**DBSCAN - an Easy Clustering Algorithm and also how to optimise it Using grid search**

DBSCAN stands for "Density-Based Spatial Clustering of Applications with Noise."

It is a powerful Non-supervised clustering algorithm that can be used to find clusters in a dataset

If you are not interested in the math behind the algorithm below given is an easy way of understanding how it works

Alright, imagine you have a bunch of colourful pebbles. You want to put the pebbles that are close to each other into groups, but you don't know how many groups there should be.

DBSCAN is like a friendly robot helper that helps you with this.

The robot starts by picking a pebble, and then it looks at the other pebbles around it. If there are many pebbles really close to the one it picked, the robot puts all of those pebbles in a special group.

Then it moves to the next pebble and does the same thing, making more groups if it finds more close pebbles.

The robot is smart and knows that some pebbles might be all by themselves, like on a big empty space. It doesn't put those pebbles in any group.

And it's also careful not to mix pebbles from different groups together.

So, DBSCAN is like a pebble-sorting robot that makes groups of pebbles that are really close to each other, and it doesn't make groups for pebbles that are far away.

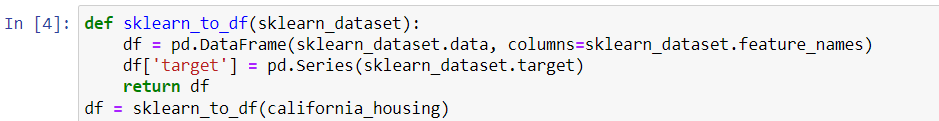
This helps you see which pebbles are hanging out together!

**DBSCAN** can be implemented using the sklearn library from python.This article will give you an overview of how This clustering algorithm can be implemented using python

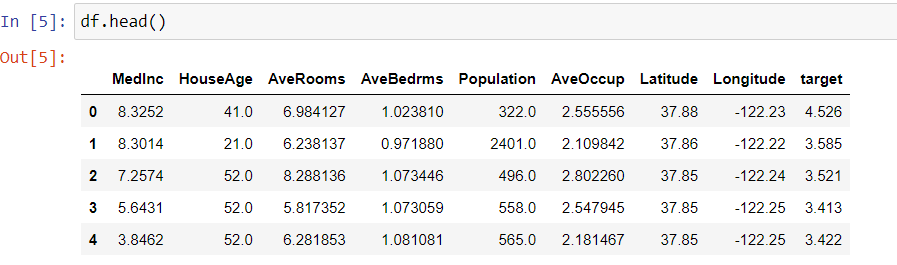
For easy implementation we are using a built in dataset from sklearn’s datasets collection called California Housing prices



Next we need to convert this sklearn dataset into a Pandas Dataframe.The below function does this for us

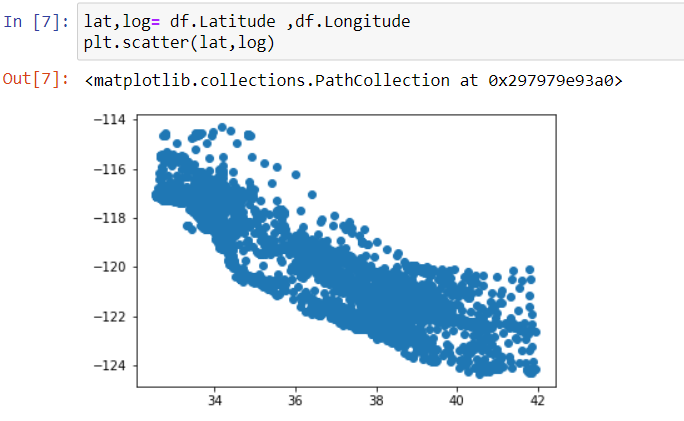


Now looking at the dataset



Although we can use many sets of features for our clustering for the demonstration here we will be limiting the features to Latitude and Longitude as more than 2 dimensional graph will be difficult for us to visualise although 3d plots are an option

