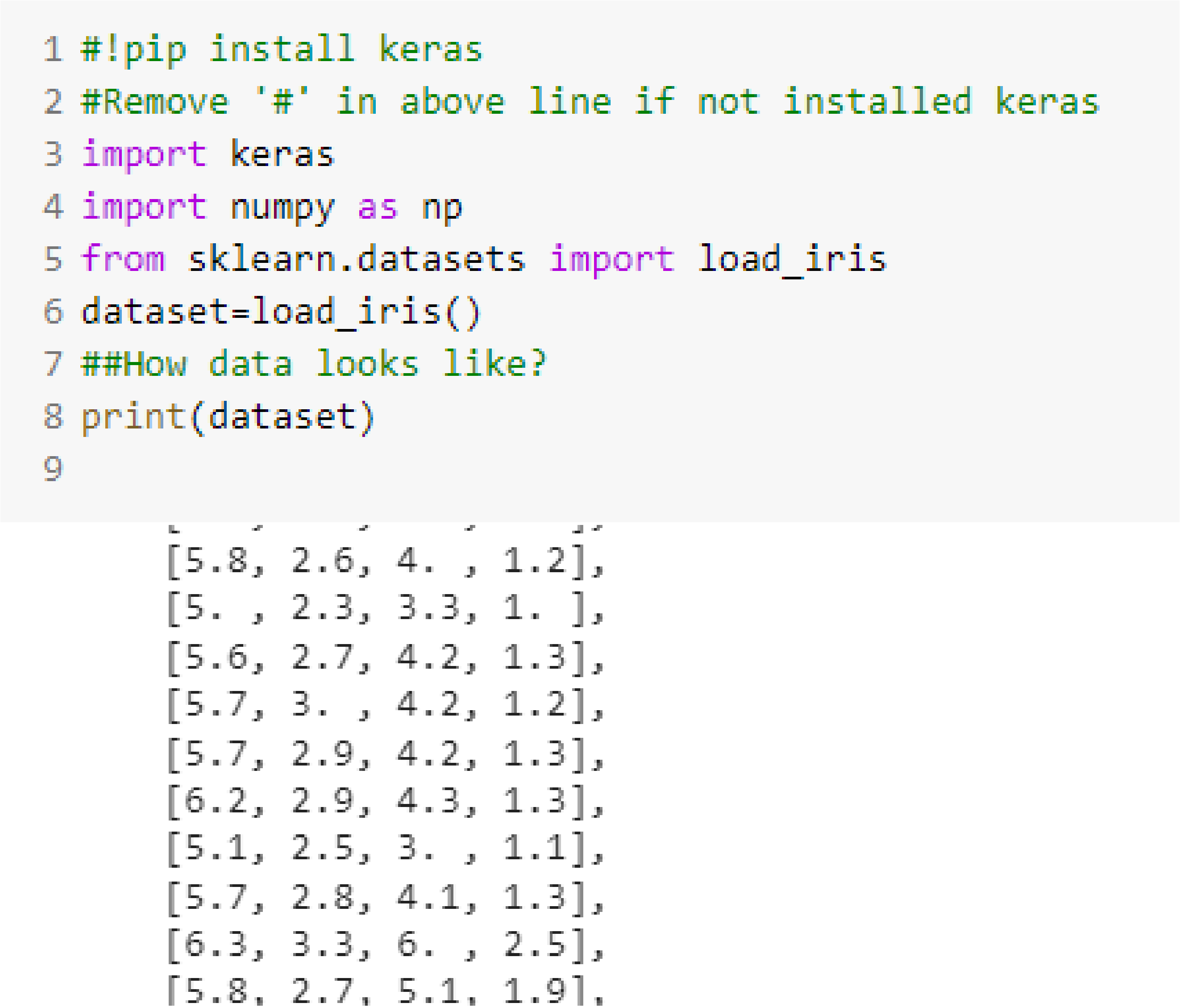
JARINGAN SYARAT TIRUAN (NEURAL NETWORK)

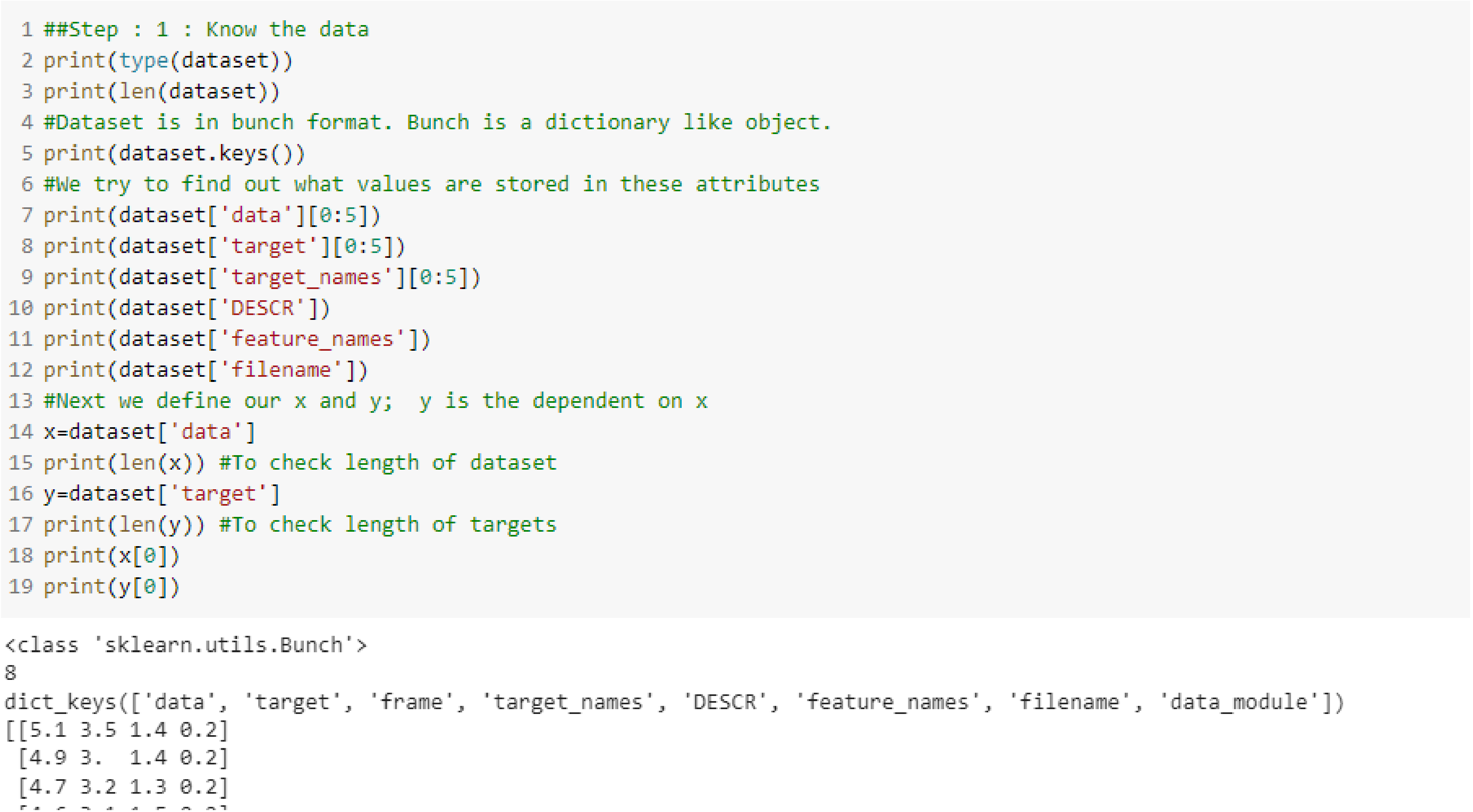
Contoh Kasus

KLASIFIKASI BUNGA IRIS DENGAN NEURAL NETWORK



```python
#!pip install keras
#Remove '#' in above line if not installed keras
import keras
import numpy as np
from sklearn.datasets import load_iris
dataset=load_iris()
##How data looks like?
print(dataset)

```

```
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
```



```
##Step : 1 : Know the data
print(type(dataset))
print(len(dataset))
#Dataset is in bunch format. Bunch is a dictionary like object.
print(dataset.keys())
#We try to find out what values are stored in these attributes
print(dataset['data'][0:5])
print(dataset['target'][0:5])
print(dataset['target_names'][0:5])
print(dataset['DESCR'])
print(dataset['feature_names'])
print(dataset['filename'])
#Next we define our x and y;  y is the dependent on x
x=dataset['data']
print(len(x)) #To check length of dataset
y=dataset['target']
print(len(y)) #To check length of targets
print(x[0])
print(y[0])
```

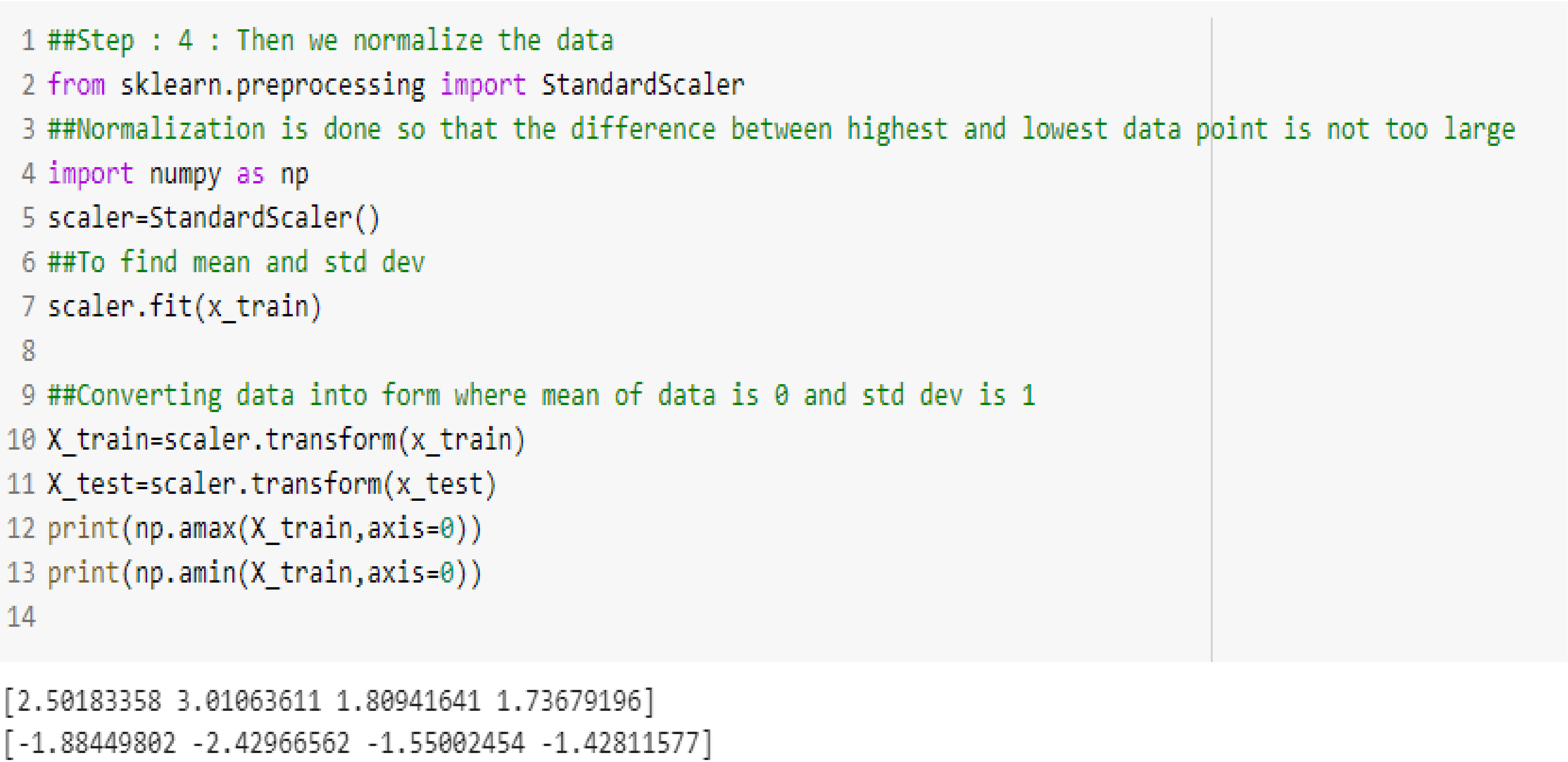
```
<class 'sklearn.utils.Bunch'>
8
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
```

```python
##Step : 2 : Convert y into one hot encoded vector
#One hot encoding is used because Y has labels (0,1,2)- which is a categorical categorical data with no ordinal relationships
from keras.utils import to_categorical
Ny=len(np.unique(y))
print(Ny)
Y=to_categorical(y,num_classes=Ny)
print(Y[0:5])
```

```
3
[[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]
```

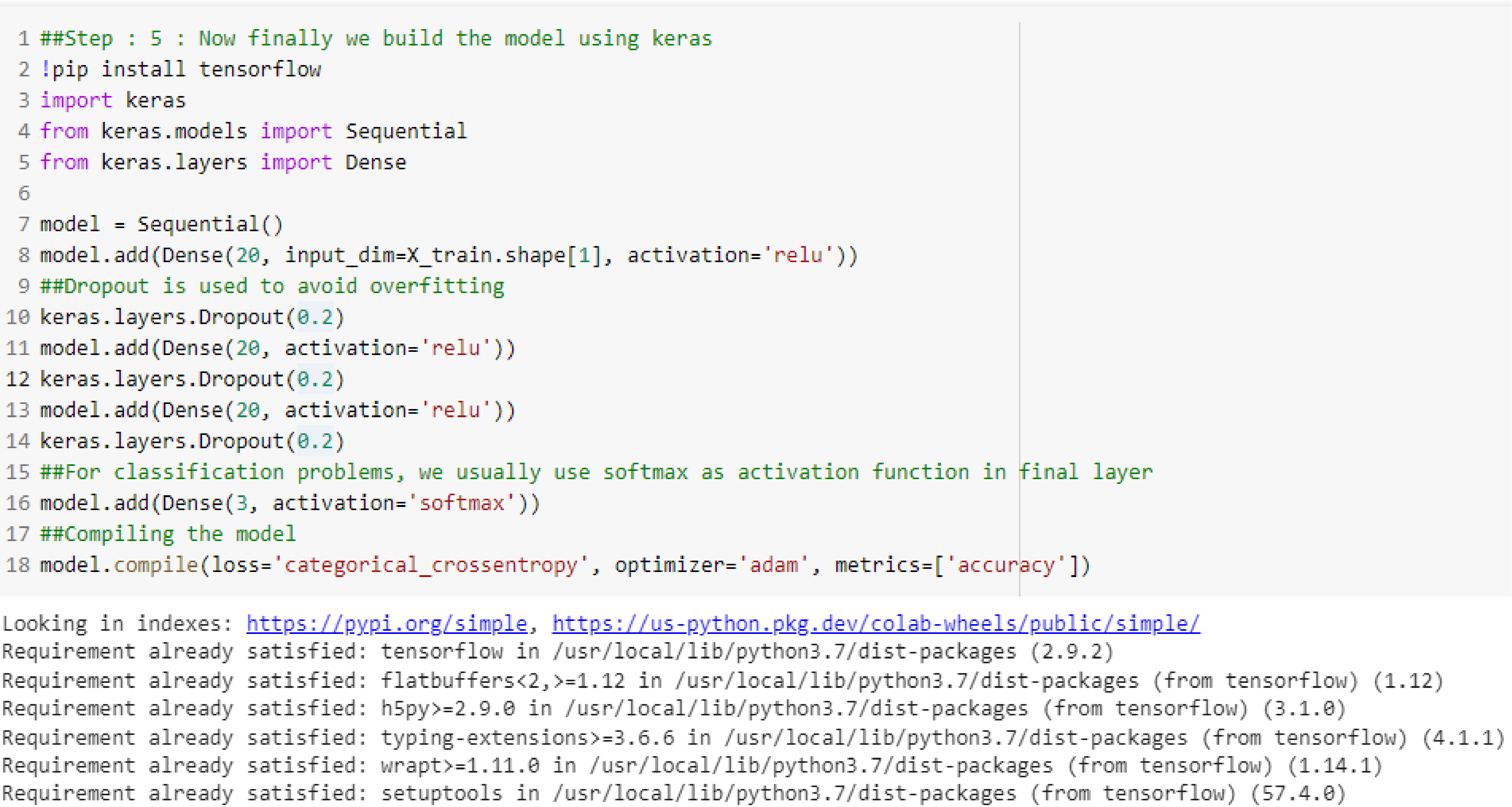
```python
##Step : 3 : Now we split the data into two parts - one for training another for testing
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,Y,test_size=0.10,shuffle=True)
print(x_train[0:5])
print(x_test[0:5])
print(y_train[0:5])
print(y_test[0:5])
```

```
[[5.4 3.9 1.7 0.4]
 [5.8 2.8 5.1 2.4]
 [6.7 3.1 4.4 1.4]
 [4.9 3.1 1.5 0.1]
 [6.3 2.5 4.9 1.5]]
[[6.8 3.2 5.9 2.3]
 [5.1 2.5 3.  1.1]
 [6.1 3.  4.9 1.8]
 [4.4 2.9 1.4 0.2]
 [6.4 3.1 5.5 1.8]]
```



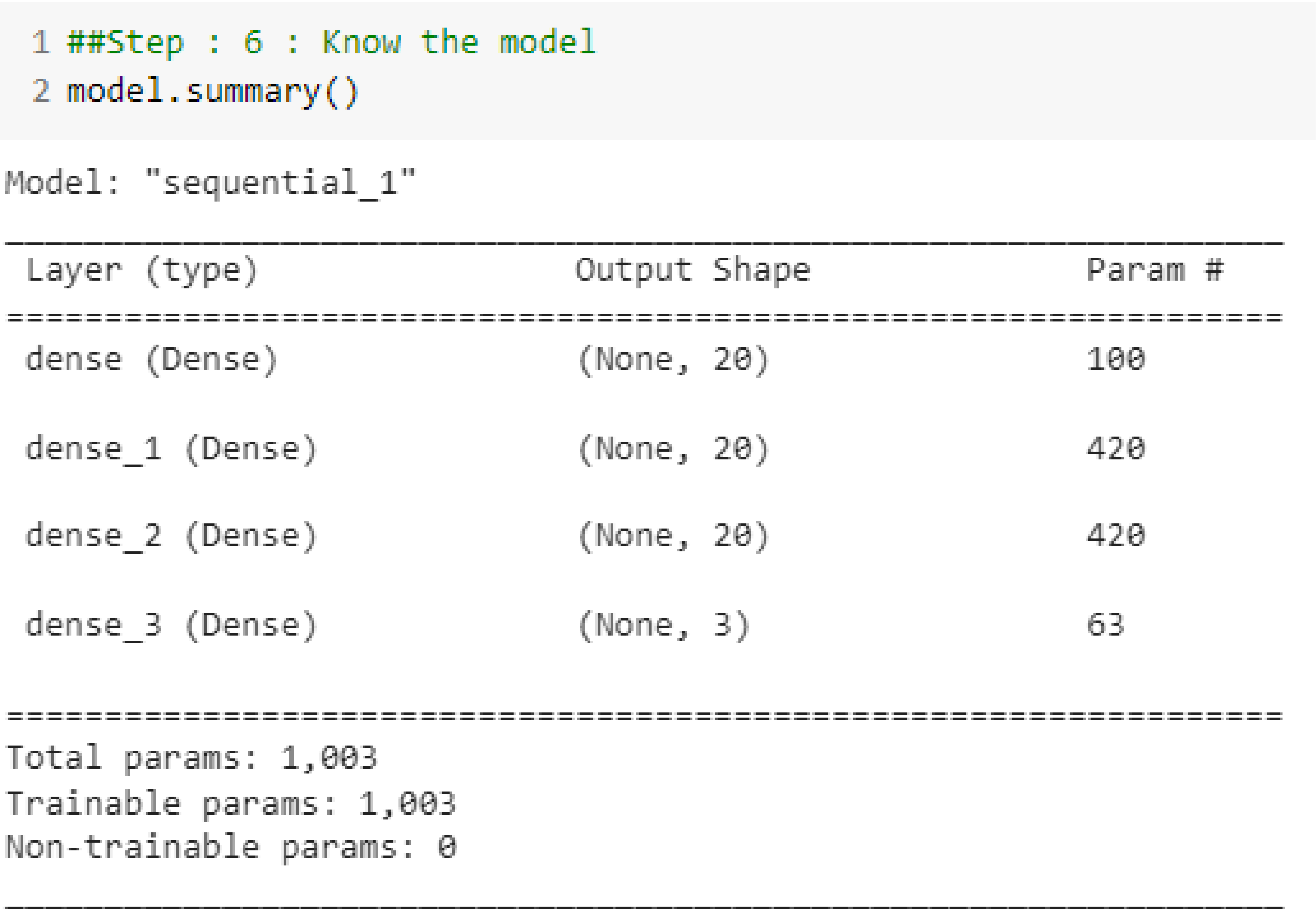
```python
##Step : 4 : Then we normalize the data
from sklearn.preprocessing import StandardScaler
##Normalization is done so that the difference between highest and lowest data point is not too large
import numpy as np
scaler=StandardScaler()
##To find mean and std dev
scaler.fit(x_train)

##Converting data into form where mean of data is 0 and std dev is 1
X_train=scaler.transform(x_train)
X_test=scaler.transform(x_test)
print(np.amax(X_train,axis=0))
print(np.amin(X_train,axis=0))

```

```
[2.50183358 3.01063611 1.80941641 1.73679196]
[-1.88449802 -2.42966562 -1.55002454 -1.42811577]
```



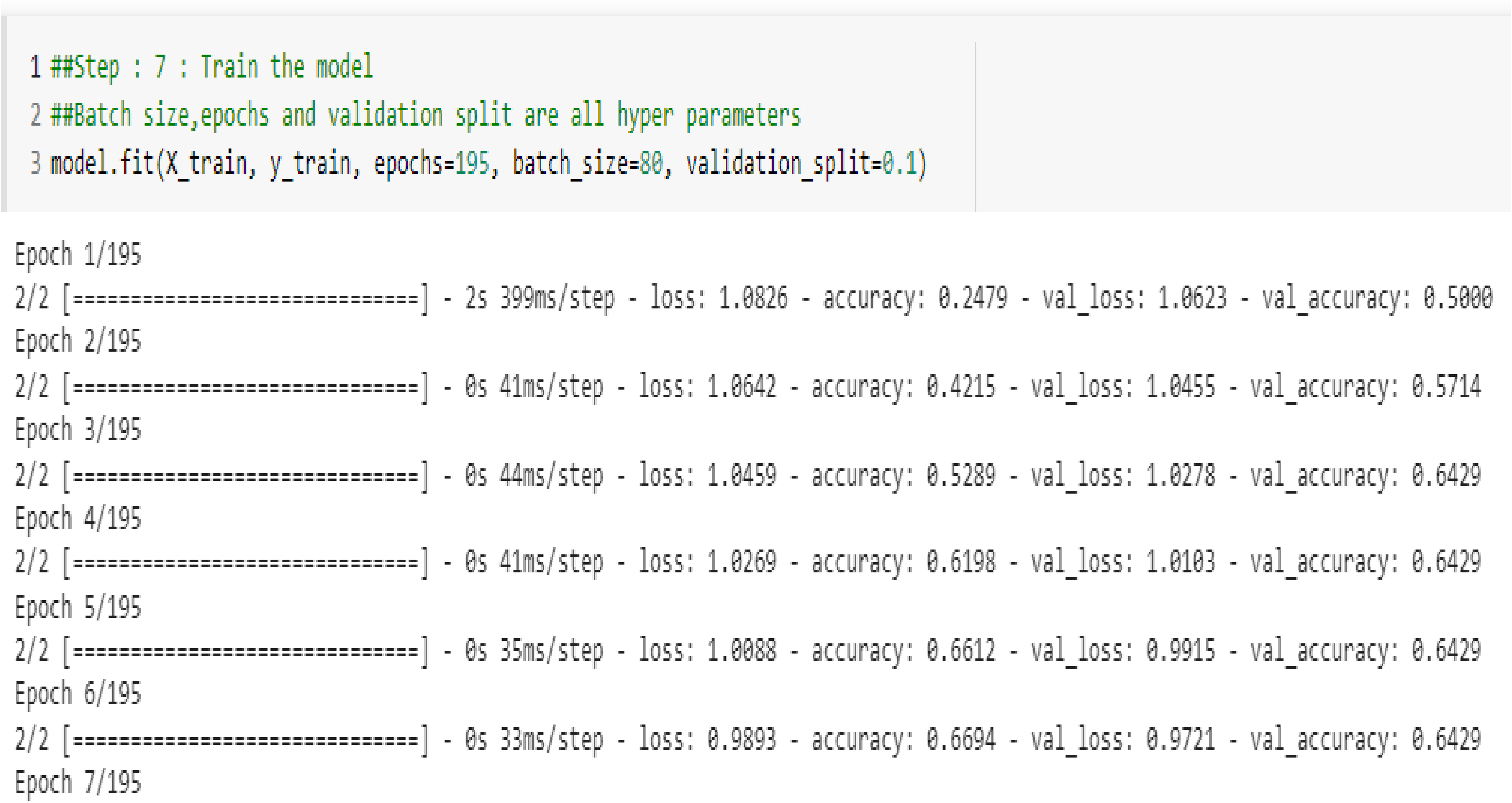
```
##Step : 5 : Now finally we build the model using keras
!pip install tensorflow
import keras
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(20, input_dim=X_train.shape[1], activation='relu'))
##Dropout is used to avoid overfitting
keras.layers.Dropout(0.2)
model.add(Dense(20, activation='relu'))
keras.layers.Dropout(0.2)
model.add(Dense(20, activation='relu'))
keras.layers.Dropout(0.2)
##For classification problems, we usually use softmax as activation function in final layer
model.add(Dense(3, activation='softmax'))
##Compiling the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.9.2)
Requirement already satisfied: flatbuffers<2,>=1.12 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.12)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (4.1.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tensorflow) (57.4.0)
```



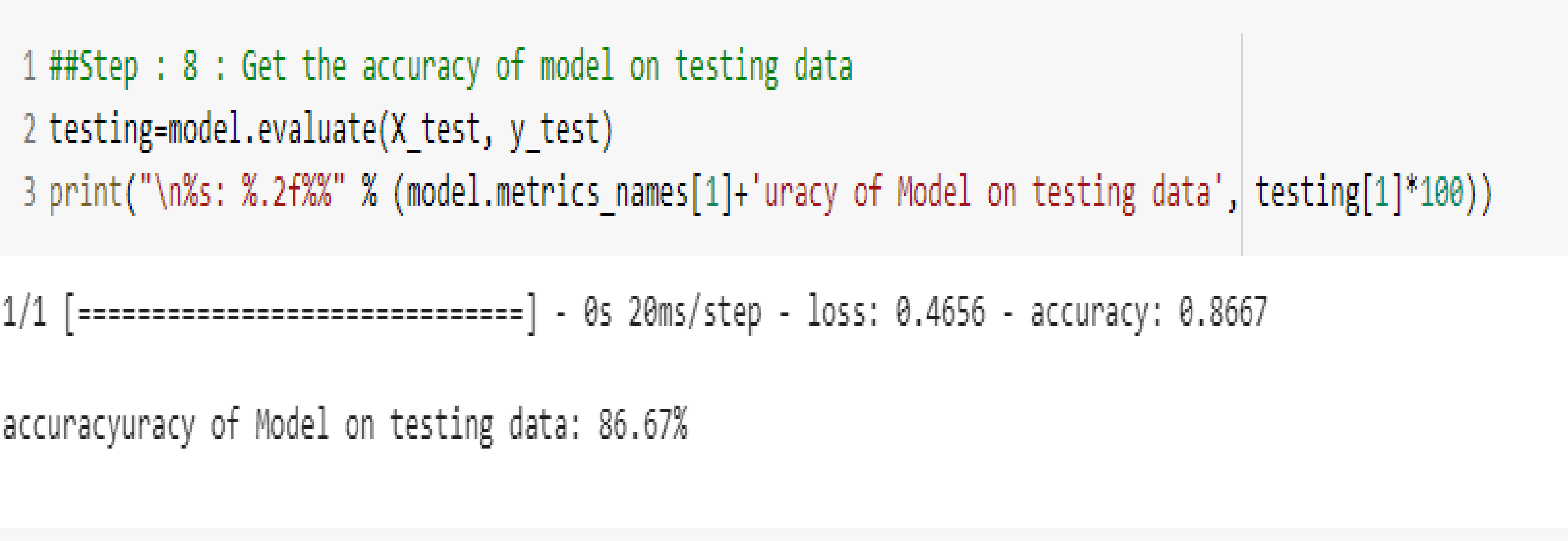
```python
##Step : 6 : Know the model
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                100

 dense_1 (Dense)             (None, 20)                420

 dense_2 (Dense)             (None, 20)                420

 dense_3 (Dense)             (None, 3)                 63

=================================================================
Total params: 1,003
Trainable params: 1,003
Non-trainable params: 0
_________________________________________________________________
```



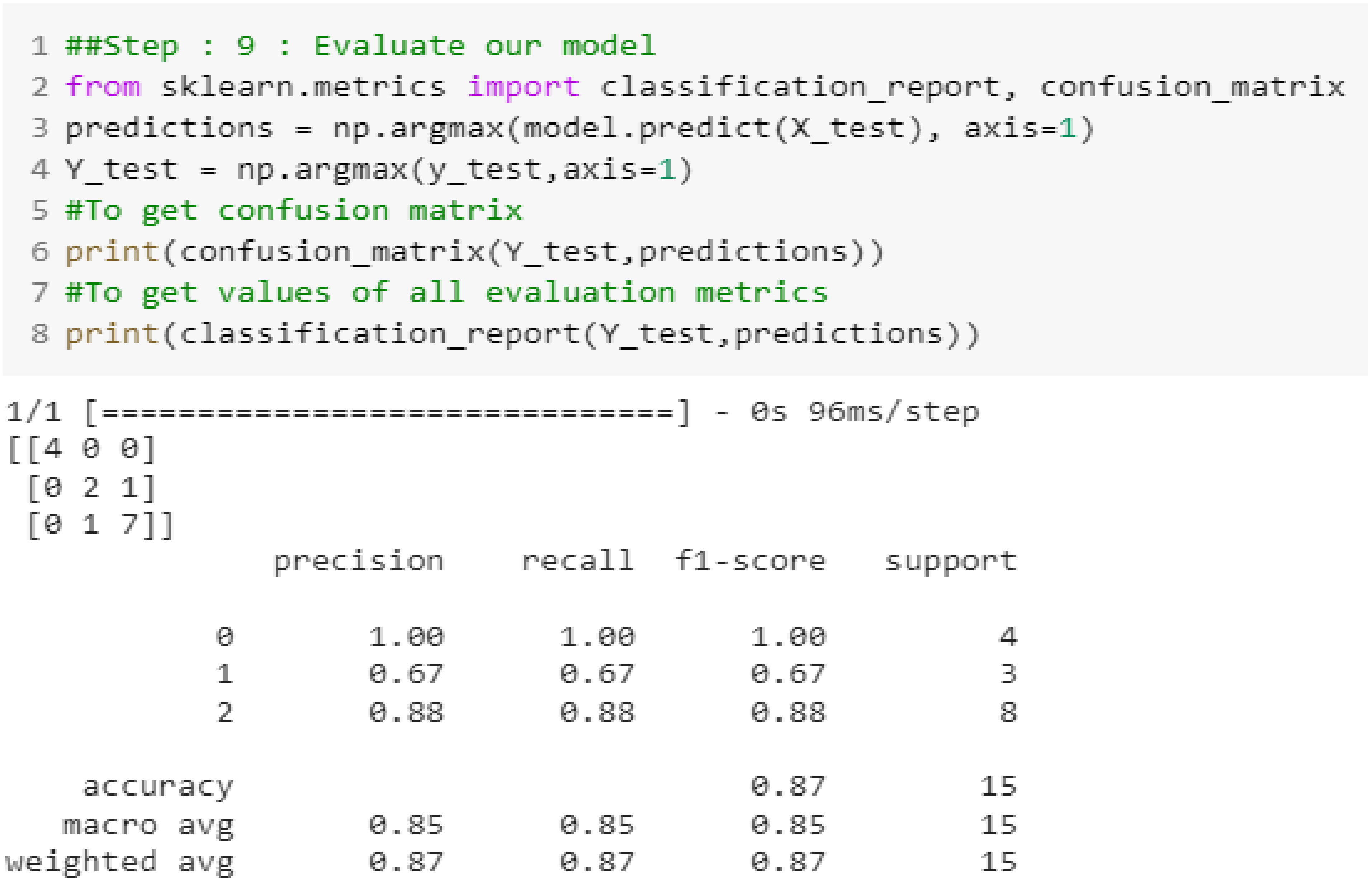
```
##Step : 7 : Train the model
##Batch size,epochs and validation split are all hyper parameters
model.fit(X_train, y_train, epochs=195, batch_size=80, validation_split=0.1)
```

```
Epoch 1/195
2/2 [==============================] - 2s 399ms/step - loss: 1.0826 - accuracy: 0.2479 - val_loss: 1.0623 - val_accuracy: 0.5000
Epoch 2/195
2/2 [==============================] - 0s 41ms/step - loss: 1.0642 - accuracy: 0.4215 - val_loss: 1.0455 - val_accuracy: 0.5714
Epoch 3/195
2/2 [==============================] - 0s 44ms/step - loss: 1.0459 - accuracy: 0.5289 - val_loss: 1.0278 - val_accuracy: 0.6429
Epoch 4/195
2/2 [==============================] - 0s 41ms/step - loss: 1.0269 - accuracy: 0.6198 - val_loss: 1.0103 - val_accuracy: 0.6429
Epoch 5/195
2/2 [==============================] - 0s 35ms/step - loss: 1.0088 - accuracy: 0.6612 - val_loss: 0.9915 - val_accuracy: 0.6429
Epoch 6/195
2/2 [==============================] - 0s 33ms/step - loss: 0.9893 - accuracy: 0.6694 - val_loss: 0.9721 - val_accuracy: 0.6429
Epoch 7/195
```



```
##Step : 8 : Get the accuracy of model on testing data
testing=model.evaluate(X_test, y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1]+'uracy of Model on testing data', testing[1]*100))
```

```
1/1 [==============================] - 0s 20ms/step - loss: 0.4656 - accuracy: 0.8667

accuracyuracy of Model on testing data: 86.67%
```



```python
##Step : 9 : Evaluate our model
from sklearn.metrics import classification_report, confusion_matrix
predictions = np.argmax(model.predict(X_test), axis=1)
Y_test = np.argmax(y_test,axis=1)
#To get confusion matrix
print(confusion_matrix(Y_test,predictions))
#To get values of all evaluation metrics
print(classification_report(Y_test,predictions))
```

```
1/1 [==============================] - 0s 96ms/step
[[4 0 0]
 [0 2 1]
 [0 1 7]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       0.67      0.67      0.67         3
           2       0.88      0.88      0.88         8

    accuracy                           0.87        15
   macro avg       0.85      0.85      0.85        15
weighted avg       0.87      0.87      0.87        15
```