Hand Written Digit Prediction -Classification Analysis

The digits dataset consist of 8X8 pixel image of digit.The image attribute of the dataset stoers 8x8 arrays of grayscale value of each image.We will use this arrays to visualize the first 4 images.The target attribute of the dataset stores the digit each image represents

Import Library

In [3]:
```
import pandas as pd
```

In [4]:
```
import numpy as np
```

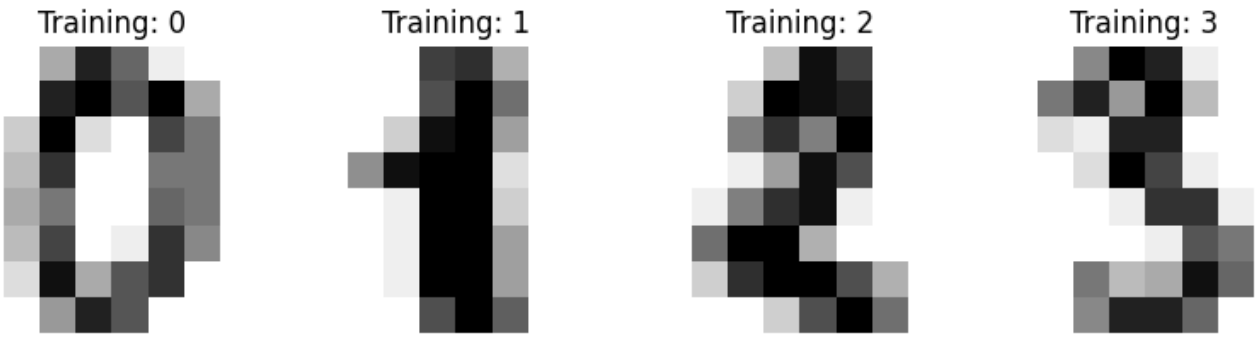
In [5]:
```
import matplotlib.pyplot as plt
```

Import Data

In [6]:
```
from sklearn.datasets import load_digits
```

In [7]:
```
df =load_digits()
```

In [8]:
```
_, 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

In [9]:
```
df.images.shape
```

Out [9]: (1797, 8, 8)

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

Out [8]:
```
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 [10]:
```
df.images[0].shape
```

Out [10]:
```
(8, 8)
```

In [ ]:
```
len(df.images)
```

Out [10]:
```
1797
```

In [11]:
```
n_samples = len(df.images)
data = df.images.reshape((n_samples, -1))
```

In [13]:
```
data[0]
```

Out [13]:
```
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 [ ]:
```
data[0].shape
```

Out [13]:
```
(64,)
```

In [12]:
```
data.shape
```

Out [12]:
```
(1797, 64)
```

Scaling Image Data

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

Out [15]:
```
0.0
```

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

Out [16]:
```
16.0
```

In [14]:
```
data = data/16
```

In [15]:
```
data.min()
```

Out [15]:
```
0.0
```

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

Out [19]:
```
1.0
```

In [16]:
```
data[0]
```

Out [16]:
```
array([0.    , 0.    , 0.3125, 0.8125, 0.5625, 0.0625, 0.    , 0.    ,
       0.    , 0.    , 0.8125, 0.9375, 0.625 , 0.9375, 0.3125, 0.    ,
       0.    , 0.1875, 0.9375, 0.125 , 0.    , 0.6875, 0.5   , 0.    ,
       0.    , 0.25  , 0.75  , 0.    , 0.    , 0.5   , 0.5   , 0.    ,
       0.    , 0.3125, 0.5   , 0.    , 0.    , 0.5625, 0.5   , 0.    ,
       0.    , 0.25  , 0.6875, 0.    , 0.0625, 0.75  , 0.4375, 0.    ,
       0.    , 0.125 , 0.875 , 0.3125, 0.625 , 0.75  , 0.    , 0.    ,
       0.    , 0.    , 0.375 , 0.8125, 0.625 , 0.    , 0.    , 0.    ])
```

Train Test Split Data

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
X_train , X_test , y_train , y_test = train_test_split(data, df.target, test_s
```

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

Out [23]:
```
((1257, 64), (540, 64), (1257,), (540,))
```

Random Forest Model

In [20]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [22]:
```
rf = RandomForestClassifier()
```

In [23]:
```
rf.fit(X_train, y_train)
```

Out [23]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

Predict Test Data

In [24]:
```
y_pred = rf.predict(X_test)
```

In [ ]:
```
y_pred
```

Out [28]:
```
array([2, 4, 7, 5, 5, 7, 2, 0, 7, 1, 6, 4, 5, 1, 2, 3, 4, 1, 1, 5, 4, 1,
       7, 5, 8, 3, 9, 5, 3, 7, 6, 5, 3, 3, 0, 5, 8, 6, 5, 1, 9, 2, 0, 1,
       2, 4, 8, 5, 5, 8, 9, 6, 8, 0, 6, 5, 6, 6, 0, 9, 8, 2, 0, 2, 5, 4,
       9, 0, 1, 0, 0, 1, 8, 4, 8, 8, 7, 7, 5, 3, 1, 3, 7, 7, 8, 4, 5, 4,
       7, 4, 3, 2, 2, 7, 6, 8, 0, 7, 4, 7, 7, 6, 4, 0, 9, 7, 9, 5, 1, 0,
       4, 5, 4, 0, 4, 5, 5, 1, 2, 4, 5, 9, 5, 6, 9, 7, 3, 6, 9, 6, 6, 8,
       9, 9, 0, 0, 3, 6, 0, 6, 0, 2, 2, 6, 1, 3, 9, 0, 7, 2, 3, 5, 4, 7,
       4, 1, 7, 9, 1, 8, 9, 6, 9, 5, 0, 6, 8, 9, 3, 1, 2, 0, 9, 2, 1, 3,
```

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

Model Accuracy

In [26]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [27]:
```
confusion_matrix(y_test, y_pred)
```

Out [27]:
```
array([[57,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 54,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 55,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 50,  0,  2,  0,  1,  0,  0],
       [ 0,  0,  0,  0, 49,  0,  0,  2,  0,  1],
       [ 0,  0,  0,  0,  1, 55,  0,  0,  0,  3],
       [ 1,  0,  0,  0,  0,  0, 55,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 48,  0,  1],
       [ 0,  2,  0,  0,  0,  0,  0,  1, 49,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  2,  1, 50]])
```

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

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        58
           1       1.00      1.00      1.00        54
           2       0.98      1.00      0.99        49
           3       1.00      0.97      0.98        59
           4       1.00      0.98      0.99        59
           5       0.95      1.00      0.97        58
           6       1.00      0.96      0.98        50
           7       0.98      1.00      0.99        53
           8       0.98      0.96      0.97        51
           9       0.98      0.98      0.98        49

    accuracy                           0.99       540
   macro avg       0.99      0.98      0.99       540
weighted avg       0.99      0.99      0.99       540
```

In [ ]:
```

```