Hands-on Exercise for CLUS Module

0. Setting up necessary packages and creating data

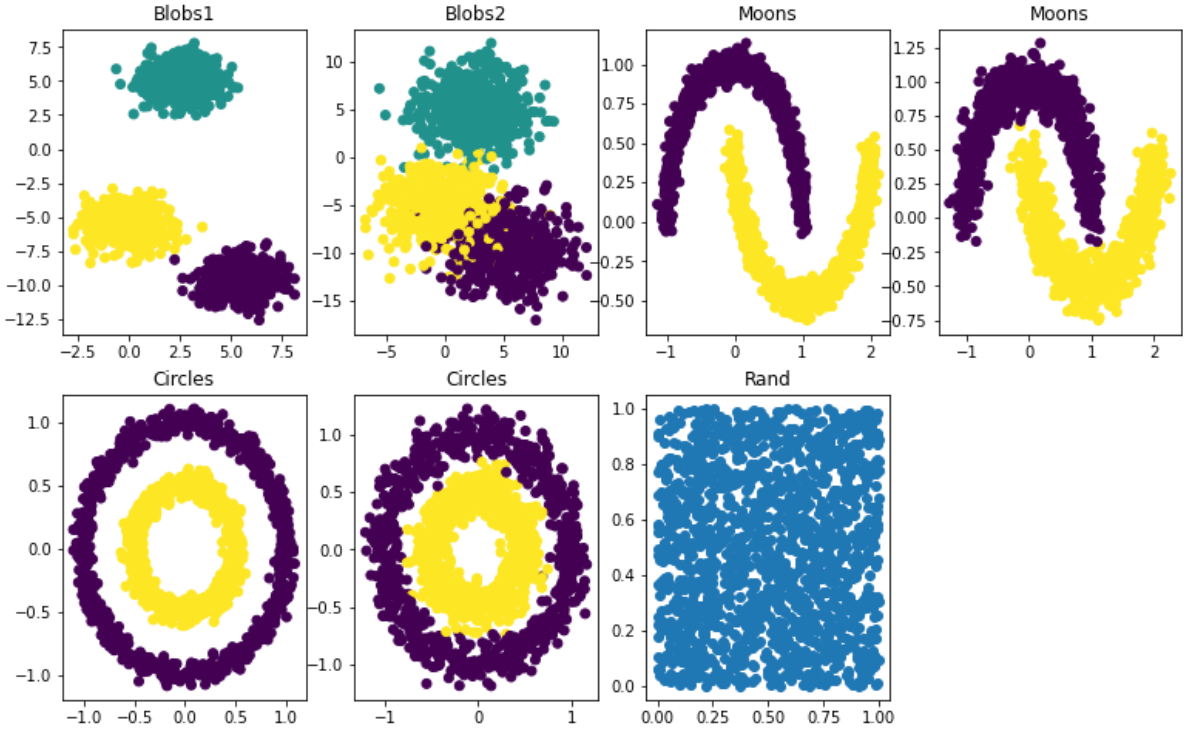
In [1]:
```
!pip install --user scikit-learn --upgrade
```

```
Traceback (most recent call last):
  File "/usr/local/anaconda5/bin/pip", line 7, in <module>
    from pip import main
ImportError: cannot import name 'main'
```

Import necessary packages

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns

from sklearn import datasets

# importing clustering algorithms
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
from sklearn.cluster import SpectralClustering

from sklearn.metrics import silhouette_samples
```

In [2]:
```
n_samples = 1500
random_state = 10

Blobs1_X, Blobs1_y = datasets.make_blobs(n_samples=n_samples,
                                random_state=random_state)
Blobs2_X, Blobs2_y = datasets.make_blobs(n_samples=n_samples,
                                cluster_std=[2.5, 2.5, 2.5],
                                random_state=random_state)
Moons1_X, Moons1_y = datasets.make_moons(n_samples=n_samples, noise=0.05,
                                random_state=random_state)
Moons2_X, Moons2_y = datasets.make_moons(n_samples=n_samples, noise=0.1,
                                random_state=random_state)
Circles1_X, Circles1_y = datasets.make_circles(n_samples=n_samples, factor=.5,
                                        noise=.05, random_state=random_state)
Circles2_X, Circles2_y = datasets.make_circles(n_samples=n_samples, factor=.5,
                                        noise=0.1, random_state=random_state)
Rand_X = np.random.rand(n_samples, 2);
plt.figure(figsize=(13,8))

plt.subplot(2,4,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c= Blobs1_y)
plt.title('Blobs1')

plt.subplot(2,4,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c= Blobs2_y)
plt.title('Blobs2')

plt.subplot(2,4,3)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c= Moons1_y)
plt.title('Moons')

plt.subplot(2,4,4)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c= Moons2_y)
plt.title('Moons')

plt.subplot(2,4,5)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c= Circles1_y)
plt.title('Circles')

plt.subplot(2,4,6)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c= Circles2_y)
plt.title('Circles')

plt.subplot(2,4,7)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1])
plt.title('Rand')
plt.show()
```



Code for RandIndex function

In [3]:
```
from scipy.special import comb
def rand_index(S, T):

    Spairs = comb(np.bincount(S), 2).sum()
    Tpairs = comb(np.bincount(T), 2).sum()

    A = np.c_[(S, T)]

    f_11 = sum(comb(np.bincount(A[A[:, 0] == i, 1]), 2).sum()
               for i in set(S))

    f_10 = Spairs - f_11
    f_01 = Tpairs - f_11
    f_00 = comb(len(A), 2) - f_11 - f_10 - f_01
    return (f_00 + f_11) / (f_00 + f_01 + f_10 + f_11)
```

Code for Hopkins statistic

In [4]:
```
from sklearn.neighbors import NearestNeighbors
from random import sample
from numpy.random import uniform
from math import isnan
def hopkins(X):
    n=X.shape[0]#rows
    d=X.shape[1]#cols
    p=int(0.1*n)#considering 10% of points
    nbrs=NearestNeighbors(n_neighbors=1).fit(X)

    rand_X=sample(range(0,n),p)
    uj=[]
    wj=[]
    for j in range(0,p):
        u_dist,_=nbrs.kneighbors(uniform(np.amin(X,axis=0),np.amax(X,axis=0),d
).reshape(1,-1),2,return_distance=True)
        uj.append(u_dist[0][1])#distances to nearest neighbors in random data
        w_dist,_=nbrs.kneighbors(X[rand_X[j]].reshape(1,-1),2,return_distance=
True)
        wj.append(w_dist[0][1])#distances to nearest neighbors in real data
    H=sum(uj)/(sum(uj)+sum(wj))
    if isnan(H):
        print(uj,wj)
        H=0

    return H
```

Code for Silhouette coefficient

In [5]:
```
def silhouette(X,labels):
    n_clusters=np.size(np.unique(labels));
    sample_silhouette_values=silhouette_samples(X,labels)
    y_lower=10
    for i in range(n_clusters):
        ith_cluster_silhouette_values=sample_silhouette_values[labels==i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i=ith_cluster_silhouette_values.shape[0]
        y_upper=y_lower+size_cluster_i
        color=cm.nipy_spectral(float(i)/n_clusters)
        plt.fill_betweenx(np.arange(y_lower,y_upper),0,ith_cluster_silhouette_
values,facecolor=color,edgecolor=color,alpha=0.7)# Label the silhouette plots
 with their cluster numbers at the middle
        plt.text(-0.05,y_lower+0.5*size_cluster_i,str(i))#Compute the new y_lo
wer for next cluster
        y_lower=y_upper+10# 10 for the 0 samples
    plt.title("Silhouette plot for the various clusters.")
    plt.xlabel("Silhouette coefficient values")
    plt.ylabel("Cluster label")
    plt.show()
```

1. K-Means clustering

Question 1a: Without running K-Means clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where k-Means is expected to work well. Support your answer by explaining your rationale.

Answer: K-means is expected to work well on datasets that do not have too much noise. It is generally more efficient than the other clustering algorithms, because its complexity is linear time, so it would work well on large datasets with distinct clusters.

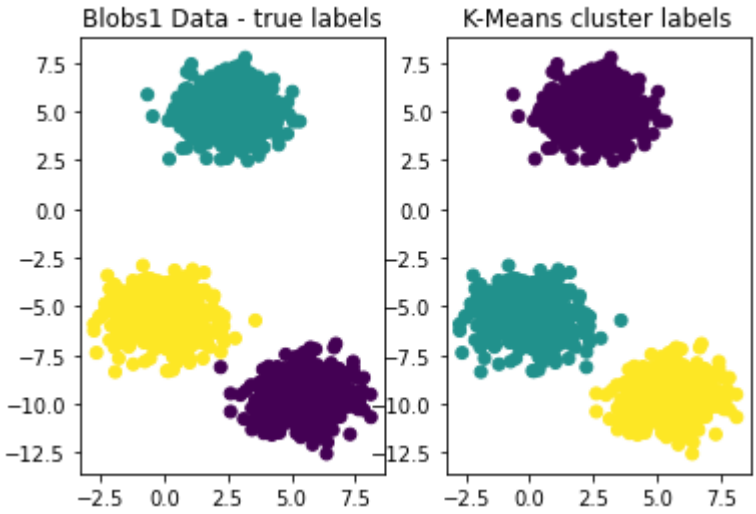
Question 1b: Without running K-Means clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where k-Means is expected to NOT work well. Support your answer by explaining your rationale.

Answer: K-means would not be expected to work well on datasets with non-blob like boundaries. It generally would not do so well with a lot of noise either.

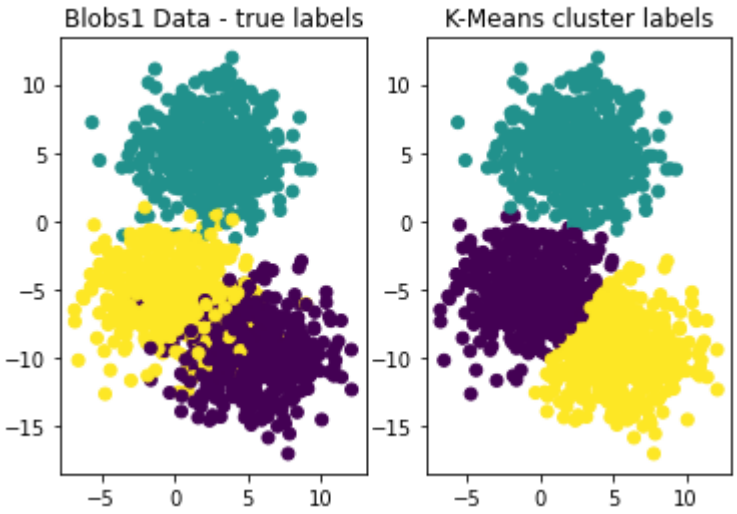
Question 1c: Run K-Means algorithm on all the datasets (except Rand). Choose n_clusters based on the number of clusters present in these datasets. Visualize the clusters for each of them. Based on the visualization, rank the datasets in decreasing order of K-means performance. Describe your rationale for your ranking.

Answer: Blobs_1 had the best performance with k means over all the datasets. This is because it has very distinct blob like cluster. These are the types of clusters that k means are very good at capturing. It did not perform as well on blobs_2 because some of the clusters overlap, though it does still perform very well on this dataset regardless. Due to the irregular boundaries of the other datasets, it did not perform very well. The moons performed slightly better than the circles by capturing a little more than half of the proper classifications while the circles were an even split.

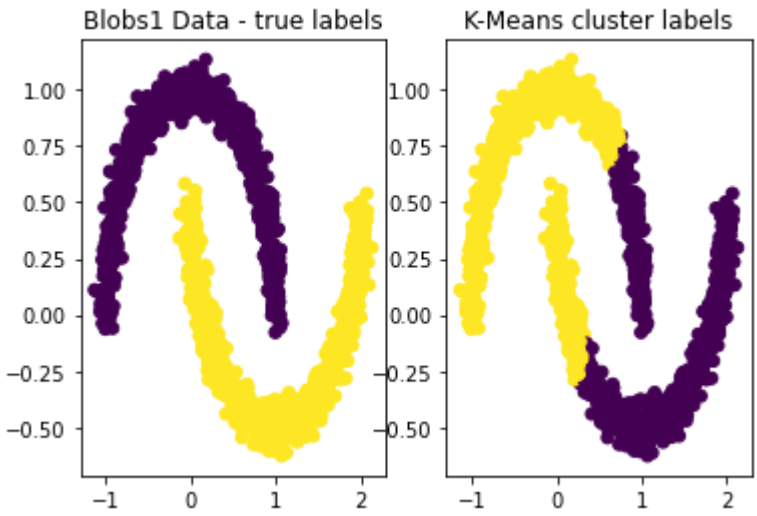
In [6]:
```
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred = kmeans.fit_predict(Blobs1_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y) # true clusters
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



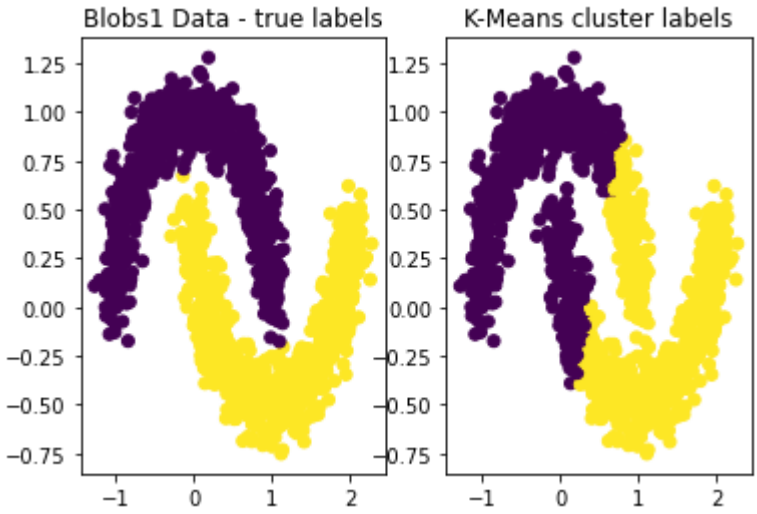
In [7]:
```
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred = kmeans.fit_predict(Blobs2_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y) # true clusters
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



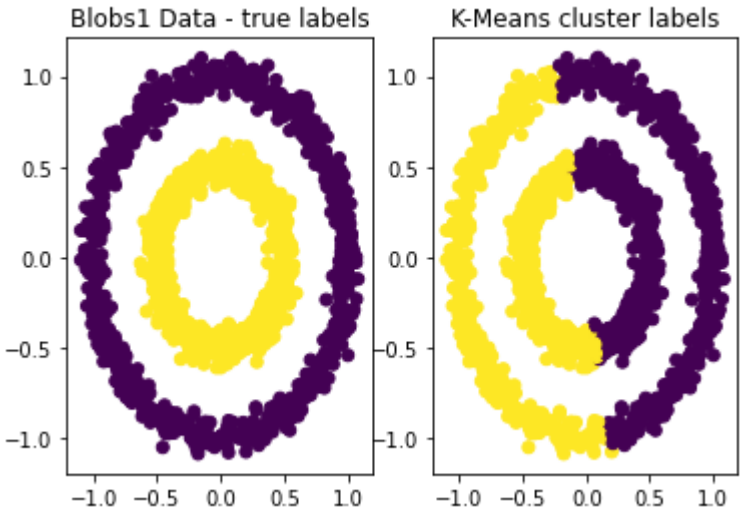
In [8]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred = kmeans.fit_predict(Moons1_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y) # true clusters
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



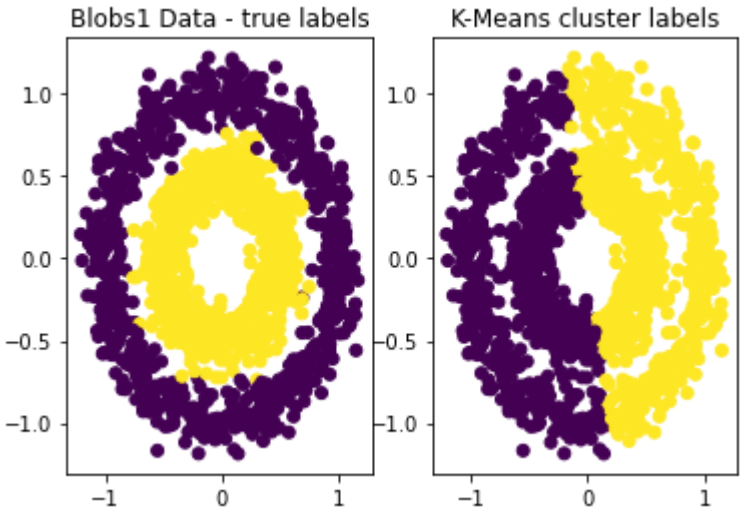
In [9]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred = kmeans.fit_predict(Moons2_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y) # true clusters
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



In [10]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred = kmeans.fit_predict(Circles1_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=Circles1_y) # true clusters
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



In [11]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred = kmeans.fit_predict(Circles2_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=Circles2_y) # true clusters
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



In [ ]:

Question 1d: For each of the datasets, compute Rand-Index value between the true labels and cluster memberships computed using K-means. Rank the datasets in decreasing order of Rand-Index scores.

Answer: In decreasing order the rand index order is blob1, blobs 2, moons 2, moons 1, circles 2, circles 1

In [12]:
```
n_clusters = 3;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred_kmeans = kmeans.fit_predict(Blobs1_X)
print(rand_index(y_pred_kmeans, Blobs1_y))
```

```
0.99911140760507
```

In [13]:
```
n_clusters = 3;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred_kmeans = kmeans.fit_predict(Blobs2_X)
print(rand_index(y_pred_kmeans, Blobs2_y))
```

```
0.9199573048699132
```

In [14]:
```
n_clusters = 2;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred_kmeans = kmeans.fit_predict(Moons1_X)
print(rand_index(y_pred_kmeans, Moons1_y))
```

```
0.6201236379808761
```

In [15]:
```
n_clusters = 2;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred_kmeans = kmeans.fit_predict(Moons2_X)
print(rand_index(y_pred_kmeans, Moons2_y))
```

```
0.6240836112964199
```

In [16]:
```
n_clusters = 2;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred_kmeans = kmeans.fit_predict(Circles1_X)
print(rand_index(y_pred_kmeans, Circles1_y))
```

```
0.4997233711363131
```

In [17]:
```
n_clusters = 2;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred_kmeans = kmeans.fit_predict(Circles2_X)
print(rand_index(y_pred_kmeans, Circles2_y))
```

```
0.49979452968645766
```

Question 1e: Are the rankings in (c) consistent with your observations in (d)? If not, explain the reason why your rankings were inconsistent.

Answer: The rankings were consistent with my observations in (c). The rand statistic indicated the highest similarity for the blob datasets, moons, and then circles, respectively.

2. Agglomerative Clustering - Single Link

Question 2a: Without running Single-link agglomerative clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Single-link agglomerative clustering is expected to work well. Support your answer by explaining your rationale.

Answer: The data sets single link clustering will perform well on are data set Blobs 1, Moons 1, and Circles 1. This is because there are there no points between clusters that are close enough to agglomerate the separate clusters together when choosing the closest link.

Question 2b: Without running Single-link agglomerative clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Single-link agglomerative clustering is expected to NOT work well. Support your answer by explaining your rationale.

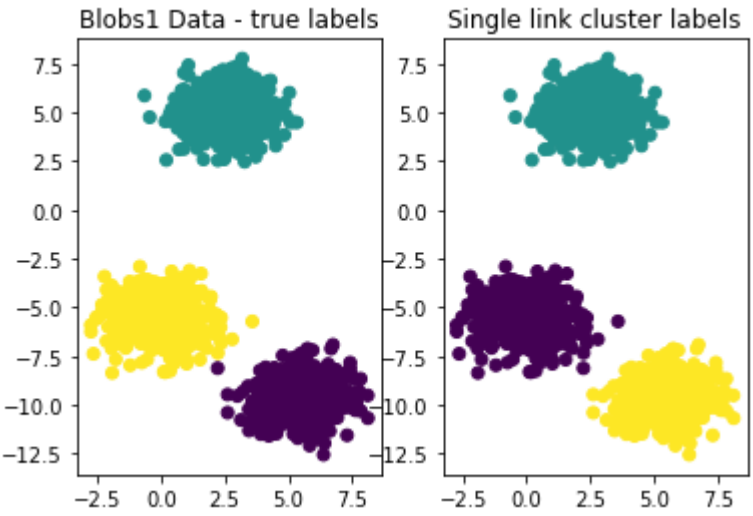
Answer: Single link clustering will not work well on any of the 2 data sets, because there are points from separate clusters that are close enough to be agglomerated before the clustering is over.

Question 2c: Run Single-link agglomerative clustering algorithm on all the datasets (except Rand). Choose n_clusters based on the number of clusters present in these datasets. Visualize the clusters for each of them. Based on the visualization, rank the datasets in decreasing order of Single-link agglomerative algorithm performance. Describe your rationale for your ranking.

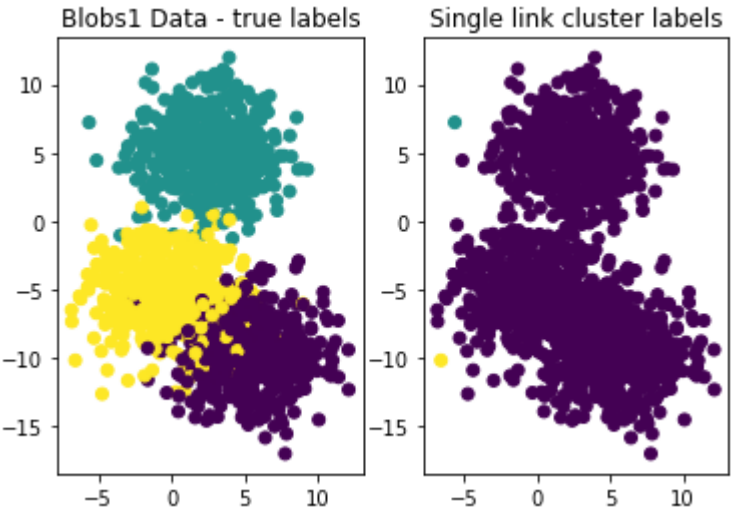
Answer:

Blobs 1, Moons 1, and Circles 1 are all very well separated. The others are poor, because different clusters all became agglomerated together.

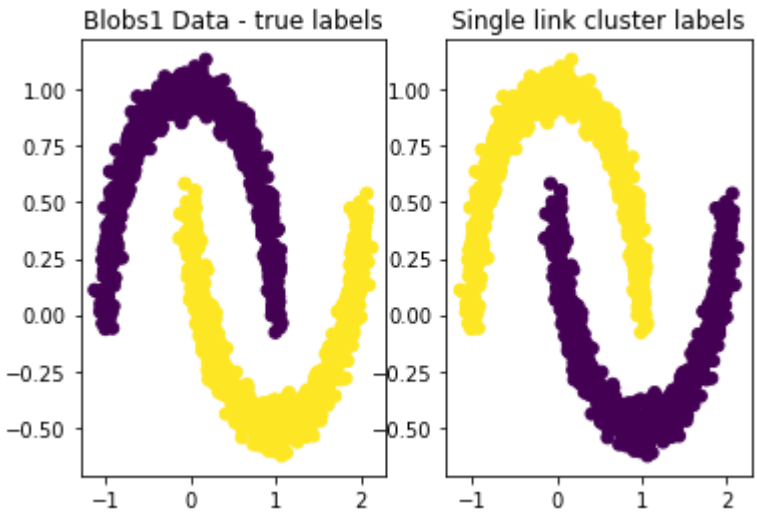
In [18]:
```
n_clusters = 3
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred = single_linkage.fit_predict(Blobs1_X)
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



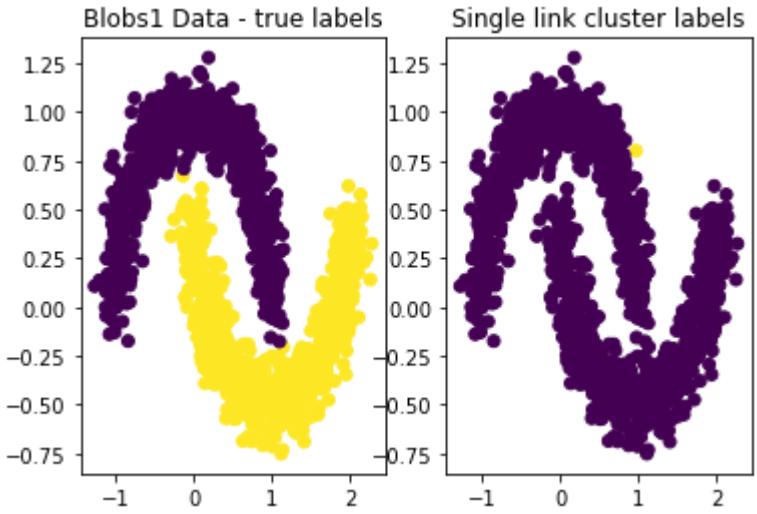
In [19]:
```
n_clusters = 3
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred = single_linkage.fit_predict(Blobs2_X)
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



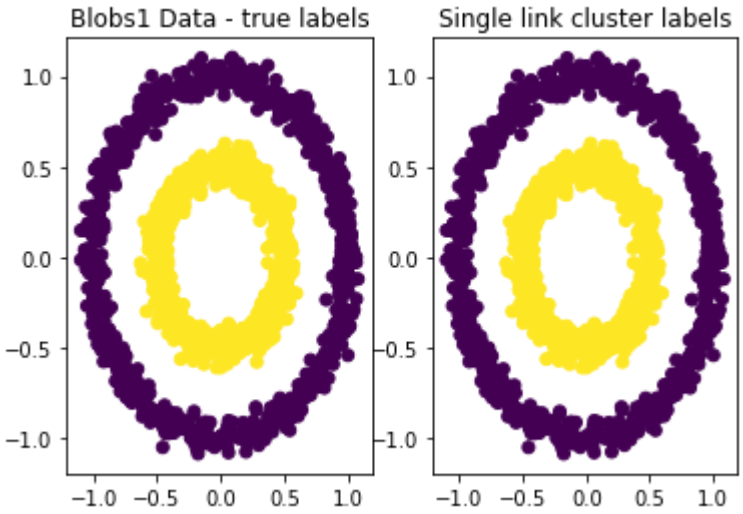
In [20]:
```
n_clusters = 2
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred = single_linkage.fit_predict(Moons1_X)
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



