PROJECT TITLE:

The following code implements a single layer perceptron on the wheat seeds dataset. There are 3 classes each denoting a variety of wheat (Kama, Rosa and Canadian). Out of which for the sake of simplicity I have clubbed 2 classes into one class (Kama and Canadian and as a single class and Rosa as the other class). The Perceptron classifies the dataset into 2 classes. Kama and Canadian as 1 and Rosa as 2. The accuracy that we get is 50%.

MOTIVATION:

This project was developed as part of an assignment for the subject Soft Computing.
Language used: Python 3

SCREENSHOTS:

```
attributes=7
learning_parameter=0.1
epochs=5000
N = Neuron(attributes,learning_parameter)

for j in range(epochs):
    for i in range(len(X_train)):
        N.update_weights(X_train[i],Y_train[i])

count=0

for i in range(len(Y_test)):
    Y_pred = N.predict(X_test[i])
    error = Y_test[i] - Y_pred
    if(error):
        count += 1
print('Accuracy:',100 -(count/len(X_test))*100)
```

```
Accuracy: 42.85714285714286
```

TECH/FRAMEWORK USED:

Anaconda navigator was used to implement the project

FEATURES:

The code is a very easy to understand implementation of the single layer perceptron. Modularity has been introduced so that the function of each module can be understood. Object Oriented concepts have been used which ensures that multiple neurons can be created from the same "Neuron" class and can be extended to create larger networks.

CODE EXAMPLE:

The code implements a single layer perceptron via the following functions:

1)Preprocessing: The values have been first preprocessed by normalizing the data. The values have been converted to float and then normalized. This ensures that the values of the dataset are between 0 and 1.

2) init: It's a constructor which initializes a neuron. It initializes the weight matrix and accepts the learning rate which is passed by the driver code

3) predict: This function calculates the weighted sum of the weight vector and the input vector and returns a 1 if the value is greater than 0 or else returns -1

4)update weights: this function is for training the weights. It first uses the predict function to find the predicted output. the predicted output is subtracted from the desired output to obtain the error. If an error is there then the new weight vector is calculated by adding the change in weight to the previous weight. The change in weight is found by multiplying the input vector, the learning rate and the error.

5)driver code:
The driver code splits the dataset into 4 arrays namely X_train, Y_train, X_test and Y_test. The X_train consists of rows of training data points with the attribute values and the bias(=1). the Y_train consists of the corresponding class labels of the training dataset. Similarly, X_test and Y_test consists of the testing data points and class labels respectively.

here we have adopted a train test split of 90-10, i.e 90 percent points of the dataset are training points and the rest are testing points.

The random module has been used to randomly pick training and testing points.

A neuron is created with the number of attributes and the learning parameter passed to the constructor.
The value of epochs has been kept high because the size of the dataset is small. This improves accuracy.

Finally for each epoch the number of misclassifications are counted and the accuracy score is calculated.

HOW TO USE:
Specify the path to the dataset. make sure the dataset is in csv or comma seperate format. Define the number of the epochs, the value of the learning rate and the number of attributes and you are good to go.

TESTS:

```
attributes=7
learning_parameter=0.1
epochs=5000
N = Neuron(attributes,learning_parameter)

for j in range(epochs):
    for i in range(len(X_train)):
        N.update_weights(X_train[i],Y_train[i])

count=0

for i in range(len(Y_test)):
    Y_pred = N.predict(X_test[i])
    error = Y_test[i] - Y_pred
    if(error):
        count += 1
print('Accuracy:',100 -(count/len(X_test))*100)
```

```
Accuracy: 70.0
```