Lets visualise the Latitude vs Longitude on a graph

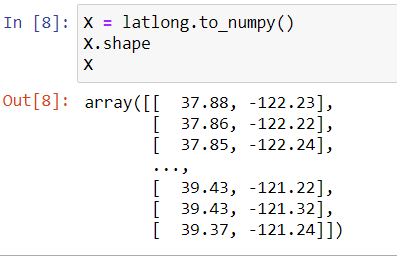


Now lets extract our required features from the dataframe

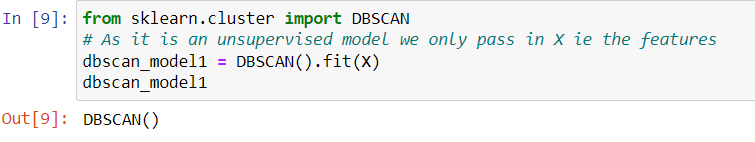


From the graph above we can see that both latitude and longitude falls within a scale and is not too far so we omit the scaling part here but if we are to use this with any other features whose values vary a lot then it is advised to scale these features using an appropriate scaler

Step 2 : now we need to convert the lat long data frame to a numpy array



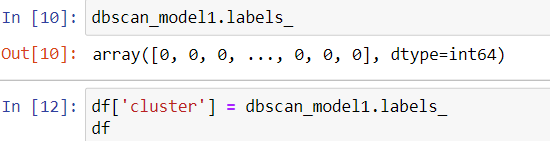
Step 3 : Create the model ,we import the DBSCAN class from sklearn.clusters module



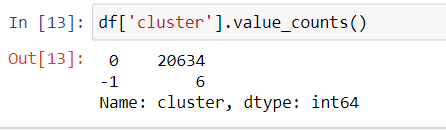
DBSCAN is a density based clustering algorithm so it tries to find the clusters where the density of the points is high

The next question will be what does this model return , The model will return a list containing labels for each set of points

Step 4: now we must use the labels returned by the model to plot and cluster the points

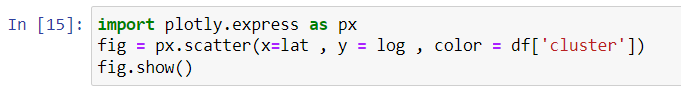


To find the number of clusters we use the value\_counts function



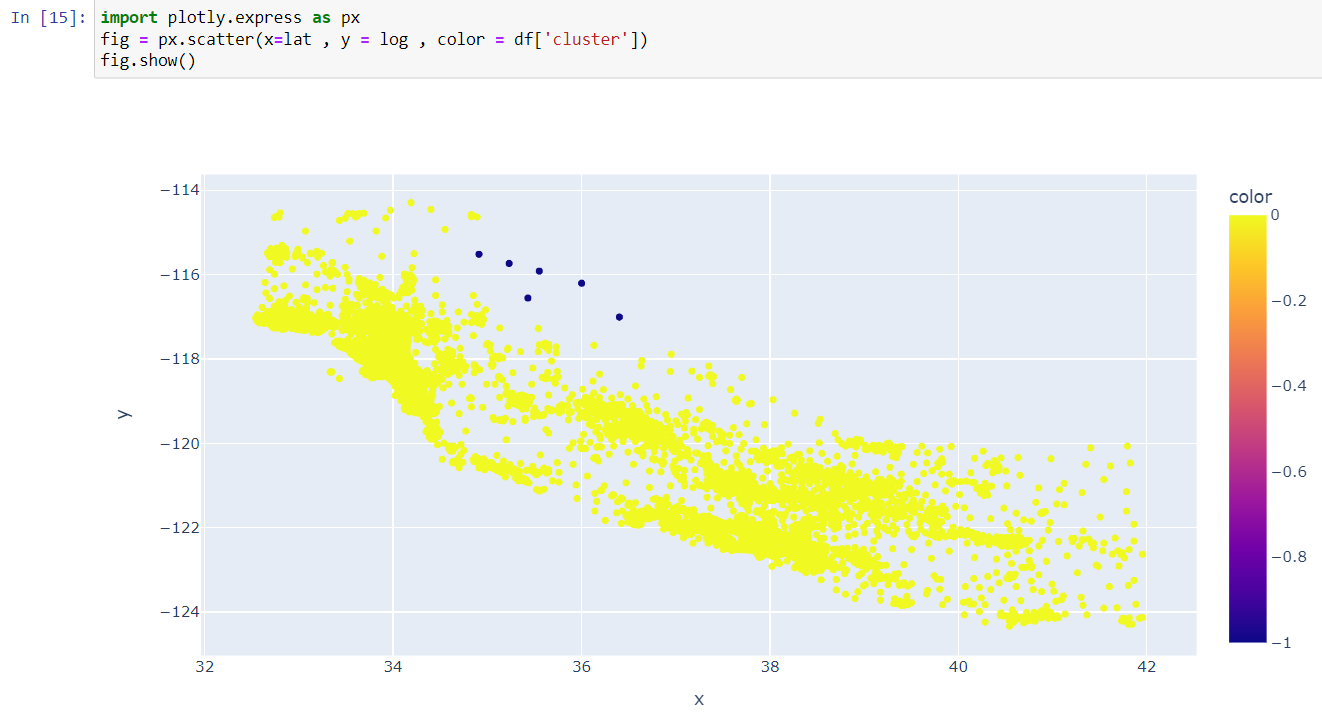
Here -1 shows the outliers or in other words number of points that doesn't belong to any category

To plot this we are using a plotly.express library as it is an easy way of plotting these clusters



If you are getting an error that plotly is not found the try

pip install plotly==5.15.0



Now we see that the model did not do anything special as it labelled the entire dataset as a single cluster

**Using Hyper parameters**

As mentioned earlier DBSCAN has a few hyperparameters which ensures the best results

Most of the time you will be tuning these 2 hyper parameters i.e

**Epsilon (ε)**: This is also called the "eps" parameter. When DBSCAN looks at a dot, it draws a circle around it with a radius of epsilon.

If there are enough dots inside that circle, they become a group. Choosing the right

epsilon depends on how spread out your dots are.

* **MinPoints**: This is the minimum number of dots that should be inside the circle for DBSCAN to consider them a group. If there are fewer dots than this number, the dot might be considered as noise or an outlier. The MinPoints value helps control how strict or lenient the algorithm is when forming groups.

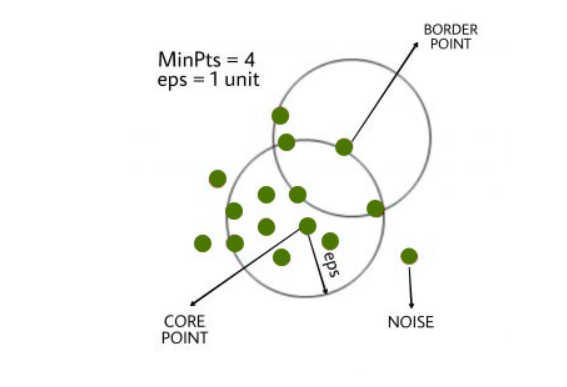
Here we got 3 types of data points.

**Core Point** : A point is a core point if it has more than MinPts points within eps.

**Border Point** : A point which has fewer than MinPoints within eps but it is in the

neighbourhood of a core point.

**Noise** : A point which is not a core point or border point.



These two hyperparameters, epsilon and MinPoints, help you fine-tune how DBSCAN works on your data. You might need to try different values to see what works best for your data

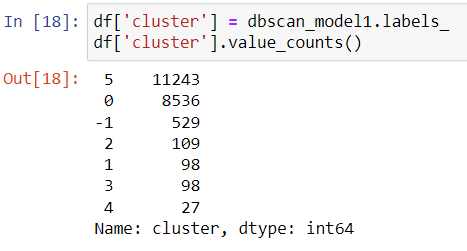
You can check the detailed documentation on dbscan and its parameters here :<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.htm>l

Now you have an idea of how these parameters affect the model and our results

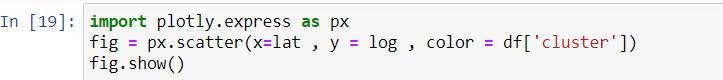
So lets train a new model that takes in these parameters and we can check if it will give us a good result

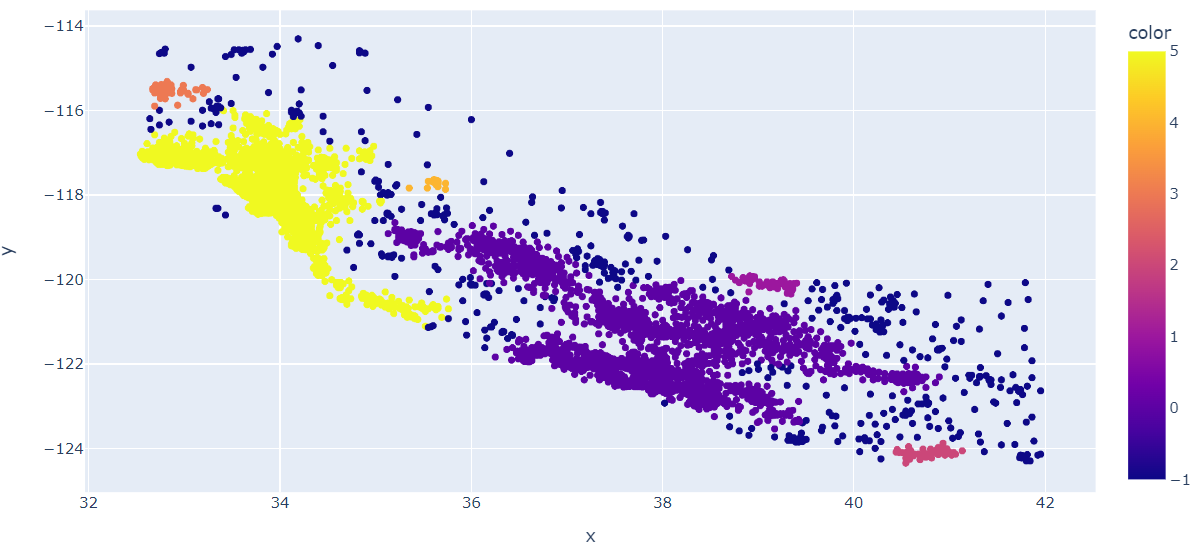


Here we gave epsilon as 0.2 and minimum number of samples as 20



As we can see here it did split these points into few clusters ,lets plot these and see the results





So here we just gave some values to the model as parameters and the changes were significant

So in our quest to find the best possible model we now have to define a way to compare 2 models so that we can say one is better than the other

There are a few methods by which we can compare different clustering models and one we are going to use here is the **Silhouette Score**

**Silhouette Score**

The silhouette score is a way to measure how good the clusters are when you use a clustering algorithm like DBSCAN. It helps you understand how well-separated the clusters are and how similar the objects within each cluster are.

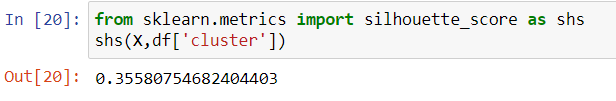
Imagine you have used DBSCAN to group some dots on a map. The silhouette score works like this:

* **Distance and Similarity**: For each dot, the silhouette score considers two things:
  + How close the dot is to other dots in its own cluster (a measure of cohesion).
  + How far the dot is from dots in other clusters (a measure of separation).
* **Silhouette Value**: The silhouette value ranges from -1 to 1:
  + If the silhouette value is close to 1, it means the dot is well inside its own cluster and far from other clusters. This is good!
  + If the silhouette value is close to 0, it means the dot is on the border between clusters.
  + If the silhouette value is negative, it means the dot might be in the wrong cluster.
  + Average Silhouette Score: To get the overall silhouette score for all the dots, you take the average of all the individual silhouette values.

Now let's dive into to evaluate these models and compare them based on this Silhouette Score

Step 5 : evaluate the model

Silhouette score can be fetched from sklearn.metrics library



Here we got a score of 0.35 that's not bad for this dataset but it can be improved further by changing the value of epsilon and min\_samples of the model

So the next question arise as how can we find the best values of epsilon and min\_samples

There are many methods via this can be done but here we are going to use the easiest one ie a simple variation of **Grid Search**

**Grid Search:**

Grid search is a technique used in machine learning to find the best combination of hyperparameters for a model.

Think of hyperparameters as settings that control how a machine learning algorithm works. For example, in the DBSCAN clustering algorithm, the hyperparameters are the "epsilon" (circle size) and "MinPoints" (minimum number of points in a cluster).

Grid search works like this:

* **Making a Grid**: Imagine you have a grid, like a table. Along the rows, you list different values for one hyperparameter (like different epsilon values), and along the columns, you list values for another hyperparameter (like different MinPoints values).
* **Trying All Combinations**: Grid search then tries out every possible combination of these values. It trains and evaluates your model with each combination.
* **Finding the Best**: After trying out all the combinations, it checks which combination gives the best performance. This could be measured using something like the silhouette score, accuracy, or any other relevant metric.
* **Picking the Winner** : The combination that gives the best performance is considered the winner. These are the hyperparameters you should use for your model

Here we are going to implement a version of Gridsearch where we are going to try all possible combinations of epsilon and min\_samples within a range ,This can be done by mainly 4 steps

1 .creating a list a numpy array of these parameters with desired range

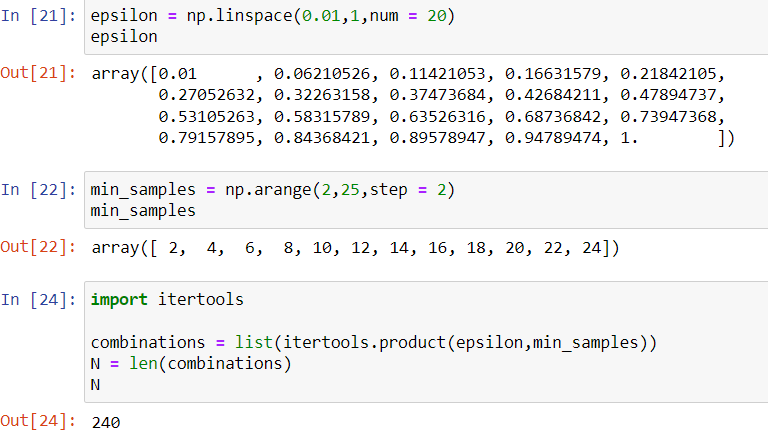
2. Converting these list to a combination list containing hyperparameters

3.passing each set of parameters and training the model

4.calculate the scores of each model and then using it find the best n - models

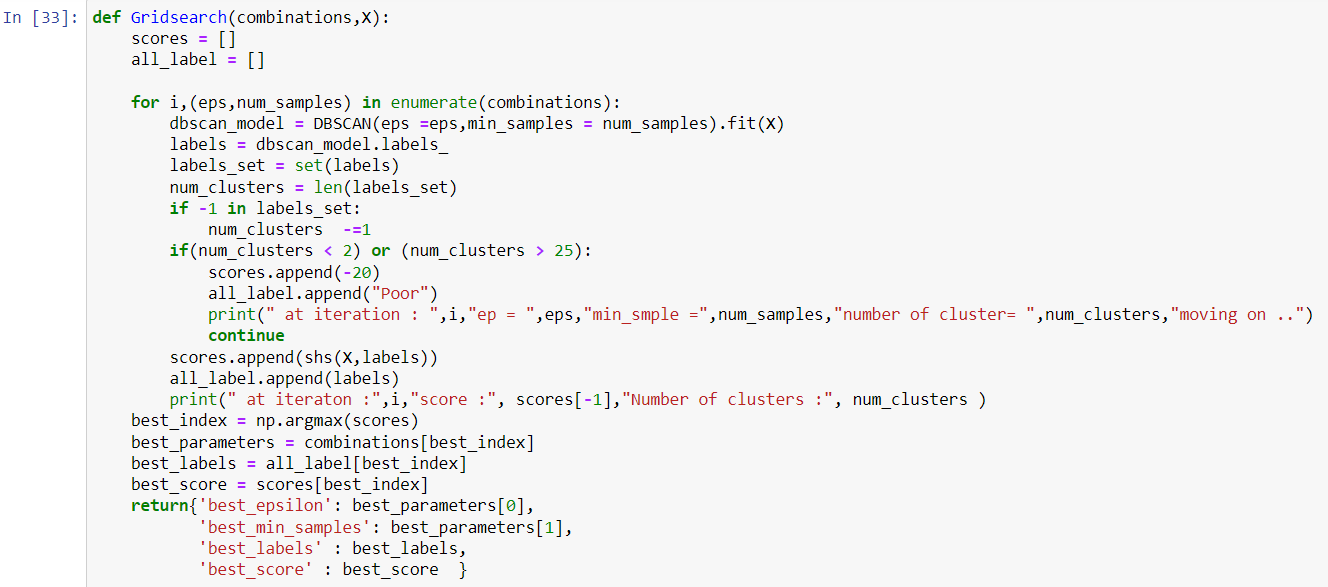
Step 6: finding the best set of hyper-parameters

As shown in the above steps we create 2 lists of both epsilon and min\_samples and use iter tools to generate a combination



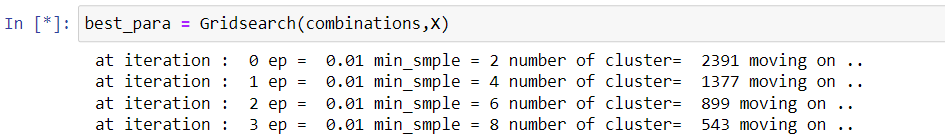
Here we have 240 such combinations to try and get a good result from them

Now to implement grid search we pass this combinations list and the function should give us the best set of parameters ,and the best model based on silhouette score

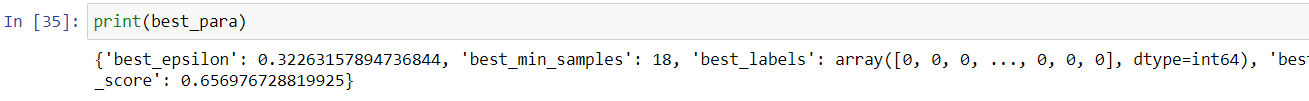


Here we have given a minimum and maximum limit for the number of clusters formed but mostly that upper cap is not needed but here for demonstration purposes we use it.

So we now need to get best parameters and labels by passing our combinations list and training data onto the function defined above



It takes few minutes to complete as it needs to compute the score for 240 combinations after completion best para will have a dictionary that contains the details of best model along with the score



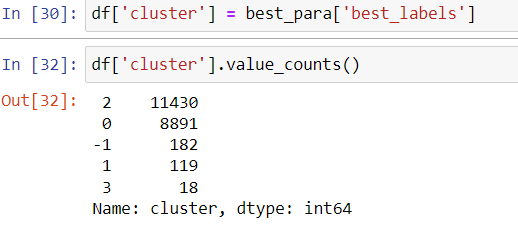
Now we got our best set of hyper parameters

Epsilon as 0.32

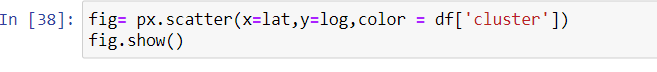
Min\_samples as 18

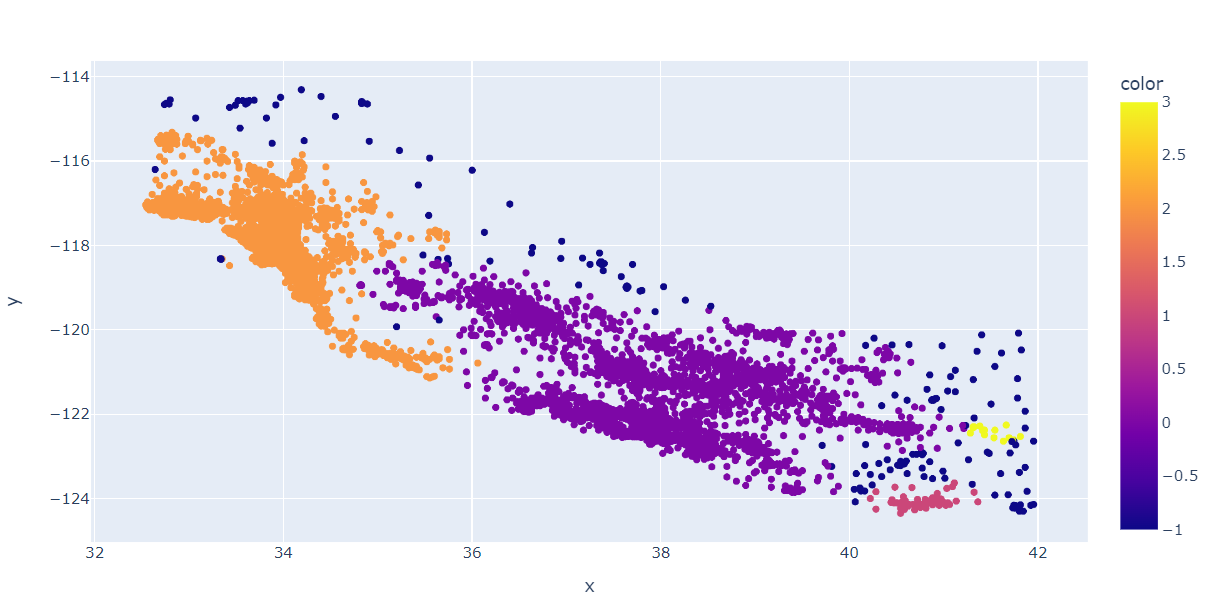
Score of 0.656

Also we got our labels.lets add these labels to our dataframe and this to see our improvement



Now we got 4 clusters ,lets plot them





This way we can use grid search for DBSCAN to get better results than random guessing

Here we have used only 2 features to form clusters but when there are multiple features we must trust the silhouette score

Also another way will be to reduce the components to its principle components and then use clustering