In [8]:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

In [9]:

```
data = datasets.fetch_lfw_people(min_faces_per_person=5)
data
```

Out[9]:

```
{'data': array([[ 83.       ,  94.666664,  78.666664, ..., 241.       , 233.33
333 ,
         230.66667 ],
        [249.       , 249.66667 , 249.66667 , ...,  54.       ,  48.666668,
          43.666668],
        [ 59.666668,  70.333336,  79.333336, ..., 115.333336,  78.666664,
          74.666664],
        ...,
        [110.666664,  88.666664,  59.666668, ..., 104.       ,  39.666668,
          44.       ],
        [ 56.       ,  61.666668,  79.333336, ...,   8.666667,   9.       ,
           8.666667],
        [ 30.666666,  43.       ,  55.       , ...,  17.333334,  16.666666,
          20.666666]], dtype=float32),
 'images': array([[[ 83.       ,  94.666664,  78.666664, ...,  54.       ,
           81.666664,  82.333336],
         [ 92.333336,  97.333336,  80.333336, ...,  52.666668,
           73.333336,  81.333336],
         [ 98.666664,  95.666664,  79.333336, ...,  60.333332,
           74.       ,  78.333336],
         ...,
         [ 38.       ,  35.333332,  34.666668, ..., 240.66667 ,
          230.66667 , 223.66667 ],
         [ 25.       ,  28.333334,  31.       , ..., 241.       ,
          231.66667 , 224.66667 ],
         [ 24.666666,  29.       ,  31.       , ..., 241.       ,
          233.33333 , 230.66667 ]],

        [[249.       , 249.66667 , 249.66667 , ...,  79.666664,
           62.666668,  55.333332],
         [249.       , 249.66667 , 249.66667 , ...,  81.666664,
           62.666668,  53.       ],
         [249.       , 249.33333 , 249.33333 , ...,  85.       ,
           64.333336,  47.666668],
         ...,
         [ 22.       ,  36.333332,  55.333332, ...,  47.333332,
           40.       ,  34.333332],
         [ 26.333334,  40.       ,  48.       , ...,  51.333332,
           44.666668,  38.       ],
         [ 28.       ,  37.       ,  38.666668, ...,  54.       ,
           48.666668,  43.666668]],

        [[ 59.666668,  70.333336,  79.333336, ..., 111.       ,
           86.666664,  80.333336],
         [ 61.333332,  67.333336,  73.666664, ..., 172.66667 ,
          135.66667 , 113.       ],
         [ 62.666668,  60.333332,  60.       , ..., 210.       ,
          182.33333 , 152.66667 ],
         ...,
         [ 71.333336,  74.333336,  78.666664, ..., 119.666664,
           75.666664,  56.       ],
         [ 48.       ,  50.666668,  51.666668, ..., 121.666664,
           81.666664,  69.       ],
         [ 47.      ,  47.333332,  47.333332, ..., 115.333336,
           78.666664,  74.666664]],

        ...,

        [[110.666664,  88.666664,  59.666668, ..., 191.33333 ,
          156.66667 , 112.      ],
         [ 96.      ,  68.666664,  43.333332, ..., 199.66667 ,
          169.33333 , 131.33333 ],
         [ 92.666664,  55.333332,  44.333332, ..., 201.33333 ,
          181.      , 143.      ],
         ...,
         [ 35.333332,  35.666668,  42.666668, ...,  33.666668,
           21.      ,  30.      ],
         [ 34.666668,  36.      ,  41.      , ...,  63.666668,
           25.666666,  33.      ],
         [ 35.      ,  35.333332,  39.666668, ..., 104.      ,
           39.666668,  44.      ]],

        [[ 56.      ,  61.666668,  79.333336, ...,  39.666668,
           40.333332,  37.666668],
         [ 53.333332,  71.666664,  92.      , ...,  40.666668,
           40.      ,  40.666668],
         [ 62.333332,  87.333336, 103.      , ...,  37.333332,
           35.333332,  40.666668],
         ...,
         [ 21.      ,  76.666664, 164.      , ...,   9.333333,
           10.      ,   9.333333],
         [ 18.666666,  67.      , 163.33333 , ...,   9.      ,
           10.      ,   9.      ],
         [ 21.333334,  61.      , 157.      , ...,   8.666667,
            9.      ,   8.666667]],

        [[ 30.666666,  43.      ,  55.      , ...,  12.333333,
           12.333333,  13.333333],
         [ 30.666666,  42.333332,  54.      , ...,  12.333333,
           13.333333,  13.333333],
         [ 30.      ,  41.      ,  52.      , ...,  11.333333,
           13.      ,  13.      ],
         ...,
         [ 82.666664,  81.666664,  79.      , ...,  14.666667,
           14.333333,  15.333333],
         [ 80.333336,  80.666664,  79.333336, ...,  16.666666,
           17.      ,  17.333334],
         [ 77.      ,  79.      ,  80.      , ...,  17.333334,
           16.666666,  20.666666]]], dtype=float32),
 'target': array([ 65,  97, 205, ..., 141, 199,  49], dtype=int64),
 'target_names': array(['Al Gore', 'Alan Greenspan', 'Albert Costa', 'Alejan
dro Toledo',
        'Ali Naimi', 'Alvaro Uribe', 'Amelie Mauresmo', 'Ana Guevara',
        'Ana Palacio', 'Andre Agassi', 'Andy Roddick', 'Angela Merkel',
        'Ann Veneman', 'Anna Kournikova', 'Ari Fleischer', 'Ariel Sharon',
        'Arminio Fraga', 'Arnoldo Aleman', 'Atal Bihari Vajpayee',
        'Benjamin Netanyahu', 'Bertie Ahern', 'Bill Clinton',
        'Bill McBride', 'Bill Simon', 'Bob Graham', 'Bob Stoops',
        'Boris Becker', 'Britney Spears', 'Bulent Ecevit',
        'Calista Flockhart', 'Carlos Menem', 'Celine Dion',
        'Cesar Gaviria', 'Charles Moose', 'Charlton Heston',
        'Chen Shui-bian', 'Choi Sung-hong', 'Christine Todd Whitman',
        'Ciro Gomes', 'Colin Farrell', 'Colin Montgomerie', 'Colin Powell',
        'Condoleezza Rice', 'Costas Simitis', 'David Beckham',
        'David Nalbandian', 'David Trimble', 'David Wells',
        'Dennis Hastert', 'Denzel Washington', 'Dick Cheney',
        'Donald Rumsfeld', 'Edmund Stoiber', 'Eduard Shevardnadze',
        'Eduardo Duhalde', 'Elsa Zylberstein', 'Enrique Bolanos',
        'Erika Harold', 'Fernando Gonzalez', 'Fernando Henrique Cardoso',
        'Fidel Castro', 'George Clooney', 'George HW Bush',
        'George Pataki', 'George Robertson', 'George W Bush',
        'Gerhard Schroeder', 'Gerry Adams', 'Gloria Macapagal Arroyo',
        'Goldie Hawn', 'Gonzalo Sanchez de Lozada', 'Gordon Brown',
        'Gray Davis', 'Gwyneth Paltrow', 'Hamid Karzai', 'Hans Blix',
        'Harrison Ford', 'Heidi Klum', 'Hillary Clinton', 'Hosni Mubarak',
        'Howard Dean', 'Hu Jintao', 'Hugo Chavez', 'Ian Thorpe',
        'Igor Ivanov', 'JK Rowling', 'Jack Straw', 'Jackie Chan',
        'Jacques Chirac', 'Jake Gyllenhaal', 'James Blake', 'James Kelly',
        'James Wolfensohn', 'Javier Solana', 'Jean Chretien',
        'Jean-Pierre Raffarin', 'Jeb Bush', 'Jennifer Aniston',
        'Jennifer Capriati', 'Jennifer Garner', 'Jennifer Lopez',
        'Jeremy Greenstock', 'Jesse Jackson', 'Joe Lieberman',
        'John Ashcroft', 'John Bolton', 'John Howard', 'John Manley',
        'John McCain', 'John Travolta', 'Jonathan Edwards',
        'Joschka Fischer', 'Jose Serra', 'Joseph Biden',
        'Juan Carlos Ferrero', 'Juan Pablo Montoya', 'Julianne Moore',
        'Junichiro Koizumi', 'Justin Timberlake', 'Justine Pasek',
        'Kate Hudson', 'Kim Dae-jung', 'King Abdullah II', 'Kofi Annan',
        'Kurt Warner', 'Lance Armstrong', 'Lance Bass', 'Larry Brown',
        'Laura Bush', 'Li Peng', 'Lindsay Davenport', 'Liza Minnelli',
        'Lleyton Hewitt', 'Lucy Liu', 'Luis Gonzalez Macchi',
        'Luiz Inacio Lula da Silva', 'Mahathir Mohamad',
        'Marco Antonio Barrera', 'Martha Stewart', 'Martin McGuinness',
        'Megawati Sukarnoputri', 'Michael Bloomberg', 'Michael Chang',
        'Michael Chiklis', 'Michael Douglas', 'Michael Schumacher',
        'Michelle Yeoh', 'Mohammad Khatami', 'Mohammed Al-Douri',
        'Monica Seles', 'Naji Sabri', 'Nancy Pelosi', 'Naomi Watts',
        'Nia Vardalos', 'Nick Nolte', 'Nicole Kidman', 'Pamela Anderson',
        'Paul Burrell', 'Paul Martin', 'Paul McCartney', 'Paul ONeill',
        'Pedro Malan', 'Pervez Musharraf', 'Pete Sampras', 'Peter Struck',
        'Pierce Brosnan', 'Prince Charles', 'Queen Elizabeth II',
        'Ray Romano', 'Ricardo Lagos', 'Richard Gephardt',
        'Richard Virenque', 'Robert Blake', 'Robert Duvall',
        'Robert Mueller', 'Robert Redford', 'Roman Polanski',
        'Rubens Barrichello', 'Rudolph Giuliani', 'Russell Simmons',
        'Saddam Hussein', 'Serena Williams', 'Sergei Ivanov',
        'Sergio Vieira De Mello', 'Sharon Stone', 'Shimon Peres',
        'Sophia Loren', 'Sourav Ganguly', 'Spencer Abraham', 'Steffi Graf',
        'Susan Sarandon', 'Sylvester Stallone', 'Taha Yassin Ramadan',
        'Tang Jiaxuan', 'Tariq Aziz', 'Thabo Mbeki', 'Tiger Woods',
        'Tim Henman', 'Tom Cruise', 'Tom Daschle', 'Tom Hanks',
        'Tom Harkin', 'Tom Ridge', 'Tommy Franks', 'Tommy Haas',
        'Tony Blair', 'Trent Lott', 'Vaclav Havel', 'Valentino Rossi',
        'Vanessa Redgrave', 'Venus Williams', 'Vicente Fernandez',
        'Vicente Fox', 'Victoria Clarke', 'Vladimir Putin',
        'Vojislav Kostunica', 'Winona Ryder', 'Woody Allen',
        'Xanana Gusmao', 'Yao Ming', 'Yashwant Sinha', 'Yasser Arafat',
