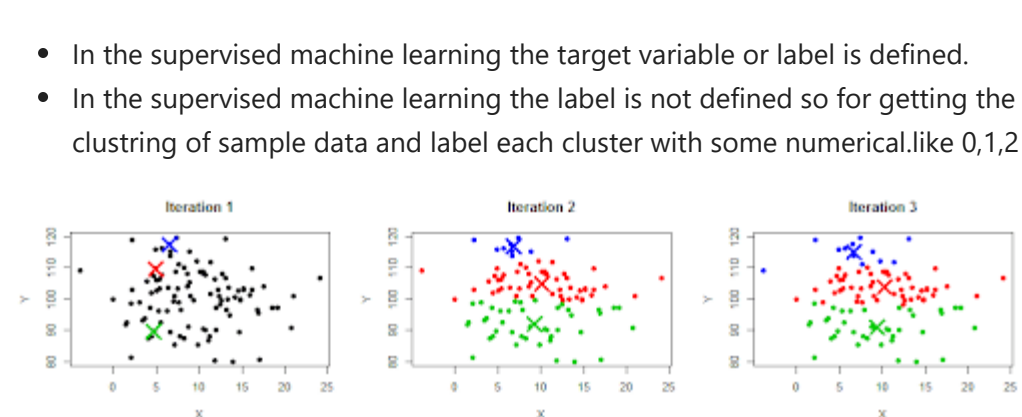
K-Means Clustering Algorithm.

Machine learning can braodly be classified three types :-

- 1.Supervised learning
- 2.Unsupervised Learning
- 3.Semi-supervised learning.

- In the supervised machine learning the target variable or label is defined.
- In the supervised machine learning the label is not defined so for getting the label on the group of the data we have to do the clustring of sample data and label each cluster with some numerical.like 0,1,2 and 3 so on.



- In this kind of unsupervied machine learning technique we will get the similar kind of observation in individual cluster whereas the data point's behaviour or characteristics from one cluster to another cluster are getting changes.
- It means each and every cluster data having different characteristics and nature.
- If we plot the scatter plot in that we make the custering(grouping) of similar kind of object or attributes those are differ from the other attributes.
- Moreover we can say here that It is an amount that reflect the straigth of the relationship between two data object or two classes.
- So,the clustering mainly use for the exploratory data mining and it used in many field such as machine learning,pattern recognition,image analysis,information retrival,bio informatics, data compression, computer graphics.
- Moreover the k-means algorithm identified that k number of centroids for example if we having the 3 grouping that means we will have the 3 numbers of centroid.
- Centroid is the each group (cluster) is the mean of that group (cluster).
- It start from the random selection of the datapoints and then it will start to conversing the final clustered one by one iteration.

Steps takes in the K-Means Clustering.

- In the K-Means clustering we do have data space inside that we have so many data points.
- While,First iteration the it will select random centroid like if we want to make the binary classification that time the centroid will be 2.
- Then it measure the distance from the that random centroid that distance might be euclidean or manhattan distance and whichever the datapoints come close to the centroid which are belonging to that particular centriod.
- Then again based on the nearest data points It will calculate the mean and consider than mean as next centroid.
- It is going happen for the some iteration until and unless the we will get the fix number of group and no point movement will there during mean is getting shited inside the data space at each iteration.

Types Of Clustering.

