THE MACHINE LEARNING JOURNEY

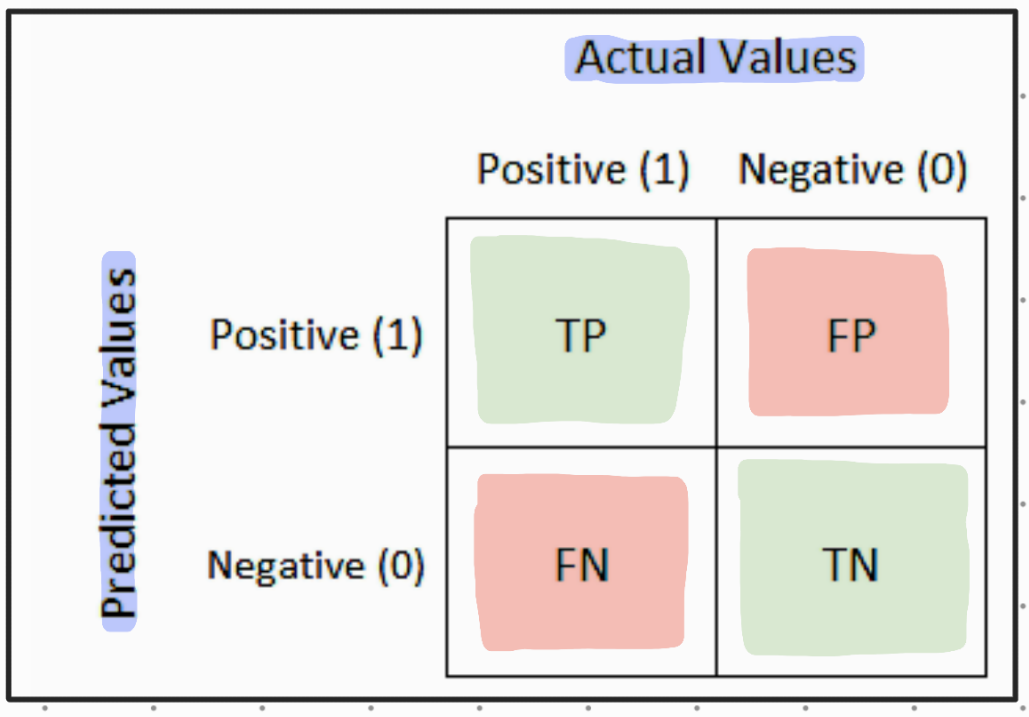
CHAPTER-05

LOGISTIC REGRESSION

Part-1 ✓ Part-2

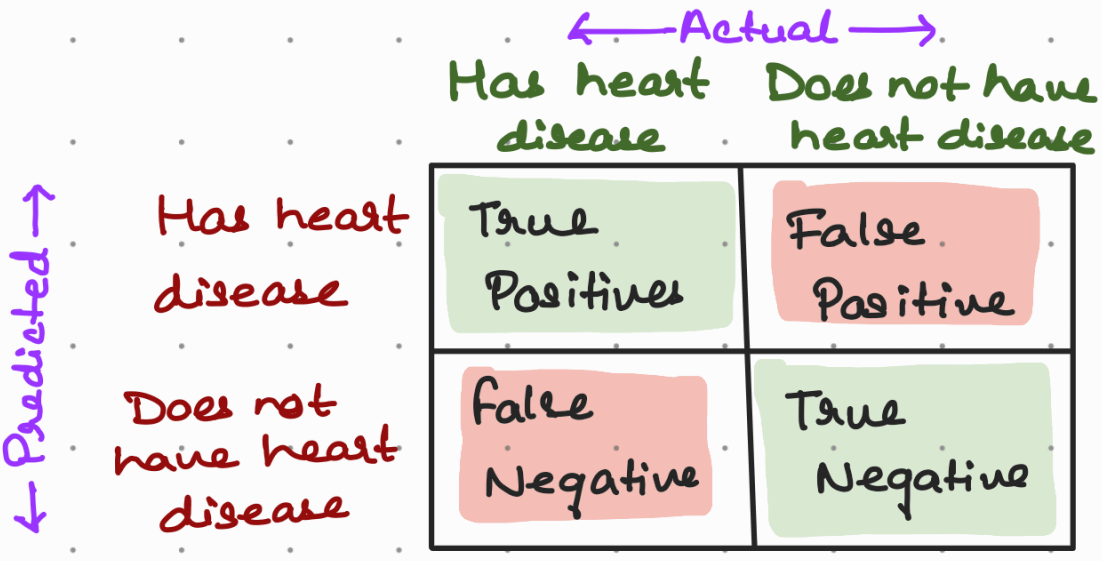
1. CONFUSION MATRIX

- A table of data that summarizes the performance of a ML model.
- Helps to understand where the model is performing well and where it is failing.
- Generally, confusion matrix is created for the classification models to know their performance on test data.



General Confusion Matrix

Let us take an example for a prediction if patients have heart diseases or not.



Here only 2 diagonal conditions are correct as both the predicted and actual values match.

Confusion matrix is created by calling confusion_matrix() function of sklearn.metrics module.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
```

It can be shown graphically using a Heatmap by passing the Confusion Matrix Object to heatmap() of seaborn package.

```
sns.heatmap(cm, annot = True)
```

↓ will show values in heatmap.

@Shreyan