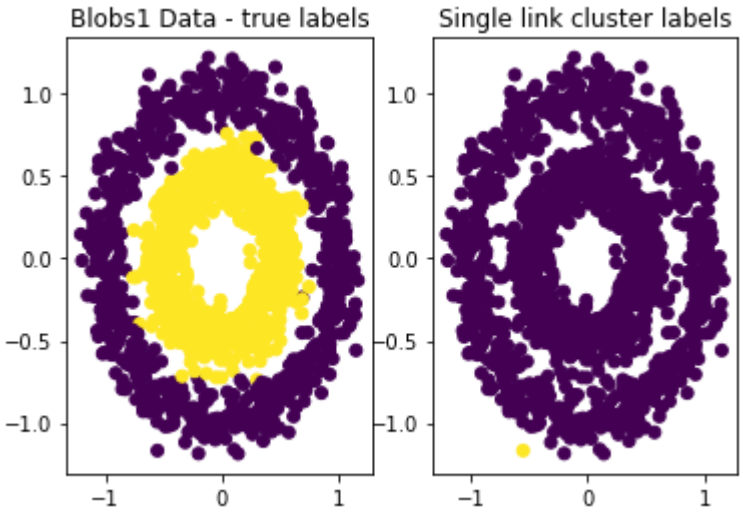
In [21]:
```
n_clusters = 2
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred = single_linkage.fit_predict(Moons2_X)
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



In [22]:
```
n_clusters = 2
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred = single_linkage.fit_predict(Circles1_X)
plt.subplot(1,2,1)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=Circles1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



In [23]:
```
n_clusters = 2
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred = single_linkage.fit_predict(Circles2_X)
plt.subplot(1,2,1)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=Circles2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



Question 2d: For each of the datasets, compute Rand-Index value between the true labels and cluster memberships computed using Single-link agglomerative clustering algorithm. Rank the datasets in decreasing order of Rand-Index scores.

Answer: Moons 1, Circles 1, Blobs 1, Moons 2, Circles 2, Blobs 2.

In [24]:
```
n_clusters = 3;
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred_single = single_linkage.fit_predict(Blobs1_X)
print(rand_index(y_pred_single, Blobs1_y))
```

```
0.99911140760507
```

In [25]:
```
n_clusters = 3;
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred_single = single_linkage.fit_predict(Blobs2_X)
print(rand_index(y_pred_single, Blobs2_y))
```

```
0.33377896375361354
```

In [26]:
```
n_clusters = 2;
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred_single = single_linkage.fit_predict(Moons1_X)
print(rand_index(y_pred_single, Moons1_y))
```

```
1.0
```

In [27]:
```
n_clusters = 2;
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred_single = single_linkage.fit_predict(Moons2_X)
print(rand_index(y_pred_single, Moons2_y))
```

```
0.49966733377807426
```

In [28]:
```
n_clusters = 2;
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred_single = single_linkage.fit_predict(Circles1_X)
print(rand_index(y_pred_single, Circles1_y))
```

```
1.0
```

In [29]:
```
n_clusters = 2;
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred_single = single_linkage.fit_predict(Circles2_X)
print(rand_index(y_pred_single, Circles2_y))
```

```
0.49966733377807426
```

Question 2e: Are the rankings in 2(c) consistent with your observations in 2(d)? If not, explain the reason why your rankings were inconsistent.

Answer: The ranking are mostly consistent, I did not point out the blobs 2 would perform worse than Moons 2 and Circles 2 because there are 3 clusters in blobs 2. The reason 3 clusters makes it worse is that the classification is uniform and when most of the points are in one cluster it only classifies with about 1/3 accuracy vs 1/2 with 2 clusters.

3. Agglomerative Clustering - Max Link

Question 3a: Without running Max-link agglomerative clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Max-link agglomerative clustering is expected to work well. Support your answer by explaining your rationale.

Answer: The only data set that I believe will perform well is the first data set. The reason I think the algorithm will perform well on this data set is, because it contains very distinct blob-like clusters with little noise.

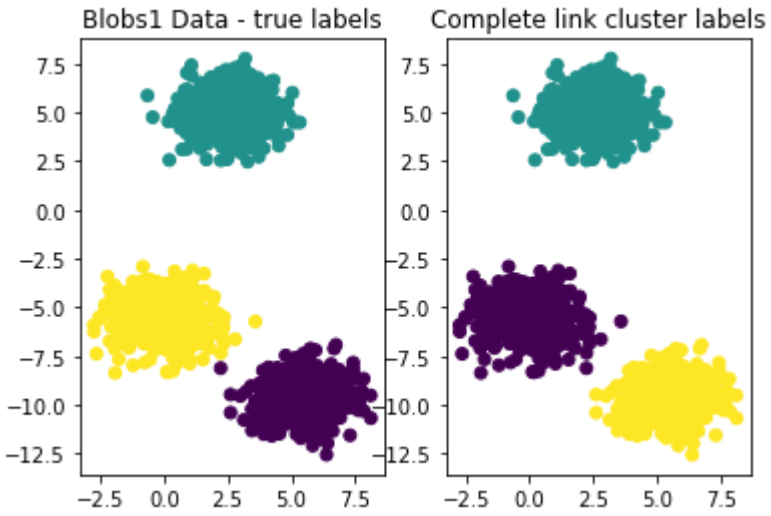
Question 3b: Without running Max-link agglomerative clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Max-link agglomerative clustering is expected to NOT work well. Support your answer by explaining your rationale.

Answer: I do not think this algorithm will work well on the other datasets because either the clusters are not distinct, or there are very irregular decision boundaries. The mechanix of max-link clustering are not conducive to picking out these clusters is, because it is not conducive to capturing irregular or not well separated clusters. Using max distance of points within clusters to measure similarity causes some clusters to bleed into others in ways it should not.

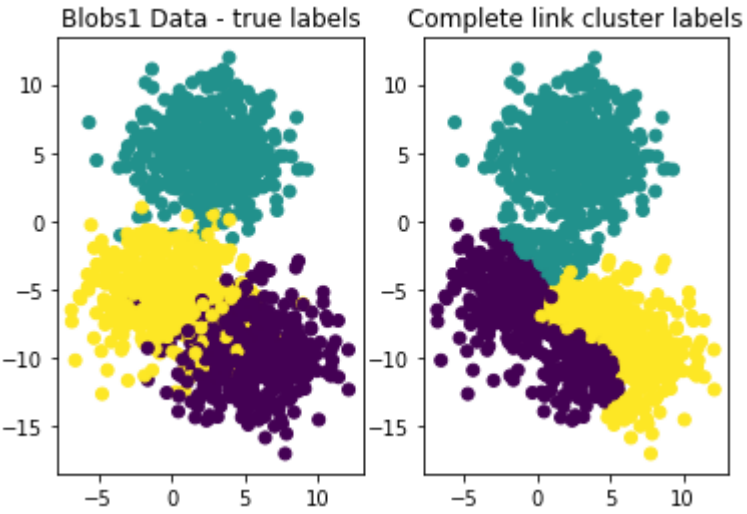
Question 3c: Run Max-link agglomerative clustering algorithm on all the datasets (except Rand). Choose n_clusters based on the number of clusters present in these datasets. Visualize the clusters for each of them. Based on the visualization, rank the datasets in decreasing order of Max-link agglomerative algorithm performance. Describe your rationale for your ranking.

Answer: Blobs 1, Blobs 2, Moons1, Moons2, Circle1, then Circle2. The order seems to be this way because blobs 1 nearly perfectly captures the boundaries as expected. Blobs 2 does worse as expected because the boundaries are not properly defined. The Moon data sets seem slightly worse, because the classification does not capture the irregular boundaries well. The circle data sets seem to perform the worst, because it cannot classifiy much better than a guess.

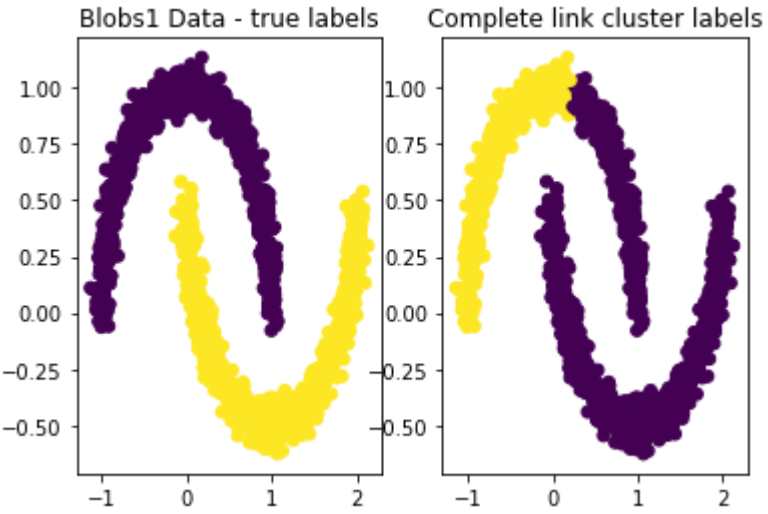
In [30]:
```
n_clusters = 3
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cl
usters)
y_pred = complete_linkage.fit_predict(Blobs1_X)
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



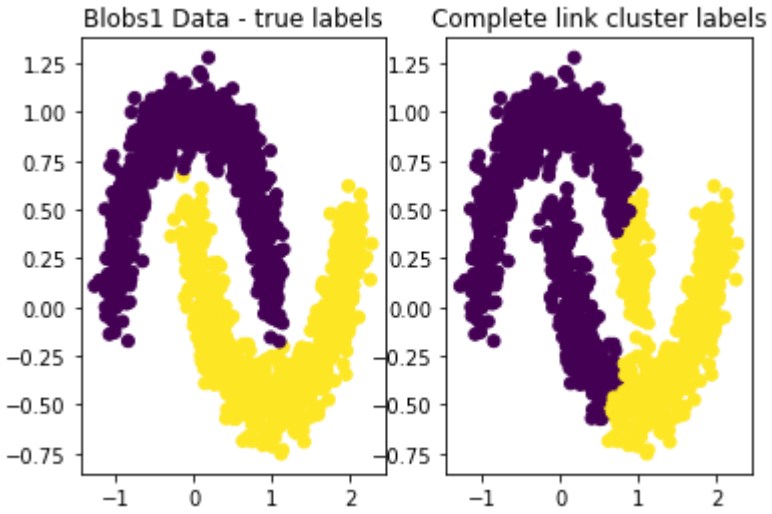
In [31]:
```
n_clusters = 3
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cl
usters)
y_pred = complete_linkage.fit_predict(Blobs2_X)
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



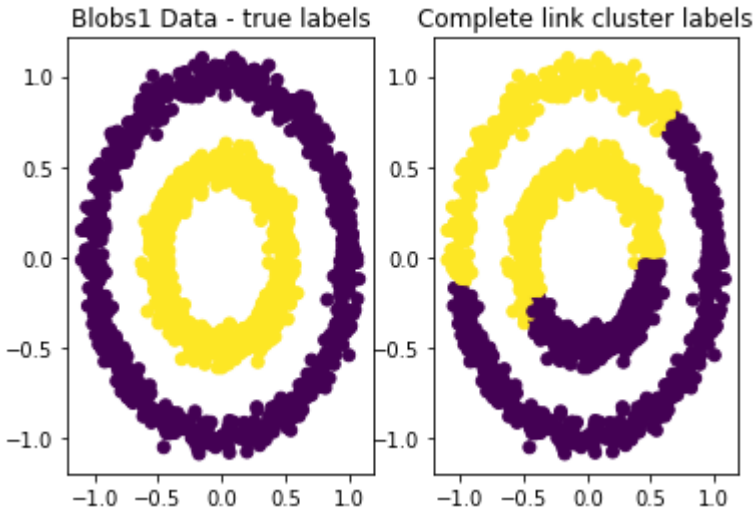
In [32]:
```
n_clusters = 2
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cl
usters)
y_pred = complete_linkage.fit_predict(Moons1_X)
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