Hard Clustring:-

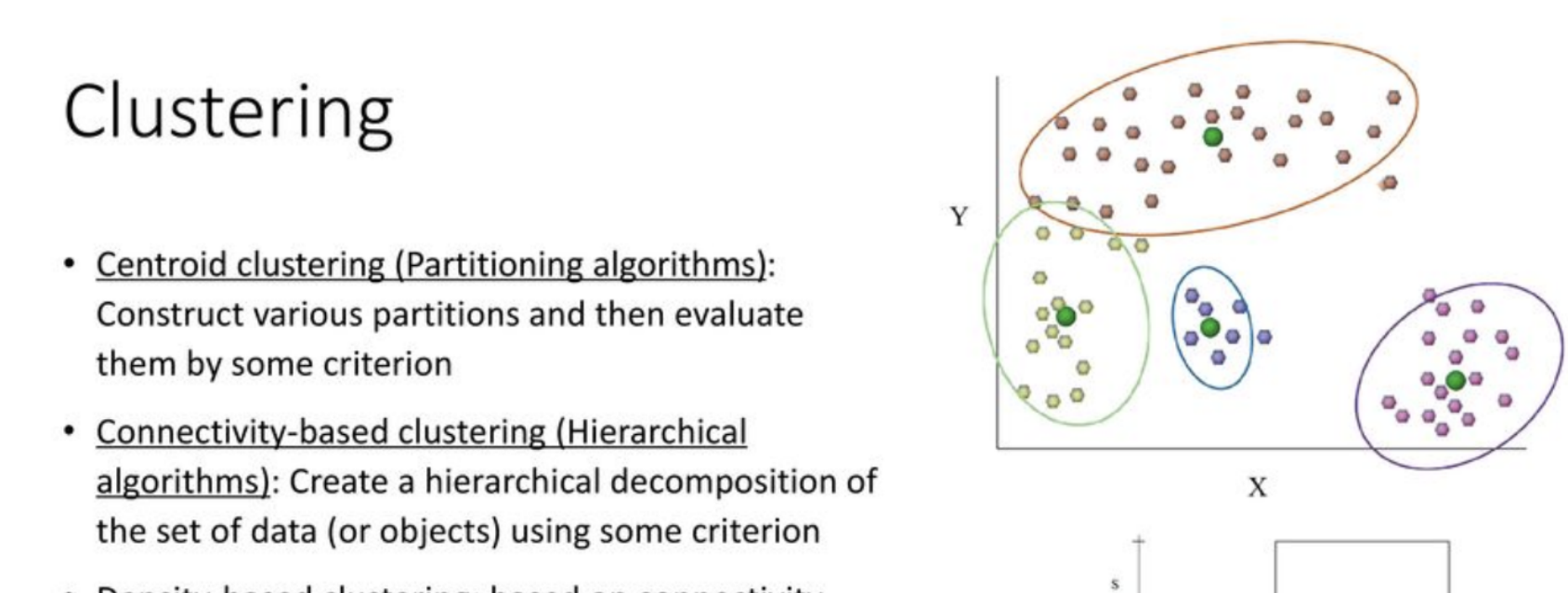
- Each data object or classes or point either belongs to cluster completly or not.In this kind of clustering points are clustered in such manner so that all attributes will belongs to the only same cluster and others.It means that both group will be the diffrent.It will never consider as part of any other cluser.Means they are perfectly classified into their classes.

Soft Clustering:-

- But in soft clustering the classification it will so strongs some points will be associated with one cluster or many cluster with some probability those neibour it or nearer to it.

Other Types Of the Clustring Algorithm.

- Connectivity Based clustering.
- Centroid Based clustering.
- Distribution Based clustering.
- Density Based clustering.



Connectivity Based Clustering :-

- The main idea behind this clustering is that data points that are the closer in the data space are more related (similar) than the to data points farther away.They are not very roubust towards outliers,which might show up as additional clusters or even cause other cluster to the merge.
- Here cluser form according to their diistances and datapoints will being connected to each are away from the centroid.
- The both side of the dataspce there could be formation of cluster and these kind of cluster connection is known as dindogram.
- This is also commonly called the **Heirarchical clustering.** This method do not produce unique partitioning of data rather heirarchi from which the user still needs to choose appropriate cluster by using the label where they want to cluster.
  - **Agglomerative:** This is a "bottom-up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
  - **Divisive:** This is a "top-down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.
  - In order to decide which clusters should be combined (for agglomerative), or where a cluster should be split (for divisive), a measure of dissimilarity between sets of observations is required. In most methods of hierarchical clustering, this is achieved by use of an appropriate **metric** (a measure of distance between pairs of observations), and a **linkage criterion** which specifies the dissimilarity of sets as a function of the pairwise distances of observations in the sets.

Centroid Based clustering.:-

- In this kind of clustering,the cluster are represented by central vector or a centroid.This centroid might not necessarily be a member of datasets.
- This is an ittrerative cluster algorithm which is the motion of the similarity derive by how close data point is to be centroid of cluster.

Distribution Based Clustring.

