Hands-on Exercise for CLUS Module

0. Setting up necessary packages and creating data

In [42]:
```
!pip install --user scikit-learn --upgrade
```

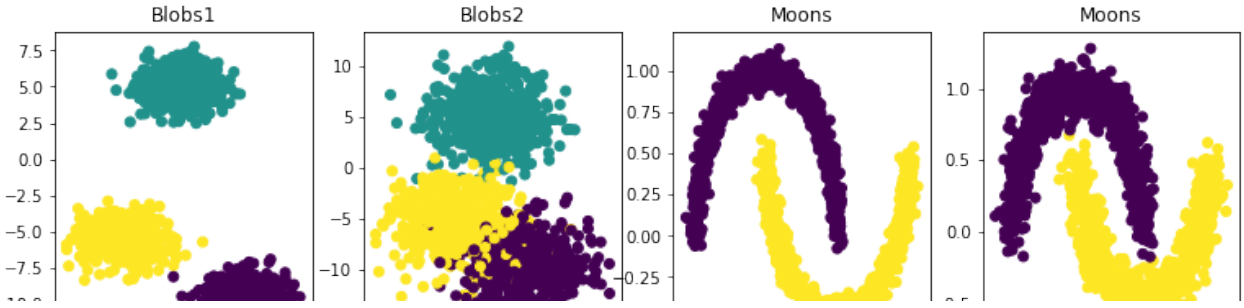
```
Requirement already up-to-date: scikit-learn in ./.local/lib/python3.6
/site-packages
Requirement already up-to-date: scipy>=0.17.0 in ./.local/lib/python3.
6/site-packages (from scikit-learn)
Requirement already up-to-date: numpy>=1.11.0 in ./.local/lib/python3.
6/site-packages (from scikit-learn)
Requirement already up-to-date: joblib>=0.11 in ./.local/lib/python3.6
/site-packages (from scikit-learn)
You are using pip version 9.0.1, however version 19.3.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' comm
and.
```

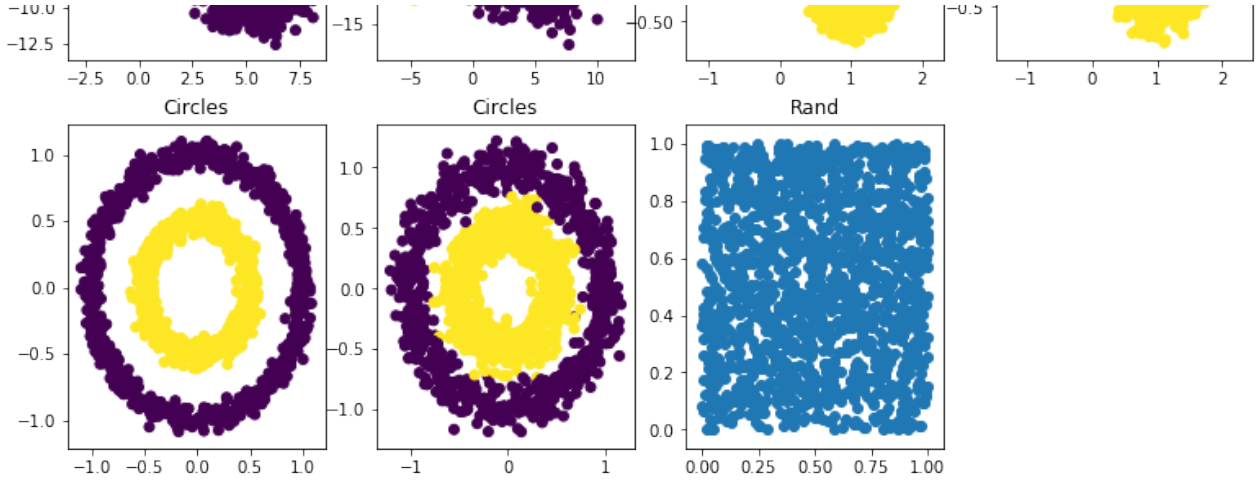
Import necessary packages

In [4]:
```
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 [5]:
```
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, facto
                                        noise=.05, random_state=random_stat
Circles2_X, Circles2_y = datasets.make_circles(n_samples=n_samples, facto
                                        noise=0.1, random_state=random_stat
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 [6]:
```
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 [8]:
```
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
        uj.append(u_dist[0][1])#distances to nearest neighbors in random
        w_dist,_=nbrs.kneighbors(X[rand_X[j]].reshape(1,-1),2,return_dist
        wj.append(w_dist[0][1])#distances to nearest neighbors in real da
    H=sum(uj)/(sum(uj)+sum(wj))
    if isnan(H):
        print(uj,wj)
        H=0

    return H
```

Code for Silhouette coefficient

In [9]:
```
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_silhou
        plt.text(-0.05,y_lower+0.5*size_cluster_i,str(i))#Compute the new
        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.

K-means works well on circular/globular data.k-means is an SSE based algorithm, Therefore, data which is not convex, k-means fails to capture clusters perfectly.Therefore k-means works really well on blob 1. When it comes to blob 2, the inter cluster distances is very less because of standard deviation.K-means does well here as well but not compared to blob 1 since in calculating SSE, when clusters are close enough, mean of one cluster tend to pull points from other 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.

As mentioned above, Since K-means is SSE based clustering, it works well on globular/circular data where there is high intra cluster cohesion and high inter-cluster separation among points. Therefore K-means is expected not to work well on Moon 1,2 and circle 1,2.

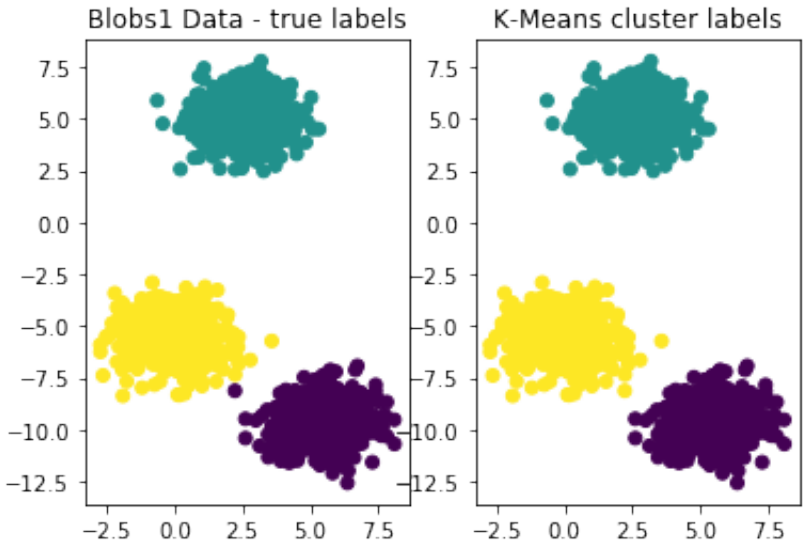
Question 1c: Run K-Means algorithm on all the datasets (except Rand). Choose n_clusters based on the number of clusters present in these datasets. Visualize the clusters for each of them. Based on the visualization, rank the datasets in decreasing order of K-means performance. Describe your rationale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

In [7]:
```
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred = kmeans.fit_predict(Blobs1_X)
```

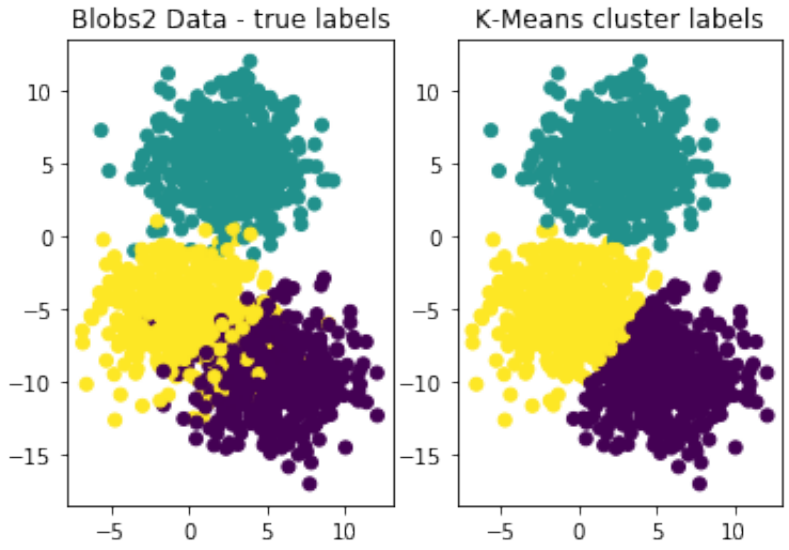
In [8]:
```
y_pred
```

Out[8]:
```
array([2, 1, 1, ..., 1, 1, 1], dtype=int32)
```

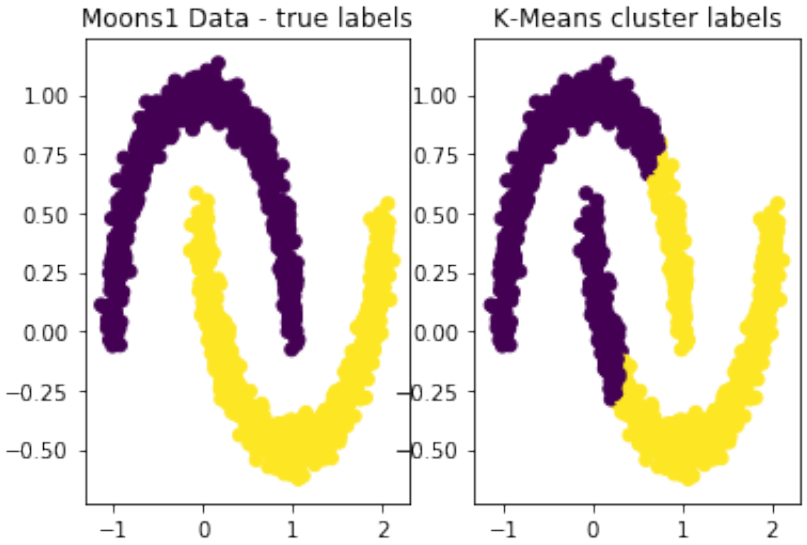
In [9]:
```
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y) # true clusters
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



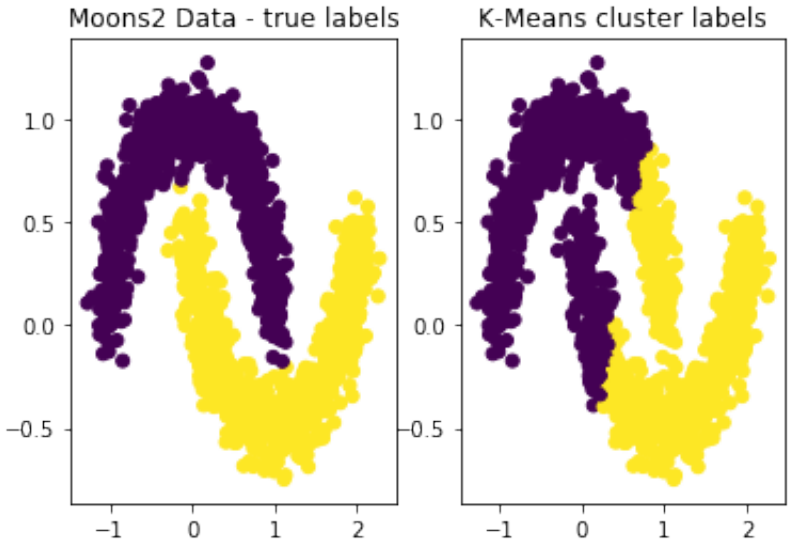
In [33]:
```
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred = kmeans.fit_predict(Blobs2_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y) # true clusters
plt.title('Blobs2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)  # KMeans clusters
plt.title('K-Means cluster labels')
plt.show()
```



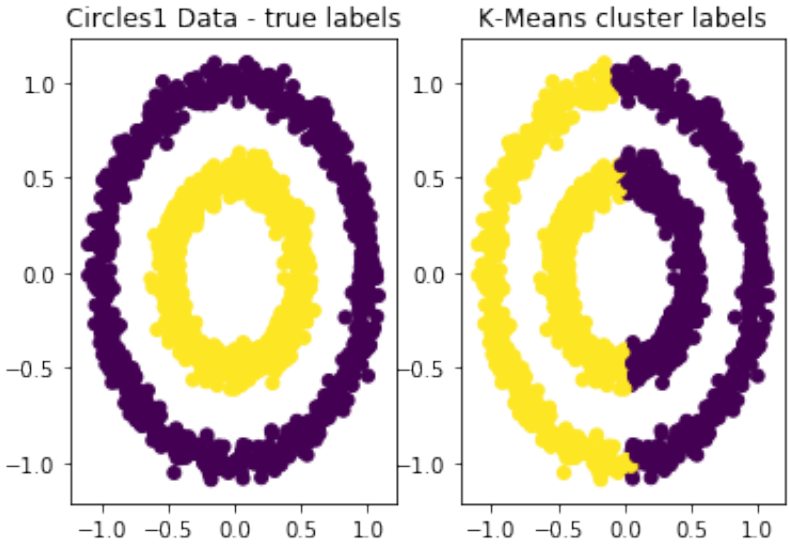
In [25]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred = kmeans.fit_predict(Moons1_X)
y_pred = kmeans.fit_predict(Moons1_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y) # true clusters
plt.title('Moons1 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()
```