        'Yoko Ono', 'Yoriko Kawaguchi', 'Zhu Rongji'], dtype='<U25'),
 'DESCR': ".. _labeled_faces_in_the_wild_dataset:\n\nThe Labeled Faces in th
e Wild face recognition dataset\n------------------------------------------
-----------\n\nThis dataset is a collection of JPEG pictures of famous peopl
e collected\nover the internet, all details are available on the official we
bsite:\n\n    http://vis-www.cs.umass.edu/lfw/\n\nEach (http://vis-www.cs.um
ass.edu/lfw/\n\nEach) picture is centered on a single face. The typical task
is called\nFace Verification: given a pair of two pictures, a binary classif
```

```
ier\nmust predict whether the two images are from the same person.\n\nAn alt
ernative task, Face Recognition or Face Identification is:\ngiven the pictur
e of the face of an unknown person, identify the name\nof the person by refe
rring to a gallery of previously seen pictures of\nidentified persons.\n\nBo
th Face Verification and Face Recognition are tasks that are typically\nperf
ormed on the output of a model trained to perform Face Detection. The\nmost
 popular model for Face Detection is called Viola-Jones and is\nimplemented
 in the OpenCV library. The LFW faces were extracted by this\nface detector
 from various online websites.\n\n**Data Set Characteristics:**\n\n    =====
============   =======================\n    Classes
5749\n    Samples total                         13233\n    Dimensionality
5828\n    Features            real, between 0 and 255\n    =================
======================\n\nUsage\n~~~~~\n\n``scikit-learn`` provides two loa
ders that will automatically download,\ncache, parse the metadata files, dec
ode the jpeg and convert the\ninteresting slices into memmapped numpy array
s. This dataset size is more\nthan 200 MB. The first load typically takes mo
re than a couple of minutes\nto fully decode the relevant part of the JPEG f
iles into numpy arrays. If\nthe dataset has  been loaded once, the following
times the loading times\nless than 200ms by using a memmapped version memoiz
ed on the disk in the\n``~/scikit_learn_data/lfw_home/`` folder using ``jobl
ib``.\n\nThe first loader is used for the Face Identification task: a multi-
class\nclassification task (hence supervised learning)::\n\n  >>> from sklea
rn.datasets import fetch_lfw_people\n  >>> lfw_people = fetch_lfw_people(min
_faces_per_person=70, resize=0.4)\n\n  >>> for name in lfw_people.target_nam
es:\n  ...     print(name)\n  ...\n  Ariel Sharon\n  Colin Powell\n  Donald
 Rumsfeld\n  George W Bush\n  Gerhard Schroeder\n  Hugo Chavez\n  Tony Blair
\n\nThe default slice is a rectangular shape around the face, removing\nmost
of the background::\n\n  >>> lfw_people.data.dtype\n  dtype('float32')\n\n
 >>> lfw_people.data.shape\n  (1288, 1850)\n\n  >>> lfw_people.images.shape
\n  (1288, 50, 37)\n\nEach of the ``1140`` faces is assigned to a single per
son id in the ``target``\narray::\n\n  >>> lfw_people.target.shape\n  (128
8,)\n\n  >>> list(lfw_people.target[:10])\n  [5, 6, 3, 1, 0, 1, 3, 4, 3, 0]
\n\nThe second loader is typically used for the face verification task: each
sample\nis a pair of two picture belonging or not to the same person::\n\n
 >>> from sklearn.datasets import fetch_lfw_pairs\n  >>> lfw_pairs_train = f
etch_lfw_pairs(subset='train')\n\n  >>> list(lfw_pairs_train.target_names)\n
['Different persons', 'Same person']\n\n  >>> lfw_pairs_train.pairs.shape\n
(2200, 2, 62, 47)\n\n  >>> lfw_pairs_train.data.shape\n  (2200, 5828)\n\n  >
>> lfw_pairs_train.target.shape\n  (2200,)\n\nBoth for the :func:`sklearn.da
tasets.fetch_lfw_people` and\n:func:`sklearn.datasets.fetch_lfw_pairs` funct
ion it is\npossible to get an additional dimension with the RGB color channe
ls by\npassing ``color=True``, in that case the shape will be\n``(2200, 2, 6
2, 47, 3)``.\n\nThe :func:`sklearn.datasets.fetch_lfw_pairs` datasets is sub
divided into\n3 subsets: the development ``train`` set, the development ``te
st`` set and\nan evaluation ``10_folds`` set meant to compute performance me
trics using a\n10-folds cross validation scheme.\n\n.. topic:: References:\n
\n * `Labeled Faces in the Wild: A Database for Studying Face Recognition\n
in Unconstrained Environments.\n   <http://vis-www.cs.umass.edu/lfw/lfw.pdf>
`_\n   Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller.\n
University of Massachusetts, Amherst, Technical Report 07-49, October, 200
7.\n\n\nExamples\n~~~~~~~~\n\n:ref:`sphx_glr_auto_examples_applications_plot
_face_recognition.py`\n"}
```

