Machine Learning Algorithms are categorized into 3 main categories :

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

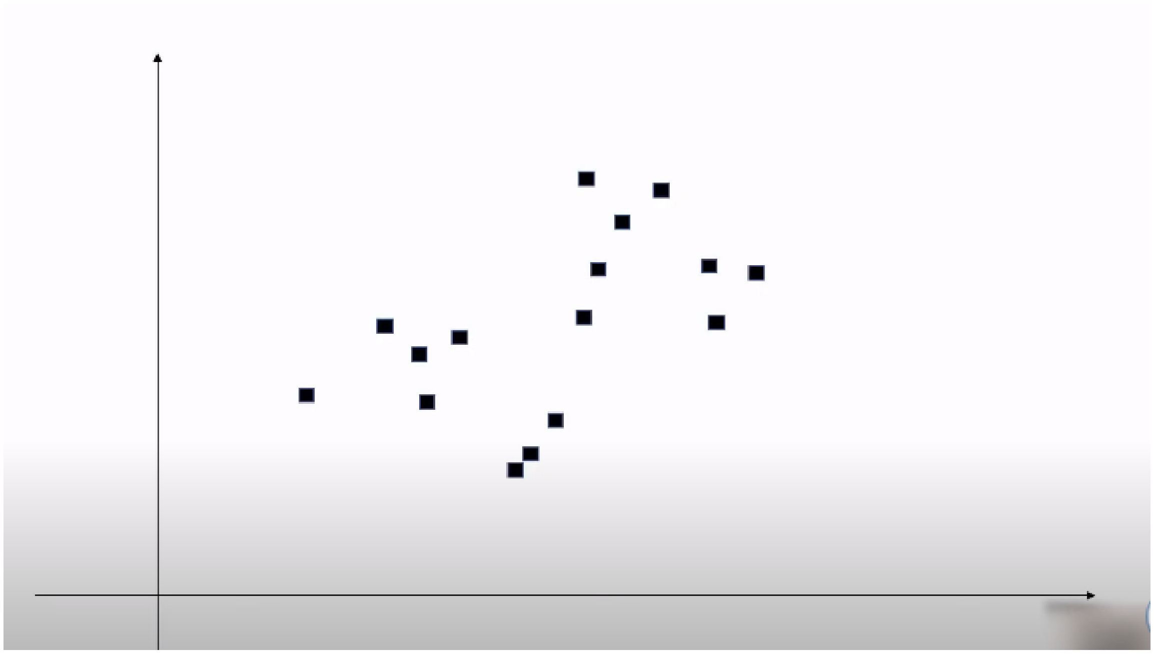
Upto now, we have looked at Supervised Learning, where in the given dataset, we have our features and target.

In unsupervised learning, all you have is set of features, you dont know about your target variable, using this dataset, we try to identify the underlying structure in that data or the cluster in the data and we can make useful predictions out of it

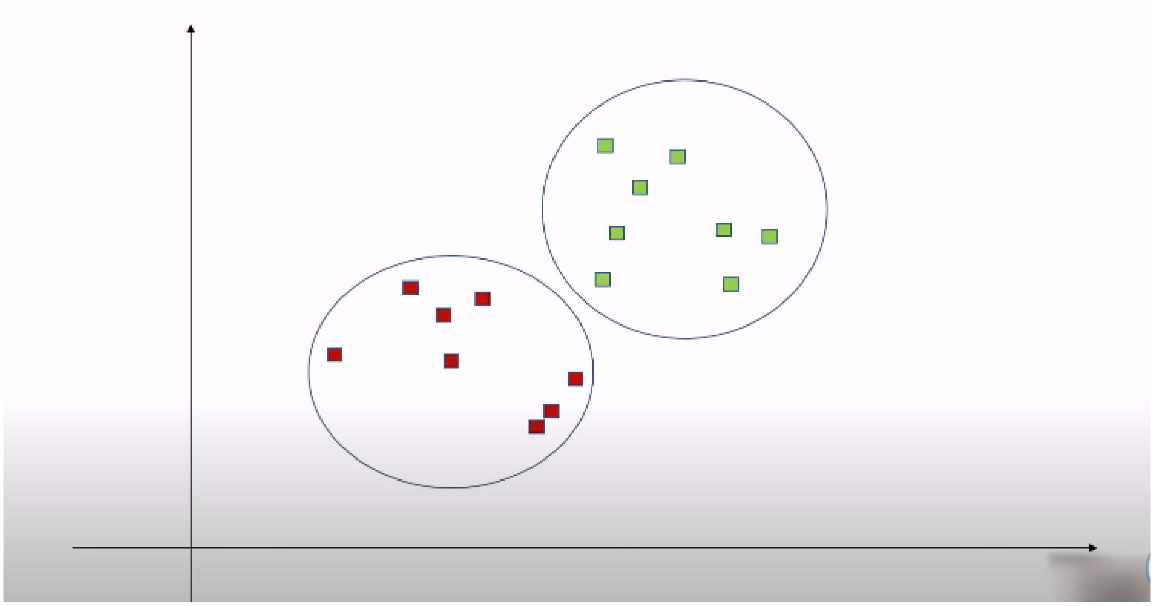
K Means Clustering

K Means is a very popular clustering algorithm and what is what we are going to look into today.

Lets say you have a data like this where X and Y axis represents two different features and you want to identify clusters in this datasets,



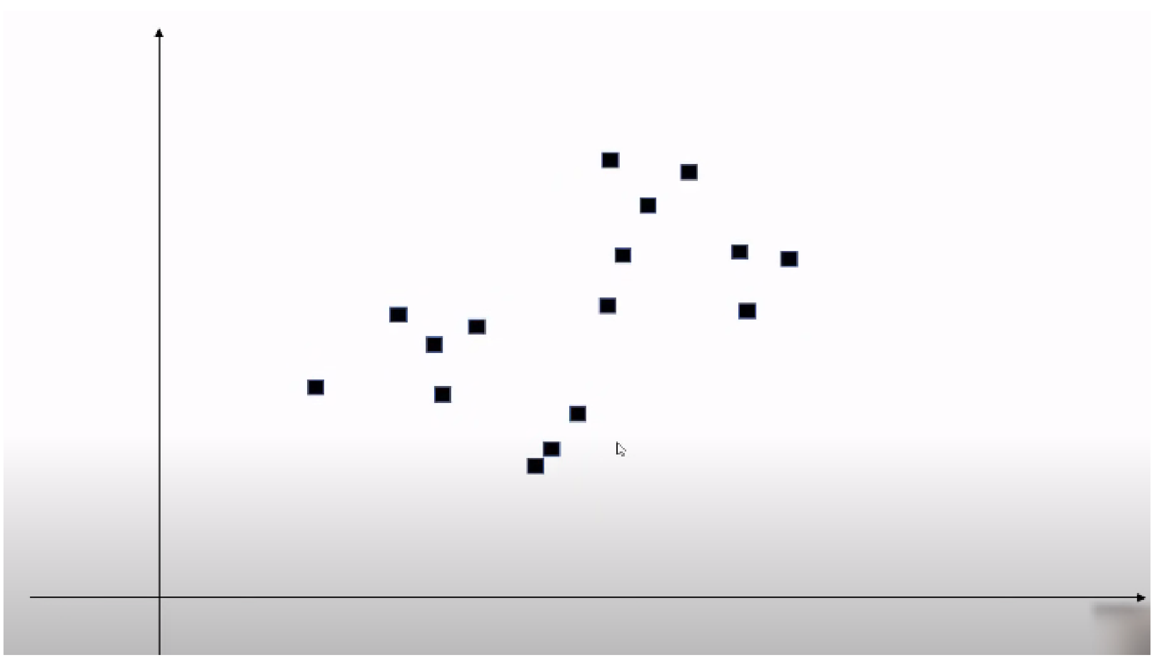
Now when the dataset is given to you, u dont have any info on the target variables so u dont know what u are looking for, all you are trying to do is identify some structure into it and one way into look at this is these two clusters



Just by Visual examination we can say that these datasets have this two clusters and K Means helps you identify these clusters,

Now, 'K' in K Means is a free parameter where in before you start the algorithm, u have to tell the algprithm what is the value of 'k' that you are looking got, here 'K' is 2.

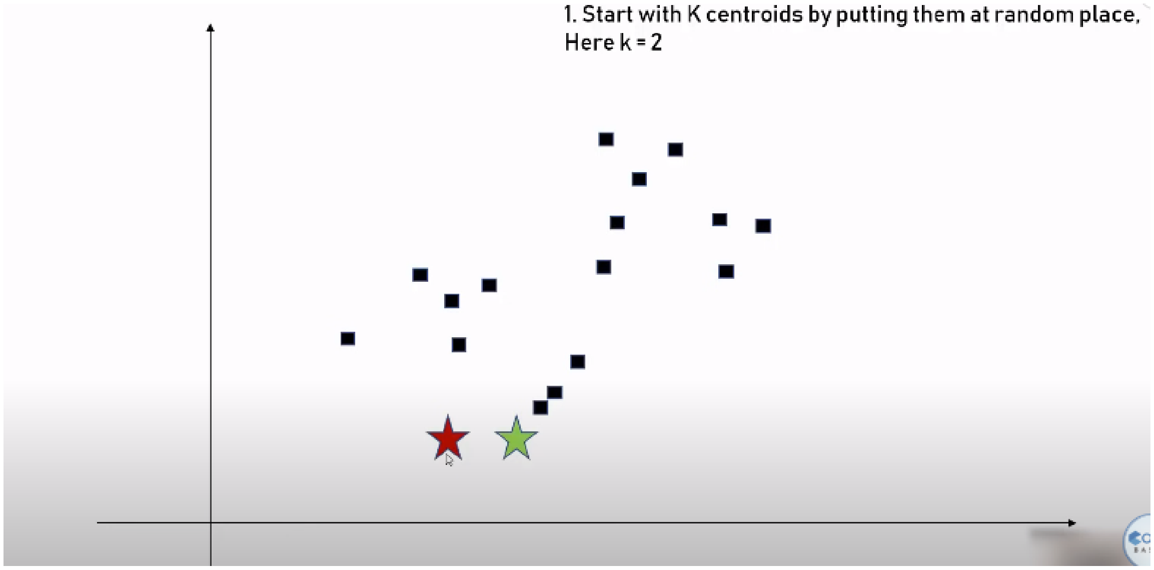
Lets say you have this dataset



You start with K = 2, and the first step is to identify two random points which you consider as the center of those clusters, We call them **Centroids**

Centroids

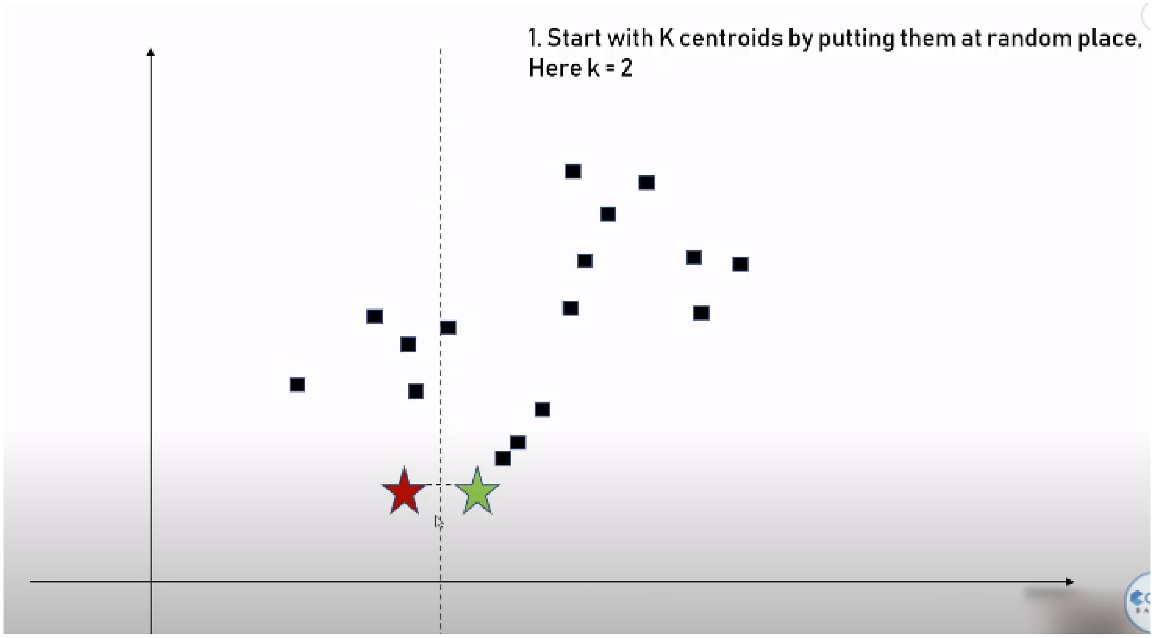
1. Start with K centroids by putting them at random place, Here K=2



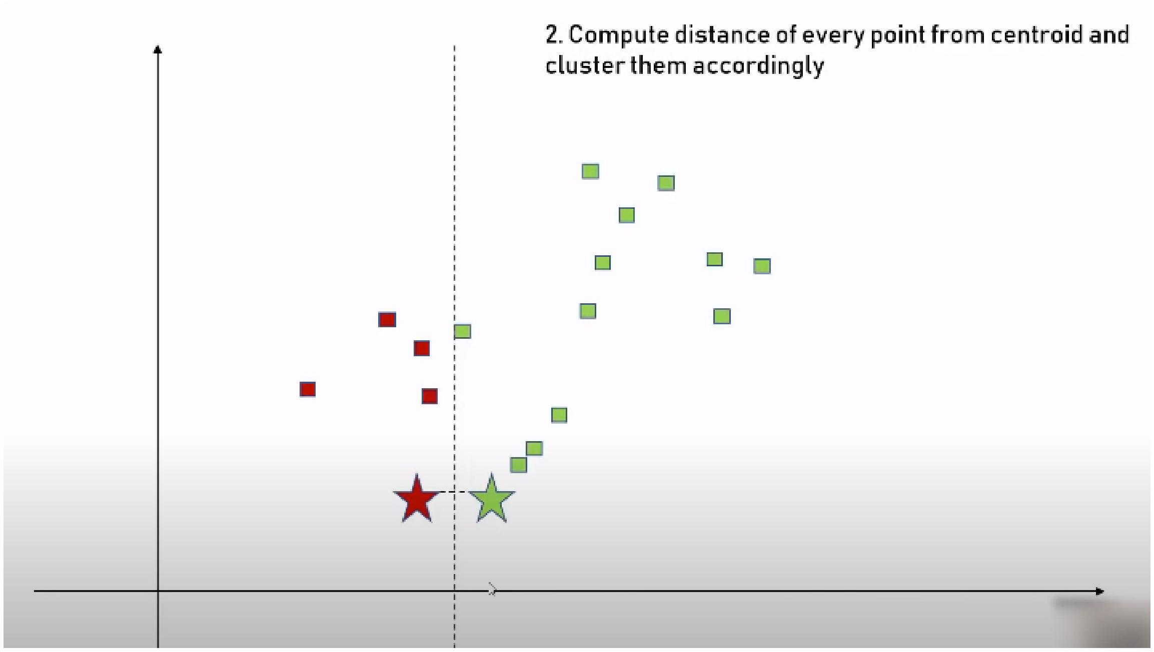
If your K was 3 then u would put 3 random points, and this could be place anywhere in this 2D place

Next step is to identify the distance of each of these data points from the centroids, for example some datapoints which are closer to the green centroid hence you can say it belongs to the green cluster, whereas if its closer to red then red cluster.

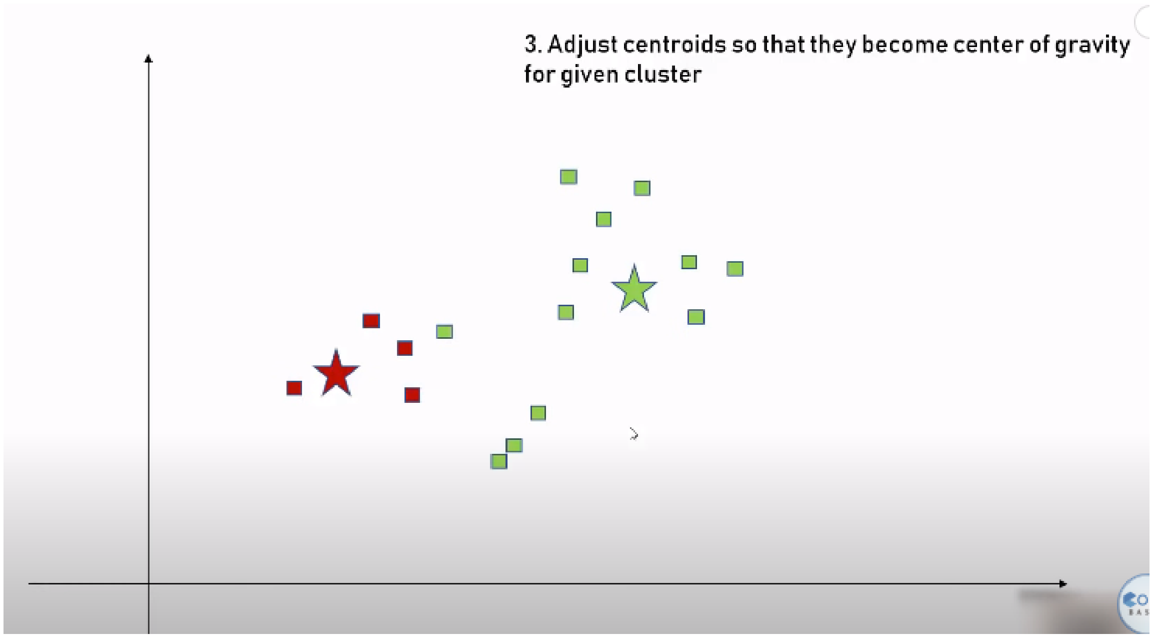
img



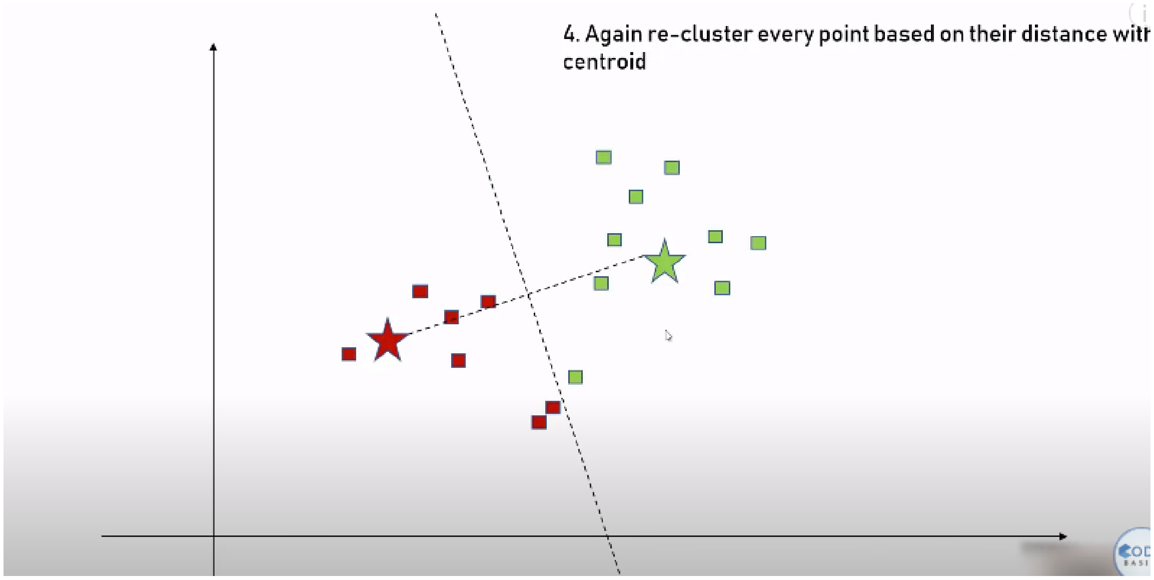
The simple mathematical way to identify the distance is to draw this kind of line and connecting a line between the two centroids. Anything on the left hand side is red cluster and anything on right is green cluster



So now we have our two imperfect clusters, and now we will try to improve these clusters, we can make it better and better at every steps, and the way u do that is, u wil try to adjust the centroids for these two clusters, for example, these red clusters which is the 4 dataponts, u will try to find the central of gravity and u will put the red centroid there and u do the same thing for the green one, so u get this below..

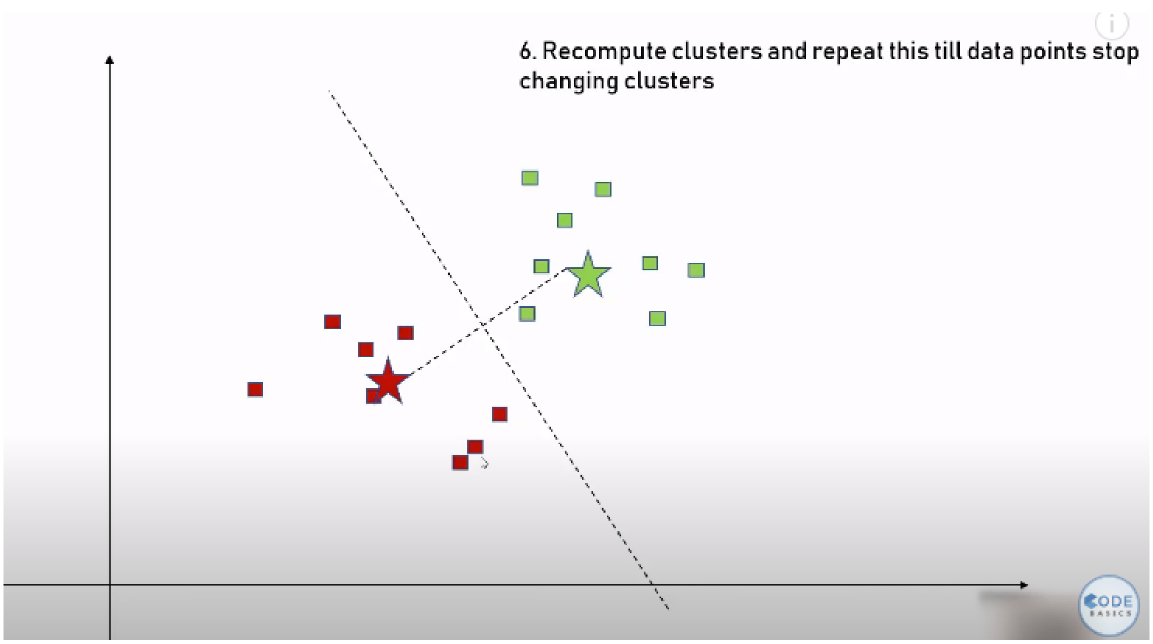


and now u do the same process again, compute the distance between each points from our cenrtoids and if the point is more near to red then u put it into the red cluster else green, as seen below now some points got change from green to red



So u basically keep on repeating these steps, u just recalculate ur centroids and u recalculate the distance of each points and readjust the clusters untill the point that none of the data points change the cluster

As you can see below, right now even if you do the step one more time, none of the points would change cluster after this, so we are done, hence, we can say that this is final. So these are now our final clusters

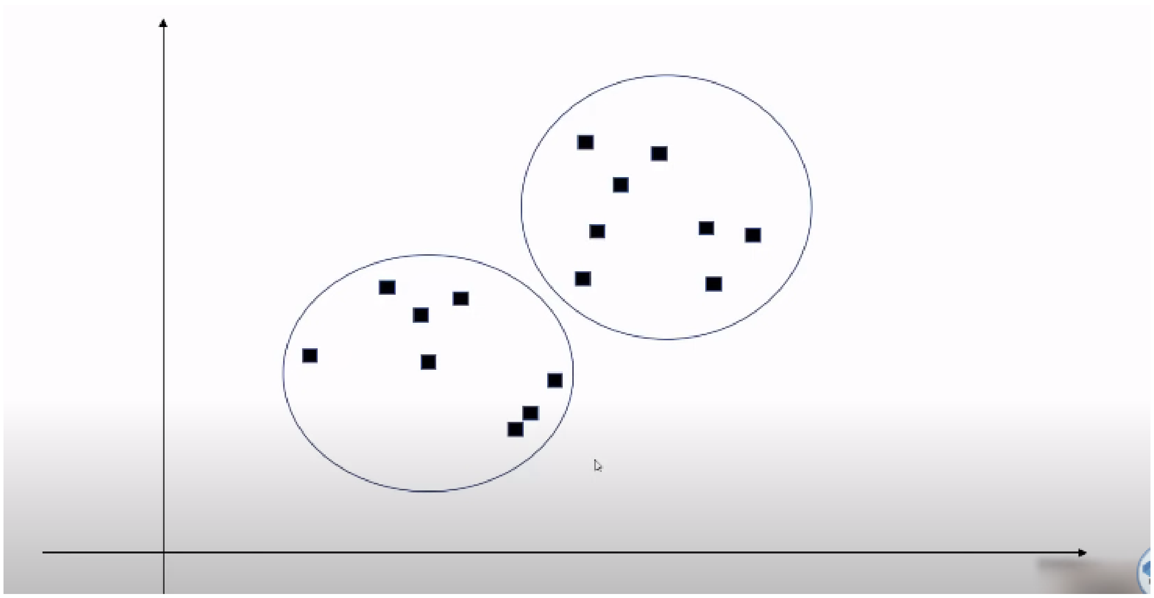


Now, the most important point here is, you need to supply 'K' to your algorithm, but what is a good number on K? cuz here we have only 2 features and 2D space but in reality u might have so many features and it is hard to visualize that data on a scatter plot, so How to determine the correct number of clusters (K)??

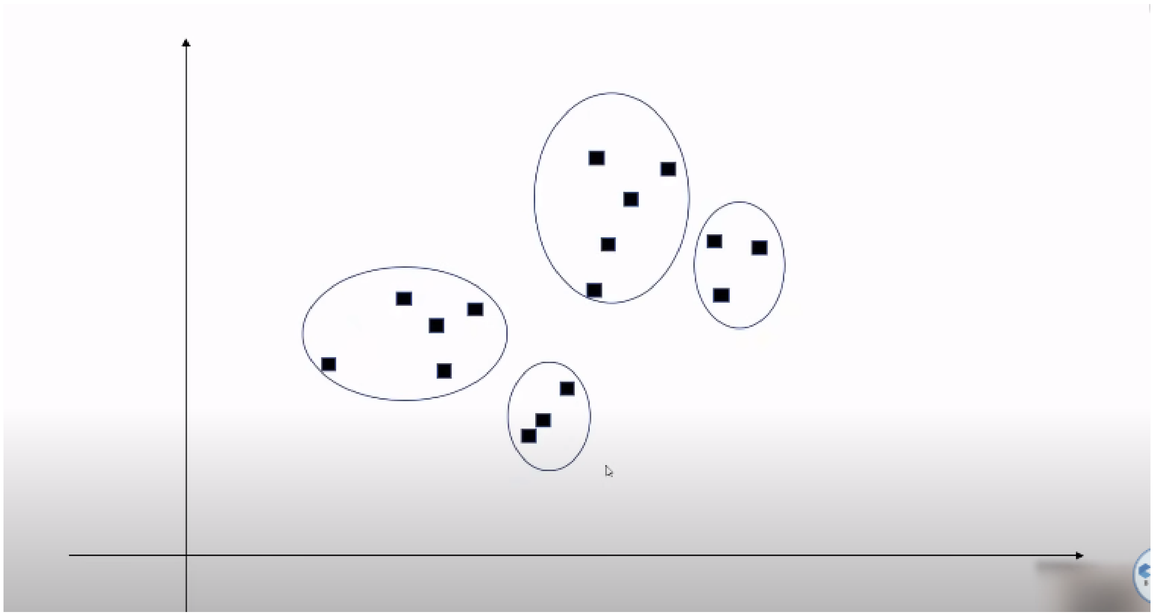
Well, there is a technique called **Elbow Method**

Elbow Method

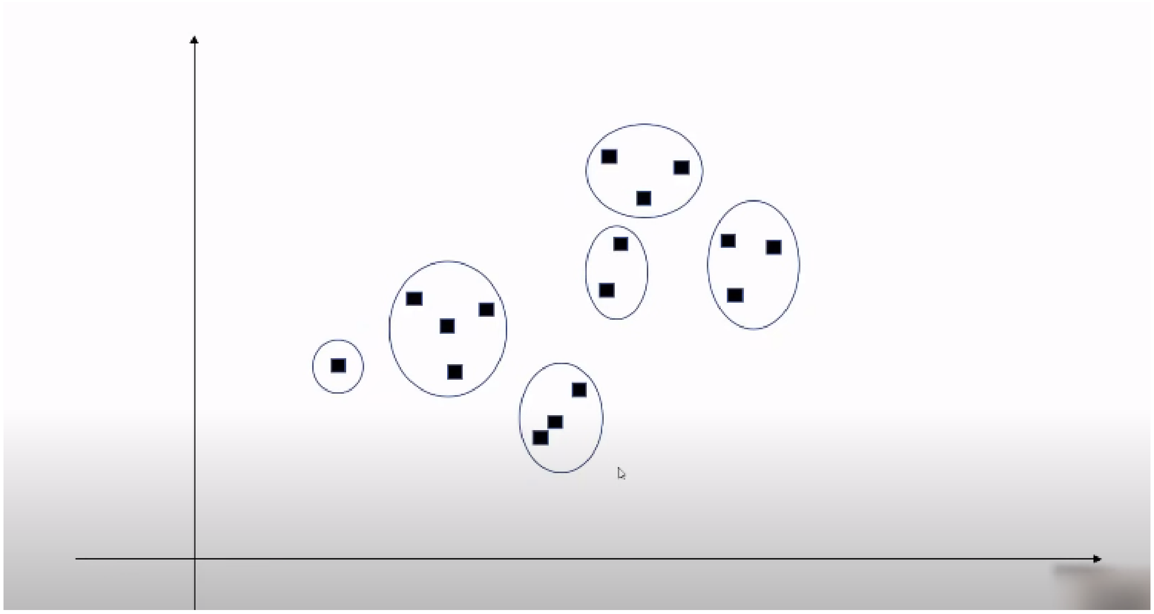
Lets look at out dataset, we started with 2 clusters



but someone might say, No these ar actually 4 clusters

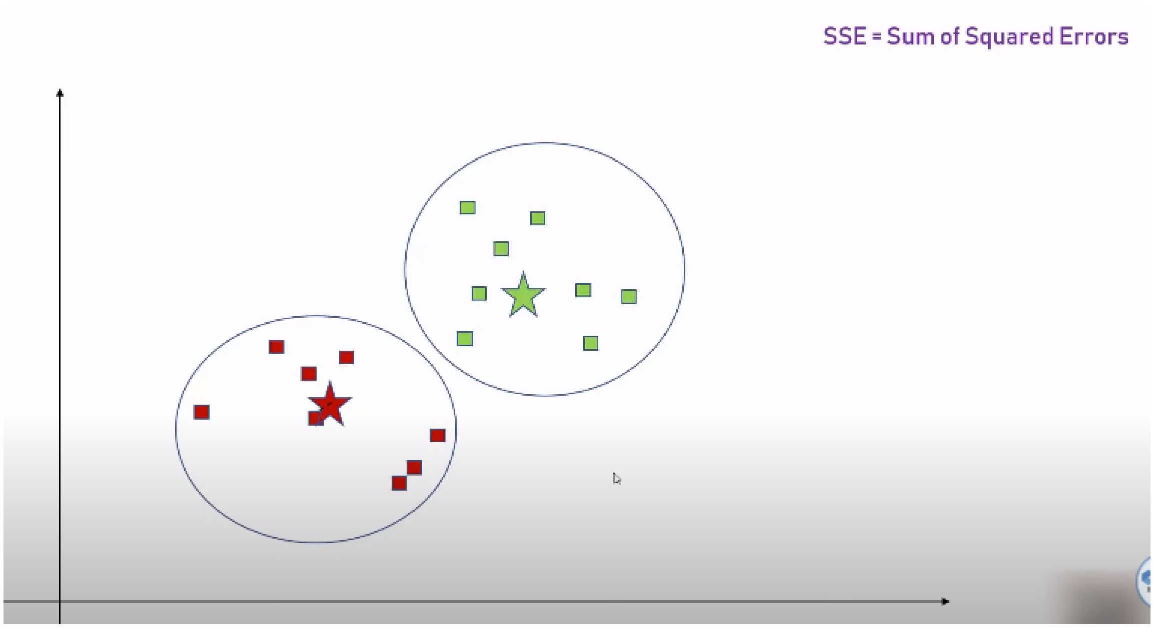


or someone might say, these are 6 clusters

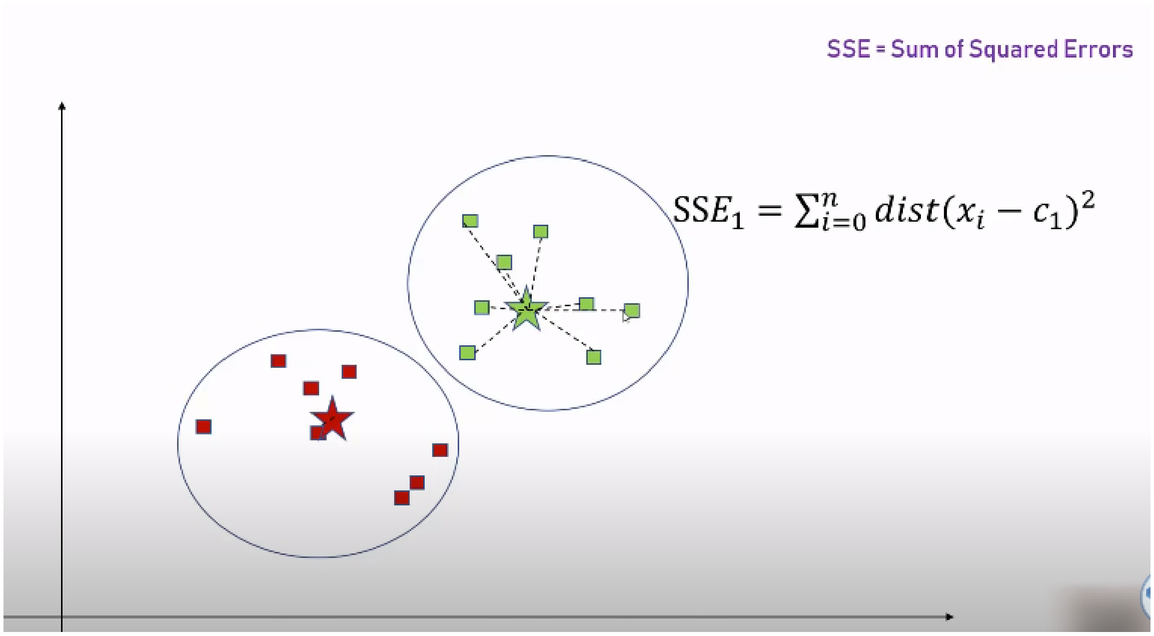


So, you can see like different ppl might intepret thing differently and ur job is to find out the best possible 'K' number, and this technique is called **Elbow Method**

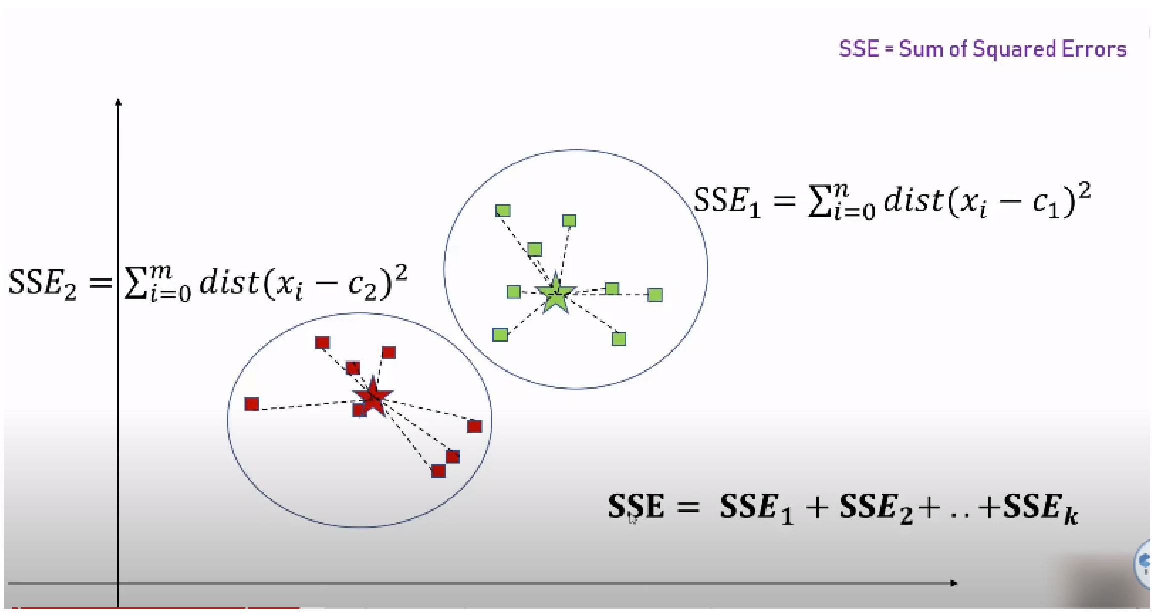
And the way this method works is, u start with some 'K', lets say we start with K = 2, and we try to compute **Sum of Squared Errors (SSE)**.



What is means is for each of the clusters, u try to compute the distance between individual datapoints from the Centroids and you square it and then you Sum it up

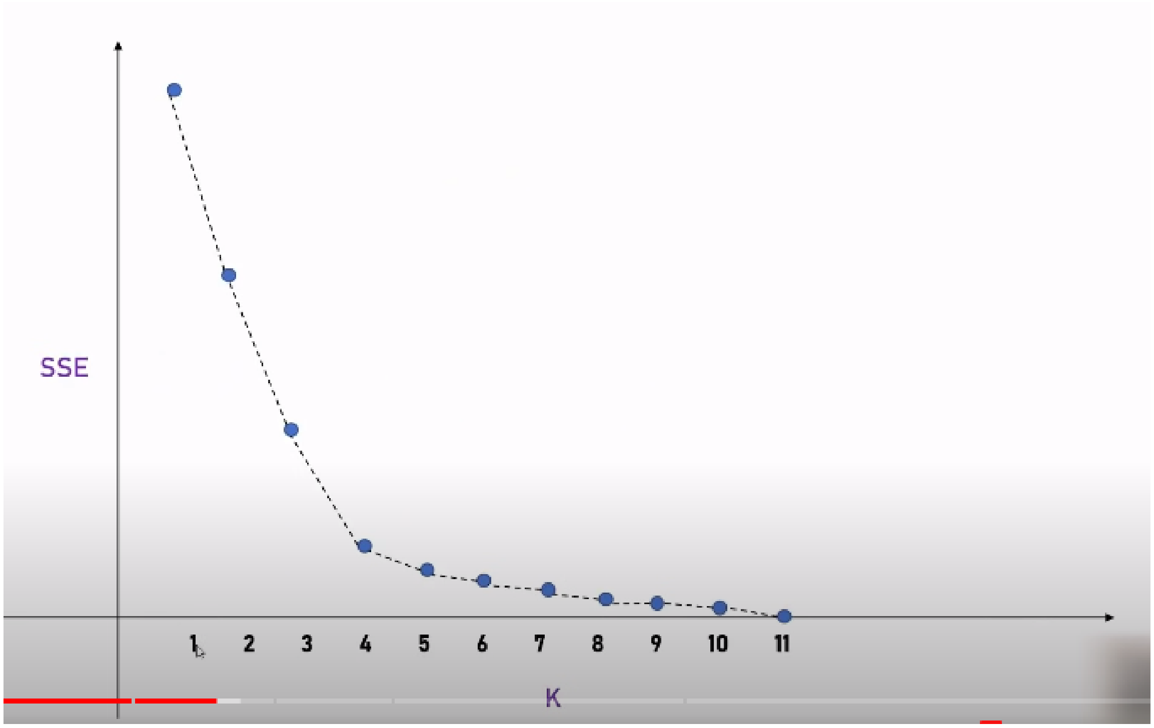


So example for green cluster we got SSE1, similary for red cluster we got SSE2 and u do this for all your clusters and in the end you Total sum of squared Errors as seen below



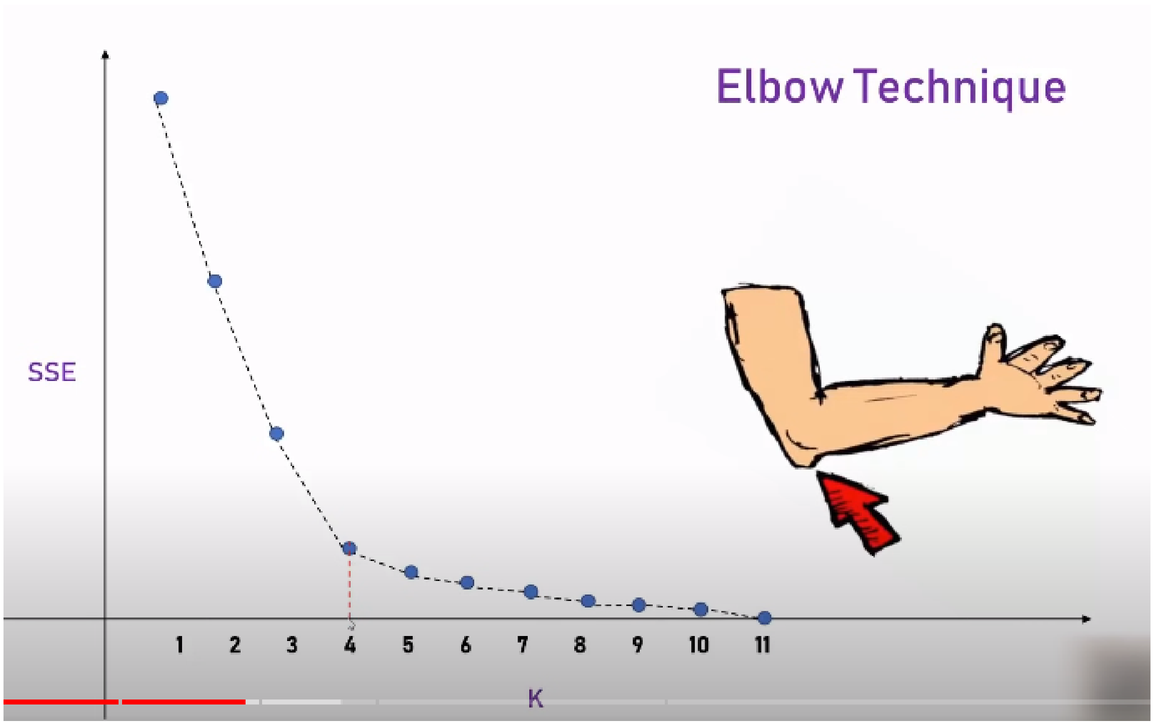
So now we computed the SSE for K=2, u repeat the same process for K=3, 4 and so on...

And once you have a number, u draw a plot like below, here we have K going from 1 t 11 in the X axis and on the Y axis we have SSE.



You will reslize that, as u increase num of Clusters, it will decrease the errors, but if u think about it, at some point u can consider all ur datapoint as one Cluster individually where ur SSE almost becomes 0, So if we assume we only have 11 data points,at 11 value of K, the error will become 0, so the more Cluster we give the Error will reduce but it also will come down to single cluster for each data points.

So what we do is, in below plot, we try to find out an 'Elbow', So the Elbow is on this chart, the 4th K value, is sort of like an 'Elbow'



So we can say that 4 can be a good cluster number

Coding Part

So the problem we are going to solve today is to Cluster this dataset where we have age and income. so by clustering these datapoints into various groups what are are tryinh to find out is some characteristics of these groups, Like maybe a group belong to a particular region in the US where the salary are higher or lower, so we try to identiy some characteristics of these groups

In [36]:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
```

In [37]:

```
data = pd.read_csv('income.csv')
data.head()
```

Out[37]:

	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000

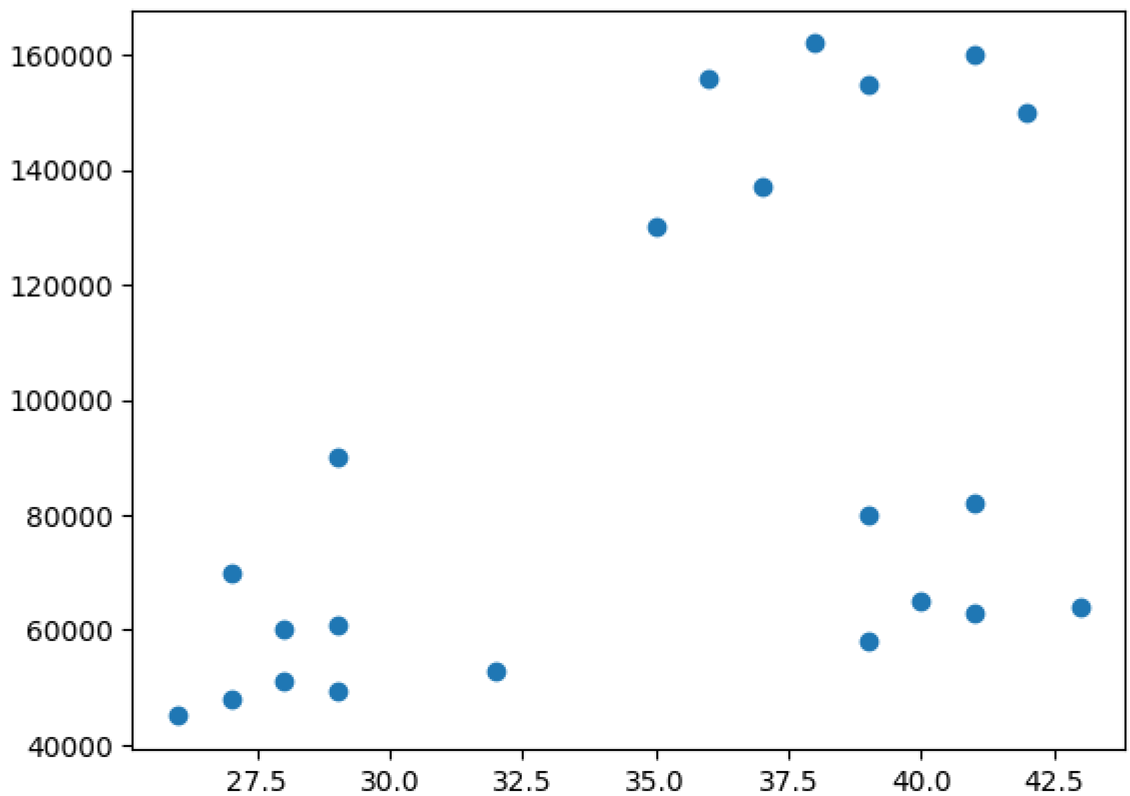
Since our data is very simple, Lets first try to plot our data in a scatter plot

In [38]:

```
plt.scatter(data['Age'], data['Income($)'])
```

Out[38]:

```
<matplotlib.collections.PathCollection at 0x2c7041a0150>
```



So, as we can see in the plot, there can be easily 3 Clusters so for this particular case choosing value of 'K' is pretty straight forward

n_clusters is the value of K

In [39]:
```
km = KMeans(n_clusters=3)
km
```

Out[39]:
```
▼ KMeans
KMeans(n_clusters=3)
```

Second step is 'Fit and Predict', previously we used to do fit and then predict, here we will just directly fit and predict

So fit and predict what? we are going to fit and predict the dataframe exluding the Name column cuz its a string

In [40]:
```
predict = km.fit_predict(data[['Age', 'Income($)']])
predict
```

```
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[40]:
```
array([0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1])
```

So now, what the above statment did is it ran KMeans algorithm on Age and Income column and it computed the Clusters as per our criteria where we told the algo to identify 3 clusters and it did it.

It assigned 3 different labels as u can see, 0, 1 and 2.

Now, visualizing this array is not fun at all, so what we want to do is, we want to plot it on a scatter plot so we can see what kind of clustering result did it produce.

First, lets append the labels to our dataframe

In [41]:
```
data['cluster']= predict
data
```

Out[41]:

	Name	Age	Income($)	cluster
0	Rob	27	70000	0
1	Michael	29	90000	0
2	Mohan	29	61000	1
3	Ismail	28	60000	1
4	Kory	42	150000	2
5	Gautam	39	155000	2
6	David	41	160000	2
7	Andrea	38	162000	2
8	Brad	36	156000	2
9	Angelina	35	130000	2
10	Donald	37	137000	2
11	Tom	26	45000	1
12	Arnold	27	48000	1
13	Jared	28	51000	1
14	Stark	29	49500	1
15	Ranbir	32	53000	1
16	Dipika	40	65000	1
17	Priyanka	41	63000	1
18	Nick	43	64000	1
19	Alia	39	80000	0
20	Sid	41	82000	0
21	Abdul	39	58000	1

So as you can see above, we can easily now see which data belongs to which group or label, i.e : 0, 1 or 2, But ofcourse its still not as good as scatter plot

Now before visualizing it on scatter plot, lets first seperate these 3 clusters into 3 different Dataframes

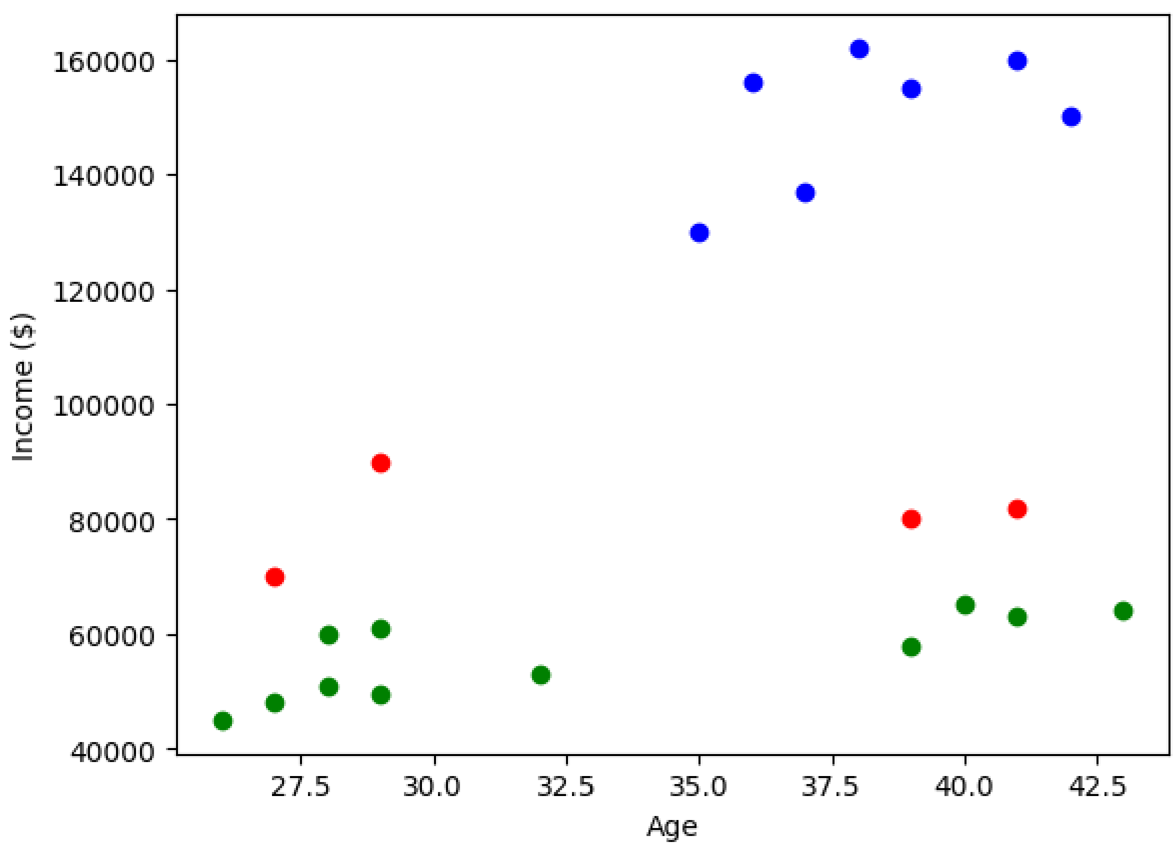
In [42]:
```
data1 = data[data.cluster == 0]
data2 = data[data.cluster == 1]
data3 = data[data.cluster == 2]
```

Now we have 3 different dataframes each belonging to 3 different Clusters, and now lets plot these 3 dataframes in a scatter plot with different colors, like cluster 0 is red, cluster 2 is green and 3 is blue

In [43]:
```
plt.scatter(data1.Age, data1['Income($)'], color="red")
plt.scatter(data2.Age, data2['Income($)'], color="green")
plt.scatter(data3.Age, data3['Income($)'], color="blue")

plt.xlabel('Age')
plt.ylabel('Income ($)')
# plt.legend()
```

Out[43]:
```
Text(0, 0.5, 'Income ($)')
```



MinMax Scaling

Loading [MathJax]/extensions/Safe.js

As you can see we have a scatter plot now with different clusters colors, but theres one problem, as u can see above, the green clusters are okay but we can see some mixup in the red and blue clusters as they arenot group correctly.

So this problem may happens cuz our Scaling is not right, because if u notice, our Y-axis is scaled from 40k to like 160k which is a massive range as compared to our X-axis which is just 0 to around 45.

So when u dont scale ur features properly, u might get into this problem, thats why we need to do some Preprocessing and use **MinMaxScaler** to scale these 2 features and then only we can run our algorithm.

So we will use MinMaxScaler, what this will do is it will try to make the Scale between 0 to 1. so after we are done Scaling, we will have a range of 0 to 1 in Yaxis and Xaxis

In [44]:
```
scaler = MinMaxScaler()

# First we will scale Income column
scaler.fit(data[['Income($)']])

# Replace income column with the new scaled income
data['Income($)'] = scaler.transform(data[['Income($)']])
data
```

Out[44]:

	Name	Age	Income($)	cluster
0	Rob	27	0.213675	0
1	Michael	29	0.384615	0
2	Mohan	29	0.136752	1
3	Ismail	28	0.128205	1
4	Kory	42	0.897436	2
5	Gautam	39	0.940171	2
6	David	41	0.982906	2
7	Andrea	38	1.000000	2
8	Brad	36	0.948718	2
9	Angelina	35	0.726496	2
10	Donald	37	0.786325	2
11	Tom	26	0.000000	1
12	Arnold	27	0.025641	1
13	Jared	28	0.051282	1
14	Stark	29	0.038462	1
15	Ranbir	32	0.068376	1
16	Dipika	40	0.170940	1
17	Priyanka	41	0.153846	1
18	Nick	43	0.162393	1
19	Alia	39	0.299145	0
20	Sid	41	0.316239	0
21	Abdul	39	0.111111	1

So u can see above now our Income column is between the range 0 to 1. Now lets do the same with our Xaxis which is Age

In [45]:

```
scaler.fit(data[['Age']])
data['Age'] = scaler.transform(data[['Age']])
data.head()
```

Out[45]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	1
3	Ismail	0.117647	0.128205	1
4	Kory	0.941176	0.897436	2

Now u can see, both our Age and Income column is between 0 and 1. Now lets use our KMeans algo once again to train our dataset

In [46]:
```
km = KMeans(n_clusters=3)
predict = km.fit_predict(data[['Age', 'Income($)']])
predict
```

```
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[46]:
```
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2])
```

So it predicted them into groups which yet we dont know how good they are, so lets Replace the cluster column with this new one again

In [47]:
```
data['cluster'] = predict
data
```

Out[47]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
4	Kory	0.941176	0.897436	1
5	Gautam	0.764706	0.940171	1
6	David	0.882353	0.982906	1
7	Andrea	0.705882	1.000000	1
8	Brad	0.588235	0.948718	1
9	Angelina	0.529412	0.726496	1
10	Donald	0.647059	0.786325	1
11	Tom	0.000000	0.000000	0
12	Arnold	0.058824	0.025641	0
13	Jared	0.117647	0.051282	0
14	Stark	0.176471	0.038462	0
15	Ranbir	0.352941	0.068376	0
16	Dipika	0.823529	0.170940	2
17	Priyanka	0.882353	0.153846	2
18	Nick	1.000000	0.162393	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	2

Now one of the things we also learned was **Centroids**, If u look at 'km' which is our KMeans instance, it has a variable called 'cluster_centers_'. And these centers are basically ur centroids

In [54]:
```
km.cluster_centers_
```

Out[54]:
```
array([[0.1372549 , 0.11633428],
       [0.72268908, 0.8974359 ],
       [0.85294118, 0.2022792 ]])
```

Above is basically the position of ur cenrtoids, for example if u look at the first centroid, [0.1372549 , 0.11633428], here 0.1372549 is ur X-axis and 0.11633428 is ur Y-axis, so we have 3 centroids.

Now lets plot this onto our Scatter plot and u will be able to see that after Scaling it using MinMax, The clusters are now structed more properly

p.s. u can plot the centroids above using the positions as shown below

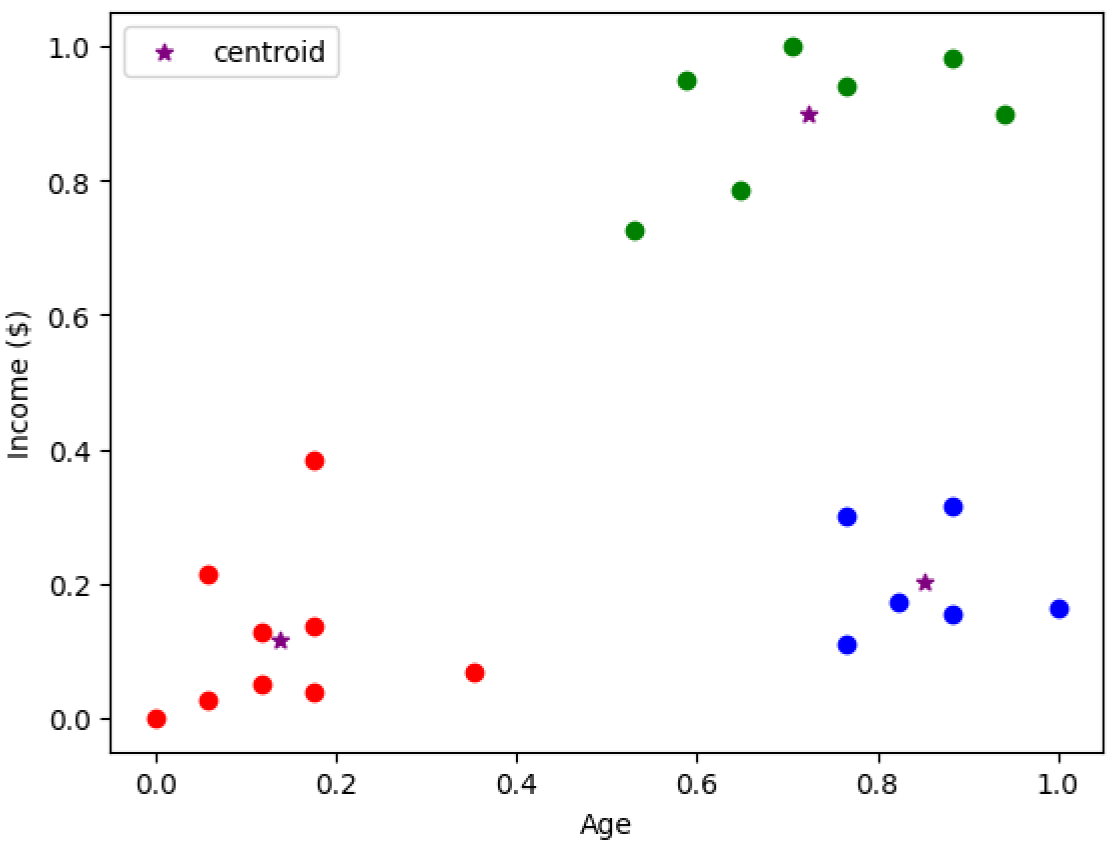
In [58]:
```
data1 = data[data.cluster == 0]
data2 = data[data.cluster == 1]
data3 = data[data.cluster == 2]

plt.scatter(data1.Age, data1['Income($)'], color="red")
plt.scatter(data2.Age, data2['Income($)'], color="green")
plt.scatter(data3.Age, data3['Income($)'], color="blue")

# Plotting the cetnroids,
# here km.cluster_centers_[:,0] is our X-axis,
# means i want to go through all the rows and the first column i.e first col
# same with km.cluster_centers_[:,1] but for 2nd column means Y-axis
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple'

plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()
```

Out[58]: <matplotlib.legend.Legend at 0x2c7059834d0>



As u see its now structed more better and we also plotted the Centroids.

Now lets look into Elbow method, this data is simple but in reality u will have complex dataset like which has 20 features or soemthing and it do be hard to even plot it in a scatter plot and it will be messy so u will be like WHAT DO I DO NOW?

Well u have **Elbow Method**.

So as we show above in Theory, we go from number of case, lets say from 1 to 11, and we try to calculate SSE and plot them and try to find the Elbow point.

So, first lets define our K range, for example lets say we want to do 1 to 10.

Then SSE will be an Array cuz we will find each SSE for the given K range and store it in this array

In [61]:

```
k_range = range(1, 10)
sse = []

# We are just itering from 1 to 9
for k in k_range:
    # Each iteration create a new model with n_cluster as k
    km = KMeans(n_clusters=k)
    km.fit(data[['Age', 'Income($)']])
    # To get SSE use 'inertia_' then will just append it to the sse array
    sse.append(km.inertia_)
```

```
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to supp
ress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Now lets check our sse

In [62]:
```
sse
```

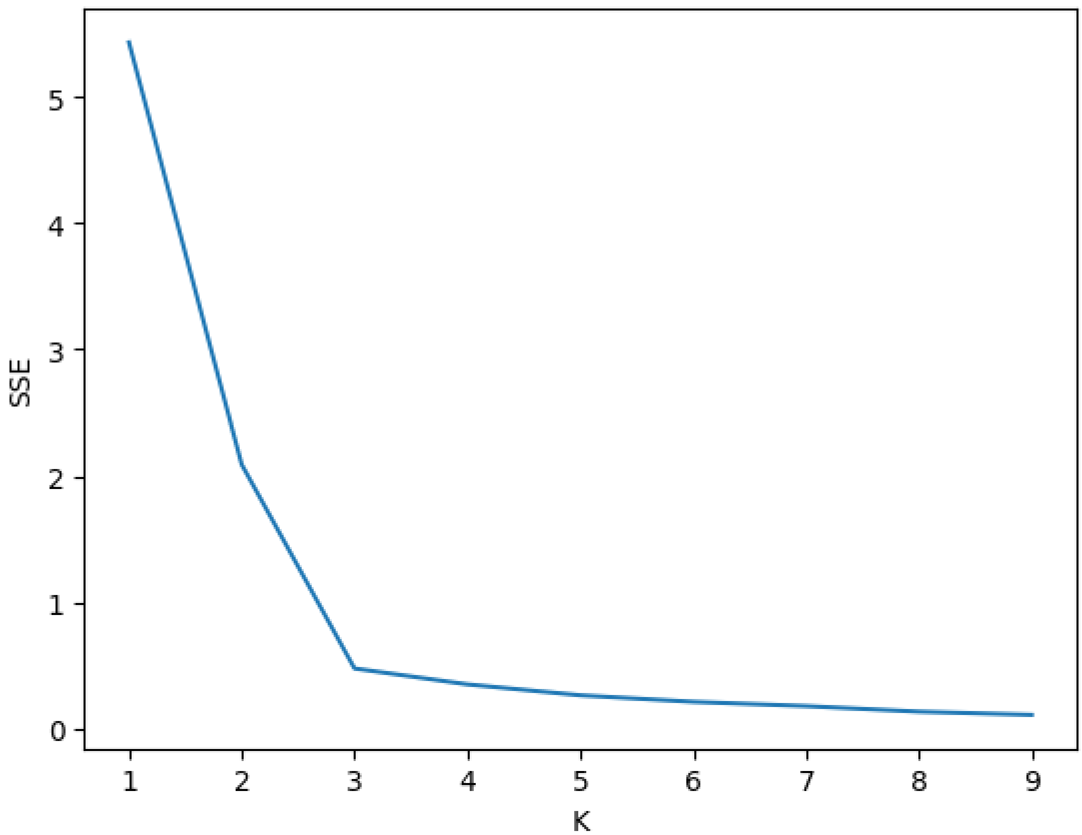
Out[62]:
```
[5.434011511988178,
 2.091136388699078,
 0.4750783498553096,
 0.3491047094419566,
 0.2621792762345213,
 0.21055478995472496,
 0.17681044133887713,
 0.13265419827245162,
 0.10740235405674733]
```

As we can see above, our SSE was high initially then it get reducing.

Now lets plot this using matplotlib and see if we can see our Elbow point

In [63]:
```
plt.xlabel('K')
plt.ylabel('SSE')
plt.plot(k_range, sse)
```

Out[63]:
```
[<matplotlib.lines.Line2D at 0x2c7059eccd0>]
```



As we can see above in the plot, it drew an Elbow lik structe and we can see that the Elbow point is at K=3, which basically means 3 will be the best Cluster number, which we already assumed above and used K as 3