In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.cluster import KMeans
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

executed in 19ms, finished 13:10:19 2022-01-12

In [3]:

```
#Importing the respective csv file and having a glance at it
wine_data = pd.read_csv("wine.csv")
wine_data.head()
```

executed in 3.45s, finished 13:15:01 2022-01-12

Out[3]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proa
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	

In [4]:

```
#checking the unique values in the given data
wine_data.Type.unique()
```

executed in 84ms, finished 13:20:57 2022-01-12

Out[4]:

```
array([1, 2, 3], dtype=int64)
```

In [5]:

```
#checking the rows and columns of the dataset
wine_data.shape
```

executed in 21ms, finished 13:21:20 2022-01-12

Out[5]:

```
(178, 14)
```

In [6]:

```
#dropping the type column since it contains three unique values which might be considered a
wine1 = wine_data.drop("Type",axis=1)
wine1.head()
```

executed in 223ms, finished 13:21:34 2022-01-12

Out[6]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocy
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	

In [7]:

```
#checking the rows and columns of new data
wine1.shape
```

executed in 18ms, finished 13:26:46 2022-01-12

Out[7]:

```
(178, 13)
```

In [8]:

```
#checking for the datatype and if any nulls values present in the dataset
wine1.info()
```

executed in 559ms, finished 13:26:59 2022-01-12

```
<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 [10]:

```
#checking for the duplicate values if present any
wine1[wine1.duplicated(keep = False)]
```

executed in 37ms, finished 13:33:02 2022-01-12

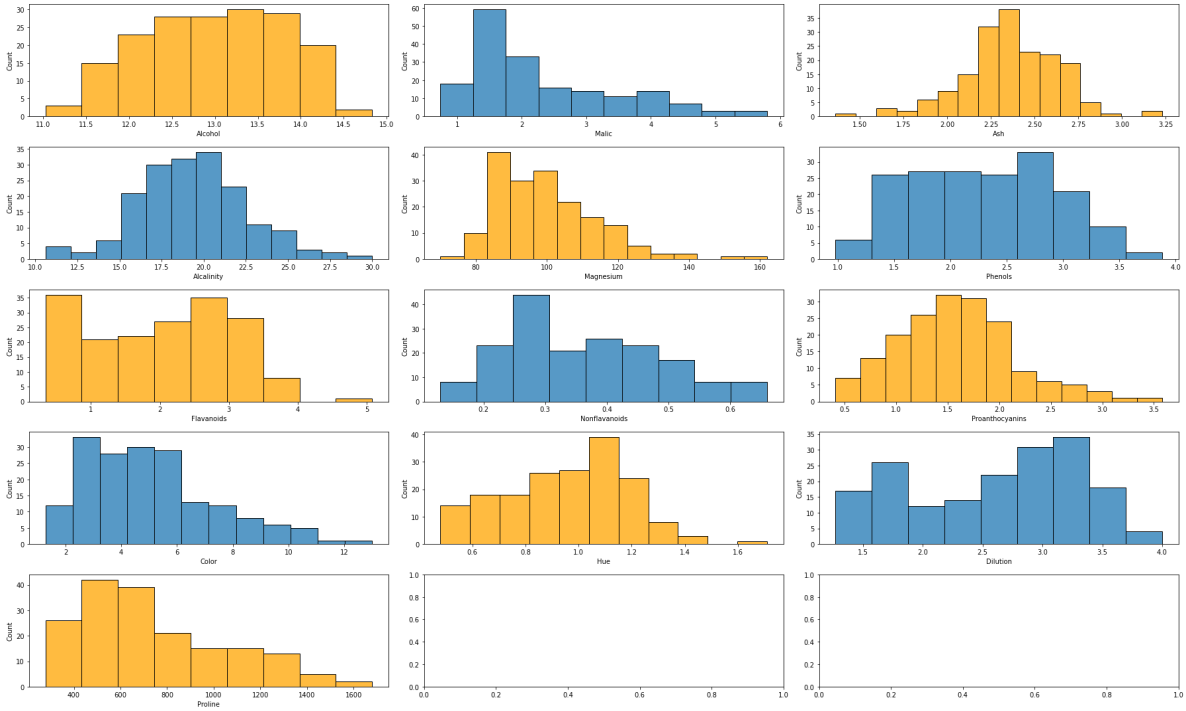
Out[10]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocya

In [11]:

```
#checking for the outliers in the dataset using the plots
fig, ax = plt.subplots(5, 3, figsize=(25,15))

sns.histplot(wine1.Alcohol,ax=ax[0,0],color='orange')
sns.histplot(wine1.Malic,ax=ax[0,1])
sns.histplot(wine1.Ash,ax=ax[0,2],color='orange')

sns.histplot(wine1.Alcalinity,ax=ax[1,0])
sns.histplot(wine1.Magnesium,ax=ax[1,1],color='orange')
sns.histplot(wine1.Phenols,ax=ax[1,2])

sns.histplot(wine1.Flavanoids,ax=ax[2,0],color='orange')
sns.histplot(wine1.Nonflavanoids,ax=ax[2,1])
sns.histplot(wine1.Proanthocyanins,ax=ax[2,2],color='orange')

sns.histplot(wine1.Color,ax=ax[3,0])
sns.histplot(wine1.Hue,ax=ax[3,1],color='orange')
sns.histplot(wine1.Dilution,ax=ax[3,2])