In [10]:

```
data.data.shape
```

Out[10]:

```
(4225, 2914)
```

In [11]:

```
data.images.shape
```

Out[11]:

```
(4225, 62, 47)
```

In [12]:

```
fig = plt.figure(figsize=(8, 8))
for i in range(64):
    ax = fig.add_subplot(8, 8, i+1)
    ax.imshow(data.images[i], cmap=plt.cm.bone)
```

In [13]:

```
def find_k(data, s):
    pca = PCA()
    pca.fit_transform(data.data)

    k = 0
    total_sum = sum(pca.explained_variance_)
    current = 0
    while current/total_sum < s:
        current += pca.explained_variance_[k]
        k += 1
    return k
```

In [14]:

```
find_k(data.data, 0.99)
```

Out[14]:

```
500
```

In [16]:

```
find_k(data.data, 0.97)
```

Out[16]:

```
261
```

In [17]:

```
find_k(data.data, 0.95)
```

Out[17]:

```
177
```

In [18]:

```
X = data.data
Y = data.target
```

In [19]:

```
X.shape, Y.shape
```

Out[19]:

```
((4225, 2914), (4225,))
```

In [20]:

```
pca = PCA(n_components=177)
train_transform = pca.fit_transform(X)
```

In [21]:

```
data.images.shape
```

Out[21]:

```
(4225, 62, 47)
```

In [23]:

```
inversed_data = pca.inverse_transform(train_transform)
train_data = inversed_data.reshape((4225, 62, 47))
train_data.shape
```

Out[23]:

```
(4225, 62, 47)
```

In [25]:

```
fig = plt.figure(figsize=(8,8))
for i in range(64):
    ax = fig.add_subplot(8,8, i+1)
    ax.imshow(train_data[i], cmap=plt.cm.bone)
```

In [27]:

```
eigenvectors = pca.components_
eigenvectors.shape
```

Out[27]:

```
(177, 2914)
```

In [31]:

```
eigenfaces = eigenvectors.reshape((177, 62, 47))
fig = plt.figure(figsize=(8,8))
for i in range(64):
    ax = fig.add_subplot(8, 8, i+1)
    ax.imshow(eigenfaces[i], cmap=plt.cm.bone)
```