In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
from sklearn.datasets import fetch_olivetti_faces
faces_data = fetch_olivetti_faces()
```

In [ ]:
```
print(faces_data.DESCR)
```

```
.. _olivetti_faces_dataset:

The Olivetti faces dataset
--------------------------

`This dataset contains a set of face images`_ taken between April 1992 and
April 1994 at AT&T Laboratories Cambridge. The
:func:`sklearn.datasets.fetch_olivetti_faces` function is the data
fetching / caching function that downloads the data
archive from AT&T.

.. _This dataset contains a set of face images: http://www.cl.cam.ac.uk/research/d
tg/attarchive/facedatabase.html

As described on the original website:

    There are ten different images of each of 40 distinct subjects. For some
    subjects, the images were taken at different times, varying the lighting,
    facial expressions (open / closed eyes, smiling / not smiling) and facial
    details (glasses / no glasses). All the images were taken against a dark
    homogeneous background with the subjects in an upright, frontal position
    (with tolerance for some side movement).

**Data Set Characteristics:**

    =================   =====================
    Classes                                40
    Samples total                         400
    Dimensionality                       4096
    Features            real, between 0 and 1
    =================   =====================

The image is quantized to 256 grey levels and stored as unsigned 8-bit
integers; the loader will convert these to floating point values on the
interval [0, 1], which are easier to work with for many algorithms.

The "target" for this database is an integer from 0 to 39 indicating the
identity of the person pictured; however, with only 10 examples per class, this
relatively small dataset is more interesting from an unsupervised or
semi-supervised perspective.

The original dataset consisted of 92 x 112, while the version available here
consists of 64x64 images.

When using these images, please give credit to AT&T Laboratories Cambridge.
```

In [ ]:
```
faces_data.target
```

Out[ ]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  5,
        5,  5,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,
        8,  8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10,
       10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       11, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13,
       13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15,
       15, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
       17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18,
       18, 18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20,
       20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 22,
       22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 23,
       23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 25, 25, 25, 25, 25,
       25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 27, 27,
       27, 27, 27, 27, 27, 27, 27, 27, 28, 28, 28, 28, 28, 28, 28, 28, 28,
       28, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 30, 30, 30, 30, 30, 30,
       30, 30, 30, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32,
       32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
       34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35,
       35, 35, 35, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 37, 37, 37, 37,
       37, 37, 37, 37, 37, 37, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 39,
       39, 39, 39, 39, 39, 39, 39, 39, 39])
```

spliting data into a training set, a validation set, and a test set

In [ ]:
```python
from sklearn.model_selection import StratifiedShuffleSplit
split_data = StratifiedShuffleSplit(n_splits=1, test_size=40, random_state=42)
train_valid_idx, test_idx = next(split_data.split(faces_data.data, faces_data.targe
X_train_data = faces_data.data[train_valid_idx]
y_train_data = faces_data.target[train_valid_idx]
X_test = faces_data.data[test_idx]
y_test = faces_data.target[test_idx]
```

In [ ]:
```python
split_data = StratifiedShuffleSplit(n_splits=1, test_size=80, random_state=43)
train_idx, valid_idx = next(split_data.split(X_train_data,y_train_data))
X_train = X_train_data[train_idx]
y_train = y_train_data[train_idx]
X_valid = X_train_data[valid_idx]
y_valid = y_train_data[valid_idx]
```

dimension of the train, test, and validation dataset.

In [ ]:
```python
print(X_train.shape, y_train.shape)
print(X_valid.shape, y_valid.shape)
print(X_test.shape, y_test.shape)
```

```
(280, 4096) (280,)
(80, 4096) (80,)
(40, 4096) (40,)
```

From the previous assignment, here I use PCA to speed up, by reducing the data's dimensionality using PCA:

In [ ]:
```python
from sklearn.decomposition import PCA

pca = PCA(0.99)
X_train_pca = pca.fit_transform(X_train)
X_valid_pca = pca.transform(X_valid)
X_test_pca = pca.transform(X_test)
```

```
pca.n_components_
```

Out[ ]:

```
199
```

There are 199 data labels left after 1% of the labels have been removed.

Clustering the images using kmeans with a range from 5 to 150 with an interval of 5.

In [ ]:

```
from sklearn.cluster import KMeans

k_range = range(5, 150, 5)
kmeans_per_k = []
for k in k_range:
    print("k={}".format(k))
    kmeans = KMeans(n_clusters=k, random_state=42).fit(X_train_pca)
    kmeans_per_k.append(kmeans)
```

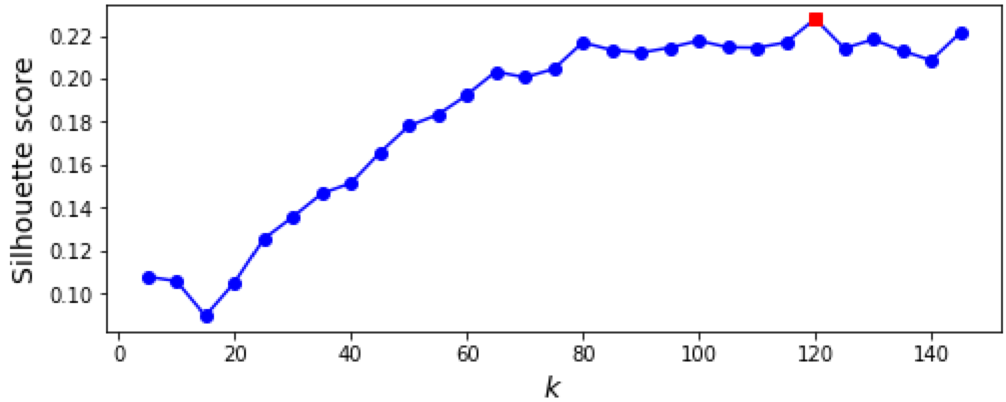
```
k=5
k=10
k=15
k=20
k=25
k=30
k=35
k=40
k=45
k=50
k=55
k=60
k=65
k=70
k=75
k=80
k=85
k=90
k=95
k=100
k=105
k=110
k=115
k=120
k=125
k=130
k=135
k=140
k=145
```

In [ ]:

```
from sklearn.metrics import silhouette_score

silhouette_scores = [silhouette_score(X_train_pca, model.labels_)
                     for model in kmeans_per_k]
best_index = np.argmax(silhouette_scores)
best_k = k_range[best_index]
best_score = silhouette_scores[best_index]

plt.figure(figsize=(8, 3))
plt.plot(k_range, silhouette_scores, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Silhouette score", fontsize=14)
plt.plot(best_k, best_score, "rs")
plt.show()
best_k
```

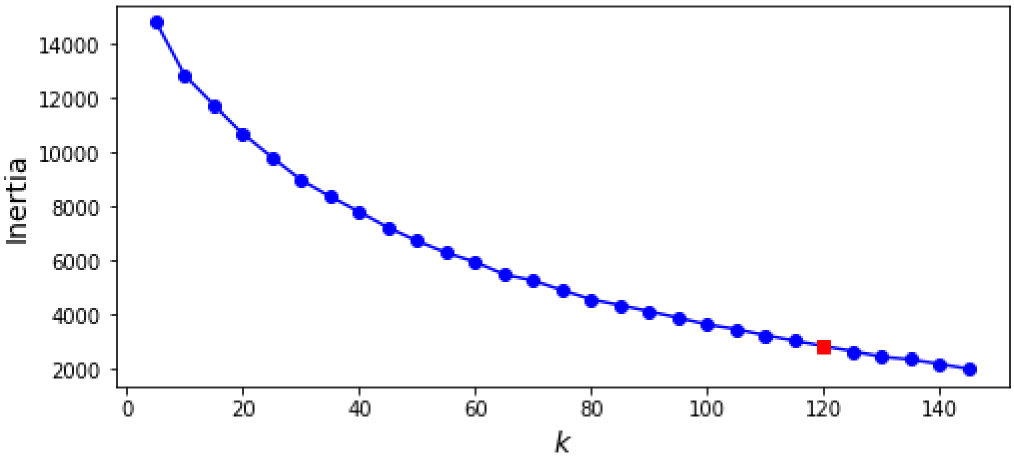


Out[ ]: 120

The above graph show the best cluster score which is 120

In [ ]:
```
inertias = [model.inertia_ for model in kmeans_per_k]
best_inertia = inertias[best_index]

plt.figure(figsize=(8, 3.5))
plt.plot(k_range, inertias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inertia", fontsize=14)
plt.plot(best_k, best_inertia, "rs")
plt.show()
```



Visualizing a cluster to view it through similar-looking faces

In [ ]:
```
best_model = kmeans_per_k[best_index]
def plot_faces(faces, labels, n_cols=5):
    faces = faces.reshape(-1, 64, 64)
    n_rows = (len(faces) - 1) // n_cols + 1
    plt.figure(figsize=(n_cols, n_rows * 1.1))
    for index, (face, label) in enumerate(zip(faces, labels)):
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(face, cmap="gray")
        plt.axis("off")
        plt.title(label)
    plt.show()
```

In [ ]:
```
for cluster_id in np.unique(best_model.labels_):
    print("Cluster", cluster_id)
    in_cluster = best_model.labels_==cluster_id
    faces = X_train[in_cluster]
```

```
    labels = y_train[in_cluster]
    plot_faces(faces, labels)
```

Cluster 0

Cluster 1

Cluster 2

Cluster 3

Cluster 4

Cluster 5

Cluster 6

Cluster 7

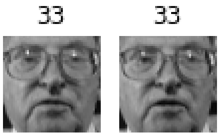
Cluster 8

Cluster 9

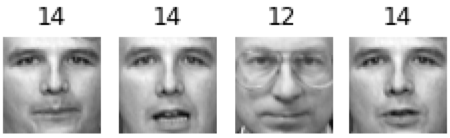
Cluster 10

Cluster 11

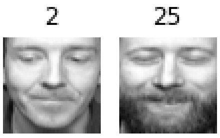
Cluster 12



Cluster 13



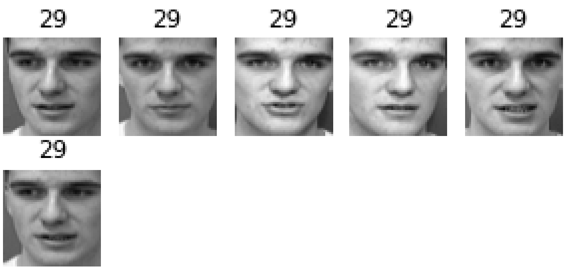
Cluster 14



Cluster 15

Cluster 16

Cluster 17



Cluster 18

Cluster 19

Cluster 20

Cluster 21

Cluster 22

Cluster 23

Cluster 24

Cluster 25

Cluster 26

Cluster 27

Cluster 28

Cluster 29

Cluster 30

Cluster 31

Cluster 32

Cluster 33

Cluster 34

Cluster 35

Cluster 36

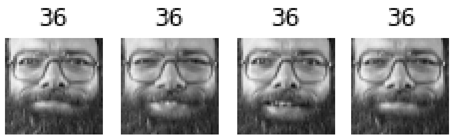
Cluster 37

Cluster 38

Cluster 39

Cluster 40

Cluster 41



Cluster 42

Cluster 43

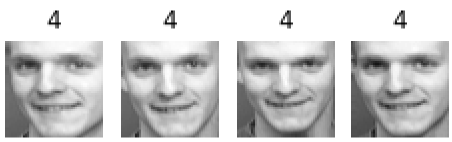
Cluster 44

Cluster 45

Cluster 46

Cluster 47

Cluster 48



Cluster 49

Cluster 50

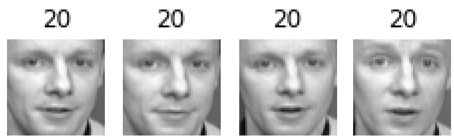
Cluster 51

Cluster 52

Cluster 53

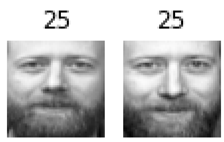
Cluster 54

Cluster 55



Cluster 56

Cluster 57



Cluster 58

Cluster 59

Cluster 60

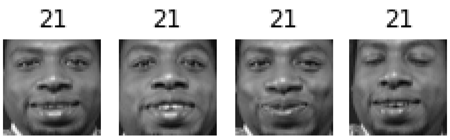
Cluster 61

Cluster 62

Cluster 63

Cluster 64

Cluster 65



Cluster 66

Cluster 67

Cluster 68

Cluster 69

Cluster 70

Cluster 71

Cluster 72

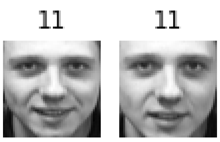
Cluster 73

Cluster 74

Cluster 75

Cluster 76

Cluster 77



Cluster 78

Cluster 79

3

Cluster 80

7

Cluster 81

9

Cluster 82

13 13

Cluster 83

10 10

Cluster 84

1 1

Cluster 85

18 18

Cluster 86

0

Cluster 87

32 32 32

Cluster 88

26 26

Cluster 89

15

Cluster 90

31

Cluster 91

25 25 25

Cluster 92

38 38 28

Cluster 93

19 19

Cluster 94

2 2

Cluster 95

11 0

Cluster 96

15

Cluster 97

0

Cluster 98

0

Cluster 99

6

Cluster 100

11 11 11

Cluster 101

28 28

Cluster 102

19 19

Cluster 103

35

Cluster 104

31

Cluster 105

12 12

Cluster 106

0

Cluster 107

24 24

Cluster 108

27

Cluster 109

12 12

Cluster 110

10

Cluster 111

2 2

Cluster 112

22 22

Cluster 113

31 31

Cluster 114

7

Cluster 115

16

Cluster 116

20 20

Cluster 117

9

Cluster 118

11

Cluster 119

While some are mixed, the majority are grouped by faces with less complexity.