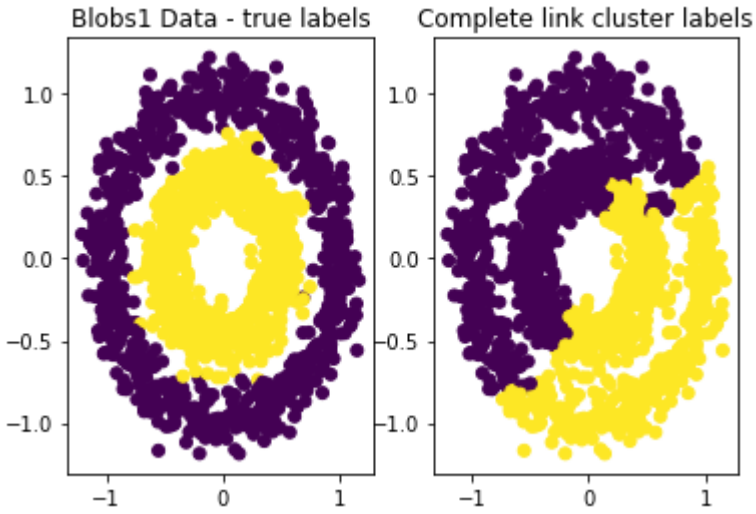
In [33]:
```
n_clusters = 2
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cl
usters)
y_pred = complete_linkage.fit_predict(Moons2_X)
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



In [34]:
```
n_clusters = 2
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cl
usters)
y_pred = complete_linkage.fit_predict(Circles1_X)
plt.subplot(1,2,1)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=Circles1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



In [35]:
```
n_clusters = 2
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cl
usters)
y_pred = complete_linkage.fit_predict(Circles2_X)
plt.subplot(1,2,1)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=Circles2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



Question 3d: For each of the datasets, compute Rand-Index value between the true labels and cluster memberships computed using Max-link agglomerative clustering algorithm. Rank the datasets in decreasing order of Rand-Index scores.

Answer: Blobs1, Blob2, Moon1, Moon2, Circle1, and then Circle2.

In [36]:
```
n_clusters = 3;
avg_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cluster
s)
y_pred_avg = avg_linkage.fit_predict(Blobs1_X)
print(rand_index(y_pred_avg, Blobs1_y))
```

```
0.99911140760507
```

In [37]:
```
n_clusters = 3;
avg_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cluster
s)
y_pred_complete = complete_linkage.fit_predict(Blobs2_X)
print(rand_index(y_pred_complete, Blobs2_y))
```

```
0.7490140093395597
```

In [38]:
```
n_clusters = 2;
avg_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cluster
s)
y_pred_complete = complete_linkage.fit_predict(Moons1_X)
print(rand_index(y_pred_complete, Moons1_y))
```

```
0.662605292417167
```

In [39]:
```
n_clusters = 2;
avg_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cluster
s)
y_pred_complete = complete_linkage.fit_predict(Moons2_X)
print(rand_index(y_pred_complete, Moons2_y))
```

```
0.5965310206804536
```

In [40]:
```
n_clusters = 2;
avg_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cluster
s)
y_pred_complete = complete_linkage.fit_predict(Circles1_X)
print(rand_index(y_pred_complete, Circles1_y))
```

```
0.5218714698688014
```

In [41]:
```
n_clusters = 2;
avg_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cluster
s)
y_pred_complete = complete_linkage.fit_predict(Circles2_X)
print(rand_index(y_pred_complete, Circles2_y))
```

```
0.5000587058038692
```

Question 3e: Are the rankings in 3(c) consistent with your observations in 3(d)? If not, explain the reason why your rankings were inconsistent.

```
<span style="color:green">**Answer:** </span> This was consistent with my observati
ons. `
```

4. Agglomerative Clustering - Average Link

Question 4a: Without running Average-link agglomerative clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Average-link agglomerative clustering is expected to work well. Support your answer by explaining your rationale.

Answer: I think the again the only data set that will perform well is blobs 1 for the same reason as max distance clustering.

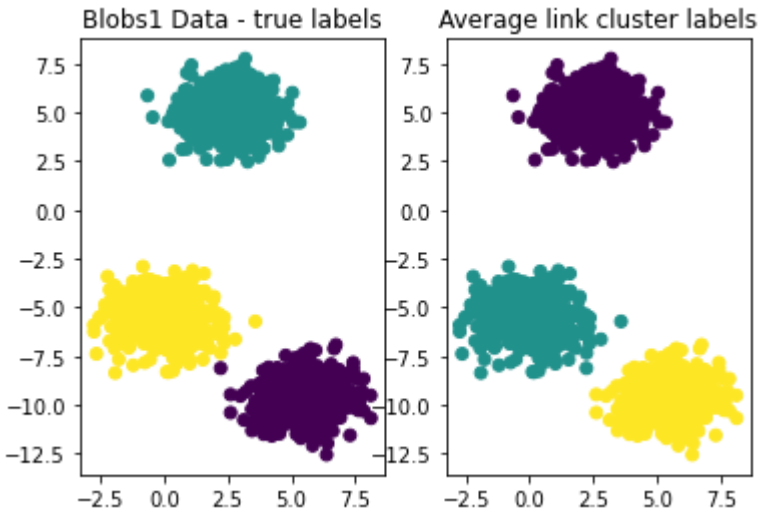
Question 4b: Without running Average-link agglomerative clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Average-link agglomerative clustering is expected to NOT work well. Support your answer by explaining your rationale.

Answer: Again, for the same reason as I put in max link, average link clustering should not perform well on the other data sets.

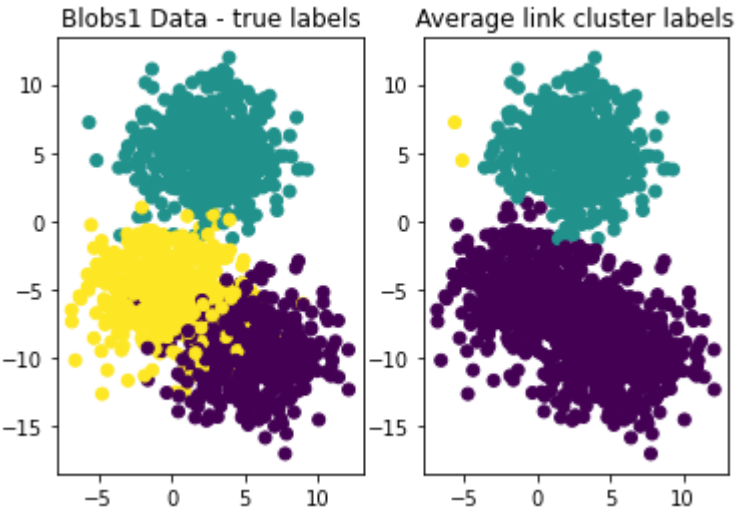
Question 4c: Run Average-link agglomerative clustering algorithm on all the datasets (except Rand). Choose n_clusters based on the number of clusters present in these datasets. Visualize the clusters for each of them. Based on the visualization, rank the datasets in decreasing order of Average-link agglomerative algorithm performance. Describe your rationale for your ranking.

Answer: From best to worst is Blobs 1 and then blobs 2. The moons data sets look the next best, but it is not easy to tell visually if one performed better. Finally the circles data sets.

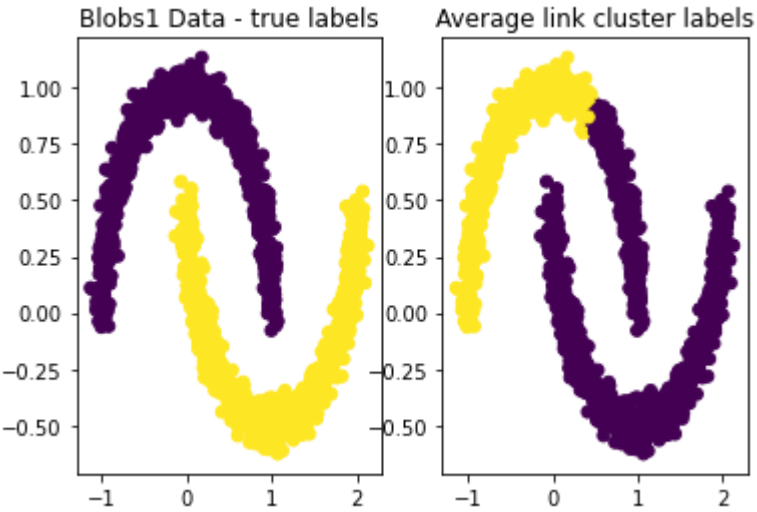
In [42]:
```
n_clusters = 3
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred = avg_linkage.fit_predict(Blobs1_X)
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)
plt.title('Average link cluster labels')
plt.show()
```



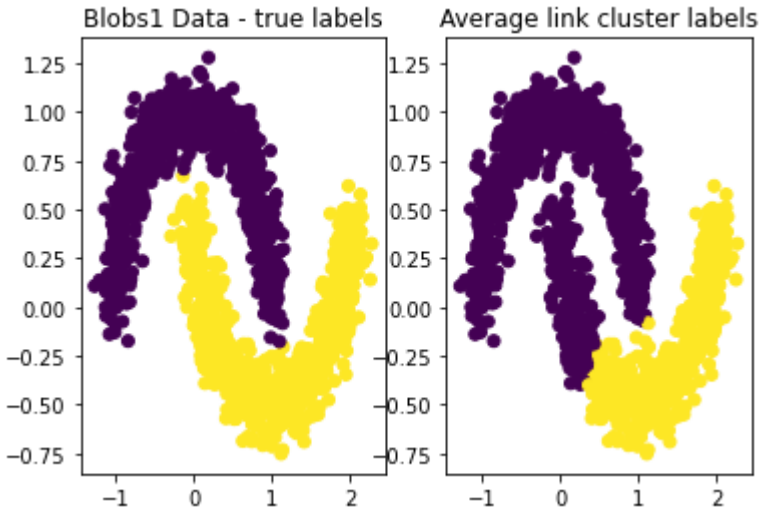
In [43]:
```
n_clusters = 3
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred = avg_linkage.fit_predict(Blobs2_X)
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)
plt.title('Average link cluster labels')
plt.show()
```



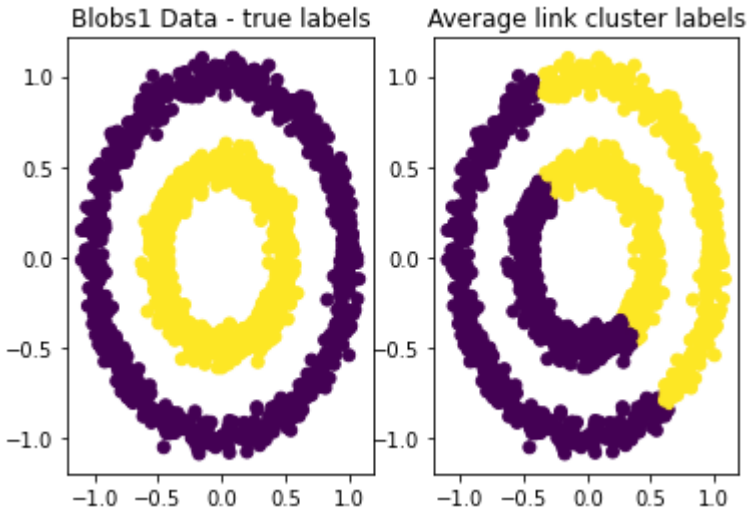
In [44]:
```
n_clusters = 2
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred = avg_linkage.fit_predict(Moons1_X)
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=y_pred)
plt.title('Average link cluster labels')
plt.show()
```



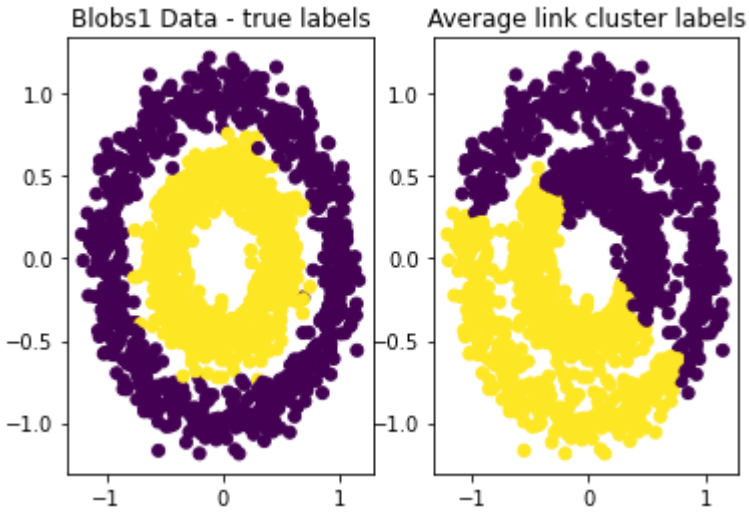
In [45]:
```
n_clusters = 2
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred = avg_linkage.fit_predict(Moons2_X)
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=y_pred)
plt.title('Average link cluster labels')
plt.show()
```



In [46]:
```
n_clusters = 2
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred = avg_linkage.fit_predict(Circles1_X)
plt.subplot(1,2,1)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=Circles1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=y_pred)
plt.title('Average link cluster labels')
plt.show()
```



In [47]:
```
n_clusters = 2
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred = avg_linkage.fit_predict(Circles2_X)
plt.subplot(1,2,1)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=Circles2_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=y_pred)
plt.title('Average link cluster labels')
plt.show()
```



Question 4d: For each of the datasets, compute Rand-Index value between the true labels and cluster memberships computed using Average-link agglomerative clustering algorithm. Rank the datasets in decreasing order of Rand-Index scores.

Answer: From best to worst was blobs 1, blobs 2, Moons 2, Moons 1, circles 2, then circles 1.

In [48]:
```
n_clusters = 3;
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred_avg = avg_linkage.fit_predict(Blobs1_X)
print(rand_index(y_pred_avg, Blobs1_y))
```

