In [1]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("notebook")
#sns.set_context("poster")
```

In [2]:

```
# For this exercise you need to install yellowbrick
# it will be also useful to take a look at the documentation
#    scikit-yb.org
#
# ! pip install yellowbrick

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from sklearn import preprocessing

from sklearn.datasets import load_wine
from sklearn.datasets import load_boston

from sklearn.cluster import KMeans
from sklearn.cluster import MiniBatchKMeans
from yellowbrick.cluster import KElbowVisualizer
from yellowbrick.cluster import SilhouetteVisualizer
from yellowbrick.cluster import InterclusterDistance
from yellowbrick.model_selection import LearningCurve
```

```
C:\Users\duart\AppData\Local\conda\conda\envs\testEnv\lib\site-packages\skle
arn\utils\deprecation.py:144: FutureWarning: The sklearn.metrics.classificat
ion module is  deprecated in version 0.22 and will be removed in version 0.2
4. The corresponding classes / functions should instead be imported from skl
earn.metrics. Anything that cannot be imported from sklearn.metrics is now p
art of the private API.
  warnings.warn(message, FutureWarning)
```

Clustering

Clustering is the most common and well-known unsupervised learning techniques. We can find clustring almost everywhere, in political campaigns, in client segmentation, ... The aglomeration of similar items is very familiar to us, Cities are clusters of people and in cities business traditionally cluster.

However, the fact that clustering is unsupervised poses some problems to validation metrics and verification. There is normally no grown truth, therefore there is not a single solution. Many times clusters, including its number, depends on the point of view of the problem and the questions that we are trying to address.

The first problem that we encounter with clustering is finding out the best number of clusters. For this the elbow method is commonly used. It performs k-means with an increasingly number of clusters finding the k (number of clusters) that minimizes intra-cluster distance.

Checking the goodness is commonly done using the Silhoutte coefficient that measures the mean intra-cluster distance relative to the nearest clusters providing an appreciation of compactness.

Clustering is very intuitive in two-dimensional spaces, but difficult to imagine in multidimensional spaces. The output of a clustering algorithm are centroids which consists of vectors with the center for each attribute. Using predict we can easily assign membership.

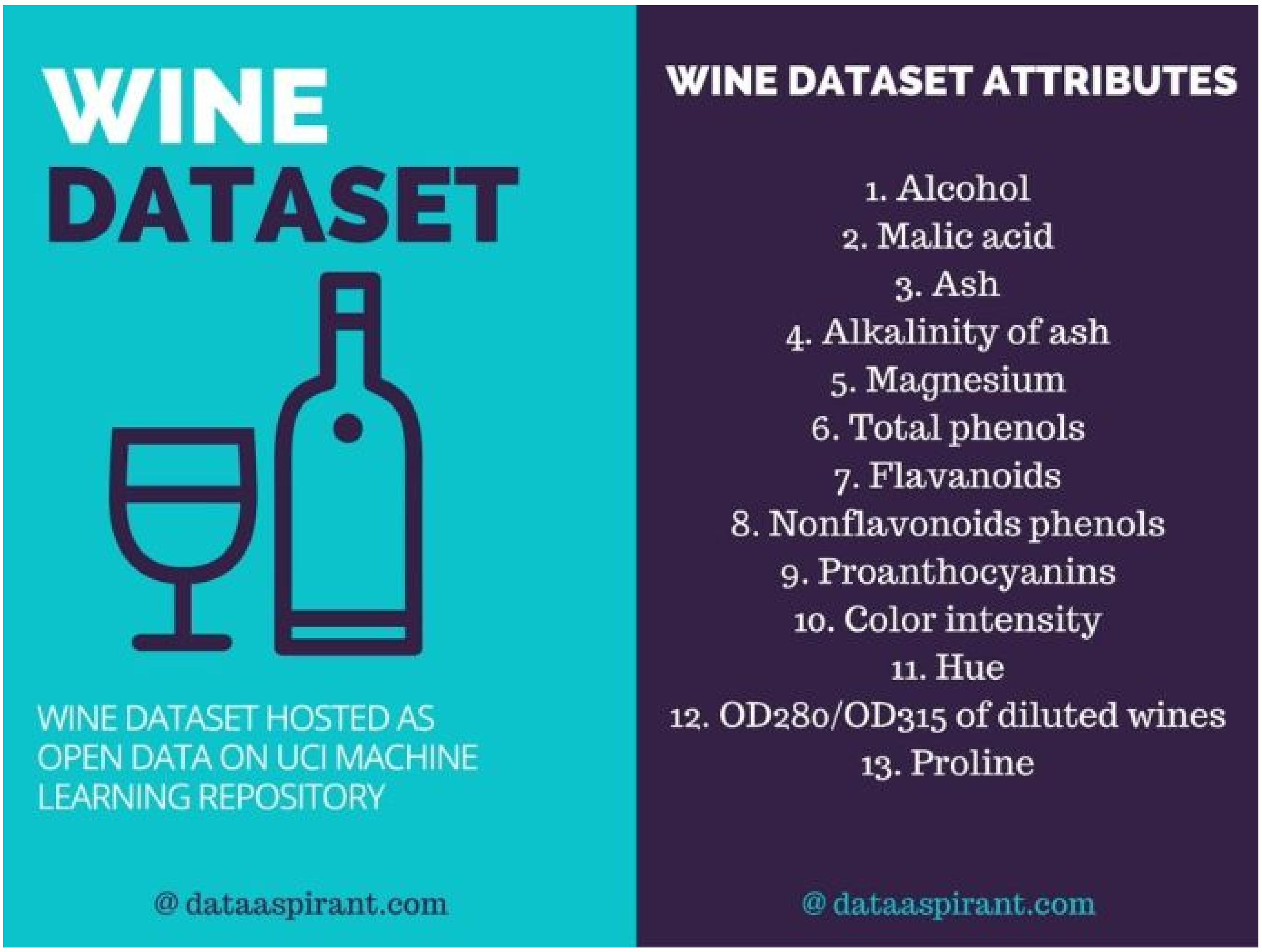
Also we aware that clustering works measuring distances, therefore we need to rescaled (normally between 0..1) all the attributes.

In this notebook we will use the UCI wine dataset, a compilation of characteristics of 178 Italian red wines divided in three families. We will see how well clustering can find without any example these three classes.

In many cases, like this one, clustering has some coincidences with classification. This is precisely this case. When this happens we know the ground truth and therefore we can apply measures such as accuracy. However, this is uncommon, and therefore assessing its goodness is difficult.

This notebook heavily uses the yellowbrick library, please install it with

pip install yellowbrick



WINE DATASET

WINE DATASET HOSTED AS OPEN DATA ON UCI MACHINE LEARNING REPOSITORY

@ dataaspirant.com

WINE DATASET ATTRIBUTES

1. Alcohol
2. Malic acid
3. Ash
4. Alkalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavonoids phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

@ dataaspirant.com

In [3]:

```
# We use the wine dataset
wine = load_wine()

print(wine["DESCR"])
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
            - Alcohol
            - Malic acid
            - Ash
            - Alcalinity of ash
            - Magnesium
            - Total phenols
            - Flavanoids
            - Nonflavanoid phenols
            - Proanthocyanins
            - Color intensity
            - Hue
            - OD280/OD315 of diluted wines
            - Proline

    - class:
            - class_0
            - class_1
            - class_2

    :Summary Statistics:

    ============================= ==== ===== ======= =====
                                   Min   Max   Mean     SD
    ============================= ==== ===== ======= =====
    Alcohol:                      11.0  14.8    13.0   0.8
    Malic Acid:                   0.74  5.80    2.34  1.12
    Ash:                          1.36  3.23    2.36  0.27
    Alcalinity of Ash:            10.6  30.0    19.5   3.3
    Magnesium:                    70.0 162.0    99.7  14.3
    Total Phenols:                0.98  3.88    2.29  0.63
    Flavanoids:                   0.34  5.08    2.03  1.00
    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
    Proanthocyanins:              0.41  3.58    1.59  0.57
    Colour Intensity:              1.3  13.0     5.1   2.3
    Hue:                          0.48  1.71    0.96  0.23
    OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
    Proline:                       278  1680     746   315
    ============================= ==== ===== ======= =====

    :Missing Attribute Values: None
    :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data (ht
tps://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data)

The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen different
measurements taken for different constituents found in the three types of
wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
  has achieved 100% correct classification.
  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
  (All results using the leave-one-out technique)

  (2) S. Aeberhard, D. Coomans and O. de Vel,
  "THE CLASSIFICATION PERFORMANCE OF RDA"
  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Journal of Chemometrics).
```

In [4]:

```
wine_df = pd.DataFrame(wine.data, columns=wine.feature_names)
wine_df['TARGET'] = wine.target
wine_df.head()

wine_df.describe()

wine_df[["TARGET", "alcohol"]].groupby("TARGET").count()

sns.distplot(wine_df["TARGET"],bins=3, kde=False)
```

Out[4]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavan
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	

Out[4]:

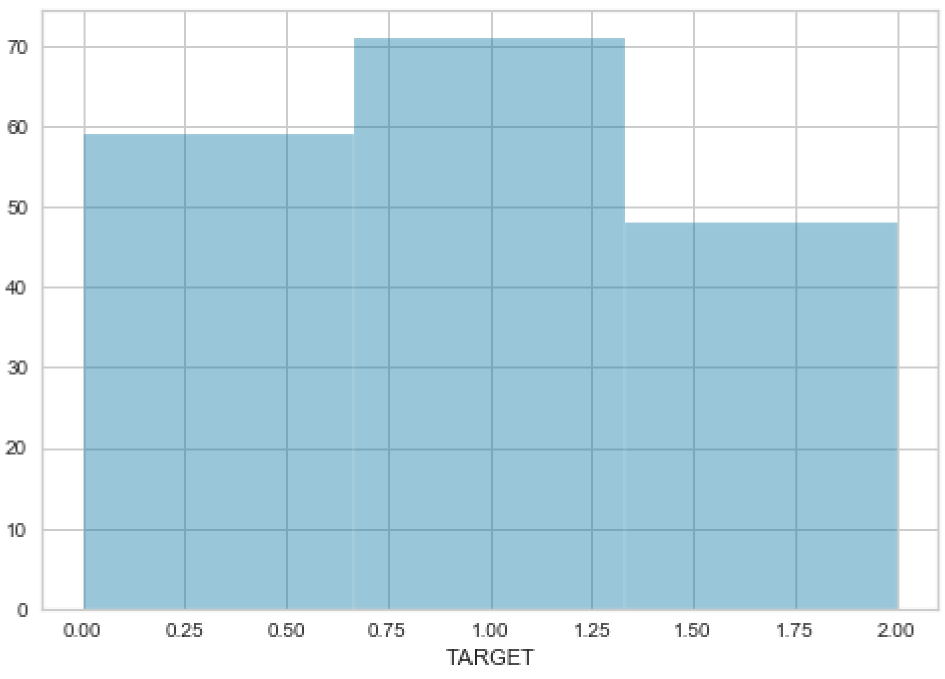
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavan
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.00
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.02
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.99
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.34
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.20
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.13
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.87
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.08

Out[4]:

	alcohol
TARGET	
0	59
1	71
2	48

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3248bbc88>
```


0
10
20
30
40
50
60
70
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
TARGET

In [5]:

```
# let's plot the alcohol percentage for each class of wine

for i in wine_df["TARGET"].unique():
    sns.distplot(wine_df["alcohol"][wine_df["TARGET"]==i], label=f'class{i:d}')
plt.legend()
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3269a4908>
```

Out[5]:

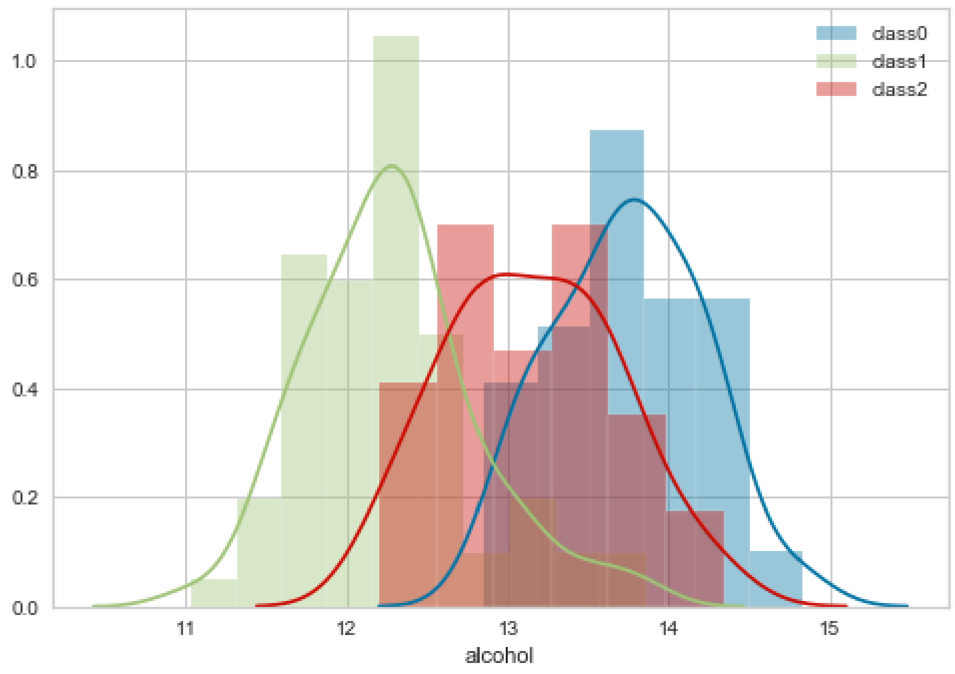
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3269a4908>
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3269a4908>
```

Out[5]:

```
<matplotlib.legend.Legend at 0x1a326a178c8>
```



In [6]:

```
# Kmeans relies in a distance metric, therefore we need to rescale all features to the same

X = wine_df.drop(['TARGET'], axis=1)
y = wine_df['TARGET']

min_max_scaler = preprocessing.MinMaxScaler()  # by default between 0 and 1

x_scaled_fit = min_max_scaler.fit(X)

x_scaled = min_max_scaler.fit_transform(X)
X_scaled = pd.DataFrame(x_scaled,columns=X.columns)
X_scaled.head()
```

Out[6]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	non
0	0.842105	0.191700	0.572193	0.257732	0.619565	0.627586	0.573840	
1	0.571053	0.205534	0.417112	0.030928	0.326087	0.575862	0.510549	
2	0.560526	0.320158	0.700535	0.412371	0.336957	0.627586	0.611814	
3	0.878947	0.239130	0.609626	0.319588	0.467391	0.989655	0.664557	
4	0.581579	0.365613	0.807487	0.536082	0.521739	0.627586	0.495781	

In [7]:

```
# Elbow method

min_max_scaler = preprocessing.MinMaxScaler()

x_scaled = min_max_scaler.fit_transform(X)
X_scaled = pd.DataFrame(x_scaled,columns=X.columns)

plt.figure(figsize=(12,9))

# Now we apply KMeans
model = KMeans()

# we want first to find out how many clusters using the elbow technique
visualizer = KElbowVisualizer(model, k=(1,8))
visualizer.fit(X_scaled)
visualizer.show()


# we know that there are 3 wine types (target=[0,1,2])
# and the elbow method correctely chooses 3
```

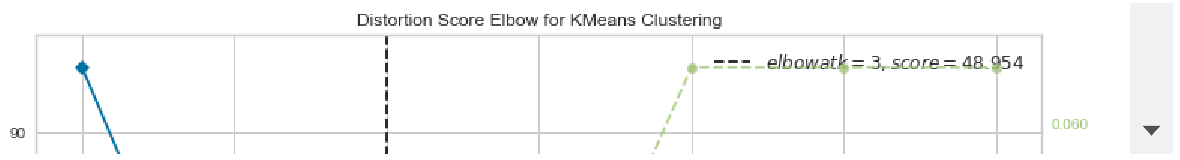
Out[7]:

```
<Figure size 864x648 with 0 Axes>
```

```
C:\Users\duart\AppData\Local\conda\conda\envs\testEnv\lib\site-packages\skle
arn\base.py:197: FutureWarning: From version 0.24, get_params will raise an
AttributeError if a parameter cannot be retrieved as an instance attribute.
Previously it would return None.
  FutureWarning)
```

Out[7]:

```
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x00000
1A326AC7208>,
                 k=None, locate_elbow=True, metric='distortion', model=None,
                 timings=True)
```


Distortion Score Elbow for KMeans Clustering
elbowatk = 3, score = 48.954
90
0.060

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a326ac7208>
```

In [8]:

```
# Now we use Silhoutte to visualize the compactness of the clusters

plt.figure(figsize=(12,9))

#model = KMeans(3)
#model=MiniBatchKMeans(n_clusters=3, verbose=True).fit(X_scaled)

model=MiniBatchKMeans(n_clusters=3).fit(X_scaled)

visualizer = SilhouetteVisualizer(model, colors='yellowbrick')
visualizer.fit(X_scaled)
visualizer.show()
```

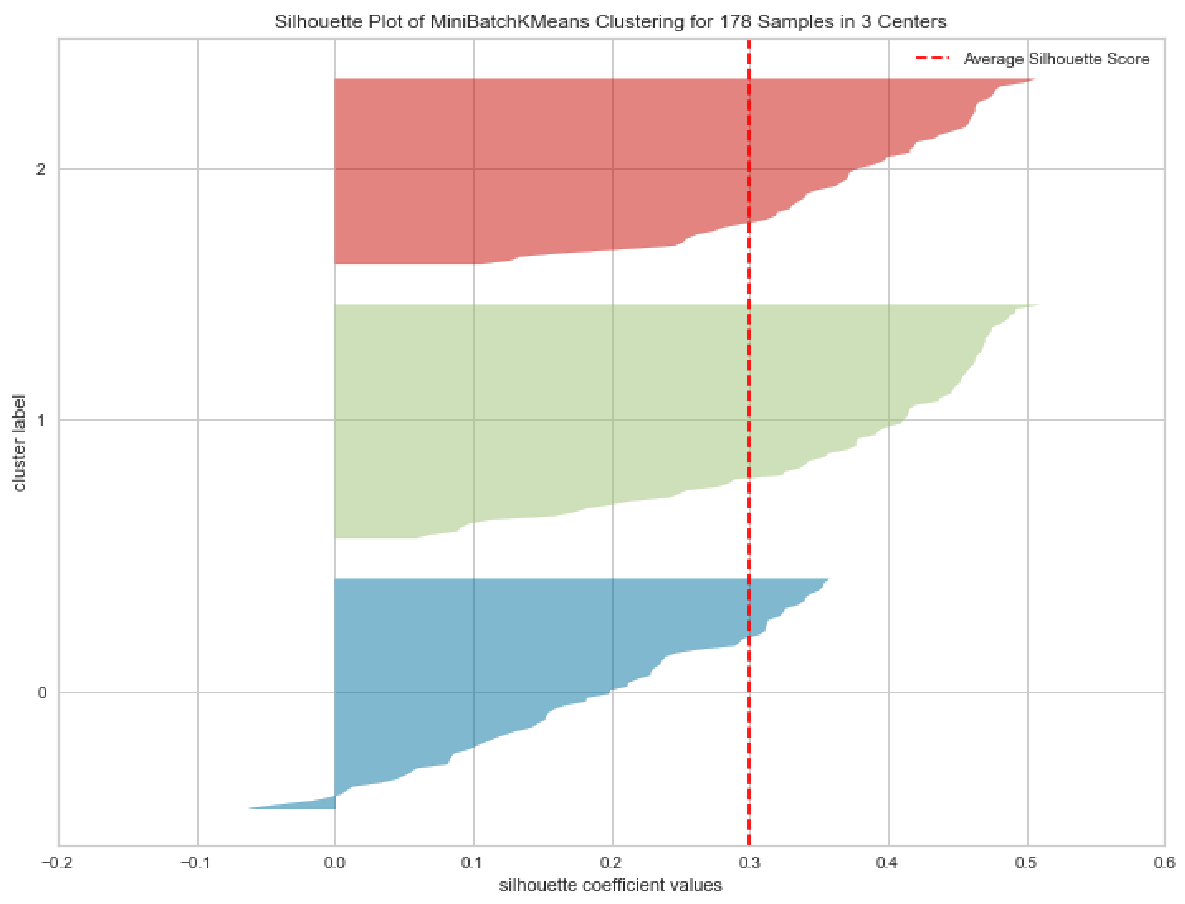
Out[8]:

```
<Figure size 864x648 with 0 Axes>
```

```
C:\Users\duart\AppData\Local\conda\conda\envs\testEnv\lib\site-packages\skle
arn\base.py:197: FutureWarning: From version 0.24, get_params will raise an
AttributeError if a parameter cannot be retrieved as an instance attribute.
Previously it would return None.
  FutureWarning)
```

Out[8]:

```
SilhouetteVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x0
00001A326E9E708>,
                     colors='yellowbrick', is_fitted='auto', model=None)
```



Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a326e9e708>
```

In [9]:

```
# Inter cluster distance

plt.figure(figsize=(12,9))

#model=MiniBatchKMeans(n_clusters=3).fit(X_scaled)

visualizer = InterclusterDistance(model, min_size=10000)
#visualizer = InterclusterDistance(model)
visualizer.fit(X_scaled)
visualizer.show()
```

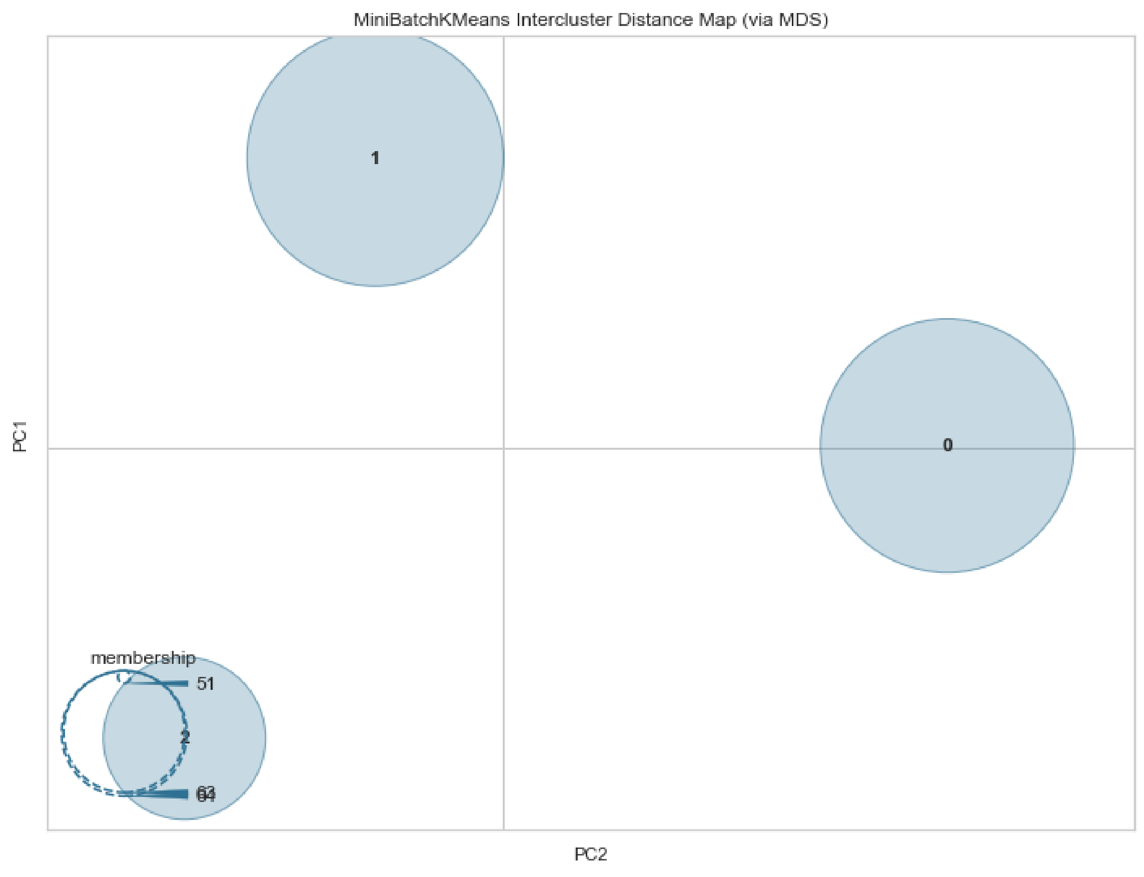
Out[9]:

```
<Figure size 864x648 with 0 Axes>
```

```
C:\Users\duart\AppData\Local\conda\conda\envs\testEnv\lib\site-packages\skle
arn\base.py:197: FutureWarning: From version 0.24, get_params will raise an
AttributeError if a parameter cannot be retrieved as an instance attribute.
Previously it would return None.
  FutureWarning)
```

Out[9]:

```
InterclusterDistance(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x0
00001A326ABF888>,
                     embedding='mds', is_fitted='auto', legend=True,
                     legend_loc='lower left', legend_size=1.5, max_size=2500
0,
                     min_size=10000, model=None, random_state=None,
                     scoring='membership')
```



```
Out[9]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a326abf888>
```

In [10]:

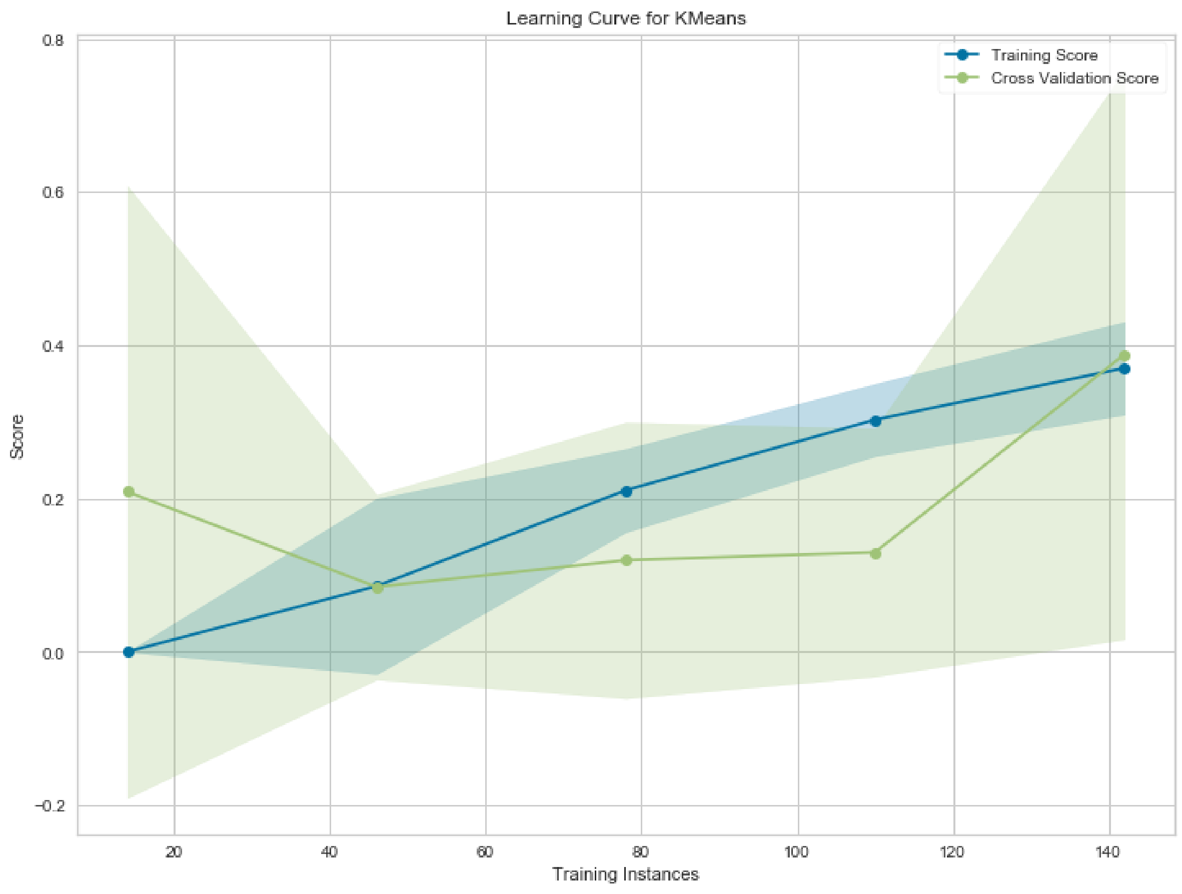
```
# Learning Curve

plt.figure(figsize=(12,9))

model = KMeans()

visualizer = LearningCurve(model, scoring="adjusted_rand_score")

visualizer.fit(X_scaled, y)        # Fit the data to the visualizer
visualizer.show()                  # Finalize and render the figure
```

Out[10]:

```
<Figure size 864x648 with 0 Axes>

C:\Users\duart\AppData\Local\conda\conda\envs\testEnv\lib\site-packages\skle
arn\base.py:197: FutureWarning: From version 0.24, get_params will raise an
AttributeError if a parameter cannot be retrieved as an instance attribute.
Previously it would return None.
  FutureWarning)
```

Out[10]:

```
LearningCurve(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x000001A3
26BA9388>,
              cv=None, exploit_incremental_learning=False, groups=None,
              model=None, n_jobs=1, pre_dispatch='all', random_state=None,
              scoring='adjusted_rand_score', shuffle=False,
              train_sizes=array([0.1  , 0.325, 0.55 , 0.775, 1.   ]))
```



Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a326ba9388>
```

In [11]:

```python
# Can we calculate accuracy ???

model=KMeans(3)
model=MiniBatchKMeans(n_clusters=3)

model.fit(X_scaled)

print("Predicted labels ----")
print(model.predict(X_scaled))
print()
print("Actual    labels ----")
print(y.values)
print()

print(" ---- Watch the labels ----")

acc_score=accuracy_score(y.values,model.predict(X_scaled))
print(f'Accuracy {acc_score*100:.3f}')
```

Out[11]:

```
MiniBatchKMeans(batch_size=100, compute_labels=True, init='k-means++',
                init_size=None, max_iter=100, max_no_improvement=10,
                n_clusters=3, n_init=3, random_state=None,
                reassignment_ratio=0.01, tol=0.0, verbose=0)
```

```
Predicted labels ----
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 1 2 2 2 2 2 2 2 2 1 2 2 0
 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 0 2 2 0 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 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]

Actual    labels ----
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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 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]

 ---- Watch the labels ----
Accuracy 35.393
```

Silhouette score

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation).

The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate.

If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

a : mean distance between a sample and all other points in the same class.
b : mean distance between a sample and all other points in the next nearest cluster.

$$s = \frac{b - a}{max(a, b}$$

In [12]:

```
# Silhouette score

from sklearn import metrics

she=metrics.silhouette_score(X_scaled, model.labels_, metric="euclidean")
print(f'Silhouette score {she:5f}')
```

```
Silhouette score 0.300809
```

In [13]:

```
# Centroids

model.labels_
model.cluster_centers_

pd.DataFrame(model.cluster_centers_, columns=X.columns)

# BECAUSE WE SCALED WE HAVE TO BRING IT BACK TO THE ORIGINAL RANGES

pd.DataFrame(x_scaled_fit.inverse_transform(model.cluster_centers_),columns=X.columns)

# --- Now with these values we can have an interpretation of what each cluster means ---
```

Out[13]:

```
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, 2, 2, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 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, 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])
```

Out[13]:

```
array([[0.69310861, 0.23556453, 0.57729866, 0.36152295, 0.40689981,
        0.64279783, 0.55265442, 0.29705938, 0.46988384, 0.35108639,
        0.47841042, 0.67744129, 0.58676687],
       [0.55347525, 0.49954684, 0.55969209, 0.54110579, 0.31470968,
        0.22866974, 0.10831161, 0.58274246, 0.2339475 , 0.51180409,
        0.16955103, 0.15860573, 0.2435647 ],
       [0.32112609, 0.21970833, 0.4706058 , 0.49218763, 0.24505604,
        0.44064323, 0.3678057 , 0.42519441, 0.38663994, 0.14852301,
        0.47549895, 0.57256537, 0.1569913 ]])
```

Out[13]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	non
0	0.693109	0.235565	0.577299	0.361523	0.406900	0.642798	0.552654	
1	0.553475	0.499547	0.559692	0.541106	0.314710	0.228670	0.108312	
2	0.321126	0.219708	0.470606	0.492188	0.245056	0.440643	0.367806	

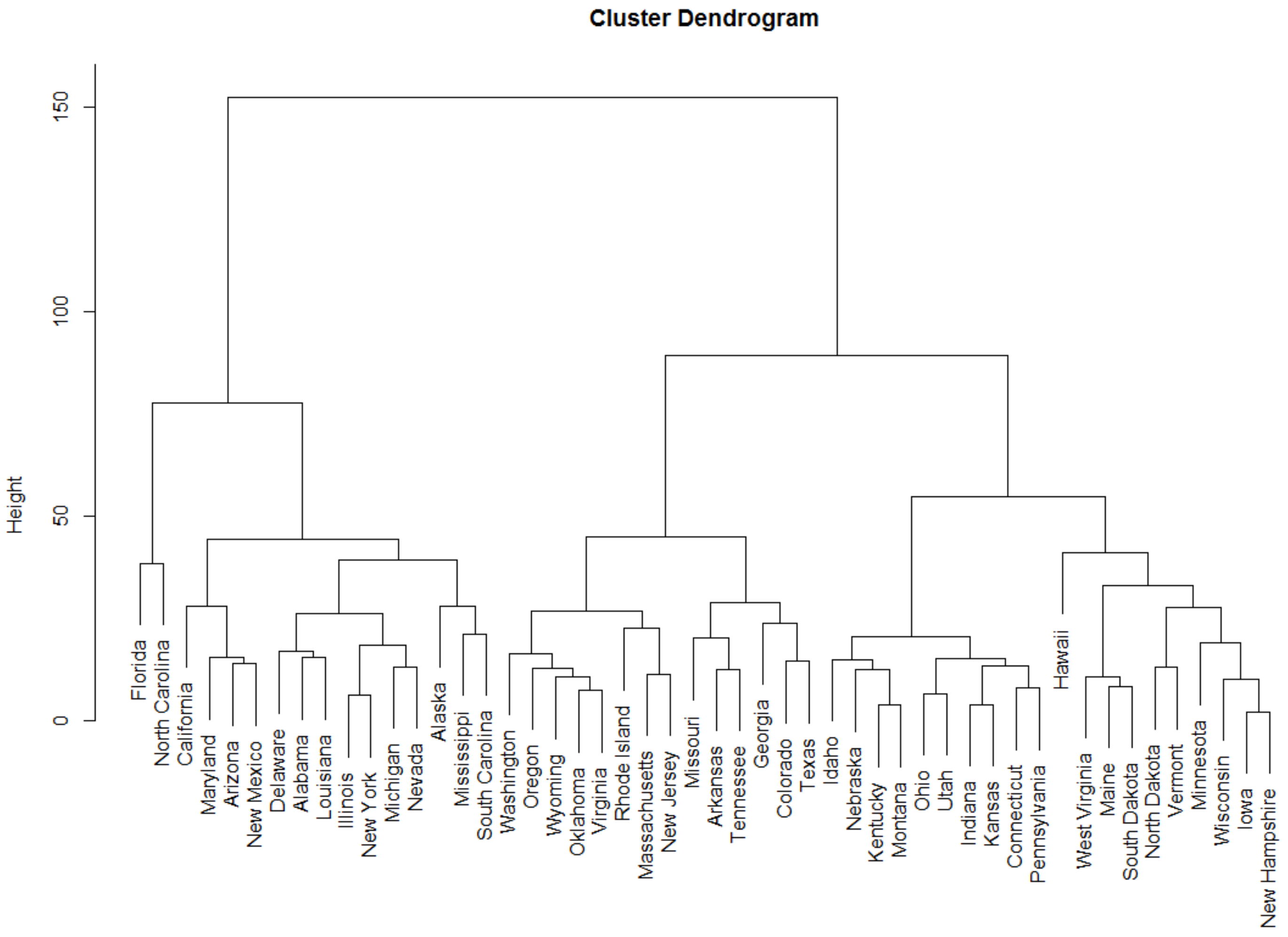
Out[13]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	no
0	13.663813	1.931957	2.439548	17.613545	107.434783	2.844114	2.959582	
1	13.133206	3.267707	2.406624	21.097452	98.953291	1.643142	0.853397	
2	12.250279	1.851724	2.240033	20.148440	92.545156	2.257865	2.083399	

Agglomerative Hierarchical Clustering

Agglomerative hierarchical clustering differs from k-means in a key way. Rather than choosing a number of clusters and starting out with random centroids, we instead begin with every point in our dataset as a “cluster.” Then we nd the two closest points and combine them into a cluster. Then, we nd the next closest points, and those become a cluster.

We repeat the process until we only have one big giant cluster.

