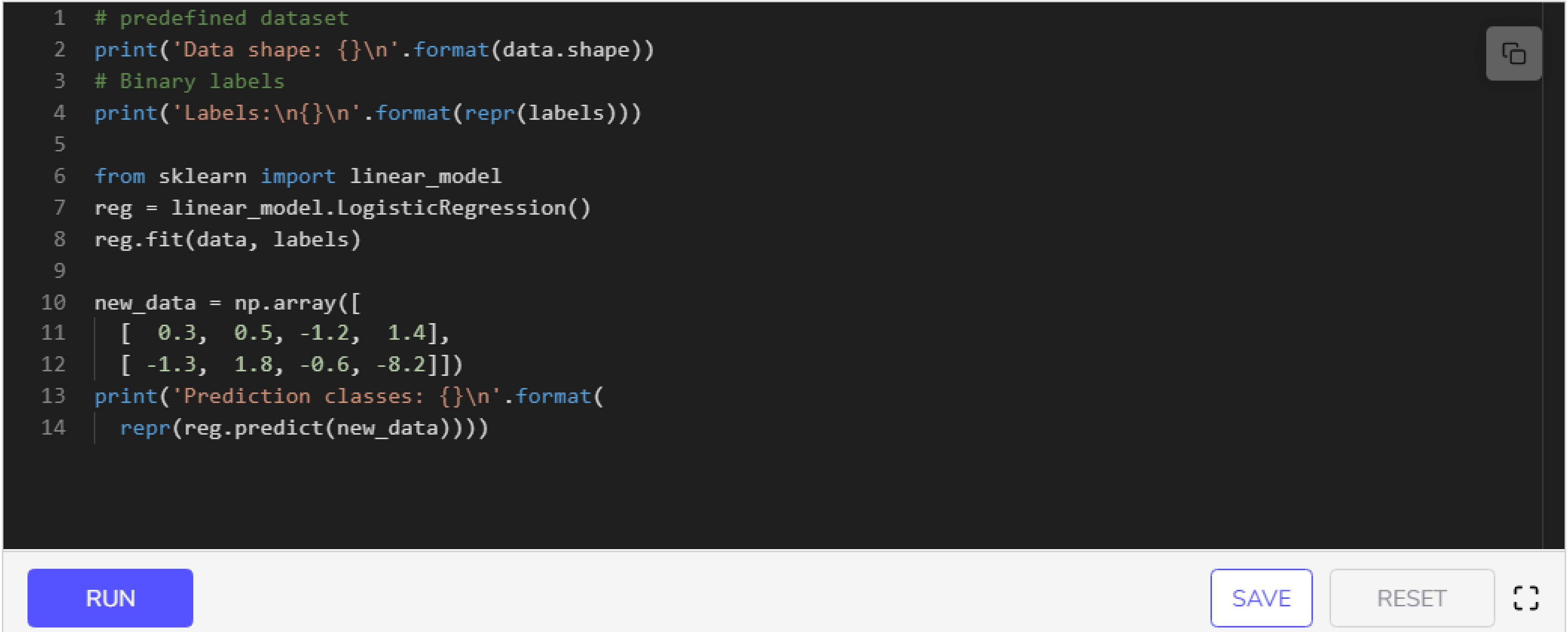
Thus far we've learned about several linear regression models and implemented them with scikit-learn. The logistic regression model, despite its name, is actually a linear model for *classification*. It is called logistic regression because it performs regression on logits, which then allows us to classify the data based on model probability predictions.

For a more detailed explanation of logistic regression, check out the **Intro to Deep Learning** section of this course, which implements logistic regression via a single layer perceptron model in TensorFlow.

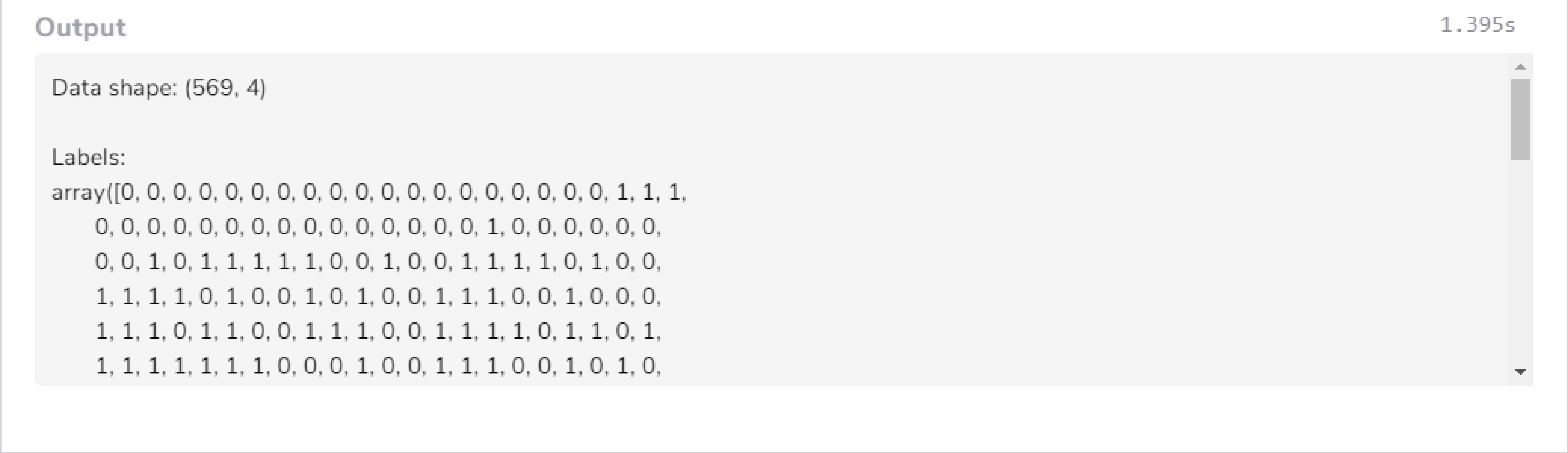
We implement logistic regression with the `LogisticRegression` object (part of the `linear_model` module). The default setting for `LogisticRegression` is *binary classification*, i.e. classifying data observations that are labeled with either a 0 or 1.



```
# predefined dataset
print('Data shape: {}\n'.format(data.shape))
# Binary labels
print('Labels:\n{}\n'.format(repr(labels)))

from sklearn import linear_model
reg = linear_model.LogisticRegression()
reg.fit(data, labels)

new_data = np.array([
  [  0.3,  0.5, -1.2,  1.4],
  [ -1.3,  1.8, -0.6, -8.2]])
print('Prediction classes: {}\n'.format(
  repr(reg.predict(new_data))))
```



Output 1.395s

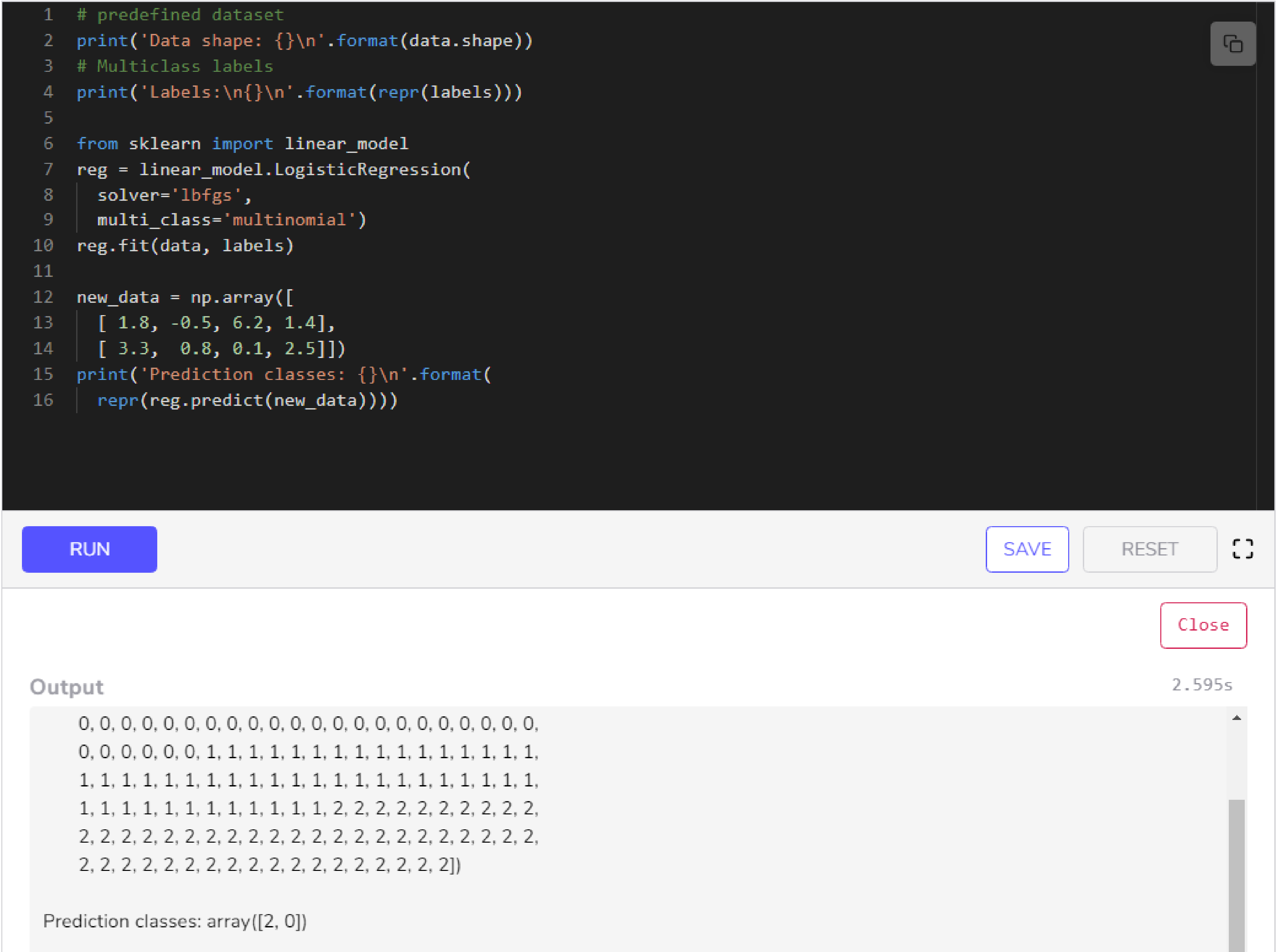
```
Data shape: (569, 4)

Labels:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
```

The code above created a logistic regression model from a labeled dataset. The model predicts 1 and 0, respectively, as the labels for the observations in `new_data`.

For *multiclass classification*, i.e. when there are more than two labels, we initialize the `LogisticRegression` object with the `multi_class` keyword argument. The default value is `'ovr'`, which signifies a One-Vs-Rest strategy. In multiclass classification, we want to use the `'multinomial'` strategy.

The code below demonstrates multiclass classification. Note that to use the `'multinomial'` strategy, we need to choose a proper solver (see below for details on solvers). In this case, we choose `'lbfgs'`.



```python
# predefined dataset
print('Data shape: {}\n'.format(data.shape))
# Multiclass labels
print('Labels:\n{}\n'.format(repr(labels)))

from sklearn import linear_model
reg = linear_model.LogisticRegression(
  solver='lbfgs',
  multi_class='multinomial')
reg.fit(data, labels)

new_data = np.array([
  [ 1.8, -0.5, 6.2, 1.4],
  [ 3.3,  0.8, 0.1, 2.5]])
print('Prediction classes: {}\n'.format(
  repr(reg.predict(new_data))))
```

```
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

Prediction classes: array([2, 0])
```

The code below demonstrates usage of the `solver` and `max_iter` keyword arguments.

```
from sklearn import linear_model
reg = linear_model.LogisticRegression(
  solver='lbfgs', max_iter=1000)
```

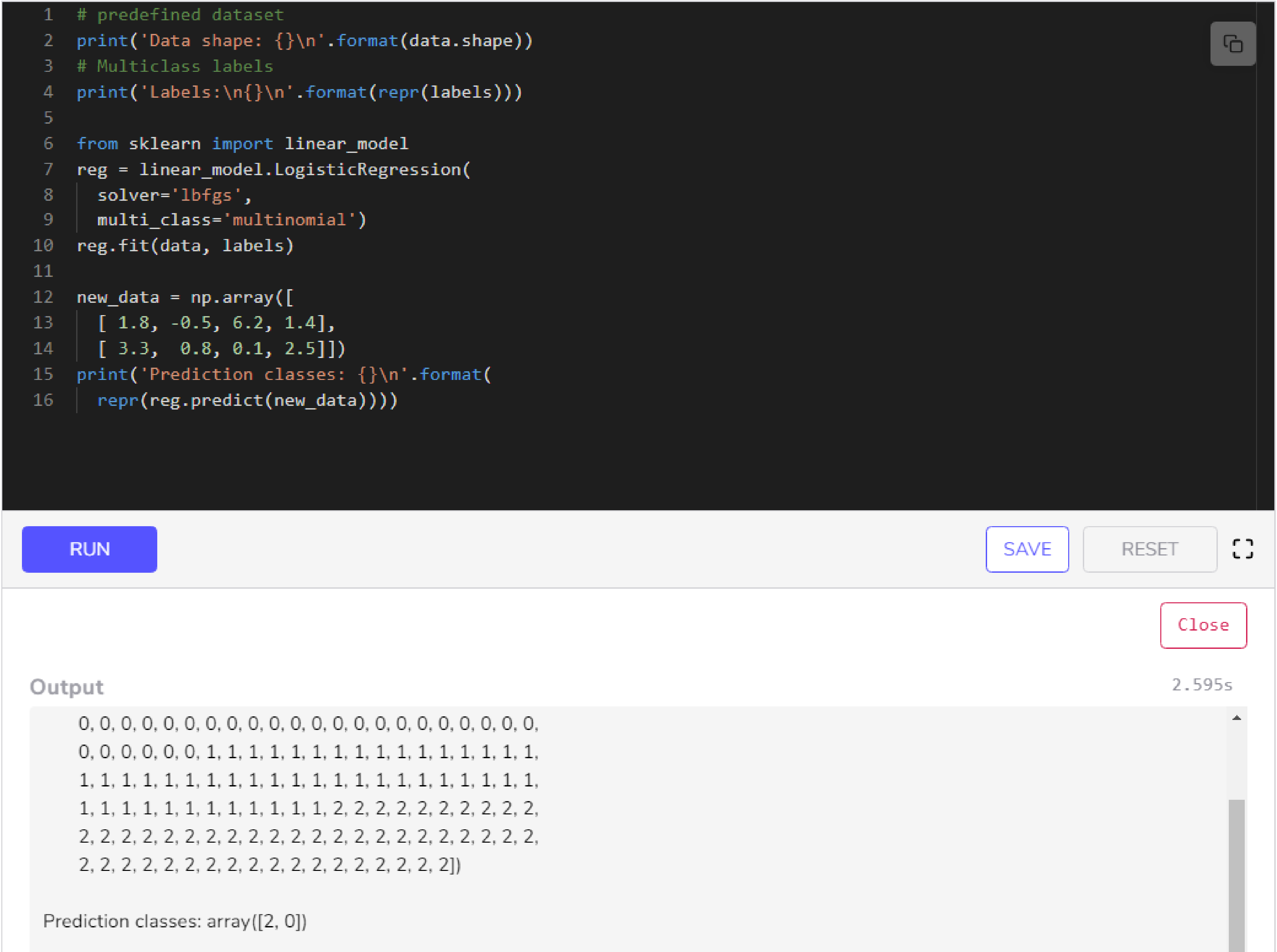
C. Cross-validated model

Like the ridge and LASSO regression models, the logistic regression model comes with a cross-validated version in scikit-learn. The cross-validated logistic regression object, `LogisticRegressionCV`, is initialized and used in the same way as the regular `LogisticRegression` object.

The code below demonstrates usage of the `LogisticRegressionCV` object.

```
from sklearn import linear_model
reg = linear_model.LogisticRegressionCV(
  solver='multinomial', max_iter=1000)
```

The code below demonstrates multiclass classification. Note that to use the `'multinomial'` strategy, we need to choose a proper solver (see below for details on solvers). In this case, we choose `'lbfgs'`.



```python
# predefined dataset
print('Data shape: {}\n'.format(data.shape))
# Multiclass labels
print('Labels:\n{}\n'.format(repr(labels)))

from sklearn import linear_model
reg = linear_model.LogisticRegression(
  solver='lbfgs',
  multi_class='multinomial')
reg.fit(data, labels)

new_data = np.array([
  [ 1.8, -0.5, 6.2, 1.4],
  [ 3.3,  0.8, 0.1, 2.5]])
print('Prediction classes: {}\n'.format(
  repr(reg.predict(new_data))))
```

```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

Prediction classes: array([2, 0])
```