In [2]:
```
import pandas as pd
from sklearn. datasets import load_digits
digits = load_digits()
dir(digits)
```

Out[2]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

In [3]:
```
digits.keys()
```

Out[3]:
```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'image
s', 'DESCR'])
```

In [13]:
```
digits
```

```
Out[13]: {'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ..., 10.,  0.,  0.],
                [ 0.,  0.,  0., ..., 16.,  9.,  0.],
                ...,
                [ 0.,  0.,  1., ...,  6.,  0.,  0.],
                [ 0.,  0.,  2., ..., 12.,  0.,  0.],
                [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
          'target': array([0, 1, 2, ..., 8, 9, 8]),
          'frame': None,
          'feature_names': ['pixel_0_0',
           'pixel_0_1',
           'pixel_0_2',
           'pixel_0_3',
           'pixel_0_4',
           'pixel_0_5',
           'pixel_0_6',
           'pixel_0_7',
           'pixel_1_0',
           'pixel_1_1',
           'pixel_1_2',
           'pixel_1_3',
           'pixel_1_4',
           'pixel_1_5',
           'pixel_1_6',
           'pixel_1_7',
           'pixel_2_0',
           'pixel_2_1',
           'pixel_2_2',
           'pixel_2_3',
           'pixel_2_4',
           'pixel_2_5',
           'pixel_2_6',
           'pixel_2_7',
           'pixel_3_0',
           'pixel_3_1',
           'pixel_3_2',
           'pixel_3_3',
           'pixel_3_4',
           'pixel_3_5',
           'pixel_3_6',
           'pixel_3_7',
           'pixel_4_0',
           'pixel_4_1',
           'pixel_4_2',
           'pixel_4_3',
           'pixel_4_4',
           'pixel_4_5',
           'pixel_4_6',
           'pixel_4_7',
           'pixel_5_0',
           'pixel_5_1',
           'pixel_5_2',
           'pixel_5_3',
           'pixel_5_4',
           'pixel_5_5',
           'pixel_5_6',
           'pixel_5_7',
           'pixel_6_0',
           'pixel_6_1',
           'pixel_6_2',
```

```
  'pixel_6_3',
  'pixel_6_4',
  'pixel_6_5',
  'pixel_6_6',
  'pixel_6_7',
  'pixel_7_0',
  'pixel_7_1',
  'pixel_7_2',
  'pixel_7_3',
  'pixel_7_4',
  'pixel_7_5',
  'pixel_7_6',
  'pixel_7_7'],
 'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ..., 15.,  5.,  0.],
         [ 0.,  3., 15., ..., 11.,  8.,  0.],
         ...,
         [ 0.,  4., 11., ..., 12.,  7.,  0.],
         [ 0.,  2., 14., ..., 12.,  0.,  0.],
         [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

        [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
         [ 0.,  0.,  0., ...,  9.,  0.,  0.],
         [ 0.,  0.,  3., ...,  6.,  0.,  0.],
         ...,
         [ 0.,  0.,  1., ...,  6.,  0.,  0.],
         [ 0.,  0.,  1., ...,  6.,  0.,  0.],
         [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

        [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
         [ 0.,  0.,  3., ..., 14.,  0.,  0.],
         [ 0.,  0.,  8., ..., 16.,  0.,  0.],
         ...,
         [ 0.,  9., 16., ...,  0.,  0.,  0.],
         [ 0.,  3., 13., ..., 11.,  5.,  0.],
         [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

        ...,

        [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ...,  2.,  1.,  0.],
         [ 0.,  0., 16., ..., 16.,  5.,  0.],
         ...,
         [ 0.,  0., 16., ..., 15.,  0.,  0.],
         [ 0.,  0., 15., ..., 16.,  0.,  0.],
         [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

        [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
         [ 0.,  0., 14., ..., 15.,  1.,  0.],
         [ 0.,  4., 16., ..., 16.,  7.,  0.],
         ...,
         [ 0.,  0.,  0., ..., 16.,  2.,  0.],
         [ 0.,  0.,  4., ..., 16.,  2.,  0.],
         [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

        [[ 0.,  0., 10., ...,  1.,  0.,  0.],
         [ 0.,  2., 16., ...,  1.,  0.,  0.],
         [ 0.,  0., 15., ..., 15.,  0.,  0.],
         ...,
```

```
        [ 0.,  4., 16., ..., 16.,  6.,  0.],
        [ 0.,  8., 16., ..., 16.,  8.,  0.],
        [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits dat
aset\n--------------------------------------------------\n\n**Data Set Characte
ristics:**\n\n    :Number of Instances: 1797\n    :Number of Attributes: 64\n
:Attribute Information: 8x8 image of integer pixels in the range 0..16.\n    :M
issing Attribute Values: None\n    :Creator: E. Alpaydin (alpaydin '@' boun.ed
u.tr)\n    :Date: July; 1998\n\nThis is a copy of the test set of the UCI ML ha
nd-written digits datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Rec
ognition+of+Handwritten+Digits\n\nThe data set contains images of hand-written
digits: 10 classes where\neach class refers to a digit.\n\nPreprocessing progra
ms made available by NIST were used to extract\nnormalized bitmaps of handwritt
en digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to
the training set and different 13\nto the test set. 32x32 bitmaps are divided i
nto nonoverlapping blocks of\n4x4 and the number of on pixels are counted in ea
ch block. This generates\nan input matrix of 8x8 where each element is an integ
er in the range\n0..16. This reduces dimensionality and gives invariance to sma
ll\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris,
J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Jane
t, and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 546
9,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Methods of Combining
Multiple Classifiers and Their\n    Applications to Handwritten Digit Recogniti
on, MSc Thesis, Institute of\n    Graduate Studies in Science and Engineering,
Bogazici University.\n  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers,
Kybernetika.\n  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Q
in.\n    Linear dimensionalityreduction using relevance weighted LDA. School of
\n    Electrical and Electronic Engineering Nanyang Technological University.\n
2005.\n  - Claudio Gentile. A New Approximate Maximal Margin Classification\n
Algorithm. NIPS. 2000.\n"}
```

In [10]:
```
digits.data.shape
```

Out[10]:
```
(1797, 64)
```

In [14]:
```
digits.data[0]
```

Out[14]:
```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

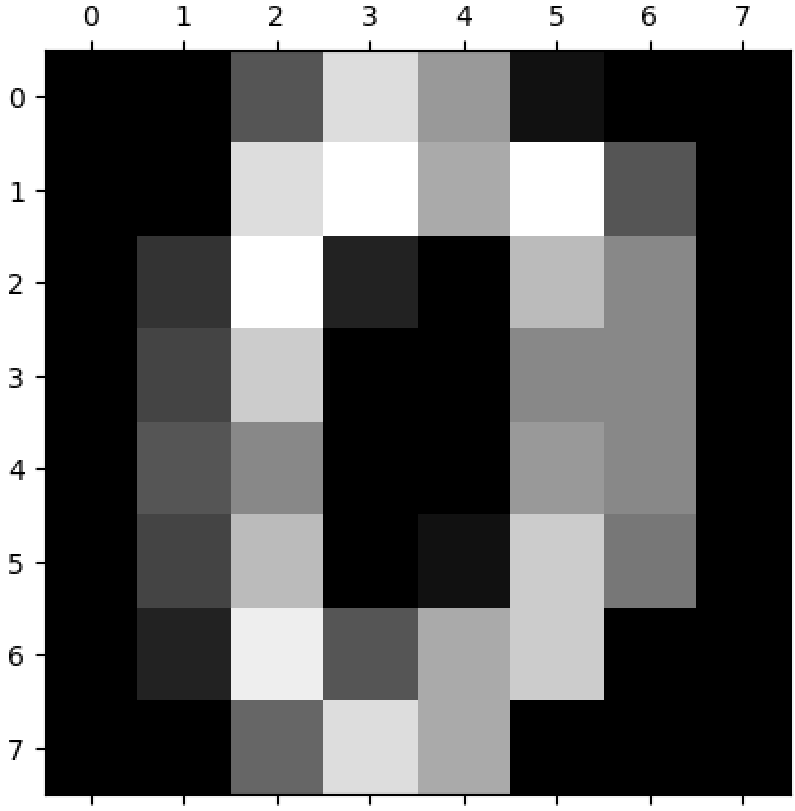
In [15]:
```
digits.data[0].reshape(8,8)
```

Out[15]:
```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

In [16]:
```
import matplotlib .pyplot as pp
pp.gray()
pp.matshow(digits.data[0].reshape(8,8))
```