sns.histplot(wine1.Proline, ax=ax[4,0],color='orange')
plt.tight_layout()
plt.show()
```

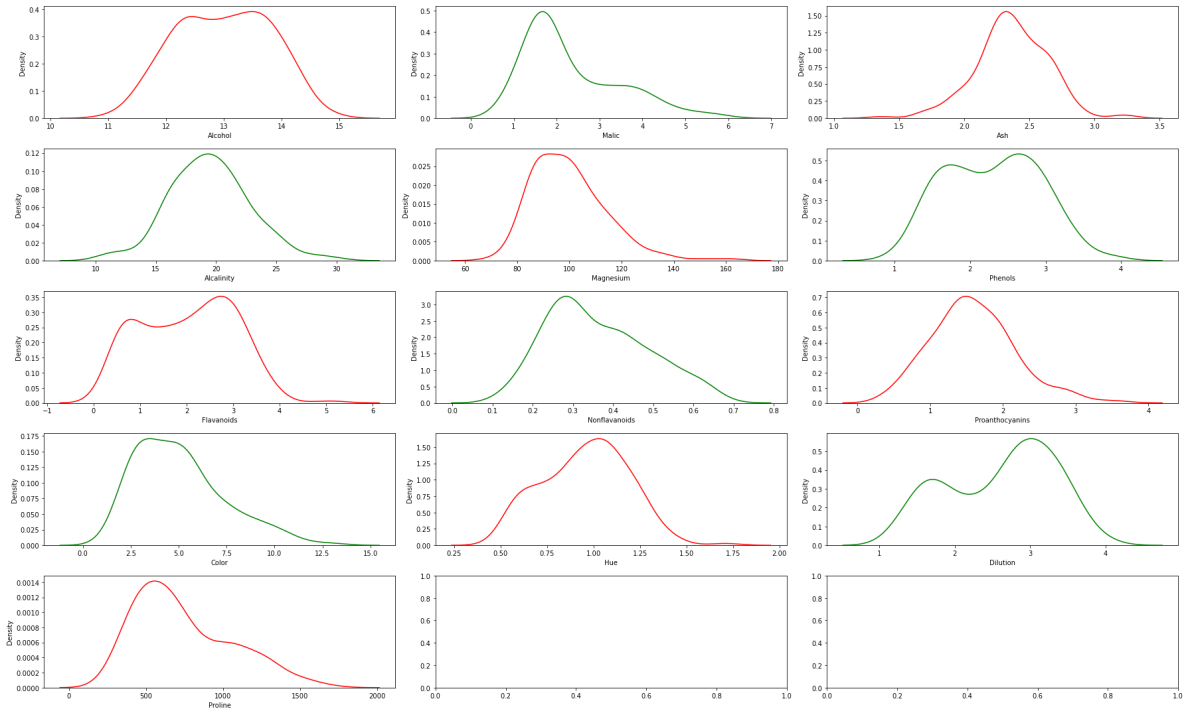
executed in 41.4s, finished 13:34:19 2022-01-12



In [12]:

```
#checking if our data follows normal distribution or not
fig, ax = plt.subplots(5, 3, figsize=(25,15))

sns.kdeplot(wine1.Alcohol,ax=ax[0,0],color='red')
sns.kdeplot(wine1.Malic,ax=ax[0,1],color='g')
sns.kdeplot(wine1.Ash,ax=ax[0,2],color='red')

sns.kdeplot(wine1.Alcalinity,ax=ax[1,0],color='g')
sns.kdeplot(wine1.Magnesium,ax=ax[1,1],color='red')
sns.kdeplot(wine1.Phenols,ax=ax[1,2],color='g')

sns.kdeplot(wine1.Flavanoids,ax=ax[2,0],color='red')
sns.kdeplot(wine1.Nonflavanoids,ax=ax[2,1],color='g')
sns.kdeplot(wine1.Proanthocyanins,ax=ax[2,2],color='red')

sns.kdeplot(wine1.Color,ax=ax[3,0],color='g')
sns.kdeplot(wine1.Hue,ax=ax[3,1],color='red')
sns.kdeplot(wine1.Dilution,ax=ax[3,2],color='g')

sns.kdeplot(wine1.Proline,ax=ax[4,0],color='red')
plt.tight_layout()
plt.show()
```

executed in 6.98s, finished 13:34:51 2022-01-12



In [13]:

```
#Normalizing the data inorder to avoid the variances in the data
wine1_norm = scale(wine1)
wine1_norm
```

executed in 612ms, finished 13:35:18 2022-01-12

Out[13]:

```
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]])
```

Performing the Heirarchical Clustering using different types of linkage models

Here Initially Heirarchical Clustering is performed on different types of linkage models and its performance is checked and then PCA is performed on it

Performing Simple Linkage Model

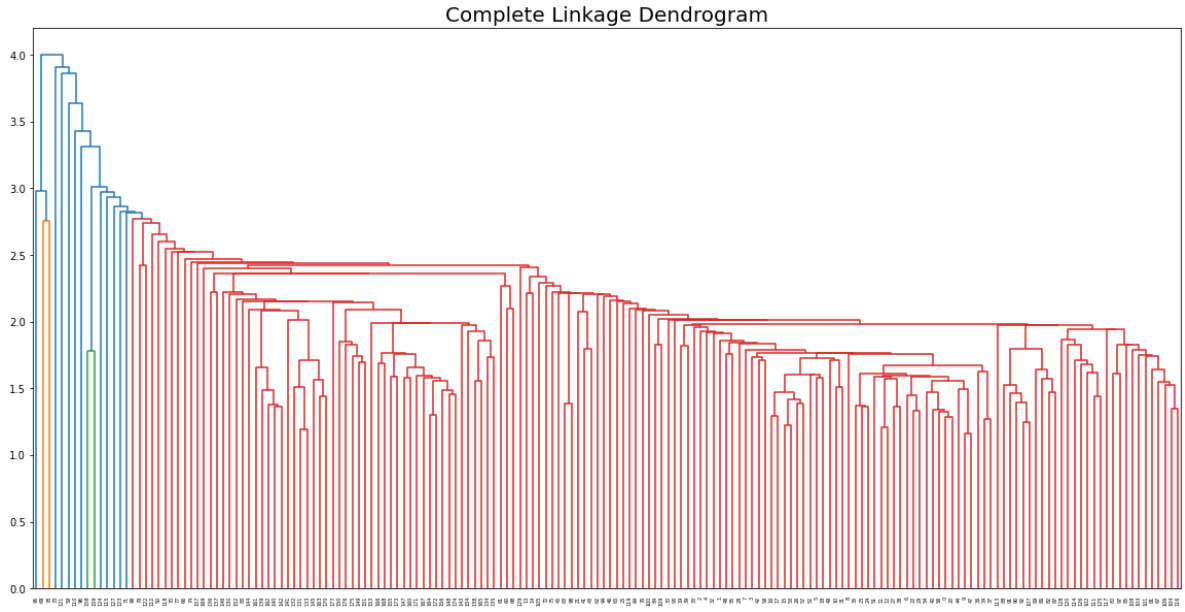
In [14]:

```
#Plotting the dendogram plot for the Complete Linkage Model
fig=plt.figure(figsize=(20,10))
dendrogram = sch.dendrogram(sch.linkage(wine1_norm, method='single'))
plt.title("Complete Linkage Dendrogram",size=20)
```

executed in 23.5s, finished 13:37:18 2022-01-12

Out[14]:

```
Text(0.5, 1.0, 'Complete Linkage Dendrogram')
```



In [60]:

```
#Building single linkage model with two cluster
hc1 = AgglomerativeClustering(n_clusters=2, affinity = 'euclidean', linkage = 'singal')
```

executed in 41ms, finished 14:31:15 2022-01-12

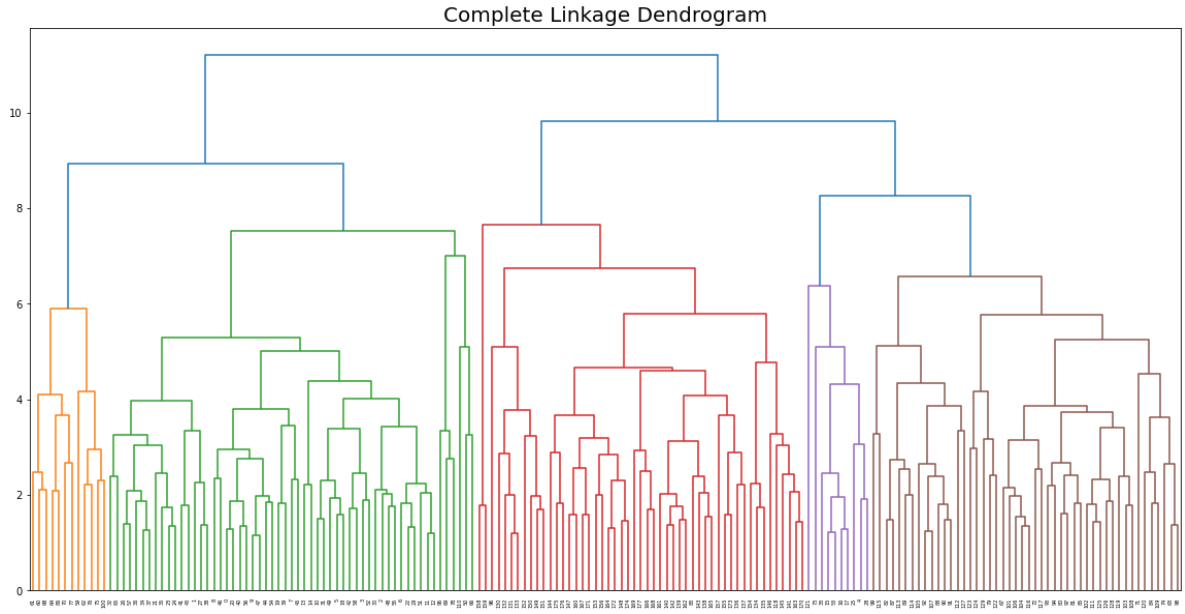
In [34]:

```
#Plotting the dendogram plot for the Complete Linkage Model
fig=plt.figure(figsize=(20,10))
dendrogram = sch.dendrogram(sch.linkage(wine1_norm, method='complete'))
plt.title("Complete Linkage Dendrogram",size=20)
```

executed in 22.1s, finished 13:53:49 2022-01-12

Out[34]:

```
Text(0.5, 1.0, 'Complete Linkage Dendrogram')
```



In [35]:

```
#Building complete Linkage model
hc2 = AgglomerativeClustering(n_clusters=4, affinity = 'euclidean', linkage = 'complete')
```

executed in 8ms, finished 13:54:35 2022-01-12

In [36]:

```
#fitting the model on the data
y_hc2 = hc2.fit_predict(wine1_norm)
Clusters=pd.DataFrame(y_hc2,columns=['Clusters'])

#Creating the cluster column for the build model
wine_data['cluster'] = y_hc2
```

executed in 118ms, finished 13:54:47 2022-01-12

In [38]:

```
#Checking how many values fall under each of the clusters created
for i in range(4):
    print("cluster", i)
    print("Total Values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 97ms, finished 13:55:42 2022-01-12

```
cluster 0
Total Values: 58
cluster 1
Total Values: 57
cluster 2
Total Values: 51
cluster 3
Total Values: 12
```

Performing Average Linkage Model

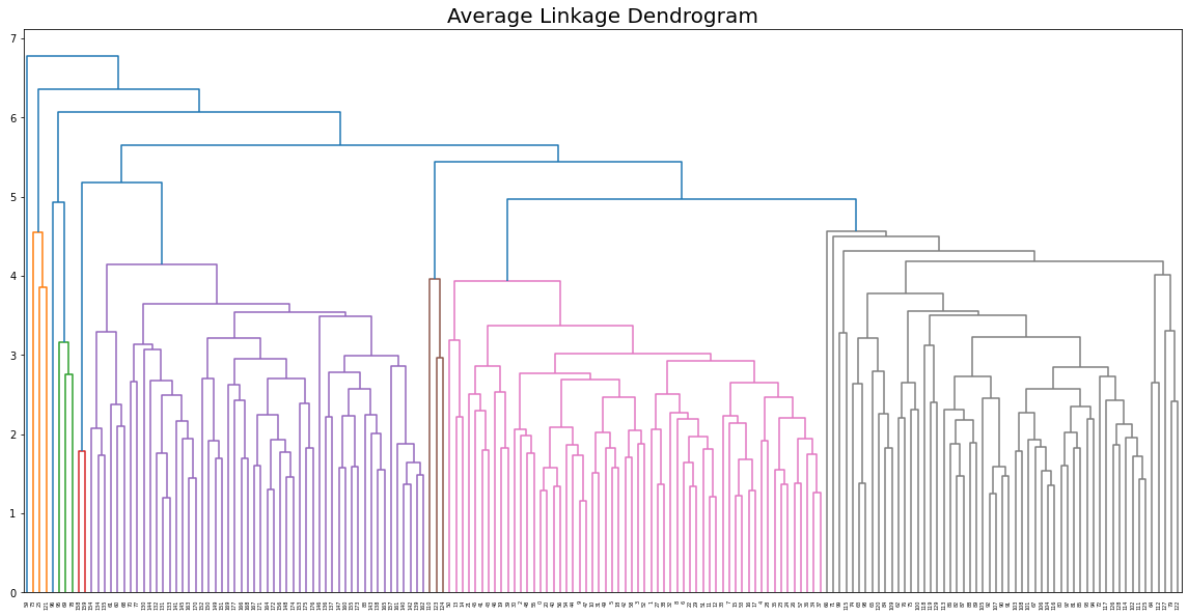
In [39]:

```
#Plotting the dendogram plot for the Average Linkage Model
fig=plt.figure(figsize=(20,10))
dendrogram = sch.dendrogram(sch.linkage(wine1_norm, method='average'))
plt.title("Average Linkage Dendrogram",size=20)
```

executed in 23.0s, finished 13:57:19 2022-01-12

Out[39]:

```
Text(0.5, 1.0, 'Average Linkage Dendrogram')
```



In [40]:

```
#Building the Average Linkage Model
hc3 = AgglomerativeClustering(n_clusters=5, affinity = 'euclidean', linkage = 'average')
```

executed in 7ms, finished 13:57:57 2022-01-12

In [41]:

```
#fitting the model on the data
y_hc3 = hc3.fit_predict(wine1_norm)
Clusters=pd.DataFrame(y_hc3,columns=['Clusters'])

##Creating the cluster column for the build model
wine_data['cluster'] = y_hc3
```

executed in 23ms, finished 13:58:08 2022-01-12

In [42]:

```
#Checking how many values fall under each of the clusters created
for i in range(5):
    print("cluster", i)
    print("Total Values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 24ms, finished 13:58:22 2022-01-12

```
cluster 0
Total Values: 116
cluster 1
Total Values: 54
cluster 2
Total Values: 3
cluster 3
Total Values: 1
cluster 4
Total Values: 4
```

Performing Centroid(ward) Linkage model

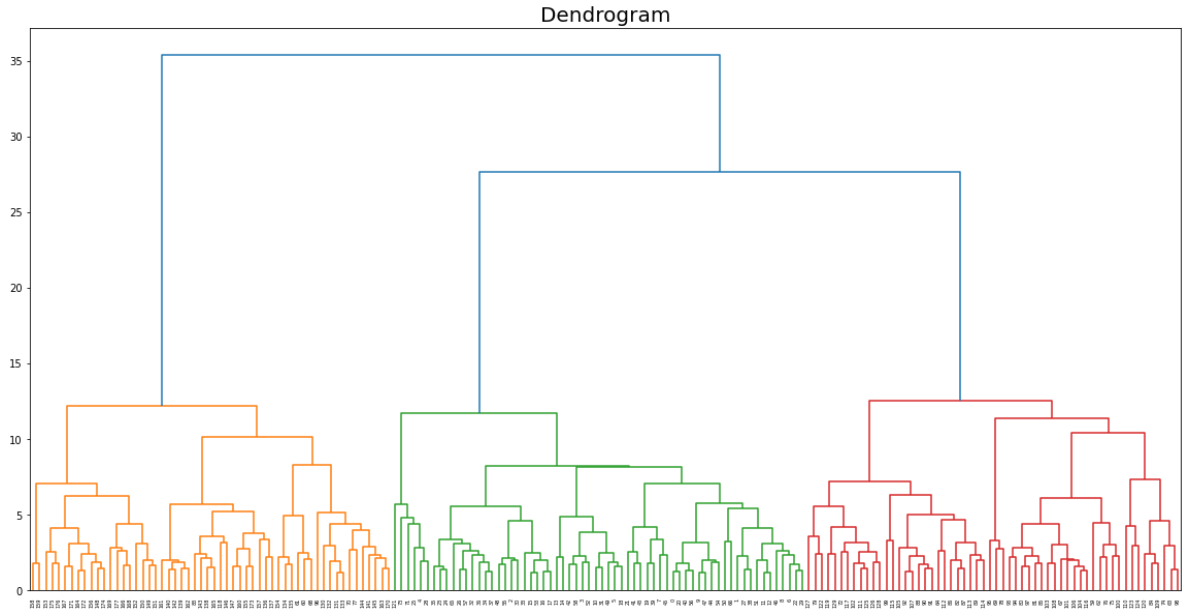
In [43]:

```
#Plotting the dendogram plot for Centroid Linkage Model
fig=plt.figure(figsize=(20,10))
dendrogram = sch.dendrogram(sch.linkage(wine1_norm, method='ward'))
plt.title("Dendrogram",size=20)
```

executed in 23.0s, finished 13:59:31 2022-01-12

Out[43]:

```
Text(0.5, 1.0, 'Dendrogram')
```



In [44]:

```
#Building Centroid Linkage model
hc4 = AgglomerativeClustering(n_clusters=3, affinity = 'euclidean', linkage = 'ward')
```

executed in 9ms, finished 13:59:39 2022-01-12

In [45]:

```
#fitting the model on the data
y_hc4 = hc4.fit_predict(wine1_norm)
Clusters=pd.DataFrame(y_hc4,columns=['Clusters'])

#Creating the cluster column for the build model
wine_data['cluster'] = y_hc4
```

executed in 27ms, finished 13:59:53 2022-01-12

In [47]:

```
#Checking how many values fall under each of the clusters created
for i in range(3):
    print("cluster", i)
    print("Total Values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 26ms, finished 14:01:00 2022-01-12

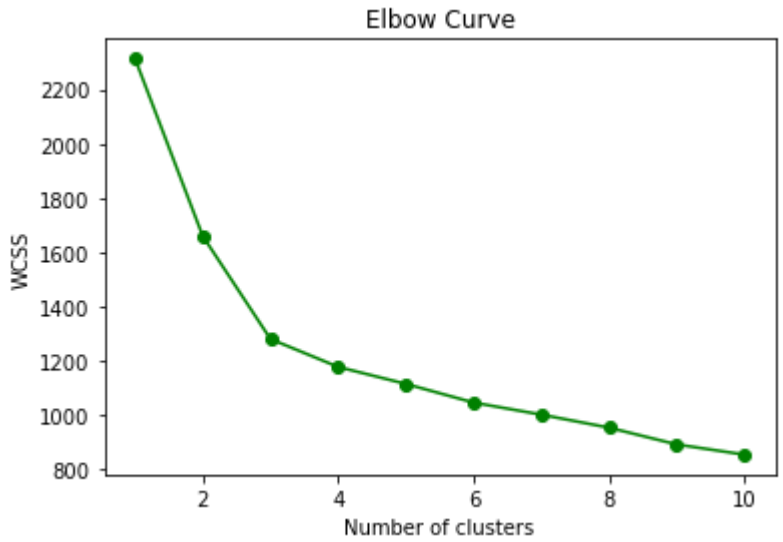
```
cluster 0
Total Values: 58
cluster 1
Total Values: 56
cluster 2
Total Values: 64
```

Performing KMeans Clustering

In [48]:

```
#Plotting an elbow curve to check for k value
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i,random_state=0)
    kmeans.fit(wine1_norm)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss,'bo-',color='g')
plt.title('Elbow Curve')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

executed in 5.06s, finished 14:01:46 2022-01-12



In [49]:

```
#Building K-means Clustering model and fitting the data into it
Clusters_newdata = KMeans(3, random_state=42)
Clusters_newdata.fit(wine1_norm)
```

executed in 57ms, finished 14:02:55 2022-01-12

Out[49]:

```
KMeans(n_clusters=3, random_state=42)
```

In [50]:

```
#Creating labels and then creating new column to put into our dataset
KMeans_label=Clusters_newdata.labels_
wine_data['cluster'] = Clusters_newdata.labels_
```

executed in 22ms, finished 14:03:07 2022-01-12

In [51]:

```
#Checking how many values fall under each of the clusters created
for i in range(3):
    print("cluster", i)
    print("Total Values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 79ms, finished 14:03:27 2022-01-12

```
cluster 0
Total Values: 51
cluster 1
Total Values: 62
cluster 2
Total Values: 65
```

Evaluating Clustering methods with the help of Silhouette Score

The Silhouette Score metric calculates the goodness of the clustering techniques and it ranges from -1 to 1.

