In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

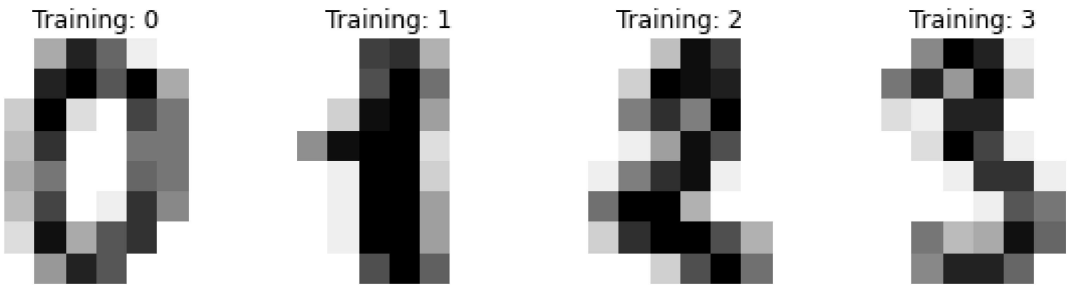
In [ ]:
```

```

Import data

In [2]:
```
from sklearn.datasets import load_digits
```

In [3]:
```
df=load_digits()
```

In [11]:
```
_, axes=plt.subplots(nrows=1,ncols=4,figsize=(10,3))
for ax, image, label in zip(axes,df.images,df.target):
  ax.set_axis_off()
  ax.imshow(image,cmap=plt.cm.gray_r,interpolation='nearest')
  ax.set_title('Training: %i' % label)
```



Data Preprocessing

Flatten Image

In [12]:
```
# 8*8 pixel  image flattened to 1*64 single dimension
```

In [13]:
```
df.images.shape
```

Out[13]:
```
(1797, 8, 8)
```

In [14]:
```
#1797 images of 8*8 pixels
```

In [15]:
```
df.images[0]
```

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 [ ]:
```
#observe above, we need to scale the data as well
```

In [16]:
```
df.images[0].shape
```

Out[16]:
```
(8, 8)
```

In [19]:
```
#Flattening the images-
n_samples=len(df.images)
data=df.images.reshape((n_samples,-1))
```

In [20]:
```
data[0]
```

Out[20]:
```
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 [21]:
```
data[0].shape
```

Out[21]:
```
(64,)
```

In [22]:
```
data.shape
```

Out[22]:
```
(1797, 64)
```

In [23]:
```
#see above,the data has reduced to 1797*64 only
```

In [ ]:
```

```

Scaling Data

In [24]:
```
data.min()
```

Out[24]:
```
0.0
```

In [25]:
```
data.max()
```

Out[25]: 16.0

In [26]:
```
data=data/16
```

In [27]:
```
data.min(), data.max()
```

Out[27]: (0.0, 1.0)

In [28]:
```
#yayy!! standardized !!
```

Train test split data

In [29]:
```
from sklearn.model_selection import train_test_split
```

In [31]:
```
X_train,X_test,y_train,y_test=train_test_split(data,df.target,train_size=0.7,rand
```

In [32]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[32]: ((1257, 64), (540, 64), (1257,), (540,))

In [ ]:
```

```

Random Forest Model

In [33]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [34]:
```
rf=RandomForestClassifier()
```

In [35]:
```
rf.fit(X_train,y_train)
```

Out[35]: RandomForestClassifier()

In [ ]:
```

```

Predict Test data

In [36]:
```
y_pred=rf.predict(X_test)
```

In [37]:
```
y_pred
```

Out[37]:
```
array([1, 9, 4, 1, 1, 0, 6, 9, 6, 0, 0, 9, 6, 4, 5, 6, 8, 8, 8, 3, 0, 1,
       2, 0, 8, 3, 1, 1, 7, 5, 0, 4, 1, 9, 5, 5, 4, 7, 1, 5, 7, 5, 7, 3,
       3, 7, 6, 1, 9, 8, 0, 9, 7, 1, 9, 3, 4, 0, 8, 3, 0, 9, 8, 1, 5, 5,
       0, 5, 7, 9, 5, 2, 6, 8, 4, 2, 2, 2, 8, 0, 4, 2, 1, 0, 7, 5, 5, 2,
       1, 4, 9, 2, 4, 8, 2, 4, 9, 5, 4, 5, 1, 7, 1, 7, 7, 4, 2, 0, 0, 5,
       1, 5, 7, 0, 1, 1, 5, 2, 5, 6, 8, 5, 7, 1, 1, 1, 9, 6, 9, 8, 7, 8,
       7, 4, 7, 2, 9, 1, 8, 1, 2, 0, 8, 1, 2, 6, 3, 2, 3, 3, 1, 2, 1, 6,
       9, 8, 9, 1, 4, 1, 7, 4, 5, 9, 1, 0, 6, 1, 3, 8, 1, 0, 7, 2, 3, 5,
       5, 5, 1, 9, 5, 9, 5, 0, 1, 0, 3, 2, 9, 0, 8, 3, 5, 7, 3, 9, 8, 7,
       4, 1, 6, 7, 8, 2, 6, 0, 6, 0, 7, 3, 1, 6, 2, 4, 6, 8, 5, 6, 2, 6,
       5, 8, 7, 4, 2, 8, 9, 9, 3, 1, 4, 4, 7, 9, 1, 6, 3, 1, 9, 6, 4, 5,
       0, 4, 2, 9, 0, 2, 0, 3, 8, 1, 2, 9, 1, 5, 2, 7, 0, 7, 9, 6, 6, 3,
       1, 6, 3, 3, 6, 1, 1, 3, 5, 7, 8, 1, 5, 1, 0, 3, 7, 2, 8, 8, 7, 8,
       8, 3, 3, 5, 9, 5, 7, 2, 0, 7, 7, 0, 6, 4, 1, 2, 4, 6, 5, 2, 4, 4,
       1, 4, 9, 6, 4, 6, 4, 3, 3, 7, 5, 8, 1, 3, 0, 5, 2, 4, 5, 5, 8, 9,
       9, 9, 4, 3, 2, 6, 3, 7, 6, 7, 1, 9, 2, 0, 1, 1, 2, 5, 8, 3, 1, 2,
       3, 4, 5, 2, 9, 3, 2, 2, 8, 4, 1, 6, 2, 5, 3, 5, 1, 8, 5, 3, 8, 7,
       7, 5, 5, 7, 0, 3, 8, 9, 8, 1, 6, 4, 7, 4, 3, 0, 0, 6, 6, 3, 6, 6,
       5, 4, 9, 1, 1, 8, 4, 2, 3, 8, 3, 6, 9, 7, 2, 3, 5, 8, 5, 1, 9, 1,
       7, 2, 4, 5, 2, 2, 9, 3, 4, 7, 1, 8, 9, 3, 6, 1, 8, 9, 4, 5, 4, 9,
       3, 2, 2, 4, 1, 9, 8, 9, 0, 2, 5, 8, 5, 2, 5, 3, 1, 8, 9, 1, 6, 3,
       4, 5, 7, 1, 0, 3, 6, 5, 1, 7, 0, 6, 0, 4, 8, 7, 6, 8, 2, 6, 6, 3,
       5, 7, 7, 7, 0, 7, 6, 4, 4, 8, 7, 2, 6, 9, 7, 2, 4, 2, 1, 4, 6, 8,
       5, 3, 3, 4, 4, 3, 4, 2, 0, 0, 2, 2, 2, 9, 1, 0, 1, 5, 5, 2, 5, 4,
       7, 1, 2, 6, 2, 4, 2, 3, 6, 2, 3, 4])
```

In [38]:
```
#0-9 digits
```

Model Accuracy

In [39]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [41]:
```
confusion_matrix(y_test,y_pred)
```

Out[41]:
```
array([[43,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 64,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 61,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 51,  0,  1,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 53,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  1, 57,  0,  0,  0,  0],
       [ 0,  1,  0,  0,  0,  1, 49,  0,  1,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 50,  0,  0],
       [ 0,  5,  0,  1,  0,  0,  0,  0, 47,  0],
       [ 0,  0,  0,  1,  0,  2,  0,  1,  1, 48]])
```

In [45]:
```
#1st row(0th index) is for 0 number
#0- 43 0s are predicted as 0
#1- 64  1s are predicted as 1
#2- 61 2s are predicted as 1 and so on...
#i.e. diagonals giving correct predictions, off diagonals giving u wrong
```

In [46]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        43
           1       0.91      1.00      0.96        64
           2       1.00      1.00      1.00        61
           3       0.96      0.98      0.97        52
           4       0.98      0.98      0.98        54
           5       0.93      0.98      0.96        58
           6       1.00      0.94      0.97        52
           7       0.96      1.00      0.98        50
           8       0.96      0.89      0.92        53
           9       1.00      0.91      0.95        53

    accuracy                           0.97       540
   macro avg       0.97      0.97      0.97       540
weighted avg       0.97      0.97      0.97       540
```

In [ ]:
```

```