Out[16]:
```
<matplotlib.image.AxesImage at 0x229274d7dd8>
```

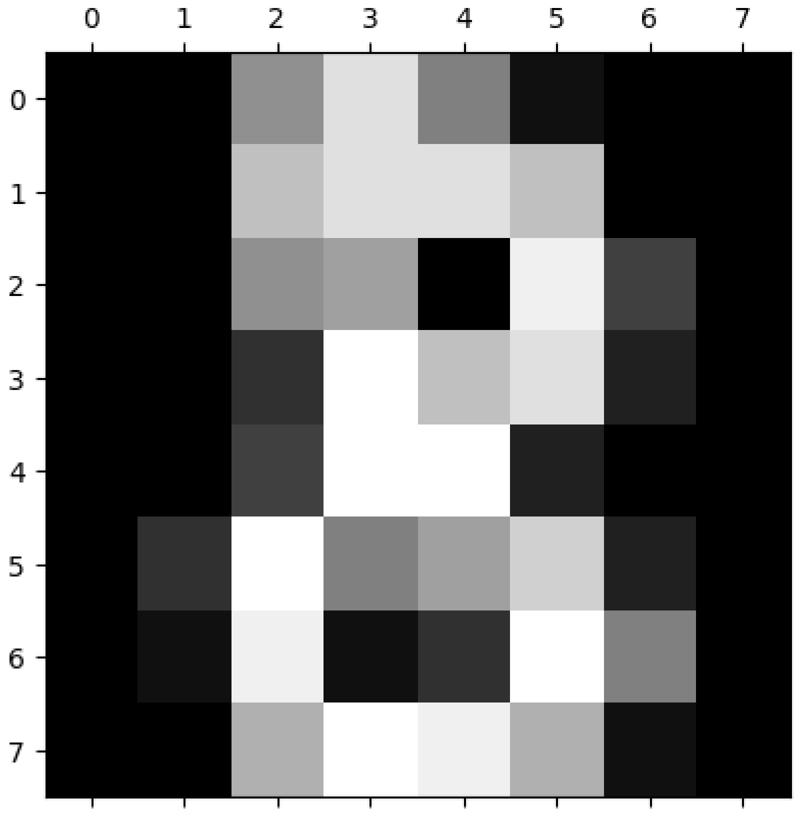
<Figure size 640x480 with 0 Axes>



In [17]:
```
import matplotlib .pyplot as pp
pp.gray()
pp.matshow(digits.data[8].reshape(8,8))
```

Out[17]: <matplotlib.image.AxesImage at 0x2292970ac88>

<Figure size 640x480 with 0 Axes>



In [20]:
```
df= pd.DataFrame(digits.data ,columns = digits.feature_names)
df
```

Out[20]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	.
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.

1797 rows × 64 columns

In [21]:
```
digits.target[:5]
```

Out[21]:
```
array([0, 1, 2, 3, 4])
```

In [22]:
```
digits.target
```

Out[22]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

In [23]:
```
X= df
y=digits.target
```

In [38]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaler= scaler.fit_transform(X)
X_scaler
```

Out[38]:
```
array([[ 0.        , -0.33501649, -0.04308102, ..., -1.14664746,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  0.54856067,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  1.56568555,
         1.6951369 , -0.19600752],
       ...,
       [ 0.        , -0.33501649, -0.88456568, ..., -0.12952258,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -0.67419451, ...,  0.8876023 ,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649,  1.00877481, ...,  0.8876023 ,
        -0.26113572, -0.19600752]])
```

In [28]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =train_test_split(X_scaler,y, test_size=0.2 )
```

In [29]:
```
from sklearn.linear_model import LogisticRegression
lg = LogisticRegression()
lg.fit(X_train,y_train)
lg.score (X_test,y_test)
```

```
c:\users\sriha\appdata\local\programs\python\python37\lib\site-packages\sklearn\l
inear_model\_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (statu
s=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Out[29]: 0.9722222222222222

Use PCA to reduce dimensions

In [30]:
```
X
```

Out[30]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	.
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.

1797 rows × 64 columns

Use components such that 95% of variance is retained

In [54]:
```
from sklearn.decomposition import PCA
pca = PCA(0.95)
X_pca =pca.fit_transform(X)
X_pca
```

Out[54]:
```
array([[ -1.25946645,  21.27488348,  -9.46305462, ...,   3.67072108,
         -0.9436689 ,  -1.13250195],
       [  7.9576113 , -20.76869896,   4.43950604, ...,   2.18261819,
         -0.51022719,   2.31354911],
       [  6.99192297,  -9.95598641,   2.95855808, ...,   4.22882114,
          2.1576573 ,   0.8379578 ],
       ...,
       [ 10.8012837 ,  -6.96025223,   5.59955453, ...,  -3.56866194,
          1.82444444,   3.53885886],
       [ -4.87210009,  12.42395362, -10.17086635, ...,   3.25330054,
          0.95484174,  -0.93895602],
       [ -0.34438963,   6.36554919,  10.77370849, ...,  -3.01636722,
          1.29752723,   2.58810313]])
```

In [55]:
```
digits.data.shape
```

Out[55]:
```
(1797, 64)
```

In [56]:
```
X_pca.shape
```

Out[56]:
```
(1797, 29)
```

In [57]:
```
pca.explained_variance_ratio_
```

Out[57]:
```
array([0.14890594, 0.13618771, 0.11794594, 0.08409979, 0.05782415,
       0.0491691 , 0.04315987, 0.03661373, 0.03353248, 0.03078806,
       0.02372341, 0.02272697, 0.01821863, 0.01773855, 0.01467101,
       0.01409716, 0.01318589, 0.01248138, 0.01017718, 0.00905617,
       0.00889538, 0.00797123, 0.00767493, 0.00722904, 0.00695889,
       0.00596081, 0.00575615, 0.00515158, 0.0048954 ])
```

In [58]:
```
pca.explained_variance_
```

Out[58]:
```
array([179.0069301 , 163.71774688, 141.78843909, 101.1003752 ,
        69.51316559,  59.10852489,  51.88453911,  44.01510667,
        40.31099529,  37.0117984 ,  28.51904118,  27.32116981,
        21.90148814,  21.32435654,  17.63672222,  16.94686385,
        15.85138991,  15.00446022,  12.23447318,  10.88685932,
        10.69356625,   9.58259779,   9.2264026 ,   8.69036872,
         8.3656119 ,   7.16577961,   6.91973881,   6.19295508,
         5.88499123])
```

In [59]:
```
pca.n_components_
```

Out[59]:
```
29
```

In [60]:
```
pca.n_components
```

Out[60]:
```
0.95
```

PCA created 29 components out of 64 original columns

In [61]:
```
X_pca
```

Out[61]:
```
array([[ -1.25946645,  21.27488348,  -9.46305462, ...,   3.67072108,
         -0.9436689 ,  -1.13250195],
       [  7.9576113 , -20.76869896,   4.43950604, ...,   2.18261819,
         -0.51022719,   2.31354911],
       [  6.99192297,  -9.95598641,   2.95855808, ...,   4.22882114,
          2.1576573 ,   0.8379578 ],
       ...,
       [ 10.8012837 ,  -6.96025223,   5.59955453, ...,  -3.56866194,
          1.82444444,   3.53885886],
       [ -4.87210009,  12.42395362, -10.17086635, ...,   3.25330054,
          0.95484174,  -0.93895602],
       [ -0.34438963,   6.36554919,  10.77370849, ...,  -3.01636722,
          1.29752723,   2.58810313]])
```

In [62]:
```
X_train_pca,X_test_pca,y_train,y_test =train_test_split(X_pca,y, test_size= 0.2)
```

In [64]:
```
from sklearn.linear_model import LogisticRegression
lr_1 =LogisticRegression(max_iter=1000)
lr_1.fit (X_train_pca,y_train)
lr_1. score (X_test_pca,y_test)
```

Out[64]: 0.95

Let's now select only two components

In [65]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
X_pca.shape
```

Out[65]: (1797, 2)

In [66]:
```
X_pca
```

Out[66]:
```
array([[ -1.25946565,  21.27487115],
       [  7.95761012, -20.76868682],
       [  6.99192197,  -9.95597322],
       ...,
       [ 10.80128664,  -6.9602787 ],
       [ -4.87209797,  12.42393803],
       [ -0.34438551,   6.36551308]])
```

In [67]:
```
pca.explained_variance_ratio_
```

Out[67]: array([0.14890594, 0.13618771])

In [68]:
```
pca.explained_variance_
```

Out[68]: array([179.0069301 , 163.71774688])

You can see that both combined retains 0.14+0.13=0.27 or 27% of important feature information

In [69]:
```
X_train_pca,X_test_pca,y_train,y_test =train_test_split(X_pca,y,test_size=0.2)
```

In [70]:
```
lr_1= LogisticRegression(max_iter=1000)
lr_1.fit(X_train_pca,y_train)
lr_1.score(X_test_pca,y_test)
```

Out[70]:
```
0.6333333333333333
```

We get less accuancy (~60%) as using only 2 components did not retain much of the feature information. However in real life you will find many cases where using 2 or few PCA components can still give you a pretty good accuracy