In [49]:

```
#Loading the dataset
```

In [50]:

```
from sklearn.datasets import load_digits
```

In [51]:

```
digits=load_digits()
```

In [52]:

```
type(digits)
```

Out[52]:

```
sklearn.utils.Bunch
```

In [53]:

```
print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

In [54]:

```
digits.target.data.shape
```

Out[54]:

```
(1797,)
```

In [55]:

```
digits.target[::10]
```

Out[55]:

```
array([0, 0, 0, 0, 8, 2, 3, 1, 1, 1, 4, 4, 5, 0, 0, 0, 0, 8, 2, 3, 1, 1,
       9, 5, 7, 4, 4, 4, 4, 6, 7, 2, 4, 5, 2, 7, 6, 8, 4, 4, 4, 4, 6, 7,
       2, 4, 5, 2, 7, 6, 8, 4, 4, 4, 4, 6, 7, 2, 4, 5, 2, 7, 6, 8, 4, 4,
       4, 4, 6, 7, 2, 4, 5, 2, 7, 6, 8, 4, 4, 4, 4, 6, 7, 2, 4, 5, 2, 7,
       6, 8, 4, 5, 5, 5, 5, 3, 6, 4, 0, 3, 1, 5, 9, 1, 1, 1, 9, 4, 8, 3,
       9, 3, 1, 3, 2, 8, 3, 3, 3, 5, 7, 8, 3, 9, 3, 1, 3, 2, 8, 3, 3, 3,
       5, 7, 8, 3, 9, 3, 1, 3, 2, 8, 5, 5, 5, 5, 3, 0, 6, 2, 1, 6, 9, 2,
       9, 5, 5, 7, 9, 1, 2, 6, 0, 3, 1, 5, 4, 3, 3, 3, 5, 7, 8, 3, 9, 3,
       1, 3, 2, 8])
```

In [56]:

```
digits.data[5]
```

Out[56]:

```
array([ 0.,  0., 12., 10.,  0.,  0.,  0.,  0.,  0.,  0., 14., 16., 16.,
       14.,  0.,  0.,  0.,  0., 13., 16., 15., 10.,  1.,  0.,  0.,  0.,
       11., 16., 16.,  7.,  0.,  0.,  0.,  0.,  0.,  4.,  7., 16.,  7.,
        0.,  0.,  0.,  0.,  0.,  4., 16.,  9.,  0.,  0.,  0.,  5.,  4.,
       12., 16.,  4.,  0.,  0.,  0.,  9., 16., 16., 10.,  0.,  0.])
```

In [57]:

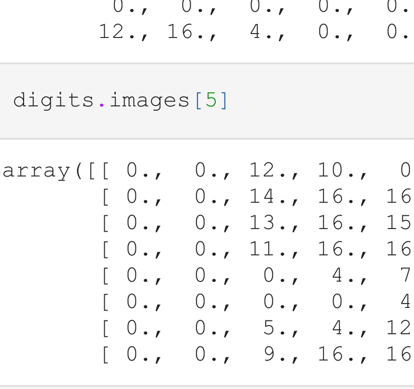
```
digits.images[5]
```

Out[57]:

```
array([[ 0.,  0., 12., 10.,  0.,  0.,  0.,  0.],
       [ 0.,  0., 14., 16., 16., 14.,  0.,  0.],
       [ 0.,  0., 13., 16., 15., 10.,  1.,  0.],
       [ 0.,  0., 11., 16., 16.,  7.,  0.,  0.],
       [ 0.,  0.,  0.,  4.,  7., 16.,  7.,  0.],
       [ 0.,  0.,  0.,  0.,  4., 16.,  9.,  0.],
       [ 0.,  0.,  5.,  4., 12., 16.,  4.,  0.],
       [ 0.,  0.,  9., 16., 16., 10.,  0.,  0.]])
```

In [58]:

```
axes=plt.subplot()
image=plt.imshow(digits.images[1700], cmap=plt.cm.gray_r)
xticks=axes.set_xticks([])
yticks=axes.set_yticks([])
```

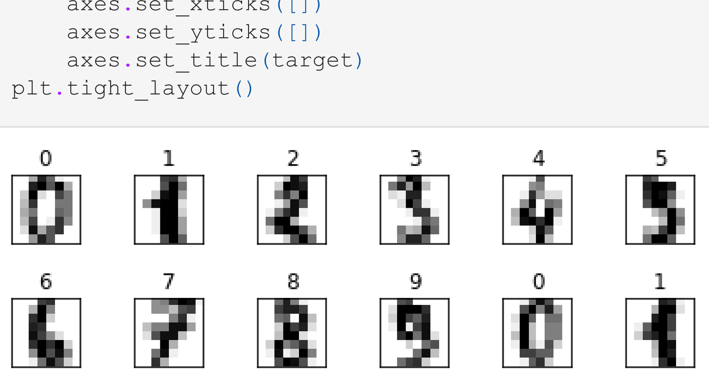


In [59]:

