Chapter 4 : Part 1

**K-Means Clustering**

**4.1.1 K-Means Clustering**

*K-Means Clustering* is an algorithm that allows you to Cluster your data. *K-Means Clustering* is an *unsupervised* *learning* algorithm that is used to solve the clustering problems in *machine learning* or *data* *science*.

* *K-Means Clustering* is an *Unsupervised* *Learning* algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if ***K=2***, there will be ***two*** ***clusters***, and for ***K=3***, there will be ***three*** ***clusters***, and so on.
* It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
* It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
* It is a ***centroid-based algorithm***, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data-point and their corresponding clusters (centroid).
* The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.
* The k-means clustering algorithm mainly performs following tasks:
* Determines the ***best value*** for ***K*** ***center*** ***points*** or ***centroids*** by an iterative process.
* Assigns each *data-point* to its closest *k-center*. Those *data points* which are near to the particular *k-center*, create a *cluster*.
* Hence each cluster has data-points with some commonalities, and it is away from other clusters.
* The below diagram explains the working of the K-means Clustering Algorithm:



* It is a very convenient tool for discovering *categories* or *groups* in your data-set.

|  |  |
| --- | --- |
| * Let's imagine that we've got two variables in our data set and we decided to plot those two variables on ***X*** and ***Y*** axis. The question is: * How our observations are configured according to these two variables? * Can we identify certain groups among all variables? * What the *K-Means* does for you is: it takes out the complexity from this decision making process and allows you to very easily identify those clusters (actually called clusters of data points) in your data-set. |  |

* Of course this is a very simplified example, in this case we have three classes and we only have two dimensions (two variables) here. so K-means can work with multidimensional objects, there can be as many as 10 or 100 any number of variables.
* Steps of K-means Algorithm: The working of the K-Means algorithm is explained in the below steps:

1. Step-1 (Choose the number K of clusters): Select the number K to decide the number of clusters. Let's imagine that we've agreed on a *number* of *clusters* for a certain challenge, say *3* or *2* or *5* *clusters*. Once you've done that then you proceed to *step 2*.

* We'll talk more about how to select the *Optimal Number Of Clusters*.

1. Step-2: Select random K points or CENTROIDS. (It can be other from the input dataset). The random k points will be the centroid of your clusters and ***not necessarily*** these ***points have to be from your dataset***.

* As you saw we had a Scatterplot, we could select any points in that Scatterplot. The points *don't have to be part of the observations*, they can be any random ***x*** and ***y*** values on your *Scatterplot*. (As long as you just selects a ***certain number of centroids*** that are going to equate to the number of clusters that you have decided upon).

1. Step-3: Assign each data point to their Closest Centroid, which will form the ***predefined*** ***K*** ***clusters***.

* So you're starting clusters and then there's going to be an Iterative process to Refine those clusters.
* Basically so you just check for every point in a data set, which of them is the closest.
* Closest is a kind of vague term here, because it depends on what kind of distance you're measuring (is it Euclidean distances? Or some other sort of distance?).
* For the purposes of simplicity, we're going to talk about Euclidean distances (that's basically geometrical distances).

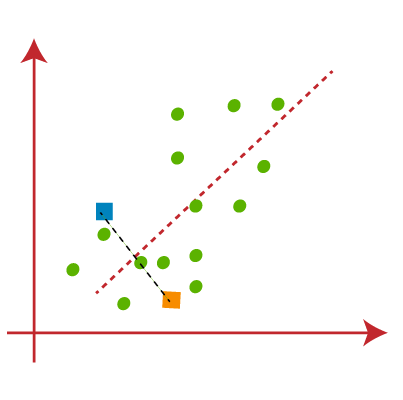
1. Step-4: Calculate the Variance and *place* a *new centroid* of each *cluster*.
2. Step-5: Repeat the 3rd steps, which means *reassign* each *datapoint* to the New Closest Centroid of each CLUSTER.
3. Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
4. Step-7: The model is ready.

**4.1.2 K-Means Clustering Example**

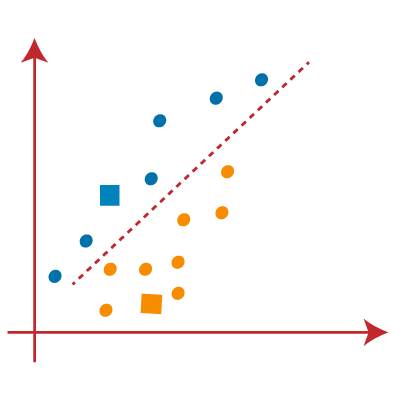
Now we do these steps manually to Understand the whole process:

|  |  |
| --- | --- |
| * This is our *scatterplot*. We can't just visually *identify* the final *clusters*, although it is *two-dimension* but it's pretty tough. * Now imagine how complex a situation would be if we had three or more variables !! We wouldn't even be able to plot a five dimensional scatterplot like that. * So that's where it came in is clustering comes into play and that's where this algorithm will help us simplify the process. * In this case we're actually going to manually perform the same k-means clustering algorithm (later we will do these in python/R). | K-Means Clustering Algorithm |
| 1. Let's take number ***k*** of clusters, i.e., ***K=2***, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into ***two*** different ***clusters***.  * We need to choose some ***random*** ***k*** ***points*** or ***centroid*** to form the cluster. * These points can be either the points from the dataset or any other point. So, here we are selecting the two points as k points (in the right-side figure), which are not the part of our dataset. Consider the right-side image: | K-Means Clustering Algorithm |

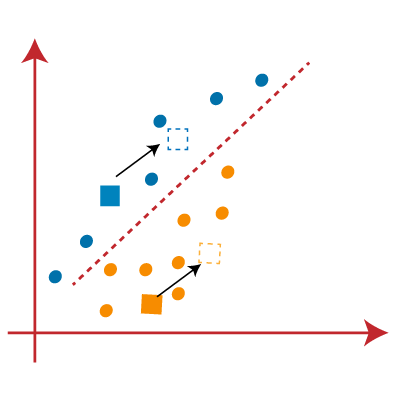
1. Now we will assign each data point of the scatter plot to its closest ***K-point*** or ***centroid***. We will compute it by applying some ***mathematics*** that we have studied to calculate the distance between two points. So, we will ***draw*** a ***median*** between both the ***centroids***. Consider following image.



* From the above image, it is clear that points ***left*** side of the ***line*** is near to the ***K1*** or ***blue*** ***centroid***, and points to the ***right*** of the line are close to the ***yellow*** ***centroid***. Let's color them as ***blue*** and ***yellow*** for clear visualization.



1. As we need to find the closest cluster, so we will repeat the process by choosing a new centroid. To choose the new centroids, we will compute the center of gravity of these centroids, and will find *new* *centroids* as below:

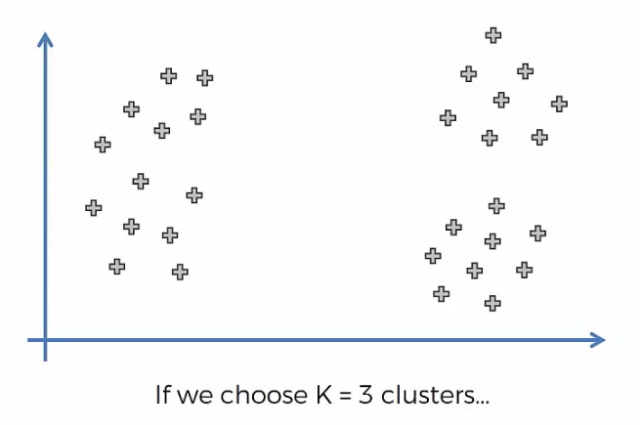


|  |  |
| --- | --- |
| 1. Next, we will ***reassign*** each ***datapoint*** to the ***new*** ***centroid***. For this, we will repeat the same process of finding a ***median*** ***line***. The median will be like right - side image:  * From the beside image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids. | K-Means Clustering Algorithm |

|  |  |
| --- | --- |
| 1. As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points. | K-Means Clustering Algorithm |
| 1. We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the right-side image: | K-Means Clustering Algorithm |
| 1. As we got the *new centroids* so again will *draw* the *median* *line* and *reassign* the *data* *points*. So, the image will be: | K-Means Clustering Algorithm |
| 1. We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image: | K-Means Clustering Algorithm |
| 1. As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image: | K-Means Clustering Algorithm |

**4.1.3 K-Means Random Initialization Trap**

Consider the following scatterplot. We have two variables represented by the *x* and *y* *coordinates*. Let's say we're going to choose three clusters. It does look like you can pretty easily spot them here.



|  |  |  |
| --- | --- | --- |
|  |  |  |

* So this is the end result if we choose the ***correct random initialization*** at ***correct location***.
* However, if we select a centroid in different locations we will end up with different result. By following the steps of the algorithm we will end up as follows:

|  |  |
| --- | --- |
|  |  |
| 1. Draw the median lines | 1. Group the points |
| 1. Find new centroids | 1. Then assign the centroids to the new positions: |

* Wrong End result: Now if we draw the median lines, we can see there are no dissimilar data points. So this is the our end point. And it is the final cluster result.

|  |  |
| --- | --- |
| True Cluster | False Cluster |
|  |  |

* So you can see that the three clusters are different and therefore, the selection of the Centroid is at the very start of the algorithm can potentially dictate the outcome of the algorithm. And that's not a good thing because the centroid are selected at random.
* ***K-means++:*** There is a *modification* to the *K-means algorithm* that allows you to correctly select the Centroid called the ***K-means++*** algorithm.
* Python Scikit-learn library can handle this automatically: From above we can see that it is obviously a trap. However we not need to worry about this because, python implementation of this k-means algorithm can automatically handle this kind of "trap" situation. The good news is that ***K-means++*** algorithm happens in background in either in R or Python or whatever tool you're using. You don't need to actually implement it.

**4.1.4 Elbow-Method: Choosing K (Right Number Of Clusters)**

The performance of the *K-means clustering* *algorithm* depends upon highly efficient clusters that it forms. But ***choosing the optimal number of clusters*** is a big task. There are some *different ways* to find the *optimal number of clusters*, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

* Elbow Method: The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. WCSS stands for Within Cluster Sum of Squares, which defines the ***total*** ***variations*** within a cluster. The formula to calculate the value of ***WCSS (for 3 clusters)*** is given below:

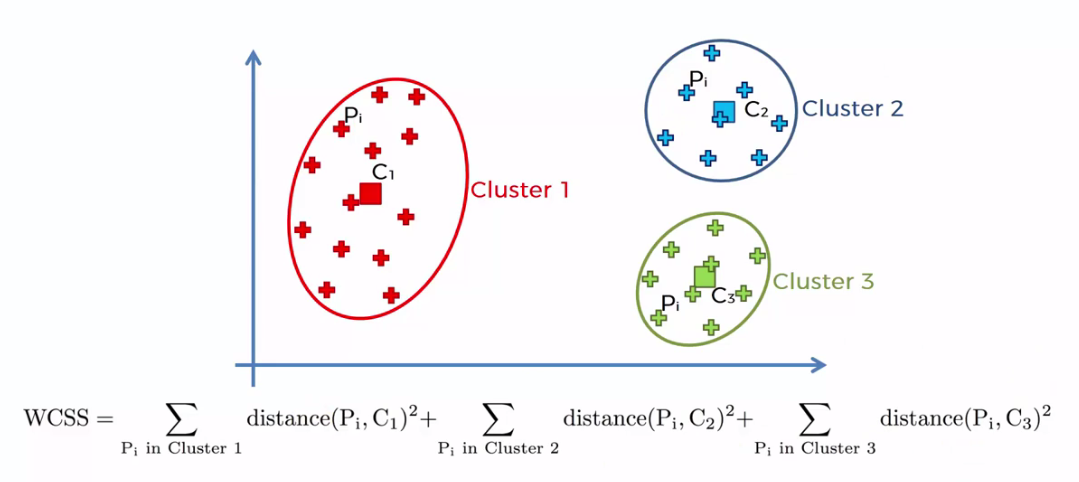
Where are the data-points in corresponding Clusters and are the centroids.

* In the above formula of WCSS, Following is the *sum of the square of the distances* between each *data point* and its *centroid* within a Cluster1 and the same for the other two terms.
* To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.
* To find the optimal value of clusters, the elbow method follows the below steps:

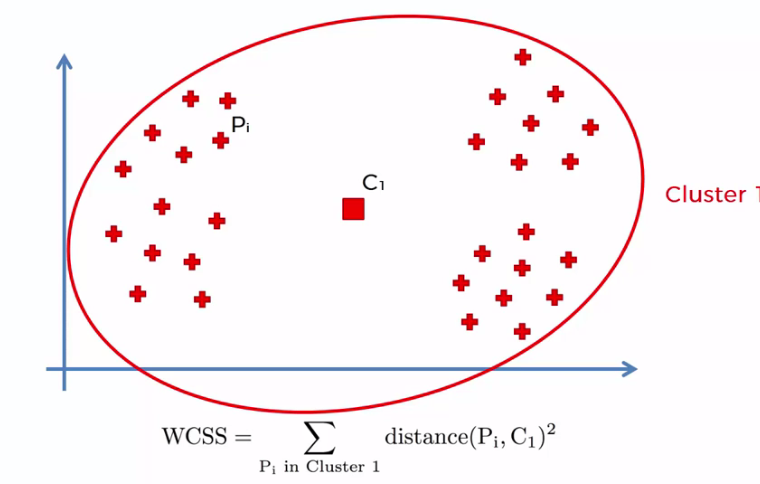
1. It executes the ***K-means*** clustering on a given dataset for different ***K values*** (ranges from 1-10).
2. For ***each*** value of ***K***, calculates the ***WCSS*** value.
3. Plots a ***curve*** between calculated ***WCSS*** values and the ***number*** of clusters ***K***.
4. The ***sharp point*** of ***bend*** or a ***point*** of the ***plot*** looks like an ***arm***, then that point is considered as the ***best*** value of ***K***.

|  |  |
| --- | --- |
| * Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the right side image: * Maximum number of Clusters: We can choose the number of clusters equal to the given data points. If we choose the number of clusters equal to the data points, then the value of WCSS becomes zero, and that will be the endpoint of the plot. * Actually we can set the K for which WCSS decrease rapidly and after that certain number the WCSS won't change rapidly. | K-Means Clustering Algorithm |

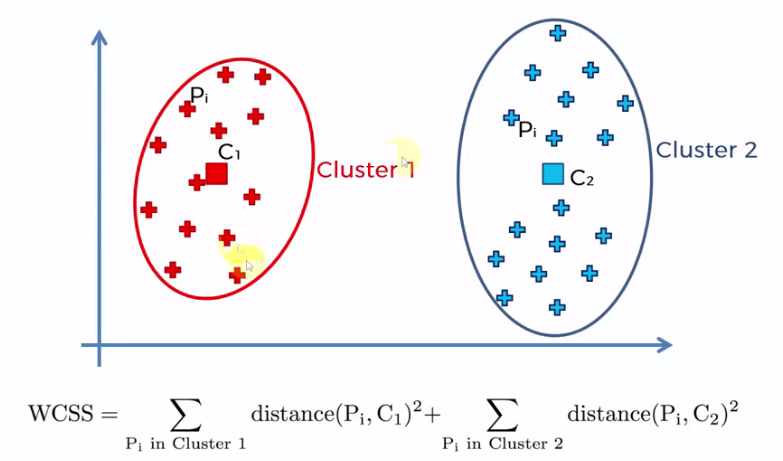
* WCSS value is a metric: We need a certain metric so that we can understand or evaluate how a certain number of clusters performs compared to a different number of clusters (and preferably that metric should be quantifiable).



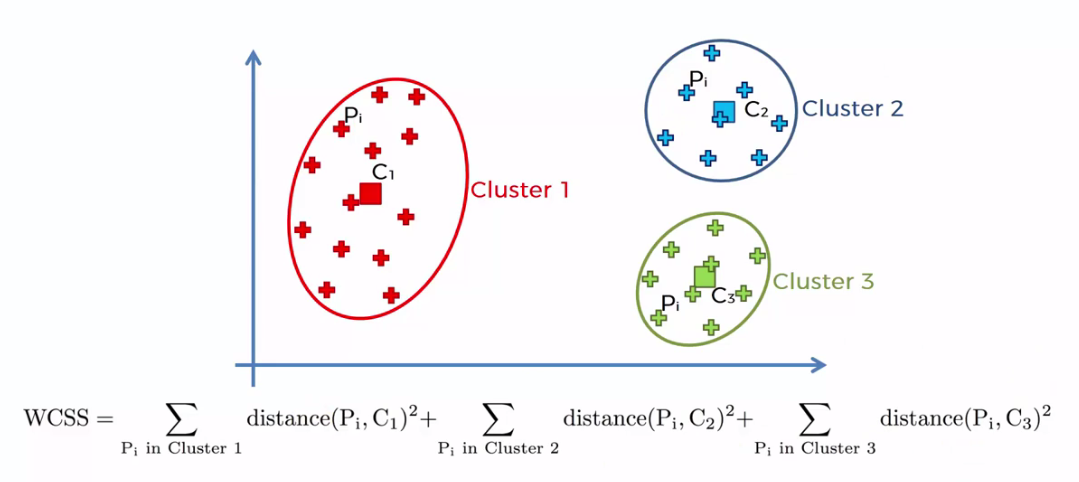
* Actually it's a quite a good metric in terms of understanding or comparing the goodness of fit between two different K-means clusterings.
* Let's see how that metric ***WCSS*** is going to change as we ***increase*** the ***number*** of ***clusters***.
* If we use one centroid in the middle, the distance from each data point is big and the squared distance will be very large.



* If we have 2 clusters then the squared distance reduces.



* For 3 clusters we get more reduced ***WCSS***. And for 4, 5, or more ***centroid*** the distance doesn't decrease ***rapidly*** (but it decreases). The WCSS tends to ***0*** as ***K*** reaches to ***number of data points***.



* So the *Elbow-method* is just an approach that can help you to decide the number of k (number of clusters). But at the end of day it is your decision.

**4.1.5 Python Implementation of K-means Clustering**

* Problem Description: We have a dataset of ***Mall\_Customers***, which is the data of customers who visit the mall and spend there.
* In the given dataset, we have ***Customer\_Id***, ***Gender***, ***Age***, ***Annual Income ($)***, and ***Spending*** ***Score*** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

#*WCSS : Within-Cluster Sum of Square*

#*K-Means Clustering*

#*libraries*

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

#*importing data*

dataSet = pd**.read\_csv**("Mall\_Customers.csv")

X = dataSet**.**iloc[:, [3, 4]]**.**values

#*There is no y for clustering*

#*Finding optimal noumber of clusters using elbow method : WCSS*

**from** sklearn**.**cluster **import** KMeans

wcss = []

**for** i **in** **range**(1, 11):

    #*setting paramter for cluster generator*

    cluster\_generator\_elbow = **KMeans**(n\_clusters = i, init="k-means++", random\_state=0, max\_iter=300, n\_init=10)

    cluster\_generator\_elbow**.fit**(X) #*fit the independent data*

    wcss**.append**(cluster\_generator\_elbow**.**inertia\_) #*capturing wccs data for each i*

    #*inertia\_ :  Sum of squared distances of samples to their closest cluster center. Is actually "wcss"*

#*visualizing the elbow digram ith clusters vs wcss*

#*range(1, 11), wcss: both are lists*

plt**.plot**(range(1, 11), wcss)

plt**.title**("The elbow Method")

plt**.xlabel**("Number of Clusters")

plt**.ylabel**("WCSS")

plt**.show**()

#*creating cluster with optimal "n\_clusters". From elbow-plot we figured out that 5 is the optimal number of clusters*

k\_mean\_cluster\_genrt = **KMeans**(n\_clusters = 5, init="k-means++", random\_state=0, max\_iter=300, n\_init=10)

y\_k\_mean\_cluster = k\_mean\_cluster\_genrt**.fit\_predict**(X)

#*plotting the cluster*

plt**.scatter**(X[y\_k\_mean\_cluster **==** 0, 0], X[y\_k\_mean\_cluster **==** 0, 1], s = 100, c = "red", label="cluster 1")

plt**.scatter**(X[y\_k\_mean\_cluster **==** 1, 0], X[y\_k\_mean\_cluster **==** 1, 1], s = 100, c = "blue", label="cluster 2")

plt**.scatter**(X[y\_k\_mean\_cluster **==** 2, 0], X[y\_k\_mean\_cluster **==** 2, 1], s = 100, c = "green", label="cluster 3")

plt**.scatter**(X[y\_k\_mean\_cluster **==** 3, 0], X[y\_k\_mean\_cluster **==** 3, 1], s = 100, c = "cyan", label="cluster 4")

plt**.scatter**(X[y\_k\_mean\_cluster **==** 4, 0], X[y\_k\_mean\_cluster **==** 4, 1], s = 100, c = "pink", label="cluster 5")

#*centroids*

plt**.scatter**(k\_mean\_cluster\_genrt**.**cluster\_centers\_[:, 0], k\_mean\_cluster\_genrt**.**cluster\_centers\_[:, 1], s=300, c="black", label = "Centroids")

plt**.title**("Clusters Of Clients")

plt**.xlabel**("Annual income ($)")

plt**.ylabel**("Spending score (1-100)")

plt**.legend**()

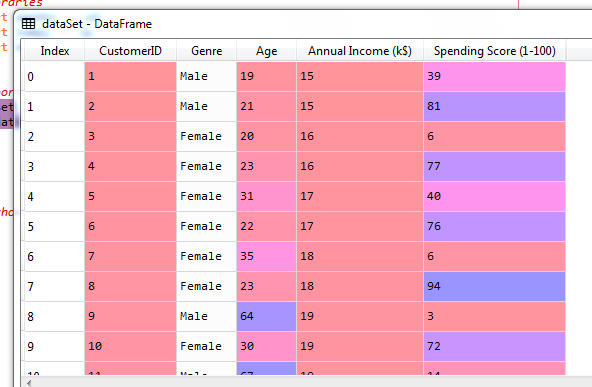
plt**.show**()

#*python prctc\_k\_mns.py*

* Data preprocessing: There is *no y dependent variable* in clustering. Here we don't need any dependent variable for data pre-processing step as it is a *clustering problem*, and we have *no idea about what to determine*. So we will just add a line of code for the matrix of features.

x = dataset.iloc[:, [3, 4]].values

* As we can see, we are extracting only ***4th and 5th feature-column:*** *Annual Income* and *Spending Score*. It is because we need a ***2D plot to visualize the model***, and some features are not required, such as customer\_id, Gender. The dataset is



* Selecting Library and Class:

**from** sklearn**.**cluster **import** KMeans

* The Elbow technique: Inside a for loop we create the object for ***1*** to ***10*** *number of clusters* and we calculate ***WCSS*** for each of them. After creating the ***list*** of ***WCSS*** values we plot the ***Elbow*** ***diagram***. We then select the ***optimal number of clusters***.

#*Finding optimal noumber of clusters using elbow method : WCSS*

**from** sklearn**.**cluster **import** KMeans

wcss = []

**for** i **in** **range**(1, 11):

    #*setting paramter for cluster generator*

    cluster\_generator\_elbow = **KMeans**(n\_clusters = i, init="k-means++", random\_state=0, max\_iter=300, n\_init=10)

    cluster\_generator\_elbow**.fit**(X) #*fit the independent data*

    wcss**.append**(cluster\_generator\_elbow**.**inertia\_) #*capturing wccs data for each i*

    #*inertia\_ :  Sum of squared distances of samples to their closest cluster center. Is actually "wcss"*

#*visualizing the elbow digram ith clusters vs wcss*

#*range(1, 11), wcss: both are lists*

plt**.plot**(range(1, 11), wcss)

plt**.title**("The elbow Method")

plt**.xlabel**("Number of Clusters")

plt**.ylabel**("WCSS")

plt**.show**()

* We create the object using:

**KMeans**(n\_clusters = i, init="k-means++", random\_state=0, max\_iter=300, n\_init=10)

* ***n\_clusters*** : The number of clusters to form as well as the number of centroids to generate. For Elbow method it is a variable. For final cluster we will use the fixed optimal number of cluster.
* ***init = 'k-means++'*** is a Method for initialization. '***k-means++***' : selects initial cluster centers for *k-mean clustering* in a smart way to *speed* up *convergence*.
* ***n\_init*** : Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia.
* ***max\_iter*** : Maximum number of iterations of the k-means algorithm for a single run.
* After fitting the data we extract the WCSS value using:

**cluster\_generator\_elbow.inertia\_**

* **inertia\_** : Sum of squared distances of samples to their closest cluster center. Is actually "wcss"
* Optimal Number Of Cluster: From the Elbow diagram we notice that, if the number of cluster is more than 5 then WCSS doesn't change rapidly. So 5 is the optimal number of cluster.

|  |  |
| --- | --- |
| * From the ***above plot***, we can see the ***elbow poin***t is at ***5***. So the ***number of clusters*** here will be ***5***. | K-Means Clustering Algorithm |

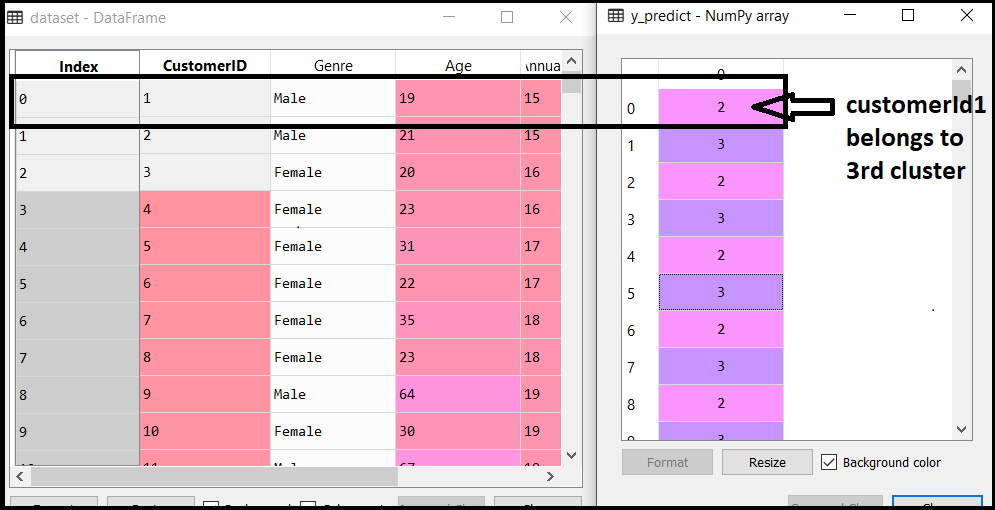
* Creating cluster with optimal "n\_clusters": Finally we analyze the data for 5 clusters. And we use ***fit\_predict()*** to find the y-values.
* As we have got the number of clusters, so we can now train the model on the dataset.
* To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using ***i***, we will use ***5***, as we know there are ***5 clusters*** that need to be formed.

#*creating cluster with optimal "n\_clusters". From elbow-plot we got 5 is the optimal number of clusters*

k\_mean\_cluster\_genrt = **KMeans**(n\_clusters = 5, init="k-means++", random\_state=0, max\_iter=300, n\_init=10)

y\_k\_mean\_cluster = k\_mean\_cluster\_genrt**.fit\_predict**(X)

* The first line is the same as above for creating the object of ***KMeans class***.
* In the second line of code, we have created the dependent variable ***y\_k\_mean\_cluste***r (we can call it ***y\_predict*** also) to train the model.
* By executing the above lines of code, we will get the ***y\_k\_mean\_cluster*** variable. We can check it under the variable explorer option in the Spyder IDE. We can now compare the values of ***y\_k\_mean\_cluster*** with our original dataset. Consider the below image:



* From the above image, we can now relate that the ***CustomerID 1*** belongs to a ***cluster 3***(as index starts from ***0***, hence ***2*** will be considered as ***3***), and ***2*** belongs to ***cluster 4***, and so on
* Visualizing the Clusters: In following lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the ***plt.scatter***, i.e., ***X[y\_k\_mean\_cluster == 0, 0]*** containing the ***x*** value for the showing the matrix of features values, and the ***y\_k\_mean\_cluster*** is ranging from ***0*** to ***1***.

#*plotting the cluster*

plt**.scatter**(X[y\_k\_mean\_cluster **==** 0, 0], X[y\_k\_mean\_cluster **==** 0, 1], s = 100, c = "red", label="cluster 1")

plt**.scatter**(X[y\_k\_mean\_cluster **==** 1, 0], X[y\_k\_mean\_cluster **==** 1, 1], s = 100, c = "blue", label="cluster 2")

plt**.scatter**(X[y\_k\_mean\_cluster **==** 2, 0], X[y\_k\_mean\_cluster **==** 2, 1], s = 100, c = "green", label="cluster 3")

plt**.scatter**(X[y\_k\_mean\_cluster **==** 3, 0], X[y\_k\_mean\_cluster **==** 3, 1], s = 100, c = "cyan", label="cluster 4")

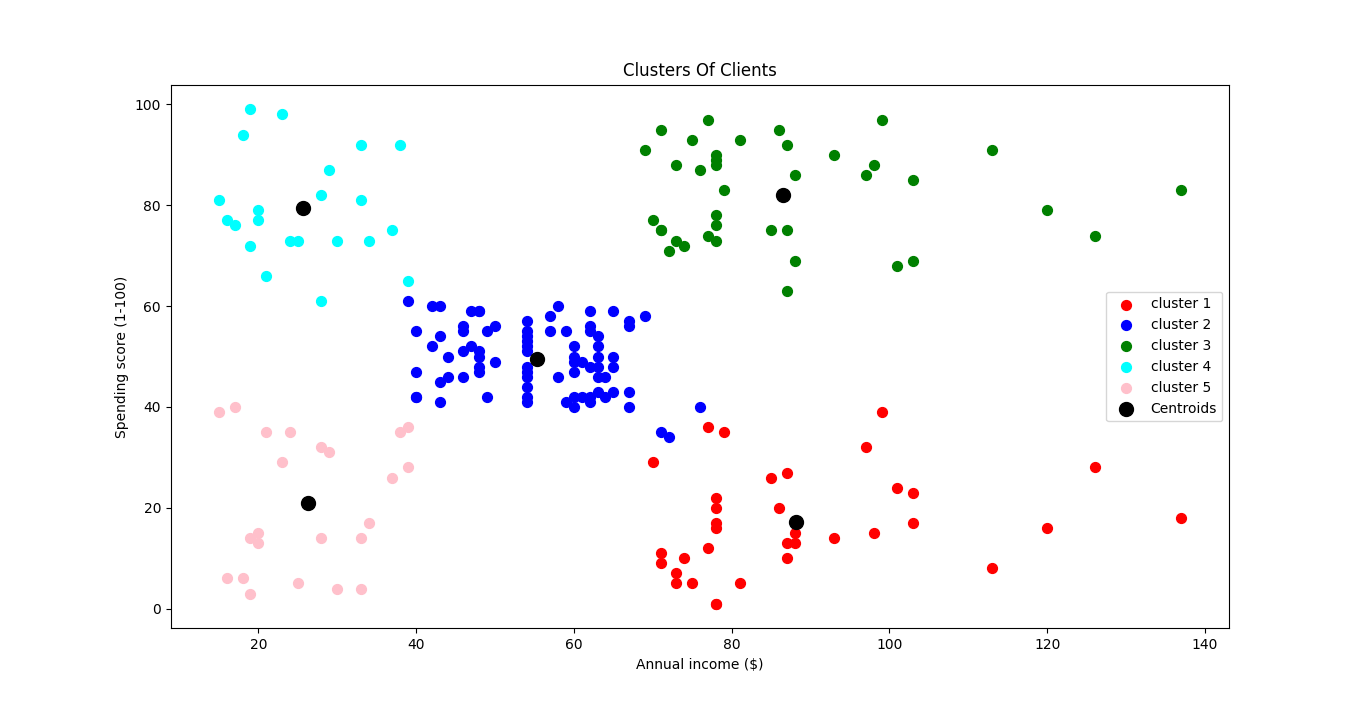
plt**.scatter**(X[y\_k\_mean\_cluster **==** 4, 0], X[y\_k\_mean\_cluster **==** 4, 1], s = 100, c = "pink", label="cluster 5")

#*centroids*

plt**.scatter**(k\_mean\_cluster\_genrt**.**cluster\_centers\_[:, 0], k\_mean\_cluster\_genrt**.**cluster\_centers\_[:, 1], s=300, c="black", label = "Centroids")

|  |  |
| --- | --- |
| plt**.title**("Clusters Of Clients")  plt**.xlabel**("Annual income ($)")  plt**.ylabel**("Spending score (1-100)")  plt**.legend**()  plt**.show**()   * ***X[y\_k\_mean\_cluster == 0, 0]*** * ***y\_k\_mean\_cluster == 0*** looks for the matching elements with 1st cluster in the feature matrix ***X***. And the second ***0*** means the 1st column of X, i.e ***Annual income***. * Similarly ***X[y\_k\_mean\_cluster == 0, 1]*** * Retrieves all matching data points with 1st cluster and their 2nd column data i.e. ***Spending*** ***Score*** in feature matrix X. |  |

* Categorize the customers: The output image is clearly showing the ***five different clusters*** with different colors. The clusters are formed between two parameters of the dataset; ***Annual*** ***income*** of customer and ***Spending***. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:



1. ***Cluster1*** shows the customer has a high income but low spending, so we can categorize them as Careful
2. ***Cluster2*** shows the customers with average salary and average spending so we can categorize these customers as Standard
3. ***Cluster3*** shows the customers with high income and high spending so they can be categorized as Target, and these customers can be the most profitable customers for the mall owner.
4. ***Cluster4*** shows the customers with low income with very high spending so they can be categorized as Careless.
5. ***Cluster5*** shows the low income and also low spending so they can be categorized as Sensible.

* Multi-Dimensional Clustering: For 3 feature variable we still can visualize the cluster in 3-D graphics, but more tan 3D we can't plot the clusters.
* However later we will learn a technique that allows us to reduce the dimensions of our data. So that you can plot the clusters.
* If you are doing clustering in more than two dimensions then don't execute the last code lines in this section to visualize the clusters. Because it's only for two dimensional clustering.