2. Multiclass Classification using Logistic Regression :

In binary classification, the ML model gives prediction as 0 or 1.

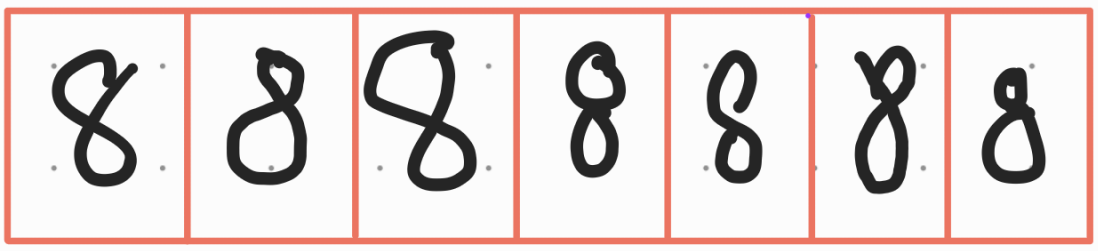
eg- 0 for 'No', 1 for 'Yes'.

When the dependent variable has more than 2 possible values, it is called **Multiclass Classification**.

Suppose, we want to identify a numeric digit that is written on paper by a person.

The problem is, different people write the same digit in different ways.

eg- the number 8 can be written as -



So if a task is given that the program should identify each handwritten digit properly, we can classify each image into one of the 10 digits.

So since 10 classes of images are there, this task comes under the Multiclass Classification.

@Shreyan

Things To Know :

@Shreyan

- Scikit-learn.org has several in-built data sets where we can find hand written digits data sets that contain a total of 1797 images, each of 8x8 bits in size (which is quite tiny)

- We can call this using load_digits() funation, as -

```
from sklearn.datasets import load_digits
digits = load_digits()
```

↑ This represents a Bunch type of objects belonging to utils module of sklearn package.

- Bunch is **not** dataframe. A dataframe represents data in the form of rows and columns. But a bunch object is similar to a dictionary where the data is stored in the form of key-value pairs.

- Key/column names in 'digits' bunch object:

- **data** : data[0] represents 0th row corresponding to 0th image. So there are 1797 rows for 1797 images.
- **DESCR** : description of dataset.
- **feature_names** : column names/pixels.
- **frame** : NoneType object that does not contain data.
- **images** : each row out of 1797, has 8 rows and 8 columns.
- **target** : actual digit names for each image.
- **target_names** : represents unique name of target.

1

Let us use Logistic Regression Machine Learning model to classify the hand written digits.

1. First train the model and then apply the model on sample images.
2. Also draw Confusion Matrix to know where our model was successful nand where it failed

In [1]:
```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
digits = load_digits()
```

In [3]:
```
#display the names of the cols in Bunch Object
dir(digits)
```

Out[3]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

In [4]:
```
#display the number of rows and cols in the datsets
digits.data.shape
```

Out[4]:
```
(1797, 64)
```

We shall be using the following :

- data[] is representation of array of the image
- images[] is the corresponding image
- target[] is the actual digit

To view the data related to an image, we should use digits.data[]. For example to view the data corresponding to **image 8**, we can use **digits.data[8]**

In [5]:
```
#to display 8th image array
digits.data[8]
```

Out[5]:
```
array([ 0.,  0.,  9., 14.,  8.,  1.,  0.,  0.,  0.,  0., 12., 14., 14.,
       12.,  0.,  0.,  0.,  0.,  9., 10.,  0., 15.,  4.,  0.,  0.,  0.,
        3., 16., 12., 14.,  2.,  0.,  0.,  0.,  4., 16., 16.,  2.,  0.,
        0.,  0.,  3., 16.,  8., 10., 13.,  2.,  0.,  0.,  1., 15.,  1.,
        3., 16.,  8.,  0.,  0.,  0., 11., 16., 15., 11.,  1.,  0.])
```

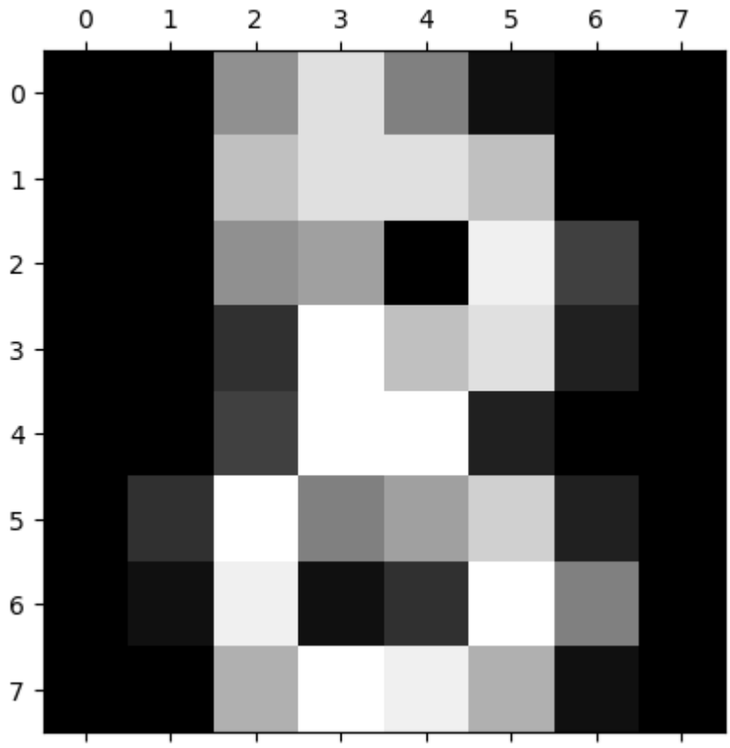
0 → black

When the image '8' was scanned and converted into an array of values, the above array is formed.

In [6]:
```
plt.gray() #uses gray colour to show image
plt.matshow(digits.images[8]) #display 8th image
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x7f7c153b3750>
<Figure size 640x480 with 0 Axes>
```

@Shreyan



The greater is the number the more is the block whiter in shade, so 0 is black.

In [7]:
```
#let us see the correct numeric digit of the above image
digits.target[8]
```

Out[7]: 8

We can use the data and target for training the model, let us split the training and testing data -

- x_train = array of handwritten digit
- y_train = digit

Let us take a default 70-30 train-test split to work with.

In [8]:
```
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test
```

In [9]:
```
model = LogisticRegression()
model.fit(x_train, y_train)
```

Out[9]:
```
▾ LogisticRegression
LogisticRegression()
```

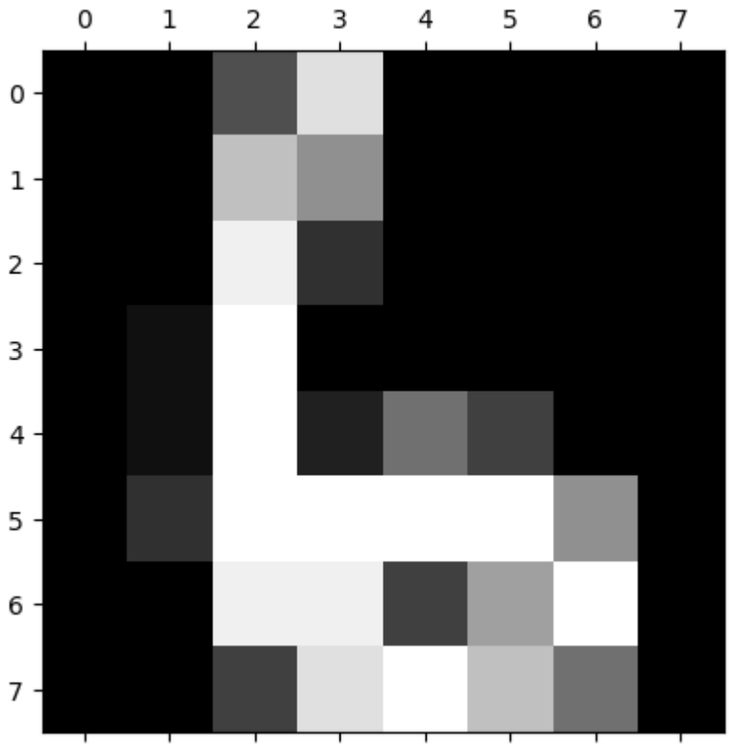
Check the model accuracy -

In [10]:
```
model.score(x_test, y_test)
```

Out[10]: 0.9666666666666667 → 96.66%

In [11]:
```
#let us take a random sample for prediction
#this is an image of a digit
plt.matshow(digits.images[67])
```

Out[11]: <matplotlib.image.AxesImage at 0x7f7c15286c90>



@Shreyan

Here we tested our model with the 67th sample from the data set. The data corresponding to this sample was passed to predict() method to judge the digit shown by the sample.

In [12]:
```
#the corresponding data for this image is data[67]
#let us supply this data for our model -> gives 6
model.predict([digits.data[67]])
```

Out[12]:
```
array([6])
```

Our model predicted here that it is digit 6.

In [13]:
```
#what is the actual digit?
digits.target[67]
```

Out[13]:
```
6
```

Let us now predict digits for data [30:35] -

In [14]:
```
model.predict(digits.data[30:35])
```

Out[14]:
```
array([0, 9, 5, 5, 6])
```
✔

Verify by comparing them with actual target digits -

In [15]:
```
#gives array([0, 9, 5, 5, 6])
digits.target[30:35]
```

Out[15]:
```
array([0, 9, 5, 5, 6])
```
✔

In [16]:
```
#to view the above images
# plt.gray()
# for i in range(30,35):
#     plt.matshow(digits.images[i])
```

Drawing Confusion Matrix :

1. Find the predicted values by the model when test data is given -

In [17]:
```
y_predicted = model.predict(x_test)
```

2. Create confusion matrix by passing expected data and predicted data -

In [18]:
```
cm = confusion_matrix(y_test, y_predicted)
cm
```

Out[18]:
```
array([[50,  0,  0,  0,  0,  2,  0,  0,  0,  0],
       [ 0, 39,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  1, 44,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 51,  0,  1,  0,  0,  4,  1],
       [ 0,  0,  0,  0, 58,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  1, 49,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 55,  0,  1,  0],
       [ 0,  0,  0,  1,  0,  0,  0, 63,  0,  0],
       [ 0,  3,  0,  0,  0,  0,  0,  0, 59,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  2, 54]])
```

3. Define width and height of the figure in inches -

In [19]:
```
#plt is pyplot object
plt.figure(figsize=(10,7))
```

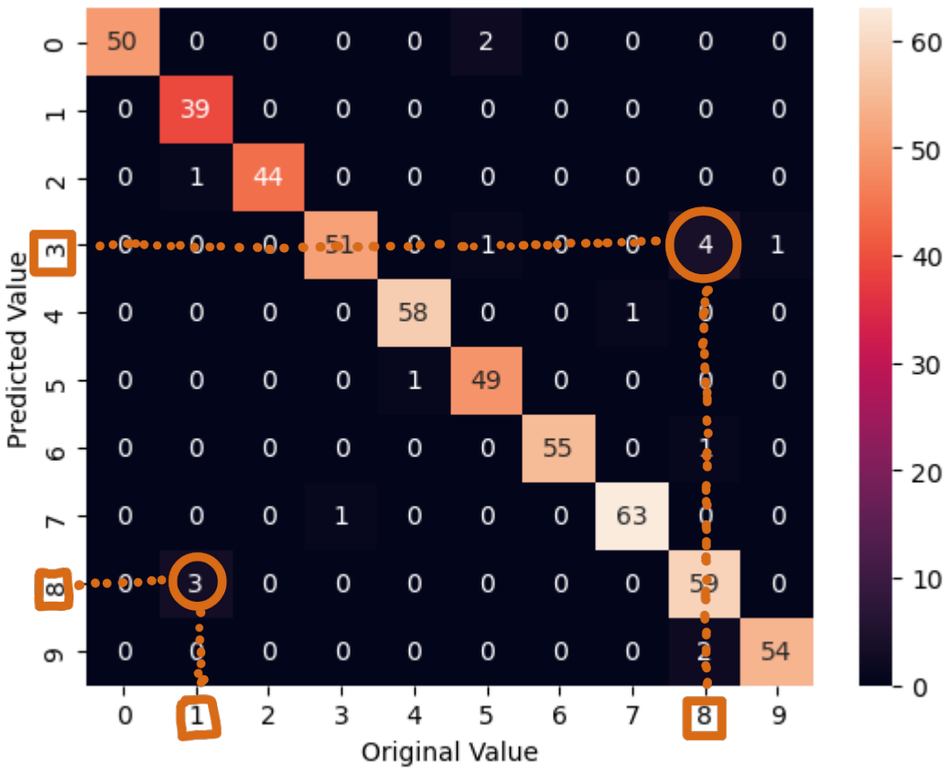
Out[19]:
```
<Figure size 1000x700 with 0 Axes>
```
<Figure size 1000x700 with 0 Axes>

@Shreyan

4. Draw heatmap with values -

In [20]:
```
sns.heatmap(cm, annot=True)
plt.xlabel('Original Value')
plt.ylabel('Predicted Value')
```

Out[20]:
```
Text(50.722222222222214, 0.5, 'Predicted Value')
```



Now understand carefully here -

In the confusion matrix shown above, observe the diagonal elements or values. They are **50, 39, 44 .... 63, 59, 54** , But **how to understand them ?**

1. Observe that on X axis we are plotting the original values and on Y axis we are showing predicted values by the model.
2. So, in the diagonal, **50** tells that the digit '0' was predicted for **50** times accurately as 0.
3. The value **39** tells that 1 was predicted **39** times as 1.

Now let us take the other values in the matrix. They represent how many times our model predicted incorrectly. If we see at the circled points, we can see that **1 was predicted as 8 , 3 times**; and **8 was predicted as 3 , 4 times.**

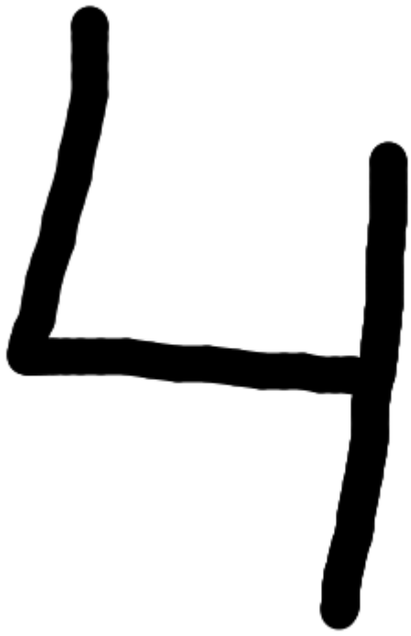
2 Use the model to Identify your Own Hand-Written Digit :

In [1]:
```
from PIL import Image, ImageOps
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

import numpy as np
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
digits = load_digits()
```

In [3]:
```
img = Image.open('four.png').convert('L')
img
```

Out[3]:



draw a digit in the Paint software and resize it to 400px x 400px and save it.

In [4]:
```
# Resize the image to 8x8 pixels
img_resized = img.resize((8, 8))
img_resized
```

Out[4]:

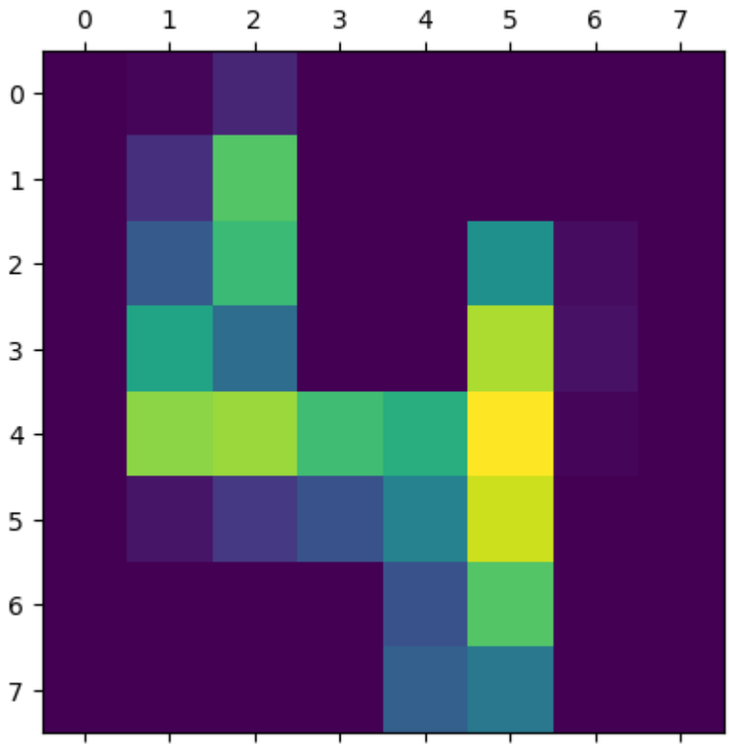
In [5]:
```
#invert the colors, black to white and white to black
img_inv = ImageOps.invert(img_resized)
img_inv
```

Out[5]:

In [6]:
```
plt.matshow(img_inv)
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x7f7181c1c3d0>
```

@Shreyan



@Shreyan

In [7]:
```
#convert image into an array - gives 2D array
arr = np.array(img_inv)
arr
```

Out[7]:
```
array([[  0,   2,  13,   0,   0,   0,   0,   0],
       [  0,  16,  88,   0,   0,   0,   0,   0],
       [  0,  34,  82,   0,   0,  60,   4,   0],
       [  0,  70,  43,   0,   0, 105,   6,   0],
       [  0,  99, 102,  83,  75, 120,   2,   0],
       [  0,   7,  20,  30,  53, 111,   0,   0],
       [  0,   0,   0,   0,  30,  88,   0,   0],
       [  0,   0,   0,   0,  37,  48,   0,   0]], dtype=uint8)
```

In [8]:
```
#convert above 2D into 1D array for prediction
arr1 = arr.flatten()
arr1
```

Out[8]:
```
array([  0,   2,  13,   0,   0,   0,   0,   0,   0,  16,  88,   0,   0,
         0,   0,   0,   0,  34,  82,   0,   0,  60,   4,   0,   0,  70,
        43,   0,   0, 105,   6,   0,   0,  99, 102,  83,  75, 120,   2,
         0,   0,   7,  20,  30,  53, 111,   0,   0,   0,   0,   0,   0,
        30,  88,   0,   0,   0,   0,   0,   0,  37,  48,   0,   0],
      dtype=uint8)
```

In [9]:
```
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test
model = LogisticRegression()
model.fit(x_train, y_train)
```

Train using the same 70-30 split.

Out[9]:
```
▾ LogisticRegression
LogisticRegression()
```

In [10]:
```
model.predict([arr1])
```

Out[10]: `array([4])` ← My image is correctly identified.

ERROR HANDLING

Possible ValueError might occur: X has 160000 features, but LogisticRegression is expecting 64 features as input.

This discrepancy is likely because the input data is a flattened 1D array representing an image with a resolution of 400x400 pixels, resulting in 160,000 elements. However, the logistic regression model is trained on the digits dataset, where each image has a resolution of 8x8 pixels, resulting in 64 features.

To resolve this issue, you need to resize the input image to match the resolution of the images in the digits dataset (8x8 pixels) before making predictions. Once the image is resized, you can flatten it to a 1D array with 64 elements and then pass it to the model for prediction.