```
import matplotlib.pyplot as plt
```

In [60]:

```
figure, axes=plt.subplots(nrows=4,ncols=6, figsize=(6,4))
for item in zip(axes.ravel(), digits.images, digits.target):
    axes, image, target=item
    axes.imshow(image, cmap=plt.cm.gray_r)
    axes.set_xticks([])
    axes.set_yticks([])
    axes.set_title(target)
plt.tight_layout()
```




In [ ]:

In [61]:

```
from sklearn.model_selection import train_test_split
```

In [62]:

```
X_train, X_test, y_train,y_test=train_test_split(digits.data, digits.target, random_state=11)
```

In [63]:

```
X_train.shape
```

Out[63]:

```
(1347, 64)
```

In [64]:

```
X_test.shape
```

Out[64]:

```
(450, 64)
```

a) The processing of familiarizing self with the data is called __- data exploration b) display the image for a sample in number 1700 and 1900 of the digits dataset c) X_train, X_test, y_train,y_test=train_test_split(digits.data, digits.target, random_state=11, test_size=0.4) - what numbers of samples would be reserved for training and tesing using the above code 1078, 719 d) You should use all you data for training and testing: False or True explain? - False -

## Creating the Model

In [ ]:

```
# model is called an estimator
# KNeighborsClassifier - implements the k-nearest Neighbors Algorithm
```

In [65]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [66]:

```
knn=KNeighborsClassifier()
```

In [67]:

```
knn
```

Out[67]:

```
KNeighborsClassifier()
```

## Training the Model with KNeighborsClassifier object fit method

In [68]:

```
knn.fit(X=X_train, y=y_train)
```

Out[68]:

```
KNeighborsClassifier()
```

In [ ]:

```
# lazy estimator - job done when you use the model to make predictions
```

## Predicting Digit Class

In [69]:

```
predicted=knn.predict(X=X_test)
```

In [72]:

```
predicted
```

Out[72]:

```
array([0, 4, 9, 9, 3, 1, 4, 1, 5, 0, 4, 9, 4, 1, 5, 3, 3, 8, 5, 6, 9, 6,
       0, 6, 9, 3, 2, 1, 8, 1, 7, 0, 4, 4, 1, 5, 3, 0, 5, 7, 3, 9, 6, 5,
       5, 8, 8, 1, 1, 2, 4, 8, 5, 6, 9, 2, 1, 8, 5, 3, 2, 7, 9, 6, 3, 7,
       4, 2, 0, 1, 0, 2, 7, 3, 5, 1, 8, 7, 7, 2, 0, 6, 6, 4, 6, 8, 3, 7,
       4, 1, 9, 3, 5, 4, 0, 3, 1, 3, 3, 1, 2, 8, 5, 0, 1, 7, 2, 1, 3, 3,
       7, 4, 0, 2, 9, 0, 4, 2, 5, 6, 1, 2, 6, 1, 8, 6, 0, 2, 6, 2, 6, 1,
       9, 4, 8, 0, 4, 0, 2, 3, 4, 4, 1, 7, 4, 7, 2, 0, 3, 7, 8, 8, 7, 5,
       4, 3, 5, 4, 9, 1, 3, 8, 8, 1, 1, 6, 7, 3, 3, 9, 9, 0, 6, 1, 0, 1,
       0, 7, 6, 1, 5, 9, 0, 2, 2, 8, 6, 8, 3, 2, 9, 2, 9, 3, 0, 1, 2, 7,
       4, 9, 9, 7, 9, 3, 2, 7, 2, 6, 9, 8, 0, 2, 6, 3, 4, 2, 7, 6, 6, 7,
       7, 6, 0, 7, 6, 6, 0, 7, 1, 4, 4, 1, 0, 9, 4, 0, 4, 2, 4, 6, 5, 3,
       8, 4, 1, 3, 9, 8, 3, 8, 9, 4, 2, 0, 4, 9, 2, 3, 5, 0, 8, 2, 5, 4,
       7, 5, 5, 1, 0, 2, 9, 0, 7, 7, 6, 2, 1, 5, 4, 1, 0, 5, 1, 6, 5, 4,
       8, 7, 5, 9, 0, 2, 2, 3, 4, 4, 8, 8, 2, 5, 3, 0, 7, 0, 3, 0, 7, 9,
       8, 8, 3, 3, 9, 8, 2, 8, 4, 7, 7, 9, 1, 3, 5, 9, 8, 2, 2, 9, 4, 6,
       8, 0, 6, 1, 2, 7, 8, 8, 9, 7, 9, 0, 3, 7, 2, 3, 0, 7, 3, 9, 9, 4,
       2, 1, 7, 4, 4, 5, 7, 4, 7, 4, 4, 5, 2, 4, 2, 0, 6, 3, 6, 4, 2, 7,
       2, 2, 3, 2, 5, 8, 1, 0, 6, 6, 1, 5, 6, 8, 6, 7, 0, 1, 1, 9, 7, 2,
       7, 8, 2, 4, 8, 9, 8, 4, 4, 2, 5, 5, 5, 2, 6, 6, 9, 6, 9, 8, 2, 1,
       2, 3, 3, 7, 5, 9, 6, 6, 0, 0, 4, 7, 7, 7, 8, 2, 5, 5, 5, 1, 4, 6,
       0, 5, 9, 1, 3, 1, 2, 2, 1, 0])
```

In [73]:

```
expected=y_test
```

In [74]:

```
expected
```

Out[74]:

```
array([0, 4, 9, 9, 3, 1, 4, 1, 5, 0, 4, 9, 4, 1, 5, 3, 3, 8, 3, 6, 9, 6,
       0, 6, 9, 3, 2, 1, 8, 1, 7, 0, 4, 4, 1, 5, 3, 0, 5, 7, 3, 9, 6, 5,
       5, 8, 8, 1, 1, 2, 4, 9, 5, 6, 9, 2, 1, 8, 5, 3, 2, 7, 9, 6, 3, 7,
       4, 2, 0, 1, 0, 2, 7, 3, 5, 1, 8, 7, 7, 2, 0, 6, 6, 4, 6, 8, 3, 7,
       4, 1, 9, 3, 5, 4, 0, 3, 1, 3, 3, 1, 2, 8, 5, 0, 1, 7, 2, 1, 3, 3,
       7, 4, 0, 2, 9, 0, 4, 2, 5, 6, 1, 2, 6, 1, 8, 6, 0, 2, 6, 2, 6, 1,
       9, 4, 8, 0, 4, 0, 2, 3, 4, 4, 1, 7, 9, 7, 2, 0, 3, 7, 8, 8, 3, 5,
       4, 3, 5, 4, 9, 1, 3, 8, 8, 1, 1, 6, 7, 3, 3, 9, 9, 0, 6, 1, 0, 1,
       0, 7, 6, 1, 5, 9, 0, 2, 2, 8, 6, 8, 3, 2, 9, 2, 9, 3, 0, 1, 2, 7,
       4, 9, 9, 4, 9, 3, 2, 7, 2, 6, 9, 8, 0, 2, 6, 3, 4, 2, 7, 6, 6, 7,
       7, 6, 0, 7, 6, 6, 0, 7, 1, 4, 4, 1, 0, 9, 4, 0, 4, 2, 4, 6, 5, 3,
       8, 4, 1, 3, 9, 8, 3, 8, 9, 4, 2, 0, 4, 9, 2, 3, 5, 0, 8, 2, 5, 4,
       7, 5, 5, 1, 0, 2, 9, 0, 7, 7, 6, 2, 1, 5, 4, 1, 0, 5, 1, 6, 5, 4,
       8, 7, 5, 9, 0, 2, 2, 3, 4, 4, 8, 8, 8, 5, 3, 0, 7, 0, 3, 0, 7, 9,
       8, 8, 3, 3, 9, 8, 2, 8, 4, 7, 7, 9, 1, 3, 5, 8, 8, 2, 2, 9, 4, 6,
       8, 0, 6, 1, 2, 7, 8, 8, 9, 7, 9, 0, 3, 7, 2, 3, 0, 7, 3, 9, 9, 4,
       2, 1, 7, 4, 4, 5, 7, 4, 7, 4, 4, 5, 2, 4, 2, 0, 6, 3, 6, 4, 2, 7,
       2, 2, 8, 2, 5, 8, 1, 0, 6, 6, 1, 5, 6, 8, 6, 7, 0, 1, 1, 9, 7, 2,
       7, 8, 2, 4, 8, 9, 8, 4, 4, 2, 5, 5, 5, 2, 6, 6, 9, 6, 9, 8, 2, 1,
       2, 3, 8, 7, 5, 9, 6, 6, 0, 0, 4, 7, 7, 7, 8, 2, 5, 5, 5, 8, 4, 6,
       0, 5, 9, 1, 3, 1, 2, 2, 1, 0])
```

In [85]:

```
predicted[:70]
```

Out[85]:

```
array([0, 4, 9, 9, 3, 1, 4, 1, 5, 0, 4, 9, 4, 1, 5, 3, 3, 8, 5, 6, 9, 6,
       0, 6, 9, 3, 2, 1, 8, 1, 7, 0, 4, 4, 1, 5, 3, 0, 5, 7, 3, 9, 6, 5,
       5, 8, 8, 1, 1, 2, 4, 8, 5, 6, 9, 2, 1, 8, 5, 3, 2, 7, 9, 6, 3, 7,
       4, 2, 0, 1])
```

In [86]:

```
expected[:70]
```

Out[86]:

```
array([0, 4, 9, 9, 3, 1, 4, 1, 5, 0, 4, 9, 4, 1, 5, 3, 3, 8, 3, 6, 9, 6,
       0, 6, 9, 3, 2, 1, 8, 1, 7, 0, 4, 4, 1, 5, 3, 0, 5, 7, 3, 9, 6, 5,
       5, 8, 8, 1, 1, 2, 4, 9, 5, 6, 9, 2, 1, 8, 5, 3, 2, 7, 9, 6, 3, 7,
       4, 2, 0, 1])
```

In [78]:

```
# locate all the incorrect predictions for the entire test set
```

In [79]:

```
wrong=[(p,e) for (p,e) in zip(predicted, expected) if p != e]
```

In [80]:

```
wrong
```

Out[80]:

```
[(5, 3),
 (8, 9),
 (4, 9),
 (7, 3),
 (7, 4),
 (2, 8),
 (9, 8),
 (3, 8),
 (3, 8),
 (1, 8)]
```

In [87]:

```
450-10
```

Out[87]:

```
440
```

In [88]:

```
440/450*100
```

Out[88]:

```
97.77777777777777
```

In [89]:

```
print(f'{knn.score(X_test,y_test):.2%}')
```

```
97.78%
```

In [90]:

```
# we can improve the models performance by hyperparameter tuning - get the optinal value of k
```

In [91]:

```
# Consusion Matrix
```

In [92]:

```
from sklearn.metrics import confusion_matrix
```

In [93]:

```
confusion=confusion_matrix(y_true=expected,y_pred=predicted)
```

In [94]:

```
confusion
```

Out[94]:

```
array([[45,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 45,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 54,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 42,  0,  1,  0,  1,  0,  0],
       [ 0,  0,  0,  0, 49,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0, 38,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 42,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 45,  0,  0],
       [ 0,  1,  1,  2,  0,  0,  0,  0, 39,  1],
       [ 0,  0,  0,  0,  1,  0,  0,  0,  1, 41]], dtype=int64)
```

In [95]:

```
45+45+54+42+49+38+42+45+39+41
```

Out[95]:

```
440
```

In [96]:

```
import pandas as pd
```

In [97]:

```
confusion_df=pd.DataFrame(confusion, index=range(10),columns=range(10))
```

In [98]:

```
confusion_df
```

Out[98]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 54 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 42 | 0 | 1 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 49 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 38 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 42 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 45 | 0 | 0 |
| 8 | 0 | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 39 | 1 |
| 9 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 41 |

In [100…

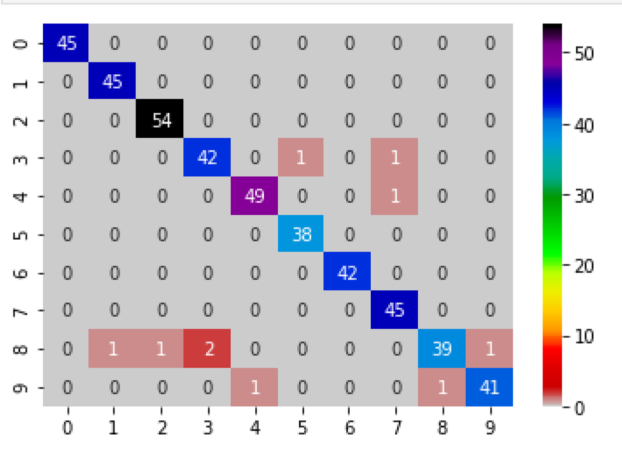
```
import matplotlib.pyplot as plt
```

In [101…

```
import seaborn as sns
```

In [102…

```
figure=sns.heatmap(confusion_df, annot=True,cmap=plt.cm.nipy_spectral_r)
```




In [ ]:

```
# accuracy - talk about it.
```