Hand Written Digit Prediction - Classification Analysis

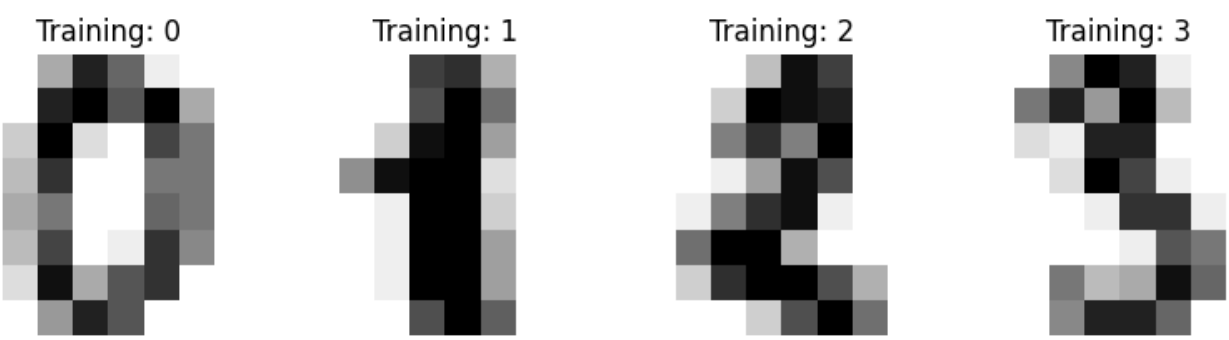
The digits dataset consists of 8×8 pixel images of digits. The digits dataset consists of 8x8 pixel images of digits. The images attribute of the dataset stores 8x8 arrays of grayscale values for each image. We will use these arrays to visualize the first 4 images. The target attribute of the dataset stores the digit each image represents

Import Library

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

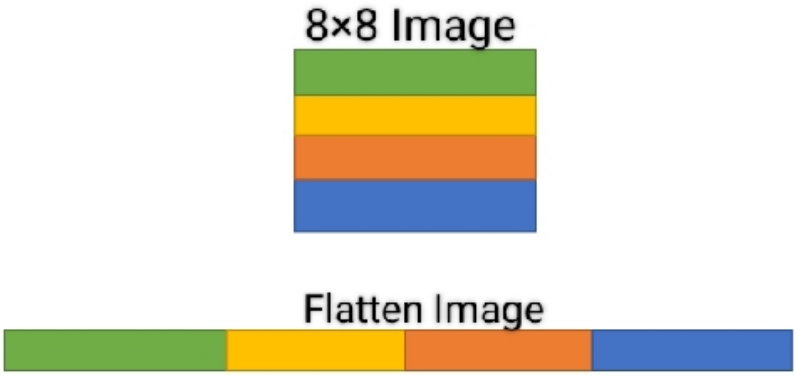
Import Data

```
from sklearn.datasets import load_digits

df = load_digits()

_, 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




```
df.images.shape
```

```
(1797, 8, 8)
```

```
df.images[0]
```

```
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.]])
```

```
df.images[0].shape
```

```
(8, 8)
```

```
len(df.images)
```

```
1797
```

```
n_samples = len(df.images)
data = df.images.reshape((n_samples, -1))
```

```
data[0]
```

```
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.])
```

```
data[0].shape
```

```
(64,)
```

```
data.shape
```

```
(1797, 64)
```

Scaling Image Data

```
data.min()
```

```
0.0
```

```
data.max()
```

```
16.0
```

```
data = data/16
```

```
data.min()
```

```
0.0
```

```
data.max()
```

```
1.0
```

```
data[0]
```

```
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

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

```
X_train, X_test, y_train, y_test = train_test_split(data, df.target, test_size=0.3)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((1257, 64), (540, 64), (1257,), (540,))
```

Random Forest Model

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier()
```

```
rf.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

Predict Test Data

```
y_pred = rf.predict(X_test)
```

```
y_pred
```

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