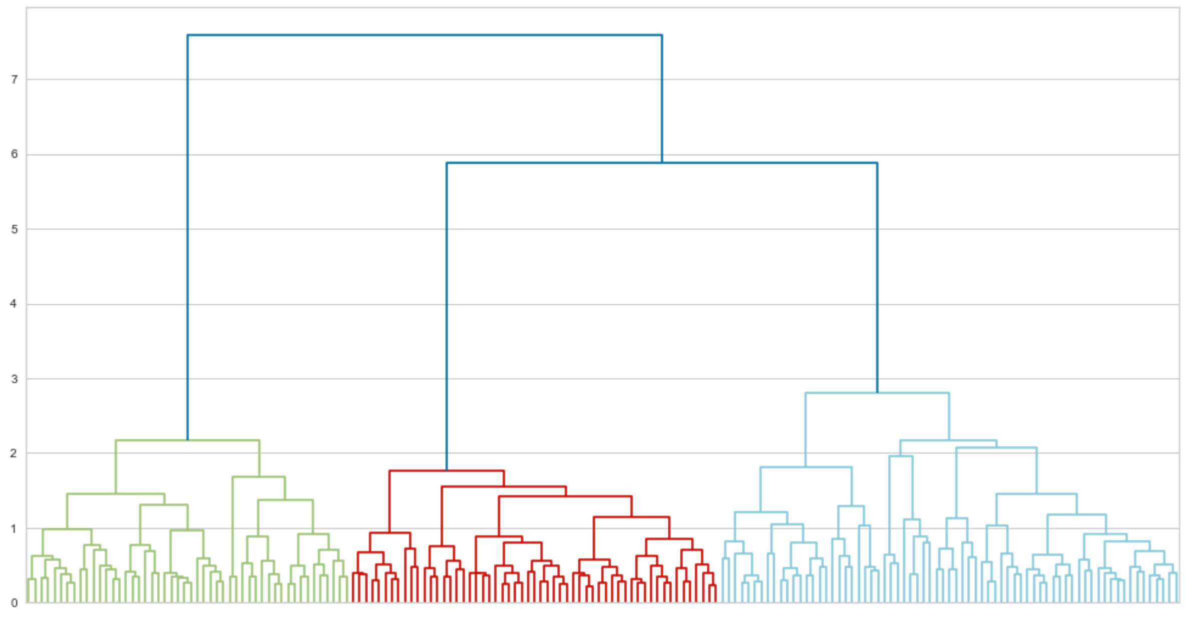


In [14]:

```
 # import hierarchical clustering libraries
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

plt.figure(figsize=(17,9))

# create dendrogram
dn = sch.dendrogram(sch.linkage(X_scaled, method='ward'), no_labels=True)
plt.show()

# create clusters
hc = AgglomerativeClustering(n_clusters=3, affinity = 'euclidean', linkage = 'ward')

# save clusters for chart y_hc = hc.fit_predict(points)
print("predictions --- ")
y_hc = hc.fit_predict(X_scaled)
y_hc

print("labels --- ")
hc.labels_

print("pre-assigned labels --- ")
y.values

dk={0:2,1:0,2:1}
acc_score=accuracy_score(list(map(lambda x:dk[x],y.values)),hc.labels_)
print(f'Accuracy {acc_score*100:.3f}')
```

Out[14]:

```
<Figure size 1224x648 with 0 Axes>
```



```
predictions ---
```

Out[14]:

```
array([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, 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, 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, 0,
       0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 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], dtype=int64)
```

```
labels ---
```

Out[14]:

```
array([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, 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, 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, 0,
       0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 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], dtype=int64)
```

```
pre-assigned labels ---
```

Out[14]:

```
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, 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, 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])
```

```
Accuracy 97.753
```

In [ ]:

In [ ]:

Mission 1

a) Cluster Europe using the EUindicators dataset and explain the clusterization using the centroids.

b) Same for the credit card dataset.

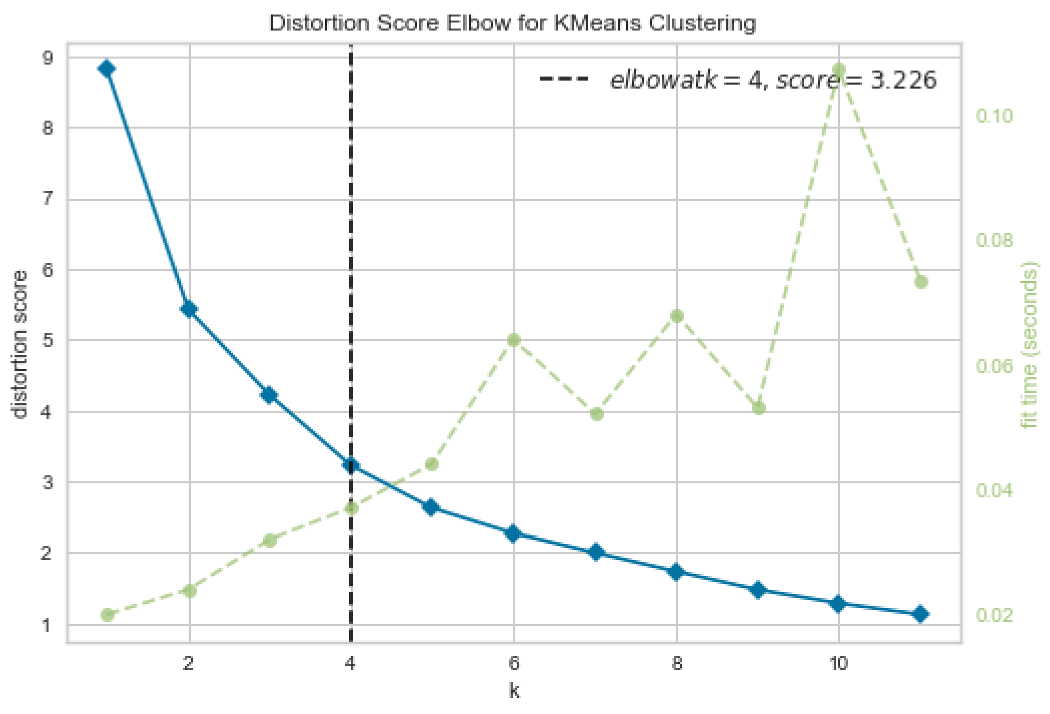
In [15]:

```
# Function to get the number of centroids

def scale_x(data):

    # Drop null values from the data:

    data.dropna(inplace = True)

    # Separate x and y

    x = data.drop([data.columns[0]], axis = 1)

    # Scale x and create a dataframe with the info

    x_scaled = min_max_scaler.fit_transform(x)

    return x_scaled


# Function to get column names

def get_columns(data):

    # Drop null values from the data:

    data.dropna(inplace = True)

    # Separate x and y

    x = data.drop([data.columns[0]], axis = 1)

    return x.columns


# Function to determine number of centroids

def cluster_centroids(data, kmeans):

    data.dropna(inplace = True)

    # Separate x and y

    x = data.drop([data.columns[0]], axis = 1)

    x_scaled = scale_x(data)

    # Using kmeans, clusterize the data

    model = KMeans(kmeans)
    model.fit(x_scaled)

    # Bring the data back to the original ranges

    x_scaled_fit = min_max_scaler.fit(x)
    centroids_rev = pd.DataFrame(x_scaled_fit.inverse_transform(model.cluster_centers_), co

    return centroids_rev
```

```
# Function to determine number of clusters to use

def elbow(data, n):

    x_scaled = scale_x(data)

    # Now we apply KMeans
    model = KMeans()

    # we want first to find out how many clusters using the elbow technique
    visualizer = KElbowVisualizer(model, k=(1,n))
    visualizer.fit(x_scaled)
    visualizer.show()


# Function for silouette scores

def sil(data, clusters):

    x_scaled = scale_x(data)

    model = MiniBatchKMeans(n_clusters = clusters)

    model.fit(x_scaled)

    sil = metrics.silhouette_score(x_scaled, model.labels_, metric="euclidean")
    print(f'Silhouette score {sil:5f}')
```

In [16]:

```
eu_data = pd.read_excel("EUIndicators-2014-2018.xlsx")

elbow(eu_data, 12)

sil(eu_data, 4)

cluster_centroids(eu_data, 4)
```



```
Silhouette score 0.245304
```

Out[16]:

	Construction Confidence Indicator	Consumer Confidence Indicator	Industrial Confidence Indicator	Retail Confidence Indicator	Service Confidence Indicator	Business Confidence Indicator (avg)
0	-10.054545	-7.581818	1.600000	10.145455	9.563636	2.813636
1	-27.900000	-18.166667	-4.983333	2.233333	4.616667	-6.508333
2	-1.650000	2.100000	4.450000	15.075000	26.800000	11.168750
3	-10.440000	-1.080000	-2.320000	-4.680000	15.580000	-0.465000

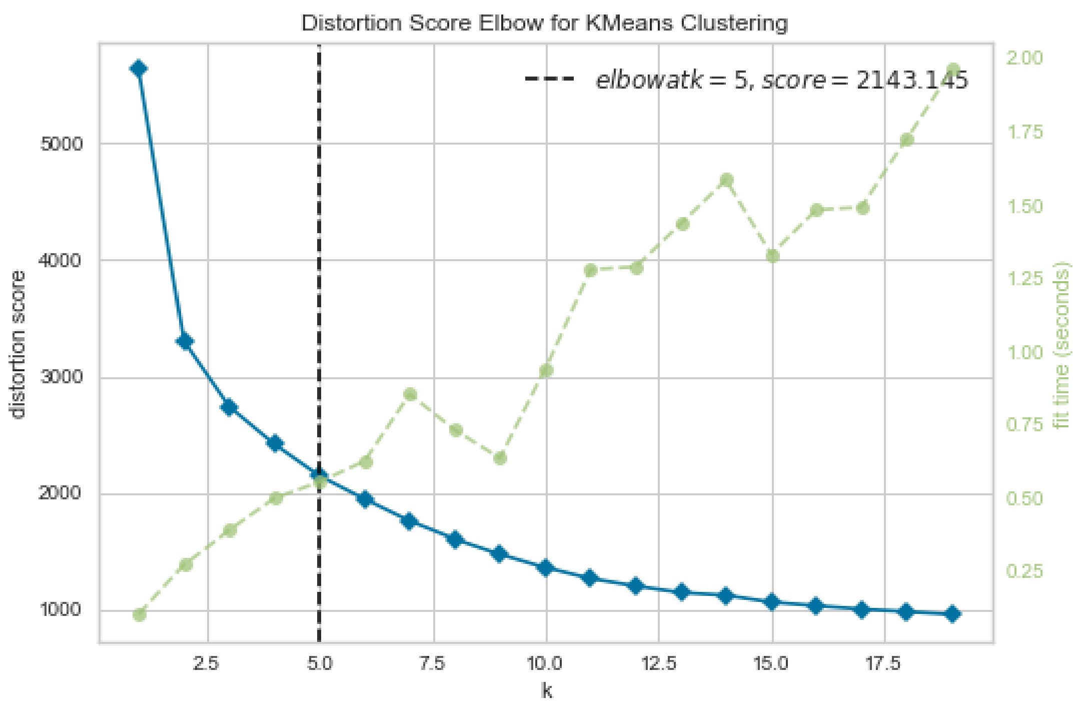
In [17]:

```
print("The first group have the highest confidence about their current and prospective econ
print("Confidence is especially high in the service and retail sectors. Also, all confidenc
print("than for the other groups. Hence, we can presume this is group is the richest.")
print("")
print("The second group has the lowest overall confidence, especially in construction and t
print("It only has positive indicators in the retail and service sectors.")
print("")
print("The third has the second highest business confidence indicators (slightly positive),
print("in the retail and service sectors.")
print("")
print("The last group has the third lowest confidence, very slightly below 0. Hence, the co
print("described as neutral. The sector with the highest confidence is the service sector."
```

```
The first group have the highest confidence about their current and prospect
ive economic status.
Confidence is especially high in the service and retail sectors. Also, all c
onfidence indicators are higher
than for the other groups. Hence, we can presume this is group is the riches
t.

The second group has the lowest overall confidence, especially in constructi
on and the consumer sectors.
It only has positive indicators in the retail and service sectors.

The third has the second highest business confidence indicators (slightly po
sitive), mostly due to high confidence
in the retail and service sectors.

The last group has the third lowest confidence, very slightly below 0. Henc
e, the confidence in this group could be
described as neutral. The sector with the highest confidence is the service
sector.
```

In [18]:

```
cc_data = pd.read_csv("CCData.csv")

elbow(cc_data, 20)

sil(cc_data, 5)

cluster_centroids(cc_data, 5)
```



```
Silhouette score 0.310512
```

Out[18]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_F
0	2192.560280	0.968621	270.112849	222.500305	
1	1638.404882	0.947600	1221.650043	392.349601	
2	1996.246376	0.975645	2921.406014	2289.860334	
3	188.761323	0.917735	1866.688564	818.322737	
4	186.185491	0.426228	373.533628	259.498171	

In [19]:

```
print("The first group has the lowest balance out of all the groups, and is the group with
print("and frequency. However, they do get significant cash advances, even though its the s
print("This group also has the lowest credit limit.")
print("")
print("The second group has the second highest balance, and the highest amount of purchases
print("group has also the highest amount of payments (since total purchases is the highest
print("this group is also the highest.")
print("")
print("This group has the highest balance of all the groups, but the lowest amount of total
print("also ask for the highest amount of cash advances and cash advance frequency. The gro
print("payments, and has an average credit limit")
print("")
print("The fourth group has an average balance, purchase total and credit limit. However, t
print("frequency.")
print("")
print("The fifth group has a very low balance, very close to the lowest, but the second hig
print("frequency, since the group also has the second highest credit limit.")
```

```
The first group has the lowest balance out of all the groups, and is the gro
up with the lowest total purchases
and frequency. However, they do get significant cash advances, even though i
ts the second lowest amongst the groups.
This group also has the lowest credit limit.

The second group has the second highest balance, and the highest amount of p
urchases (and one-off purchases). The
group has also the highest amount of payments (since total purchases is the
highest amount). The credit limit of
this group is also the highest.

This group has the highest balance of all the groups, but the lowest amount
of total purchases and frequency. They
also ask for the highest amount of cash advances and cash advance frequency.
The group makes the lowest amount of
payments, and has an average credit limit

The fourth group has an average balance, purchase total and credit limit. Ho
wever, the group does have a high purchase
frequency.

The fifth group has a very low balance, very close to the lowest, but the se
cond highest purchase total, as well as
frequency, since the group also has the second highest credit limit.
```

In [ ]:

```

```