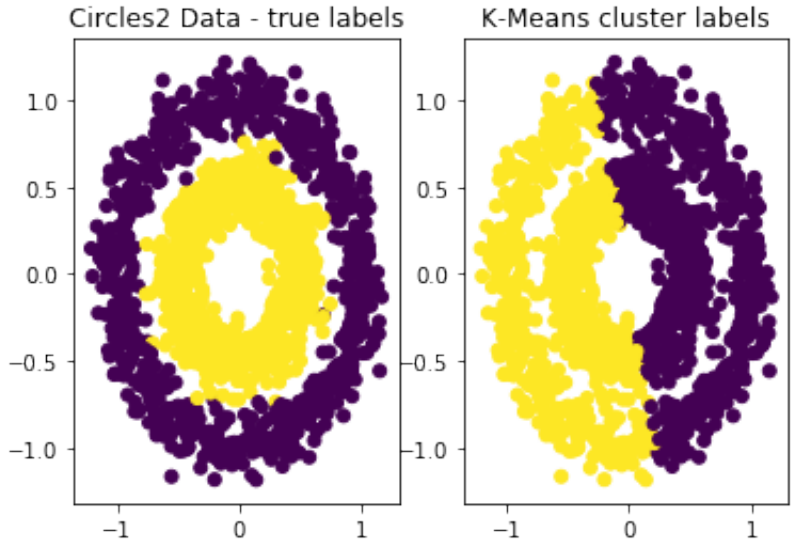
In [24]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred = kmeans.fit_predict(Moons2_X)
y_pred = kmeans.fit_predict(Moons2_X)
fig, ax = plt.subplots()
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y) # true clusters
plt.title('Moons2 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 [23]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred = kmeans.fit_predict(Circles1_X)
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 clus
plt.title('Circles1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles1_X[:, 0], Circles1_X[:, 1], c=y_pred)  # KMeans clust
plt.title('K-Means cluster labels')
plt.show()
```



In [22]:
```
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred = kmeans.fit_predict(Circles2_X)
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 clus
plt.title('Circles2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=y_pred)  # KMeans clust
plt.title('K-Means cluster labels')
plt.show()
```



In [ ]:
```
According to the plots, Blob1>Blobs2>Moons2>Moons1>Circles1>Circles2. As
k-means worked flawless on those. As intercluster distance decreases, k-m
75% was clustered properly. Moons1 and Moons2 are almost same. Similarly,
performance of k-means is considered. Because of shape of their data,near
```

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.

In [31]:
```
n_clusters=[3,3,2,2,2,2]
xs = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
ys = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]

for n_clusters,xs,ys in zip(n_clusters,xs,ys):

    kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
    y_pred = kmeans.fit_predict(xs)
    print(rand_index(y_pred,ys))
```

```
0.99911140760507
0.9207142539470758
0.6201236379808761
0.6240836112964199
0.4996744496330887
0.4996806760062264
```

```
According to rand scores, Blobs1>Blobs2>Moons2>Moons1>Circles1=Circles2
```

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

Yes, rankings in c and d are almost consistent.

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.

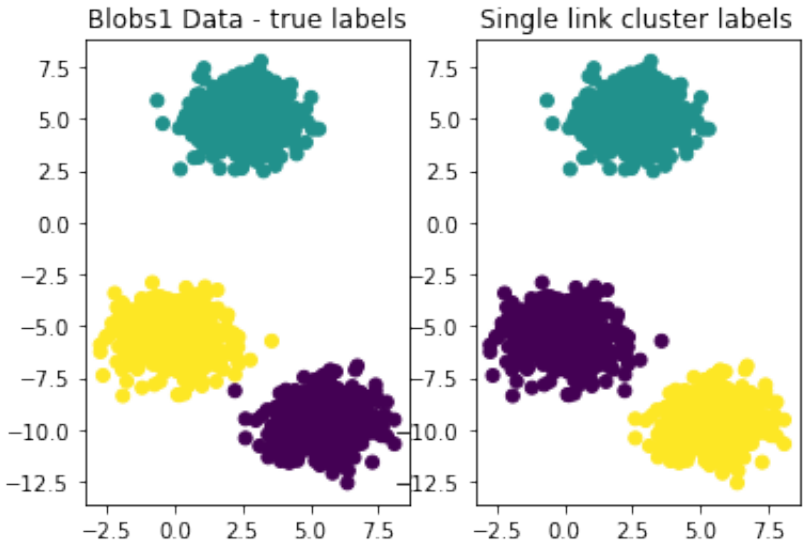
Single link clustering considers shortest distance/ high similarity between points. Therefore points in the same cluster should have less distance between eachother and should have fair separation between clusters. In the given datasets, Blobs1,Moons1,Circles1 have high cohesion and fair seperation between clusters. Therefore single link agglomerative clustering is predicted to work well on these sets.

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.

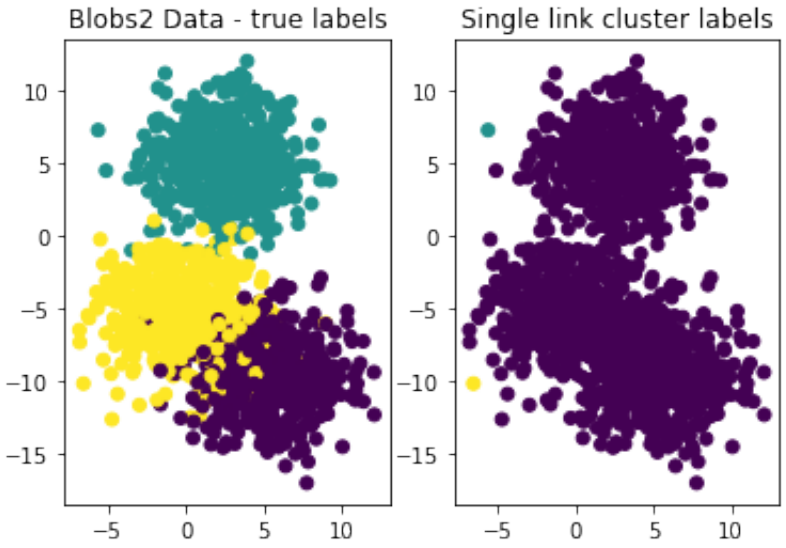
Single link clustering doesnt work well on Blobs2, Moons2, Circles2 since they have high standard deviation which makes the clusters to overlap. When clusters overlap, clustering becomes hard.

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.

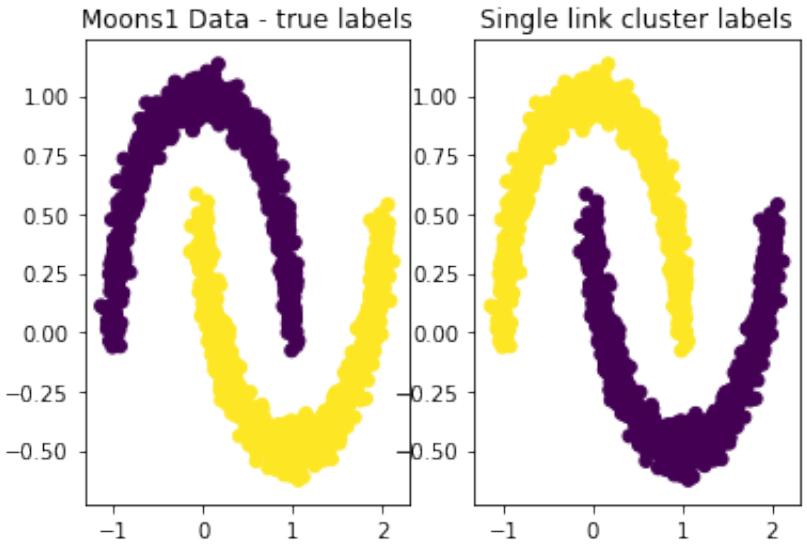
In [25]:
```
n_clusters = 3
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_c
y_pred = single_linkage.fit_predict(Blobs1_X)
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



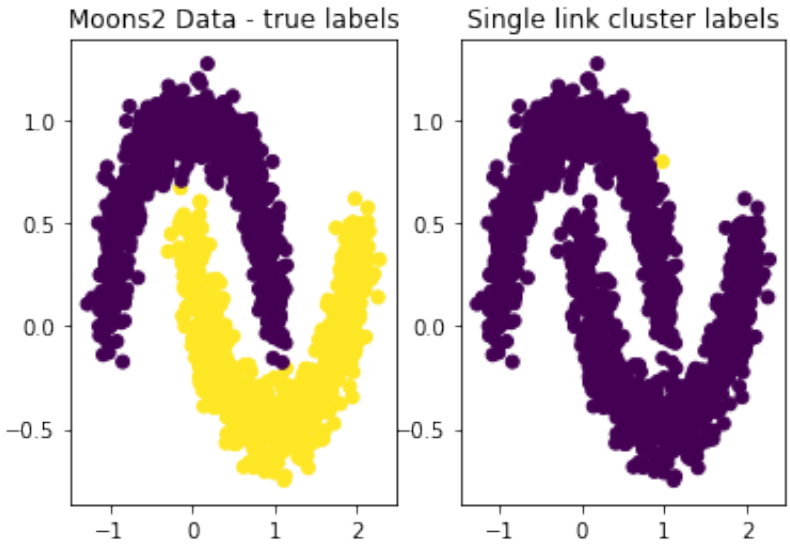
In [29]:
```
n_clusters = 3
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_c
y_pred = single_linkage.fit_predict(Blobs2_X)
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y)
plt.title('Blobs2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



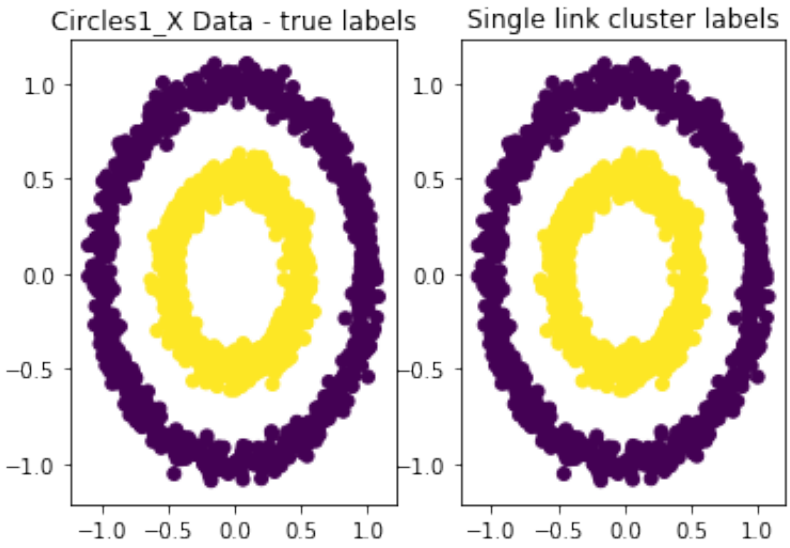
In [33]:
```
n_clusters = 2
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_c
y_pred = single_linkage.fit_predict(Moons1_X)
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y)
plt.title('Moons1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



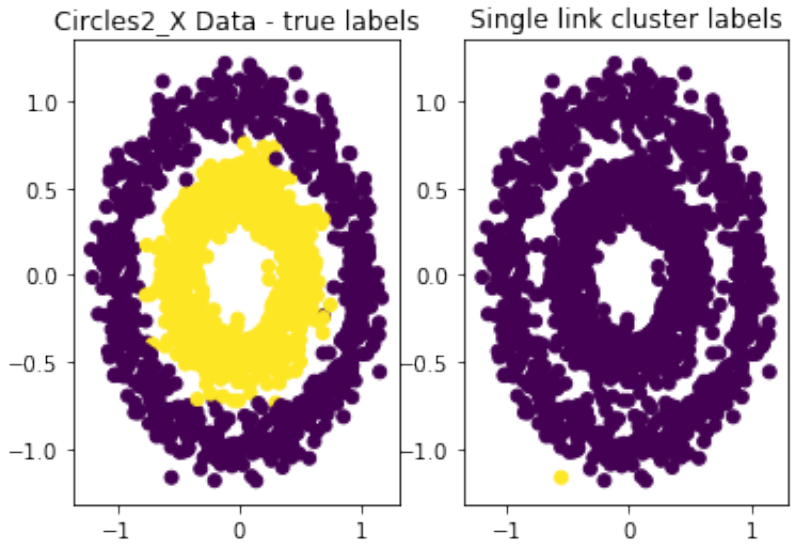
In [35]:
```
n_clusters = 2
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_c
y_pred = single_linkage.fit_predict(Moons2_X)
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y)
plt.title('Moons2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=y_pred)
plt.title('Single link cluster labels')
plt.show()
```



In [36]:
```
n_clusters = 2
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_c
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('Circles1_X 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 [37]:
```
n_clusters = 2
single_linkage = AgglomerativeClustering(linkage="single", n_clusters=n_c
y_pred = single_linkage.fit_predict(Circles2_X)
plt.subplot(1,2,1)
plt.scatter(Circles2_X[:, 0], Circles2_X[:, 1], c=Circles1_y)
plt.title('Circles2_X 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()
```



Blobs1=Moons1=Circles1> Circles2=Moons2>Blobs2. Single link clustering works on the basis of max similarity or least distance. It starts with n points(each cluster being single point) and merges points with least distances. All the dta with subscript 1 are well seperated. But data with subscript 2 has high standard deviation. That is distance between clusters very less. Therefore they cannot form clusters as expected.In Blob2, everythin is considered as single cluster leaving out one-two noise points giving them status of new clusters.

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.

In [39]:
```
n_clusters=[3,3,2,2,2,2]
xs = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
ys = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]

for n_clusters,xs,ys in zip(n_clusters,xs,ys):

    single_linkage = AgglomerativeClustering(linkage="single", n_clusters
    y_pred = single_linkage.fit_predict(xs)
    print(rand_index(y_pred,ys))
```

```
0.99911140760507
0.33377896375361354
1.0
0.49966733377807426
1.0
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.

Yes the rankings in 2c are almost consistent with 2d. In Blob1, one point is pulled by different cluster because of distance measure therefore score was a bit less than 1.0. Except for that rand index values are consistent.

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.

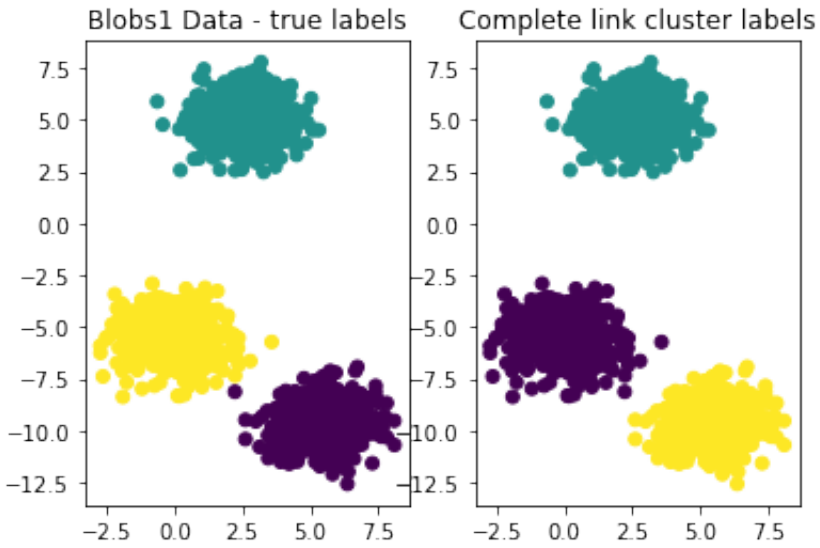
Complete link can be defined as maximum distance between points in cluster ci and cluster cj.Complete link is noise prone. Therefore data should have less noise. Also, they work well on convex shapes.Therefore they work well on Blobs1,Blobs 2(relatively better when compared to other shapes but not flawless).

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.

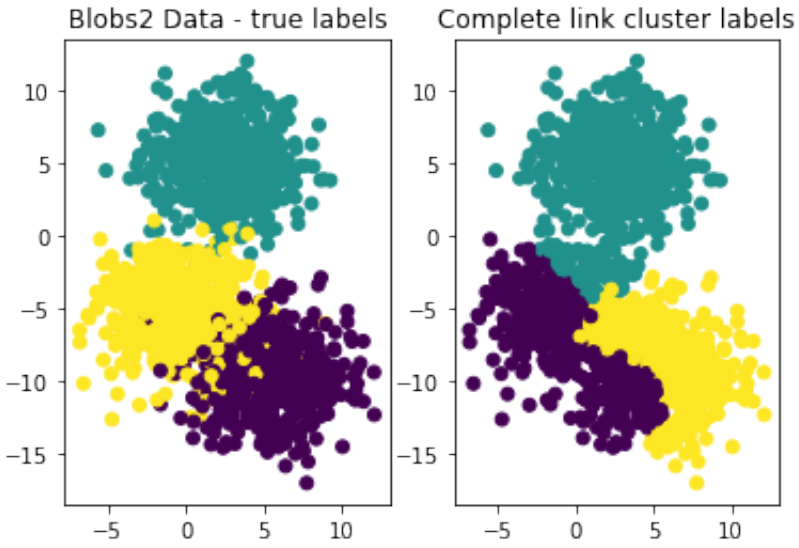
Max-link agglomerative clustering works poor on shapes which are convex. Since the measure here is max distance, points in same cluster may be distant from eachother when compared to points in other cluster. Let us consider moon shape. points in both edges of moon are distant from eachother when compared to other moon cluster. Therefore it doesn't work well on Moons1,2, Circles1,2.

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.

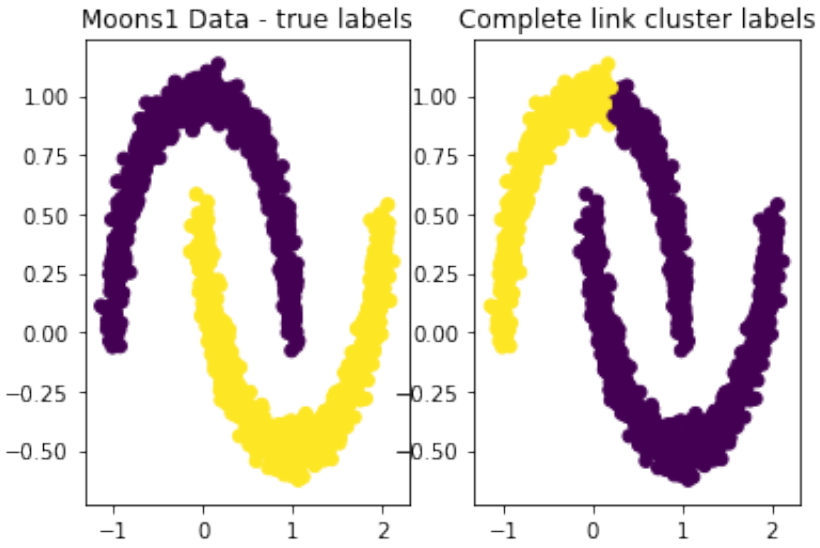
In [40]:
```
n_clusters = 3
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters
y_pred = complete_linkage.fit_predict(Blobs1_X)
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



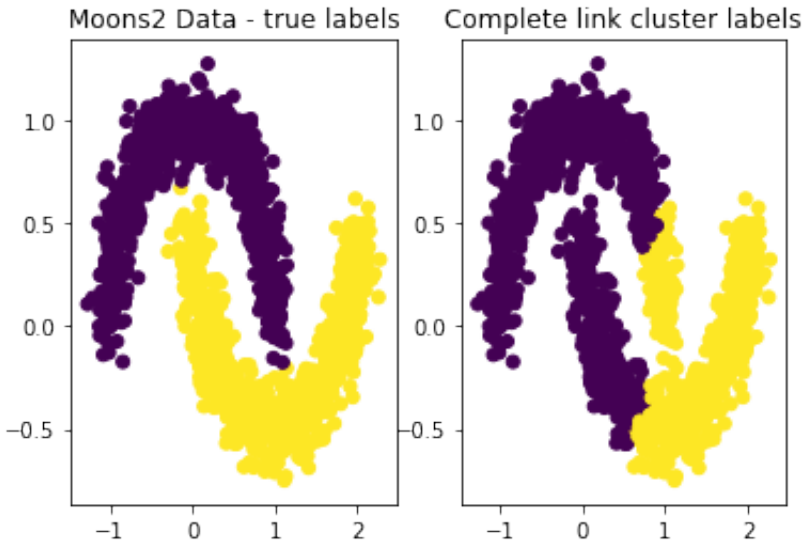
In [41]:
```
n_clusters = 3
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters
y_pred = complete_linkage.fit_predict(Blobs2_X)
plt.subplot(1,2,1)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=Blobs2_y)
plt.title('Blobs2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs2_X[:, 0], Blobs2_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



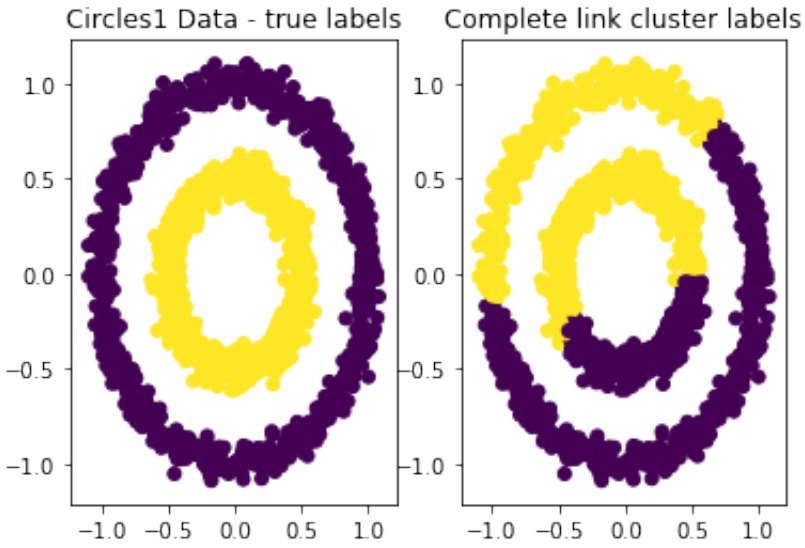
In [42]:
```
n_clusters = 2
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters
y_pred = complete_linkage.fit_predict(Moons1_X)
plt.subplot(1,2,1)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=Moons1_y)
plt.title('Moons1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons1_X[:, 0], Moons1_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



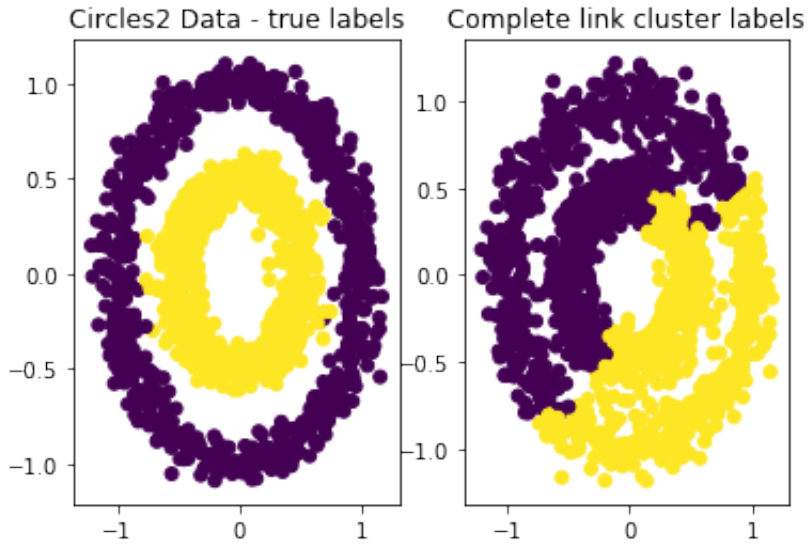
In [44]:
```
n_clusters = 2
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters
y_pred = complete_linkage.fit_predict(Moons2_X)
plt.subplot(1,2,1)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=Moons2_y)
plt.title('Moons2 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Moons2_X[:, 0], Moons2_X[:, 1], c=y_pred)
plt.title('Complete link cluster labels')
plt.show()
```



In [45]:
```
n_clusters = 2
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters
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('Circles1 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 [46]:
```
n_clusters = 2
complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters
y_pred = complete_linkage.fit_predict(Circles2_X)
plt.subplot(1,2,1)
plt.scatter(Circles2_X[:, 0], Circles1_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('Complete link cluster labels')
plt.show()
```



Blobs1>Blobs2>Moons1>Moons2>Circles1>Circles2. Here, Blobs1 got good seperated clusters. Then in Blobs2, some points were pulled by both clusters in proximity.Then moons1 got one full good cluster and one half cluster other half being pulled by other cluster. Role played by noise points are noteworthy.Then comes circles which showed up half and half pulling nearest points into their respective clusters.

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.

In [47]:
```
n_clusters=[3,3,2,2,2,2]
xs = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
ys = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]

for n_clusters,xs,ys in zip(n_clusters,xs,ys):

    complete_linkage = AgglomerativeClustering(linkage="complete", n_clus
    y_pred = complete_linkage.fit_predict(xs)
    print(rand_index(y_pred,ys))
```

```
0.99911140760507
0.7736544362908606
0.662605292417167
0.5965310206804536
0.5218714698688014
0.5000587058038692
```

In [ ]:
```
Blobs1>Blobs2>Moons1>Moons2>Circles1>Circles2
```

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.

Yes, The rankings in 3c and 3d are consistent.

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.

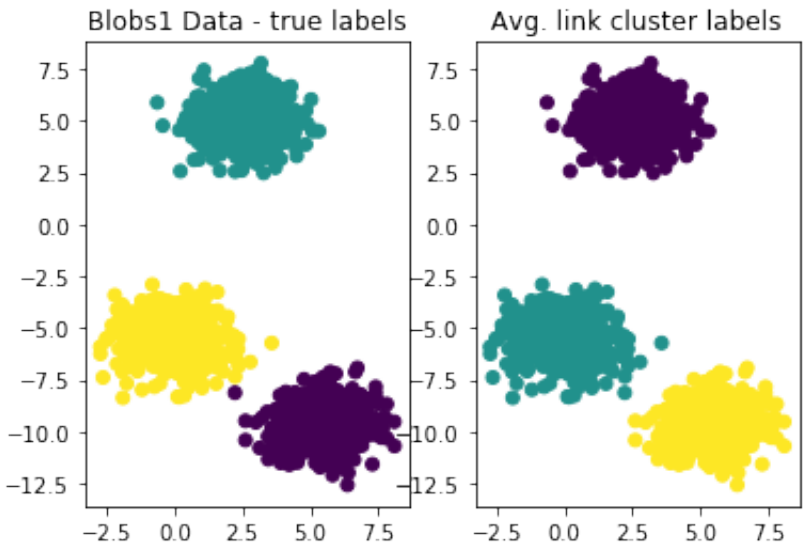
Similar to other HAC approaches, average link clustering also tends to perform poor on irreguar shapes and outliers.Average link clustering takes average distance between points in cluster and point/cluster which is to be merged.Thus avoiding pitfalls of single link and max link. Therefore, based on cluster separation and shape of clusters, average link agglomerative clustering works. Therefore I assume it works well only on Blobs1 data.

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.

Average link clustering doesnt perform well on Blobs2,Moons1,2 and Circles1,2. This is because of their shape and lack of separation.

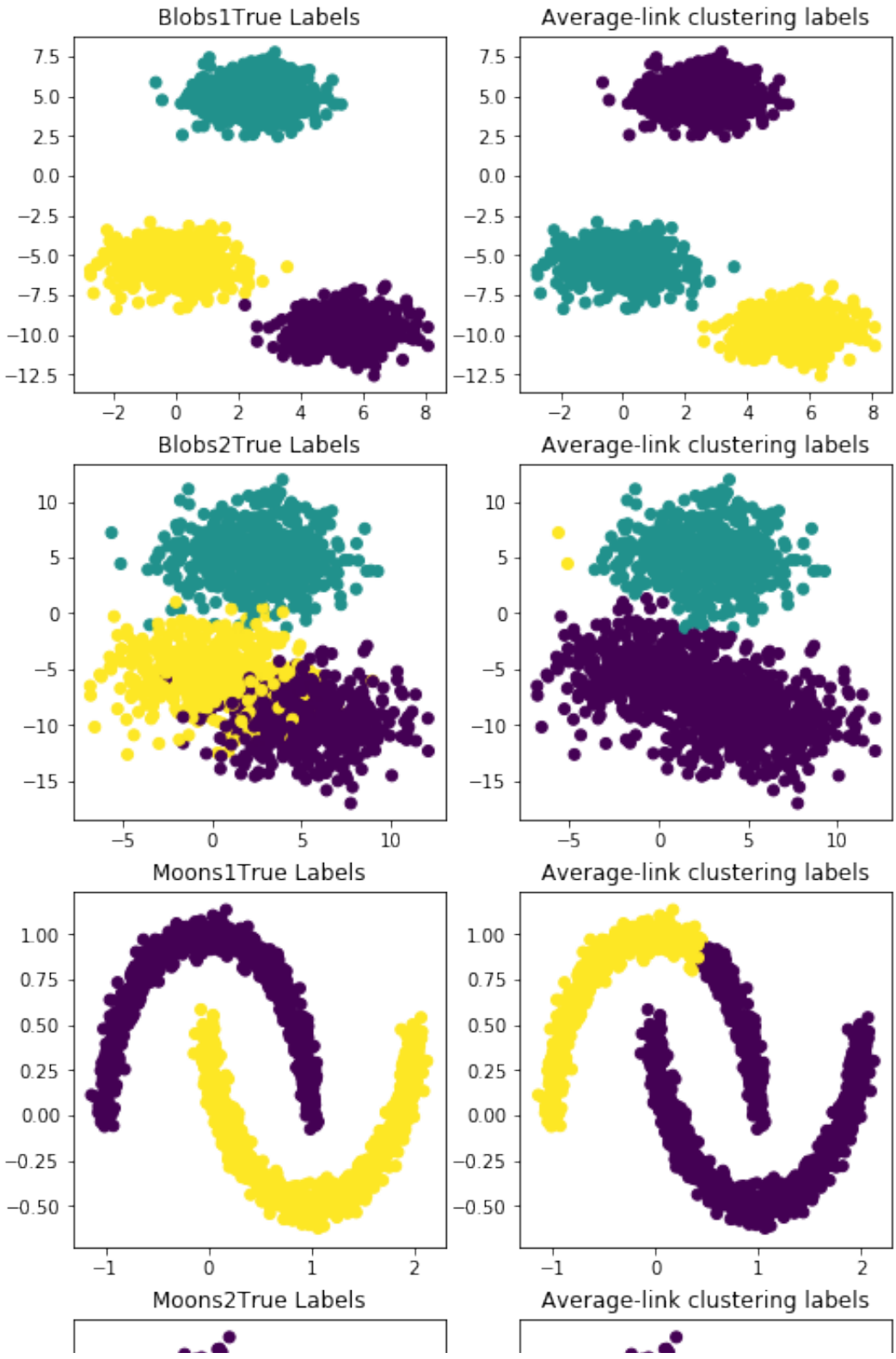
Question 4c: Run Average-link agglomerative clustering algorithm on all the datasets (except Rand). Choose n_clusters based on the number of clusters present in these datasets. Visualize the clusters for each of them. Based on the visualization, rank the datasets in decreasing order of Average-link agglomerative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

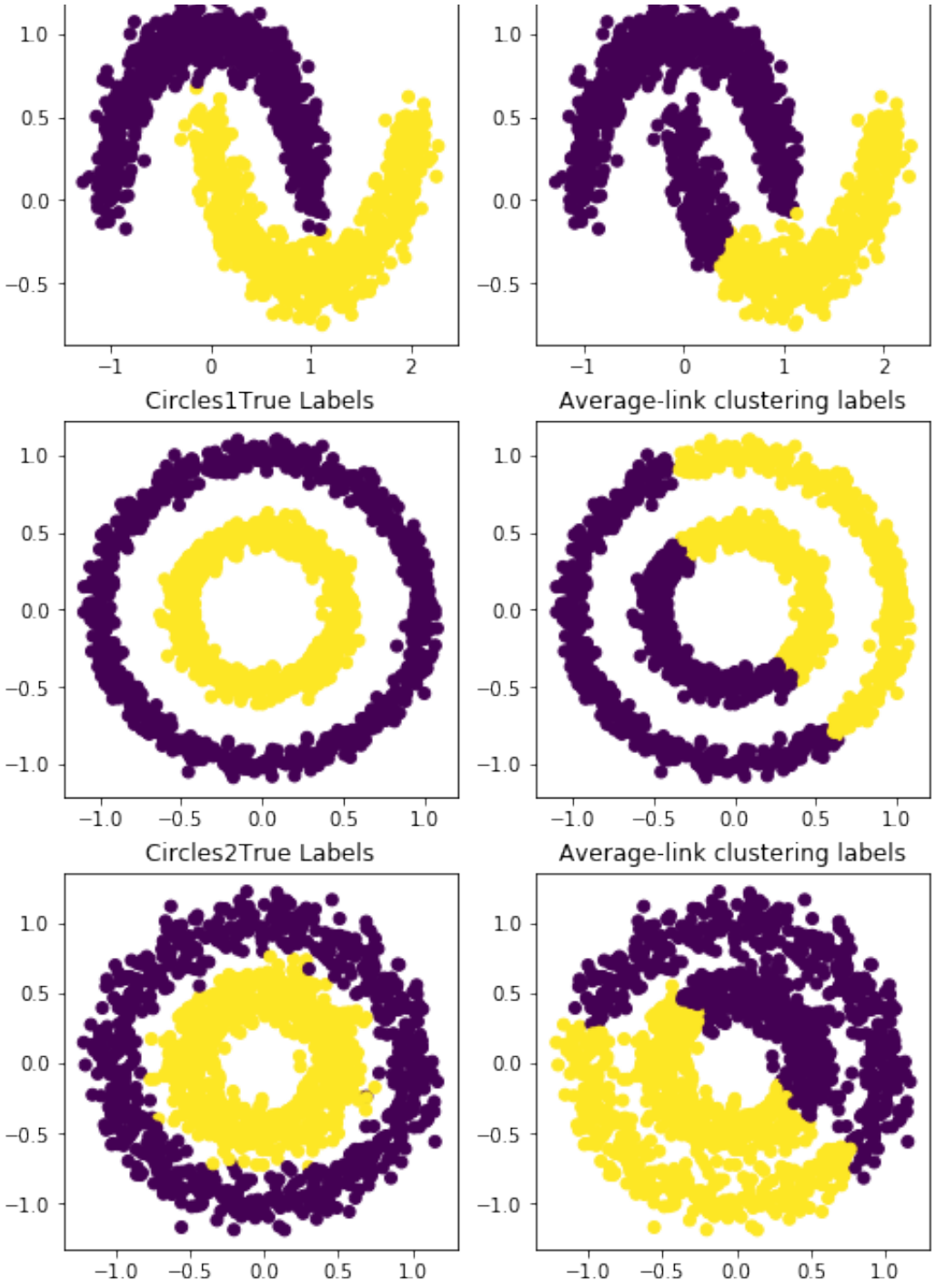
In [48]:
```
n_clusters = 3
average_linkage = AgglomerativeClustering(linkage="average", n_clusters=n
y_pred = average_linkage.fit_predict(Blobs1_X)
plt.subplot(1,2,1)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=Blobs1_y)
plt.title('Blobs1 Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Blobs1_X[:, 0], Blobs1_X[:, 1], c=y_pred)
plt.title('Avg. link cluster labels')
plt.show()
```



In [53]:
```
clusters = [3,3,2,2,2,2]
DX = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
DY = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]
labels = ['Blobs1','Blobs2','Moons1','Moons2','Circles1','Circles2']
c=0
fig= plt.figure(figsize=(8,25))
for cluster,label,D_X,D_Y in zip(clusters,labels,DX,DY):
    c= c+1
    plt.subplot(6,2,c)
    plt.scatter(D_X[:,0],D_X[:,1], c=D_Y)
    plt.title(label+'True Labels')
    c= c+1
```

```
    plt.subplot(6,2,c)
    average_linkage = AgglomerativeClustering(linkage="average", n_cluste
    y_pred = average_linkage.fit_predict(D_X)
    plt.scatter(D_X[:,0],D_X[:,1], c=y_pred)
    plt.title('Average-link clustering labels')
plt.show()
```





As we can see, Blob1 will have highest rank since it almost formed three clusters. Blobs2 forms two clusters instead of 3. 3rd cluster will be assigned to outliers.This shows even Average link clustering is bothered by outliers.Then next rank will be given to Moons1 and Moons 2. Last rank will be given to Circles1 and Circles 2 since they got clustered half correct.

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.

In [57]:
```
n_clusters=[3,3,2,2,2,2]
xs = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
ys = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]

for n_cluster,x,y in zip(n_clusters,xs,ys):

    average_linkage = AgglomerativeClustering(linkage="average", n_cluste
    y_pred = average_linkage.fit_predict(x)
    print(rand_index(y_pred,y))
```

```
0.99911140760507
0.7636575494774294
0.7132310429175005
0.7457647320435846
0.500414498554592
0.5050780520346898
```

In [ ]:
```
Blobs1>Blobs2>Moons1>Moons2>Circles2>Circles1
```

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.

Ranks are almost consistent except for moons data. This is because average link considers all points instead of max or min distances. When spread is high,average clustering slightly surpassed the accuracy of data with less spread. This could have happened to moon as well if the tail of moon was a bit away from other cluster. By this we can conclude that all HAC and SSE based algorithms are not expected to work with irregular shapes and when there are outliers.

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.

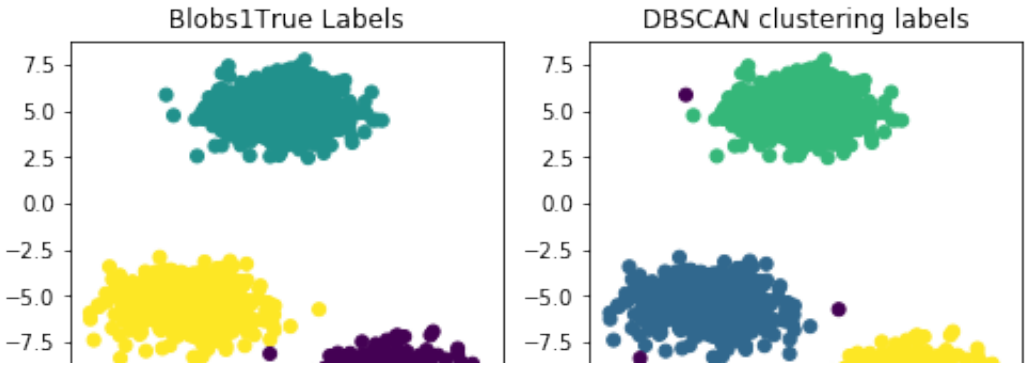
Density based clustering doesn't work with regular notion of distance. Instead it works with continuity of points. Therefore, Except for Blobs2 in which 2 clusters are overlapped, DBSCAN will consider them to be one cluster instead of two in view of their continuity.

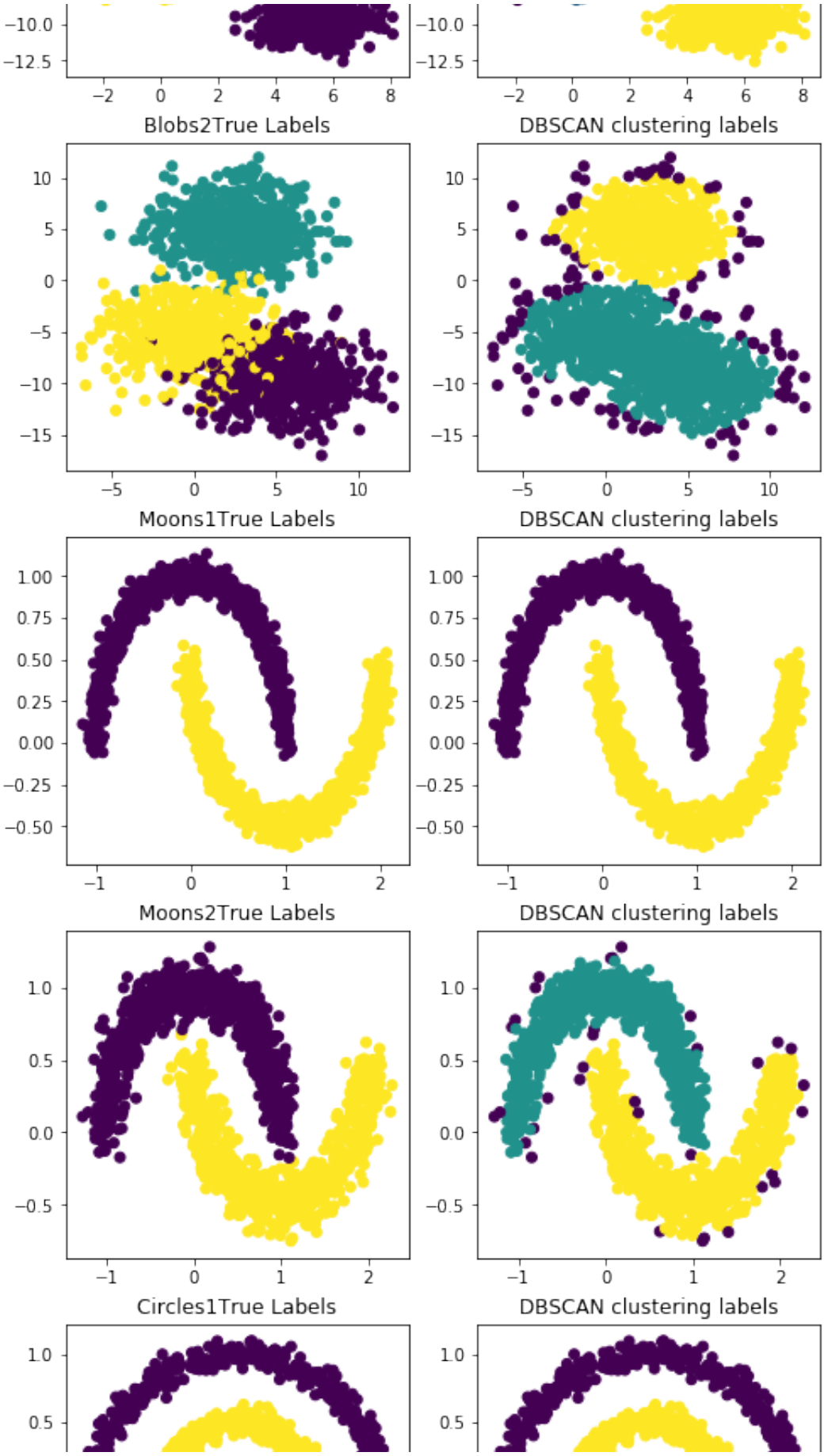
Question 5b: Without running DBSCAN clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where DBSCAN clustering is expected to NOT work well. Support your answer by explaining your rationale.

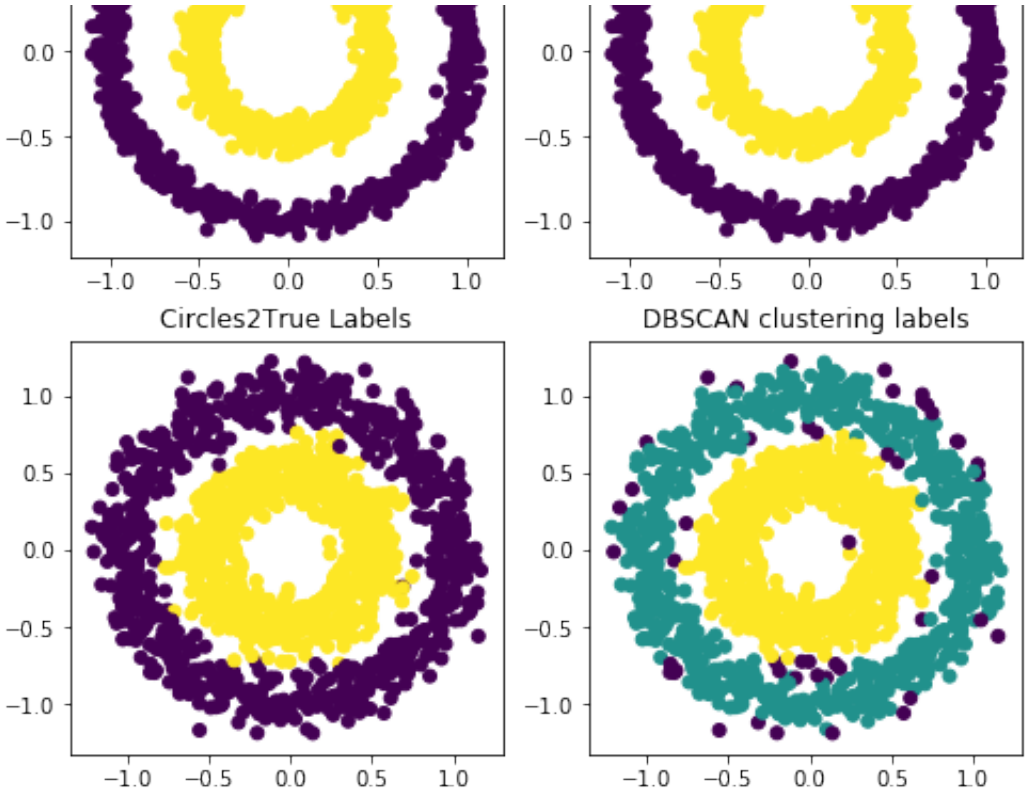
DBSCAN is not bothered by noisy data but overlapping data can be misleading since DBSCAN searches for conituity and density. Once two clusters overlap, DBSCAN fails to differentiate two clusters if eps and minimum samples conditions satisfy. Therefore DBSCAN is expected not to work well on Blobs2

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.

In [17]:
```
clusters = [3,3,2,2,2,2]
DX = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
DY = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]
labels = ['Blobs1','Blobs2','Moons1','Moons2','Circles1','Circles2']
eps=[1,1.5,0.2,0.1,0.2,0.1]
min_samples=[10,30,10,8,10,8]
c=0
fig= plt.figure(figsize=(8,25))
for cluster,label,D_X,D_Y,min_sample,ep in zip(clusters,labels,DX,DY,min_
    c= c+1
    plt.subplot(6,2,c)
    plt.scatter(D_X[:,0],D_X[:,1], c=D_Y)
    plt.title(label+'True Labels')
    c= c+1
    plt.subplot(6,2,c)
    dbscan = DBSCAN(eps=ep, min_samples=min_sample)
    y_pred = dbscan.fit_predict(D_X)
    plt.scatter(D_X[:,0],D_X[:,1], c=y_pred)
    plt.title('DBSCAN clustering labels')
plt.show()
```




Blobs2True Labels
DBSCAN clustering labels
Moons1True Labels
DBSCAN clustering labels
Moons2True Labels
DBSCAN clustering labels
Circles1True Labels
DBSCAN clustering labels



We can say that Moons1 and circes1 got perfectly clustered where Moons2,Circles2 got some outliers.Therefore I would rank Moons1=Circles1>Blobs1>Moons2>Circles2,Blobs2. This is because DBSCAN categorizes points as either core point or boundary point or noise point. This categorization depends on choice of eps and minimum_samples. DBSCAN checks for minimum core points in given radius. If enough density is found, It is categorized as core point. Here,Circles1 have good density and separation.Therefore it clustered accurately. When data is spread, and if points donot have enough points in their radius, they are classified as boundary point. If point is neither core nor boundary, it is noise point. We can see many points categorized as outliers in Moon2 and Circle2 because of this reason(spread is high with low density for outer points).

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?

In [49]:
```
clusters = [3,3,2,2,2,2]
DX = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
DY = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]
labels = ['Blobs1','Blobs2','Moons1','Moons2','Circles1','Circles2']
eps=[1,1.5,0.2,0.1,0.2,0.1]
min_samples=[10,27,10,8,10,8]
for cluster,label,D_X,D_Y,min_sample,ep in zip(clusters,labels,DX,DY,min_
    dbscan = DBSCAN(eps=ep, min_samples=min_sample)
    y_pred = dbscan.fit_predict(D_X)
    c=0
    for n, i in enumerate(y_pred):
        if i== -1:
            c= c+1
    print(c)
```

```
3
109
0
34
0
48
```

Moon1,Circles1,Blobs1 have least noise points.DBSCAN classifies points as noise points when they neither belong to core points nor boundary points. When spread is less, there is less chance that points don't have enough points in their neighborhood to become core point.Other points will become boundary points. Therefore we find quite less noise points in Blobs1,Moons1 and Circles1.

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.

In [67]:
```
clusters = [3,3,2,2,2,2]
DX = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
DY = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]
labels = ['Blobs1','Blobs2','Moons1','Moons2','Circles1','Circles2']
eps=[1,1.5,0.2,0.1,0.2,0.1]
min_samples=[10,30,10,8,10,8]
c=0
for cluster,label,D_X,D_Y,min_sample,ep in zip(clusters,labels,DX,DY,min_
    dbscan = DBSCAN(eps=ep, min_samples=min_sample)
    y_pred = dbscan.fit_predict(D_X)
    for n, i in enumerate(y_pred):
        if i== -1:
            y_pred[n] =5
    print(rand_index(y_pred,D_Y))
```

```
0.997781632199244
0.7477144763175451
1.0
0.9762721814543028
1.0
0.9590802757393818
```

In [ ]:
```
Moons1=Circles1>Blobs1>Moons2>Circles2>Blobs2
```

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.

Yes rankings in 5c and 5d are consistent

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.

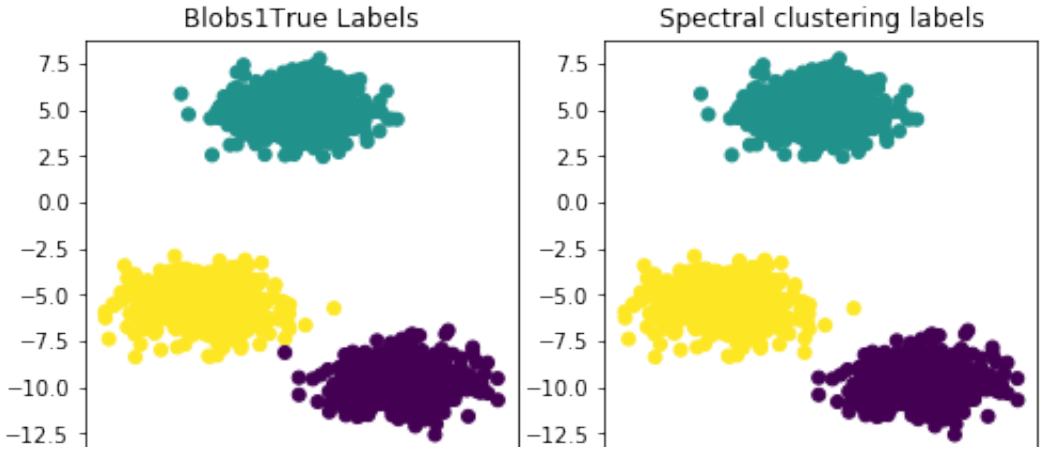
Spectral clustering is done using affinity matrix (A). Affinity matrix can be calculated many ways such as based on distance or dot product or gaussian kernel etc. If the metric is based on distance,then Blobs1 and Blobs2 do well because Blobs1 is well seperated. Blobs2 is also good clustering structure if distance is taken as measure.

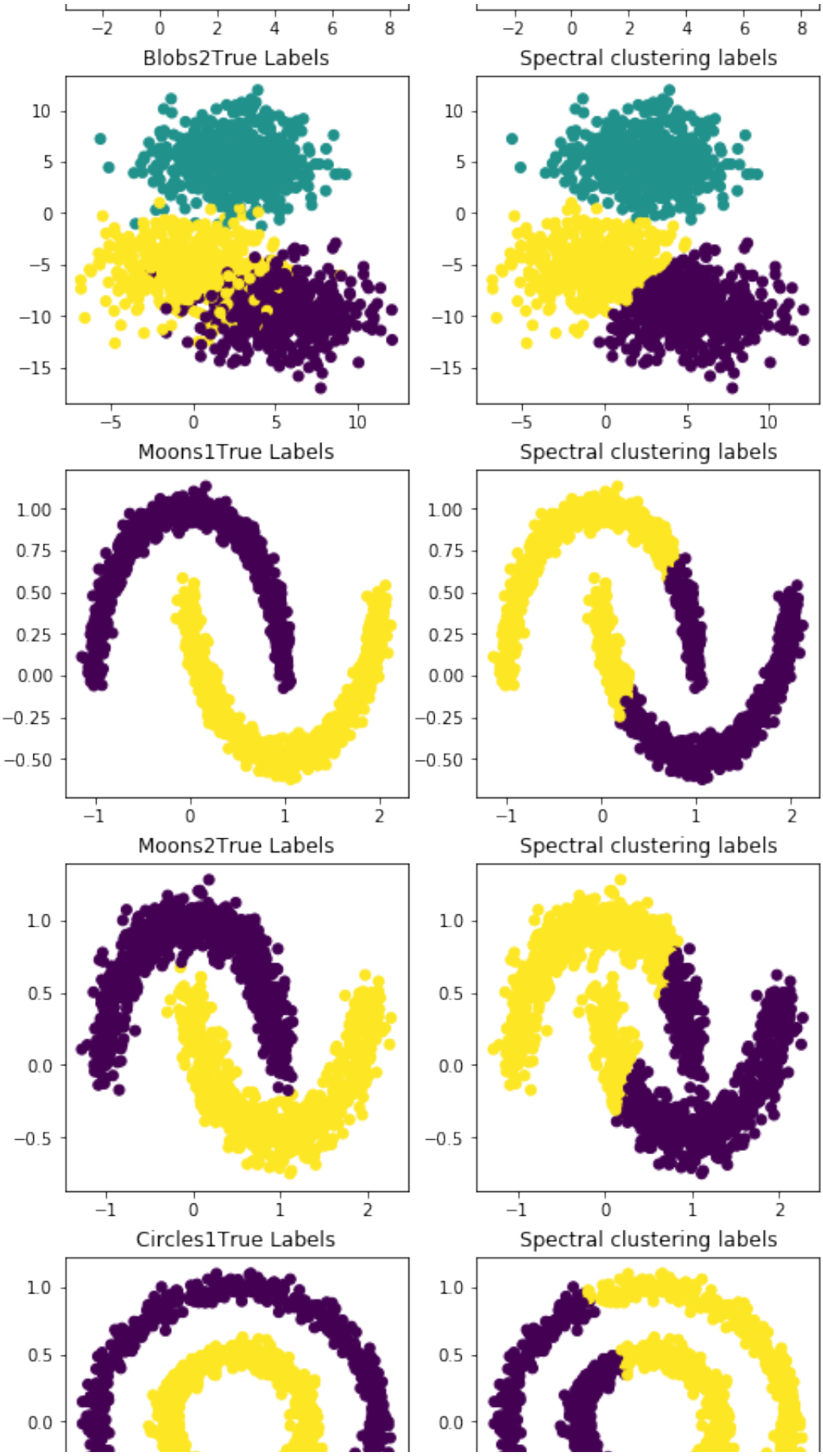
Question 6b: Without running Spectral clustering, for all the datasets (except Rand) provided in the practice session, list the datasets where Spectral clustering is expected to NOT work well. Support your answer by explaining your rationale.

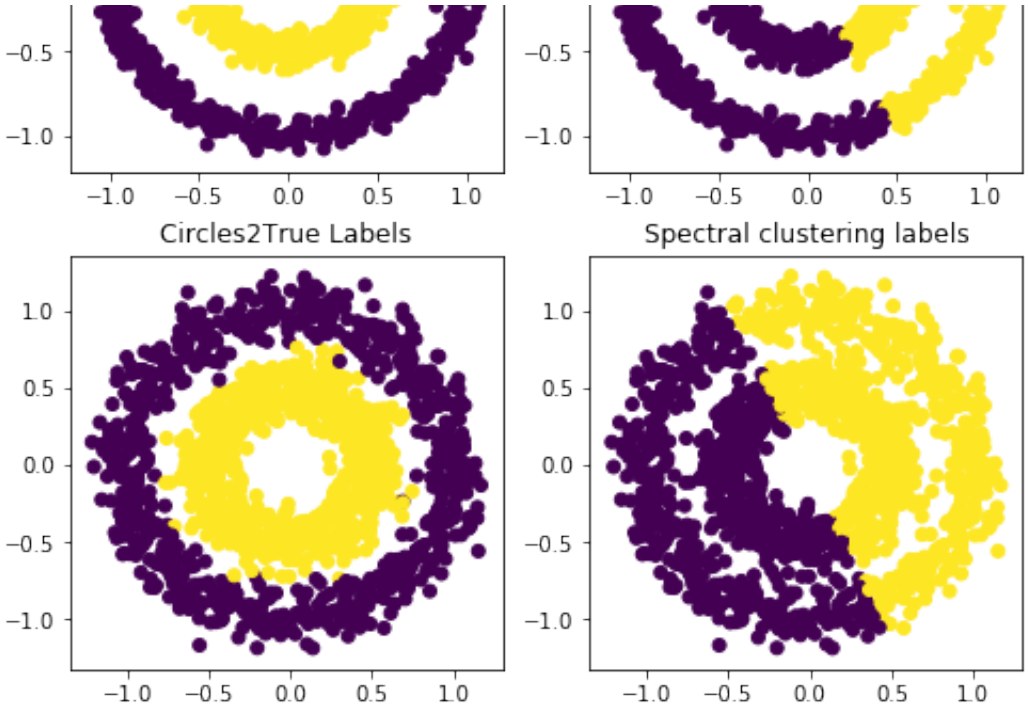
When affinity matrix is calculated based on distance, same problem arises when comapred to single link clustering. Shape of Moons and Circles is such a way that all points in one cluster are not close to eachother. Therefore it is not expected to work well on them.

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.

In [72]:
```
clusters = [3,3,2,2,2,2]
DX = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
DY = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]
labels = ['Blobs1','Blobs2','Moons1','Moons2','Circles1','Circles2']
eps=[1,1.5,0.2,0.1,0.2,0.1]
min_samples=[10,30,10,8,10,8]
c=0
fig= plt.figure(figsize=(8,25))
for cluster,label,D_X,D_Y in zip(clusters,labels,DX,DY):
    c= c+1
    plt.subplot(6,2,c)
    plt.scatter(D_X[:,0],D_X[:,1], c=D_Y)
    plt.title(label+'True Labels')
    c= c+1
    plt.subplot(6,2,c)
    spectral_clus = SpectralClustering(n_clusters= cluster,random_state=r
    y_pred = spectral_clus.fit_predict(D_X)
    plt.scatter(D_X[:,0],D_X[:,1], c=y_pred)
    plt.title('Spectral clustering labels')
plt.show()
```




Blobs2True Labels
Spectral clustering labels
Moons1True Labels
Spectral clustering labels
Moons2True Labels
Spectral clustering labels
Circles1True Labels
Spectral clustering labels



Based on visualizations, Blobs1>Blobs2>moon1=moons2>circles1=circles2. When affinity is considered to be distance, it replicates same problem we faced with HAC Complete link, complete link clusterings ie shape of the clusters bothers our clustering algorithm. Since Blobs1 is clearly separated, it got clustered very well.

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.

In [75]:
```
n_clusters=[3,3,2,2,2,2]
xs = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X]
ys = [Blobs1_y,Blobs2_y,Moons1_y,Moons2_y,Circles1_y,Circles2_y]

for n_cluster,x,y in zip(n_clusters,xs,ys):

    spectral_clus = SpectralClustering(n_clusters= n_cluster,random_state
    y_pred = spectral_clus.fit_predict(x)
    print(rand_index(y_pred,y))
```

```
0.99911140760507
0.919189682010229
0.6441263064265066
0.6448441183010896
0.49966733377807426
0.4997553924838781
```

In [ ]:
```
Blobs1>Blobs2>Moons2>Moons1>Circles2>Circles1
```

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.

Yes, the rankings in 6c and 6d are consistent.

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.

In general, Clustering tendency is measured by hopkins statistics. If spread of generated data is same as original data then we get h as 0.5. h will be zero if data has high clustering tendency. However, the measure here also is nearest neighbors ie distance. This shows good clustering tendency exists for data which has good separation and less spread. According to me Blobs1,Moons1,Circles1 data have good clustering tendency.

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.

As said in the above question, datasets with high spread is expected not to work well. Blobs2,Moons2,Circles2,Rand donot exhibit good clustering tendency.

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

In [27]:
```
clusters = [3,3,2,2,2,2]
DXs = [Blobs1_X,Blobs2_X,Moons1_X,Moons2_X,Circles1_X,Circles2_X,Rand_X]
labels = ['Blobs1','Blobs2','Moons1','Moons2','Circles1','Circles2','Rand
for DX,label in zip(DXs,labels):
    print(label,hopkins(DX))
```

```
Blobs1 0.9319059747158762
Blobs2 0.8176773779448973
Moons1 0.9250779967625641
Moons2 0.8716989606276608
Circles1 0.8661830600593758
Circles2 0.77354829252019
Rand 0.5857571134011118
```