```
0.99911140760507
```

In [49]:
```
n_clusters = 3;
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred_avg = avg_linkage.fit_predict(Blobs2_X)
print(rand_index(y_pred_avg, Blobs2_y))
```

```
0.7636575494774294
```

In [50]:
```
n_clusters = 2;
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred_avg = avg_linkage.fit_predict(Moons1_X)
print(rand_index(y_pred_avg, Moons1_y))
```

```
0.7132310429175005
```

In [51]:
```
n_clusters = 2;
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred_avg = avg_linkage.fit_predict(Moons2_X)
print(rand_index(y_pred_avg, Moons2_y))
```

```
0.7457647320435846
```

In [52]:
```
n_clusters = 2;
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred_avg = avg_linkage.fit_predict(Circles1_X)
print(rand_index(y_pred_avg, Circles1_y))
```

```
0.500414498554592
```

In [53]:
```
n_clusters = 2;
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred_avg = avg_linkage.fit_predict(Circles2_X)
print(rand_index(y_pred_avg, Circles2_y))
```

```
0.5050780520346898
```

Question 4e: Are the rankings in 4(c) consistent with your observations in 4(d)? If not, explain the reason why your rankings were inconsistent.

Answer: The rankings were consistent with my observations except there was a noticeable difference between moons 1 and moons 2. Moons 2 might have performed better because there was less separation in the data due to noise.

5. Density Based Clustering: DBSCAN

Question 5a: Without running DBSCAN clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where DBSCAN clustering is expected to work well. Support your answer by explaining your rationale.

Answer:

I expect density based clustering to perform well on every data set except blobs 2. The reason I think this is the case is because in all the other data sets there are clusters which are clearly defined by the density. In these clusters you can choose a density level that should allow you to capture the cluster that are necessary and not have core point take in in the wrong cluster.

Question 5b: Without running DBSCAN clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where DBSCAN clustering is expected to NOT work well. Support your answer by explaining your rationale.

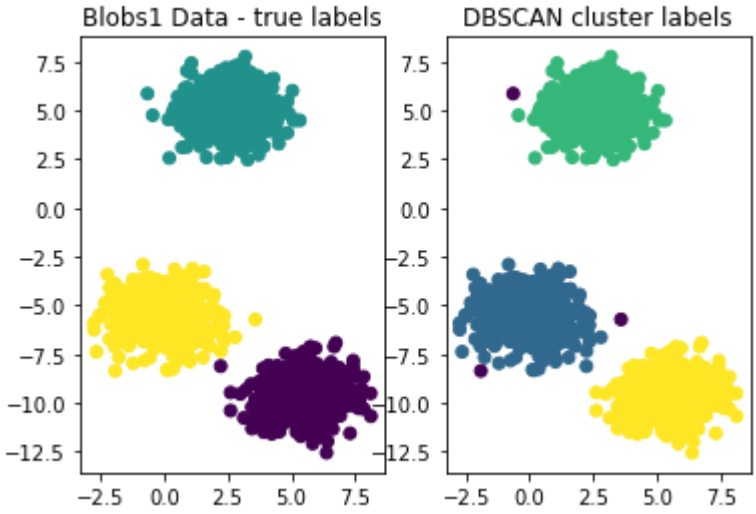
Answer:

I do not think dbscan will perform well on Blobs 2 because there is significant overlap with the clusters and it will probably cause some of the clusters to expand into unwanted territory.

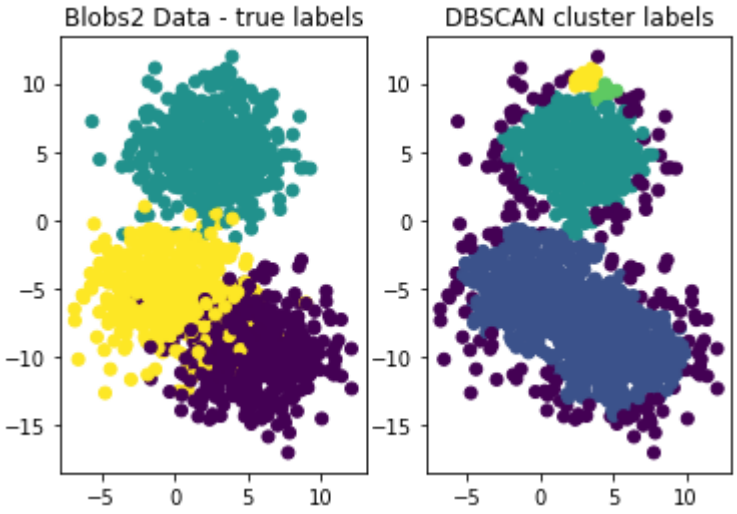
Question 5c: Run DBSCAN clustering algorithm on all the datasets (except Rand). **Choose eps and min_samples parameters to make sure that DBSCAN finds the same number of clusters as in the ground truth ('Data_y').** Visualize the clusters for each of them. Based on the visualization, rank the datasets in decreasing order of DBSCAN clustering algorithm performance. Describe your rationale for your ranking.

Answer: Blobs 1, Moon 1, Circle 1, Moons 2, Circle 2, then Blobs 2. The 1 data sets performed best because they had uniform density and well separated clusters. The Moons 2 and circle 2 datasets performed slightly less well, because it classified some points as noise, and blobs 2 was worst for the point I made early. In blobs 2, the clusters had significant overlap.

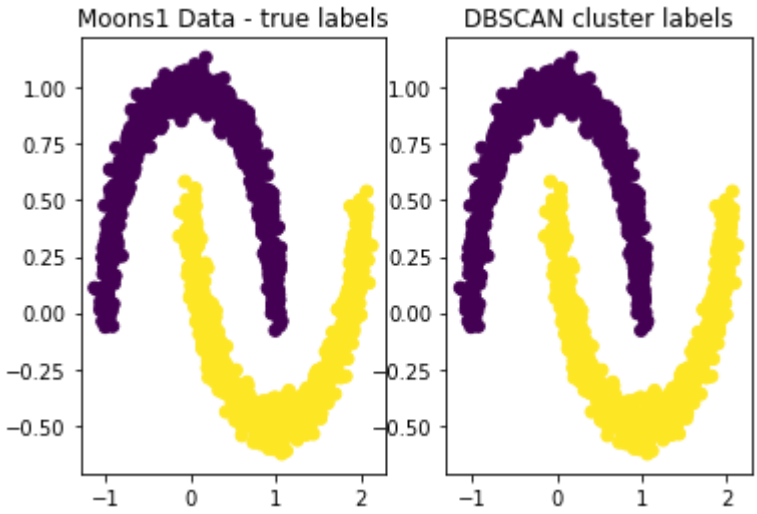
In [54]:
```
dbscan = DBSCAN(eps=1, min_samples=10)
y_pred = dbscan.fit_predict(Blobs1_X)
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)
plt.title('DBSCAN cluster labels')
plt.show()
```



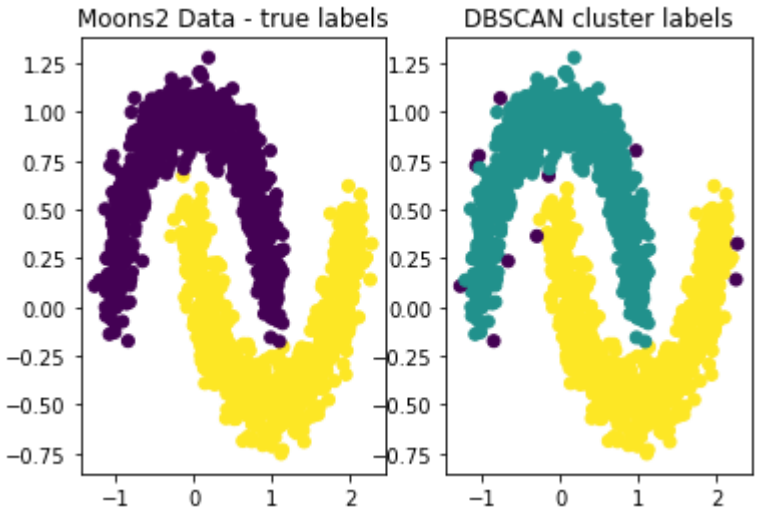
In [55]:
```
dbscan = DBSCAN(eps=.9, min_samples=10)
y_pred = dbscan.fit_predict(Blobs2_X)
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y)
plt.title('Blobs2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)
plt.title('DBSCAN cluster labels')
plt.show()
```



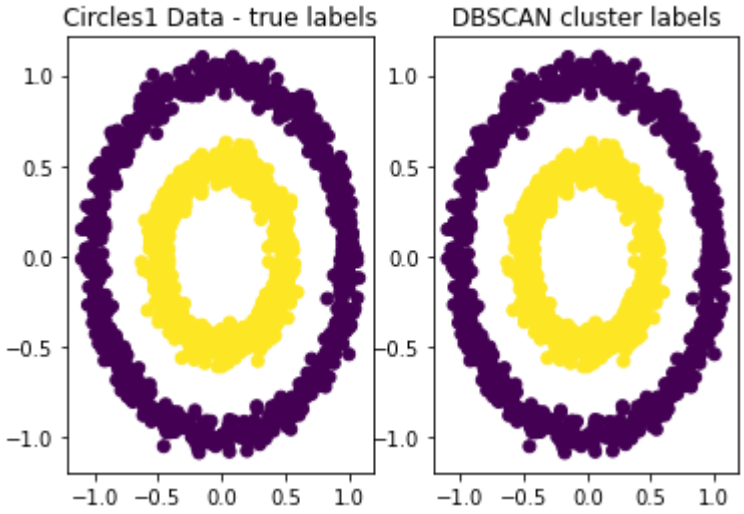
In [56]:
```
dbscan = DBSCAN(eps=.2, min_samples=10)
y_pred = dbscan.fit_predict(Moons1_X)
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y)
plt.title('Moons1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=y_pred)
plt.title('DBSCAN cluster labels')
plt.show()
```



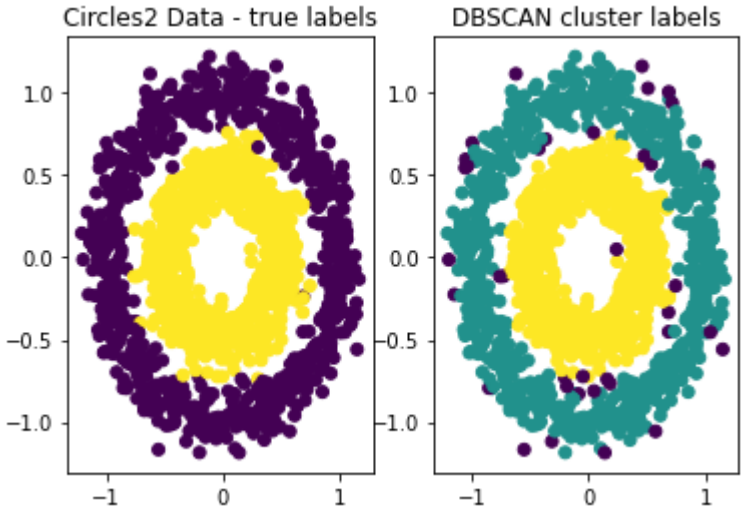
In [57]:
```
dbscan = DBSCAN(eps=.13, min_samples=10)
y_pred = dbscan.fit_predict(Moons2_X)
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y)
plt.title('Moons2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=y_pred)
plt.title('DBSCAN cluster labels')
plt.show()
```



In [58]:
```
dbscan = DBSCAN(eps=.15, min_samples=10)
y_pred = dbscan.fit_predict(Circles1_X)
plt.subplot(1,2,1)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=Circles1_y)
plt.title('Circles1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=y_pred)
plt.title('DBSCAN cluster labels')
plt.show()
```



In [59]:
```
dbscan = DBSCAN(eps=.11, min_samples=10)
y_pred = dbscan.fit_predict(Circles2_X)
plt.subplot(1,2,1)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=Circles2_y)
plt.title('Circles2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=y_pred)
plt.title('DBSCAN cluster labels')
plt.show()
```



Question 5d: For each of the datasets, how many noise points did the DBSCAN algorithm find? Which three datasets had the least number of noise points? Explain the reason(s) why these datasets had least noise points?

Answer: DBscan found no noise points for all the 1 datasets except a 2 in blobs 1. Of the 2, datsets Moons 2 had the least noise with finding about 10 noise points, because there was less blending of the boundaries of class which allowed us to pick a higher epsilon than circles 2 which found about 20. Blobs 2 had the most noise because there was a lot of blending and we had to choose a high epsilon to do any meaningful classification. There was at least 40 noise points. The 1 data sets did not have much noise because there was enough separation between the clusters so that we could choose a high epsilon value to not allow any noise.

Question 5e: For each of the datasets, compute Rand-Index value between the true labels and cluster memberships computed using DBSCAN clustering algorithm. Rank the datasets in decreasing order of Rand-Index scores.

Answer:

From best to worst it was Moons 1, Circles 1, Blobs 1, Moons 2, Circles 2, then Blobs 2

In [60]:
```
dbscan = DBSCAN(eps=1, min_samples=10)
y_pred = dbscan.fit_predict(Blobs1_X)
for i in range(0,len(y_pred)):
    if y_pred[i] == -1:
        y_pred[i] = 2
print(rand_index(y_pred, Blobs1_y))
```

```
0.9964500778296642
```

In [ ]:
```

```

In [61]:
```
dbscan = DBSCAN(eps=.9, min_samples=10)
y_pred = dbscan.fit_predict(Blobs2_X)
for i in range(0,len(y_pred)):
    if y_pred[i] == -1:
        y_pred[i] = 2
print(rand_index(y_pred, Blobs2_y))
```

```
0.7345581498776962
```

In [62]:
```
dbscan = DBSCAN(eps=.2, min_samples=10)
y_pred = dbscan.fit_predict(Moons1_X)
for i in range(0,len(y_pred)):
    if y_pred[i] == -1:
        y_pred[i] = 2
print(rand_index(y_pred, Moons1_y))
```

```
1.0
```

In [63]:
```
dbscan = DBSCAN(eps=.12, min_samples=10)
y_pred = dbscan.fit_predict(Moons2_X)
for i in range(0,len(y_pred)):
    if y_pred[i] == -1:
        y_pred[i] = 2
print(rand_index(y_pred, Moons2_y))
```

```
0.9907066933511229
```

In [64]:
```
dbscan = DBSCAN(eps=.15, min_samples=10)
y_pred = dbscan.fit_predict(Circles1_X)
print(rand_index(y_pred, Circles1_y))
```

```
1.0
```

In [65]:
```
dbscan = DBSCAN(eps=.11, min_samples=10)
y_pred = dbscan.fit_predict(Circles2_X)
for i in range(0,len(y_pred)):
    if y_pred[i] == -1:
        y_pred[i] = 2
print(rand_index(y_pred, Circles2_y))
```

```
0.9643344451856793
```

Question 5f: Are the rankings in 5(c) consistent with your observations in 5(e)? If not, explain the reason why your rankings were inconsistent.

Answer: These were consistent with my observations

6. Spectral Clustering

Question 6a: Without running Spectral clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Spectral clustering is expected to work well. Support your answer by explaining your rationale.

Answer: The data sets I expect spectral clustering to work well on are blobs1 and blobs 2. The reason I think it will work well on this dataset is because the density of these clusters are regular and blob like.

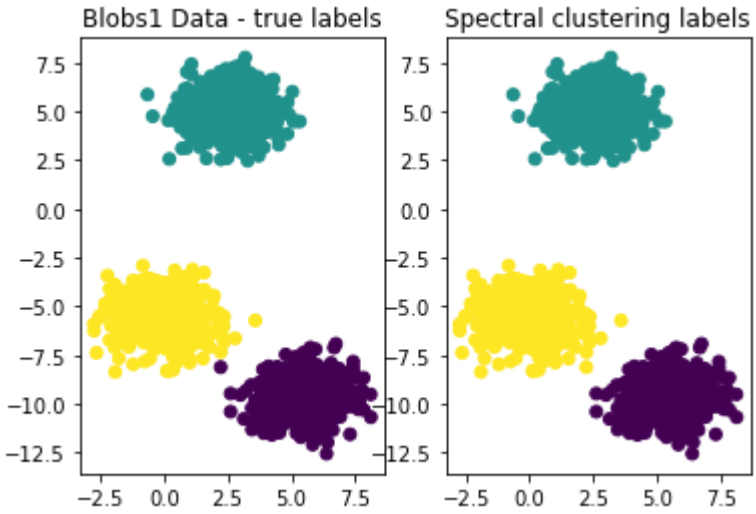
Question 6b: Without running Spectral clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Spectral clustering is expected to NOT work well. Support your answer by explaining your rationale.

Answer: Spectral clustering will not work well on the Moons or circle data set. This is because the cluster boundaries are irregular. The algorithm captures these clusters based on the most prominant aspects of the similarity matrix using k means

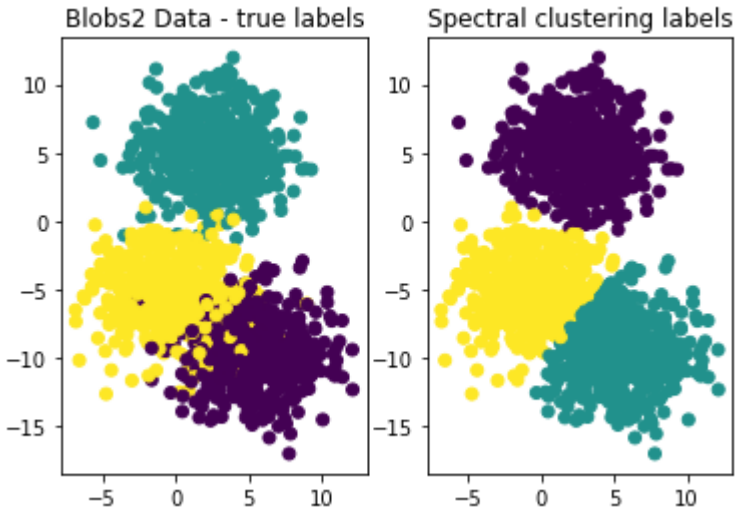
Question 6c: Run Spectral clustering algorithm on all the datasets (except Rand). Choose n_clusters based on the number of clusters present in these datasets. Visualize the clusters for each of them. Based on the visualization, rank the datasets in decreasing order of Spectral clustering algorithm performance. Describe your rationale for your ranking.

Answer: From best to worst looks like blobs 1, blobs 2, moons 1 and 2, and then the circles data sets. This is so because it uses a kmeans algorithm for clustering which works best on well separated blob-like data sets.

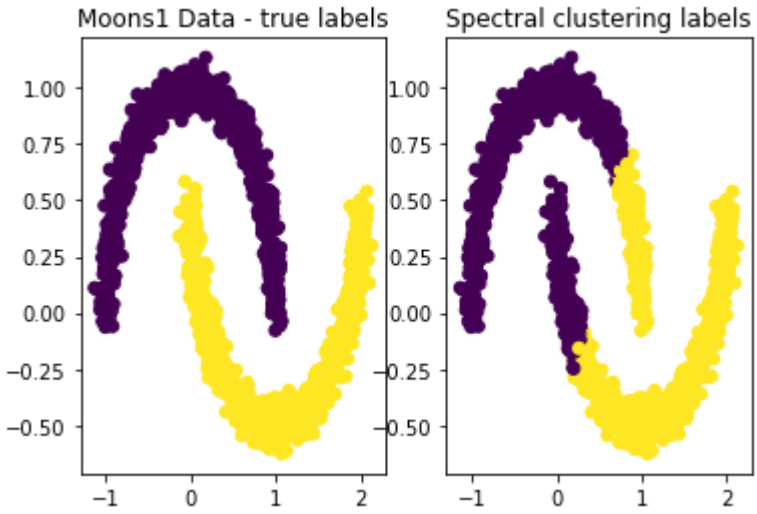
In [66]:
```
n_clusters = 3
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Blobs1_X)
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)
plt.title('Spectral clustering labels')
plt.show()
```



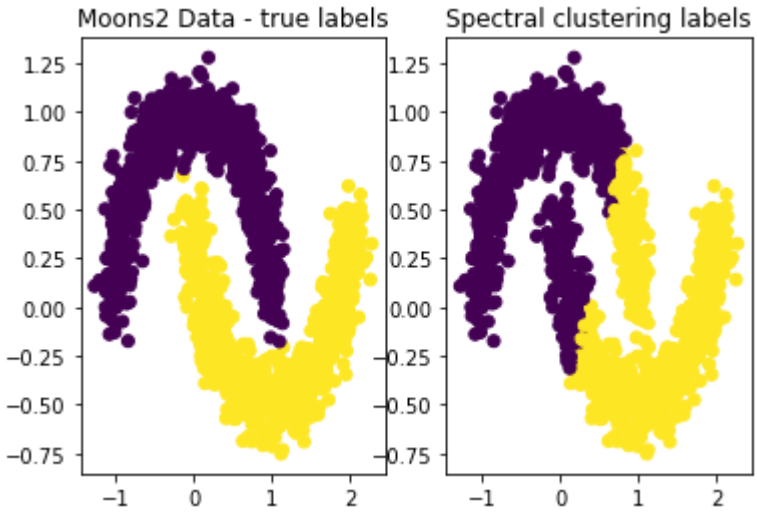
In [67]:
```
n_clusters = 3
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Blobs2_X)
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y)
plt.title('Blobs2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)
plt.title('Spectral clustering labels')
plt.show()
```



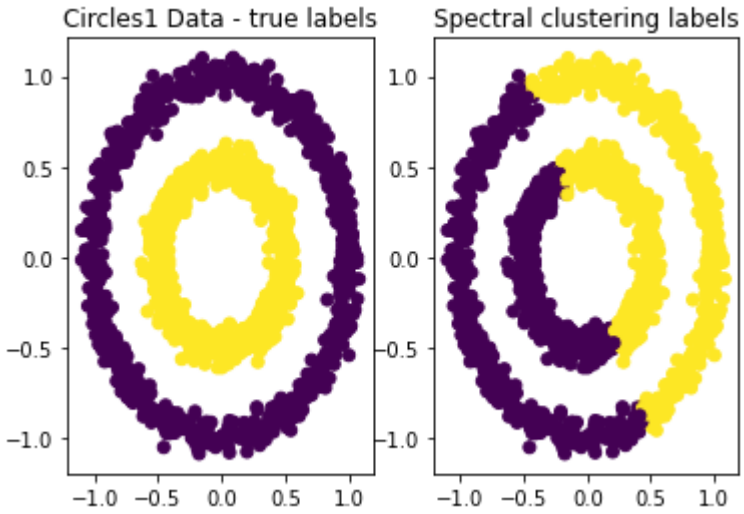
In [68]:
```
n_clusters = 2
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Moons1_X)
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y)
plt.title('Moons1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=y_pred)
plt.title('Spectral clustering labels')
plt.show()
```



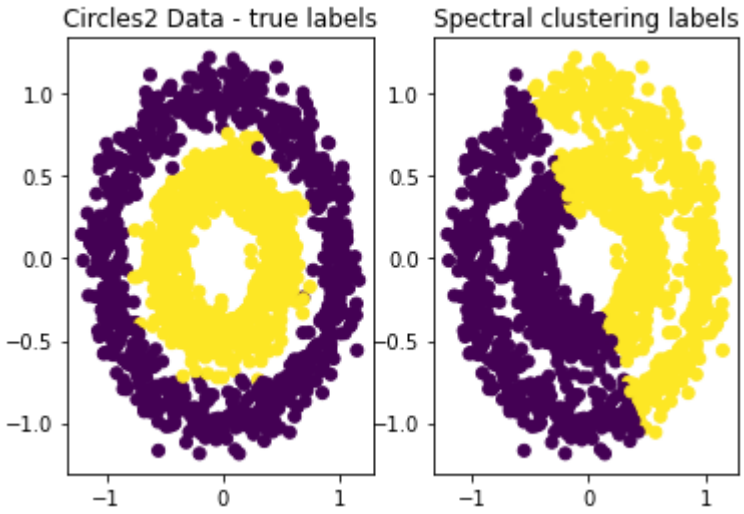
In [69]:
```
n_clusters = 2
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Moons2_X)
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y)
plt.title('Moons2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=y_pred)
plt.title('Spectral clustering labels')
plt.show()
```



In [70]:
```
n_clusters = 2
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Circles1_X)
plt.subplot(1,2,1)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=Circles1_y)
plt.title('Circles1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=y_pred)
plt.title('Spectral clustering labels')
plt.show()
```



In [71]:
```
n_clusters = 2
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Circles2_X)
plt.subplot(1,2,1)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=Circles2_y)
plt.title('Circles2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=y_pred)
plt.title('Spectral clustering labels')
plt.show()
```



Question 6d: For each of the datasets, compute Rand-Index value between the true labels and cluster memberships computed using Spectral clustering algorithm. Rank the datasets in decreasing order of Rand-Index scores.

Answer: Blobs 1, blobs 2, moons 1 and 2, then circles 1 and 2.

In [72]:
```
n_clusters = 3
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Blobs1_X)
print(rand_index(y_pred, Blobs1_y))
```

```
0.99911140760507
```

In [73]:
```
n_clusters = 3
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Blobs2_X)
print(rand_index(y_pred, Blobs2_y))
```

```
0.919189682010229
```

In [74]:
```
n_clusters = 2
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Moons1_X)
print(rand_index(y_pred, Moons1_y))
```

```
0.6434102735156771
```

In [75]:
```
n_clusters = 2
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Moons2_X)
print(rand_index(y_pred, Moons2_y))
```

```
0.6441263064265066
```

In [76]:
```
n_clusters = 2
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Circles1_X)
print(rand_index(y_pred, Circles1_y))
```

```
0.49966644429619744
```

In [77]:
```
n_clusters = 2
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Circles2_X)
print(rand_index(y_pred, Circles2_y))
```

```
0.49977407160329107
```

Question 6e: Are the rankings in 6(c) consistent with your observations in 6(d)? If not, explain the reason why your rankings were inconsistent.

Answer: They were consistent with my observations

7. Clustering Tendency

Question 7a: Without using any metrics, for all the datasets (INCLUDING **Rand**) provided in the practice session, list the datasets that exhibit good clustering tendency. Support your answer by explaining your rationale.

Answer: All of the data sets excluding Rand exhibit good clustering tendency. This is because of my intuitation about how the hopkins statistic works. Some will definiely be better than others and the data sets I expect to have better clustering tendency will be the datasets with less noise and tighter clusters. This is because when the points are close together an have less noise every point should

Question 7b: Without using any metrics, for all the datasets (INCLUDING Rand) provided in the practice session, list the datasets that do NOT exhibit good clustering tendency. Support your answer by explaining your rationale.

Answer: The Rand dataset will be the only dataset with poor clustering tendency, because the points in the data set are random. The distances of the nearest neighbor for the newly randomly generated data set and the Rand dataset should sum up to be about equal. The Hopkins statistic should then be close to 1/2 which indicates poor clustering tendency.

Question 7c: Compute Hopkins Statistic statistic for all the datasets and rank them based on decreasing order of this metric.

Answer: Moons 1, Blobs 1, Circles 1, Moons 2, Blobs 2, and Circles 2

In [78]:
```
hopkins(Blobs1_X)
```

Out[78]:
```
0.9349630424586175
```

```
In [79]: hopkins(Blobs2_X)
Out[79]: 0.8327902200872102

In [80]: hopkins(Moons1_X)
Out[80]: 0.920326419276779

In [81]: hopkins(Moons2_X)
Out[81]: 0.8809426957438586

In [82]: hopkins(Circles1_X)
Out[82]: 0.8478040068749064

In [83]: hopkins(Circles2_X)
Out[83]: 0.7572608314092971

In [84]: hopkins(Rand_X)
Out[84]: 0.5985300737320572
```

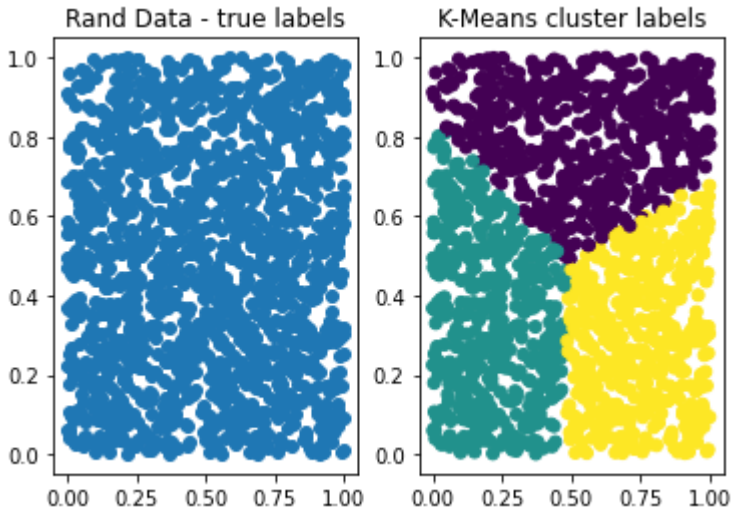
Question 7d: Are your answers for 7(a) and 7(b) consistent with that of (c)? If not, explain the reason for this inconsistency.

Answer: These were consistent. The hopkins statistic for Rand, however, was farther from 1/2 than I was expecting.

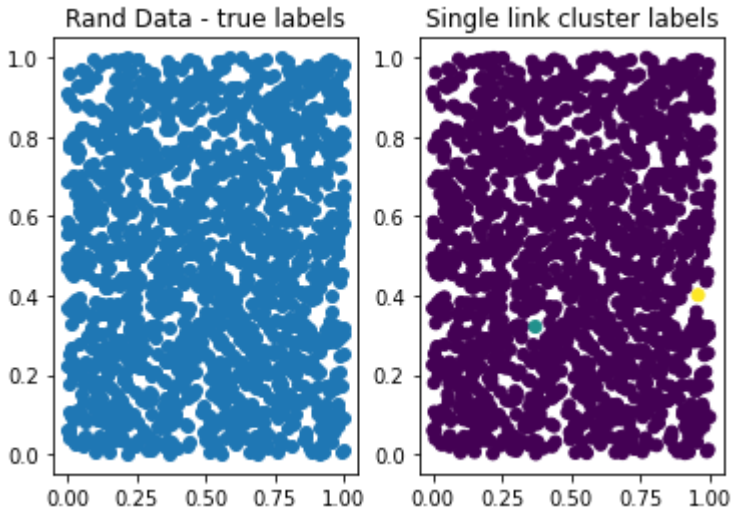
Question 7e: Run all the above clustering algorithms (KMeans, GMM, Agglomerative (single, max, average), DBSCAN, Spectral), using n_clusters = 3, on Rand dataset and visualize the clusters. Explain the reason for the shapes of clusters dervied using each clustering approach.

Answer: Kmeans and spectral clustering partition the data set into 3 equal spaces, because it reduces the SSE the most to have the data partitioned in such a way. If it was not so, the SSE would be much higher. Single link clustering makes one large cluster and 2 single point clusters because all the points are so close that by the time the agglomeration stops with the first cluster, it captures all other points. Complete link and average link clustering partition in the way they do because they are not as nearsighted as single link clustering. These take into account the distances of farther away points which gives the other cluster a chance to have the closest average or maximum distances. This allows these alorithms to form clusters that balance each other out. DBscan will put everything in one cluster, because you do not specify the number of clusters before hand, and there is uniform density.

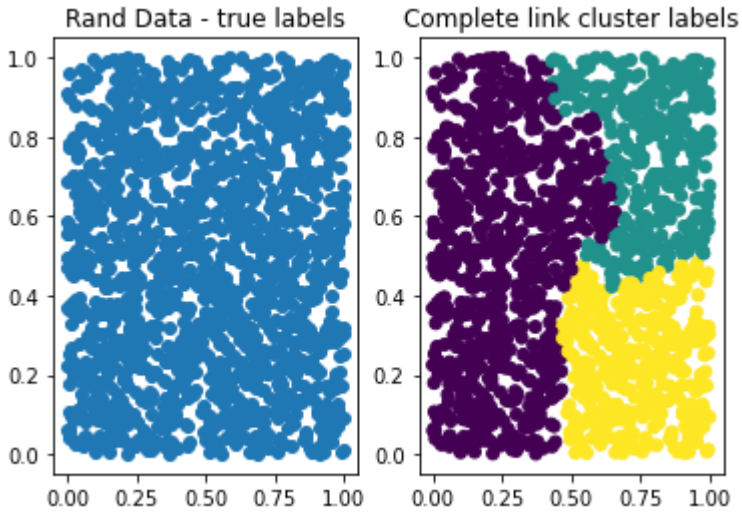
In [85]:
```
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred = kmeans.fit_predict(Rand_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1]) # true clusters
plt.title('Rand Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



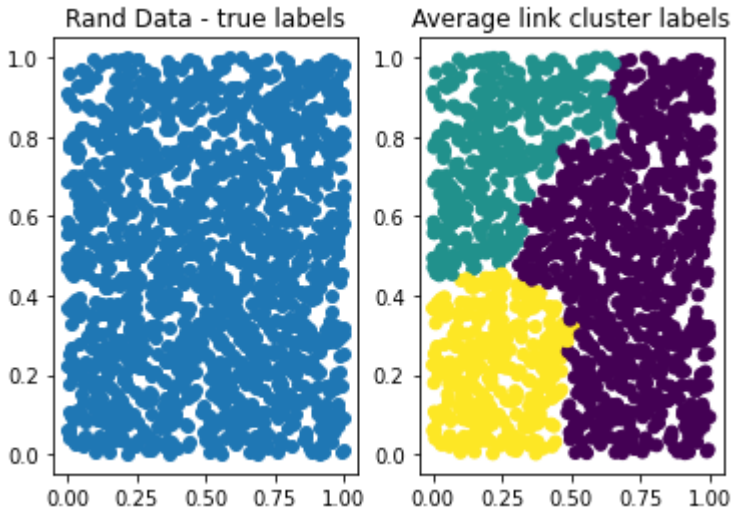
In [86]:
```
n_clusters = 3
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_cluste
rs)
y_pred = single_linkage.fit_predict(Rand_X)
plt.subplot(1,2,1)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1])
plt.title('Rand Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



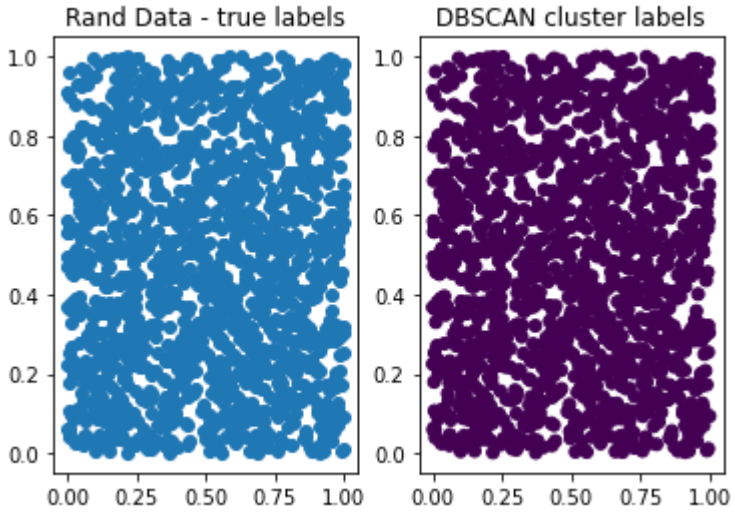
In [87]:
```
n_clusters = 3
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters=n_cl
usters)
y_pred = complete_linkage.fit_predict(Rand_X)
plt.subplot(1,2,1)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1])
plt.title('Rand Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



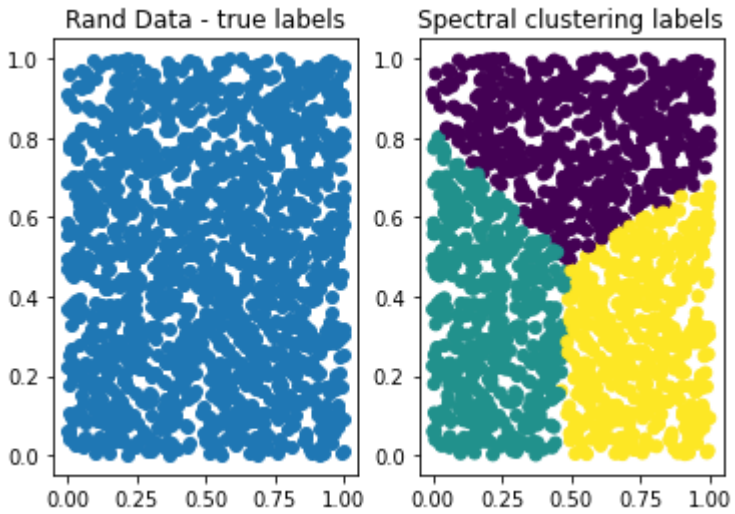
In [88]:
```
n_clusters = 3
avg_linkage = AgglomerativeClustering(linkage="average", n_clusters=n_clusters
)
y_pred = avg_linkage.fit_predict(Rand_X)
plt.subplot(1,2,1)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1])
plt.title('Rand Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1], c=y_pred)
plt.title('Average link cluster labels')
plt.show()
```



In [89]:
```
dbscan = DBSCAN(eps=1, min_samples=10)
y_pred = dbscan.fit_predict(Rand_X)
plt.subplot(1,2,1)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1])
plt.title('Rand Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1], c=y_pred)
plt.title('DBSCAN cluster labels')
plt.show()
```



In [90]:
```
n_clusters = 3
spectral = SpectralClustering(n_clusters=n_clusters, random_state=random_state
)
y_pred = spectral.fit_predict(Rand_X)
plt.subplot(1,2,1)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1])
plt.title('Rand Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Rand_X[:, 0], Rand_X[:, 1], c=y_pred)
plt.title('Spectral clustering labels')
plt.show()
```



8. Real-world dataset

We will use the same breast cancer dataset we used for Classificatione exercise here.

