## 7BUIS008W- Unsupervised Learning

## Clustering (KMeans Vs Hierarchical)

# Example-1 KMeans Clustering

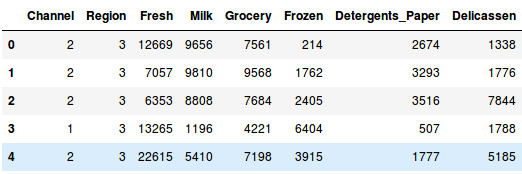
We will be working on a wholesale customer segmentation problem. You can download the dataset using [this link](https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%20customers%20data.csv). The data is hosted on the UCI Machine Learning repository.

**The aim of this problem is to segment the clients of a wholesale distributor based on their annual spending on diverse product categories, like milk, grocery, region, etc.** So, let’s start coding!

We will first import the required libraries:

|  |  |
| --- | --- |
|  | # importing required libraries |
|  | import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  import pandas\_profiling  # pip install pandas-profiling  # Using Anaconda Power Shell Prompt  from sklearn.cluster import KMeans  from sklearn.preprocessing import StandardScaler |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  | # reading the data and looking at the first five rows of the data |
|  | data=pd.read\_csv('/…./…./……/Wholesale-customers-data.csv') |
|  | data.head() |

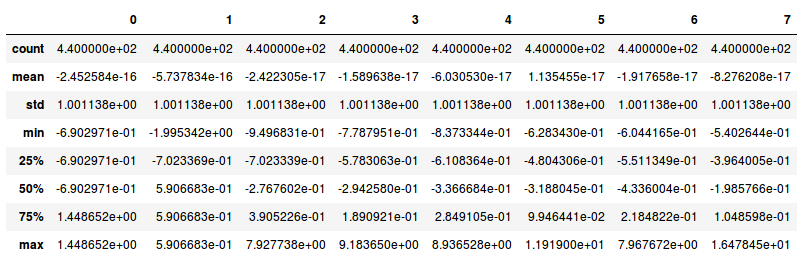
[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/08/Screenshot-from-2019-08-09-16-34-04.png)  
We have the spending details of customers on different products like Milk, Grocery, Frozen, Detergents, etc. Now, we have to segment the customers based on the provided details. Before doing that, let’s pull out some statistics related to the data:

|  |  |
| --- | --- |
|  | # statistics of the data |
|  | data.profile\_report() |

Here, we see that there is a lot of variation in the magnitude of the data. Variables like Channel and Region have low magnitude whereas variables like Fresh, Milk, Grocery, etc. have a higher magnitude.

Since K-Means is a distance-based algorithm, this difference of magnitude can create a problem. So let’s first bring all the variables to the same magnitude:

|  |  |
| --- | --- |
|  | # standardizing the data |
|  | from sklearn.preprocessing import StandardScaler |
|  | scaler = StandardScaler() |
|  | data\_scaled = scaler.fit\_transform(data) |
|  |  |
|  | # statistics of scaled data |
|  | pd.DataFrame(data\_scaled).describe() |

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/08/Screenshot-from-2019-08-09-16-39-31.png)

The magnitude looks similar now. Next, let’s create a kmeans function and fit it on the data:

|  |  |
| --- | --- |
|  | # defining the kmeans function with initialization as k-means++ |
|  | kmeans = KMeans(n\_clusters=2, init='k-means++') |
|  |  |
|  | # fitting the k means algorithm on scaled data |
|  | kmeans.fit(data\_scaled) |

We have initialized two clusters and pay attention – the initialization is not random here. We have used the k-means++ initialization which generally produces better results.

Let’s evaluate how well the formed clusters are. To do that, we will calculate the inertia of the clusters:

|  |  |
| --- | --- |
|  | # inertia on the fitted data |
|  | kmeans.inertia\_ |

**Output:** 2599.38555935614

We got an inertia value of almost 2600. Now, let’s see how we can use the elbow curve to determine the optimum number of clusters in Python.

We will first fit multiple k-means models and in each successive model, we will increase the number of clusters. We will store the inertia value of each model and then plot it to visualize the result:

|  |  |
| --- | --- |
|  | # fitting multiple k-means algorithms and storing the values in an empty list Using   1. Elbow method |
|  | Error =[]  for i in range(2, 11):  kmeans = KMeans(n\_clusters=i, init='k-means++').fit(data\_scaled)  Error.append(kmeans.inertia\_)  import matplotlib.pyplot as plt  plt.plot(range(2, 11), Error)  plt.title('Elbow method')  plt.xlabel('No of clusters')  plt.ylabel('Error')  plt.show() |
|  | Elbow Method   1. Elbow method with Yellobrick Visualiser   #pip install yellowbrick  from yellowbrick.cluster import KElbowVisualizer  visualizer = KElbowVisualizer(kmeans, k=(2,10))  visualizer.fit(data\_scaled)  visualizer.show()  **yellowbrick-elbow** |

Can you tell the optimum cluster value from this plot? Looking at the above elbow curve, **we can choose any number of clusters between 5 to 8**. Let’s set the number of clusters as 6 and fit the model:

|  |  |
| --- | --- |
|  | # k means using 5 clusters and k-means++ initialization |
|  | # k means using 5 clusters and k-means++ initialization |
|  | kmeans5 = KMeans(n\_clusters=5) |
|  | y\_kmeans5 = kmeans5.fit\_predict(data\_scaled) |

Finally, let’s look at the value count of points in each of the above-formed clusters:

|  |  |
| --- | --- |
|  | frame = pd.DataFrame(data\_scaled) |
|  | frame['cluster'] = y\_kmeans5 |
|  | frame['cluster'].value\_counts() |

2 210

1 125

0 91

4 10

3 4

This is how we can implement K-Means Clustering in Python.

# Example-2 Hierarchical Clustering

In this example, we will perform hierarchical clustering on real-world data and see how it can be used to solve an actual problem.

The problem that we are going to solve in this section is to segment customers into different groups based on their shopping trends.

The dataset for this problem can be downloaded from the following link:[shopping-data.csv](https://stackabuse.s3.amazonaws.com/files/hierarchical-clustering-with-python-and-scikit-learn-shopping-data.csv)

To cluster this data into groups we will follow the same steps that we performed in the previous section.

Execute the following script to import the desired libraries:

import matplotlib.pyplot as plt

import pandas as pd

%matplotlib inline

import numpy as np

Next, to import the dataset for this example, run the following code:

customer\_data = pd.read\_csv('/…/…/…/shopping-data.csv')

Let's explore our dataset a bit. To check the number of records and attributes, execute the following script:

customer\_data.shape

The script above will return (200, 5) which means that the dataset contains 200 records and 5 attributes.

To eyeball the dataset, execute the head() function of the data frame. Take a look at the following script:

customer\_data.head()

The output will look like this:

Table : Sample Data

|  | **CustomerID** | **Genre** | **Age** | **Annual Income (k$)** | **Spending Score (1-100)** |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | Male | 19 | 15 | 39 |
| 1 | 2 | Male | 21 | 15 | 81 |
| 2 | 3 | Female | 20 | 16 | 6 |
| 3 | 4 | Female | 23 | 16 | 77 |
| 4 | 5 | Female | 31 | 17 | 40 |

Our dataset has five columns: CustomerID, Genre, Age, Annual Income, and Spending Score. To view the results in two-dimensional feature space, we will retain only two of these five columns. We can remove CustomerID column, Genre, and Age column. We will retain the Annual Income (in thousands of dollars) and Spending Score (1-100) columns. The Spending Score column signifies how often a person spends money in a mall on a scale of 1 to 100 with 100 being the highest spender. Execute the following script to filter the first three columns from our dataset:

data = customer\_data.iloc[:, 3:5].values

Next, we need to know the clusters that we want our data to be split to. We will again use the scipy library to create the dendrograms for our dataset. Execute the following script to do so:

import scipy.cluster.hierarchy as shc

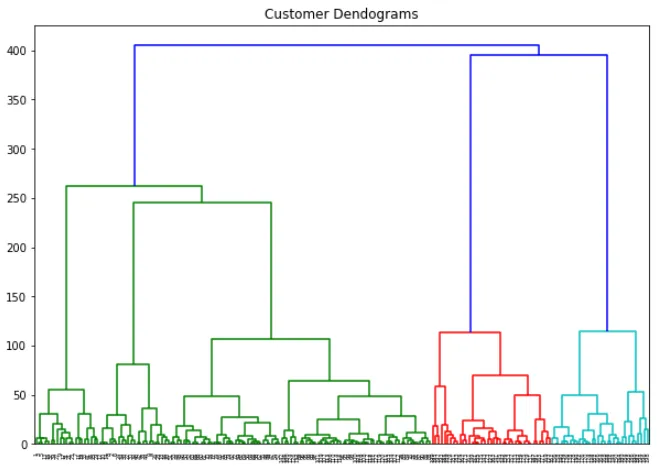
plt.figure(figsize=(10, 7))

plt.title("Customer Dendograms")

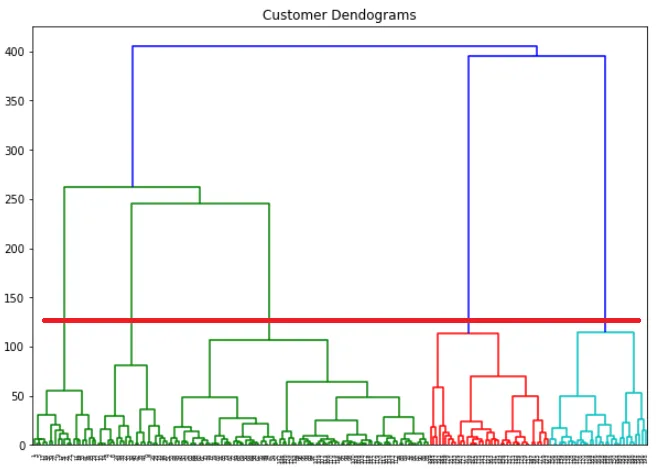
dend = shc.dendrogram(shc.linkage(data, method='ward'))

In the script above we import the hierarchy class of the scipy.cluster library as shc. The hierarchy class has a dendrogram method which takes the value returned by the linkage method of the same class. The linkage method takes the dataset and the method to minimize distances as parameters. We use 'ward' as the method since it minimizes the variants of distances between the clusters (see lecture notes).

The output of the script above looks like this:



If we draw a horizontal line that passes through longest distance without crossing a horizontal line, we get 5 clusters as shown in the following figure:



Now we know the number of clusters for our dataset, the next step is to group the data points into these five clusters. To do so we will again use the AgglomerativeClustering class of the sklearn.cluster library. Take a look at the following script:

from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')

cluster.fit\_predict(data)

The output of the script above looks like this:

array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4,

3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1, 4, 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, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2,

0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 1,

2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,

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

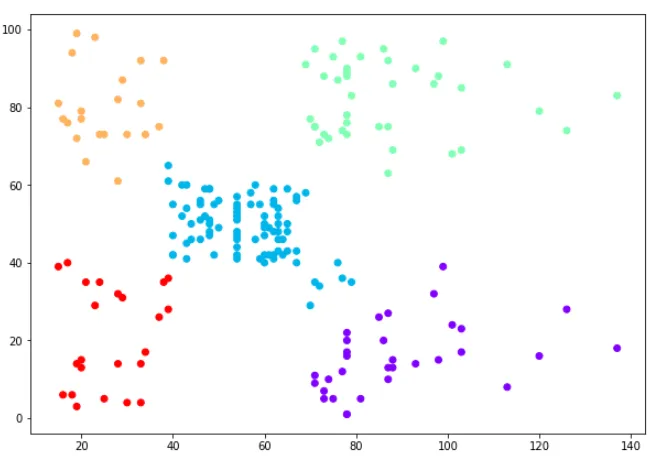
You can see the cluster labels from all of your data points. Since we had five clusters, we have five labels in the output i.e. 0 to 4.

As a final step, let's plot the clusters to see how actually our data has been clustered:

plt.figure(figsize=(10, 7))

plt.scatter(data[:,0], data[:,1], c=cluster.labels\_, cmap='rainbow')

The output of the code above looks like this:



You can see the data points in the form of five clusters. The data points in the bottom right belong to the customers with high salaries but low spending. These are the customers that spend their money carefully. Similarly, the customers at top right (green data points), these are the customers with high salaries and high spending. These are the type of customers that companies target. The customers in the middle (blue data points) are the ones with average income and average salaries. The highest numbers of customers belong to this category. Companies can also target these customers given the fact that they are in huge numbers, etc.