Manaranjan Pradhan
manaranjan@enablecloud.com
*This notebook is given as part of **Data Science for everyone** workshop.*
(Forwarding this document to others is strictly prohibited.)

Introduction to Clustering - Unsupervised Learnings

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib as plt
import seaborn as sn
%matplotlib inline
```

Generate some random points

In [2]:

```
from sklearn.datasets.samples_generator import make_blobs
```

In [3]:

```
X, y = make_blobs(n_samples=300, centers=3,
                  random_state=0, cluster_std=0.60)
```

In [4]:

```
all_points = pd.concat( [pd.DataFrame( X ),
                         pd.DataFrame( y ) ],
                       axis = 1 )
```

In [6]:

```
all_points.columns = ["x1", "x2", "y"]
```

In [7]:

```
all_points.head()
```

Out[7]:

	x1	x2	y
0	0.428577	4.973997	0
1	1.619909	0.067645	1
2	1.432893	4.376792	0
3	-1.578462	3.034458	2
4	-1.658629	2.267460	2

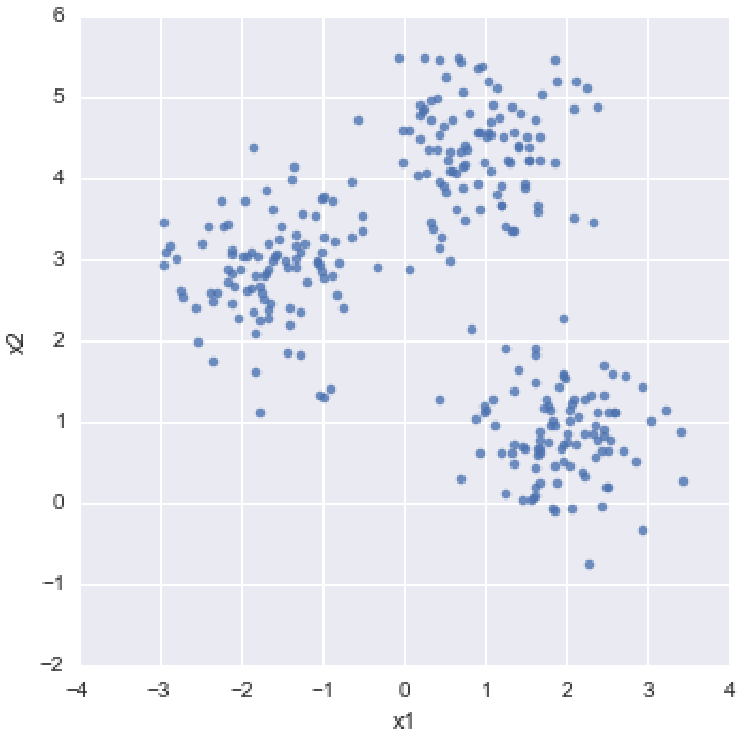
Draw the points on a graph and find out how they are scattered

In [8]:

```
sn.lmplot( "x1", "x2", data=all_points, fit_reg=False, size = 5 )
```

Out[8]:

```
<seaborn.axisgrid.FacetGrid at 0x8f01e48>
```



Can a clustering algorithm group them together by how nearer they are to each other

Using K-means clustering technique

In [9]:

```
from sklearn.cluster import KMeans
```

In [10]:

```
X = all_points[["x1", "x2"]]
clusters = KMeans(3)  # 3 clusters
clusters.fit( X )
```

Out[10]:

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=3, n_i
nit=10,
    n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0
001,
    verbose=0)
```

In [11]:

```
clusters.cluster_centers_
```

Out[11]:

```
array([[ 1.95159369,  0.83467497],
       [ 0.95625704,  4.37226546],
       [-1.60811992,  2.85881658]])
```

In [12]:

```
clusters.labels_
```

Out[12]:

```
array([1, 0, 1, 2, 2, 2, 0, 1, 2, 2, 0, 0, 0, 1, 0, 2, 1, 1, 2, 0, 2,
1, 0,
       1, 2, 2, 1, 2, 0, 0, 2, 1, 1, 0, 0, 2, 0, 2, 1, 0, 2, 0, 1, 0,
0, 2,
       0, 2, 2, 0, 2, 0, 2, 2, 0, 1, 1, 2, 2, 1, 0, 0, 1, 2, 0, 2, 1,
0, 1,
       0, 2, 2, 2, 2, 0, 1, 0, 2, 1, 1, 2, 1, 0, 1, 1, 1, 0, 2, 1, 1,
2, 0,
       2, 1, 0, 0, 1, 0, 2, 1, 0, 2, 1, 0, 1, 1, 2, 1, 0, 0, 1, 2, 1,
1, 2,
       2, 1, 1, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 2, 2, 1, 2, 2, 0,
2, 1,
       2, 2, 0, 1, 0, 1, 2, 2, 1, 2, 2, 0, 1, 2, 1, 0, 2, 2, 0, 0, 1,
0, 1,
       1, 0, 1, 2, 1, 1, 1, 1, 2, 0, 1, 2, 0, 0, 0, 1, 0, 1, 1, 0, 2,
1, 1,
       1, 1, 0, 2, 1, 2, 1, 1, 0, 0, 2, 1, 0, 2, 1, 2, 0, 2, 1, 2, 0,
2, 1,
       2, 1, 0, 1, 1, 2, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 0, 1, 2, 2, 1,
1, 2,
       1, 0, 0, 1, 0, 2, 2, 2, 1, 1, 0, 2, 2, 2, 2, 0, 2, 2, 1, 0, 0,
1, 0,
       0, 2, 1, 0, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 0, 0, 0, 0, 1, 1, 1,
1, 1,
       2, 2, 0, 1, 1, 0, 0, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 0, 1, 1, 2,
0, 1,
       2])
```

In [13]:

```
all_points["clusterid_1"] = clusters.labels_
```

In [14]:

```
all_points.head()
```

Out[14]:

	x1	x2	y	clusterid_1
0	0.428577	4.973997	0	1
1	1.619909	0.067645	1	0
2	1.432893	4.376792	0	1
3	-1.578462	3.034458	2	2
4	-1.658629	2.267460	2	2

In [16]:

```
sn.lmplot( "x1", "x2", data=all_points,
            hue = "clusterid_1",
            fit_reg=False, size = 5 )
```

Out[16]:

```
<seaborn.axisgrid.FacetGrid at 0xac1aa90>
```

How well the points were clustered

In [25]:

```
from sklearn.metrics import adjusted_rand_score
adjusted_rand_score(all_points.y, all_points.clusterid_1)
```

Out[25]:

```
1.0
```

Does the scale of dimensions impact the clustering?

In [26]:

```
all_points["x1"] = all_points.x1 * 100
```

In [27]:

```
all_points.head()
```

Out[27]:

	x1	x2	y	clusterid_1	clusterid_2
0	4285.767433	4.973997	0	1	0
1	16199.090944	0.067645	1	0	2
2	14328.927136	4.376792	0	1	2
3	-15784.624734	3.034458	2	2	1
4	-16586.286302	2.267460	2	2	1

In [28]:

```
X = all_points[["x1", "x2"]]
clusters = KMeans(3)  # 3 clusters
clusters.fit( X )
```

Out[28]:

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=3, n_i
nit=10,
    n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0
001,
    verbose=0)
```

In [29]:

```
all_points["clusterid_2"] = clusters.labels_
sn.lmplot( "x1", "x2", data=all_points,
          hue = "clusterid_2",
          fit_reg=False, size = 5 )
```

Out[29]:

