Assignment2-solution-DBSCAN-Clustering
Group Members Name: Suresh Kumar Choudhary, Sofya Laskina, Emilio Kuhlmann
Master Data Science

November 22, 2020

[120]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

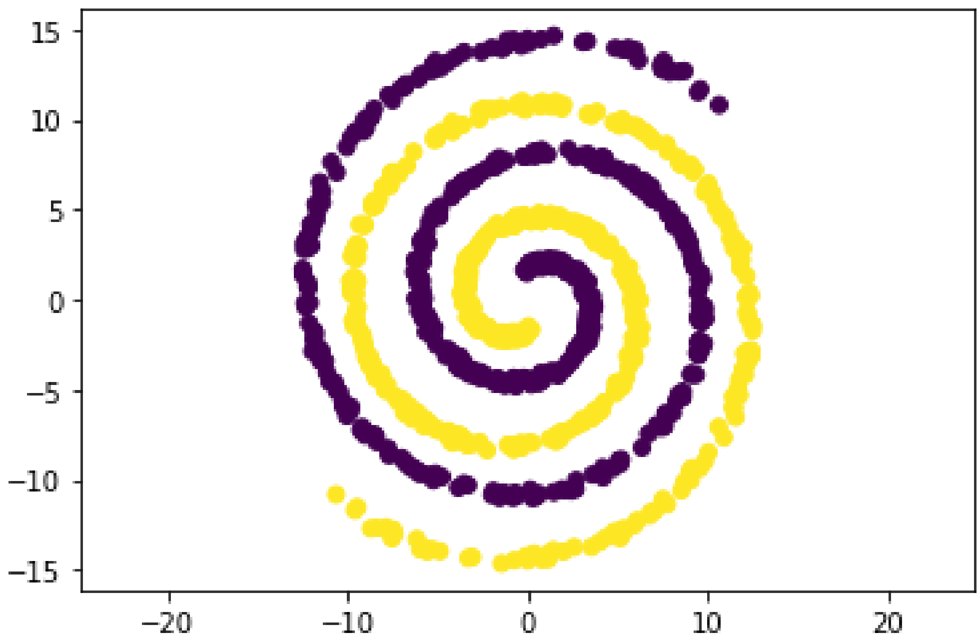
First of all: we are going to introduce a dataset on that we apply our clustering method on:

[121]:
```
def twospirals(n_points, noise=.5):
    """
     Returns the two spirals dataset.
    """
    epsilon = 0.1
    n = (np.random.rand(n_points,1)+epsilon) * 780 * (2*np.pi)/360
    d1x = -np.cos(n)*n + np.random.rand(n_points,1) * noise
    d1y = np.sin(n)*n + np.random.rand(n_points,1) * noise

    # hstack/vstack stacks data on top of each other (print shape to see what I␣
 ↪mean)
    C_1 = np.hstack((d1x,d1y))
    C_2 = np.hstack((-d1x,-d1y))
    return np.vstack((C_1, C_2))
```

This is a dataset consisting of clusters twisting around each other. You don't need to understand the mathematics behind it, but you can play around with it if you like (make sure to train on the original dataset, not one you created)

[122]:
```
def dataset_creation(data_size=500):
    dataset = twospirals(data_size)
    labels = np.hstack((np.zeros(data_size),np.ones(data_size)))
    return (dataset,labels)
# scatter makes a 2D scatter plot. Unfortunately you have to seperate the x-dim␣
 ↪from the y-dim
```

```python
# the labels are helpful for coloring. The algorithm does not use them, since this is unsupervised
np.random.seed(10)
dataset,labels_true=dataset_creation(data_size=500)
plt.scatter(dataset[:,0], dataset[:,1], c = labels_true)
plt.axis('equal')
plt.show()
```



a) Implement the DBSCAN algorithm to classify points of the two clusters.

b) Plot a scatter plot highlighting the clusters that were found after finding good hyperparameter values eps and minPts.

c) Print accuracies for different data_size values.

d) For what kind of data_size values does the algorithm fail and why? What would you say are disadvantages of DBSCAN?

[123]:
```python
import sys

# the setrecursionlimit function is
# used to modify the default recursion
# limit set by python. Using this,
# we can increase the recursion limit
# to satisfy our needs

sys.setrecursionlimit(10**6)
```

```
def euclidean_distance(x_1, x_2):
    #print(x_1,x_2)
    return np.sqrt(np.sum((x_1-x_2)**2, axis = 1))
```

[124]:

```
def lookup_table(dataset,eps):
    lookup_2d_table={}
    for pos, val in enumerate(dataset):
        dist=euclidean_distance(val, dataset)
        lookup_2d_table[pos]= np.argwhere((dist<=eps)==True).flatten()
    return lookup_2d_table
```

[125]:

```
'Noise=None, undefined/unvisited=-1'

def dbscan(data,Eps=2,minPts=1.7):
    n_points = len(data)
    #print(n_points)
    label = [-1] * n_points
    C=-1 #cluster initialization
    S=set()
    rangeQuery=lookup_table(data,Eps)
    for x, val in enumerate(data):
        if label[x]!=-1: #we only want unvisited points
            #print('unclassifief:',x)
            continue
        N=rangeQuery[x]
        if len(N) < minPts: #x is border or noise point
            label[x]=None #noise point
            continue #check next point in Dataset X
        C = C + 1
        label[x]=C #x gets new cluster id
        S.update(N)
        S.discard(x)
        while len(S)>0:
            y=S.pop()
            if label[y] == None:
                label[y] = label[x] # y is border point
            if label[y] !=-1: # y has label, get new y
                continue
            label[y] = label[x] #y belongs to x‘s cluster
            N = rangeQuery[y]
            if len(N) >= minPts:# y is core point of same cluster
                S.update(N)
```

```
    return label
```

a) Use Mathplotlib to create a scatter plot highlighting the clusters that were found after finding good hyperparameter values eps and minPts.

[126]:
```
def cluster_plot(dataset,labesl,Eps,minPts,data_size,acc,sil_coef):
    # scatter makes a 2D scatter plot. Unfortunately you have to seperate the␣
↪x-dim from the y-dim
    # the labels are helpful for coloring. The algorithm does not use them,␣
↪since this is unsupervised
    plt.scatter(dataset[:,0], dataset[:,1], c = labels)
    plt.axis('equal')
    plt.title(f'Eps={Eps},minPts={minPts},data_size={data_size},Accuracy={acc}%␣
↪and silhouette_score={sil_coef}')
    plt.show()
```

[127]:
```
#Eps and minPts hyperparameter tuning
Eps_list=[x for x in np.arange(0.1,2,0.05)]
minPts_list=[x for x in np.arange(2,10,1)]
acc_list=[]
sil_coef=[]
tup_list=[]
for minPts in minPts_list:
    for Eps in Eps_list:
        np.random.seed(10)
        dataset,labels_true=dataset_creation(data_size=500)
        labels=dbscan(dataset,Eps,minPts)
        #cluster_plot(dataset,labels)
        correct_classified=len(np.argwhere((labels_true==labels)==True))
        acc=(correct_classified*100)/len(dataset)
        acc_list.append(acc)
        #print('acc with funct',accuracy_score(labels_true,labels))
        n_clusters = len(set(labels)) - (1 if None in labels else 0)

        #print(f'Accuracy with data size {data_size} is {acc}%')

        labels=[x if x !=None else -1 for x in labels]
        if n_clusters>1:
            #print("Silhouette Coefficient: %0.3f"% metrics.
↪silhouette_score(dataset, labels))
            sil_coef.append(metrics.silhouette_score(dataset, labels))
        else:
            sil_coef.append(-1)
        tup_list.append([Eps,minPts,acc,sil_coef[-1]])
```

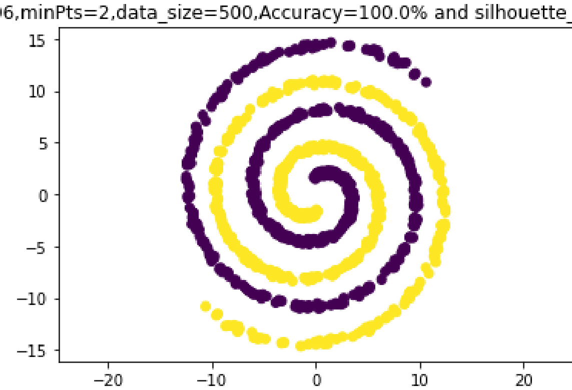
[128]:

```
best_Eps,best_minPts,best_acc,best_s_c=sorted(tup_list,key=lambda x:␣
↪(x[2],x[3]),reverse=True)[0]
```

[132]:

```
#best hyperparameters DBSCAN clustering
np.random.seed(10)
dataset,labels_true=dataset_creation(data_size=500)
labels=dbscan(dataset,best_Eps,best_minPts)
cluster_plot(dataset,labels,Eps=best_Eps,minPts=best_minPts,data_size=500,acc=best_acc,sil_coef=
```

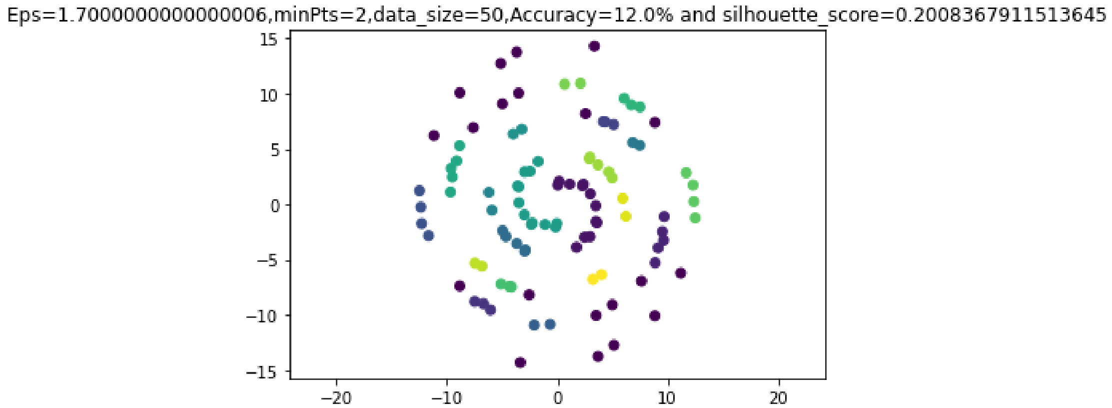
Eps=1.7000000000000006,minPts=2,data_size=500,Accuracy=100.0% and silhouette_score=0.012244321540891445

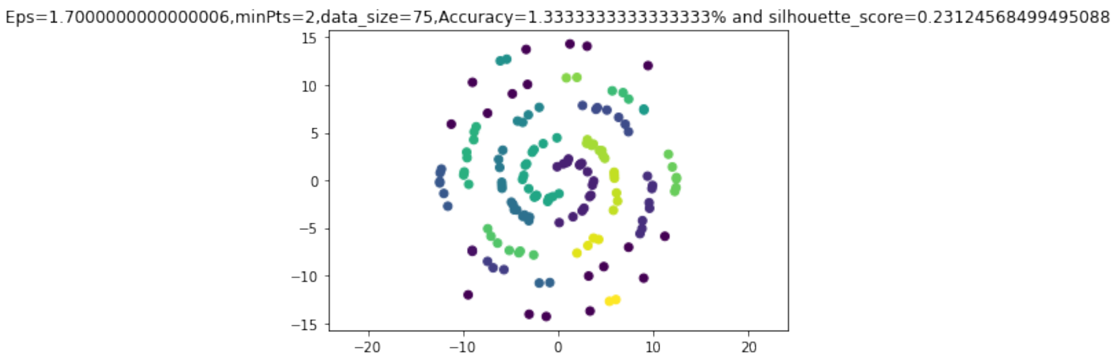


[130]:

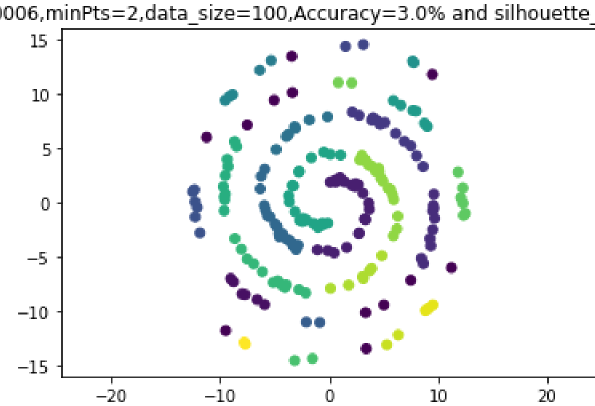
```
data_size_list=[x for x in range(50,2000,25)]
acc_list=[]
sil_coef=[]
for data_size in data_size_list:
    np.random.seed(10)
    dataset,labels_true=dataset_creation(data_size)
    labels=dbscan(dataset,Eps=best_Eps,minPts=best_minPts)
    #cluster_plot(dataset,labels)
    correct_classified=len(np.argwhere((labels_true==labels)==True))
    acc=(correct_classified*100)/len(dataset)
    acc_list.append(acc)
    #print('acc with funct',accuracy_score(labels_true,labels))
    n_clusters = len(set(labels)) - (1 if None in labels else 0)

    #print(f'Accuracy with data size {data_size} is {acc}%')

    labels=[x if x !=None else -1 for x in labels]
    if 1 < n_clusters:
        #print("Silhouette Coefficient: %0.3f"% metrics.
↪silhouette_score(dataset, labels))
        sil_coef.append(metrics.silhouette_score(dataset, labels))
    else:
        sil_coef.append(-1)
```

```
    ␣
  ↪cluster_plot(dataset,labels,Eps=best_Eps,minPts=best_minPts,data_size=data_size,acc=acc,sil_c

plt.plot(data_size_list, acc_list)
plt.title('Accuracy vs Data Size')
plt.xlabel('Data Size')
plt.ylabel('Accuracy')
plt.show()

plt.plot(data_size_list,sil_coef)
plt.title(' silhouette_score vs Data Size')
plt.xlabel('Data Size')
plt.ylabel('silhouette_score')

plt.show()
```

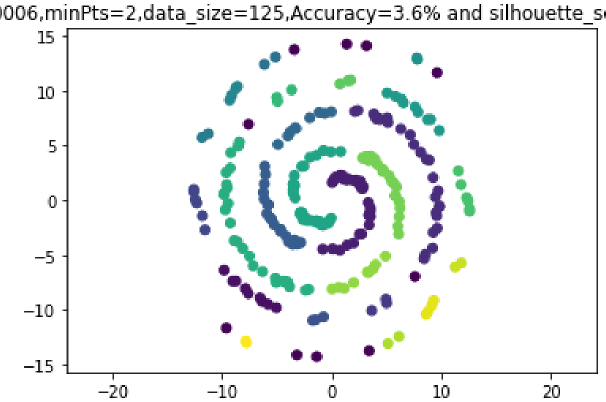




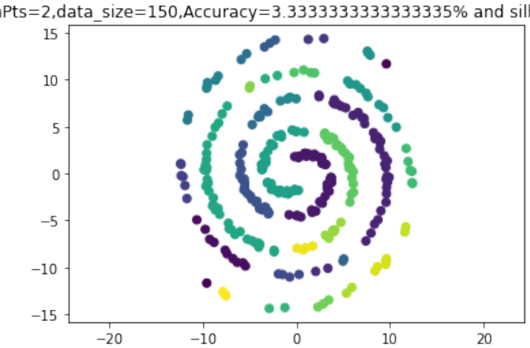
Eps=1.7000000000000006,minPts=2,data_size=100,Accuracy=3.0% and silhouette_score=0.2637407220660134



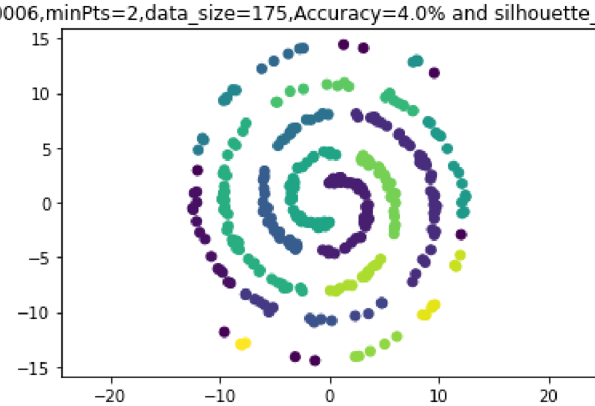
Eps=1.7000000000000006,minPts=2,data_size=125,Accuracy=3.6% and silhouette_score=0.17398187024503972



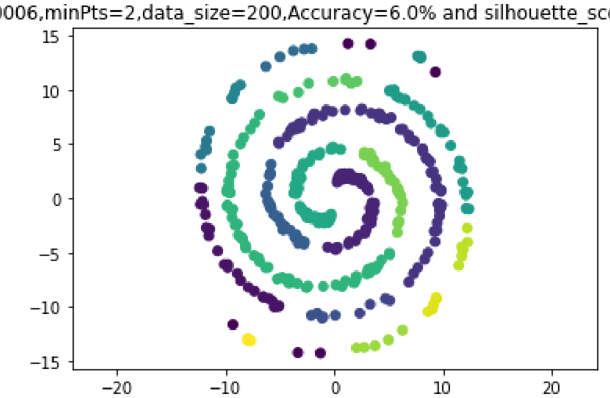
Eps=1.7000000000000006,minPts=2,data_size=150,Accuracy=3.3333333333333335% and silhouette_score=0.19529285577666836



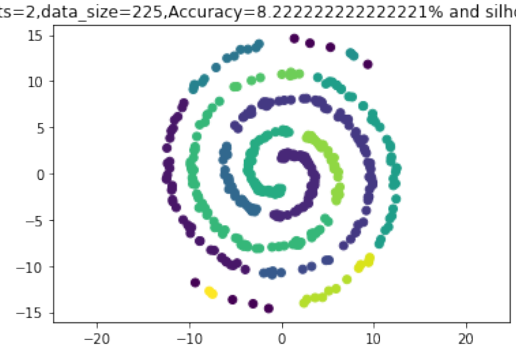
Eps=1.7000000000000006,minPts=2,data_size=175,Accuracy=4.0% and silhouette_score=0.1569257851211373



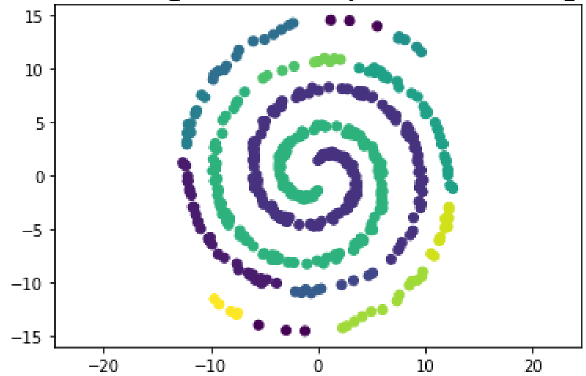
Eps=1.7000000000000006,minPts=2,data_size=200,Accuracy=6.0% and silhouette_score=0.050542646783161393



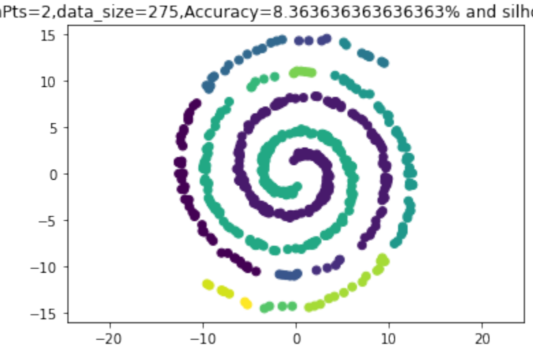
Eps=1.7000000000000006,minPts=2,data_size=225,Accuracy=8.222222222222221% and silhouette_score=0.013939699590679255



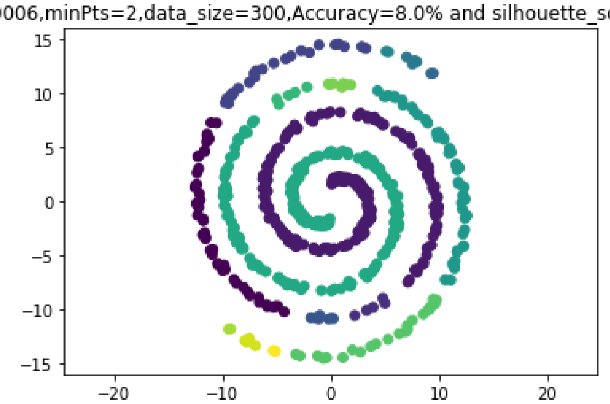
Eps=1.7000000000000006,minPts=2,data_size=250,Accuracy=6.2% and silhouette_score=-0.09527084678212139



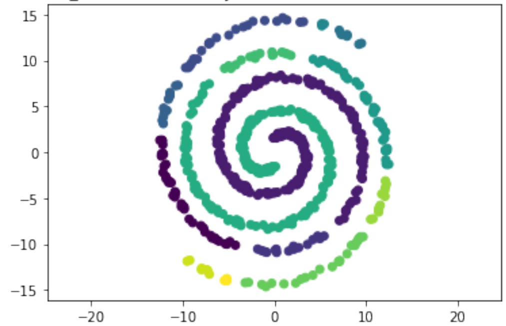
Eps=1.7000000000000006,minPts=2,data_size=275,Accuracy=8.363636363636363% and silhouette_score=-0.06340062718611442



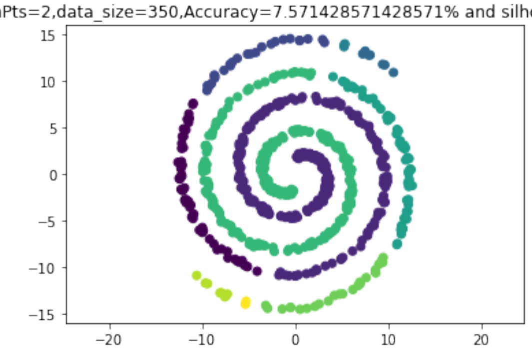
Eps=1.7000000000000006,minPts=2,data_size=300,Accuracy=8.0% and silhouette_score=-0.0941172781275175



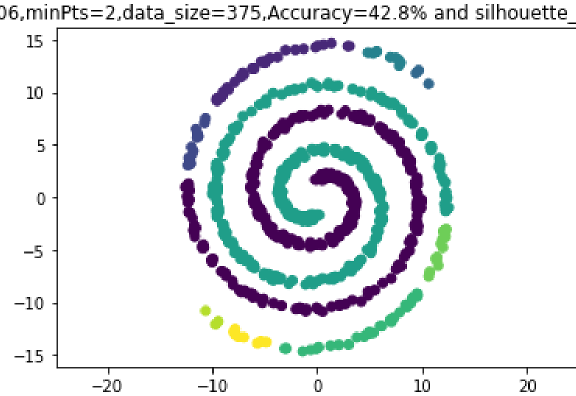
Eps=1.7000000000000006,minPts=2,data_size=325,Accuracy=6.153846153846154% and silhouette_score=-0.061474746292681064



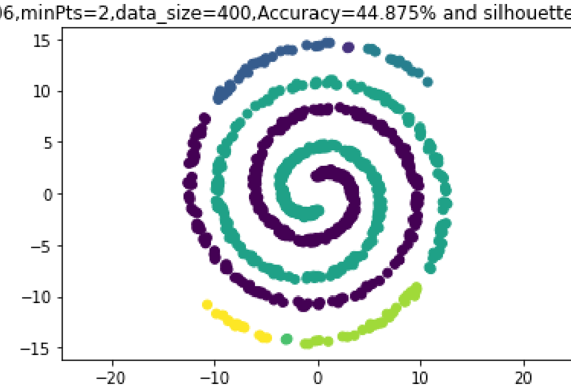
Eps=1.7000000000000006,minPts=2,data_size=350,Accuracy=7.571428571428571% and silhouette_score=-0.05042942816136573



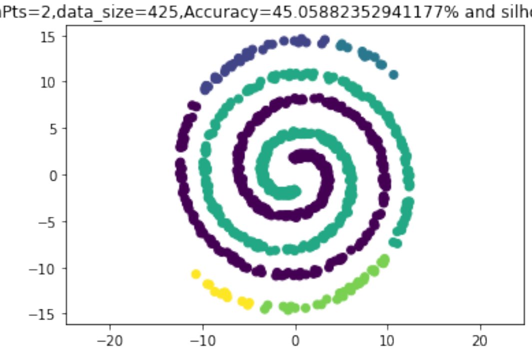
Eps=1.7000000000000006,minPts=2,data_size=375,Accuracy=42.8% and silhouette_score=-0.14946110385434855



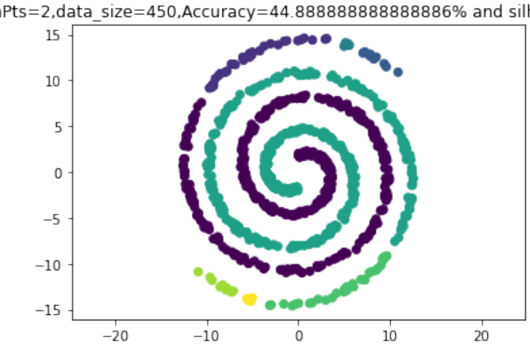
Eps=1.7000000000000006,minPts=2,data_size=400,Accuracy=44.875% and silhouette_score=-0.10632549835640237



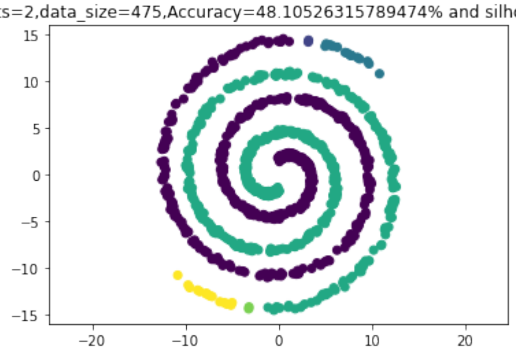
Eps=1.7000000000000006,minPts=2,data_size=425,Accuracy=45.05882352941177% and silhouette_score=-0.08555220047180072



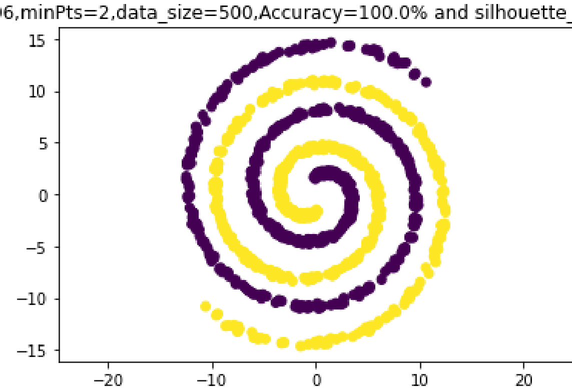
Eps=1.7000000000000006,minPts=2,data_size=450,Accuracy=44.888888888888886% and silhouette_score=-0.10747471370324556



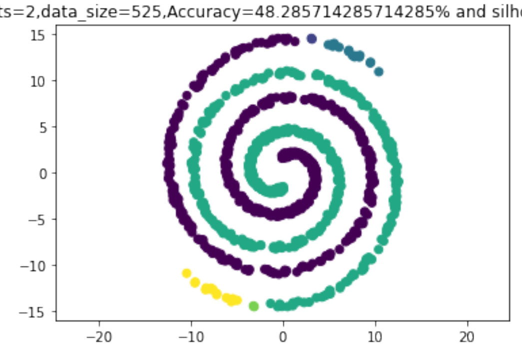
Eps=1.7000000000000006,minPts=2,data_size=475,Accuracy=48.10526315789474% and silhouette_score=-0.12578714775942426



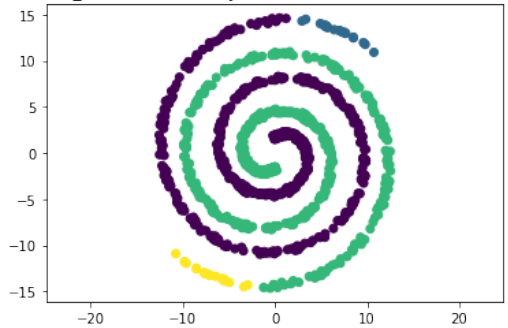
Eps=1.7000000000000006,minPts=2,data_size=500,Accuracy=100.0% and silhouette_score=0.012244321540891445



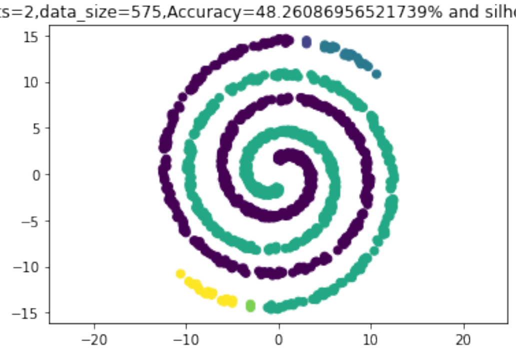
Eps=1.7000000000000006,minPts=2,data_size=525,Accuracy=48.285714285714285% and silhouette_score=-0.12091811370874772



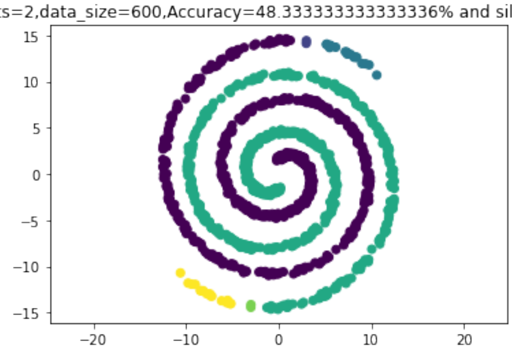
Eps=1.7000000000000006,minPts=2,data_size=550,Accuracy=48.27272727272727% and silhouette_score=-0.07072518713861597



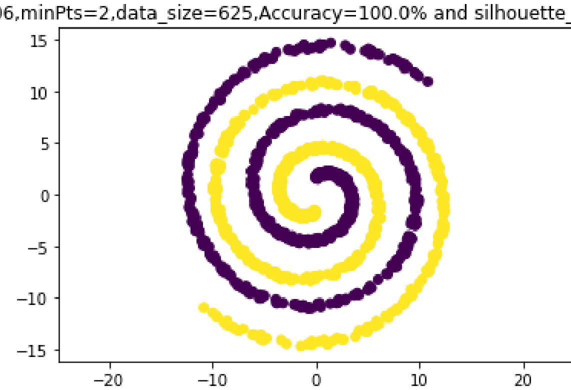
Eps=1.7000000000000006,minPts=2,data_size=575,Accuracy=48.26086956521739% and silhouette_score=-0.12502027176835023



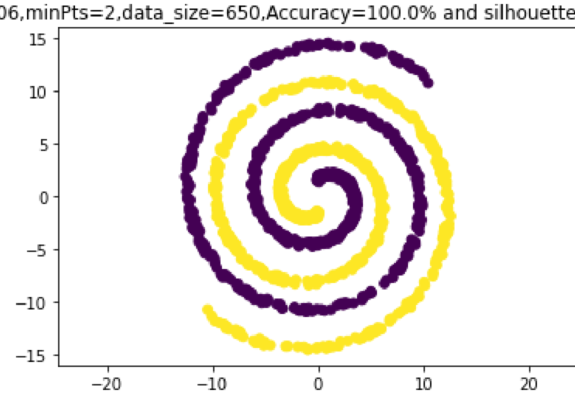
Eps=1.7000000000000006,minPts=2,data_size=600,Accuracy=48.333333333333336% and silhouette_score=-0.1257434784097256



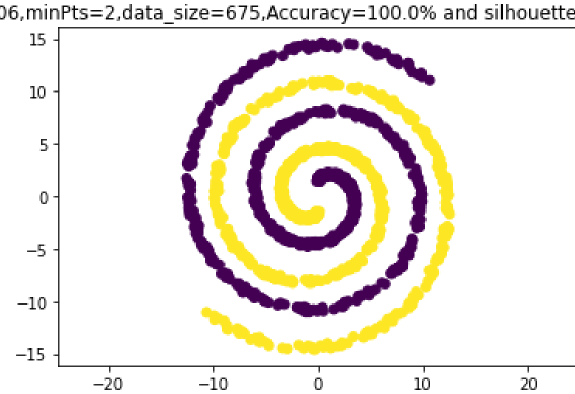
Eps=1.7000000000000006,minPts=2,data_size=625,Accuracy=100.0% and silhouette_score=0.011312990192626445



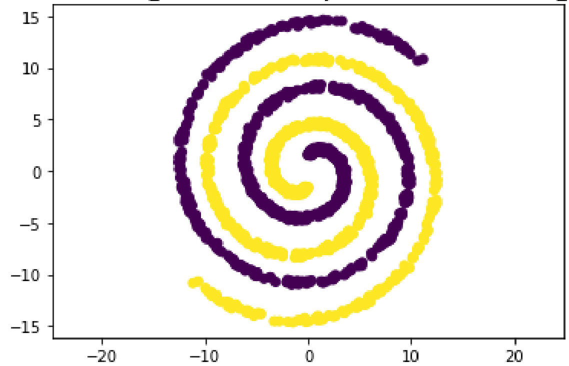
Eps=1.7000000000000006,minPts=2,data_size=650,Accuracy=100.0% and silhouette_score=0.01117413594717043



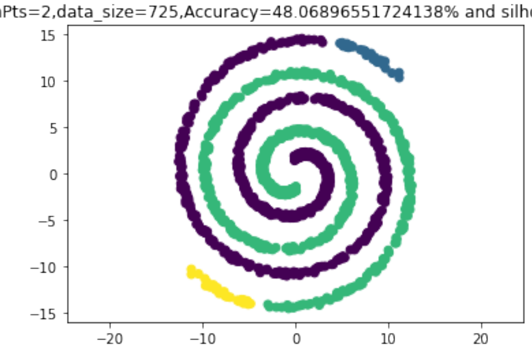
Eps=1.7000000000000006,minPts=2,data_size=675,Accuracy=100.0% and silhouette_score=0.01113453475896022



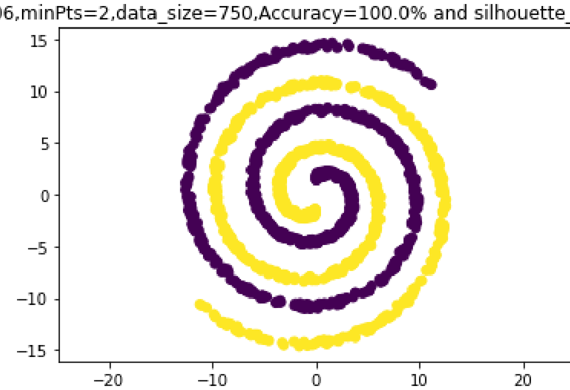
Eps=1.7000000000000006,minPts=2,data_size=700,Accuracy=100.0% and silhouette_score=0.011817597382017878



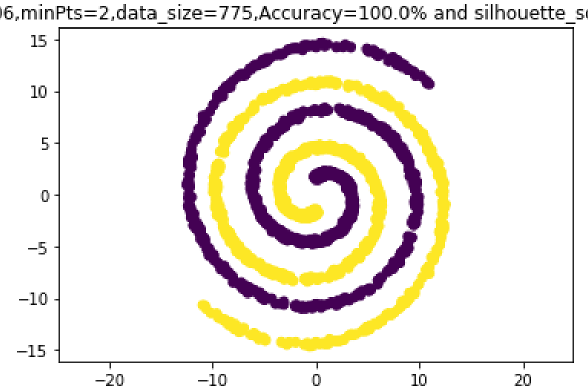
Eps=1.7000000000000006,minPts=2,data_size=725,Accuracy=48.06896551724138% and silhouette_score=-0.06655012877321667



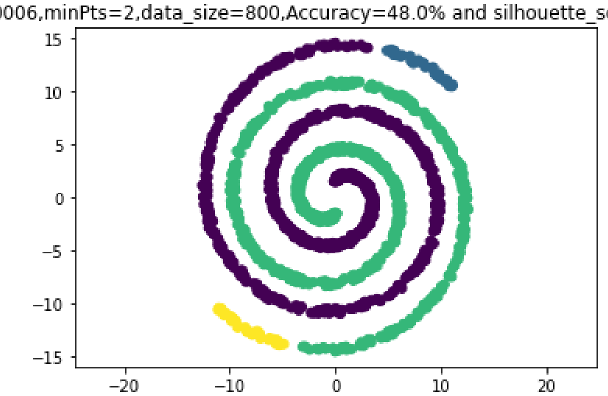
Eps=1.7000000000000006,minPts=2,data_size=750,Accuracy=100.0% and silhouette_score=0.009764399037481325



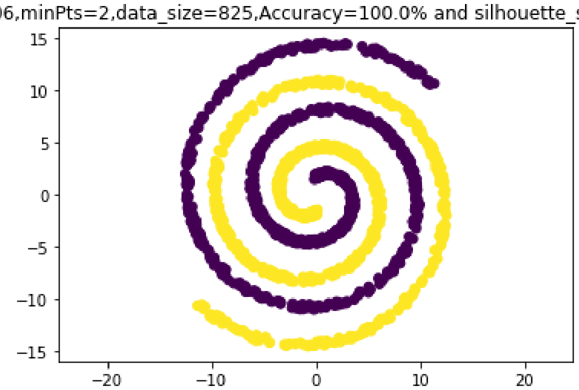
Eps=1.7000000000000006,minPts=2,data_size=775,Accuracy=100.0% and silhouette_score=0.009513268889659659



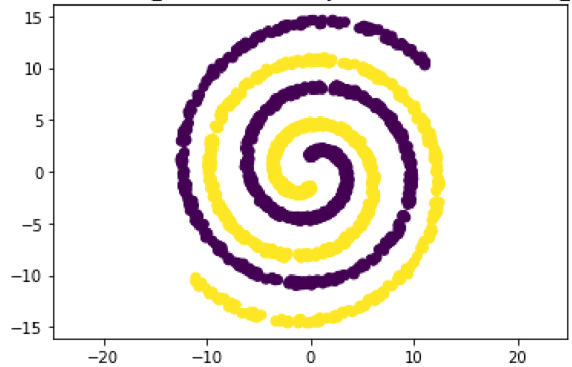
Eps=1.7000000000000006,minPts=2,data_size=800,Accuracy=48.0% and silhouette_score=-0.07004732903107024



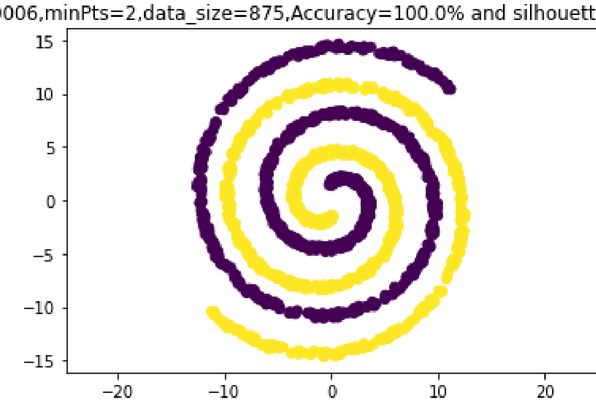
Eps=1.7000000000000006,minPts=2,data_size=825,Accuracy=100.0% and silhouette_score=0.010016288009463576



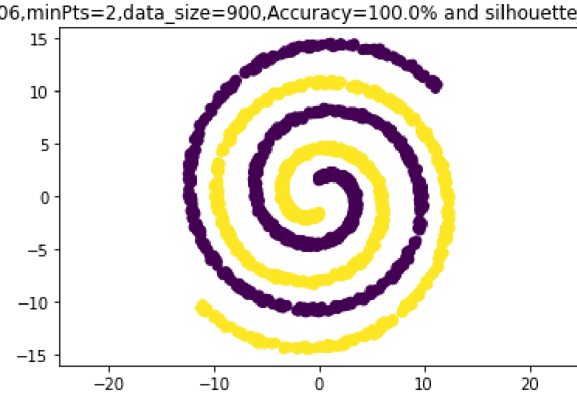
Eps=1.7000000000000006,minPts=2,data_size=850,Accuracy=100.0% and silhouette_score=0.010125750329298275



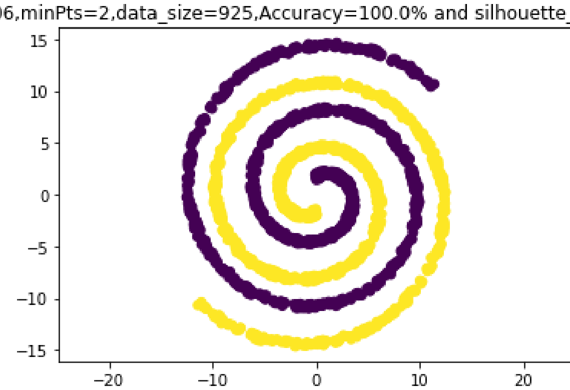
Eps=1.7000000000000006,minPts=2,data_size=875,Accuracy=100.0% and silhouette_score=0.011144945349692



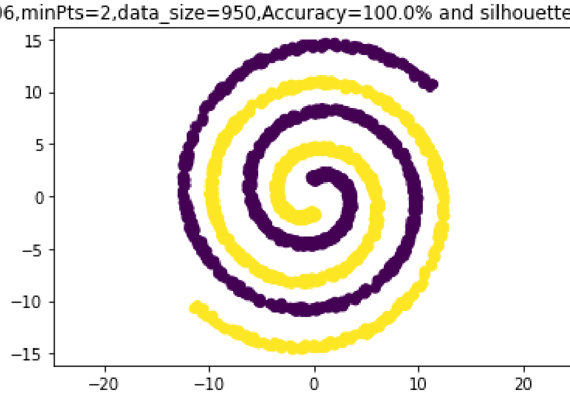
Eps=1.7000000000000006,minPts=2,data_size=900,Accuracy=100.0% and silhouette_score=0.01067301722385056



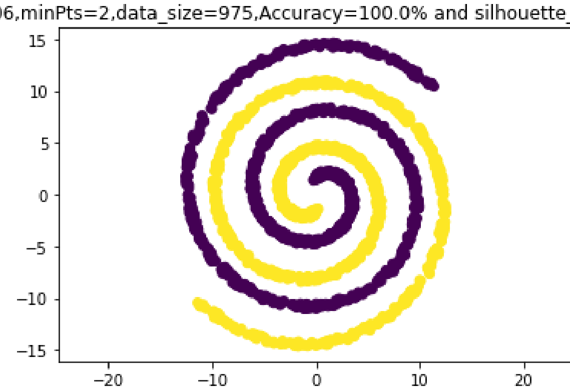
Eps=1.7000000000000006,minPts=2,data_size=925,Accuracy=100.0% and silhouette_score=0.011144815319109347



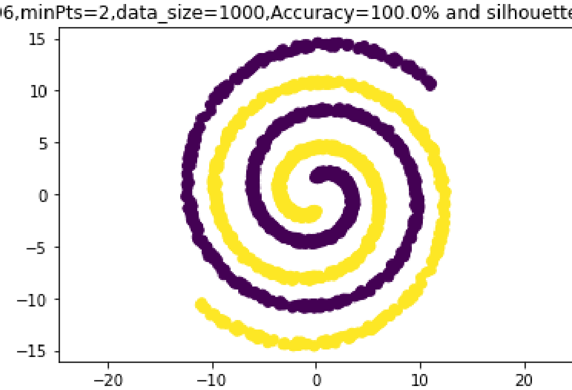
Eps=1.7000000000000006,minPts=2,data_size=950,Accuracy=100.0% and silhouette_score=0.01057131145170482



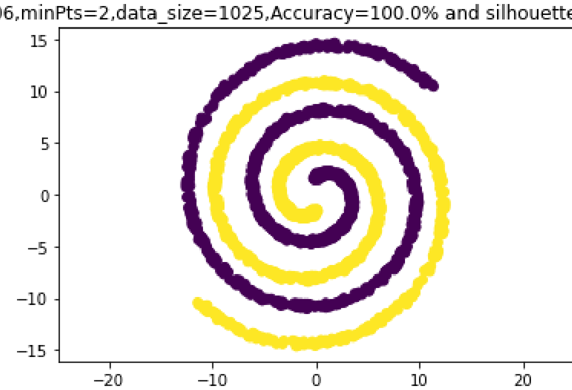
Eps=1.7000000000000006,minPts=2,data_size=975,Accuracy=100.0% and silhouette_score=0.010706706355984042



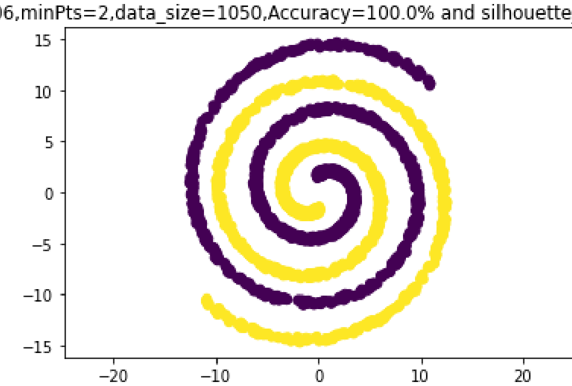
Eps=1.7000000000000006,minPts=2,data_size=1000,Accuracy=100.0% and silhouette_score=0.01040579478394302



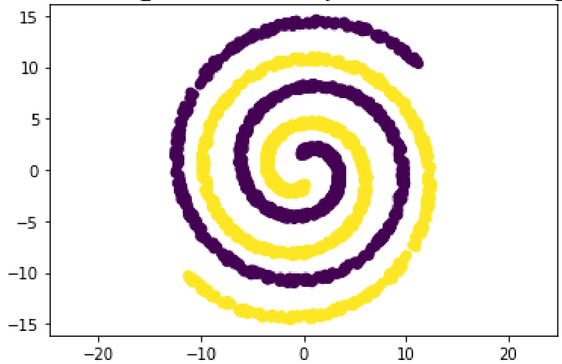
Eps=1.7000000000000006,minPts=2,data_size=1025,Accuracy=100.0% and silhouette_score=0.01024834615955139



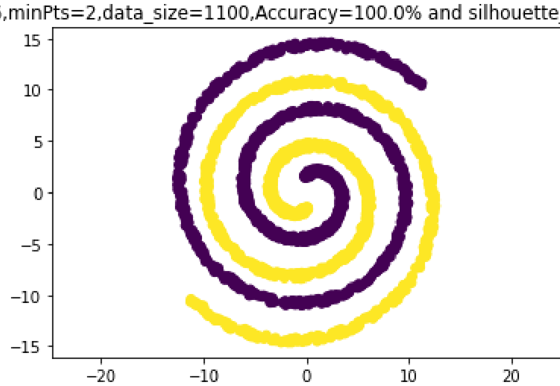
Eps=1.7000000000000006,minPts=2,data_size=1050,Accuracy=100.0% and silhouette_score=0.010500461902693653



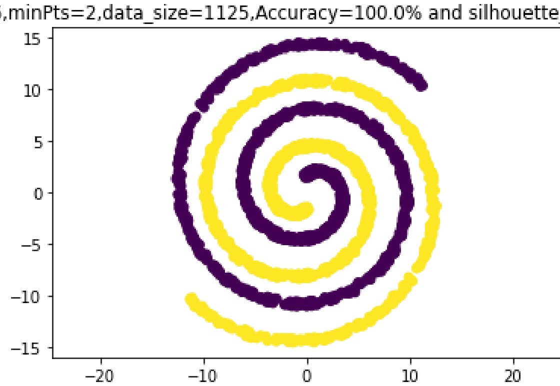
Eps=1.7000000000000006,minPts=2,data_size=1075,Accuracy=100.0% and silhouette_score=0.011346726194734221



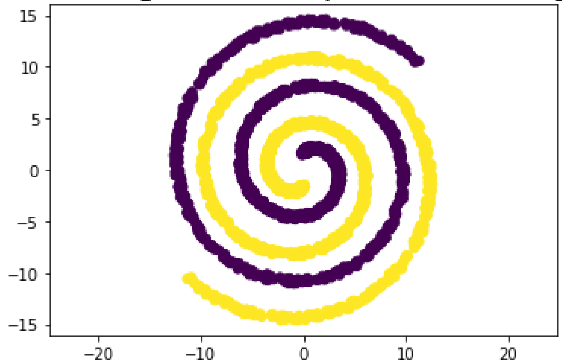
Eps=1.7000000000000006,minPts=2,data_size=1100,Accuracy=100.0% and silhouette_score=0.011717842736868054



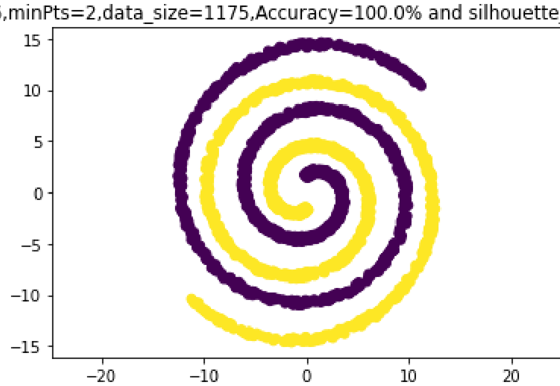
Eps=1.7000000000000006,minPts=2,data_size=1125,Accuracy=100.0% and silhouette_score=0.011832367583451478



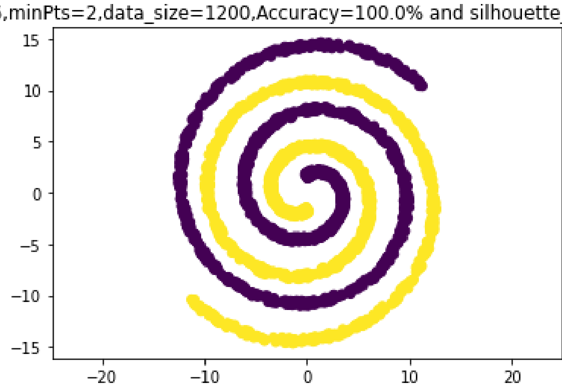
Eps=1.7000000000000006,minPts=2,data_size=1150,Accuracy=100.0% and silhouette_score=0.011850682424675598



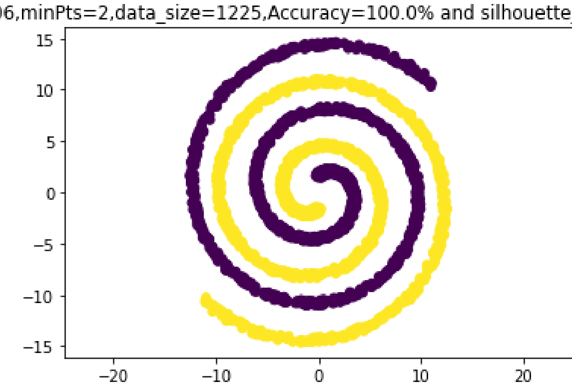
Eps=1.7000000000000006,minPts=2,data_size=1175,Accuracy=100.0% and silhouette_score=0.012078593388803375



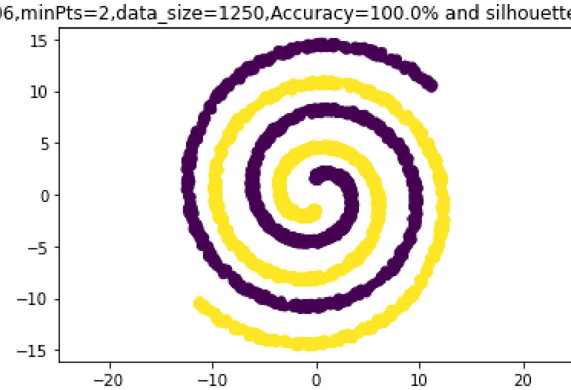
Eps=1.7000000000000006,minPts=2,data_size=1200,Accuracy=100.0% and silhouette_score=0.011971635584463437



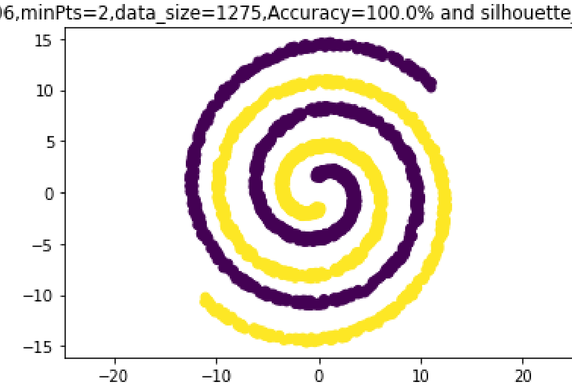
Eps=1.7000000000000006,minPts=2,data_size=1225,Accuracy=100.0% and silhouette_score=0.011258752832758014



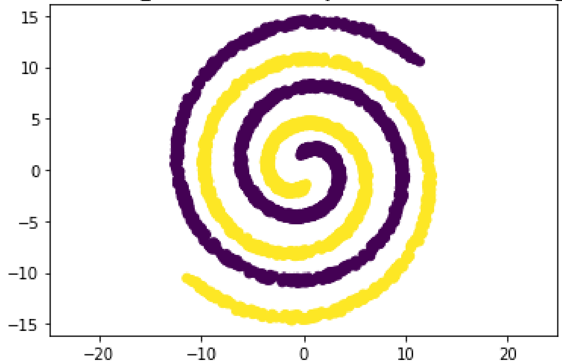
Eps=1.7000000000000006,minPts=2,data_size=1250,Accuracy=100.0% and silhouette_score=0.01156644712702995



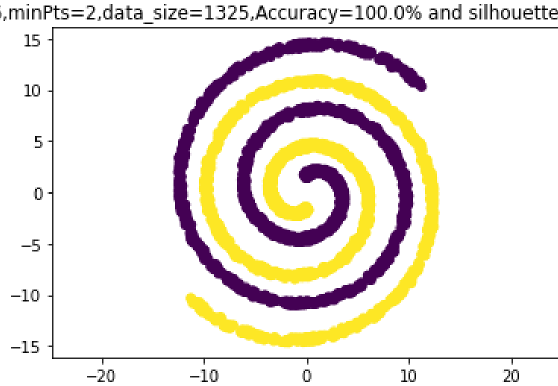
Eps=1.7000000000000006,minPts=2,data_size=1275,Accuracy=100.0% and silhouette_score=0.012104826468294501



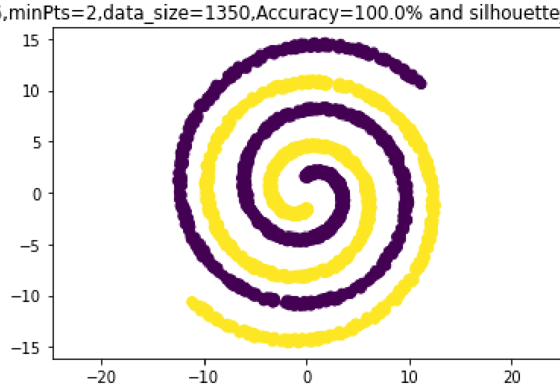
Eps=1.7000000000000006,minPts=2,data_size=1300,Accuracy=100.0% and silhouette_score=0.012306791813037527



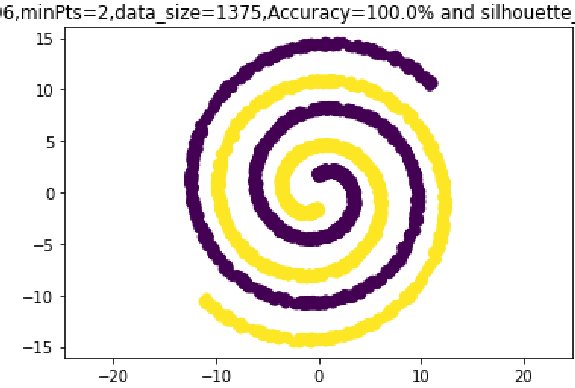
Eps=1.7000000000000006,minPts=2,data_size=1325,Accuracy=100.0% and silhouette_score=0.012607949783173252



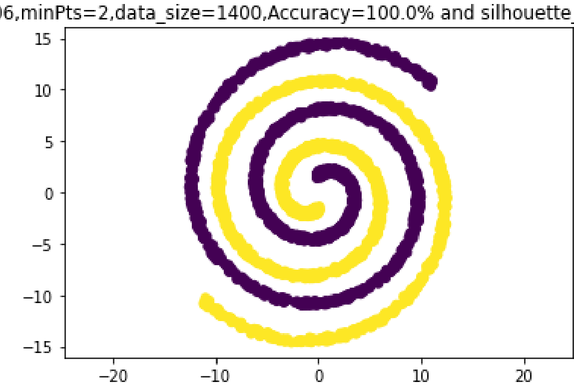
Eps=1.7000000000000006,minPts=2,data_size=1350,Accuracy=100.0% and silhouette_score=0.012116402357904113



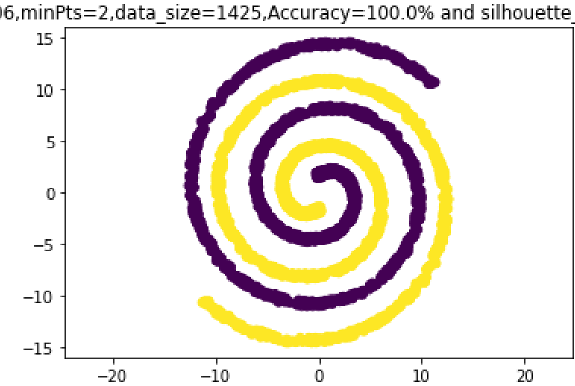
Eps=1.7000000000000006,minPts=2,data_size=1375,Accuracy=100.0% and silhouette_score=0.012175731173002148



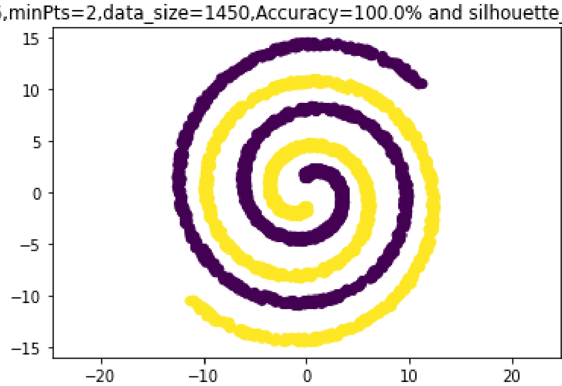
Eps=1.7000000000000006,minPts=2,data_size=1400,Accuracy=100.0% and silhouette_score=0.012526490334264173



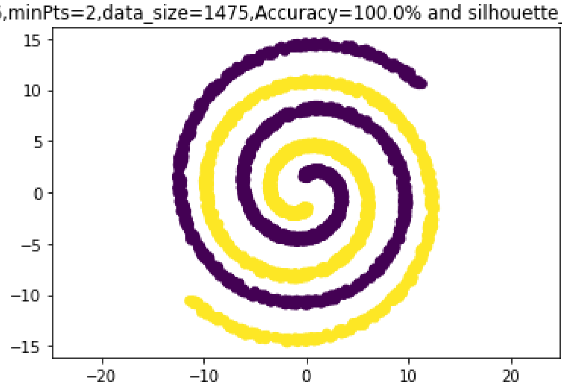
Eps=1.7000000000000006,minPts=2,data_size=1425,Accuracy=100.0% and silhouette_score=0.012367556734626892



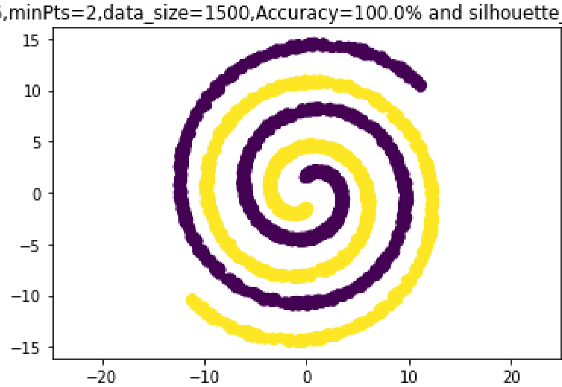
Eps=1.7000000000000006,minPts=2,data_size=1450,Accuracy=100.0% and silhouette_score=0.012117578472492424



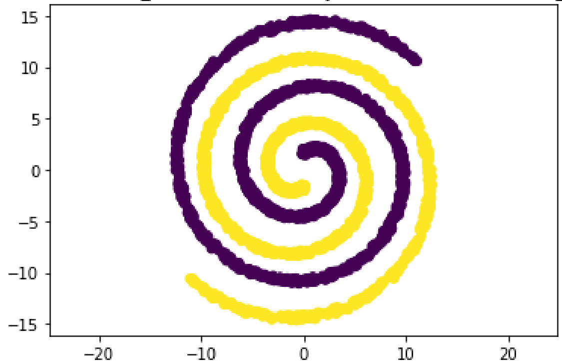
Eps=1.7000000000000006,minPts=2,data_size=1475,Accuracy=100.0% and silhouette_score=0.011642138461134082



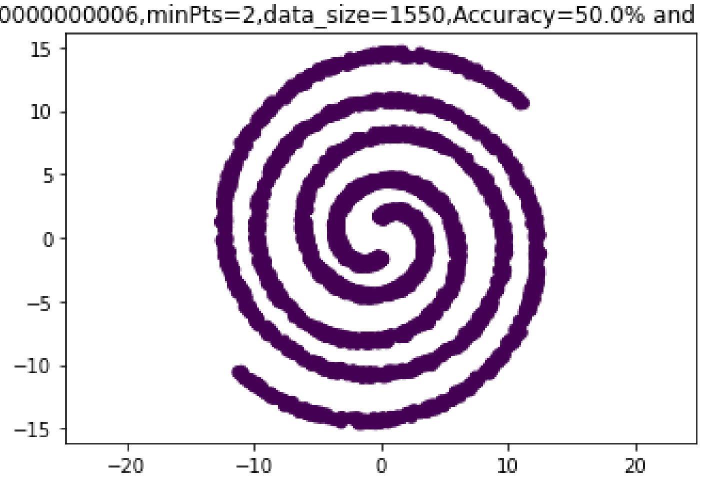
Eps=1.7000000000000006,minPts=2,data_size=1500,Accuracy=100.0% and silhouette_score=0.011429221137357243



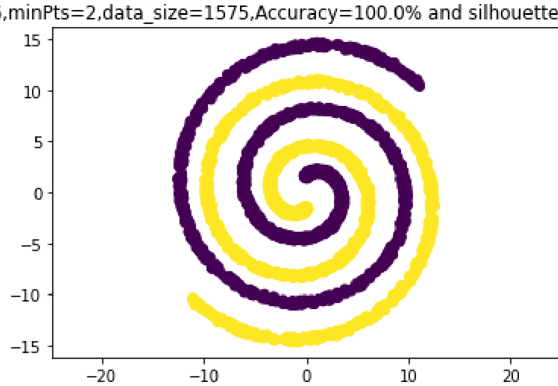
Eps=1.7000000000000006,minPts=2,data_size=1525,Accuracy=100.0% and silhouette_score=0.011361068123009188

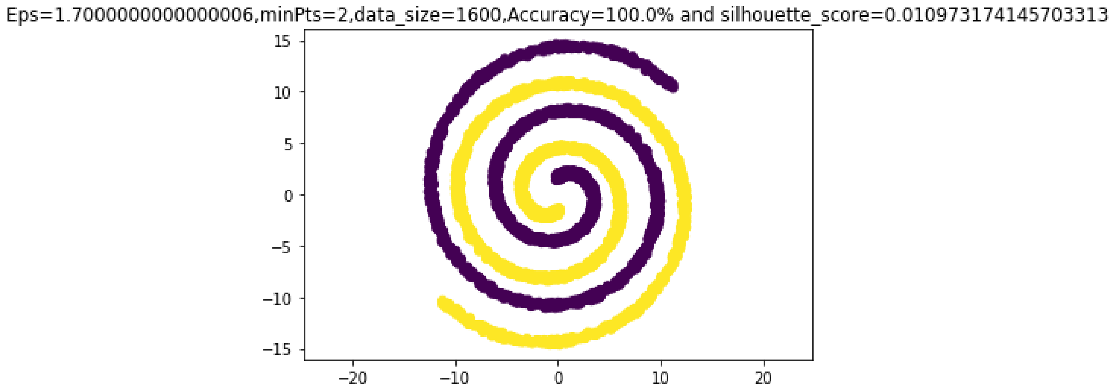


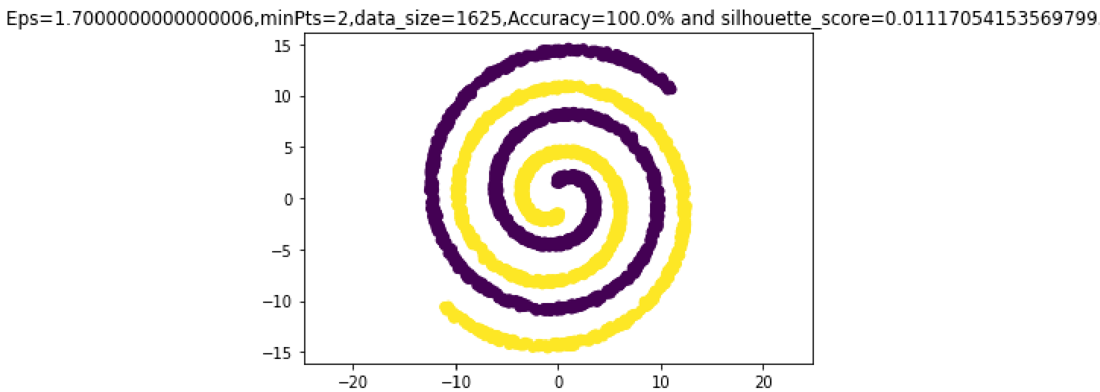
Eps=1.7000000000000006,minPts=2,data_size=1550,Accuracy=50.0% and silhouette_score=-1

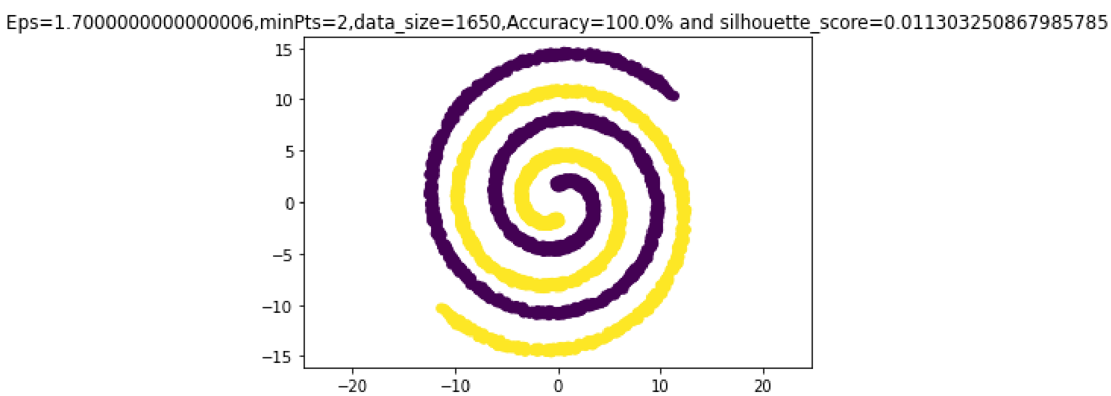


Eps=1.7000000000000006,minPts=2,data_size=1575,Accuracy=100.0% and silhouette_score=0.011579323984580895

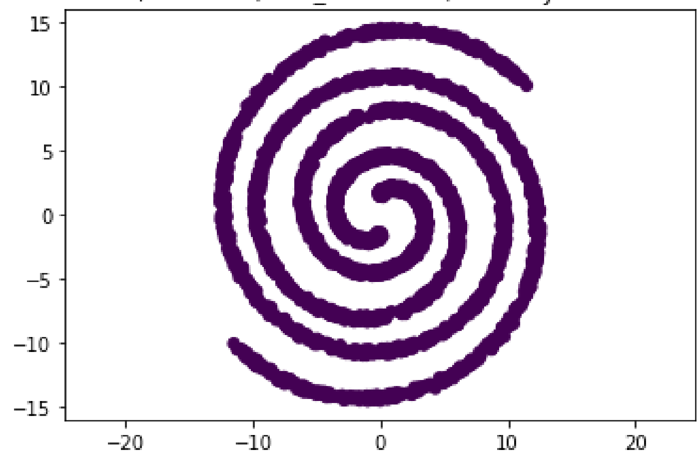



Eps=1.7000000000000006,minPts=2,data_size=1600,Accuracy=100.0% and silhouette_score=0.010973174145703313
15
10
5
0
−5
−10
−15
−20
−10
0
10
20

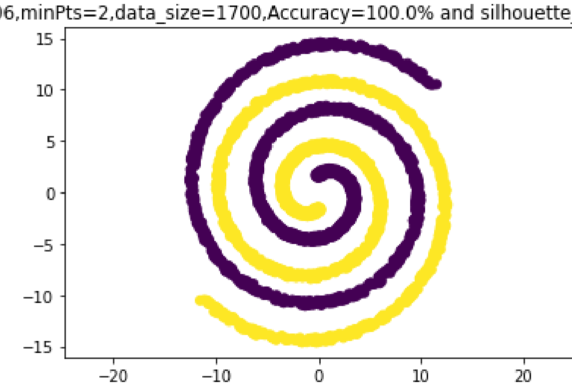

Eps=1.7000000000000006,minPts=2,data_size=1625,Accuracy=100.0% and silhouette_score=0.011170541535697995
15
10
5
0
−5
−10
−15
−20
−10
0
10
20


Eps=1.7000000000000006,minPts=2,data_size=1650,Accuracy=100.0% and silhouette_score=0.011303250867985785
15
10
5
0
−5
−10
−15
−20
−10
0
10
20

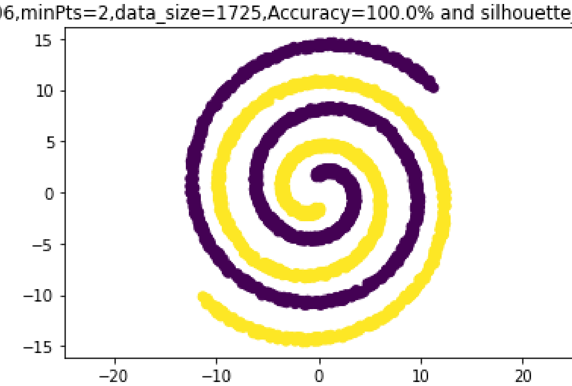
Eps=1.7000000000000006,minPts=2,data_size=1675,Accuracy=50.0% and silhouette_score=-1



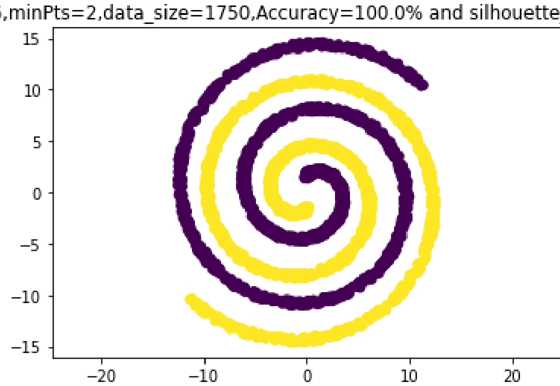
Eps=1.7000000000000006,minPts=2,data_size=1700,Accuracy=100.0% and silhouette_score=0.011389671845722465



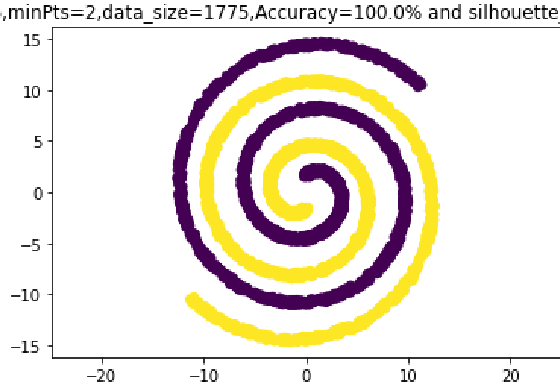
Eps=1.7000000000000006,minPts=2,data_size=1725,Accuracy=100.0% and silhouette_score=0.011067676792955596



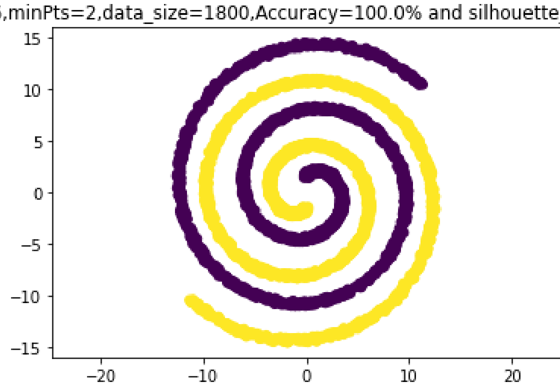
Eps=1.7000000000000006,minPts=2,data_size=1750,Accuracy=100.0% and silhouette_score=0.011353011399665259



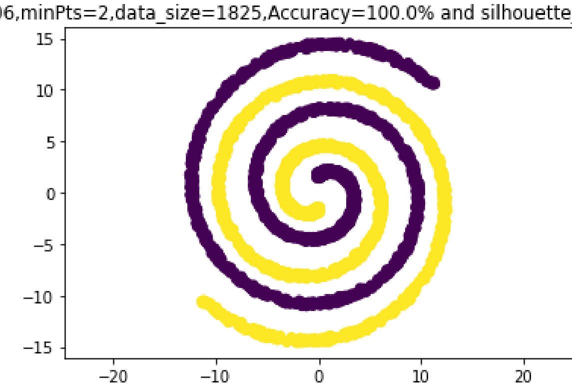
Eps=1.7000000000000006,minPts=2,data_size=1775,Accuracy=100.0% and silhouette_score=0.011672712399032309



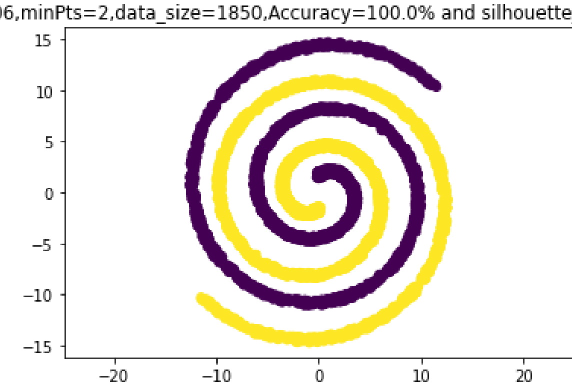
Eps=1.7000000000000006,minPts=2,data_size=1800,Accuracy=100.0% and silhouette_score=0.011437002208963326



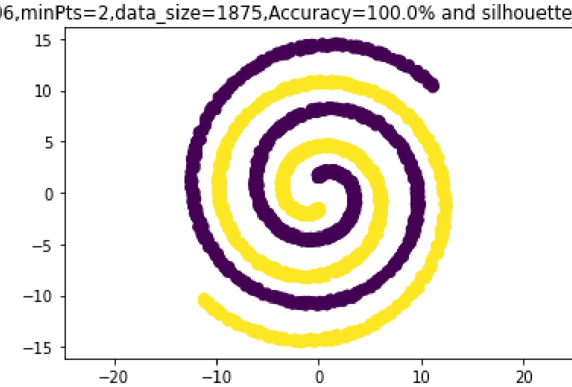
Eps=1.7000000000000006,minPts=2,data_size=1825,Accuracy=100.0% and silhouette_score=0.011734574967226517



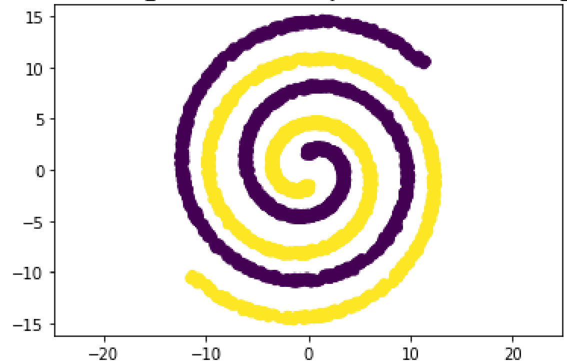
Eps=1.7000000000000006,minPts=2,data_size=1850,Accuracy=100.0% and silhouette_score=0.011853078336505547



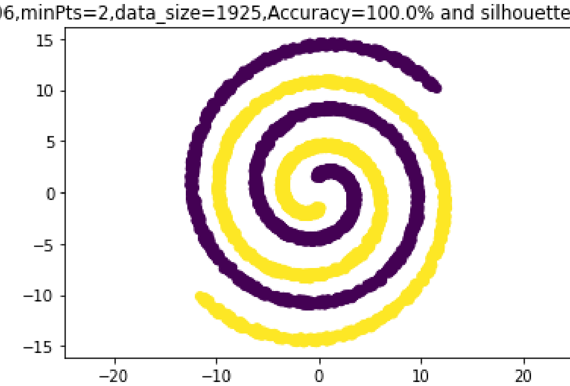
Eps=1.7000000000000006,minPts=2,data_size=1875,Accuracy=100.0% and silhouette_score=0.011543740048755835



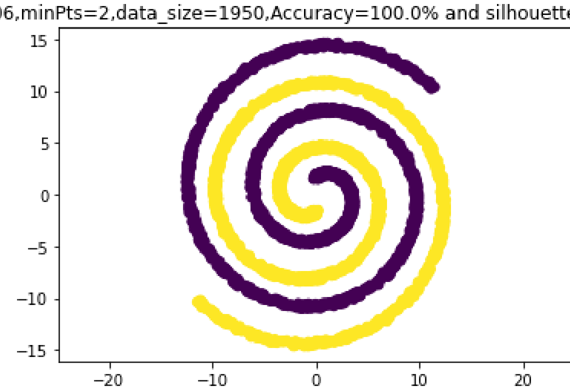
Eps=1.7000000000000006,minPts=2,data_size=1900,Accuracy=100.0% and silhouette_score=0.011308711599192072

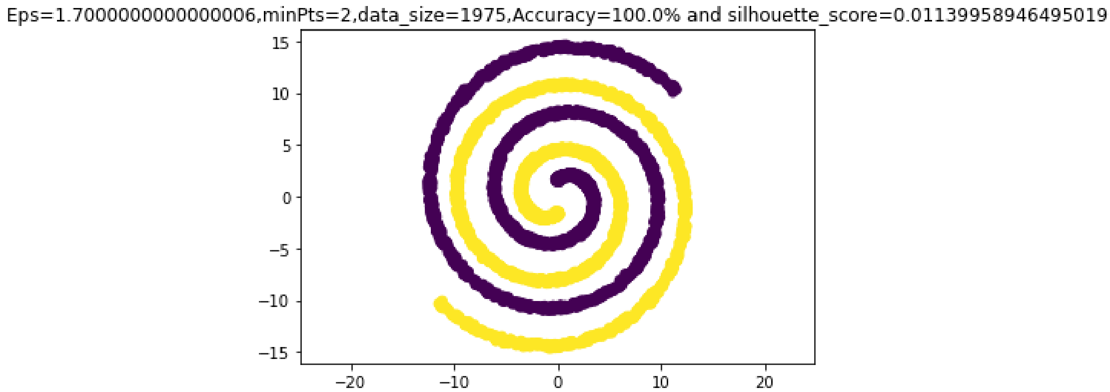


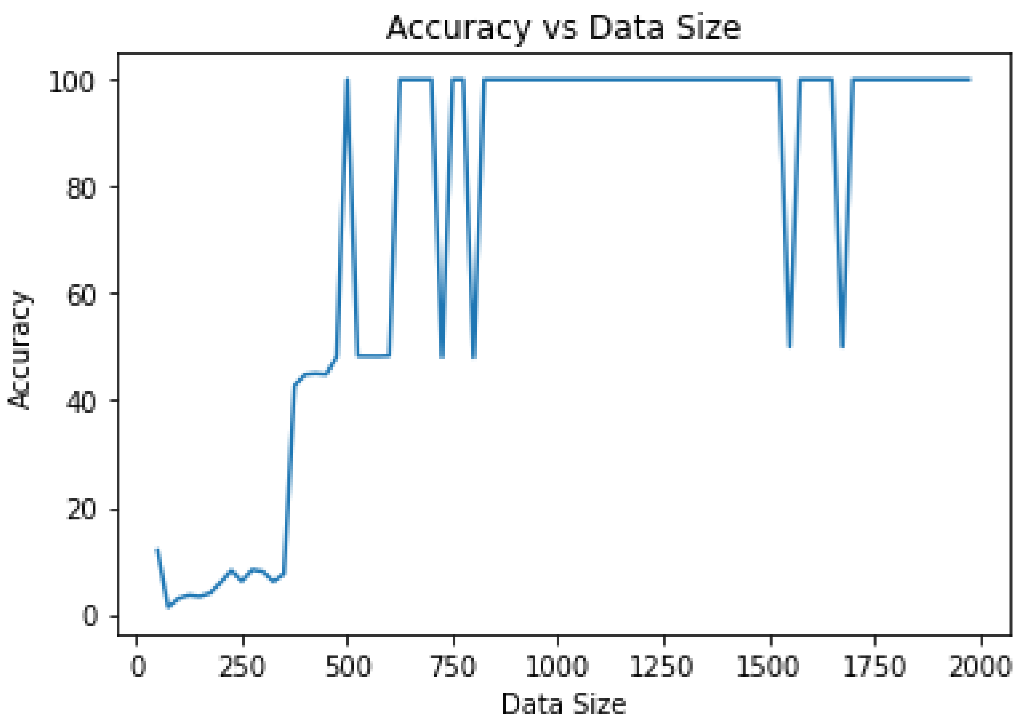
Eps=1.7000000000000006,minPts=2,data_size=1925,Accuracy=100.0% and silhouette_score=0.011393617760104986

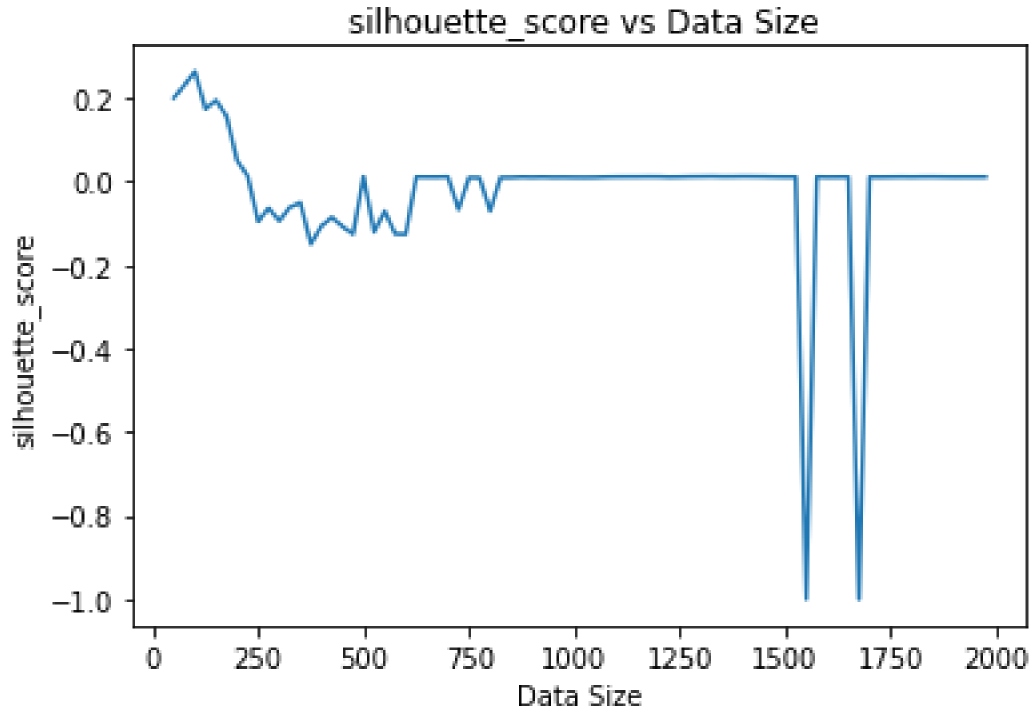


Eps=1.7000000000000006,minPts=2,data_size=1950,Accuracy=100.0% and silhouette_score=0.01131961135073266




Eps=1.7000000000000006,minPts=2,data_size=1975,Accuracy=100.0% and silhouette_score=0.01139958946495019
15
10
5
0
−5
−10
−15
−20
−10
0
10
20


Accuracy vs Data Size
Accuracy
100
80
60
40
20
0
0
250
500
750
1000
1250
1500
1750
2000
Data Size



[134]:
```
#(b) Print accuracies for different data_size values.
for data_size, acc in zip(data_size_list, acc_list):
    print('Data Size:',data_size,' Accuracy:',acc)
```

```
Data Size: 50  Accuracy: 12.0
Data Size: 75  Accuracy: 1.3333333333333333
Data Size: 100  Accuracy: 3.0
Data Size: 125  Accuracy: 3.6
Data Size: 150  Accuracy: 3.3333333333333335
Data Size: 175  Accuracy: 4.0
Data Size: 200  Accuracy: 6.0
Data Size: 225  Accuracy: 8.222222222222221
Data Size: 250  Accuracy: 6.2
Data Size: 275  Accuracy: 8.363636363636363
Data Size: 300  Accuracy: 8.0
Data Size: 325  Accuracy: 6.153846153846154
Data Size: 350  Accuracy: 7.571428571428571
Data Size: 375  Accuracy: 42.8
Data Size: 400  Accuracy: 44.875
Data Size: 425  Accuracy: 45.05882352941177
Data Size: 450  Accuracy: 44.888888888888886
Data Size: 475  Accuracy: 48.10526315789474
Data Size: 500  Accuracy: 100.0
Data Size: 525  Accuracy: 48.285714285714285
```

```
Data Size: 550  Accuracy: 48.27272727272727
Data Size: 575  Accuracy: 48.26086956521739
Data Size: 600  Accuracy: 48.333333333333336
Data Size: 625  Accuracy: 100.0
Data Size: 650  Accuracy: 100.0
Data Size: 675  Accuracy: 100.0
Data Size: 700  Accuracy: 100.0
Data Size: 725  Accuracy: 48.06896551724138
Data Size: 750  Accuracy: 100.0
Data Size: 775  Accuracy: 100.0
Data Size: 800  Accuracy: 48.0
Data Size: 825  Accuracy: 100.0
Data Size: 850  Accuracy: 100.0
Data Size: 875  Accuracy: 100.0
Data Size: 900  Accuracy: 100.0
Data Size: 925  Accuracy: 100.0
Data Size: 950  Accuracy: 100.0
Data Size: 975  Accuracy: 100.0
Data Size: 1000  Accuracy: 100.0
Data Size: 1025  Accuracy: 100.0
Data Size: 1050  Accuracy: 100.0
Data Size: 1075  Accuracy: 100.0
Data Size: 1100  Accuracy: 100.0
Data Size: 1125  Accuracy: 100.0
Data Size: 1150  Accuracy: 100.0
Data Size: 1175  Accuracy: 100.0
Data Size: 1200  Accuracy: 100.0
Data Size: 1225  Accuracy: 100.0
Data Size: 1250  Accuracy: 100.0
Data Size: 1275  Accuracy: 100.0
Data Size: 1300  Accuracy: 100.0
Data Size: 1325  Accuracy: 100.0
Data Size: 1350  Accuracy: 100.0
Data Size: 1375  Accuracy: 100.0
Data Size: 1400  Accuracy: 100.0
Data Size: 1425  Accuracy: 100.0
Data Size: 1450  Accuracy: 100.0
Data Size: 1475  Accuracy: 100.0
Data Size: 1500  Accuracy: 100.0
Data Size: 1525  Accuracy: 100.0
Data Size: 1550  Accuracy: 50.0
Data Size: 1575  Accuracy: 100.0
Data Size: 1600  Accuracy: 100.0
Data Size: 1625  Accuracy: 100.0
Data Size: 1650  Accuracy: 100.0
Data Size: 1675  Accuracy: 50.0
Data Size: 1700  Accuracy: 100.0
Data Size: 1725  Accuracy: 100.0
```

```
Data Size: 1750  Accuracy: 100.0
Data Size: 1775  Accuracy: 100.0
Data Size: 1800  Accuracy: 100.0
Data Size: 1825  Accuracy: 100.0
Data Size: 1850  Accuracy: 100.0
Data Size: 1875  Accuracy: 100.0
Data Size: 1900  Accuracy: 100.0
Data Size: 1925  Accuracy: 100.0
Data Size: 1950  Accuracy: 100.0
Data Size: 1975  Accuracy: 100.0
```

(c) For what kind of data_size values does the algorithm fail and why? What would you say are disadvantages of DBSCAN?

Ans: From above graph of accuracy versus data size, we can depict that with Data Size less than 500 DBSCAN fail as data is too sparse and if we keep varying the density then with some values of data size higher than 500 it fail sometimes. Note: Here data size 500 means 500(positive)+500(negative)
Disadvantages of DBSCAN:
-> Does not work well when dealing with clusters of varying densities and if the dataset is too sparse.
-> While DBSCAN is great at separating high density clusters from low density clusters, DBSCAN struggles with clusters of similar density.
-> Struggles with high dimensionality data.
-> Sampling affects density measures
-> Sensitive to clusters parameters Eps and minPts