K means clustering is an unsupervised learning algorithm that partitions n objects into k clusters, based on the nearest mean. This module highlights what the K-means algorithm is, and the use of K means clustering, and toward the end of this module we will build a K means clustering model with the help of the Iris Dataset.

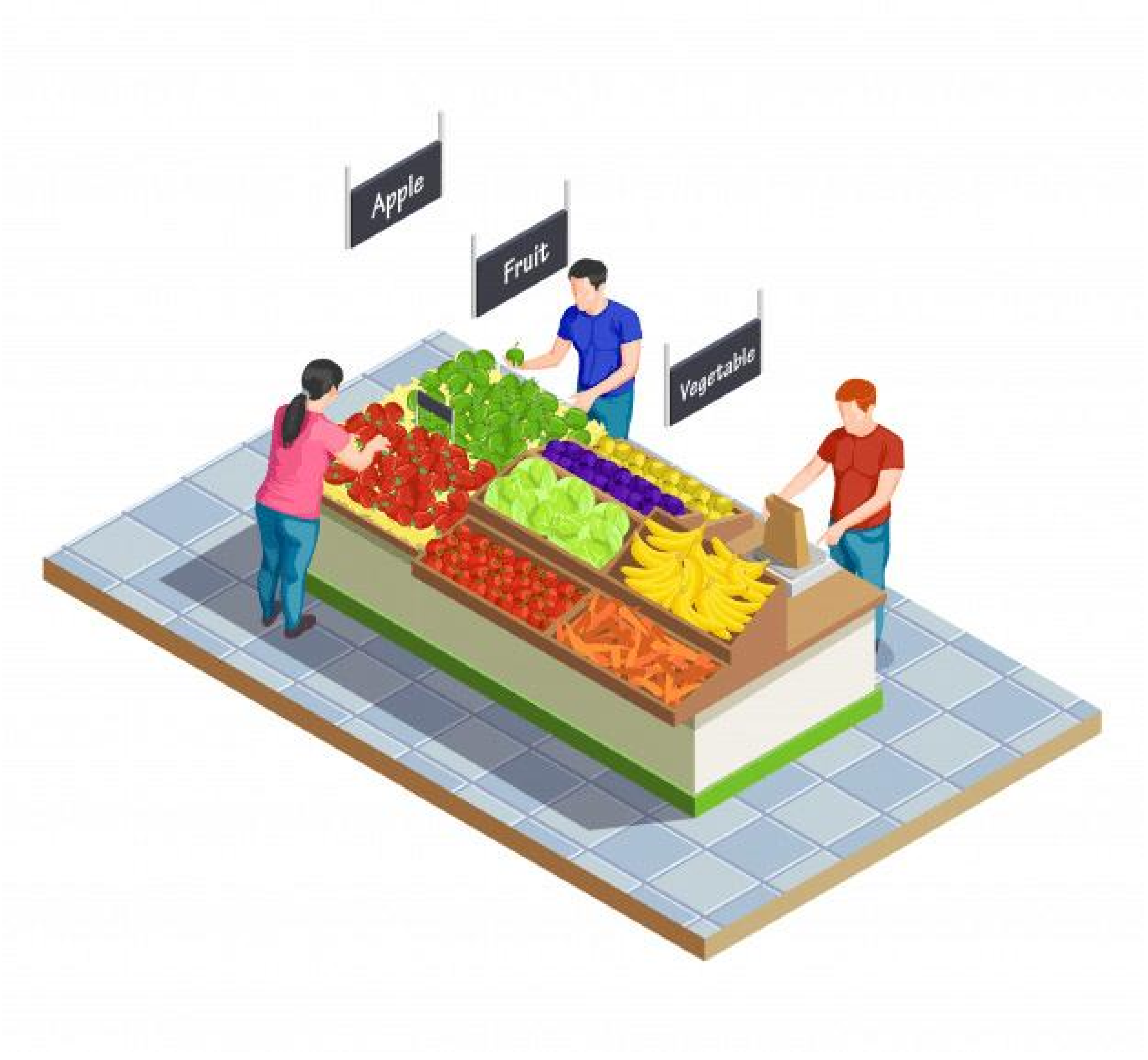
Introduction to K means Clustering in Python

K means clustering algorithm is a very common unsupervised learning algorithm. This algorithm clusters ***n*** objects into ***k*** clusters, where each object belongs to a cluster with the nearest mean.

Here's the table of contents for this module:

What Is Clustering Algorithms?

Clustering is nothing but dividing a set of data into groups of similar points or features, where data points in the same group are as similar as possible and data points in different groups are as dissimilar as possible. We use clustering in our day-to-day lives; for instance, in a supermarket, all vegetables are grouped as one group and all fruits are grouped as another group. This clustering helps customers fasten their shopping process.



Another clustering example we might have come across is the Amazon or Flipkart product recommendation. Amazon or Flipkart recommend us products based on our previous search. How do they do it? Well, the concept behind this is clustering.

Now that we know what clustering is, let us discuss the categories that we have in clustering.

What Is K means Clustering Algorithm?

K means clustering is an algorithm, where the main goal is to group similar data points into a cluster. In K means clustering, *k* represents the total number of groups or clusters. K means clustering runs on Euclidean distance calculation. Now, let us understand K means clustering with the help of an example.

Say, we have a dataset consisting of height and weight information of 10 players. We need to group them into two clusters based on their height and weight.

Say, we have a dataset consisting of height and weight information of 10 players. We need to group them into two clusters based on their height and weight.

Height Weight

180	80
172	73
178	69
189	82
164	70
186	71
180	69
170	76
166	71
180	72

Step 1: Initialize a cluster centroid

Initial Clusters	**Height**	**Weight**
K1	185	70
K2	170	80

Step 2: Calculate the Euclidean distance from each observation to the initial clusters

$$\text{Euclidean Distance} = \sqrt{(x_{height} - H_{centroid})^2 + (x_{weight} - W_{centroid})^2}$$

Observation	**Height**	**Weight**	**Distance from Cluster 1**	**Distance from Cluster 2**	**Assign Clusters**
1	180	80	11.18	10	2
2	172	73	13.3	7.28	2
3	178	69	7.07	13.6	1
4	189	82	12.64	19.10	1
5	164	70	21	11.66	2
6	186	71	1.41	18.35	1
7	180	69	5.09	14.86	1
8	170	76	16.15	4	2
9	166	71	19.02	9.84	2
10	180	72	5.38	12.80	1

Step 3: Find the new cluster centroid

Observation	**Height**	**Weight**	**Assign Clusters**
1	180	80	2
2	172	73	2
3	178	69	1
4	189	82	1
5	164	70	2
6	186	71	1

7	180	69	1
8	170	76	2
9	166	71	2
10	180	72	1
New Cluster 1	(178+189+186+180+180)/5	(69+82+71+69+72)/5	
New Cluster 2	(180+172+164+170+166)/5	(80+73+70+76+76+71)/5	

New Cluster 1 = (182.6, 72.6)

New Cluster 2 = (170.4, 89.2)

Step 4: Again, calculate the Euclidean distance

Calculate the Euclidean distance from each observation to both Cluster 1 and Cluster 2

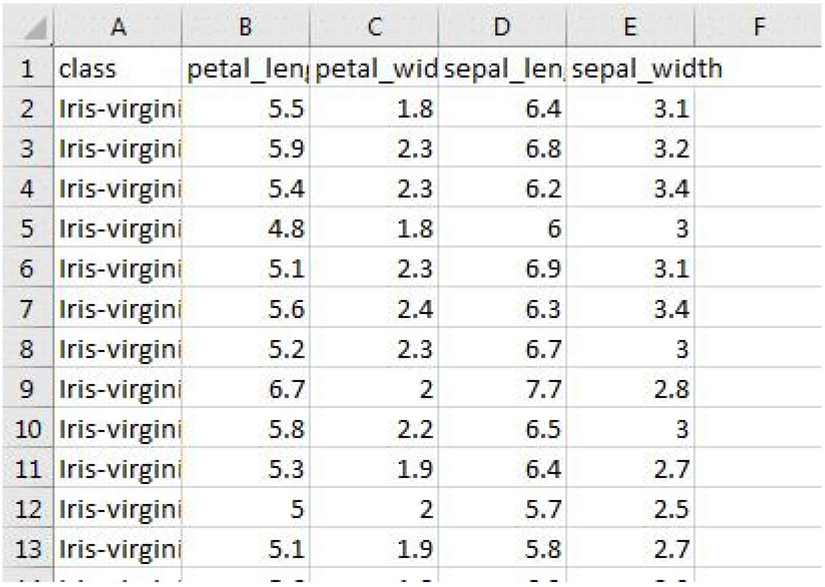
Repeat Steps 2, 3, and 4, until cluster centers don't change any more

Now, let us look at the hands-on given below to have a deeper understanding of K-means algorithm.

Hands-on: K-means Clustering Algorithm Using Sklearn in Python- Iris Dataset

Dataset

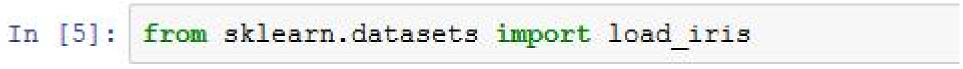
We will be using the famous Iris Dataset, collected in the 1930s by Edgar Anderson. In this example, we are going to apply Kmeans.



	A	B	C	D	E	F
1	class	petal_len	petal_wid	sepal_len	sepal_width	
2	Iris-virgini	5.5	1.8	6.4	3.1	
3	Iris-virgini	5.9	2.3	6.8	3.2	
4	Iris-virgini	5.4	2.3	6.2	3.4	
5	Iris-virgini	4.8	1.8	6	3	
6	Iris-virgini	5.1	2.3	6.9	3.1	
7	Iris-virgini	5.6	2.4	6.3	3.4	
8	Iris-virgini	5.2	2.3	6.7	3	
9	Iris-virgini	6.7	2	7.7	2.8	
10	Iris-virgini	5.8	2.2	6.5	3	
11	Iris-virgini	5.3	1.9	6.4	2.7	
12	Iris-virgini	5	2	5.7	2.5	
13	Iris-virgini	5.1	1.9	5.8	2.7	

Let us see how we can implement K means clustering using Python, in this problem statement. To implement K-means we will be using sklearn library in Python. Let's get started.

Step 1: Load the Iris Dataset



```
In [5]: from sklearn.datasets import load_iris
```

Step 2: Have a glance at the shape

```
In [3]: iris_data = load_iris()
```

Step 3: Have a glance at the features

```
In [23]: iris_data.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
```

Step 4: Have a glance at the targets

```
In [24]: iris_data.target
```

```
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, 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, 2, 2])
```

Step 5: Build the model

```
In [12]: from sklearn.cluster import KMeans
```

Step 6: Set up the number of clusters

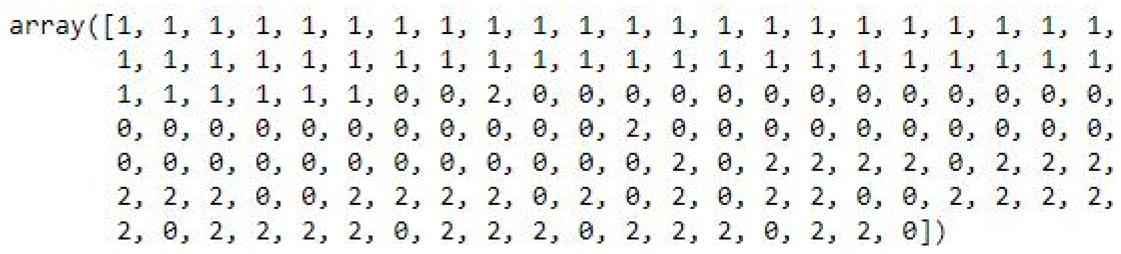
```
In [25]: # set the number of clusters
         kmeans = KMeans(n_clusters = 3)
```

Step 7: Fit the features into the model

```
In [26]: kmodel = kmeans.fit(iris_data.data)
```

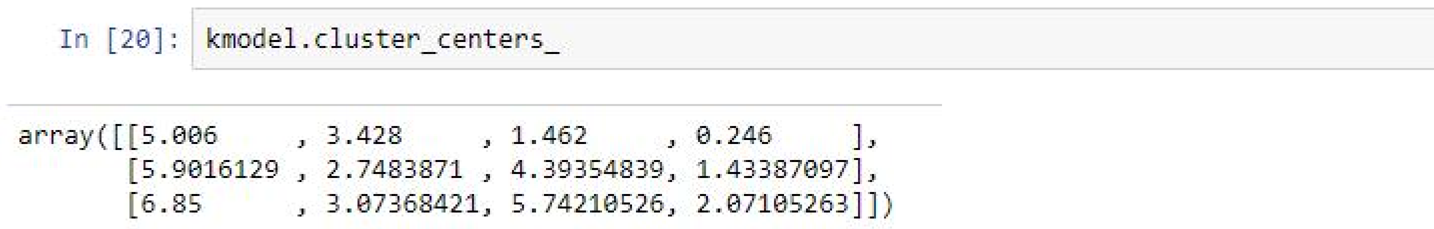
Step 8: Predict and label the data

```
In [32]: kmodel1 = kmeans.predict(iris_data.data)
         kmodel1
```



```
array([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, 0, 0, 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, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0])
```

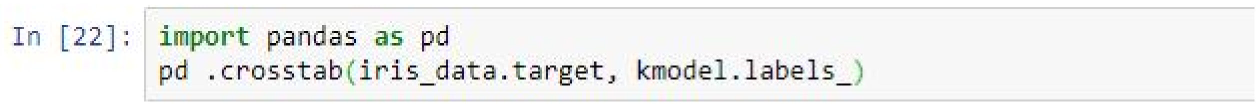
Step 9: Have a look at the cluster centers



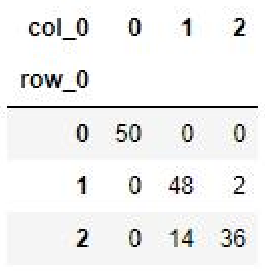
```
In [20]: kmodel.cluster_centers_
```

```
array([[5.006     , 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

Step 10: Evaluate the algorithm



```
In [22]: import pandas as pd
         pd .crosstab(iris_data.target, kmodel.labels_)
```



col_0	0	1	2
row_0			
0	50	0	0
1	0	48	2
2	0	14	36

From this cross-tabulation, we can conclude that all the 50 items in Category 0 are predicted correctly; 48 items in Category 1 are predicted correctly, but 2 items are predicted incorrectly to be of Category 2. 36 items in Category 2 are predicted correctly, but 14 items are predicted incorrectly.