In [ ]:
```
Blobs1> Moons1> Moons2> Circles1>Circles2>Rand
```

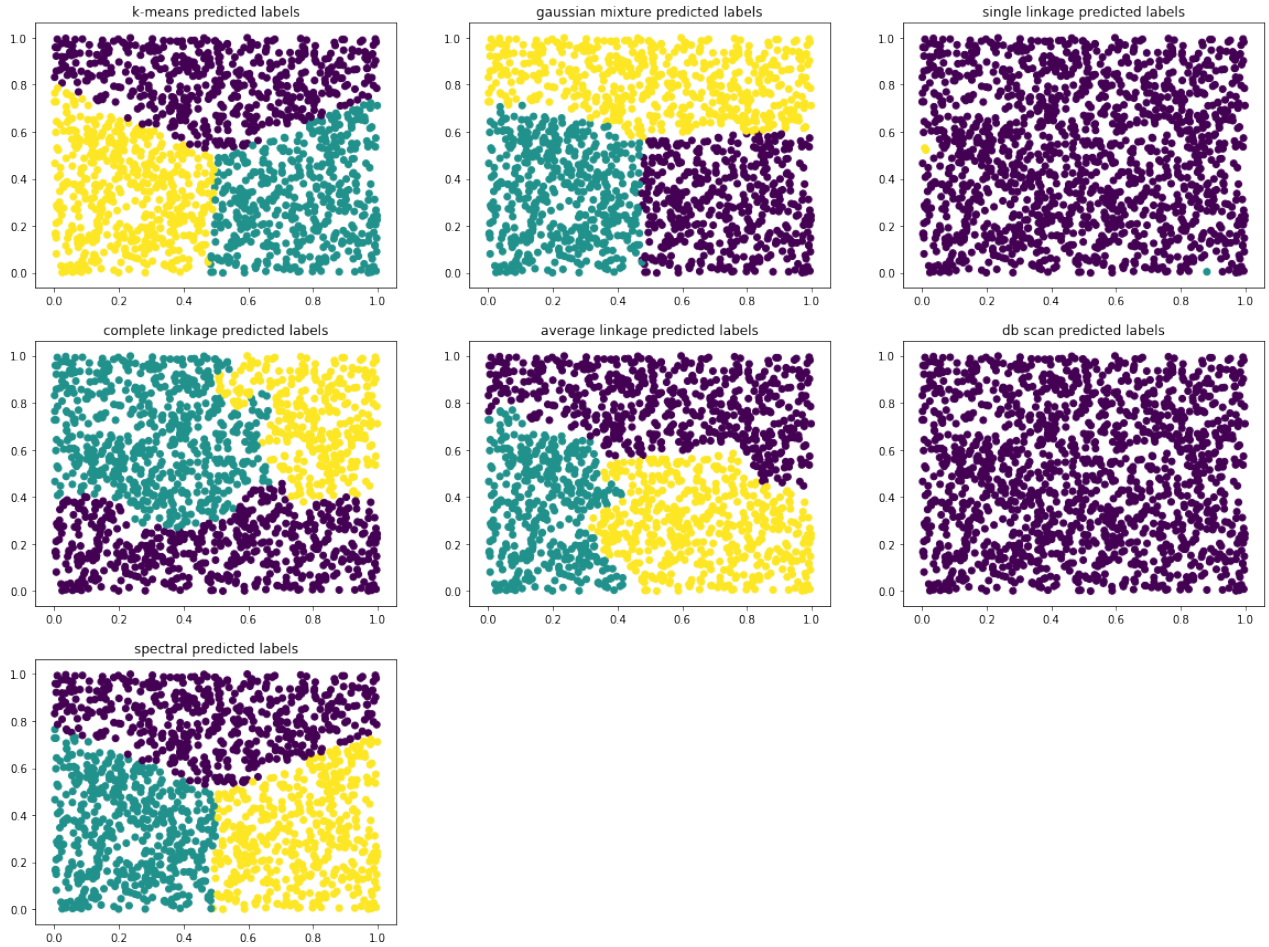
Question 7d: Are your answers for 7(a) and 7(b) consistent with that of (c)? If not, explain the reason for this inconsistency.

Except for Moons2, all the rankings were consistent as per my inference.

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.

In [58]:
```
labels=['k-means','gaussian mixture','single linkage','complete linkage',

kmeans = KMeans(n_clusters=3, random_state=random_state)
kmeans_pred = kmeans.fit_predict(Rand_X)

gmm = GaussianMixture(n_components=3, covariance_type='full')
gmm_pred = gmm.fit_predict(Rand_X)

single_linkage = AgglomerativeClustering(linkage="single", n_clusters=3)
single_pred = single_linkage.fit_predict(Rand_X)

complete_linkage = AgglomerativeClustering(linkage="complete", n_clusters
complete_pred = complete_linkage.fit_predict(Rand_X)
```

```
linkage = AgglomerativeClustering(linkage="average", n_clusters=3)
average_pred = linkage.fit_predict(Rand_X)

dbscan = DBSCAN(eps=1, min_samples=10)
dbscan_pred = dbscan.fit_predict(Rand_X)

spectral = SpectralClustering(n_clusters=3, random_state=random_state)
spectral_pred = spectral.fit_predict(Rand_X)

y_preds=[kmeans_pred,gmm_pred, single_pred, complete_pred, average_pred,

fig = plt.figure(figsize=(20,15))
count = 0;

for y_pred,label in zip(y_preds, labels):
    count=count+1;
    plt.subplot(3,3,count)
    plt.scatter(Rand_X[:, 0], Rand_X[:, 1], c=y_pred)
    plt.title(label+' predicted labels')
plt.show()
```



Irrespective of shape of the data, every clustering algorithm diligently tries to cluster the data.

K-means works on random assignment of representatives(initialization). Since we mentioned k=3, it chose 3 cluster representatives randomly.It does cluster assignment in next step.This is based on distance metric(SSE).Then it recomputes mean of formed cluster and chooses its right representative This process goes on till convergence.In the given random data, three clusters were formed based on proximity and minimum SSE to its representative.

Gaussian mixture models assume cluster to be normally distributed and selects a point based on its probability of belonging to that cluster or not. Therefore it's a random process for random data points.It changes evry time for every run.

singlle link clustering merges points based on minimum distance between points in cluster and to the point/cluster which is to be merged.In random data, every point becomes close to one point or the other in cluster. Therefore,it forms entire data to be one cluster making noise point as single cluster. This shows how error prone single link clustering is.

Complete link considers maximum distance between points. It worked fairly well in case of complete link since it doesnot cluster nearest point but looks if other points in cluster are closer as well. This makes complete link to form clusters.

Average link takes average of distance from and to clusters.The idea behind it is to overcome the shortcomings of single and complete link which considers two extremes of min and max. Therefore we see some better cluster structure in average link.

DBSCAN works on continuity of points. As long as there is enough density, it bags all points in same cluster. We can see the same from random dataset.since points are spread and dense, it considered all the points as one cluster. Separation should be there when we want to cluster DBSCAn. DBSCAN is expected not to work well if points are overlapped.If we increase minimum_samples, it might yield better result in random dataset.

Spectral clustering also comes with notion of affinity matrix. It projects points into new space and then performs k-means clustering(can be any clustering). Therefore random points are projected into new space where we expect better clustering structure. We then apply some clustering alogorithm on those points.In case of rand data, it pretty much looks like k-means.

8. Real-world dataset

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

In [11]:
```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
```

The features are:

In [12]:
```
cancer.feature_names
```

Out[12]:
```
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 [13]:
```
cancer.target_names
```

Out[13]:
```
array(['malignant', 'benign'], dtype='<U9')
```

Create dataset for classification

In [14]:
```
Cancer_X = cancer.data
Cancer_y = cancer.target
```

Size of Cancer_X and Cancer_y

In [15]:
```
Cancer_X.shape
```

Out[15]:
```
(569, 30)
```

In [16]:
```
Cancer_y.shape
```

Out[16]:
```
(569,)
```

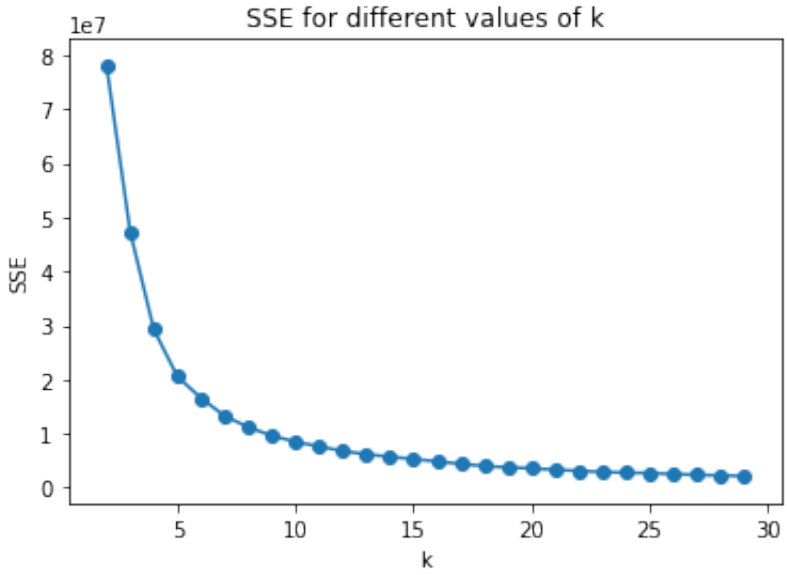
Question 8a: Compute SSE for k = range(2,40), i.e, for k=2,3,4,...,40

In [17]:
```
score = np.zeros(41);
for i in range(1,41):
    kmeans = KMeans(n_clusters=i, random_state=random_state); #Initializi
    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= 1 : 256677243.95
SSE for k= 2 : 77943099.88
SSE for k= 3 : 47285926.9
SSE for k= 4 : 29226541.65
SSE for k= 5 : 20539877.62
SSE for k= 6 : 16558716.7
SSE for k= 7 : 13249736.07
SSE for k= 8 : 11183535.78
SSE for k= 9 : 9609383.58
SSE for k= 10 : 8487166.05
SSE for k= 11 : 7613587.21
SSE for k= 12 : 6784588.86
SSE for k= 13 : 6157087.42
SSE for k= 14 : 5708365.13
SSE for k= 15 : 5286031.4
SSE for k= 16 : 4848940.46
SSE for k= 17 : 4398276.58
SSE for k= 18 : 4009831.04
SSE for k= 19 : 3738118.1
SSE for k= 20 : 3578729.29
SSE for k= 21 : 3312041.59
SSE for k= 22 : 3102392.22
SSE for k= 23 : 2894387.69
SSE for k= 24 : 2768624.68
SSE for k= 25 : 2685795.48
SSE for k= 26 : 2514580.5
SSE for k= 27 : 2362959.95
SSE for k= 28 : 2257591.62
SSE for k= 29 : 2148955.49
SSE for k= 30 : 2036764.0
SSE for k= 31 : 1969448.35
SSE for k= 32 : 1833170.63
SSE for k= 33 : 1791369.0
SSE for k= 34 : 1722589.76
SSE for k= 35 : 1677340.13
SSE for k= 36 : 1656114.54
SSE for k= 37 : 1528956.63
SSE for k= 38 : 1496563.45
SSE for k= 39 : 1417439.02
SSE for k= 40 : 1435813.77
```

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

In [18]:
```
plt.plot(range(2,30),score[2:30])
plt.scatter(range(2,30),score[2:30])
plt.xlabel('k')
plt.ylabel('SSE')
plt.title('SSE for different values of k')
plt.show()
```

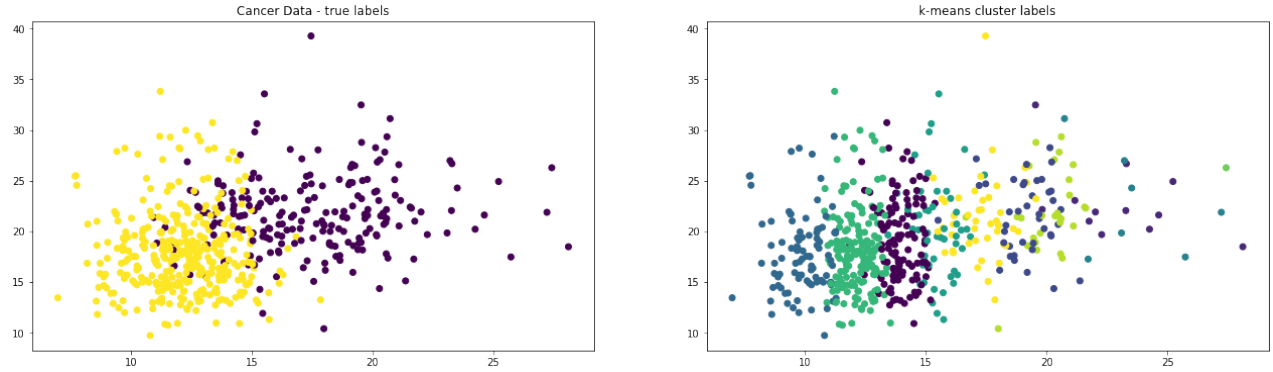


Question 8c: Using this plot, determine the 'k' that you will use to do K-Means clustering.

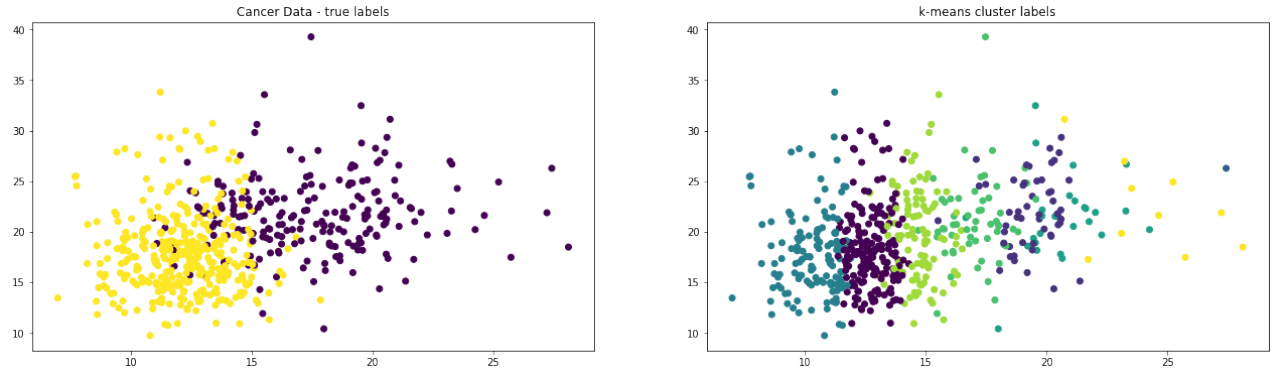
After 10 clusters, there is no substantial reduction is SSE. Therefore I would choose 10 as my number of clusters.

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

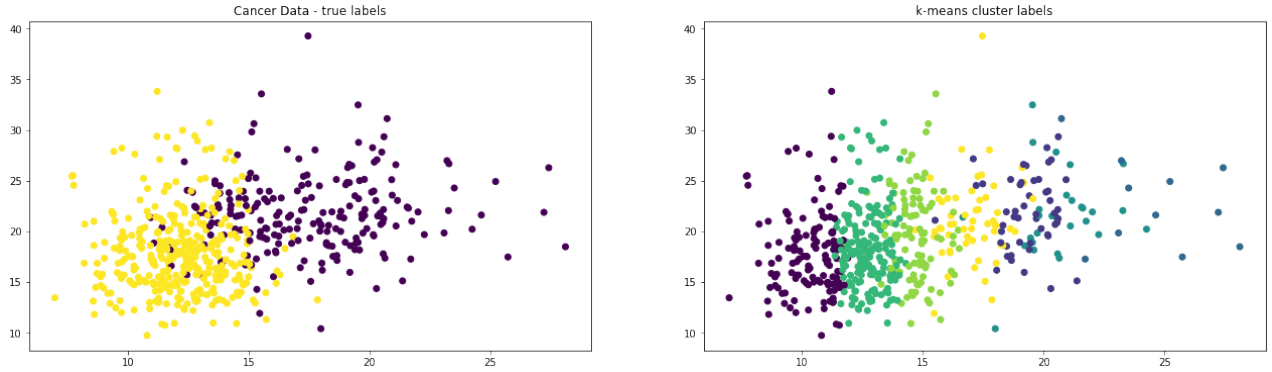
In [19]:
```
n_clusters = 10;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred1 = kmeans.fit_predict(Cancer_X)
fig = plt.figure(figsize=(22,6))
plt.subplot(1,2,1)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=Cancer_y)
plt.title('Cancer Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=y_pred1)
plt.title('k-means cluster labels')
plt.show()
```



In [65]:
```
n_clusters = 8;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred2 = kmeans.fit_predict(Cancer_X)
fig = plt.figure(figsize=(22,6))
plt.subplot(1,2,1)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=Cancer_y)
plt.title('Cancer Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=y_pred2)
plt.title('k-means cluster labels')
plt.show()
```

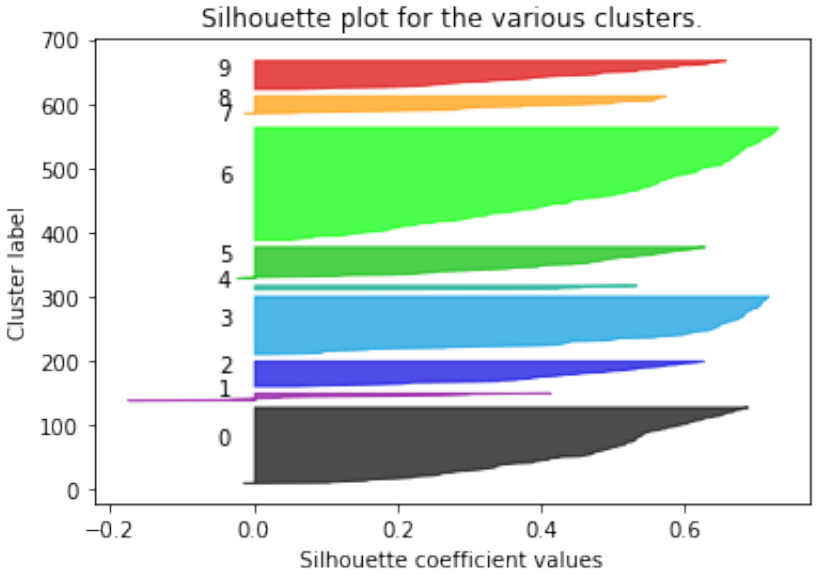


In [66]:
```
n_clusters = 7;
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state);
y_pred3 = kmeans.fit_predict(Cancer_X)
fig = plt.figure(figsize=(22,6))
plt.subplot(1,2,1)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=Cancer_y)
plt.title('Cancer Data - true labels')
plt.subplot(1,2,2)
plt.scatter(Cancer_X[:, 0], Cancer_X[:, 1], c=y_pred3)
plt.title('k-means cluster labels')
plt.show()
```

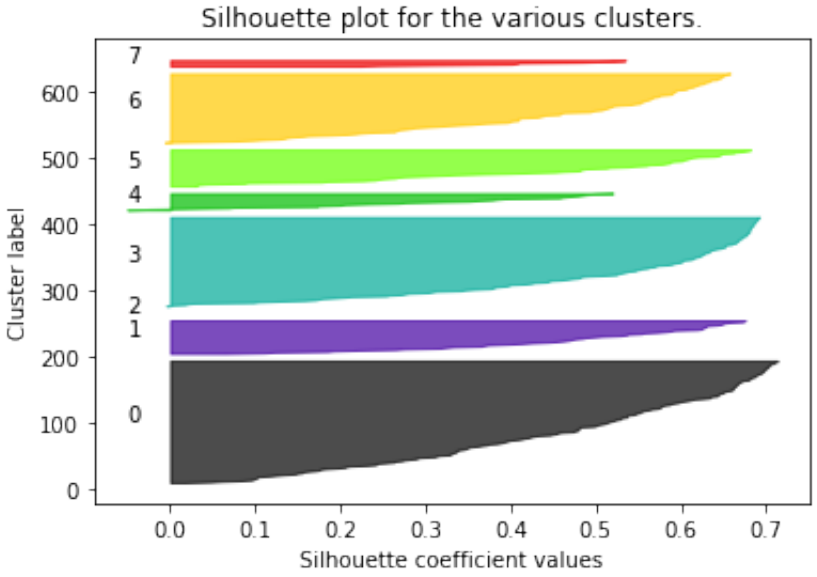


Question 8e: Plot the silhouette values for points in each cluster (using the silhouette() function provided in the practice notebook). .

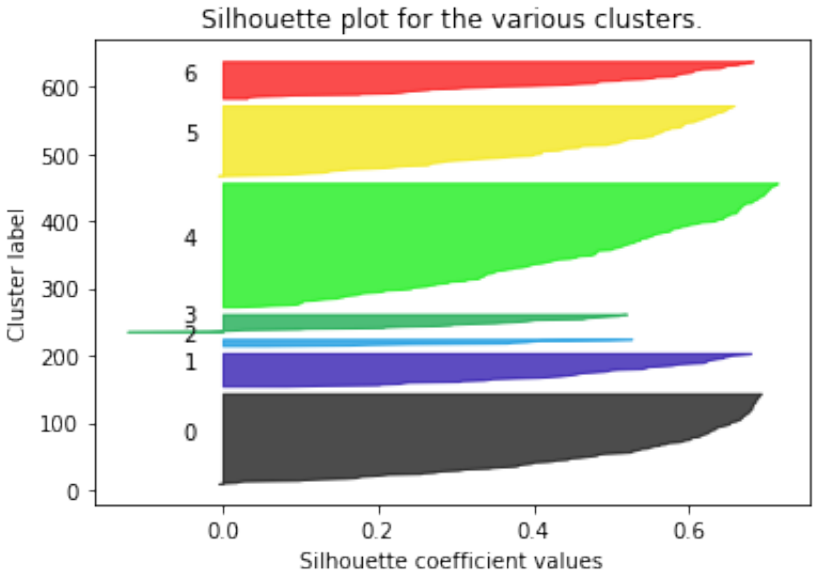
In [81]:
```
silhouette(Cancer_X,y_pred1)
```



In [68]:
```
silhouette(Cancer_X,y_pred2)
```



In [69]:
```
silhouette(Cancer_X,y_pred3)
```



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?

Silhouette coefficient is calculated on single point. It is enumerated on entire cluster by taking average of all points silhouette coefficient. We desire it to be as close as 0 which signifies good separation between clusters.Here, clusters 0,1,2,5,6 are better than other clusters.

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 [33]:
```
rand_index(y_pred1,Cancer_y)
```

Out[33]:
```
0.5989875987029382
```

```
Rand Index defines cluster validity. Since we got rand index to be 59%
approx, we can say that quality of the clusters is not so great. At the
same time we cannot say totally that it's bad clustering.
```

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.

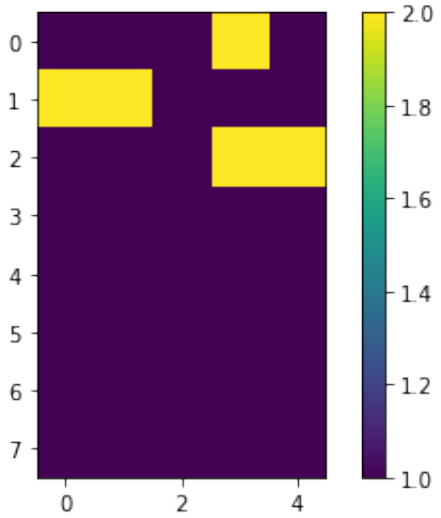
For these range of eps and min_samples values, compute an 8x5 matrix (with rows as eps values and cols as min_samples) to show the number of clusters obtained at each of these parameters. Visualize this matrix using imshow() in matplotlib.

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

y_pred = dbscan.fit_predict(Cancer_X)

max(y_pred)+1

In [31]:
```
eps=[50, 100, 150, 200, 250, 300, 400, 500]
min_samples=[10, 15, 20, 25, 30]


count = 0;
i=0;
rows, columns = 5, 8;
matrix = np.zeros((columns,rows))
j=0
fig,ax=plt.subplots()
for min_sample in min_samples:
    i=0;
    for ep in eps:
        dbscan = DBSCAN(eps=ep, min_samples=min_sample)
        y_pred = dbscan.fit_predict(Cancer_X)
        matrix[i][j]= max(y_pred)+1;
        i=i+1;
    j=j+1;
image=plt.imshow(matrix)
fig.colorbar(image)
plt.show()
```



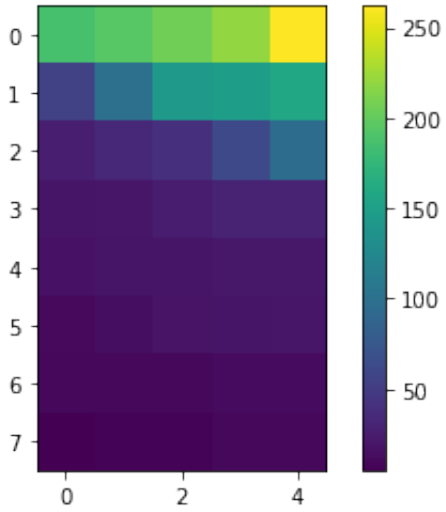
Question 8i: For these range of eps and min_samples values, compute an 8x5 matrix (with rows as eps values and cols as min_samples) to show the number of noise points obtained at each of these parameters. Visualize this matrix using imshow() in matplotlib.

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

y_pred = dbscan.fit_predict(Cancer_X)

sum(y_pred==-1)

In [32]:
```
eps=[50, 100, 150, 200, 250, 300, 400, 500]
min_samples=[10, 15, 20, 25, 30]


count = 0;
i=0;
rows, columns = 5, 8;
matrix = np.zeros((columns,rows))
j=0
fig,ax=plt.subplots()
for min_sample in min_samples:
    i=0;
    for ep in eps:
        dbscan = DBSCAN(eps=ep, min_samples=min_sample)
        y_pred = dbscan.fit_predict(Cancer_X)
        matrix[i][j]= sum(y_pred==-1);
        i=i+1;
    j=j+1;

image=plt.imshow(matrix)
fig.colorbar(image)
plt.show()
```



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)?

```
We can say that as eps increase, noise points decrease since it covers
huge area. But this makes the entire data into one huge cluster.As eps
decrease and sample points increase, majority of points are left out as
noise points since we cannot guarantee density of points every where.
```

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 [27]:
```
dbscan = DBSCAN(eps=100, min_samples=10)
y_pred = dbscan.fit_predict(Cancer_X)
for n, i in enumerate(y_pred):
        if i == -1:
          y_pred[n] = 5
rand_index(Cancer_y,y_pred)
```

Out[27]:
```
0.6680115844451595
```

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:

```
For k-means, rand index is around 59% where DBSCAN gets around 66%.
Therefore we can say that DBSCAN works better
on this data set. By this we can say that data set had good continuity
of points without much globular structures in data.At the same time we
can also tell that there is not very much separation in the data. It is
well spread but not much separation cluster wise.
```