```
<seaborn.axisgrid.FacetGrid at 0xade6da0>
```

Scale the dimensions to remove the impact

In [30]:

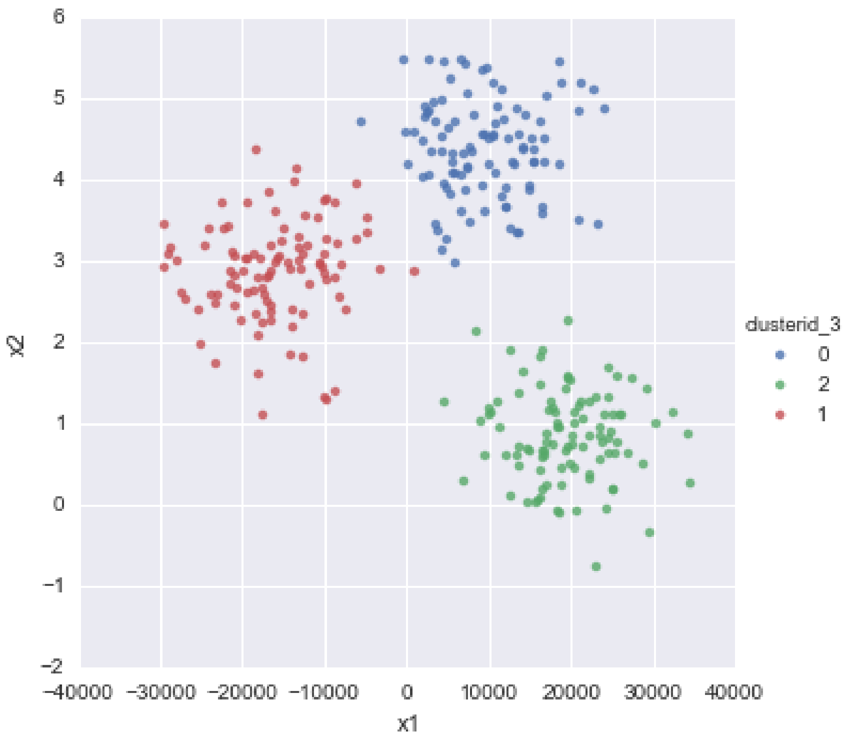
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform( X )
```

In [31]:

```
clusters = KMeans(3)  # 3 clusters
clusters.fit( X_scaled )
all_points["clusterid_3"] = clusters.labels_
sn.lmplot( "x1", "x2", data=all_points,
          hue = "clusterid_3",
          fit_reg=False, size = 5 )
```

Out[31]:

<seaborn.axisgrid.FacetGrid at 0x7d10b8>



Can K-means work if the clusters are not well segregated.. what if the clustered are interspersed

In [32]:

```
from sklearn import datasets
moon_points = datasets.make_moons(n_samples=1000, noise=.05)
```

In [33]:

```
X, y = enumerate( moon_points )
```

In [34]:

```
moon_points = pd.DataFrame( X[1] )
```

In [35]:

```
moon_points.columns = ["x1", "x2"]
```

In [36]:

```
moon_points["y"] = y[1]
```

In [37]:

```
moon_points.head()
```

Out[37]:

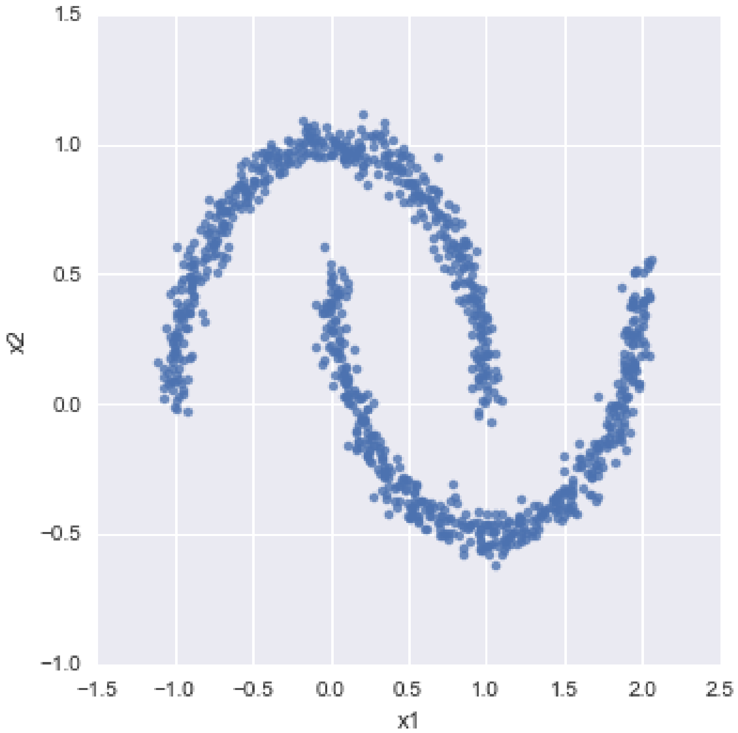
	x1	x2	y
0	0.113708	0.024278	1
1	-0.574124	0.913837	0
2	1.889860	0.191694	1
3	1.130859	-0.534824	1
4	1.068145	-0.532721	1

In [38]:

```
sn.lmplot( "x1", "x2", data=moon_points, fit_reg=False, size = 5 )
```

Out[38]:

<seaborn.axisgrid.FacetGrid at 0x808470>

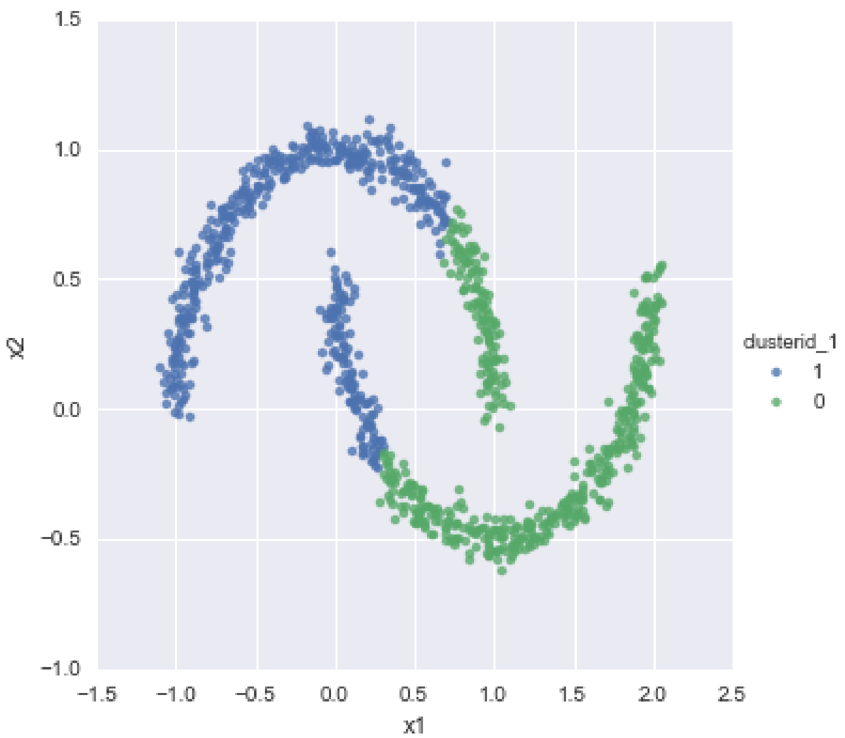


In [40]:

```
moon_clusters = KMeans(2)  # 3 clusters
moon_clusters.fit( moon_points[["x1", "x2"]] )
moon_points["clusterid_1"] = moon_clusters.labels_
sn.lmplot( "x1", "x2", data=moon_points,
          hue = "clusterid_1",
          fit_reg=False, size = 5 )
```

Out[40]:

```
<seaborn.axisgrid.FacetGrid at 0x8e3e10>
```



Using DBSCAN for density based clutering

In [41]:

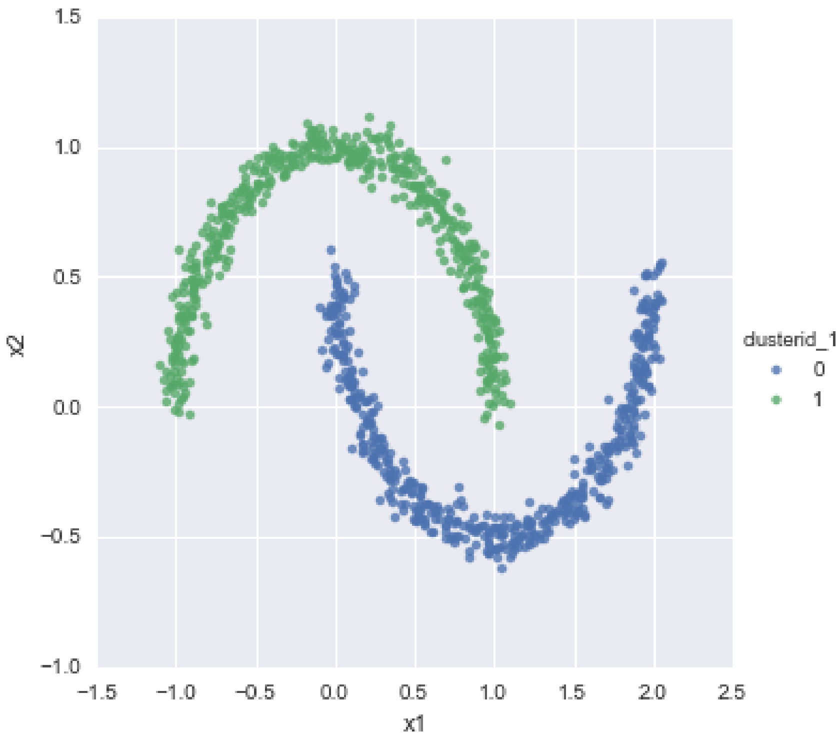
```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=.2)
```

In [42]:

```
moon_clusters = DBSCAN( eps=.2 )
moon_clusters.fit( moon_points[["x1", "x2"]] )
moon_points["clusterid_1"] = moon_clusters.labels_
sn.lmplot( "x1", "x2", data=moon_points,
          hue = "clusterid_1",
          fit_reg=False, size = 5 )
```

Out[42]:

```
<seaborn.axisgrid.FacetGrid at 0x977e80>
```



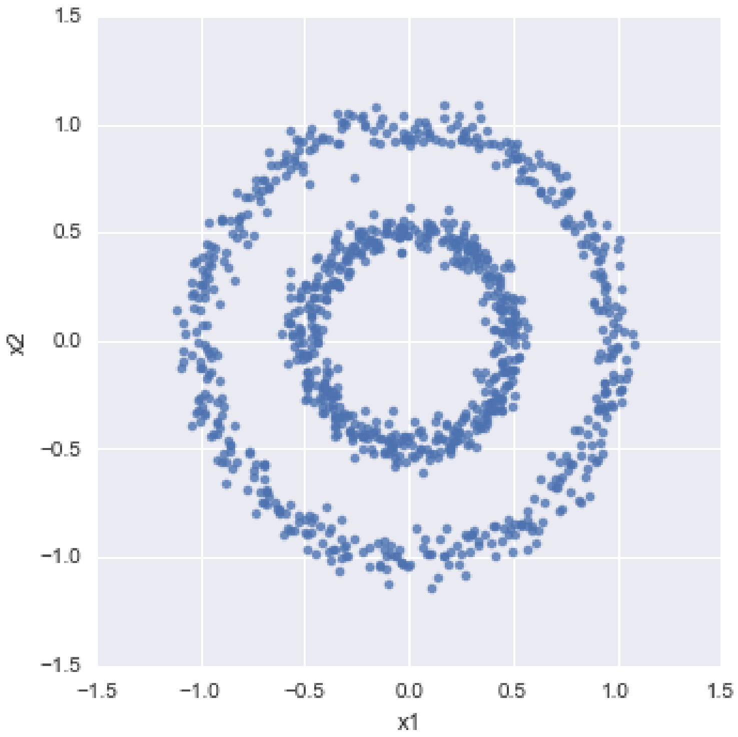
Using DBSCAN for points in circles..

In [46]:

```
circle_points = datasets.make_circles(n_samples=1000, factor=.5,
                                      noise=.05)
X, y = enumerate( circle_points )
circle_points = pd.DataFrame( X[1] )
circle_points.columns = ["x1", "x2"]
circle_points["y"] = y[1]
circle_points.head()
sn.lmplot( "x1", "x2", data=circle_points, fit_reg=False, size = 5 )
```

Out[46]:

<seaborn.axisgrid.FacetGrid at 0x23b5080>

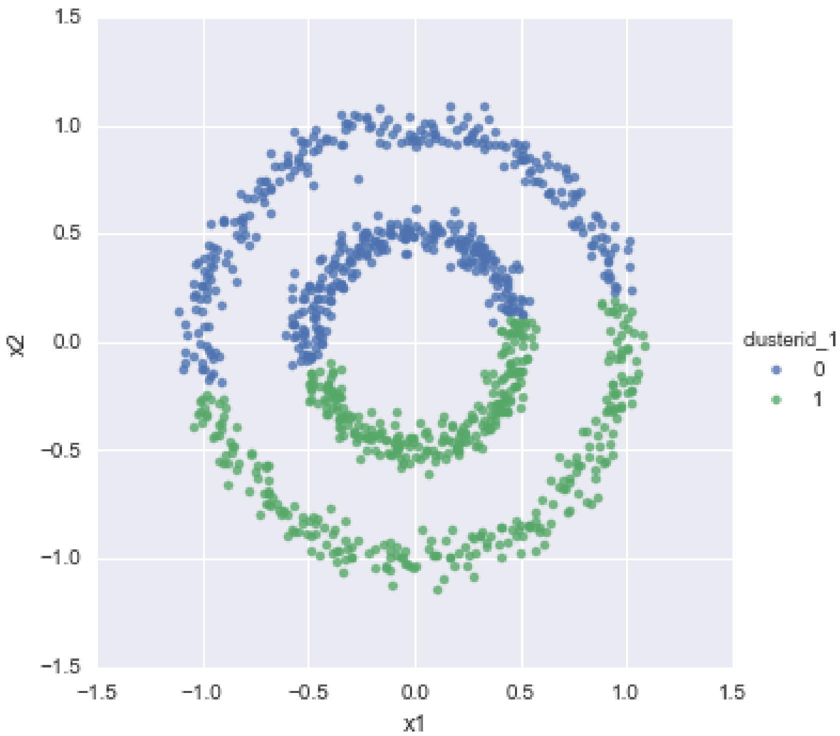


In [50]:

```
circle_clusters = KMeans(2)  # 3 clusters
circle_clusters.fit( circle_points[["x1", "x2"]] )
circle_points["clusterid_1"] = circle_clusters.labels_
sn.lmplot( "x1", "x2", data=circle_points,
          hue = "clusterid_1",
          fit_reg=False, size = 5 )
```

Out[50]:

```
<seaborn.axisgrid.FacetGrid at 0x23ed748>
```

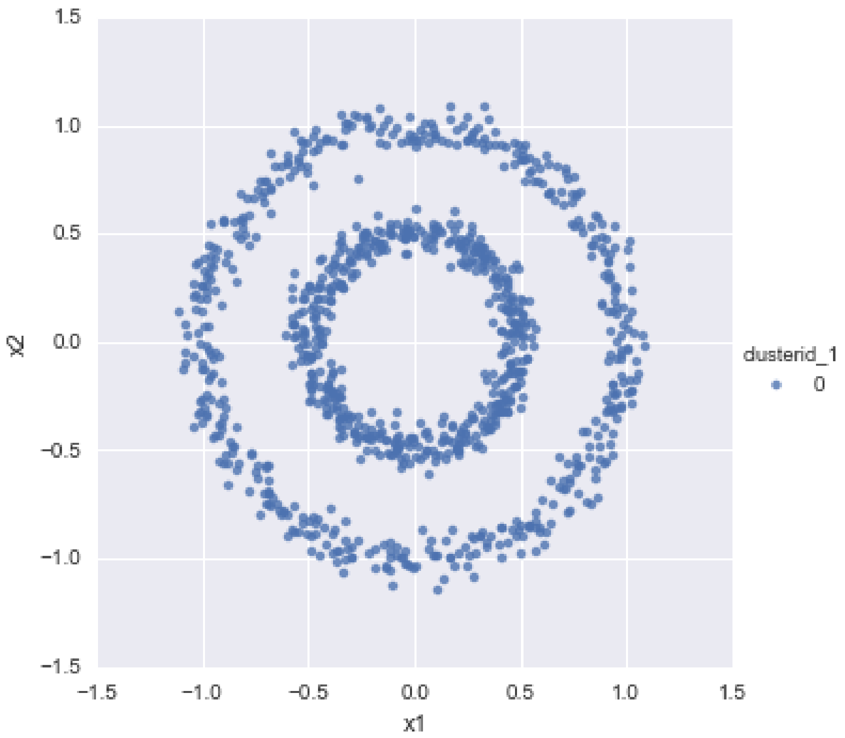


In [52]:

```
circle_clusters = DBSCAN( eps=.2 )
circle_clusters.fit( circle_points[["x1", "x2"]] )
circle_points["clusterid_1"] = circle_clusters.labels_
sn.lmplot( "x1", "x2", data=circle_points,
          hue = "clusterid_1",
          fit_reg=False, size = 5 )
```

Out[52]:

```
<seaborn.axisgrid.FacetGrid at 0xbe39748>
```



Make note of lessons learnt in this exercise