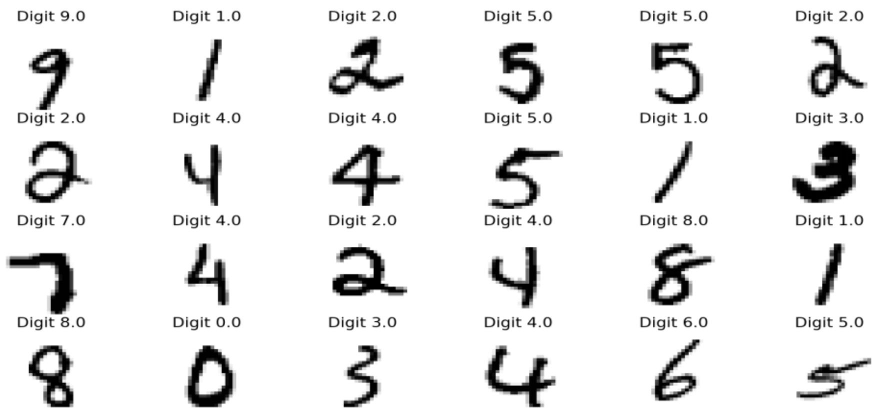
Recognizing Handwritten Digits with Scikit-learn

The handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image. Think about, for example, the ZIP codes on letters at the post office and the automation needed to recognize these five digits. Perfect recognition of these codes is necessary in order to sort mail automatically and efficiently. Included among the other applications that may come to mind is OCR (Optical Character Recognition) software. OCR software must read the handwritten text, or pages of printed books, for general electronic documents in which each character is well defined. ***Digits*** data set consists of 1,797 images that are 8x8 pixels in size. Each image is a handwritten digit in grayscale.



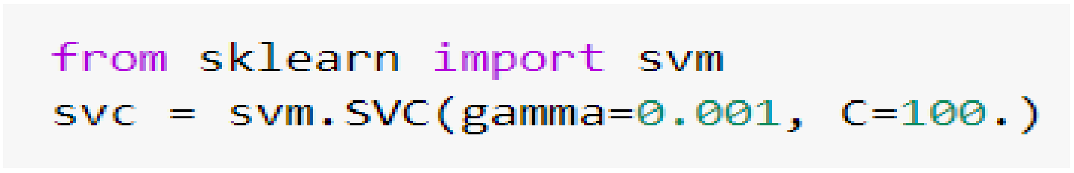
Hypothesis :

The Digits data set of the Scikit-learn library provides numerous data-sets that are useful for testing many problems of data analysis and prediction of the results. Some Scientist claims that it predicts the digit accurately 95% of the times. Perform data Analysis to accept or reject this Hypothesis. ***Scikit-Learn*** is a library for Python that contains numerous useful algorithms that can easily be implemented and altered for the purpose of classification and other machine learning tasks.

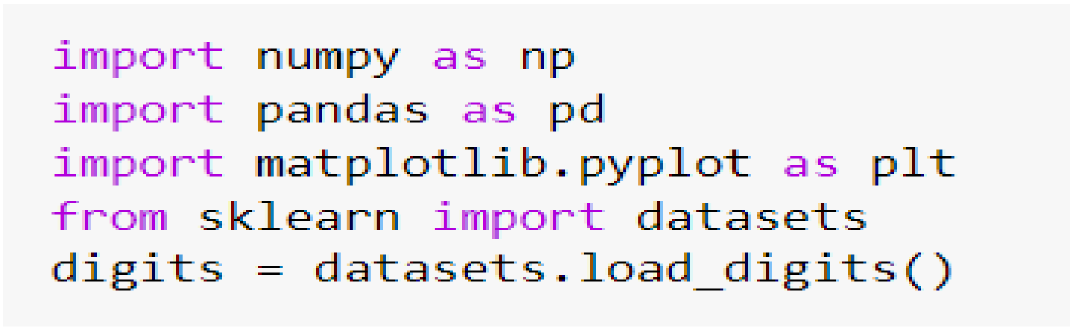
Prerequists :

- Sklearn
- Matplotlib
- Basics of Machine learning

<*1*>.Importing the libraries, we use numpy, matplotlib, sklearn and seaborn.



```
from sklearn import svm
svc = svm.SVC(gamma=0.001, C=100.)
```



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
digits = datasets.load_digits()
```

<2>.The images of the handwritten digits are contained in a ***digits.images*** array. Each element of this array is an image that is represented by an 8x8 matrix of numerical values that correspond to a grayscale from white, with a value of 0, to black, with the value 15.

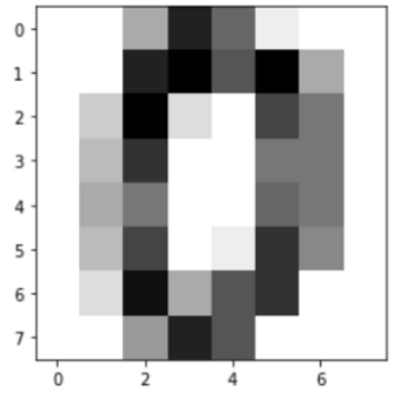
```
digits.images[0]
```

```
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.]])
```

<*3*>.Our data-set is stored in digits. By the command given below, you will obtain a grayscale image of digit.

```
%matplotlib inline
plt.imshow(digits.images[0], cmap=plt.cm.gray_r, interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x7f5c7e212940>
```



<*4*>.The numerical values represented by images, i.e., the targets, are contained in the ***digit.targets*** array. And also the dataset is a training set consisting of 1,797 images. We can determine if that is true.

This is optional command just for checking

```
digits.target
```

```
array([0, 1, 2, ..., 8, 9, 8])
```

```
digits.target.size
```

```
1797
```

<*5*>. Visualizing the images and labels in our Dataset.

This dataset contains 1,797 elements, and so let us consider the first 1,791 as a training set and will use the last six as a validation set. We can see in detail these six handwritten digits by using the matplotlib library

Visualizing the images of six digits of the validation set

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.subplot(321)
plt.imshow(digits.images[1791], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(322)
plt.imshow(digits.images[1792], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(323)
plt.imshow(digits.images[1793], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(324)
plt.imshow(digits.images[1794], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(325)
plt.imshow(digits.images[1795], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(326)
plt.imshow(digits.images[1796], cmap=plt.cm.gray_r,
interpolation='nearest')
```

<*6*>. Now we are training the svc estimator that we have defined earlier.

```
svc.fit(digits.data[1:1790], digits.target[1:1790])
```

```
SVC(C=100.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

As we can see that the svc estimator has learned correctly. It is able to recognize the handwritten digits, interpreting correctly all six digits of the validation set.

<*7*>.Now let us see the Scikit-Learn ***4-Step Modeling Pattern***.

First let's split our Dataset into training and test sets to make sure that after we train our model, it is able to generalize well to new data

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.25, random_state=0)
```

STEP_1:

Importing the model we want to use.

Importing using ***Logistic Regression***.

```
from sklearn.linear_model import LogisticRegression
```

STEP_2:

Making an instance of the Model.

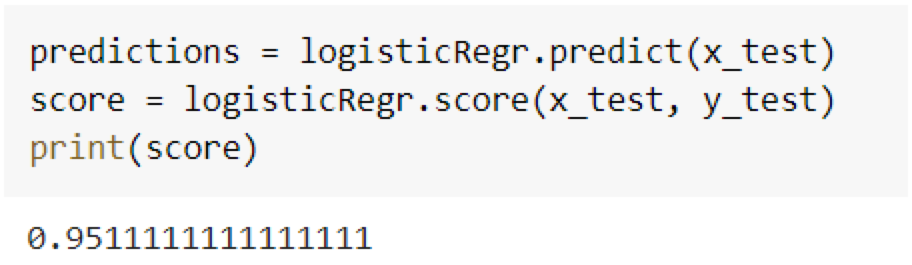
```
logisticRegr = LogisticRegression()
```

STEP_3:

```
logisticRegr.fit(x_train, y_train)
```

STEP_4:

Predicting the labels of new data and measuring performance of our model.



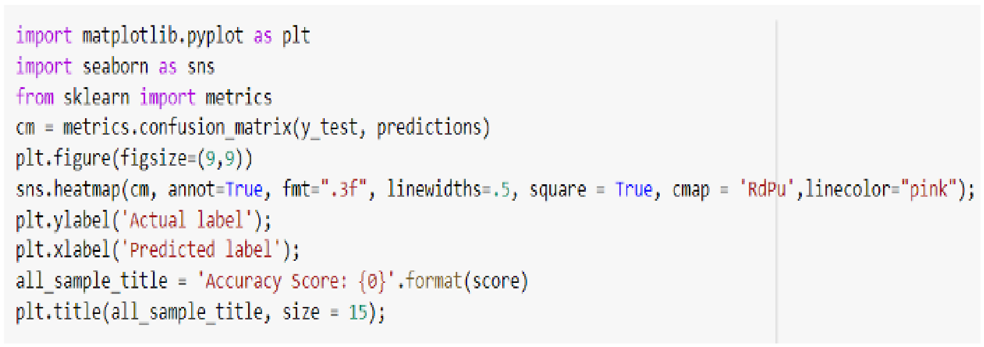
```
predictions = logisticRegr.predict(x_test)
score = logisticRegr.score(x_test, y_test)
print(score)
```

```
0.9511111111111111
```

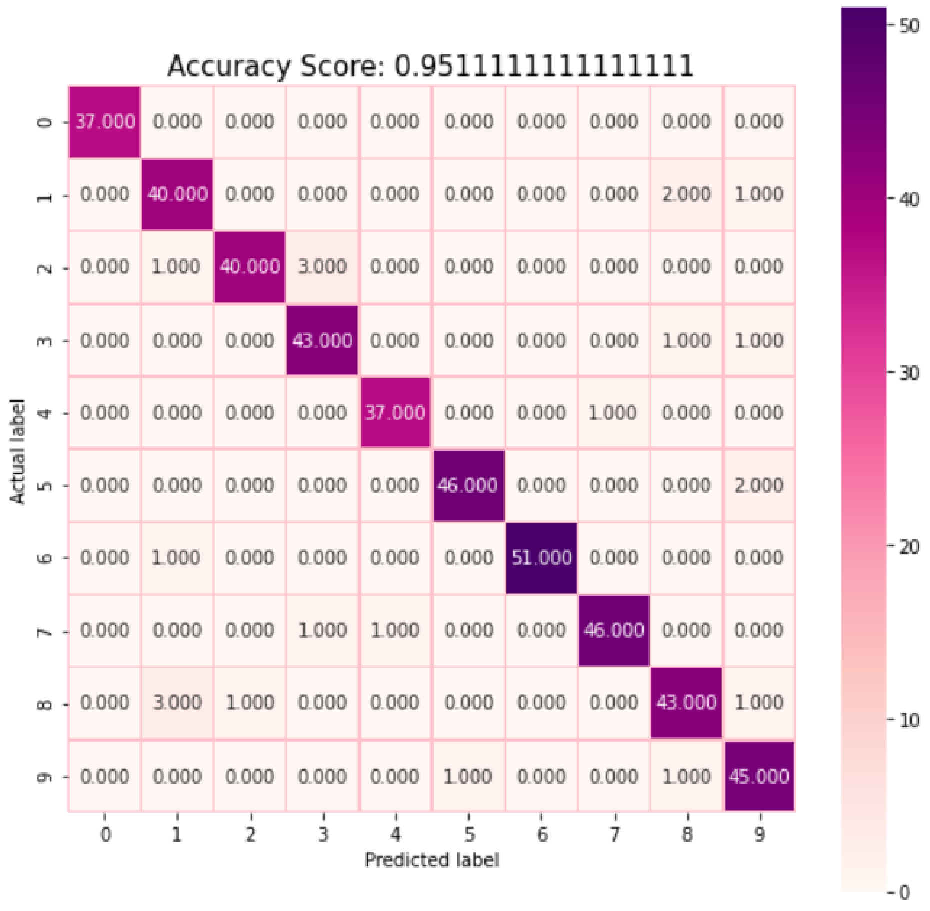
<***8***>. Confusion matrix:

A confusion matrix is a table that is often used to evaluate the accuracy of a classification model. We can use Seaborn or Matplotlib to plot the confusion matrix. We will be using Seaborn for our confusion matrix.

Code for Confusion matrix



```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, predictions)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'RdPu',linecolor="pink");
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```



	0	1	2	3	4	5	6	7	8	9
0	37.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1	0.000	40.000	0.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000
2	0.000	1.000	40.000	3.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	43.000	0.000	0.000	0.000	0.000	1.000	1.000
4	0.000	0.000	0.000	0.000	37.000	0.000	0.000	1.000	0.000	0.000
5	0.000	0.000	0.000	0.000	0.000	46.000	0.000	0.000	0.000	2.000
6	0.000	1.000	0.000	0.000	0.000	0.000	51.000	0.000	0.000	0.000
7	0.000	0.000	0.000	1.000	1.000	0.000	0.000	46.000	0.000	0.000
8	0.000	3.000	1.000	0.000	0.000	0.000	0.000	0.000	43.000	1.000
9	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	1.000	45.000

Accuracy matrix

<*9*>. From this article, we can see how easily we can import a dataset, build a model using Scikit-Learn, train the model, make predictions with it, and can find the accuracy of our prediction(which in our case is ***95.11%***).

As we can clearly see above, 95% of our models the achieved accuracy is 100% . Hence we can easily conclude that our model works for more than 95% of the time.