1: Means clusters are well apart from each other and clearly distinguised.

0:Means the distance between clusters is not significant

-1:Means clusters are assigned in the wrong way

In [53]:

```
#Silhouette Score of Single Linkage Method
Silhou_SLM=metrics.silhouette_score(wine1_norm,y_hc2)
Silhou_SLM
```

executed in 352ms, finished 14:05:19 2022-01-12

Out[53]:

```
0.19382526203175696
```

In [54]:

```
#Silhouette Score of Average Linkage Method
Silhou_Average=metrics.silhouette_score(wine1_norm,y_hc3)
Silhou_Average
```

executed in 83ms, finished 14:05:39 2022-01-12

Out[54]:

```
0.22945756295901437
```

In [55]:

```
#Silhouette Score of Centroid(ward) Linkage Method
Silhou_CenLM=metrics.silhouette_score(wine1_norm,y_hc4)
Silhou_CenLM
```

executed in 28ms, finished 14:06:11 2022-01-12

Out[55]:

```
0.2774439826952265
```

In [56]:

```
#Silhouette Score of Kmeans Clustering
Silhou_KMeans=metrics.silhouette_score(wine1_norm,KMeans_label)
Silhou_KMeans
```

executed in 21ms, finished 14:06:31 2022-01-12

Out[56]:

```
0.2848589191898987
```

In [64]:

```
#Listing into the table
Table={'Model':pd.Series(['HC_SingleLinakge','HC_AverageLinkage','HC_CentroidLinkage','KMea
  'Silhouette score':[Silhou_SLM,Silhou_Average,Silhou_CenLM,Silhou_KMeans]
  }
Table=pd.DataFrame(Table)
Table
```

executed in 60ms, finished 14:34:42 2022-01-12

Out[64]:

	Model	Silhouette score
0	HC_SingleLinakge	0.193825
1	HC_AverageLinkage	0.229458
2	HC_CentroidLinkage	0.277444
3	KMeans	0.284859

PCA Method

In [65]:

```
pca = PCA()
pca_values = pca.fit_transform(wine1_norm)
pca_values
```

executed in 1.03s, finished 14:35:52 2022-01-12

Out[65]:

```
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 [66]:

```
#Applying the PCA on the dataset with 13components first
pca = PCA(n_components = 13)
pca_values = pca.fit_transform(wine1_norm)
```

executed in 47ms, finished 14:36:14 2022-01-12

In [67]:

```
#checking the variance of the PCA components
var = pca.explained_variance_ratio_
var
```

executed in 89ms, finished 14:36:33 2022-01-12

Out[67]:

```
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])
```

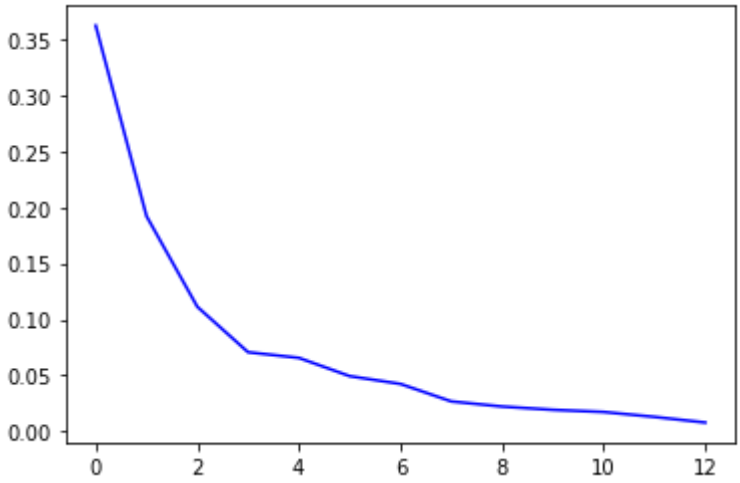
In [68]:

```
plt.plot(var,color="blue")
```

executed in 465ms, finished 14:36:47 2022-01-12

Out[68]:

```
[<matplotlib.lines.Line2D at 0x19cc3b704f0>]
```



In [69]:

```
#checking the cummulative variance of the PCA components
varc = np.cumsum(np.round(var,decimals = 4)*100)
varc
```

executed in 88ms, finished 14:37:06 2022-01-12

Out[69]:

```
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])
```

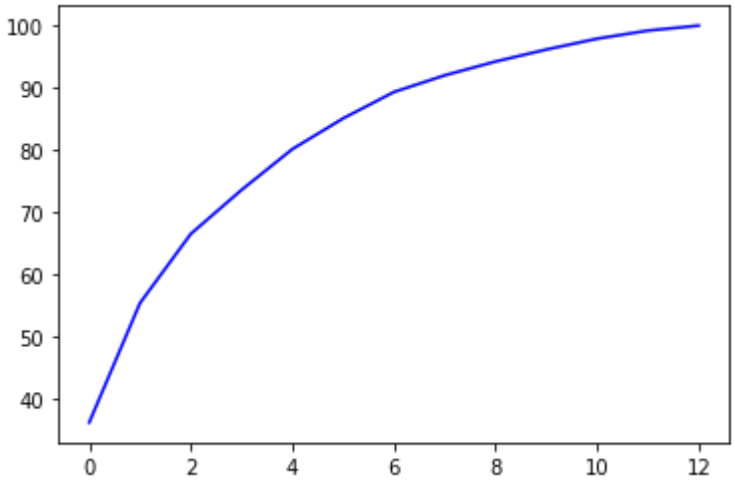
In [70]:

```
plt.plot(varc,color="blue")
```

executed in 396ms, finished 14:37:19 2022-01-12

Out[70]:

```
[<matplotlib.lines.Line2D at 0x19cc1cadd30>]
```



In [71]:

```
#Checking the components of the PCA
pca.components_
```

executed in 115ms, finished 14:37:39 2022-01-12

Out[71]:

```
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 [72]:

```
#Considering the PCA of 3 components
PCA_wine=pca_values[:,0:3]
```

executed in 133ms, finished 14:38:25 2022-01-12

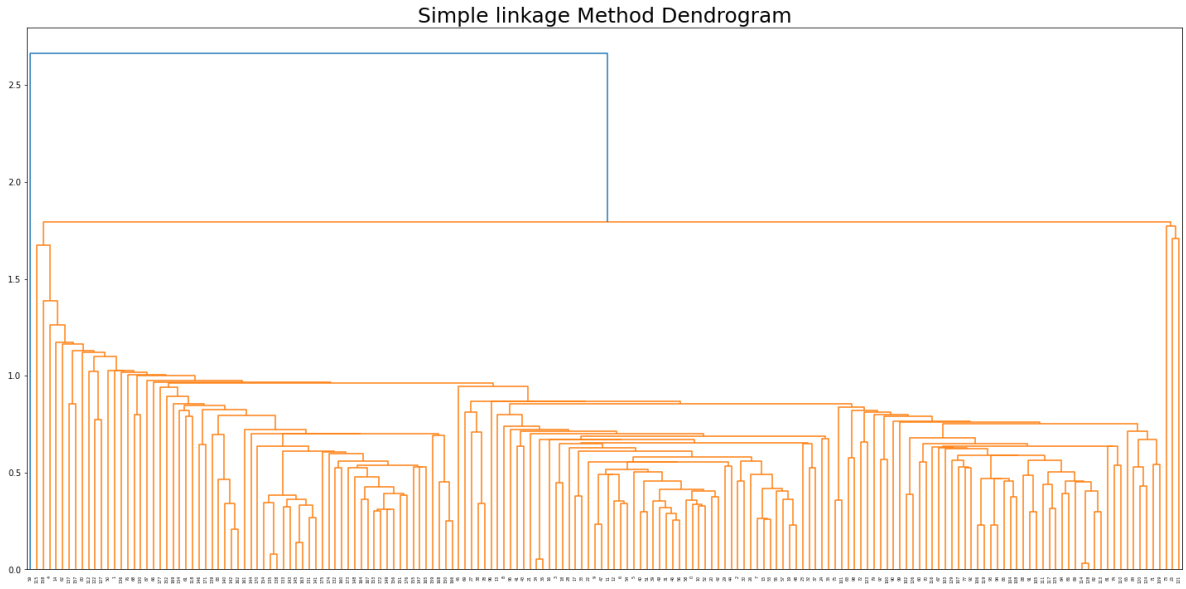
In [73]:

```
#Plotting the dendogram plot for the Single Linkage Model
fig=plt.figure(figsize=(25,12))
dendrogram = sch.dendrogram(sch.linkage(PCA_wine, method='single'),)
plt.title("Simple linkage Method Dendrogram",size=25)
```

executed in 28.3s, finished 14:39:21 2022-01-12

Out[73]:

```
Text(0.5, 1.0, 'Simple linkage Method Dendrogram')
```



In [74]:

```
#Building single linkage model with five cluster
PCA_hc1 = AgglomerativeClustering(n_clusters=2, affinity = 'euclidean', linkage = 'single')
```

executed in 79ms, finished 14:39:34 2022-01-12

In [75]:

```
#Fitting the model and Creating the cluster column for the built model
y_PCAhc1 = PCA_hc1 .fit_predict(PCA_wine)
Clusters=pd.DataFrame(y_PCAhc1,columns=['Clusters'])

wine_data['cluster'] = y_PCAhc1
```

executed in 477ms, finished 14:39:54 2022-01-12

In [76]:

```
#Checking how many values fall under each of the clusters created
for i in range(2):
    print("cluster", i)
    print("Total Values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 21ms, finished 14:40:07 2022-01-12

```
cluster 0
Total Values: 177
cluster 1
Total Values: 1
```

Performing Complete Linkage Model on 3 Components

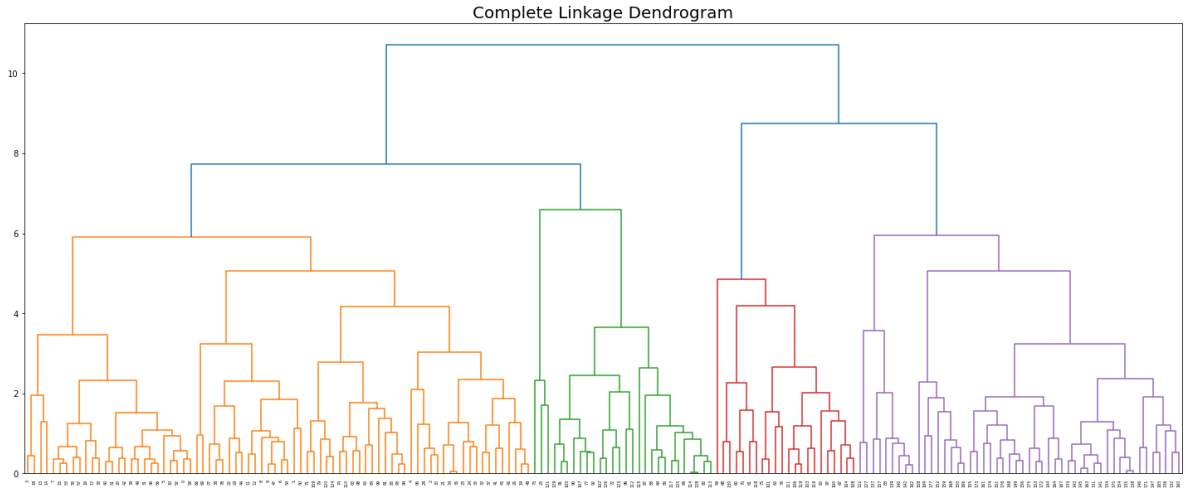
In [77]:

```
#Plotting the dendogram plot for the Complete Linkage Model
fig=plt.figure(figsize=(25,10))
dendrogram = sch.dendrogram(sch.linkage(PCA_wine, method='complete'))
plt.title("Complete Linkage Dendrogram",size=20)
```

executed in 26.3s, finished 14:41:23 2022-01-12

Out[77]:

```
Text(0.5, 1.0, 'Complete Linkage Dendrogram')
```



In [78]:

```
#Building single linkage model with four cluster
PCA_hc2= AgglomerativeClustering(n_clusters=4, affinity = 'euclidean', linkage = 'complete'
```

executed in 6ms, finished 14:41:31 2022-01-12

In [79]:

```
#Fitting the model and Creating the cluster column for the built model
y_PCAhc2 = PCA_hc2 .fit_predict(PCA_wine)
Clusters=pd.DataFrame(y_PCAhc2,columns=['Clusters'])

wine_data['cluster'] = y_PCAhc2
```

executed in 89ms, finished 14:41:45 2022-01-12

In [80]:

```
#Checking how many values fall under each of the clusters created
for i in range(4):
    print("cluster", i)
    print("Total Values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 33ms, finished 14:41:59 2022-01-12

```
cluster 0
Total Values: 28
cluster 1
Total Values: 50
cluster 2
Total Values: 22
cluster 3
Total Values: 78
```

Performing Average Linkage Model on 3 Components

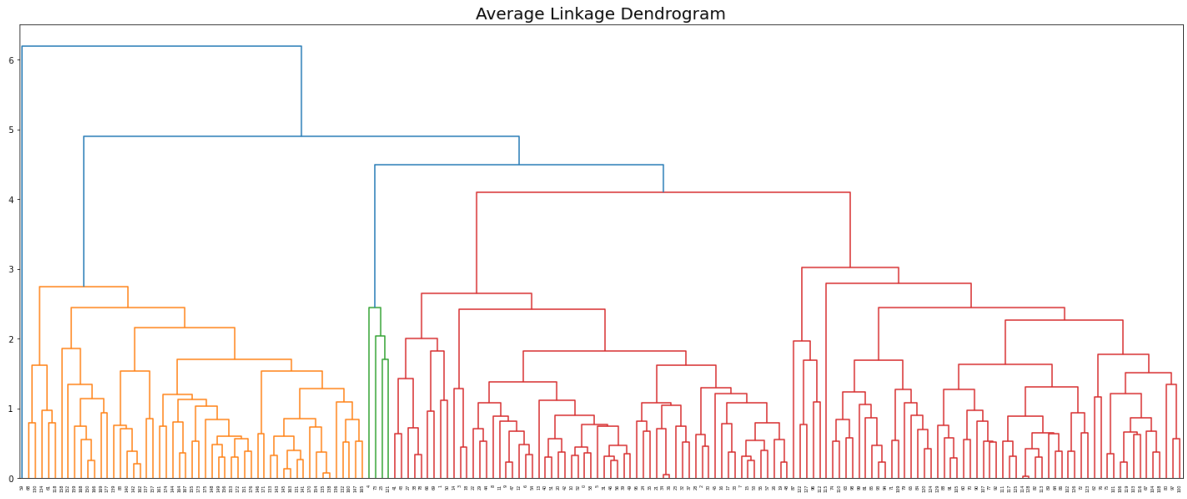
In [81]:

```
#Plotting the dendogram plot for the Average Linkage Model
fig=plt.figure(figsize=(25,10))
dendrogram = sch.dendrogram(sch.linkage(PCA_wine, method='average'))
plt.title("Average Linkage Dendrogram",size=20)
```

executed in 27.6s, finished 14:43:17 2022-01-12

Out[81]:

```
Text(0.5, 1.0, 'Average Linkage Dendrogram')
```



In [82]:

```
#Building the Average Linkage Model
PCA_hc3= AgglomerativeClustering(n_clusters=5, affinity = 'euclidean', linkage = 'average')
```

executed in 10ms, finished 14:43:48 2022-01-12

In [83]:

```
#Fitting the model and Creating the cluster column for the built model
y_PCAhc3 = PCA_hc3.fit_predict(PCA_wine)
Clusters=pd.DataFrame(y_PCAhc3,columns=['Clusters'])

wine_data['cluster'] = y_PCAhc3
```

executed in 20ms, finished 14:44:03 2022-01-12

In [84]:

```
#Checking how many values fall under each of the clusters created
for i in range(5):
    print("cluster", i)
    print("Total Values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 60ms, finished 14:44:20 2022-01-12

```
cluster 0
Total Values: 60
cluster 1
Total Values: 61
cluster 2
Total Values: 52
cluster 3
Total Values: 1
cluster 4
Total Values: 4
```

Performing Centroid(Ward) Linkage Model on 3 Components

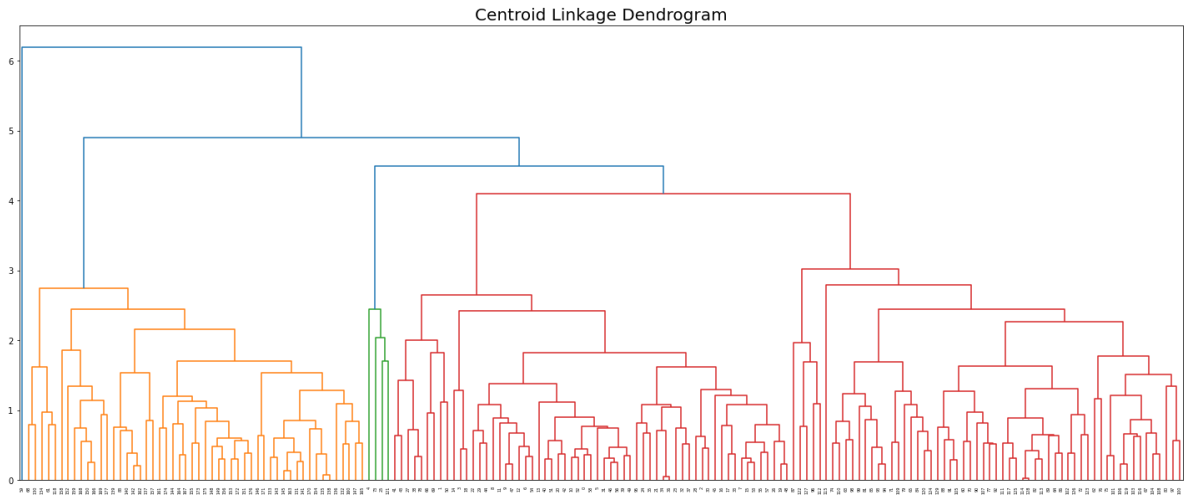
In [85]:

```
#Plotting the dendogram plot for Centroid Linkage Model
fig=plt.figure(figsize=(25,10))
dendrogram = sch.dendrogram(sch.linkage(PCA_wine, method='average'))
plt.title("Centroid Linkage Dendrogram",size=20)
```

executed in 25.3s, finished 14:48:28 2022-01-12

Out[85]:

```
Text(0.5, 1.0, 'Centroid Linkage Dendrogram')
```



In [86]:

```
#Building Centroid Linkage model
PCA_hc4= AgglomerativeClustering(n_clusters=3, affinity = 'euclidean', linkage = 'ward')
```

executed in 7ms, finished 14:48:38 2022-01-12

In [87]:

```
#Fitting the model and Creating the cluster column for the built model
y_PCAhc4 = PCA_hc4.fit_predict(PCA_wine)
Clusters=pd.DataFrame(y_PCAhc4,columns=['Clusters'])

wine_data['cluster'] = y_PCAhc4
```

executed in 25ms, finished 14:48:55 2022-01-12

In [88]:

```
#Checking how many values fall under each of the clusters created
for i in range(3):
    print("cluster", i)
    print("Total Values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 22ms, finished 14:49:18 2022-01-12

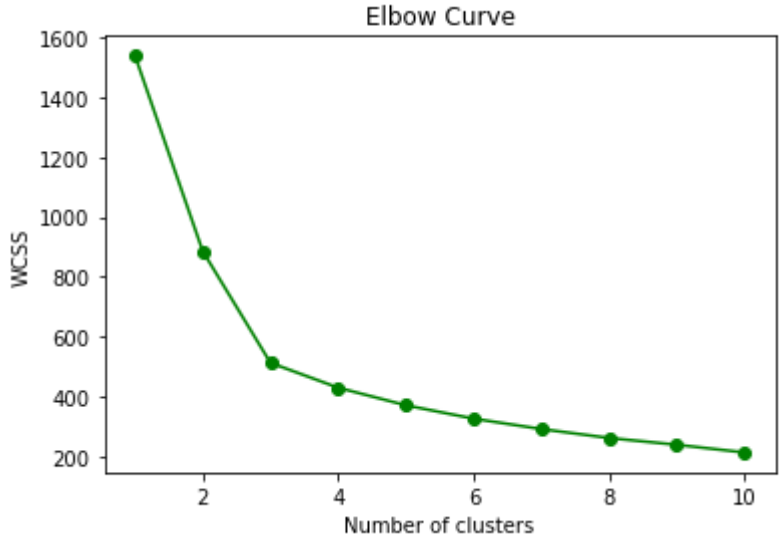
```
cluster 0
Total Values: 66
cluster 1
Total Values: 47
cluster 2
Total Values: 65
```

Performing Kmeans Clustering on 3 Components

In [89]:

```
#Plotting an elbow curve to check for k value
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i,random_state=0)
    kmeans.fit(PCA_wine)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss,'bo-',color='g')
plt.title('Elbow Curve')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

executed in 2.80s, finished 14:51:09 2022-01-12



In [90]:

```
#Building K-means Clustering model and fitting the data into it
from sklearn.cluster import KMeans
clusters_new = KMeans(3, random_state=42)
clusters_new.fit(PCA_wine)
```

executed in 113ms, finished 14:51:51 2022-01-12

Out[90]:

```
KMeans(n_clusters=3, random_state=42)
```

In [91]:

```
#Creating labels and then creating new column to put into our dataset
KMeans_pca_label=clusters_new.labels_
wine_data['cluster'] = clusters_new.labels_
```

executed in 99ms, finished 14:52:14 2022-01-12

In [92]:

```
#Checking how many values fall under each of the clusters created
for i in range(3):
    print("cluster", i)
    print("Total values:", len(list(wine_data[wine_data['cluster'] == i]['Type'].values)))
```

executed in 29ms, finished 15:00:19 2022-01-12

```
cluster 0
Total values: 51
cluster 1
Total values: 62
cluster 2
Total values: 65
```

Evaluating PCA Clustering methods with the help of Silhouette Score

The Silhouette Score metric calculates the goodness of the clustering techniques and it ranges from -1 to 1.

1: Means clusters are well apart from each other and clearly distinguised.

0:Means the distance between clusters is not significant

-1:Means clusters are assigned in the wrong way

In [93]:

```
#Silhouette Score of Single Linkage Method
PCA_Silhou_SLM=metrics.silhouette_score(PCA_wine,y_PCAhc1)
PCA_Silhou_SLM
```

executed in 45ms, finished 15:01:42 2022-01-12

Out[93]:

```
0.36310673051041414
```

In [94]:

```
#Silhouette Score of Complete Linkage Method
PCA_Silhou_ComLM=metrics.silhouette_score(PCA_wine,y_PCAhc2)
PCA_Silhou_ComLM
```

executed in 81ms, finished 15:01:54 2022-01-12

Out[94]:

```
0.35784842685673063
```

In [95]:

```
#Silhouette Score of Average Linkage Method
PCA_Silhou_ALM=metrics.silhouette_score(PCA_wine,y_PCAhc3)
PCA_Silhou_ALM
```

executed in 132ms, finished 15:02:12 2022-01-12

Out[95]:

```
0.44654492780235827
```

In [96]:

```
#Silhouette Score of Centroid(ward) Linkage Method
PCA_Silhou_CenLM=metrics.silhouette_score(PCA_wine,y_PCAhc4)
PCA_Silhou_CenLM
```

executed in 90ms, finished 15:02:36 2022-01-12

Out[96]:

```
0.44594921980629704
```

In [97]:

```
#Silhouette Score of Kmeans Clustering
PCA_Silhou_KMeans=metrics.silhouette_score(PCA_wine,KMeans_pca_label)
PCA_Silhou_KMeans
```

executed in 75ms, finished 15:03:20 2022-01-12

Out[97]:

```
0.4537999848257617
```

In [98]:

```
#Listing into the table
Table1={'Model':pd.Series(['PCA_SingleLinakge','PCA_CompleteLinkage','PCA_AverageLinkage','
  'PCA_Silhouette score':[PCA_Silhou_SLM,PCA_Silhou_ComLM,PCA_Silhou_ALM,PCA_Silhou_CenLM,P
  }
Table1=pd.DataFrame(Table1)
Table1
```

executed in 91ms, finished 15:03:32 2022-01-12

Out[98]:

	Model	PCA_Silhouette score
0	PCA_SingleLinakge	0.363107
1	PCA_CompleteLinkage	0.357848
2	PCA_AverageLinkage	0.446545
3	PCA_CentroidLinkage	0.445949
4	PCA_KMeans	0.453800

In [102]:

```
#Tabulating the clustering silhouette score and PCA Clustered Silhouette score
Final={'Model_HC':pd.Series(['HC_SingleLinakge','HC_AverageLinkage','HC_CentroidLinkage','K
  'HC_Silhouette score':[Silhou_SLM,Silhou_Average,Silhou_CenLM,Silhou_KMeans],
  'Mode1_PCA':['PCA_SingleLinakge','PCA_AverageLinkage','PCA_CentroidLinkage','PCA_KMeans']
  'PCA_Silhouette score':[PCA_Silhou_SLM,PCA_Silhou_ALM,PCA_Silhou_CenLM,PCA_Silhou_KMeans]
  }
Final=pd.DataFrame(Final)
Final
```

executed in 27ms, finished 15:05:17 2022-01-12

Out[102]:

	Model_HC	HC_Silhouette score	Mode1_PCA	PCA_Silhouette score
0	HC_SingleLinakge	0.193825	PCA_SingleLinakge	0.363107
1	HC_AverageLinkage	0.229458	PCA_AverageLinkage	0.446545
2	HC_CentroidLinkage	0.277444	PCA_CentroidLinkage	0.445949
3	KMeans	0.284859	PCA_KMeans	0.453800

Insights Drawn:

From the above table it depicts that before performing PCA the cluster result is similar. But, after performing PCA, we gain a double silhohuette score, which means that the complexity or overlaping of data is decreased after performing PCA.

In [ ]: