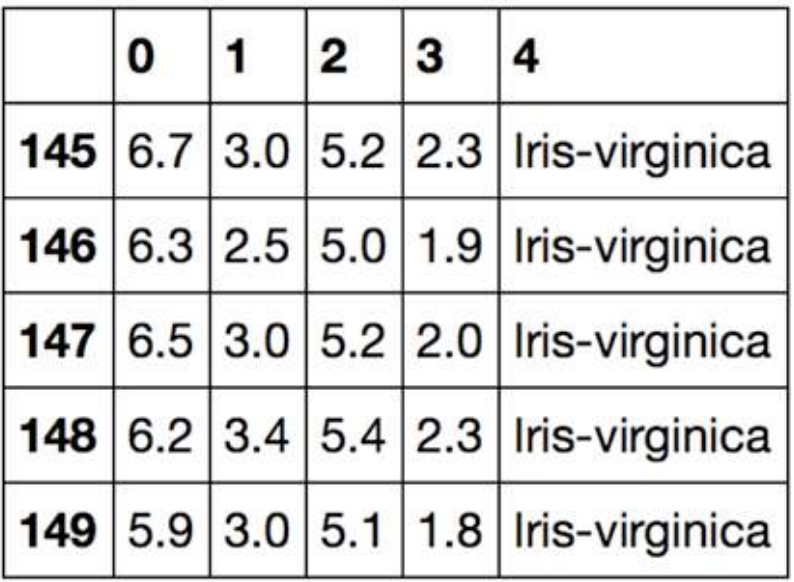
**Assignment 3: Multi level Perceptron**

[Implementation of a single perceptron](https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L03_perceptron/code/perceptron-numpy.ipynb) You basically have to extend this to MLP and understand the code behind this. I have given this to make your life easier.

Let's now implement MLP in Python and apply it to the Iris dataset

First, we will use the pandas library to load the Iris dataset directly from the UCI Machine Learning Repository into a DataFrame object and print the last five lines via the tail method to check that the data was loaded correctly:

URL: [IRIS DATASET](https://archive.ics.uci.edu/ml/machine-learning-databases/iris/%20iris.data)



We will take an object-oriented approach to defining the perceptron interface as a Python class, which will allow us to initialize new Perceptron objects that can learn from data via a fit method, and make predictions via a separate predict method. As a convention, we append an underscore (\_) to attributes that are not created upon the initialization of the object, but we do this by calling the object's other methods, for example, self.w\_. We will then use objects of this class to populate the MLP.

Initialize new Perceptron objects with a given learning rate, eta, and the number of epochs, n\_iter (passes over the training dataset). Via the fit method, we initialize the weights in self.w\_ to a vector, ℝm+!, where m stands for the number of dimensions (features) in the dataset, and we add 1 for the first element in this vector that represents the bias unit. Remember that the first element in this vector, self.w\_[0], represents the so-called bias unit that we discussed earlier.

It is important to keep in mind that we don't initialize the weights to zero because the learning rate, 𝜂 (eta), only has an effect on the classification outcome if the weights are initialized to non-zero values. If all the weights are initialized to zero, the learning rate parameter, eta, affects only the scale of the weight vector, not the direction.

After the weights have been initialized, the fit method loops over all individual examples in the training dataset and updates the weights according to the perceptron learning rule

Note that we will also only consider two flower classes, Setosa and Versicolor, from the Iris dataset for practical reasons—remember, the perceptron is a binary classifier. However, the perceptron algorithm can be extended to multi-class classification

Now, it's time to train our perceptron algorithm on the Iris data subset that we just extracted. Also, we will plot the misclassification error for each epoch to check whether the algorithm converged and found a decision boundary that separates the two Iris flower classes. You must implement this.