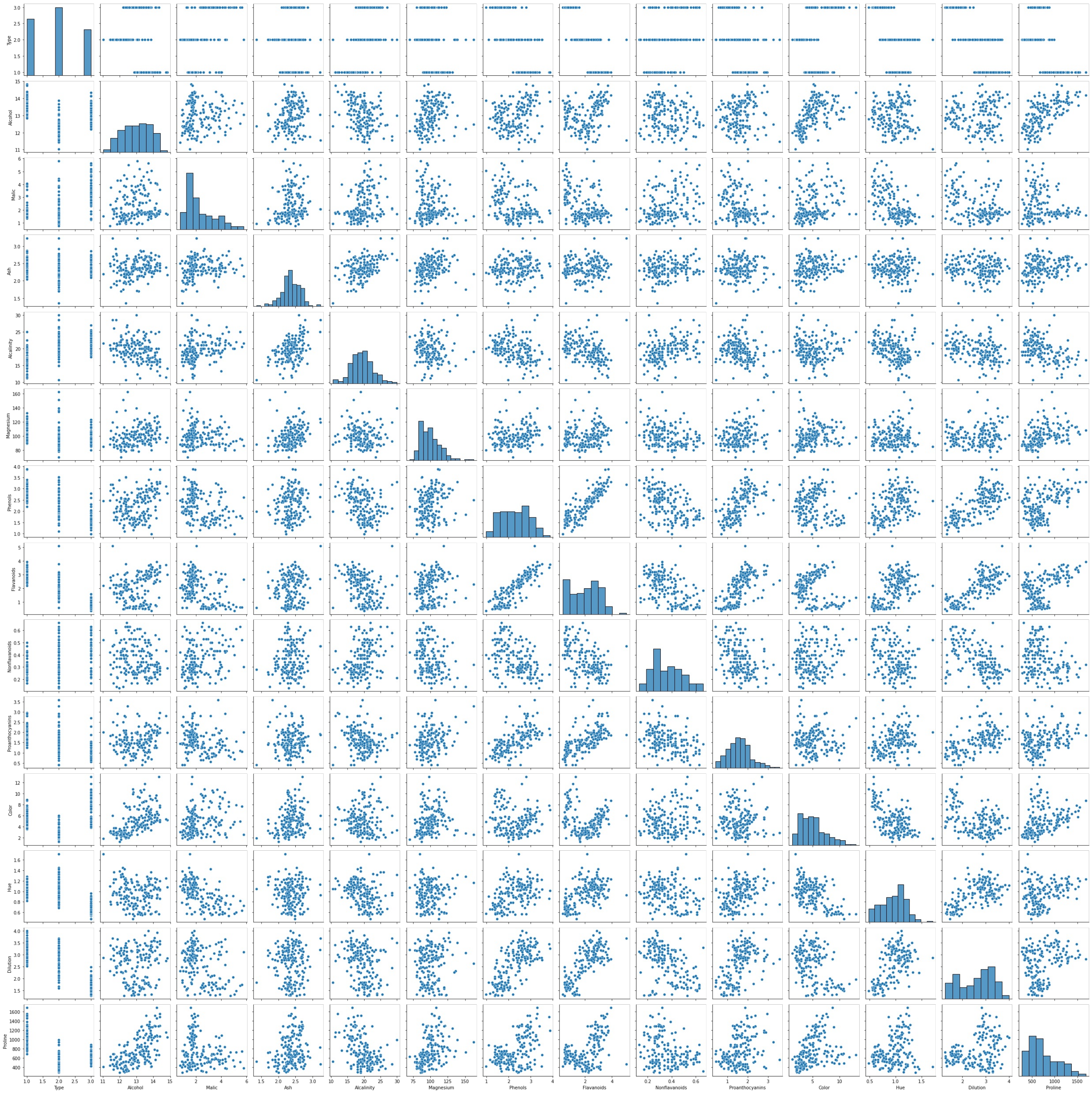


In [7]:

```
sns.pairplot(data)
```

Out[7]:

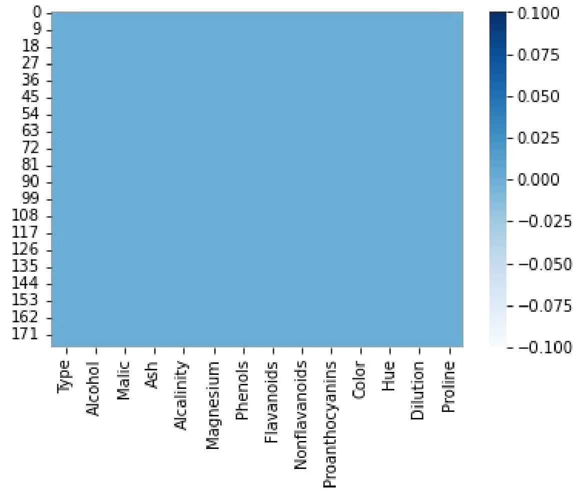
```
&lt;seaborn.axisgrid.PairGrid at 0x24d66197c10&gt;
```



In [8]:

```python
sns.heatmap(data.isnull(),cmap='Blues')
```

Out[8]: <AxesSubplot:>



In [9]:

```python
# there are no null values
```

there are no null values

Split the data

In [10]:
```python
# Normalizing the numerical data
from sklearn.preprocessing import scale
data_norm= scale(data)
data_norm
```

Out[10]:
```
array([[-1.21394365,  1.51861254, -0.5622498 , ...,  0.36217728,
         1.84791957,  1.01300893],
       [-1.21394365,  0.24628963, -0.49941338, ...,  0.40605066,
         1.1134493 ,  0.96524152],
       [-1.21394365,  0.19687903,  0.02123125, ...,  0.31830389,
         0.78858745,  1.39514818],
       ...,
       [ 1.37386437,  0.33275817,  1.74474449, ..., -1.61212515,
        -1.48544548,  0.28057537],
       [ 1.37386437,  0.20923168,  0.22769377, ..., -1.56825176,
        -1.40069891,  0.29649784],
       [ 1.37386437,  1.39508604,  1.58316512, ..., -1.52437837,
        -1.42894777, -0.59516041]])
```

In [11]:
```python
from sklearn.decomposition import PCA
```

In [12]:
```python
# Applying PCA Fit Transform to dataset
pca=PCA(n_components=13)

data_pca=pca.fit_transform(data_norm)
data_pca
```

Out[12]:
```
array([[-3.52293390e+00, -1.45309844e+00, -1.64795488e-01, ...,
        -4.20493905e-01,  5.52927766e-01, -3.02978176e-01],
       [-2.52885806e+00,  3.30019252e-01, -2.02670665e+00, ...,
        -1.30019629e-01,  3.94971160e-01, -1.46645308e-01],
       [-2.78502898e+00, -1.03693595e+00,  9.83237703e-01, ...,
        -2.79074108e-01,  1.89799314e-03,  2.12780166e-02],
       ...,
       [ 3.02727243e+00, -2.75604024e+00, -9.40803036e-01, ...,
         5.02640272e-01,  6.93336340e-01,  1.67035660e-01],
       [ 2.75522166e+00, -2.29378408e+00, -5.50473677e-01, ...,
         3.13785741e-01,  3.44119826e-01, -1.09514873e-01],
       [ 3.49633565e+00, -2.76060799e+00,  1.01315115e+00, ...,
        -2.38282390e-01, -1.89866131e-01, -1.64090011e-01]])
```

In [13]:
```python
# PCA Components matrix or covariance Matrix
pca.components_
```

Out[13]:
```
array([[ 0.39366953, -0.13632501,  0.22267638, -0.00225793,  0.22429849,
        -0.12463016, -0.35926404, -0.39071171,  0.2670012 , -0.2790625 ,
         0.08931829, -0.27682265, -0.35052618, -0.26951525],
       [-0.00569041, -0.48416087, -0.22359095, -0.31585588,  0.01161574,
        -0.30055143, -0.06711983,  0.00131345, -0.0269887 , -0.04122256,
        -0.52978274,  0.27790735,  0.16277625, -0.36605886],
       [ 0.00121795, -0.20740081,  0.08879606,  0.62610236,  0.6119896 ,
         0.13098458,  0.14650775,  0.15096275,  0.16997551,  0.14987959,
        -0.1372663 ,  0.08532854,  0.16620436, -0.12668685],
       [ 0.12246373, -0.08191848,  0.46988824, -0.24984122,  0.07199322,
        -0.16321412,  0.19098521,  0.14461667, -0.32801272,  0.46275771,
         0.07211248, -0.43466618,  0.15672341, -0.2557949 ],
       [ 0.15758395, -0.25089415, -0.18860015, -0.0935236 ,  0.0465675 ,
         0.77833048, -0.14466563, -0.11200553, -0.43257916,  0.0915882 ,
        -0.0462696 , -0.02986657, -0.14419358, -0.08440794],
       [ 0.20033864, -0.13517139, -0.59841948, -0.10799983,  0.08811224,
        -0.14483831,  0.14809748,  0.06247252,  0.25868639,  0.46627764,
         0.42525454, -0.01565089, -0.21770365, -0.0665655 ],
       [-0.05938234, -0.09269887,  0.3743698 , -0.16708856, -0.26872469,
         0.32957951, -0.03789829, -0.06773223,  0.61111195,  0.42292282,
        -0.18613617,  0.19204101, -0.0785098 ,  0.0542037 ],
       [-0.07179553, -0.42154435, -0.08757556,  0.17208034, -0.41324857,
         0.14881189,  0.36343884,  0.175405  ,  0.23075135, -0.3437392 ,
         0.04069617, -0.48362564,  0.06865116, -0.11146671],
       [-0.16236882, -0.45019071, -0.00602569,  0.26249446, -0.11863342,
        -0.25253628, -0.40637354, -0.09091933, -0.15912282,  0.26578679,
        -0.07526459, -0.21241681, -0.08426484,  0.54490539],
       [-0.19899373,  0.31127983, -0.32592413, -0.12452347,  0.15716811,
         0.12773363, -0.30772263, -0.14044   ,  0.24054263,  0.10869629,
```

```
        -0.21704255, -0.50966073,  0.45570504, -0.04620802],
       [ 0.01444169, -0.22154641,  0.06839251, -0.49452428,  0.47461722,
         0.07119731,  0.29740957, -0.03219187,  0.12200984, -0.23292405,
         0.01972448, -0.06140493,  0.06646166,  0.55130818],
       [ 0.01575769, -0.26411262,  0.1192121 , -0.04502305, -0.06131271,
         0.06116074, -0.30087591, -0.05001396,  0.04266558, -0.09334264,
         0.59795428,  0.25774292,  0.61109218, -0.07268036],
       [-0.49224318, -0.05610645,  0.06675544, -0.19201787,  0.20007784,
         0.05829909, -0.35952714,  0.59834288,  0.06403952, -0.11013538,
         0.15917751, -0.04923091, -0.32941979, -0.17322892]])
```

In [14]:

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

Out[14]:

```
array([0.39542486, 0.17836259, 0.10329102, 0.06627984, 0.06267875,
       0.0480556 , 0.03955707, 0.02500244, 0.02103871, 0.01873615,
       0.01613203, 0.01205691, 0.00925458])
```

In [15]:

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

Out[15]:

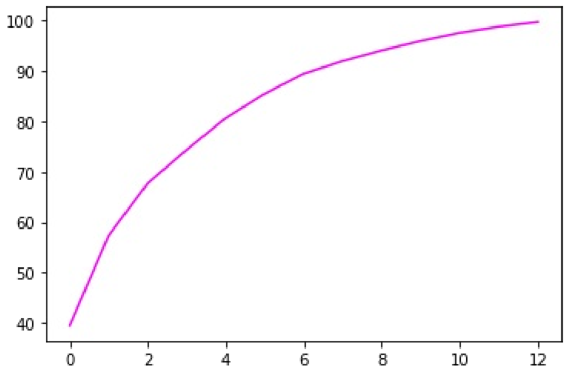
```
array([39.54, 57.38, 67.71, 74.34, 80.61, 85.42, 89.38, 91.88, 93.98,
       95.85, 97.46, 98.67, 99.6 ])
```

In [16]:

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

Out[16]:

```
[<matplotlib.lines.Line2D at 0x24d70abb370>]
```



In [17]:

```
# Final Dataframe
final_df=pd.concat([data['Type'],pd.DataFrame(data_pca[:,0:3],columns=['PC1','PC2','PC3'])],axis=1)
final_df
```

Out[17]:

	Type	PC1	PC2	PC3
0	1	-3.522934	-1.453098	-0.164795
1	1	-2.528858	0.330019	-2.026707
2	1	-2.785029	-1.036936	0.983238
3	1	-3.922588	-2.768210	-0.174968
4	1	-1.407511	-0.867773	2.025829
...	...	...	...	...
173	3	3.627996	-2.206617	-0.343668
174	3	2.942729	-1.752263	0.207480
175	3	3.027272	-2.756040	-0.940803
176	3	2.755222	-2.293784	-0.550474
177	3	3.496336	-2.760608	1.013151

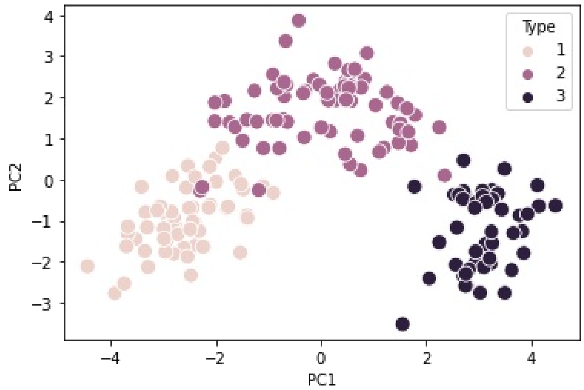
178 rows × 4 columns

visualization

In [18]:

```
import seaborn as sns
sns.scatterplot(data=final_df,x='PC1',y='PC2',hue='Type',s = 100)
```

