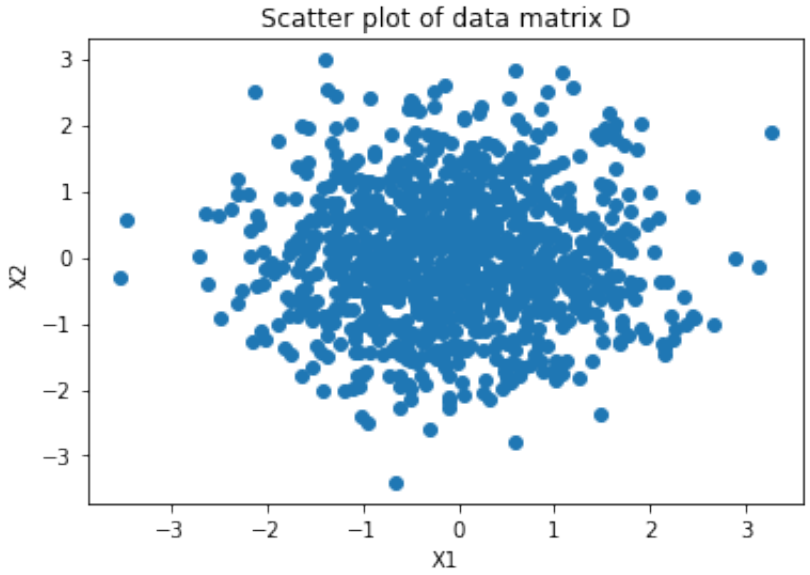
In [11]:
```
import numpy as np

mu = np.array([0,0])
Sigma = np.array(([1,0], [0,1]))
X1, X2 = np.random.multivariate_normal(mu, Sigma, 1000).T
D = np.array([X1, X2]).T
```

In [12]:
```
##################  1  ##################
```

In [13]:
```
import matplotlib.pyplot as plt
plt.scatter(D[:, 0], D[:,1])
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Scatter plot of data matrix D')
```

Out[13]:
```
Text(0.5, 1.0, 'Scatter plot of data matrix D')
```



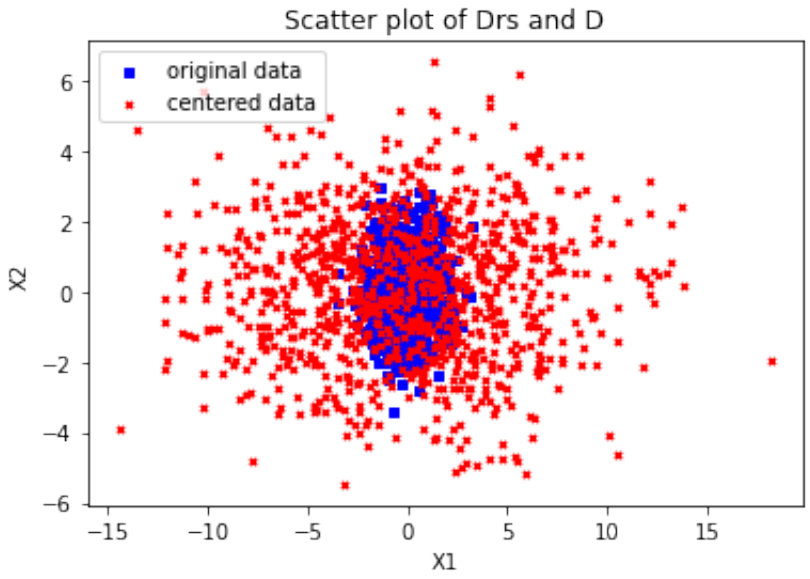
In [14]:
```
##################  2  ##################
```

In [15]:
```
import math
piOver4 = math.pi/4
R = [[math.cos(piOver4),-(math.sin(piOver4))],
    [math.sin(piOver4), math.cos(piOver4)]]
S = [[5,0],
     [0,2]]

Rs = np.dot(R, S)
Drs = np.matmul(D,Rs)
```

In [16]:
```
##################  A  ##################
```

In [17]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(D[:,0], D[:,1], s=10, c='b', marker='s', label='original data')
ax.scatter(Drs[:,0], Drs[:,1], s=10, c='r', marker='x', label='centered data'
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend(loc='upper left')
plt.title('Scatter plot of Drs and D')
```

Out[17]:
```
Text(0.5, 1.0, 'Scatter plot of Drs and D')
```



In [18]:
```
##################  B  ##################
```

In [19]:
```
np.cov(Drs, ddof=1)
```

Out[19]:
```
array([[  0.13715095,  -1.02662002,   0.20408674, ...,   1.75243313,
          0.90075037,  -0.31942   ],
       [ -1.02662002,   7.68458865,  -1.52765638, ..., -13.11753867,
         -6.74241292,   2.39096382],
       [  0.20408674,  -1.52765638,   0.30369017, ...,   2.60769867,
          1.34035673,  -0.47531121],
       ...,
       [  1.75243313, -13.11753867,   2.60769867, ...,  22.39154607,
         11.50925134,  -4.0813584 ],
       [  0.90075037,  -6.74241292,   1.34035673, ...,  11.50925134,
          5.91575347,  -2.09781761],
       [ -0.31942   ,   2.39096382,  -0.47531121, ...,  -4.0813584 ,
         -2.09781761,   0.74391854]])
```

In [20]:
```
##################  C  ##################
```

In [21]:
```
np.var(Drs)
```

Out[21]:
```
14.724273745636431
```

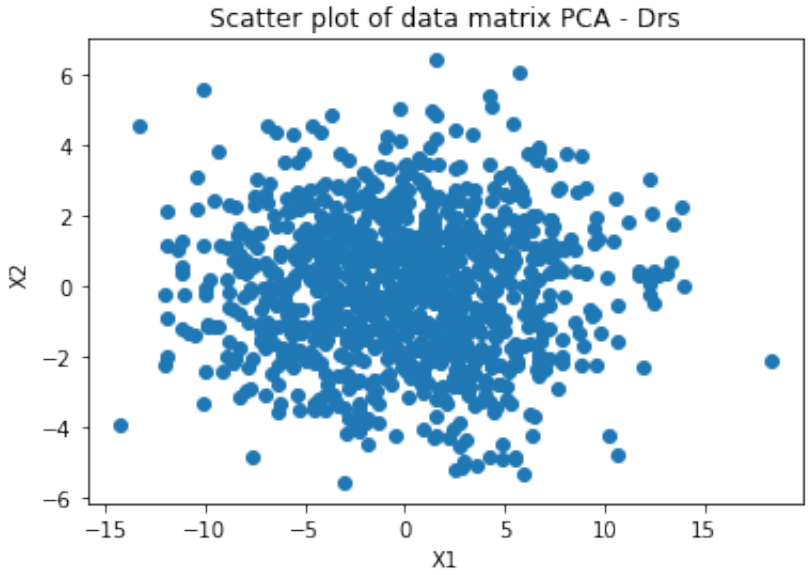
In [22]:
```
##################  3  ##################
```

In [23]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pcaDrs = pca.fit_transform(Drs)
```

In [24]:
```
##################  A  ##################
```

In [25]:
```
plt.scatter(pcaDrs[:,0], pcaDrs[:,1])
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Scatter plot of data matrix PCA - Drs')
```

Out[25]:
```
Text(0.5, 1.0, 'Scatter plot of data matrix PCA - Drs')
```



In [26]:
```
##################  B  ##################
```

In [27]:
```
np.cov(pcaDrs, ddof=1)
```

Out[27]:
```
array([[  0.04042889,  -0.5906395 ,   0.07583799, ...,   0.92251921,
          0.45434892,  -0.20640037],
       [ -0.5906395 ,   8.62885511,  -1.10794329, ..., -13.47739967,
         -6.63773935,   3.01537381],
       [  0.07583799,  -1.10794329,   0.1422597 , ...,   1.73049546,
          0.85228442,  -0.3871734 ],
       ...,
       [  0.92251921, -13.47739967,   1.73049546, ...,  21.05033631,
         10.36747808,  -4.70970917],
       [  0.45434892,  -6.63773935,   0.85228442, ...,  10.36747808,
          5.10607528,  -2.3195737 ],
       [ -0.20640037,   3.01537381,  -0.3871734 , ...,  -4.70970917,
         -2.3195737 ,   1.0537295 ]])
```

In [28]:
```
##################  C  ##################
```

In [29]:
```
np.var(pcaDrs[:,0])
```

Out[29]:
```
25.37935554613441
```

In [30]:
```
np.var(pcaDrs[:,1])
```

Out[30]:
```
4.042242996950797
```

In [31]:
```
##################  4  ##################
```

In [32]:
```
from sklearn.datasets import load_boston
data = load_boston()
```

```
/Users/jayforbes/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

In [33]:
```
import pandas as pd
import numpy as np


data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
pca = PCA()

D_pca = pca.fit_transform(data)
```

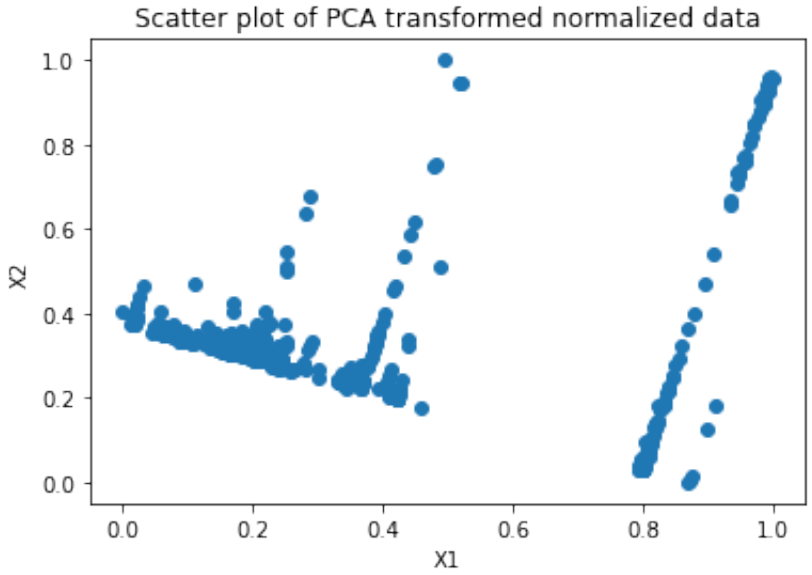
In [34]:
```
##################  A  ##################
```

In [61]:
```
from sklearn.preprocessing import MinMaxScaler
Dn = MinMaxScaler().fit_transform(D_pca)
Dn
```

Out[61]:
```
array([[0.18213887, 0.29839122, 0.33344586, ..., 0.44821065, 0.15407723,
        0.50300154],
       [0.0973027 , 0.33686188, 0.14571541, ..., 0.38390231, 0.13445192,
        0.299447  ],
       [0.09657391, 0.34659061, 0.24110143, ..., 0.28322808, 0.12395942,
        0.3868602 ],
       ...,
       [0.15003732, 0.31433814, 0.10149585, ..., 0.31629246, 0.12613992,
        0.5321277 ],
       [0.15156213, 0.32247027, 0.11031379, ..., 0.33653927, 0.13384446,
        0.53828939],
       [0.14864877, 0.31451231, 0.15257926, ..., 0.45558661, 0.15490044,
        0.53969636]])
```

In [62]:
```
import matplotlib.pyplot as plt
plt.scatter(Dn[:,0], Dn[:,1])
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Scatter plot of PCA transformed normalized data')
```

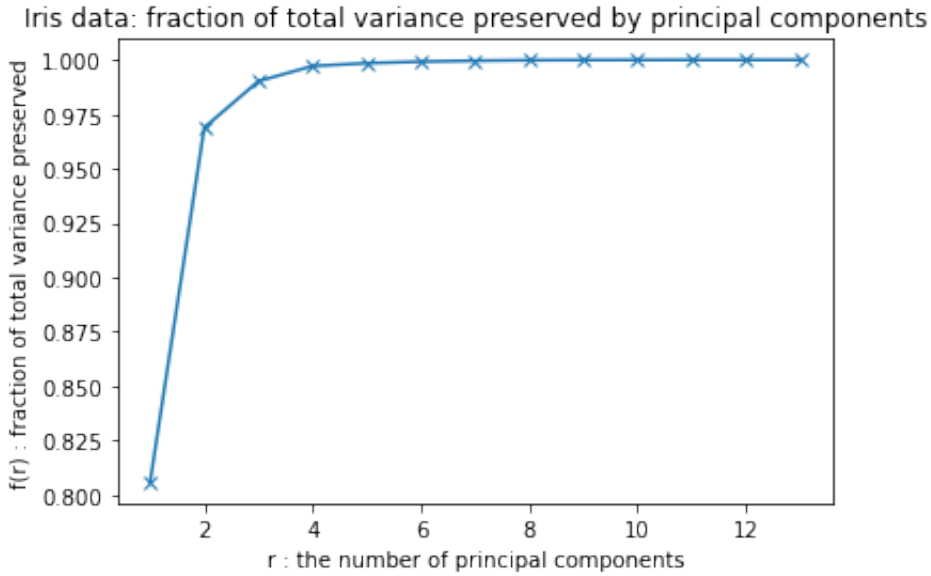
Out[62]:
```
Text(0.5, 1.0, 'Scatter plot of PCA transformed normalized data')
```



In [63]:
```
##################  B  ##################
```

In [64]:
```
var_ratio = pca.explained_variance_ratio_
var_ratio
plt.plot(range(1,len(var_ratio)+1), np.cumsum(var_ratio), marker='x')
plt.title('Iris data: fraction of total variance preserved by principal compo
plt.xlabel('r : the number of principal components')
plt.ylabel('f(r) : fraction of total variance preserved')
```

Out[64]:
```
Text(0, 0.5, 'f(r) : fraction of total variance preserved')
```



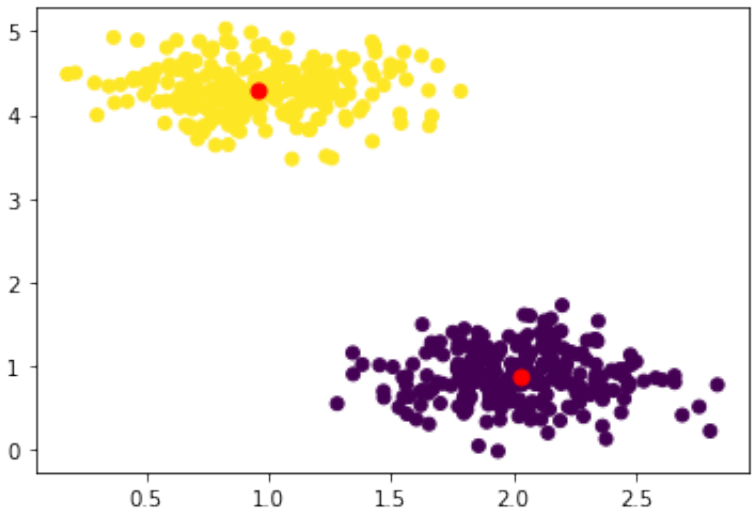
In [65]:
```
##################  C  ##################

##################  I ##################

2

##################  II  ##################

2
```

Out[65]:
```
2
```

In [79]:
```
##################  D  ##################
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
Dn, labels = make_blobs(n_samples=500, centers=2, cluster_std=.3, random_stat
```

In [80]:
```
kmeans = KMeans(n_clusters=2, init='random', max_iter=300, random_state=0)
pred_labels = kmeans.fit_predict(Dn)
centers = kmeans.cluster_centers_
plt.scatter(Dn[:,0], Dn[:,1], c=pred_labels)
plt.scatter(centers[:,0], centers[:,1], s=50, c='red')
```

Out[80]:
```
<matplotlib.collections.PathCollection at 0x7fcaa313ddc0>
```



In [82]:
```
##################  E  ##################
```

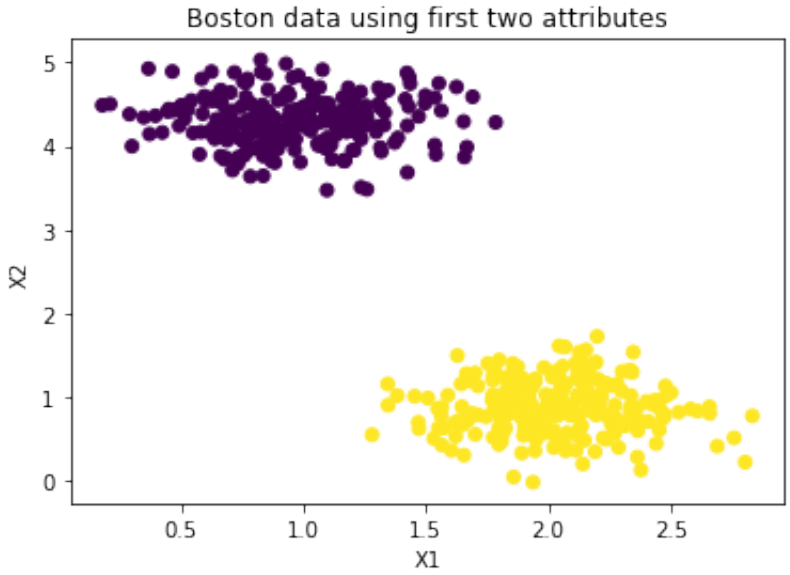
In [83]:
```
from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=.3, min_samples=5)
pred_labels = dbs.fit_predict(Dn)
pred_labels
```

Out[83]:
```
array([ 0,  1,  0,  0,  1,  0,  0,  0,  1,  0,  0,  0,  0,  1,  0,  1,  0,
        1,  1,  0,  0,  1,  1,  0,  1,  1,  0,  1,  0,  0,  1,  0,  0,  1,
        0,  0,  1,  0,  0,  1,  0,  0,  1,  0,  1,  0,  1,  1,  0,  0,  0,
        0,  0,  1,  0,  0,  0,  0,  0,  0,  1,  1,  0,  0,  1,  1,  0,  1,
        0,  1,  0,  0,  1,  1,  1,  0,  1,  0,  1,  0,  0,  0,  0,  0,  1,
        0,  1,  1,  0,  1,  1,  1,  1,  0,  1,  0,  0,  1,  0,  0,  1,  1,
        1,  1,  1,  0,  0,  0,  1,  0,  1,  1,  0,  0,  1,  1,  1,  1,  0,
        0,  1,  0,  0,  0,  0,  0,  1,  1,  0,  1,  1,  1,  0,  0,  0,  0,
        1,  1,  0,  0,  0,  0,  1,  1,  1,  0,  0,  1,  0,  1,  1,  0,  0,
        1,  0,  1,  1,  0,  0,  1,  0,  1,  1,  1,  1,  0,  0,  1,  1,  1,
        1,  0,  1,  1,  1,  0,  1,  0,  0,  0,  1,  1,  0,  1,  1,  0,  1,
        1,  0,  1,  1,  1,  0,  0,  1,  0,  1,  1,  1,  0,  0,  0,  1,  1,
        1,  1,  0,  1,  1,  1,  1,  0,  0,  0,  0,  1,  1,  0,  0,  1,  1,
        0,  1,  1,  0,  0,  1,  0,  0,  1,  0,  1,  0,  1, -1,  1,  1,  1,
        0,  1,  1,  0,  1,  1,  0,  1,  0,  1,  0,  0,  1,  0,  1,  1,  0,
        1,  0,  0,  1,  0,  0,  1,  0,  1,  0,  1,  1,  1,  0,  0,  0,  0,
        0,  1,  0,  1,  1,  0,  0,  0,  0,  0,  1,  1,  0,  1,  1,  1,  0,
        0,  0,  0,  1,  0, -1,  1,  1,  1,  0,  0,  0,  0,  1,  1,  1,  1,
        0,  0,  0,  1,  1,  1,  0,  0,  1,  1,  0,  0,  0,  0,  0,  0,  1,
        1,  0,  1,  1,  0,  0,  0,  1,  0,  1,  0,  1,  0,  0,  0,  0,  0,
        0,  0,  1,  0,  0,  1,  1,  1,  0,  0,  0,  1,  1,  0,  1,  1,  0,
        0,  0,  1,  0,  1,  0,  1,  1,  0,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  1,  1,  1,
        0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  1,  0,  0,  0,  0,  1,
        0,  0,  0,  1,  1,  0,  1,  0,  0,  1,  0,  1,  0,  1,  0,  0,  1,
        0,  0,  0,  0,  1,  0,  1,  1,  0,  1,  0,  1,  1,  1,  1,  1,  1,
        0,  1,  1,  1,  1,  1,  1,  0,  1,  0,  1,  0,  0,  0,  1,  1,  0,
        1,  1,  0,  0,  1,  1,  1,  1,  1,  1,  0,  1,  1,  1,  0,  1,  1,
        0,  0,  1,  1,  0,  1,  1,  0,  1,  0,  0,  1,  1,  0,  0,  1,  0,
        1,  1,  0,  1,  0,  1,  0])
```

In [84]:
```
dbs = DBSCAN(eps=.5, min_samples=5)
pred_labels = dbs.fit_predict(Dn)
```

In [85]:
```
plt.scatter(Dn[:,0], Dn[:,1], c=labels)
plt.title('Boston data using first two attributes')
plt.xlabel('X1')
plt.ylabel('X2')
```

Out[85]: Text(0, 0.5, 'X2')



In [86]:
```
#2 clusters found
```