Understanding Cluster Formation in KMeans

In [1]:

```
# importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.cluster import KMeans
```

The below function produces data points in the shape of a circle of specified centre, radius and number of samples.

In [2]:

```
def circles(centre=[0,0],radius=1,samples=1000):
    square_box=np.random.uniform(-radius,radius,[4*samples,2])
    count=0
    data=[]
    for point in square_box:
        if (point**2).sum()<=radius**2:
            data.append(list(point+np.array(centre)))
            count+=1
            if count==samples:
                return np.array(data)
```

Like any other ML algorithm, KMeans clustering also has its own set of pros and cons. Here we're going to discuss a specific kind of disadvantage of this algorithm which is the fact that what appears to be a 'natural cluster' for a human eye is not necessarily what appears to be a cluster from machine's perspective. Kindly note that I'm NOT using the word 'disadvantage' in a strict sense. There's nothing inherently wrong with the clusters formed by KMeans clustering algorithm, it meets its objective of reducing the sum of squared distance from its respective cluster centres, pretty well (except for the annoying fact that it may converge at a local optima).

Intuition meets algorithm!

KMeans clustering matches with our intuition well only in the case of presence of clusters which are well-separated, atleast roughly spherical of same size and with more or less same number of data points. In all the following scenarios we're going to deal with only two clusters to keep the discussion & visualization simple.

In [3]:

```
cir1=circles()
cir2=circles(centre=[2.5,0])
cir=np.vstack([cir1,cir2])
```

In [4]:

```
plt.title('Before Clustering')
plt.ylim(-1.5,1.5)
sns.scatterplot(x=cir[:,0],y=cir[:,1]);
```

In [5]:

```
km1=KMeans(n_clusters=2, random_state=101)
```

In [6]:

```
label1=km1.fit_predict(cir)
label1
```

Out[6]:

```
array([0, 0, 0, ..., 1, 1, 1])
```

In [7]:

```
centroid1=km1.cluster_centers_
centroid1
```

Out[7]:

```
array([[ 6.17942918e-04, -1.45861054e-03],
       [ 2.47484543e+00, -1.78983736e-02]])
```

In [8]:

```
plt.title('After Clustering')
plt.ylim(-1.5,1.5)
sns.scatterplot(x=cir[:,0],y=cir[:,1],hue=label1,palette=['Green','Red'])
plt.scatter(x=centroid1[:,0],y=centroid1[:,1],marker='*', color='Black', s=100);
```

Those two stars are the centroids of their respective clusters. In the above case, the clusters formed by the algorithm is in line with how we would form the clusters ourselves.

What if the clusters are NOT of same size?

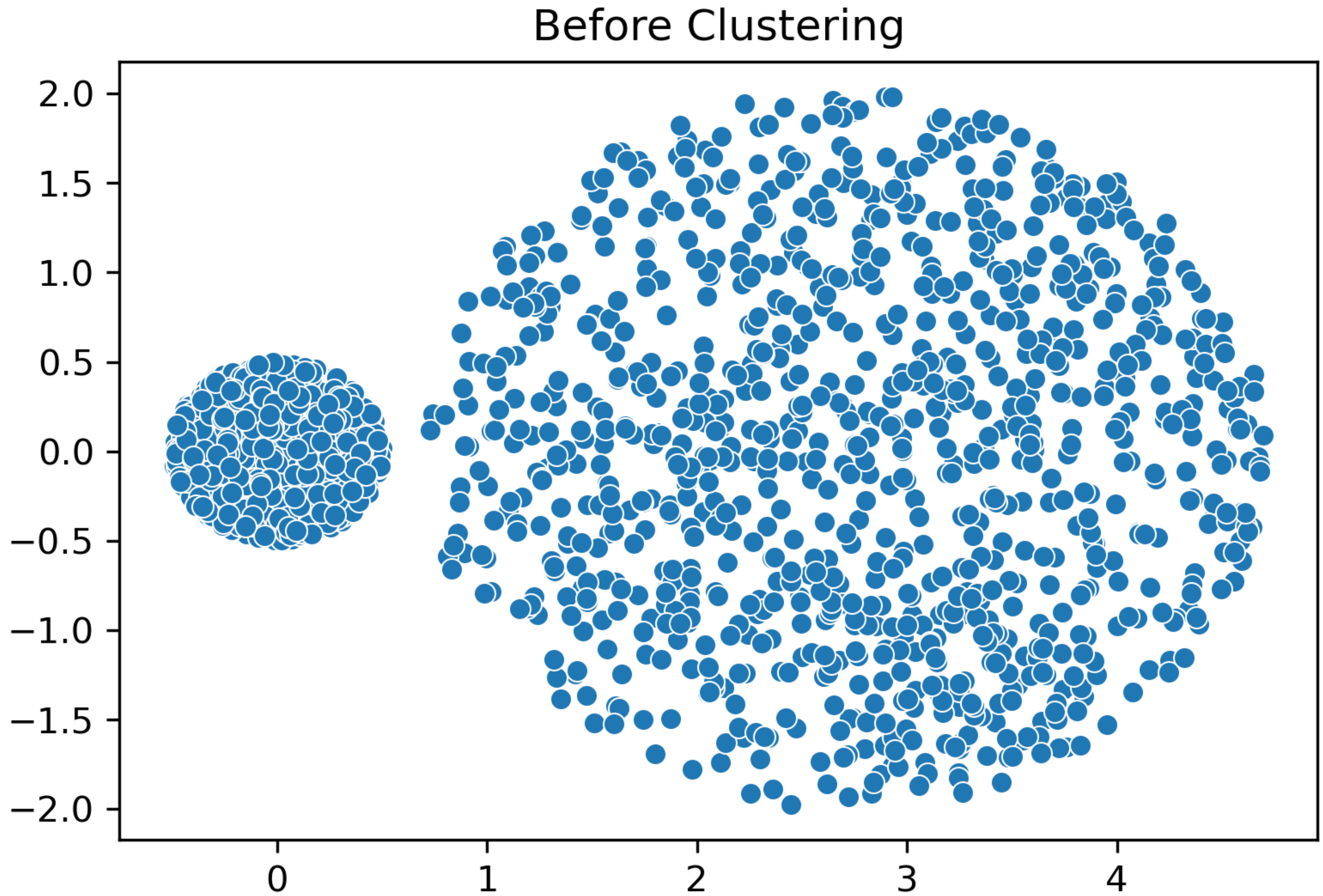
Here we are going to use two well separated circular clusters with same number of samples in each but different radii. Lets see how KMeans clustering algorithm clusters it.

In [9]:

```
cir1=circles(radius=0.5)
cir2=circles(centre=[2.7,0],radius=2)
cir=np.vstack([cir1,cir2])
```

In [10]:

```
plt.title('Before Clustering')
sns.scatterplot(x=cir[:,0],y=cir[:,1]);
```



In [11]:
```
km2=KMeans(n_clusters=2, random_state=101)
```

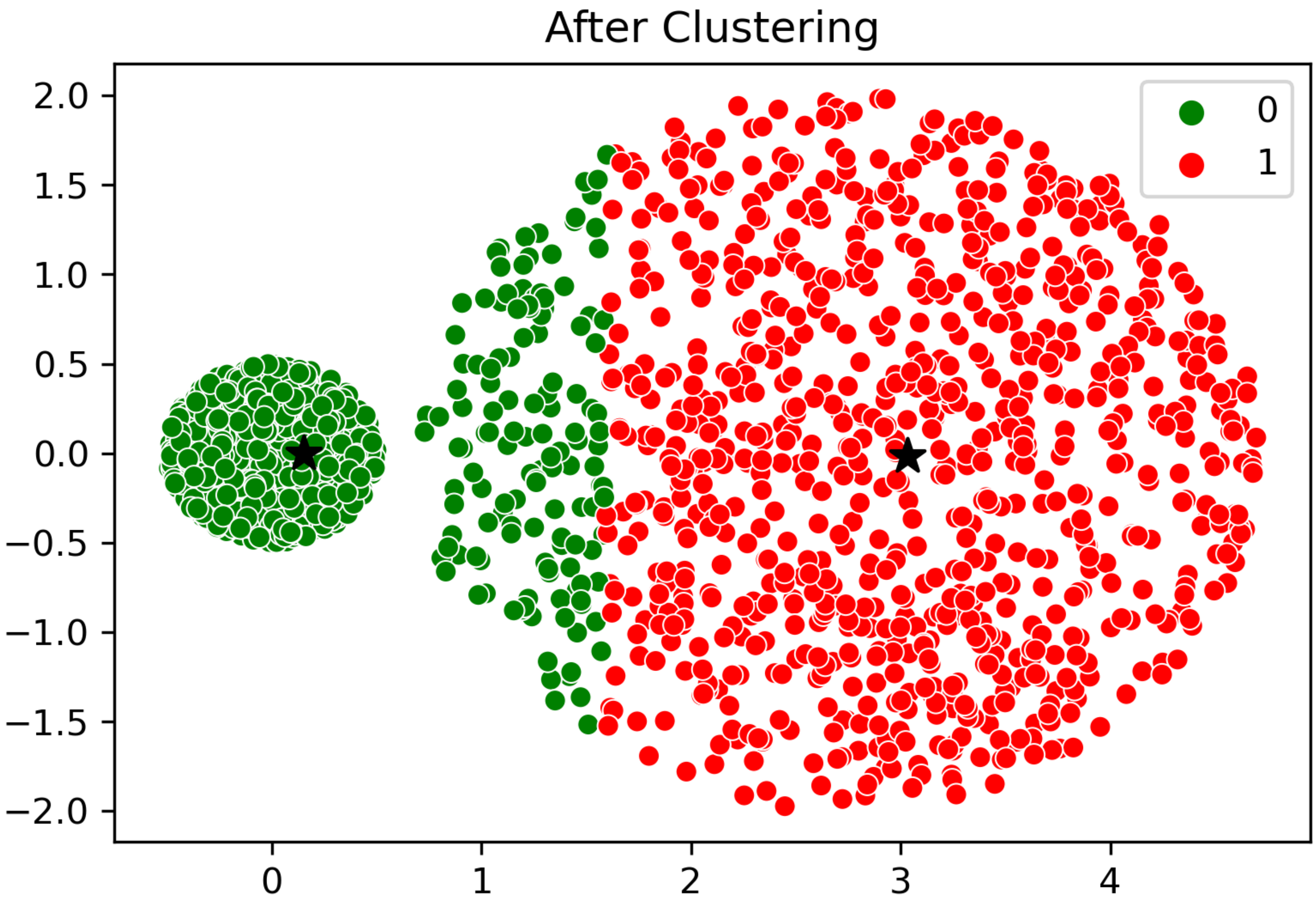
In [12]:
```
label2=km2.fit_predict(cir)
label2
```

Out[12]:
```
array([0, 0, 0, ..., 1, 1, 1])
```

In [13]:
```
centroid2=km2.cluster_centers_
centroid2
```

Out[13]:
```
array([[ 1.52076699e-01,  2.71184564e-04],
       [ 3.03189321e+00, -1.60171782e-02]])
```

In [14]:
```
plt.title('After Clustering')
sns.scatterplot(x=cir[:,0],y=cir[:,1],hue=label2,palette=['Green','Red'])
plt.scatter(x=centroid2[:,0],y=centroid2[:,1],marker='*', color='Black', s=100);
```



Those two stars are the centroids of their respective clusters. In the above case, what would have been otherwise interpreted as one small cluster and one big cluster by a human, is now clustered in the above manner by the algorithm. The smaller circle is gobbling up a portion of the bigger circle.

What if the clusters do NOT have same number of samples?

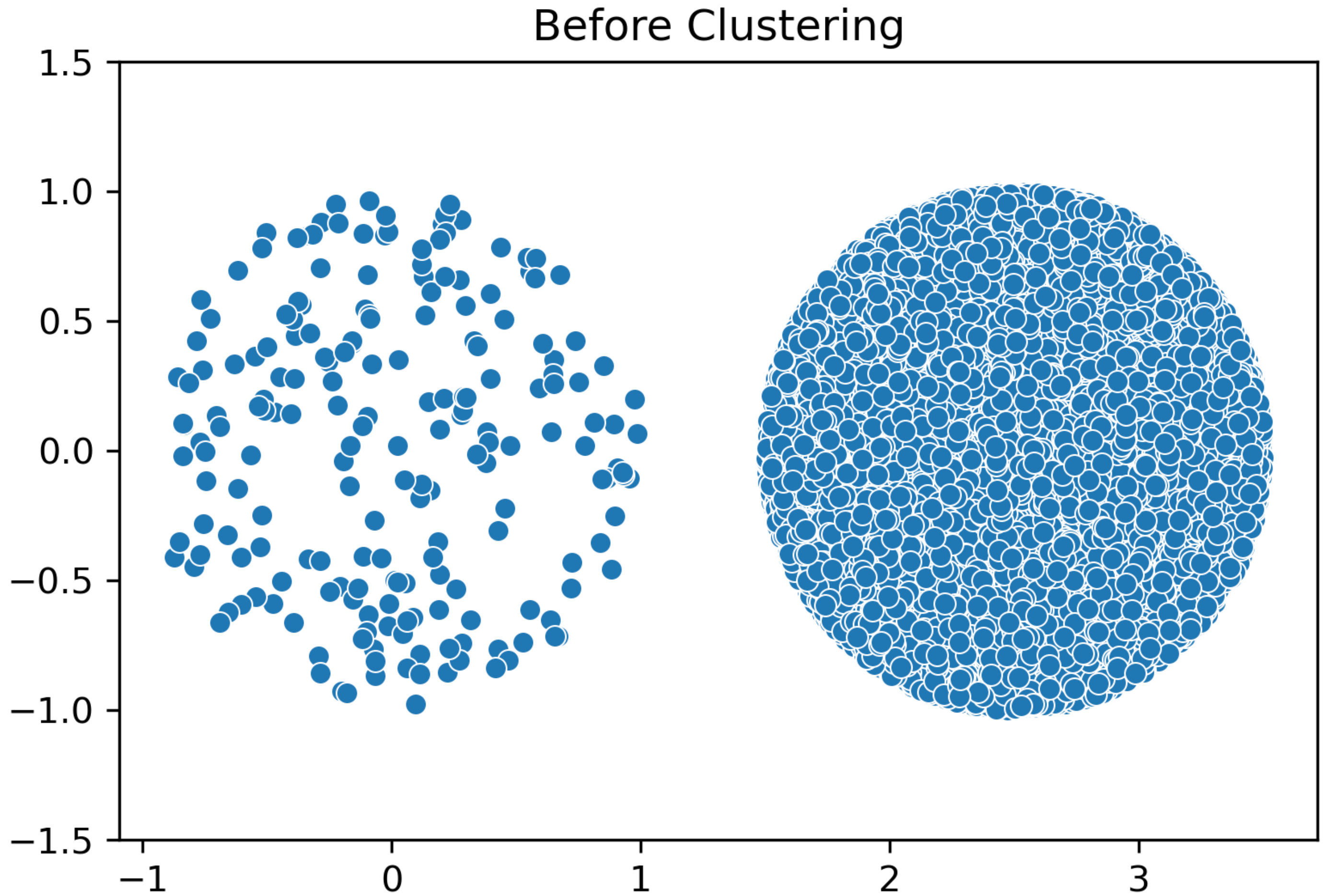
Here we are going to use two well separared circular clusters of same radii but different number of samples. Lets see how KMeans clustering algorithm clusters it.

In [15]:

```
cir1=circles(samples=200)
cir2=circles(centre=[2.5,0],samples=20000)
cir=np.vstack([cir1,cir2])
```

In [16]:

```
plt.title('Before Clustering')
plt.ylim(-1.5,1.5)
sns.scatterplot(x=cir[:,0],y=cir[:,1]);
```



In [17]:
```
km3=KMeans(n_clusters=2, random_state=101)
```

In [18]:
```
label3=km3.fit_predict(cir)
label3
```

Out[18]:
```
array([0, 0, 0, ..., 0, 1, 1])
```

In [19]:
```
centroid3=km3.cluster_centers_
centroid3
```

Out[19]:
```
array([[ 2.00164732,  0.05403784],
       [ 2.88454635, -0.04785949]])
```

In [20]:
```
plt.title('After Clustering')
plt.ylim(-1.5,1.5)
sns.scatterplot(x=cir[:,0],y=cir[:,1],hue=label3,palette=['Green','Red'])
plt.scatter(x=centroid3[:,0],y=centroid3[:,1],marker='*', color='Black', s=100);
```

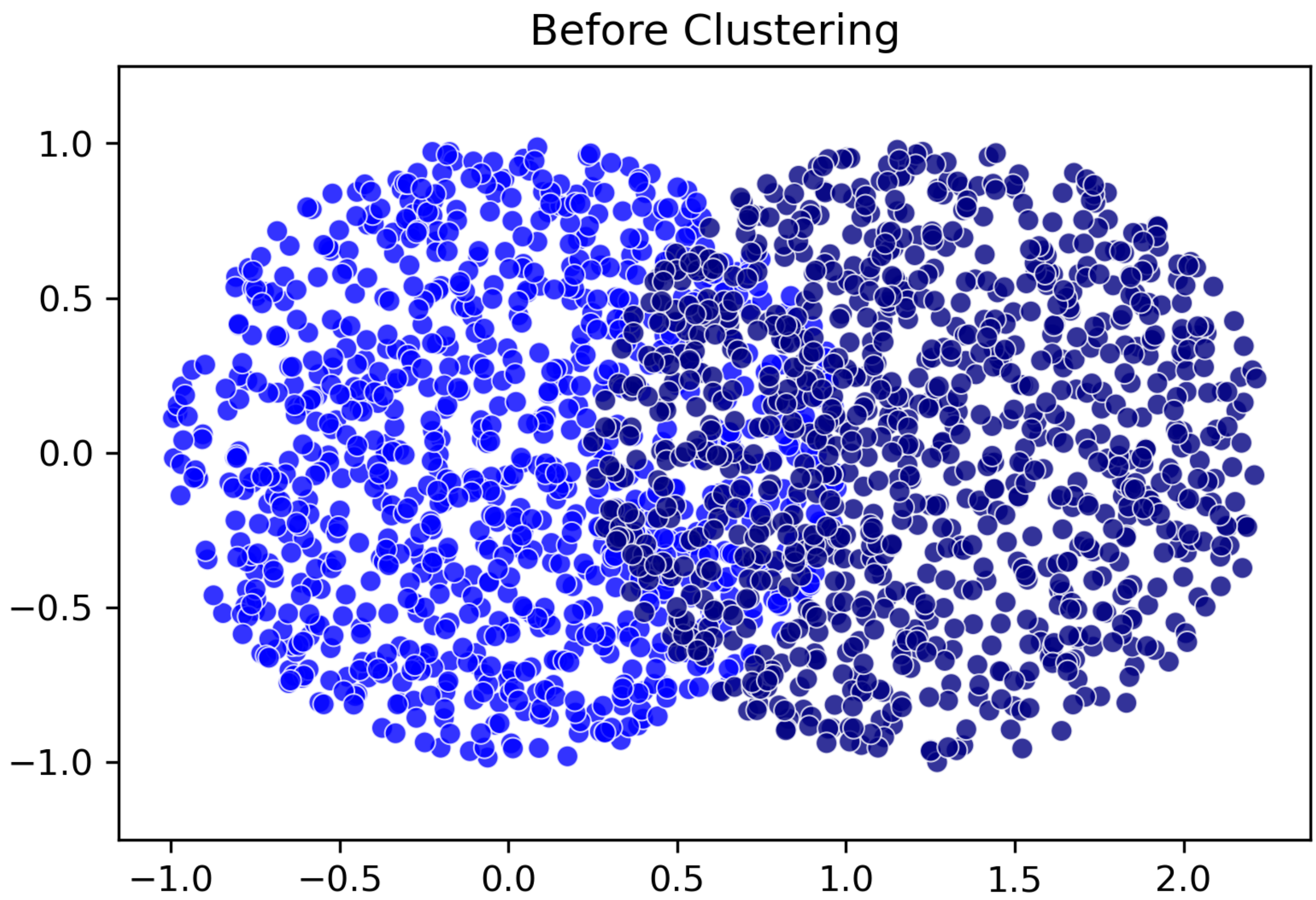
Those two stars are the centroids of their respective clusters. In the above case, what would have been otherwise interpreted as one sparse cluster and one dense cluster by a human, is now clustered in the above manner by the algorithm. Basically we are better off having two centroids in the extremely dense circle than to have one centroid for each circle.

What if the clusters are of same size and same density but NOT well separated?

Here we are going to use two slightly overlapping circular clusters of same radii and same density. Lets see how KMeans clustering algorithm clusters it.

In [21]:
```
cir1=circles()
lab1=np.zeros(1000,)
cir2=circles(centre=[1.25,0])
lab2=np.ones(1000,)
cir=np.vstack([cir1,cir2])
lab=np.hstack([lab1,lab2])
```

In [22]:
```
plt.title('Before Clustering')
plt.ylim(-1.25,1.25)
sns.scatterplot(x=cir[:,0],y=cir[:,1],hue=lab,alpha=0.8,palette=['Blue','Navy'])
plt.legend('',frameon=False);
```



Note that the above data is NOT clustered by the algorithm. Instead look at it this way, we somehow have a prior knowledge about the existence of these two clusters and we also knew that these two clusters are overlapping. With this prior knowledge lets see how the KMeans clustering algorithm performs on the above data

In [23]:
```
km4=KMeans(n_clusters=2, random_state=101)
```

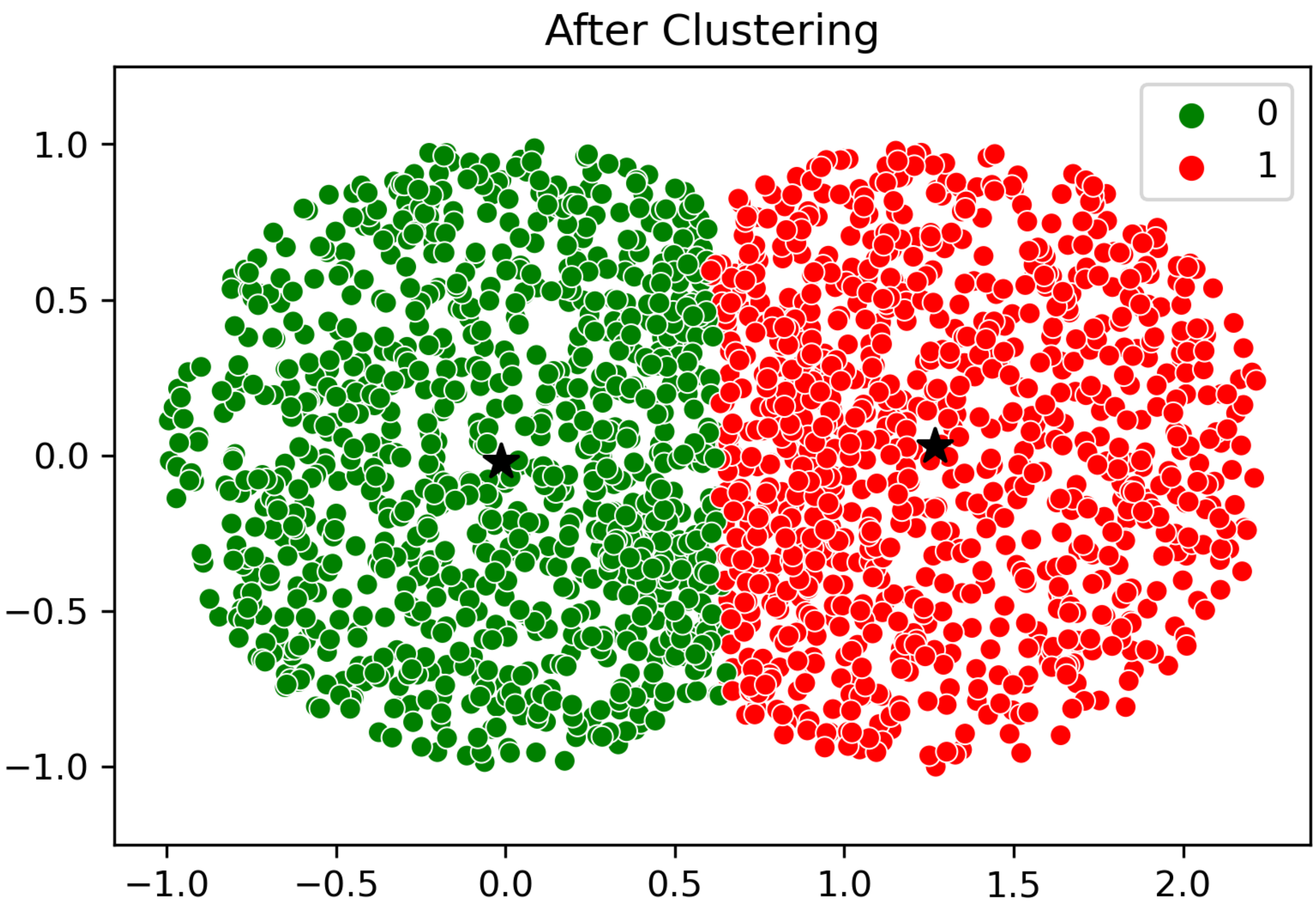
In [24]:
```
label4=km4.fit_predict(cir)
label4
```

Out[24]:
```
array([0, 1, 0, ..., 1, 1, 1])
```

In [25]:
```
centroid4=km4.cluster_centers_
centroid4
```

Out[25]:
```
array([[-0.01285595, -0.0200944 ],
       [ 1.26711263,  0.02886077]])
```

In [26]:
```
plt.title('After Clustering')
plt.ylim(-1.25,1.25)
sns.scatterplot(x=cir[:,0],y=cir[:,1],hue=label4,palette=['Green','Red'])
plt.scatter(x=centroid4[:,0],y=centroid4[:,1],marker='*', color='Black', s=100);
```



Those two stars are the centroids of their respective clusters. In the above case, neither humans nor the algorithm can figure out the 'true' clusters. But by looking at the graph, at the very least we would easily guess that there are actually 2 circles which are overlapping, whereas KMeans clustering algorithm will cut right through the 'centre' of the overlap using the perpendicular bisector of the line segment joining the centroids.

What if the clusters are of same density but NOT of same shape?

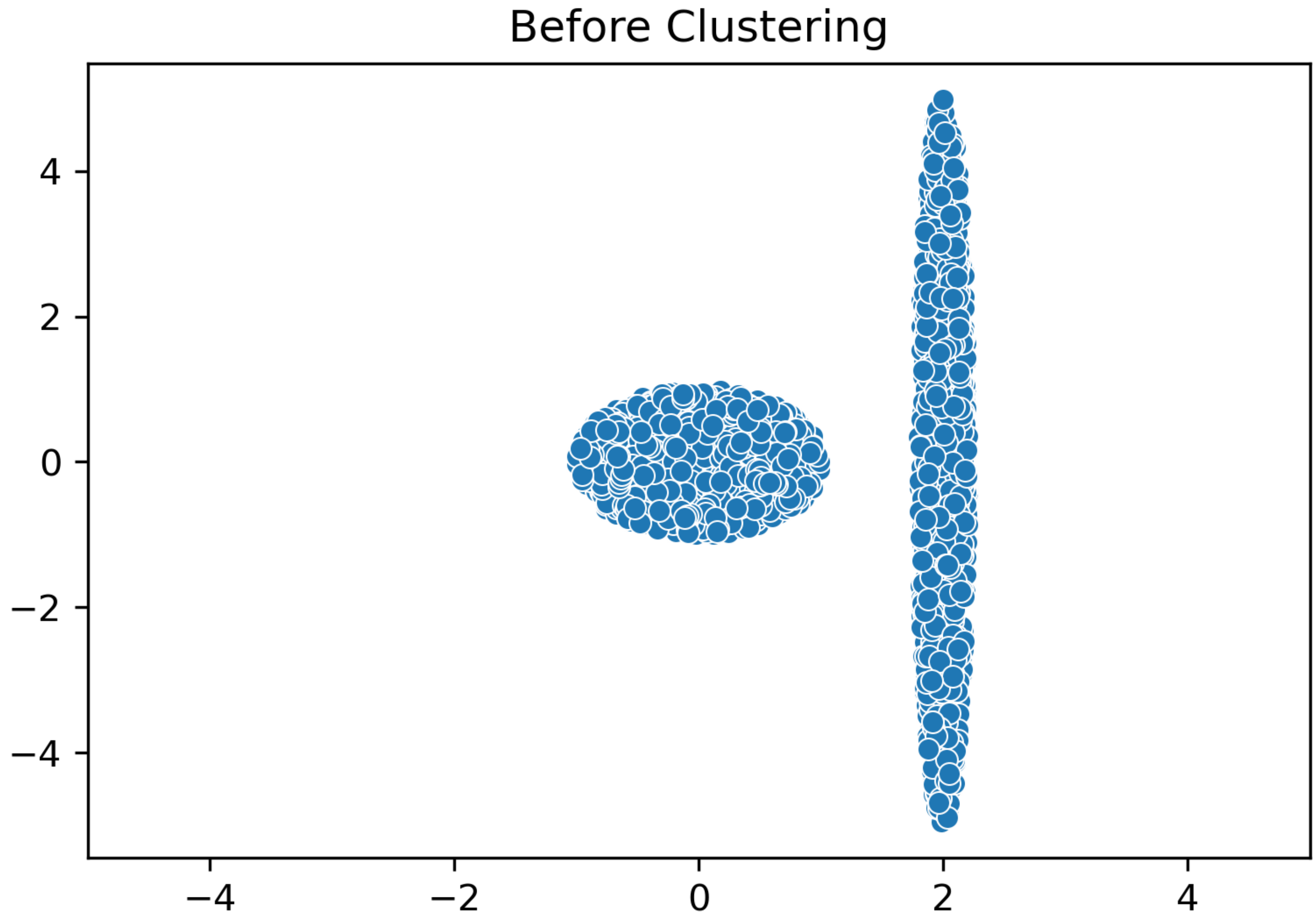
Here we are going to use two well-separated different-shaped clusters (say 1 circular and 1 elliptical) of same density. Lets see how KMeans clustering algorithm clusters it.

In [27]:

```
cir1=circles()
ellp1=np.array([2,0])+circles()*np.array([0.2,5])
diff_shape=np.vstack([cir1,ellp1])
```

In [28]:

```
plt.title('Before Clustering')
plt.xlim(-5,5)
sns.scatterplot(x=diff_shape[:,0],y=diff_shape[:,1]);
```



In [29]:
```
km5=KMeans(n_clusters=2, random_state=101)
```

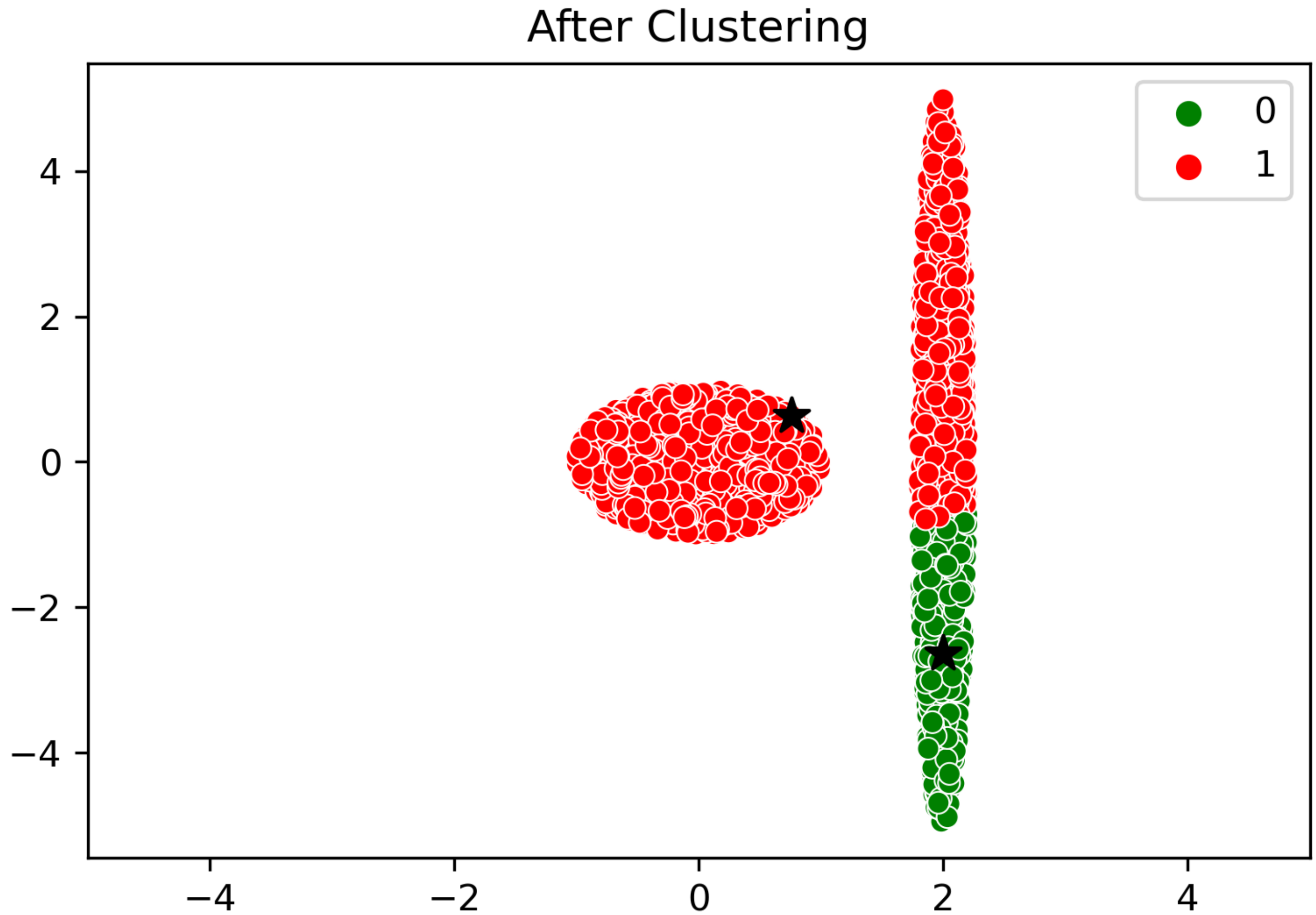
In [30]:
```
label5=km5.fit_predict(diff_shape)
label5
```

Out[30]:
```
array([1, 1, 1, ..., 1, 0, 1])
```

In [31]:
```
centroid5=km5.cluster_centers_
centroid5
```

Out[31]:
```
array([[ 1.9971303 , -2.63457346],
       [ 0.76209844,  0.63070972]])
```

In [32]:
```
plt.title('After Clustering')
plt.xlim(-5,5)
sns.scatterplot(x=diff_shape[:,0],y=diff_shape[:,1],hue=label5,palette=['Green','Red'])
plt.scatter(x=centroid5[:,0],y=centroid5[:,1],marker='*', color='Black', s=100);
```



Those two stars are the centroids of their respective clusters. In the above case, what would have been otherwise interpreted as one circular cluster and one elliptical cluster by a human, is now clustered in the above manner by the algorithm.

Note: In all the above scenarios, if the clusters were far enough, then the KMeans algorithmic clustering and our intuitive clustering would match.

Hope you found this notebook useful. Lets connect!
https://www.linkedin.com/in/fazil-mohammed-4062711b2/
https://github.com/Fazil-Math