In sklearn, different machine learning models are represented as classes. Below, we see a LogisticRegression classifier imported to use at our disposal.

In [1]:
```
import numpy as np
import sklearn.datasets # imports 'classical' machine learning datasets
from sklearn.linear_model import LogisticRegression # imports LogisticRegression class
from sklearn.model_selection import train_test_split # imports the function train_test_
                                                     # which allows us to easily split o
```

The iris dataset is built into sklearn and can be loaded with the function `sklearn.datasets.load_iris()`. However, `load_iris()` returns a dictionary, which contains both our input, `X`, and our target output, `y`, with the keys `'data'` and `'target'`, respectively.

In [2]:
```
iris_data = sklearn.datasets.load_iris()  # loads in the Iris dataset through sklearn.d
X = iris_data['data']  # gets the input values, our input X, from iris_data
y = iris_data['target'] # gets the target values, our output y, from iris_data
```

Note in the output of the cell below that X has 150 rows and 4 columns, while y is a 1D array with 150 values.

In [3]:
```
print(type(iris_data)) # prints the type of iris_data, which says 'sklearn.utils.Bunch'
                       # but is very similar to a Python dictionary
print(X.shape) # prints the shape of X
print(y.shape) # prints the shape of y
```

```
<class 'sklearn.utils.Bunch'>
(150, 4)
(150,)
```

Before training our model, we will split our model into a training set and testing set, so that we can check the accuracy of our model on unseen data.

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # splits our X
                                                                         # with the tes
                                                                         # of the train
```

In [5]:
```
clf = LogisticRegression()  # initializes a LogisticRegression class,
                            # which will allow us to train and test a Logistic Regressio
```

`clf.fit(X, y)` will train our LogisticRegression classifier to our training dataset.
In other words, the `fit` function updates our LogisticRegression object with the set of parameters W and bias vector B needed to make predictions.

In [6]:
```
clf.fit(X_train, y_train) # fits or trains our LogisticRegression model on our training
```

Out[6]:
```
LogisticRegression()
```

Once we fit our model, we have a set of weights and biases that we then apply to our test data to measure the accuracy of our model.

In [7]:
```
y_pred = clf.predict(X_test)  # predicts the y_test values based on our trained model a
y_pred[:5] # outputs our first 5 predictions (our output is either class 0, 1, or 2)
```

Out[7]:
```
array([0, 2, 1, 2, 2])
```

To evaluate our model's accuracy, we will see how many values from `y_pred` are equal to

`y_test` , divided by the length of either array.

In [8]:
```
sum(y_pred == y_test) / len(y_pred)  # gets the accuracy of our model by summing all th
                                     # divided by the length of y_pred
```

Out[8]:
```
0.9777777777777777
```