Out[18]: <AxesSubplot:xlabel='PC1', ylabel='PC2'>



Checking with other Clustering Algorithms

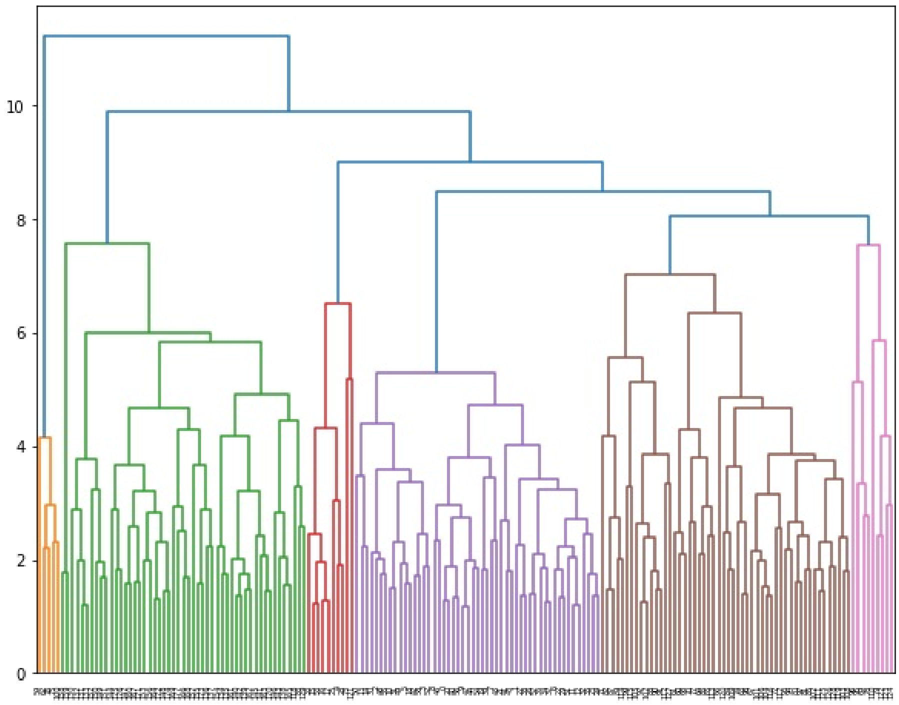
1.Hiearchical Clustering

In [19]:

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

In [20]:

```
plt.figure(figsize=(10,8))
dendrogram = sch.dendrogram(sch.linkage(data_norm,method='complete'))
```



In [21]:

```
hc = AgglomerativeClustering(n_clusters=3,affinity = 'euclidean',linkage='complete')
y_hc = hc.fit_predict(data_norm)
y_hc
```

Out[21]:

```
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, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 2, 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, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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], dtype=int64)
```

In [22]:

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

Out[22]:

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

In [23]:

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

Out[23]:

```
0    65
2    65
1    48
Name: clustersid, dtype: int64
```

In [24]:

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

Out[24]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline	cl
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740	
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750	
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835	
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840	
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560	

178 rows × 15 columns

In [25]:

```
set.head()
```

Out[25]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline	clus
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	

2. K-Means Clustering

In [26]:

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

In [27]:

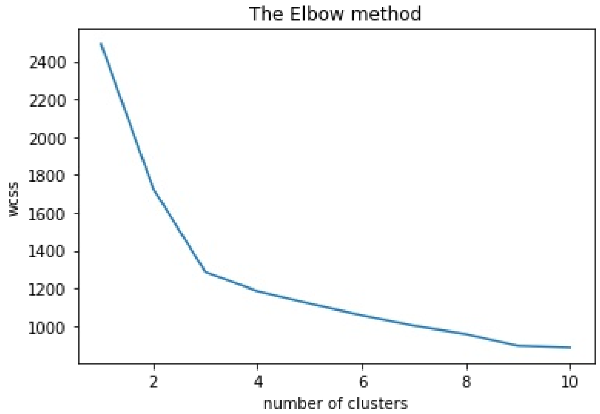
```
# As we already have normalized data
# Use Elbow Graph to find optimum number of  clusters (K value) from K values range
# The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum-of-squares crit
```

```
# random state can be anything from 0 to 42, but the same number to be used everytime,so that the results don't c
```

In [28]:

```
wcss = []
for i in range (1,11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state= 42)
    kmeans.fit(data_norm)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('The Elbow method')
plt.xlabel('number of clusters')
plt.ylabel('wcss')
plt.show()
```

```
C:\Users\rajesh\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 [29]:

```
wcss
```

Out[29]:

```
[2491.9999999999995,
 1727.2286609320033,
 1285.5622587402038,
 1184.189834564335,
 1117.9508459195683,
 1056.348790833104,
 1000.8514446090732,
 956.8151460962835,
 894.206930241249,
 884.9968687910574]
```

Build the cluster using k=3

In [30]:

```
# Cluster algorithm using K=3
clusters3=KMeans(3,random_state=30).fit(data_norm)
clusters3
```

Out[30]:

```
KMeans(n_clusters=3, random_state=30)
```

In [31]:

```
clusters3.labels_
```

Out[31]:

```
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, 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, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 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, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       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 [32]:

```
# Assign clusters to the data set
wine=data.copy()
```

```
wine['clusters3id']=clusters3.labels_
wine
```

Out[32]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline	cl
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740	
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750	
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835	
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840	
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560	

178 rows × 15 columns

In [33]:

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

Out[33]:

```
1    68
2    61
0    49
Name: clusters3id, dtype: int64
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js