- This model having the strong therotical foundation,howeever they ofter suffer for the overfitting.Gaussian mixture models,using expectation-maximization algorithm is the fomous distribution based clustring method.
- This is clustring is closed to the stastical distribution modeling.
- Datapoints are belongs the same distribution and model have strong therotical foundation however they often suffer from overfitting.

Density Based Clustering

- Density based methods search the data space for the area of varied density of data points.
- Cluster will define the area of highly density within the dataspace to region.

Working With K-Means Clustering.

In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [11]:
```
#for indexing the columns used index_col whenever uname is avalible that time use the index_col =0
data = pd.read_csv('data.csv',index_col=0)
data.head()
```

Out[11]:

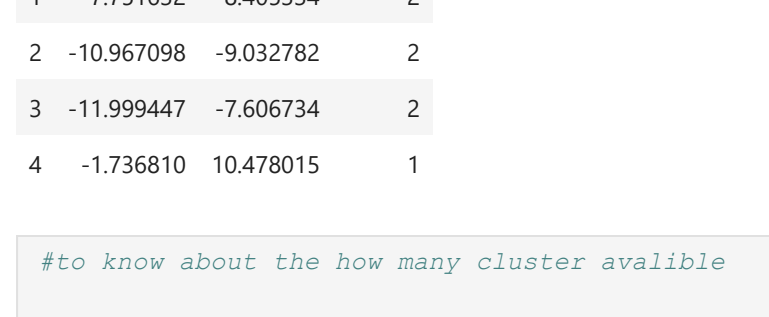
	x	y	cluster
0	-8.482852	-5.603349	2
1	-7.751632	-8.405334	2
2	-10.967098	-9.032782	2
3	-11.999447	-7.606734	2
4	-1.736810	10.478015	1

In [12]:
```
#to know about the how many cluster avalible

data['cluster'].value_counts()
```

Out[12]:
```
1    67
0    67
2    66
Name: cluster, dtype: int64
```

- Please check this to get more about the cmap:-https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
- Here if we can see that there is no any perfect the clustering has been done due to some points these are the yellow and voilet....

In [13]:
```
#c based on the clustering
#cmap (colormap)in the python
plt.scatter(data['x'],data['y'],c=data['cluster'],cmap='viridis')
plt.show()
```



In [14]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

- We had created the vector of x and y
- We had taken out the two columns x and y from the data datasets.
- In unsupervised machine learning we don't need any labeling but for comparision we will using this final output y as well.

In [15]:
```
#We had created the vector of x and y

x = data[['x','y']]  #we had taken out the two columns x and y from the data datasets.
y = data['cluster']  #In unsupervised machine learning we don't need any labeling but for comparision we will u
```

If we see here x is neither standardized or normalized..

In [16]:
```
#If we see here x is neither standardized or normalized
x
```

Out[16]:

	x	y
0	-8.482852	-5.603349
1	-7.751632	-8.405334
2	-10.967098	-9.032782
3	-11.999447	-7.606734
4	-1.736810	10.478015
...	...	...
195	-8.820126	-9.479259
196	-1.573419	-6.650994
197	-2.619581	8.269253
198	-2.634418	6.697531
199	-11.951634	-7.121327

200 rows × 2 columns

First we need to stardardized and normalized the data by using Standardscaler.

In [18]:
```
scaler = StandardScaler()
x = scaler.fit_transform(x)
x
```

Out[18]:
```
array([[-1.01200363, -0.60606415],
       [-0.86550679, -1.04265203],
       [-1.5097118 , -1.14041707],
       [-1.71653856, -0.91821912],
       [ 0.33953731,  1.89963378],
       [ 0.38183922,  1.1890842 ],
       [ 0.24668446, -0.74232845],
       [-1.44313178, -0.76182643],
       [ 0.32130834, -0.75849023],
       [ 0.83110494,  1.27684537],
       [ 0.31775088, -0.39338956],
       [ 1.20365218, -0.4290876 ],
       [-0.07305683,  1.51966781],
       [ 0.63827173,  0.98134203],
       [ 1.52205697, -0.42280098],
       [ 0.17043691,  1.06578799],
       [ 1.5381542 , -0.52935538],
       [ 0.27161133, -0.71651135],
       [-1.35289151, -0.93641563],
       [ 0.79308339, -0.35212834],
       [ 0.14633873,  2.31330628],
       [ 1.14228061, -0.30354302],
       [ 1.12503491, -0.04177872],
       [-1.17977302, -1.09578279],
       [-2.01238779, -0.86540408],
       [ 0.19152374,  1.57228952],
       [ 0.83261025,  1.80799546],
       [ 0.66508288, -0.6110459 ],
       [-1.12827817, -0.15250042],
       [ 0.52241815, -0.65585313],
       [ 1.08583214,  0.93547726],
       [-0.97693538, -0.88741691],
       [-1.33595694, -0.63086238],
       [ 0.19953335,  1.28083486],
       [-0.94950185, -0.56065403],
       [ 0.66982823, -0.13484351],
       [ 0.771665  , -0.5499059 ],
       [ 0.2774257 ,  1.22860396],
       [-0.73472586, -1.2187408 ],
       [-0.63187335, -1.00655651],
       [-0.69416438, -1.0000677 ],
       [ 0.83271023, -0.6731856 ],
       [-1.23344655, -1.15789171],
       [ 0.65096239, -0.19455043],
       [-1.39325586, -1.1523753 ],
       [-1.49484828, -0.92704231],
       [-1.45695253, -0.70502951],
       [ 0.5305949 ,  0.81280245],
       [ 0.83400406, -0.42253729],
       [ 0.48904363,  1.24633059],
       [ 1.23175068, -0.82532629],
       [ 0.69624505, -0.30159923],
       [ 0.39391193, -0.396964  ],
       [ 0.79094876,  1.60143791],
       [ 0.15899074,  1.26432118],
       [ 0.39413305,  1.76704755],
       [ 0.86405762, -0.59555499],
       [-2.03584663, -0.69245567],
       [ 0.59854512,  1.21885094],
       [ 0.46524738,  1.14933569],
       [ 1.13205009, -0.79021258],
       [-0.43326051,  1.24378251],
       [ 1.18403178, -0.2105801 ],
       [ 1.104018  , -0.45293091],
       [ 0.90917625, -0.41741704],
       [ 0.24940112,  1.47054385],
       [-1.34378511, -0.96019197],
       [-1.15864684, -0.53793509],
       [-1.38106913, -1.24446575],
       [ 1.32888112, -0.15656749],
       [-1.97542172, -0.86113934],
       [ 0.55709844,  0.94043404],
       [-1.16375964, -1.04708045],
       [-0.92116118, -0.60408404],
       [ 0.19744613, -0.320971  ],
       [-1.17058201, -0.94932292],
       [-1.30394578, -0.6337949 ],
       [-1.3764337 , -0.98303292],
       [ 0.29643725, -0.16983389],
       [-1.48660911, -1.07632091],
       [ 0.88544854,  0.85471602],
       [ 0.9006815 , -0.4157147 ],
       [ 0.2206326 ,  1.52350226],
       [ 1.13326512,  1.24514921],
       [ 1.59514484, -0.57810326],
       [ 0.15529192, -0.46145428],
       [ 1.07078756, -0.73342709],
       [-1.84390846, -0.81992791],
       [ 0.90990583,  1.40916629],
       [ 0.3576306 , -0.73136536],
       [ 1.29042424, -0.89780574],
       [-1.36084266, -0.77007109],
       [-1.0236826 , -1.24868072],
       [ 0.7946996 , -1.12120567],
       [ 0.79884711,  1.66676256],
       [ 0.99941321,  1.13889426],
       [-1.36837982, -1.15198567],
       [-0.3156759 , -1.46559894],
       [ 0.70861868, -0.75627388],
       [-0.21082427, -0.41430294],
       [ 0.32589705,  1.68264012],
       [ 0.66619897,  1.21855463],
       [ 0.49361345,  0.86887702],
       [ 0.5068162 , -0.32337893],
       [ 0.74177357, -0.61694863],
       [-1.2885399 , -0.96008709],
       [ 1.02551131,  1.48533141],
       [ 0.43201879,  1.6076944 ],
       [ 1.08249536, -0.21716763],
       [-0.15265406,  0.0228466 ],
       [ 0.97586982, -0.63467294],
       [-1.10238107, -0.75993911],
       [ 0.24621536, -0.32010977],
       [ 0.53736804,  1.0345916 ],
       [ 1.21425236,  1.07790672],
       [-0.00967353,  1.27374229],
       [ 0.53207638,  1.10912864],
       [-2.00277598, -1.07059812],
       [ 1.33013807, -0.90971954],
       [-1.25463745, -1.18101253],
       [ 0.53775645,  1.29669311],
       [ 0.27952731, -0.01116382],
       [ 0.32201814,  1.62673893],
       [ 0.4329941 , -0.57783171],
       [ 0.57365299,  1.3368478 ],
       [ 0.65082523,  1.46367753],
       [ 0.65445327,  1.46194122],
       [ 0.30583921,  0.15038392],
       [ 0.58866226, -0.19466067],
       [ 0.77058657, -0.65992081],
       [-1.73473402, -0.85951427],
       [ 1.21530606, -0.27942699],
       [ 0.37146484,  1.57464784],
       [ 0.13101932,  1.17557222],
       [ 0.55048135, -0.073215  ],
       [ 0.65268228, -0.76005108],
       [-0.12524786,  1.16172152],
       [ 0.9416221 , -0.8284188 ],
       [ 0.72962605, -0.78012446],
       [-0.7595301 , -1.02429527],
       [ 0.42537401,  2.05365336],
       [ 0.41686597,  1.17237934],
       [-1.36156287, -0.58182547],
       [-1.63215702, -0.80053828],
       [ 0.16868097,  1.78349137],
       [-1.25198849, -1.21104408],
       [ 0.45461854,  1.24370166],
       [-1.99356038, -0.90548851],
       [ 0.84572607,  1.2164603 ],
       [ 1.11029307, -0.23099879],
       [-0.56892205, -0.70265972],
       [ 0.93402653,  1.29095874],
       [ 0.94118216, -0.72375673],
       [ 0.23509561,  0.96041989],
       [ 0.60301693,  1.08704203],
       [-1.1171891 , -1.21457231],
       [ 1.21399433,  1.29509665],
       [-1.25249385, -0.47819655],
       [-0.986743  , -0.98144191],
       [-1.7860204 , -1.39231882],
       [ 0.55320627,  1.41715403],
       [-2.0662249 , -0.69934998],
       [-1.29343007, -0.77757608],
       [ 0.27276956,  0.9418664 ],
       [ 0.5434236 , -0.41841969],
       [ 1.44526084,  0.0087187 ],
       [-0.34167386,  1.31912378],
       [-1.77472219, -0.84542156],
       [ 0.8071665 ,  1.92415722],
       [-1.41813112, -0.95059979],
       [ 0.26971005,  1.01953102],
       [ 0.90067843, -0.28122456],
       [ 0.65661936,  0.89004119],
       [ 0.42245075, -0.4677546 ],
       [-1.75542164, -0.6517002 ],
       [ 0.57522939,  1.23645334],
       [ 0.4217509 ,  1.70837732],
       [-1.23985549, -0.31419264],
       [-1.50342086, -0.14237115],
       [ 0.77945616,  1.27706047],
       [ 0.9279614 , -0.16082871],
       [-1.39989229, -0.77754158],
       [ 0.94719192, -0.41984737],
       [-2.06564945, -0.83849631],
       [-0.71883787, -0.62819834],
       [-1.17765783, -1.01506935],
       [ 0.70341832, -0.73729614],
       [-1.26359243, -0.85907062],
       [-1.08899436, -0.88801288],
       [ 1.132047  , -0.55152432],
       [ 0.52620981,  1.43321291],
       [ 0.74377108, -0.4080321 ],
       [ 0.77533462, -0.82445019],
       [ 1.43342856, -0.2291756 ],
       [ 0.72870678,  1.06563353],
       [-1.07957503, -1.20998434],
       [ 0.37227198, -0.76930174],
       [ 0.16267793,  1.55547825],
       [ 0.15970539,  1.31058228],
       [-1.7069593 , -0.84258607]])
```

We have to give the name as x to the columns x and y.

In [20]:
```
#we have to give the name as x to the columns x and y
data[['x','y']] = x
```

We already fitted our data it's means training is done.

In [21]:
```
#we already fitted our data it's means training is done.
kmeans = KMeans(n_clusters =2,random_state =42)
kmeans.fit(x)
```

Out[21]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

In [22]:
```
kmeans.labels_
```

Out[22]:
```
array([0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0])
```

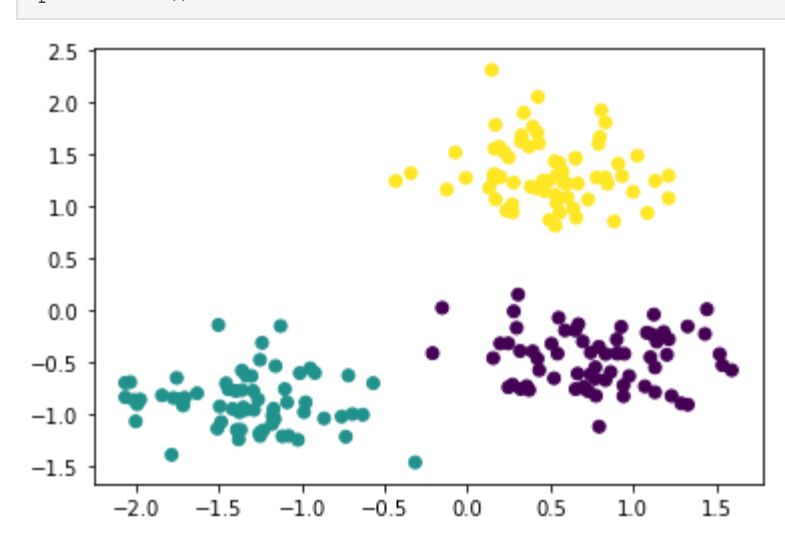
Here we having only two centerse that's why.

In [23]:
```
#here we having only two centerse that's why
kmeans.cluster_centers_
```

Out[23]:
```
array([[-1.30618271, -0.87560626],
       [ 0.64334372,  0.43126875]])
```

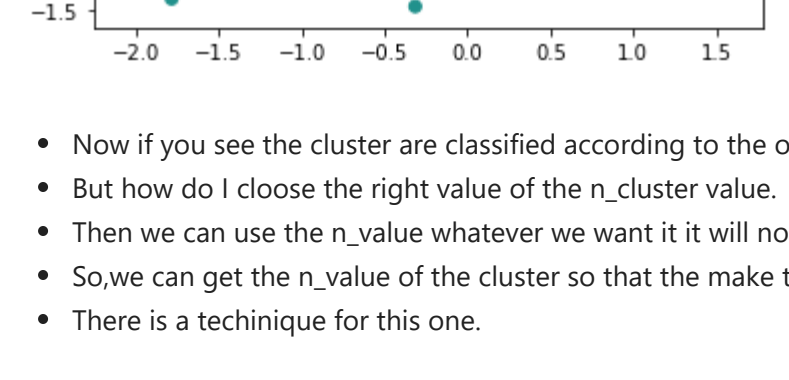
- Now, do the scatter plot and check the how the data being clustered.
- In the above the code we had used the 2 cluster .If we use he scatter with the 3 cluster that time we will get the 3 cluster in the sactter plot.

In [24]:
```
#Now, do the scatter plot and check the how the data being clustered
#In the above the code we had used the 2 cluster .If we use he scatter with the 3 cluster that time we will get
plt.scatter(data['x'],data['y'],c=kmeans.labels_,cmap='viridis')
plt.show()
```



- If we use the 3 clustering at place of 2 then...

In [25]:
```
kmeans = KMeans(n_clusters =3,random_state =42)
kmeans.fit(x)
kmeans.labels_
kmeans.cluster_centers_
#In the above the code we had used the 2 cluster .If we use he scatter with the 3 cluster that time we will get
plt.scatter(data['x'],data['y'],c=kmeans.labels_,cmap='viridis')
plt.show()
```



- Now if you see the cluster are classified according to the our convenience of perfect classification.
- But how do I cloose the right value of the n_cluster value.
- Then we can use the n_value whatever we want it it will not always give us the better clustering over the scatter plot so
- So,we can get the n_value of the cluster so that the make the data clustring precisely.
- There is a techinique for this one.

How do choose the right value of K ?

You need to understand how it works ?

- Assuming we having input X_1,X_2,X_3,...,X_nx
- Pick k-random points are cluster centers called centroid and at the meantime calculate the mean square error or sum of square error(SSE).For every different value of k we need to calculate the sum of squared error.
- Assign the each x_i to the nearest cluster by calculating its distance to each centroid.
- Finally new cluster by taking the avarage of the assigned points.
- Repeat the step 2 and 3 until none of cluster assignments change.

Relationship between the sum of squared error and k-mean value

- Let's consider that we having the graph of the k values against SSE (sum of squared error).
- When the value of k is increasing that time the SSE(sum of squared error) start Decreasing.After the some point the The decresing of the SSE is becoming the low and It's slop is become is very low and horizontal line having the 0 slop.This is konw as **elbow method.** Because this graph having the shape of elbow.
- This graph can able to help to get the perfect value of the k at this condition the value of the sse(sum of squared error) is become low and at the same points the k value is to be consider as perfect value of k.

For that we need to find the value of the SSE (sum of squared error).

- SSE = Sum Of Squared Error empty list has been created.
- Make the variable as index in that we had passed the range from the 1 to 10.Beacsue we want to iterate the k value from the 1 to 10.
- Create the kmeans variable in that we had put attributes that are the n_clusters and random state with the help of KMeans.
- Make the fit on the x to train it.
- Append the values of kmeans*ineria* in the empty list i.e.SSE
- **kmeans*ineria*** = It's helps to get the values of the sum squared errors inside the list of the kmeans from the from the range 1 to 10.
- Then the print the values of the kmeans.inertia_ .
- It will help us give it back the sum of square error.

- If we kindly obesrved that the sum of square errors are decresing from the 400 to 156 then to 44 but after the 44 it will not decresing if you observe the previous decreasing order.
- The decrement is going foreword not going the highly decrement.
- Decrement strat from the 400 to 156.
- Decrement done succefully upto 44 but after that the there so much diffrences in the declination SSE (sum of square error).

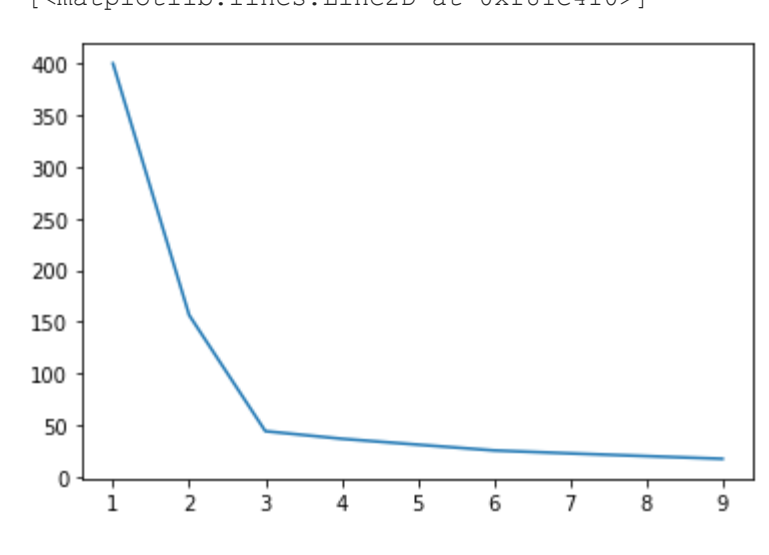
In [26]:
```
SSE = []
index = range(1,10)

for i in index:
    kmeans = KMeans(n_clusters=i,random_state=42)
    kmeans.fit(x)
    SSE.append(kmeans.inertia_)
    print(i,kmeans.inertia_)
```

```
1 400.0000000000001
2 156.41078579574986
3 44.05704845329278
4 36.72638711866608
5 31.016427615314644
6 25.39959047192452
7 22.547184727743293
8 19.923438178477447
9 17.295836404824975
```

- We have to plot the k vs SSE.
- If we can check it then we will get to know that the 3 is the k value over the declination SSE is lowest declination point over the SSE slop is being low and giving the less diffrences after the k=3.
- So,We have to consider the out appropriate value of k as 3.

In [30]:
```
plt.plot(index,SSE)
plt.show
```

Out[30]:
```
[&lt;matplotlib.lines.Line2D at 0xf81e410&gt;]
```



In [ ]: