Task-6A:

The K-Means algorithm is an unsupervised learning (unsupervised learning) and clustering algorithm. For data that does not contain label information, a grouping process is performed using the clustering method.

The purpose of this algorithm is to divide the observations into clusters according to their similarity to each other. The clusters formed are expected to be homogeneous within themselves and heterogeneous with respect to each other.

Clustering methods are divided into hierarchical and non-hierarchical clustering methods. From this point of view, k-means is a non-hierarchical clustering method. The main purpose of these two approaches is to try to make the similarity high within the clusters and low between the clusters. It is especially used in segmentation problems.

While applying the K-means algorithm, the following steps are applied in order:

- The number of clusters k is determined.
- k centers are randomly selected.
- For each observation, the distances to the k centers are calculated and the observations are assigned to the nearest center k.
- Each observation is assigned to the closest center, the cluster.
- Center calculations are made again for the clusters formed after the assignment operations.
- This process is repeated for a specified number of iterations, and the clustering structure of the observations in the case where the sum of intra-cluster error squares (total within cluster variation) is minimum is selected as the final cluster.

As mentioned in the steps above, it is an algorithm that works using the **Expectation-Maximization** approach. For each observation, it makes an assignment to the closest cluster. After the assignment process, it calculates the center for the clusters again. It continues this way until it gets close.

With the expectation step, observations are assigned to the nearest cluster center. Since the maximization step involves maximizing the fitness functions that define the location of each cluster centers, it tries to average the cluster centers by taking a simple average of the data in each cluster. Each iteration of this E and M steps will always provide a better estimate of the cluster properties.

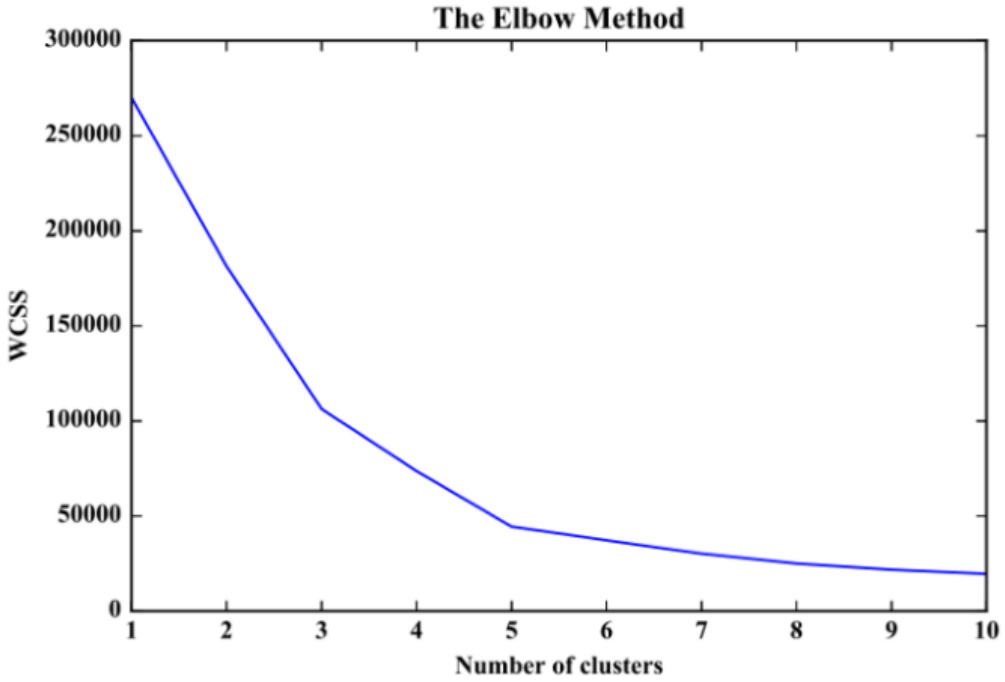
Finding the optimum number of clusters:

Elbow method can be used to find the optimum number of clusters in the K-means algorithm. This method is very important in determining the most appropriate number of clusters for the success of the WCSS (with cluster sum of squares) model. The formula used is as follows:

$$\text{WCSS} = \sum_{P_i \text{ in Cluster 1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster 2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster 3}} \text{distance}(P_i, C_3)^2$$

The formula expresses the sum of the squares of the distance from the center points of the points in that set for each cluster. The most suitable number of clusters is evaluated as the success criteria of the model created by the K-means algorithm. It can be said that the more successful the clustering is, the minimum it can keep the distance between the cluster elements and the maximum the distance between each cluster.

In fact, as the number of clusters increases, the distance between each element in the cluster will decrease. however, the WCSS value will be reduced. In this way, if we include every element in the data set to the model, the probability of ovefitting will increase. Because the model will start to memorize the dataset.



For example, in this graph above, one of the points where the slope changes suddenly can be taken as the k-point, such as the value 3. This value can be considered as the optimum number of clusters for the model using the k-means algorithm.

Pros:

- Relatively simple to apply.
- Scalable to large data sets
- It can warm the positions of the centroids.
- It can easily adapt to new examples.
- Generalizes clusters of different shapes and sizes, such as elliptical clusters.

Cons:

- Since k (number of clusters) in the model is received from the user, this number may not always be the optimum value.
- Center points assigned randomly at the beginning may not always initiate a successful clustering process. Because the algorithm is shaped according to these center points, which are randomly assigned at the beginning.
- k-means has trouble clustering data when clusters are of different sizes and densities.
- Center points can be dragged by outliers or outliers can be assigned to clusters instead of being ignored. It is very important to remove outliers before clustering.
- As the number of dimensions increases, the distance-based similarity measure converges to a fixed value among the given samples. For this reason, dimensionality reduction can be done by using PCA on the features or by changing the clustering algorithm to Spectral Clustering.

Task-6B:

In [2]:
```
from sklearn.datasets import load_sample_image
import matplotlib.pyplot as plt
china = load_sample_image("china.jpg")
ax = plt.axes(xticks=[], yticks=[])
ax.imshow(china);
```

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

Out[3]:
```
(427, 640, 3)
```

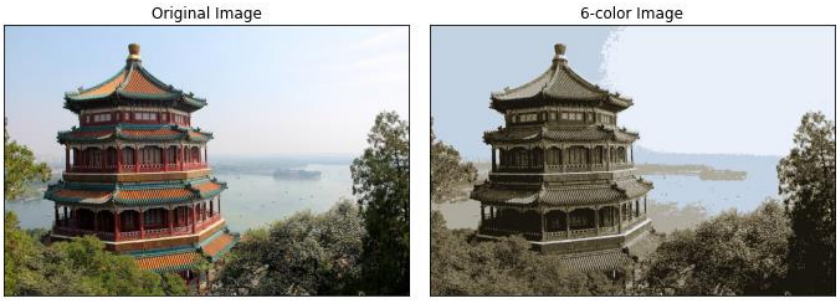
In [4]:
```
data = china / 255.0 # use 0...1 scale
data = data.reshape(427 * 640, 3)
data.shape
```

Out[4]:
```
(273280, 3)
```

K=6

In [21]:
```
import warnings; warnings.simplefilter('ignore')

from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(6)
kmeans.fit(data)
new_colors_6 = kmeans.cluster_centers_[kmeans.predict(data)]
```

In [22]:
```
china_recolored = new_colors_6.reshape(china.shape)

fig, ax = plt.subplots(1, 2, figsize=(12, 6),
                       subplot_kw=dict(xticks=[], yticks=[]))
fig.subplots_adjust(wspace=0.05)
ax[0].imshow(china)
ax[0].set_title('Original Image', size=12)
ax[1].imshow(china_recolored)
ax[1].set_title('6-color Image', size=12);
```



We can see the difference when I use K = 6, it's a colorless photo, completely far from the colors in the original picture. It turned out to be black and white.

K=10

In [23]:
```
import warnings; warnings.simplefilter('ignore')  # Fix NumPy issues.

from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(10)
kmeans.fit(data)
new_colors_10 = kmeans.cluster_centers_[kmeans.predict(data)]
```

In [24]:
```
china_recolored = new_colors_10.reshape(china.shape)

fig, ax = plt.subplots(1, 2, figsize=(12, 6),
                       subplot_kw=dict(xticks=[], yticks=[]))
fig.subplots_adjust(wspace=0.05)
ax[0].imshow(china)
ax[0].set_title('Original Image', size=12)
ax[1].imshow(china_recolored)
ax[1].set_title('10-color Image', size=12);
```

When I used K = 10, the photo got a little more colorful, but we can still say that it is far from the original photo.

K=16

```
In [25]: import warnings; warnings.simplefilter('ignore')  # Fix NumPy issues.

         from sklearn.cluster import MiniBatchKMeans
         kmeans = MiniBatchKMeans(16)
         kmeans.fit(data)
         new_colors_16 = kmeans.cluster_centers_[kmeans.predict(data)]
```

```
In [26]: china_recolored = new_colors_16.reshape(china.shape)

         fig, ax = plt.subplots(1, 2, figsize=(12, 6),
                                subplot_kw=dict(xticks=[], yticks=[]))
         fig.subplots_adjust(wspace=0.05)
         ax[0].imshow(china)
         ax[0].set_title('Original Image', size=12)
         ax[1].imshow(china_recolored)
         ax[1].set_title('16-color Image', size=12);
```

When I use K = 16, it looks a lot more like the original picture. So, we can say that there are actually 16 color sets. Some detail is certainly lost but the overall image is still easily recognizable.

K=20

```
In [27]: import warnings; warnings.simplefilter('ignore')  # Fix NumPy issues.

         from sklearn.cluster import MiniBatchKMeans
         kmeans = MiniBatchKMeans(20)
         kmeans.fit(data)
         new_colors_20 = kmeans.cluster_centers_[kmeans.predict(data)]
```

```
In [28]: china_recolored = new_colors_20.reshape(china.shape)

         fig, ax = plt.subplots(1, 2, figsize=(12, 6),
                                subplot_kw=dict(xticks=[], yticks=[]))
         fig.subplots_adjust(wspace=0.05)
         ax[0].imshow(china)
         ax[0].set_title('Original Image', size=12)
         ax[1].imshow(china_recolored)
         ax[1].set_title('20-color Image', size=12);
```

When I used K = 20, it looks closer clearly almost like the original picture.

K=30

In [32]:
```
import warnings; warnings.simplefilter('ignore')  # Fix NumPy issues.

from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(30)
kmeans.fit(data)
new_colors_30 = kmeans.cluster_centers_[kmeans.predict(data)]
```

In [33]:
```
china_recolored = new_colors_30.reshape(china.shape)

fig, ax = plt.subplots(1, 2, figsize=(12, 6),
                       subplot_kw=dict(xticks=[], yticks=[]))
fig.subplots_adjust(wspace=0.05)
ax[0].imshow(china)
ax[0].set_title('Original Image', size=12)
ax[1].imshow(china_recolored)
ax[1].set_title('30-color Image', size=12);
```

In this clustering, I defined the k value as 30 higher than the others. When we look at the two photos, there is almost no difference. The same can be said with the original photo. With K = 30, it means that there are actually 30 different clusters.

Some detail is might be lost in the 30-color image, but the overall image is still easily recognizable.

Task-6C:

Gaussian mixture model (GMM) is an Unsupervised clustering model whose dataset is described as a mixture of k-pieces Gaussian distribution under the assumption of normality.

It is the optimization of the Gaussian parameters of these sources to maximize the probability density function of the mixture, assuming that the sources of the examples given are k-Gaussian distributions. Thus, GMM is successful even at the points where the modeling algorithms are at a deadlock, assuming that the dataset is produced from a single distribution.

GMM vs K-means

- In k-means clustering, given an untagged dataset, we try to group the samples into k clusters. In GMM algorithms, it is assumed that different subpopulations of the data set follow a normal distribution and we only have information about the probability distribution of the general population.
- Since K-Means uses some kind of distance function and the distance from the cluster center is measured, it usually does not work when clusters are not round in shape. GMM doesn't make it circular like in k-mean clustering algorithms. Clusters are more prone to ellipses.
- Probability-based approach gives probability of belonging to all clusters of each sample in the dataset
- The GMM approach is similar to the K-means clustering algorithm, but it is more robust and therefore more useful due to its complexity.
- GMM capture works better because it captures the uncertainty of data points belonging to different clusters by using soft-assignments and it does not have bias for circular clusters. Hence, it works well even with nonlinear data distributions.

In K-means clustering, it is a clustering method that aims to find the locations of clusters that minimize the Euclidian distance from the data points to the cluster by minimizing the distance loss function. This means that a point belongs entirely to a cluster or does not exist at all.

In the GMM probabilistic approach, each cluster is defined according to the center of gravity (average), covariance and the size of the cluster (Weight). Rather than the clusters are defined by their 'closest' weighted center, a k gauss cluster is created to the data. Afterwards, gaussian distribution parameters such as mean and variance for each cluster and the weight of a cluster are estimated. After the parameters for each data point are learned, their probability of belonging to each cluster is calculated. Mathematically, the gaussian mixture model can be defined as the mixture of k gaussian distribution. this means that it is the weighted average of the k gaussian distribution.

Expectation-Maximization (EM) Algorithm

The Expectation-Maximization (EM) algorithm is an iterative algorithm that finds maximum likelihood estimates for parameters when there are missing observations or some hidden variables of the sample.

Expectation Step: The best probabilities for the unknown data are predicted with the random means and variances for the k-number set.

Maximization Step: New estimates of the parameters are obtained by substituting the estimated missing value and calculating the maximum likelihood over the data.

The algorithm starts with a value, this initial value is either chosen arbitrarily or determined with the help of other clustering algorithms. These steps are performed sequentially until a certain criterion is met or the maximum number of iterations is reached.

PROS

- Soft clustering
- Clustering: Captures non-spherical cluster distributions
- Density estimation: Captures multimodal distribution
- EM (expectation-maximization) algorithm for GMM always converges and often works well
- It can be used easily while generating new data.
- It Works successfully when they cannot model k-means
- It has elliptical approaches instead of circular and spherical clustering.

CONS

- Requires defining number of clusters
- EM algorithm is dependent on initialization
- Uses all the components it has access to, so initialization of clusters will be difficult when dimensionality of data is high.
- Difficult to interpret.

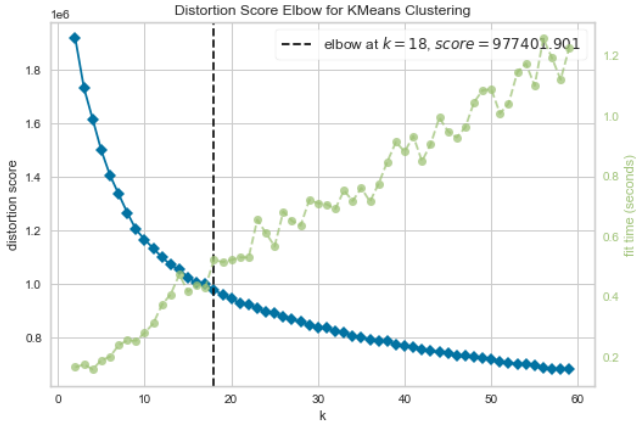
Task-6D:

K-means

In [92]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

Out[92]: (1797, 64)

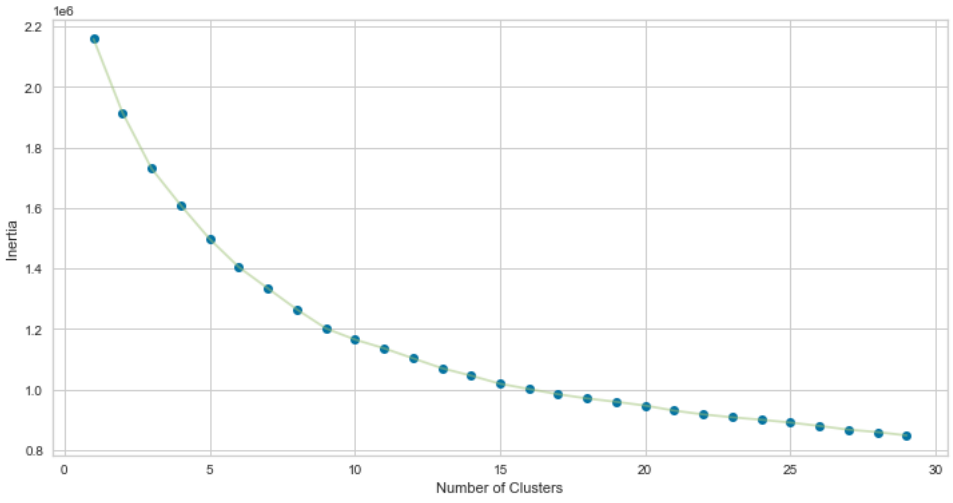
In [85]:
```
from yellowbrick.cluster import KElbowVisualizer
kmeans=KMeans()
visualizer = KElbowVisualizer(kmeans, k=(2,60))
visualizer.fit(X)
visualizer.poof()
#distortion score düşük olması bizim için iyi
```



Out[85]: <AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>

In [110]:
```
inertia = []
for n in range(1 , 30):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    algorithm.fit(X)
    inertia.append(algorithm.inertia_)
```

In [143]:
```
plt.figure(1 , figsize = (12 ,6))
plt.plot(np.arange(1 , 30) , inertia , 'o')
plt.plot(np.arange(1 , 30) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.show()
```



In [136]:
```
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits.data)
kmeans.cluster_centers_.shape
```

Out[136]:
```
(10, 64)
```

In [137]:
```
from scipy.stats import mode

labels = np.zeros_like(clusters)
for i in range(10):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]
```

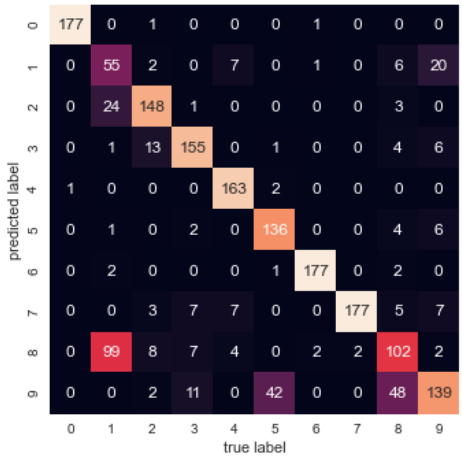
In [141]:
```
labels
```

Out[141]:
```
array([0, 8, 8, ..., 8, 9, 9])
```

In [142]:
```
from sklearn.metrics import classification_report, accuracy_score
report = classification_report(digits.target, labels)
score = accuracy_score(y_true= digits.target, y_pred=labels)
print(report)
print("{} {:0.2f}%".format("Accuracy Score : ", score*100))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       178
           1       0.60      0.30      0.40       182
           2       0.84      0.84      0.84       177
           3       0.86      0.85      0.85       183
           4       0.98      0.90      0.94       181
           5       0.91      0.75      0.82       182
           6       0.97      0.98      0.98       181
           7       0.86      0.99      0.92       179
           8       0.45      0.59      0.51       174
           9       0.57      0.77      0.66       180

    accuracy                           0.80      1797
   macro avg       0.80      0.80      0.79      1797
weighted avg       0.81      0.80      0.79      1797

Accuracy Score :  79.52%
```

In [140]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(digits.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=digits.target_names,
            yticklabels=digits.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



Accuracy score: The ratio of correctly predicted values to the total number of predictions

0.79 = 177+55+148+155+163+136+177+177+102+139 / 1797

For label 1 : Precision value = How many of the values labeled as 1 really are 1.

0.60 = 55 / 55+2+7+1+6+20

For label 1: Recall value = How many of the values with the label 1 was correctly predicted by the model?

0.30 = 55 / 55+24+1+1+2+99

GMM

In [186]:
```
from sklearn import mixture
gmm = mixture.GaussianMixture(n_components=10, covariance_type='full', random_state=0).fit(digits.data)
clusters = gmm.predict(digits.data)
```

In [187]:
```
from scipy.stats import mode

labels = np.zeros_like(clusters)
for i in range(13):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]
```

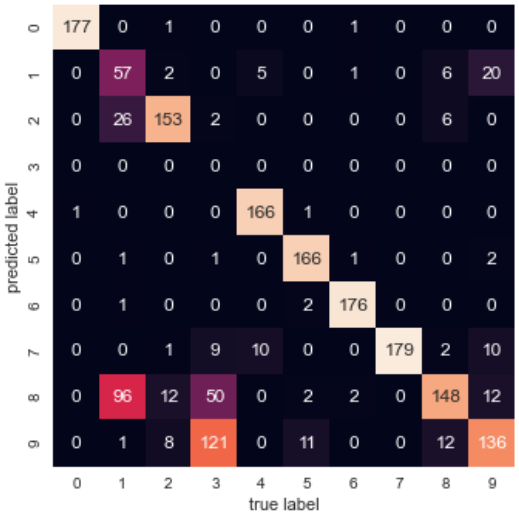
In [188]:
```
from sklearn.metrics import classification_report, accuracy_score
report = classification_report(digits.target, labels)
score = accuracy_score(y_true= digits.target, y_pred=labels)
print(report)
print("{} {:0.2f}%".format("Accuracy Score : ", score*100))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       178
           1       0.63      0.31      0.42       182
           2       0.82      0.86      0.84       177
           3       0.00      0.00      0.00       183
           4       0.99      0.92      0.95       181
           5       0.97      0.91      0.94       182
           6       0.98      0.97      0.98       181
           7       0.85      1.00      0.92       179
           8       0.46      0.85      0.60       174
           9       0.47      0.76      0.58       180

    accuracy                           0.76      1797
   macro avg       0.72      0.76      0.72      1797
weighted avg       0.71      0.76      0.72      1797

Accuracy Score :  75.57%
```

When we run the GMM model for the k = 10 value in the K-means model, we see that the accuracy score value comes to 75.57%. In the K-means model, the accuracy score values was the 79.52%.

For label 3, When we examine this difference, GMM model did not find any precision, recall and f1-score values. So, we can say that no value has been assigned to cluster 3.

In [189]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(digits.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=digits.target_names,
            yticklabels=digits.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



When we want to find the Explained variance ratio of 0.79, the number of components required for the model is 13.

In [182]:
```
from sklearn.decomposition import PCA
pca = PCA(0.79, whiten=True)
data = pca.fit_transform(digits.data)
data.shape
```

Out[182]:
```
(1797, 13)
```

In [190]:
```
from sklearn import mixture
gmm = mixture.GaussianMixture(n_components=13, covariance_type='full', random_state=0).fit(digits.data)
clusters = gmm.predict(digits.data)
```

In [193]:
```
from scipy.stats import mode

labels = np.zeros_like(clusters)
for i in range(13):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]
```

When we find the required number of components using PCA, the features that will input the model are not the same. Feature extraction was performed with PCA.

In [194]:
```
from sklearn.metrics import classification_report, accuracy_score
report = classification_report(digits.target, labels)
score = accuracy_score(y_true= digits.target, y_pred=labels)
print(report)
print("{} {:0.2f}%".format("Accuracy Score : ", score*100))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       178
           1       0.78      0.84      0.81       182
           2       0.84      0.86      0.85       177
           3       0.59      0.37      0.45       183
           4       0.98      0.83      0.90       181
           5       0.94      0.97      0.95       182
           6       0.98      0.97      0.98       181
           7       0.80      1.00      0.89       179
           8       0.75      0.46      0.57       174
           9       0.38      0.58      0.46       180

    accuracy                           0.79      1797
   macro avg       0.80      0.79      0.79      1797
weighted avg       0.80      0.79      0.79      1797

Accuracy Score :  78.80%
```

In the model set up with n = 10, no value could be predicted for cluster 3.

When we look at the output of the model established with n = 13, values are assigned to the cluster number 3.

The model correctly labeled 60% of the values it labelled as 3.