In [91]:
```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
```

The features are:

In [92]:
```
cancer.feature_names
```

Out[92]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Class labels are:

In [93]:
```
cancer.target_names
```

Out[93]:
```
array(['malignant', 'benign'], dtype='<U9')
```

Create dataset for classification

In [94]:
```
Cancer_X = cancer.data
Cancer_y = cancer.target
```

Size of Cancer_X and Cancer_y

In [95]:
```
Cancer_X.shape
```

Out[95]:
```
(569, 30)
```

In [96]:
```
Cancer_y.shape
```

Out[96]:
```
(569,)
```

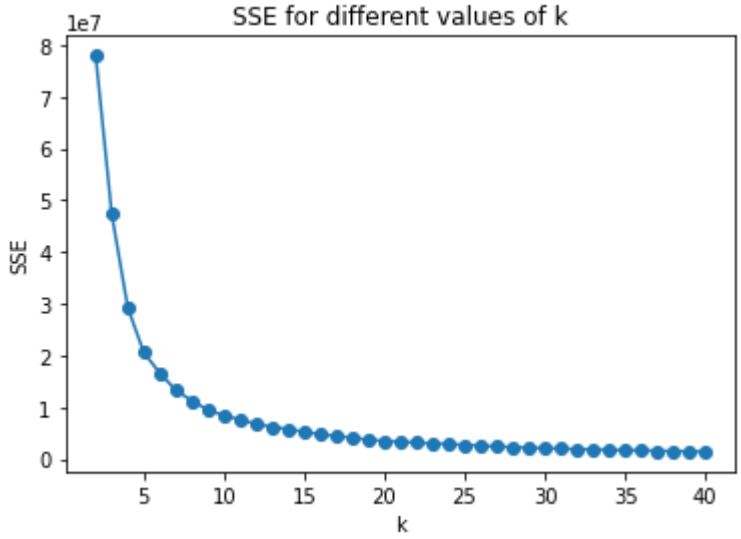
Question 8a: Compute SSE for k = range(2,40), i.e, for k=2,3,4,...,40

In [97]:
```
score = np.zeros(41);
for i in range(2,41):
    kmeans = KMeans(n_clusters=i, random_state=random_state); #Initializing KM
eans for different n_clusters
    kmeans.fit_predict(Cancer_X)  #Clustering using KMeans
    score[i] = -kmeans.score(Cancer_X)  #Computing SSE
    print("SSE for k=",i,":", round(score[i],2)) #Printing SSE
```

```
SSE for k= 2 : 77943099.88
SSE for k= 3 : 47285926.9
SSE for k= 4 : 29226541.65
SSE for k= 5 : 20535235.91
SSE for k= 6 : 16575383.9
SSE for k= 7 : 13252389.84
SSE for k= 8 : 11231884.68
SSE for k= 9 : 9466539.93
SSE for k= 10 : 8389832.42
SSE for k= 11 : 7589081.6
SSE for k= 12 : 6744629.25
SSE for k= 13 : 6183686.07
SSE for k= 14 : 5713696.6
SSE for k= 15 : 5239506.34
SSE for k= 16 : 4753962.04
SSE for k= 17 : 4413637.14
SSE for k= 18 : 4062241.36
SSE for k= 19 : 3691701.37
SSE for k= 20 : 3445817.04
SSE for k= 21 : 3277343.7
SSE for k= 22 : 3106236.47
SSE for k= 23 : 2979532.27
SSE for k= 24 : 2821492.47
SSE for k= 25 : 2678897.75
SSE for k= 26 : 2543910.27
SSE for k= 27 : 2353377.89
SSE for k= 28 : 2281031.53
SSE for k= 29 : 2124248.88
SSE for k= 30 : 2039623.96
SSE for k= 31 : 1984886.65
SSE for k= 32 : 1865795.18
SSE for k= 33 : 1790316.23
SSE for k= 34 : 1725446.6
SSE for k= 35 : 1649402.36
SSE for k= 36 : 1620853.84
SSE for k= 37 : 1543326.4
SSE for k= 38 : 1492824.84
SSE for k= 39 : 1431699.03
SSE for k= 40 : 1392815.77
```

Question 8b: Plot SSE values for k = range(2,40), i.e, for k=2,3,4,...,40

In [98]:
```
plt.plot(range(2,41),score[2:41])
plt.scatter(range(2,41),score[2:41])
plt.xlabel('k')
plt.ylabel('SSE')
plt.title('SSE for different values of k')
plt.show()
```



Question 8c: Using this plot, determine the 'k' that you will use to do K-Means clustering.

Answer: I would choose a k value of 2 for this K-Means Clustering

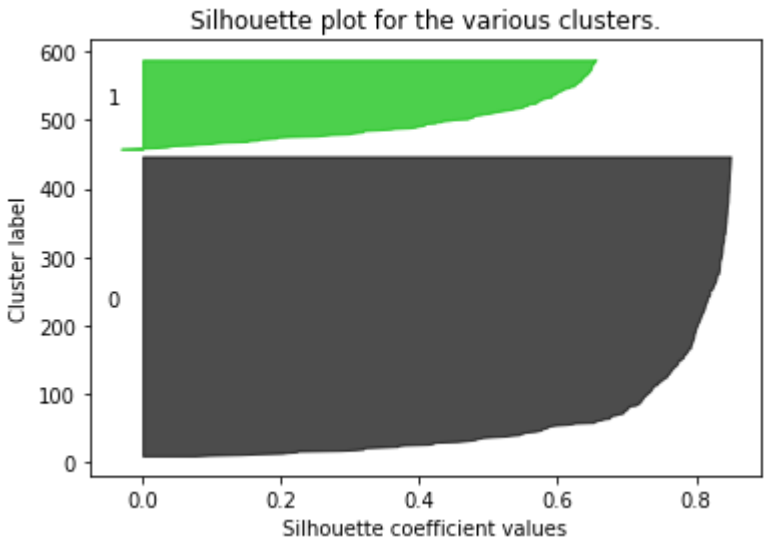
Question 8d: Using the 'k' you chose in (c), compute k-Means clustering.

In [99]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred = kmeans.fit_predict(Cancer_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=Cancer_y) # true clusters
plt.title('Cancer Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```

Question 8e: Plot the silhouette values for points in each cluster (using the silhouette() function provided in the practice notebook). .

In [100]:
```
def silhouette(X,labels):
    n_clusters = np.size(np.unique(labels));
    sample_silhouette_values = silhouette_samples(X, labels)
    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_values[labels == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        plt.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        #Compute the new y_lower for next cluster
        y_lower = y_upper + 10  # 10 for the 0 samples
    plt.title("Silhouette plot for the various clusters.")
    plt.xlabel("Silhouette coefficient values")
    plt.ylabel("Cluster label")
    plt.show()
```

In [101]:
```
silhouette(Cancer_X,y_pred)
```



Question 8f: Comment on the quality of the clusters discovered using k-Means. Which of the clusters would you treat as good clusters and which clusters do you treat as not-so-good clusters?

Answer: Neither of the silhouette coefficients seem to be terrible, but cluster 1 is a much better cluster than cluster 0. I would accept both of these clusters as viable.

Question 8g: Compute the Rand Index of the k-means clusters with respect to the true labels. Comment on the quality of the clustering based on the Rand-Index score.

In [102]:
```
n_clusters = 2;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred_kmeans = kmeans.fit_predict(Cancer_X)
print(rand_index(y_pred_kmeans, Cancer_y))
```

```
0.7503774845912028
```

Answer:

THe quality of the clustering is okay but not optimal for the purpose it is trying to serve.

Question 8h: To use DBSCAN to find clusters in this data, one needs to determine eps and min_samples. To do this, consider the range of values eps = 50, 100, 150, 200, 250, 300, 400, 500 and min_samples = 10, 15, 20, 25, 30.

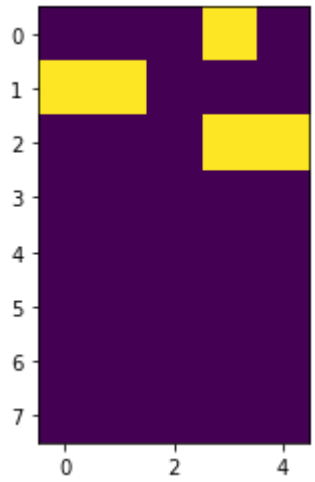
For these range of eps and min_samples values, compute an 8x5 matrix (with rows as eps values and cols as min_samples) to show the number of clusters obtained at each of these parameters. Visualize this matrix using imshow() in matplotlib.

Hint: To compute the number of clusters, you may use:

y_pred = dbscan.fit_predict(Cancer_X)

max(y_pred)+1

In [103]:
```
score = np.zeros(shape=(8,5));
epsilon = [50,100,150,200,250,300,400,500]
min_samples = [10,15,20,25,30]
for i in range(0,8):
    for j in range(0,5):
        dbscan = DBSCAN(eps=epsilon[i], min_samples=min_samples[j])
        y_pred = dbscan.fit_predict(Cancer_X)
        score[i][j] = max(y_pred)+1
plt.imshow(score)
plt.show()
```



In [ ]:

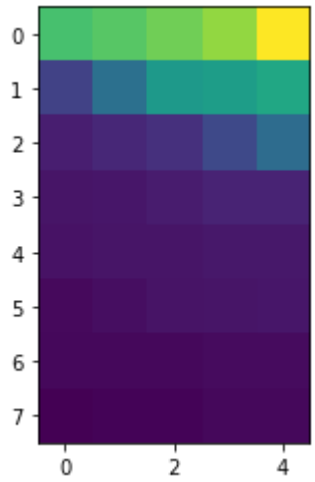
Question 8i: For these range of eps and min_samples values, compute an 8x5 matrix (with rows as eps values and cols as min_samples) to show the number of noise points obtained at each of these parameters. Visualize this matrix using imshow() in matplotlib.

Hint: To compute the number of noise points, you may use:

y_pred = dbscan.fit_predict(Cancer_X)

sum(y_pred==-1)

In [104]:
```
score = np.zeros(shape=(8,5));
epsilon = [50,100,150,200,250,300,400,500]
min_samples = [10,15,20,25,30]
for i in range(0,8):
    for j in range(0,5):
        dbscan = DBSCAN(eps=epsilon[i], min_samples=min_samples[j])
        y_pred = dbscan.fit_predict(Cancer_X)
        score[i][j] = sum(y_pred==-1)
plt.imshow(score)
plt.show()
```



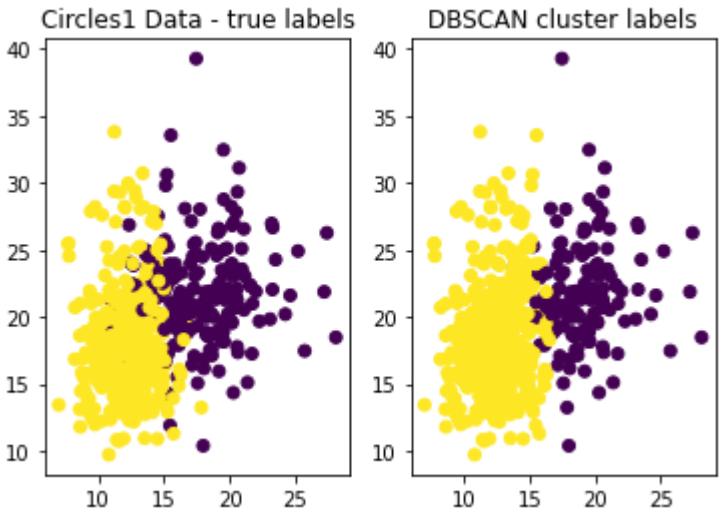
In [ ]:

Question 8j: What observations can you make about the clustering structure in this data, based on the matrices you generated for 8(g) and 8(h)?

Answer: Generally lower epsilon values will lead to less clusters. Lower epsilon and more min samples will lead to more noise points

Question 8k: Select the parameters for eps, min_samples based on your answers for 8(g), 8(h) and 8(i). Compute cluster assigments using DBSCAN. Compute RandIndex of the cluster assignments with respect to the true labels.

In [110]:
```
dbscan = DBSCAN(eps=100, min_samples=20)
y_pred = dbscan.fit_predict(Cancer_X)
plt.subplot(1,2,1)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=Cancer_y)
plt.title('Circles1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=y_pred)
plt.title('DBSCAN cluster labels')
plt.show()
```



In [111]:
```
dbscan = DBSCAN(eps=100, min_samples=20)
y_pred = dbscan.fit_predict(Cancer_X)
for i in range(0,len(y_pred)):
    if y_pred[i] == -1:
        y_pred[i] = 2
print(rand_index(y_pred, Cancer_y))
```

```
0.7759350478972252
```

Question 8l: Compare RandIndex from 8(g) with that of 8(k) and determine which algorithm performed best? Based on this, comment on how the data/clusters may be distributed in R^d.

Answer: The DBscan performed slightly better than kmeans. Based on this information, I believe that you can infer the clusters are not well separated from each other because neither kmeans or dbscan does a good job of capturing the clusters. You could also infer that there is probably significant overlap between clusters and that these two dimensions may not be enough to properly separate the clusters.

In [ ]:

In [ ]: