58. DBScan Clustering Algorithm (Practical)

In [33]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_moons
import pandas as pd
```

In [34]:
```
x, y = make_moons(n_samples=250, noise=0.05)
```

In [35]:
```
# First column data
x[:,0]
```

Out[35]:
```
array([ 1.82154686,  0.33039765,  1.11621412,  1.75338817,  0.79406254,
        0.09592268,  0.77490969, -0.96252073,  0.78806259,  0.23980237,
        1.89848596,  0.01310029,  0.07091097,  0.58314369, -0.96699757,
        1.04928892,  0.57445541,  0.74501548, -1.00650714, -0.82152061,
        0.86190156,  0.49126856,  1.8694458 ,  1.49802939,  0.6728559 ,
        0.98720469,  0.45047353, -0.2865276 ,  0.42354262,  0.88007291,
        0.36500296,  1.03385062,  0.3360013 , -0.01448323,  0.89163708,
       -0.49418874, -0.35425069,  0.50247661,  0.85778933, -0.64054093,
        0.04832991,  2.01015645,  1.2204363 ,  1.88956628, -0.13252252,
       -0.49523227,  0.33669497,  0.76153465, -0.27186532,  1.28870288,
        0.93076493,  1.90104103,  0.02543826,  0.44853317,  1.00448433,
       -0.25031522,  1.92847973,  0.42609288,  1.27725596, -0.87659926,
        0.61413887,  0.22505794, -0.04512895,  0.69131929,  0.259285  ,
        1.84036998,  0.47866329, -0.2844214 ,  1.31106194, -0.23293884,
        1.31976039,  1.93209062,  0.01521537,  1.80101873, -0.0391596 ,
       -0.49585375,  1.58238117,  0.01637799,  0.72630669,  0.91538441,
       -0.02865494,  1.45737014,  0.64884606, -0.74483793,  1.06418573,
        2.01778072, -0.37167832, -0.81173842, -0.91412945,  0.91606385,
        0.6136725 , -0.43538794, -0.7255368 ,  0.96102712,  1.91454596,
       -0.89253863,  0.12816908,  0.1788169 ,  0.12848506, -0.63151123,
       -0.86587253,  0.71871253,  0.04011572, -0.98749825,  1.50781231,
        0.87760821, -1.00423248,  1.44746249,  0.78287742, -0.44868368,
        1.89144999,  0.80058548,  1.88844507, -0.42427662, -1.02605411,
       -0.06589804,  1.92603512, -0.80463147,  1.51462527,  0.09590861,
        1.67803099, -0.70384095,  0.99436775, -0.10206081,  0.05106181,
        1.62449215, -0.90402427,  1.42745557,  1.92430944,  0.29091015,
        0.80306352,  1.4728536 ,  0.6063805 ,  1.07151034,  0.77765877,
       -0.79018459,  0.07756758, -0.6810462 ,  0.23387981, -0.9338624 ,
        0.4196469 ,  1.0743206 , -1.07958865,  0.66710905, -0.98994215,
        0.66449098,  0.49932715,  0.82120004,  0.48939705,  1.63825117,
        1.86809186, -0.91683607,  1.96477994, -0.48955672,  0.56543806,
        0.22563531, -0.93097574,  0.07804955,  0.68306482,  0.61259675,
        2.0154066 ,  1.63111271,  0.99459508,  0.613061  ,  0.34693487,
       -0.65125101, -1.08710097,  0.05320543, -0.97583562,  0.14590846,
        1.28499957, -0.01456049,  0.99183249,  1.95048888,  0.22331433,
        0.09418366,  0.84157974,  0.15503721, -0.73386657,  1.05716677,
        0.4758061 ,  1.20194107,  1.76139437,  0.99175489,  0.88443974,
        1.77304674,  0.72292497,  0.02948542,  0.19793304,  0.77849153,
        0.23332147, -0.07093789,  0.35773924, -0.80533347,  0.36514192,
       -0.00781175,  0.54951133,  1.74424235,  0.05659029, -0.85842631,
        1.69412089,  0.08545157,  0.30568356,  1.72318384, -0.04890115,
       -0.07264175, -0.63270247,  0.15343203, -0.69145099,  0.94662445,
        0.84613359,  0.22515933,  1.64017442,  0.95298684, -0.0749229 ,
        0.32118047, -0.56206221, -1.06808907,  1.20337727,  0.03416578,
       -1.00746399,  0.94031733,  1.99598102, -0.19158745,  0.28515341,
        1.30521283, -0.55735516,  1.36320489, -0.83877886,  1.08372752,
        0.45287718,  0.8876371 ,  1.03408887,  0.23404788,  1.89919484,
        0.80054814,  2.0020812 ,  1.0442318 ,  1.86594006,  1.72375506,
       -0.3131739 ,  0.13166397,  1.04628395,  1.88899169,  1.98677442,
        2.03239475,  0.43570842,  1.04879911,  1.04383872, -0.80846152])
```

In [36]:
```
df = {"data1":x[:,0], "data2":x[:,1], "output":y}
```

In [37]:
```
dataset = pd.DataFrame(df)
```

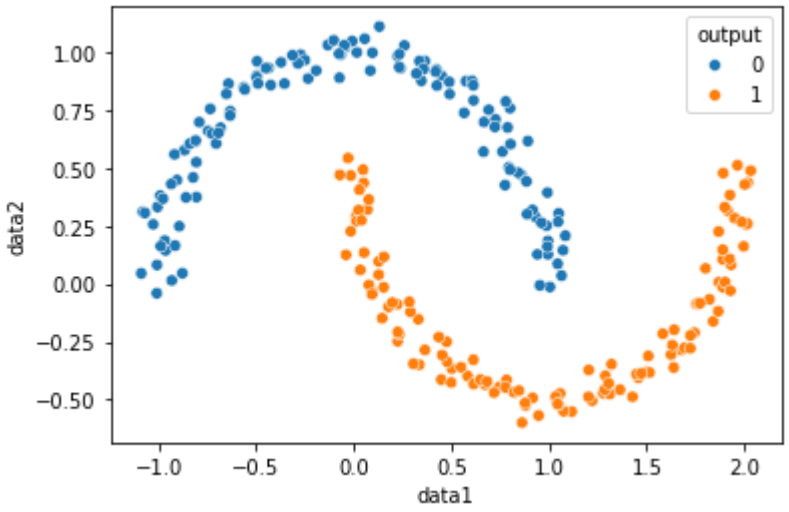
In [38]:
```
dataset
```

Out[38]:

	data1	data2	output
0	1.821547	-0.067198	1
1	0.330398	-0.152611	1
2	1.116214	-0.551311	1
3	1.753388	-0.086491	1
4	0.794063	0.502981	0
...	...	...	...
245	2.032395	0.488380	1
246	0.435708	-0.229602	1
247	1.048799	-0.520698	1
248	1.043839	-0.518346	1
249	-0.808462	0.618646	0

250 rows × 3 columns

In [39]:
```
sns.scatterplot(x='data1', y='data2', data=dataset, hue='output')
plt.show()
```



- The data is **non-linear**, so we will apply DBSCAN Clustering algorithm

In [40]:
```
dataset.head(3)
```

Out[40]:

	data1	data2	output
0	1.821547	-0.067198	1
1	0.330398	-0.152611	1
2	1.116214	-0.551311	1

- We cannot apply DBSCAN on 0 and 1 as this is present in output column, so will remove this column before applying this algo.

In [41]:
```
dataset.drop('output', axis=1, inplace=True)
```

In [42]:
```
dataset
```

Out[42]:

	data1	data2
0	1.821547	-0.067198
1	0.330398	-0.152611
2	1.116214	-0.551311
3	1.753388	-0.086491
4	0.794063	0.502981
...	...	...
245	2.032395	0.488380
246	0.435708	-0.229602
247	1.048799	-0.520698
248	1.043839	-0.518346
249	-0.808462	0.618646

250 rows × 2 columns

In [43]:
```
from sklearn.cluster import DBSCAN
```

In [44]:
```
db = DBSCAN(eps=0.2, min_samples=5)
db.fit_predict(dataset)
```

Out[44]:
```
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1], dtype=int64)
```

In [45]:
```
dataset['Predict'] = db.fit_predict(dataset)
```

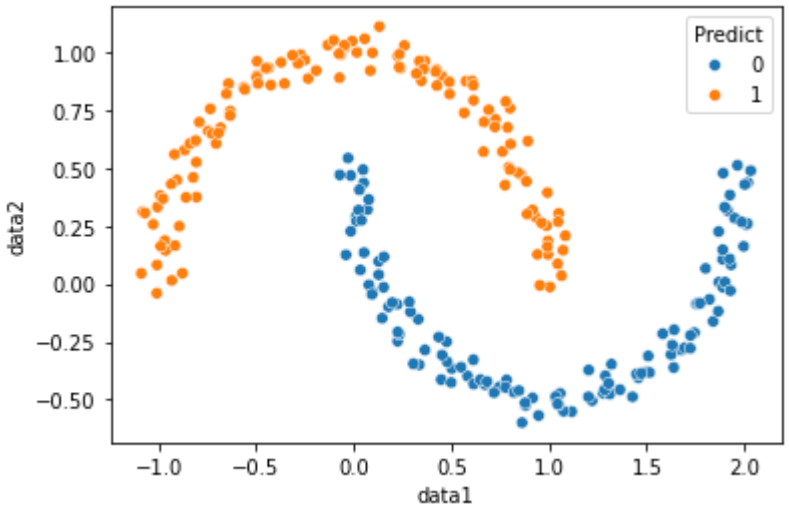
In [46]:
```
dataset
```

Out[46]:

	data1	data2	Predict
0	1.821547	-0.067198	0
1	0.330398	-0.152611	0
2	1.116214	-0.551311	0
3	1.753388	-0.086491	0
4	0.794063	0.502981	1
...	...	...	...
245	2.032395	0.488380	0
246	0.435708	-0.229602	0
247	1.048799	-0.520698	0
248	1.043839	-0.518346	0
249	-0.808462	0.618646	1

250 rows × 3 columns

In [47]:
```
sns.scatterplot(x='data1', y='data2', data=dataset, hue='Predict')
plt.show()
```



So predicted data resembles with actual output as shown in the graphs of predict and original data

In [ ]: