Sane's STATS Academy of Statistics

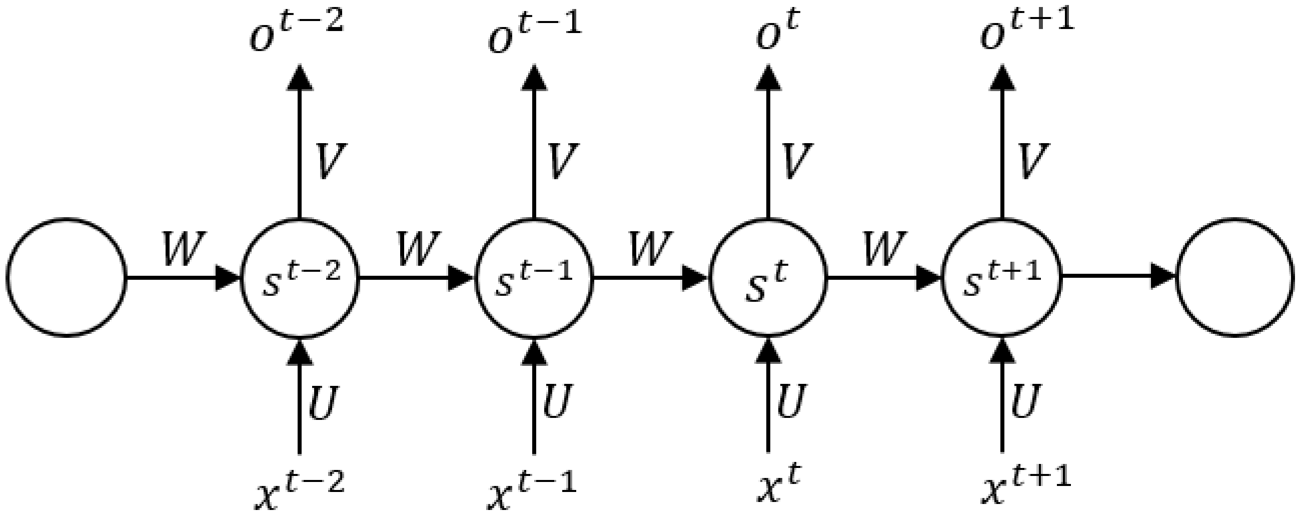
Recurrent Neural Network

Applications(Some of) of RNN

- Speech Recognition
- Language Translation
- Image Recognition and its characterization
- Image Caption Generation
- Time Series Forecasting

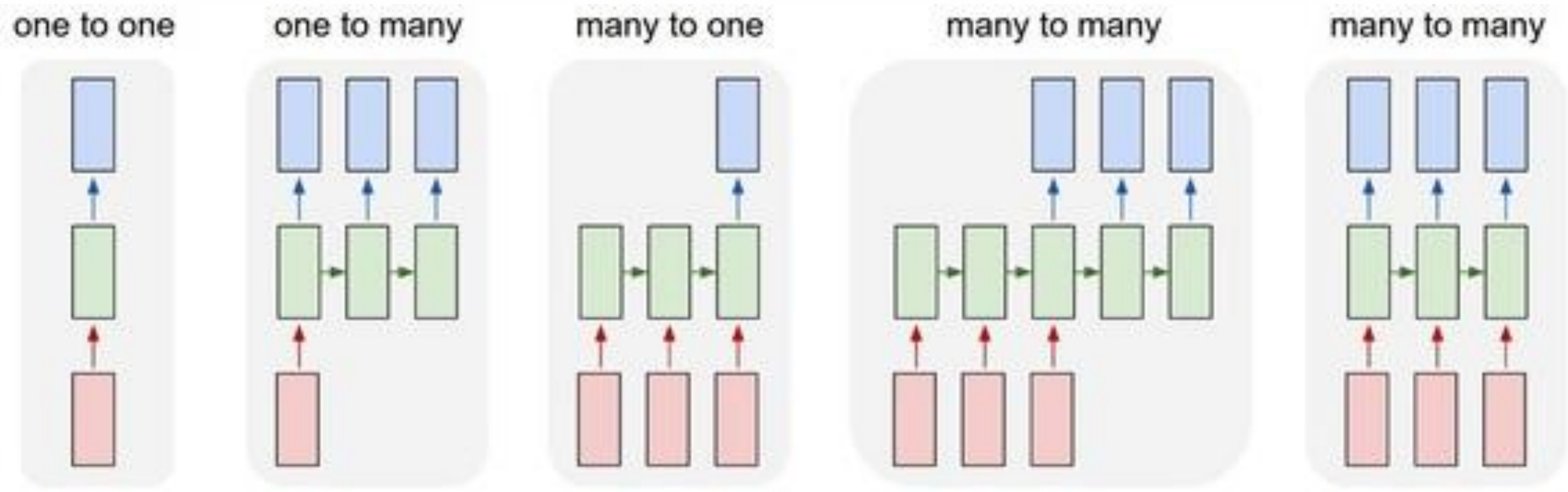
What are RNNs?

- A series of feed forward neural networks in which the hidden nodes are connected in series
- RNN has multiple series predictions unlike the CNN



Types of RNNs

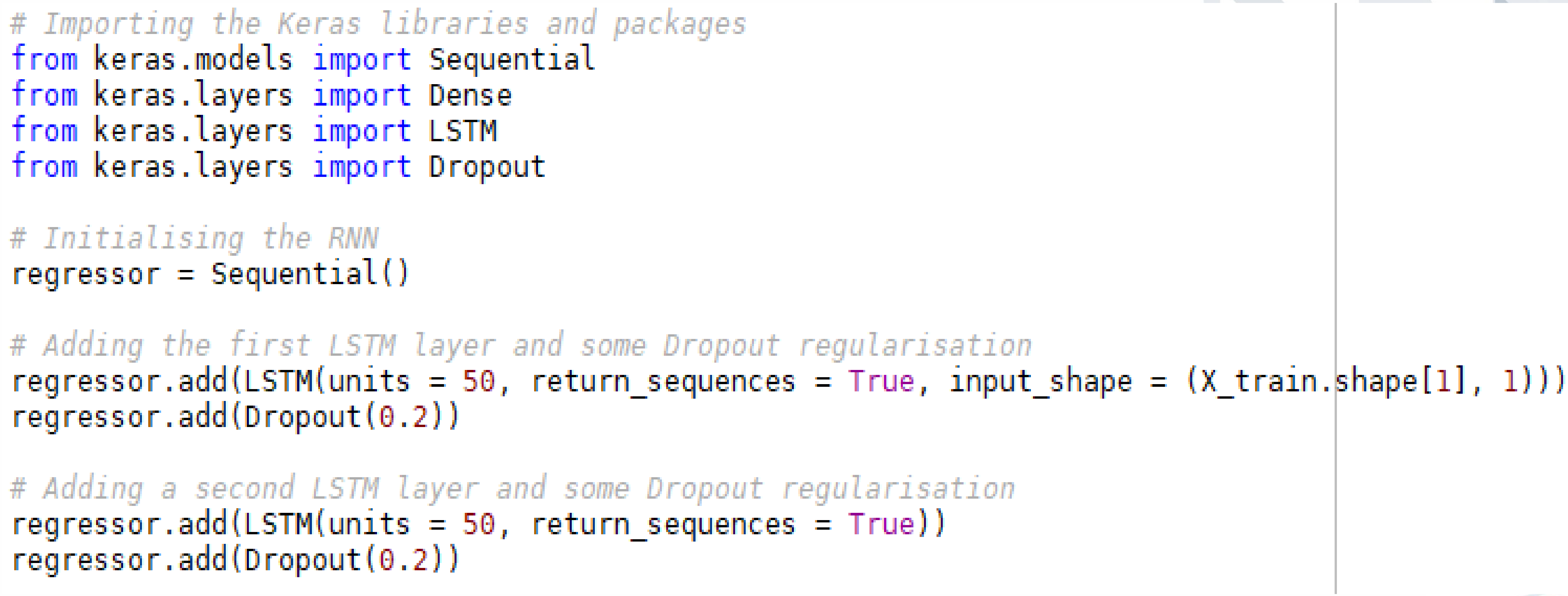
- One to many: Music Generation, Image Caption Generation
- Many to One: Sentiment classification, prediction of the next word
- Many to Many: Language Translation



Improvements to RNN

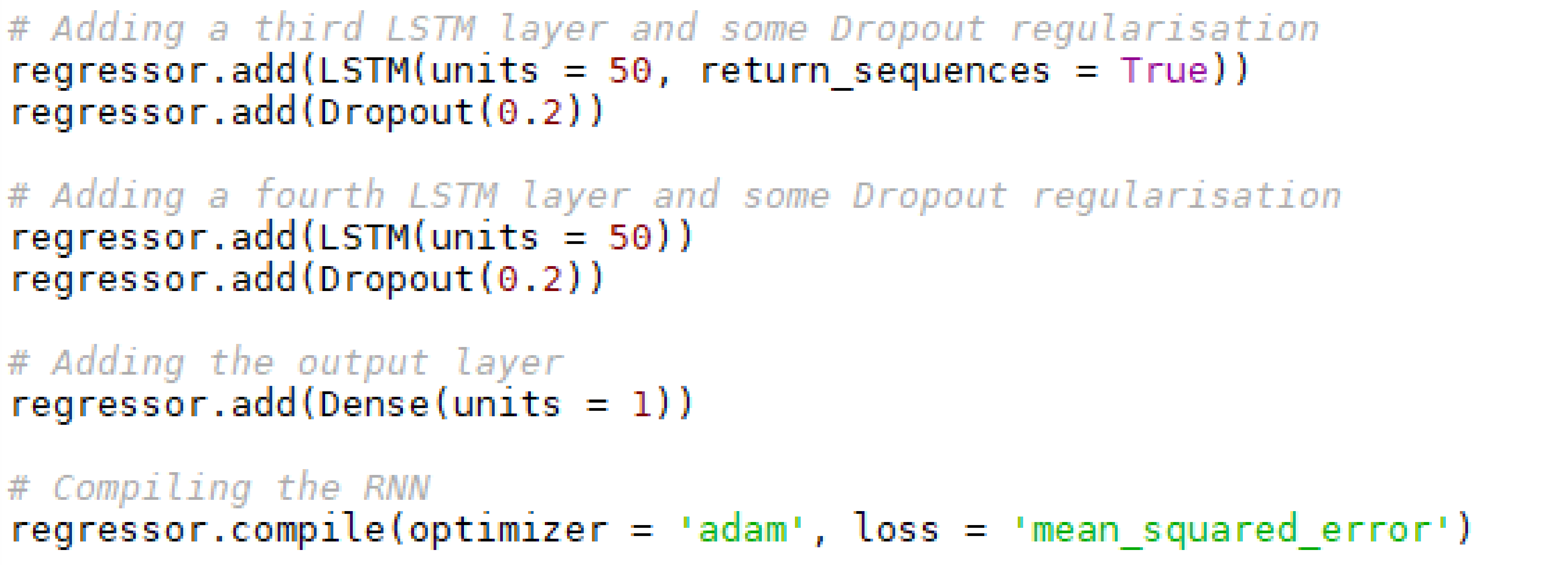
- RNNs have a problem called vanishing gradient descent
- Hence there two improvements over it
  - Gated Recurrent Unit (GRU)
  - Long Short Term Memory (LSTM) (https://colah.github.io/posts/2015-08-Understanding-LSTMs/ and https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714)
- Among these more popular is LSTM more often used in time series forecasting also

LSTM Example: Time Series



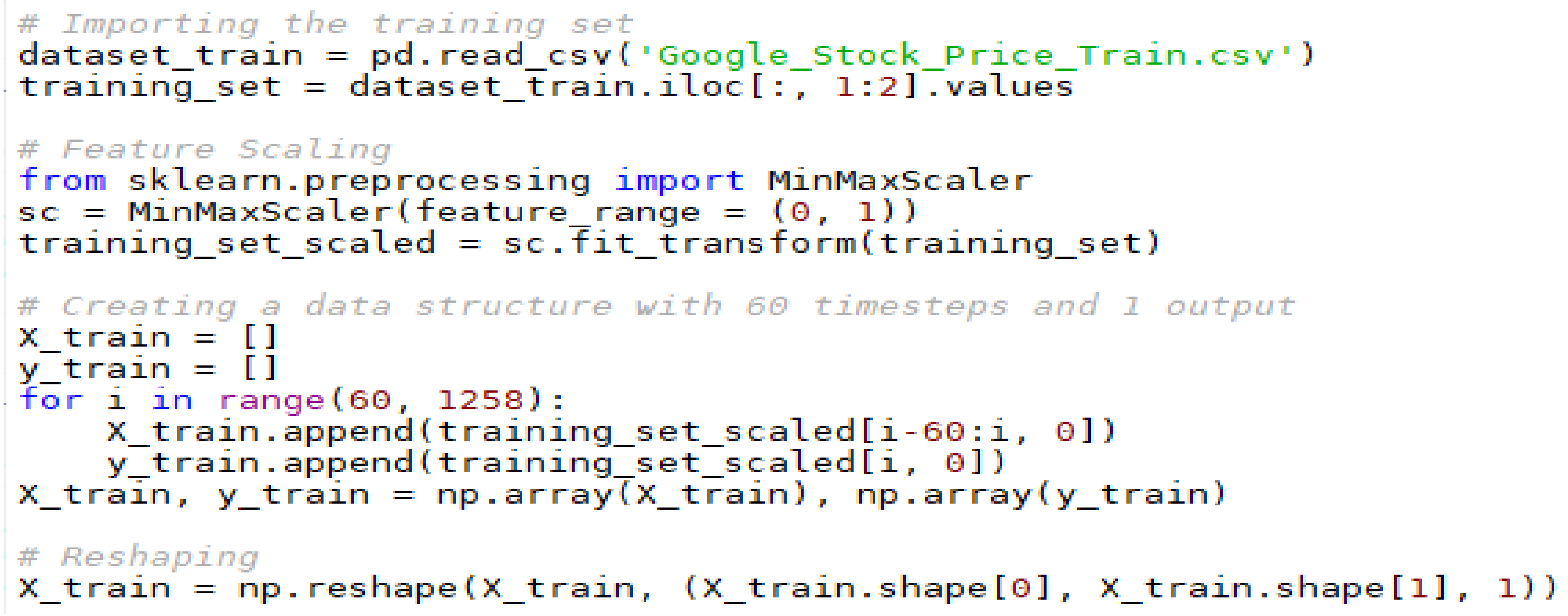
```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

# Initialising the RNN
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
```

Adding the layers to RNN

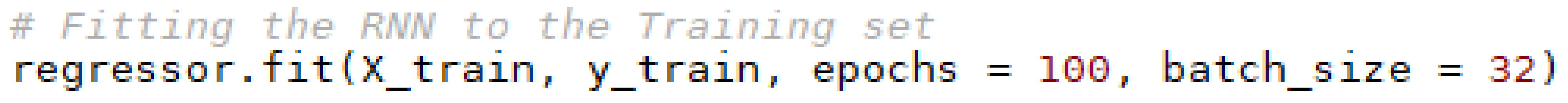


```
# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

Training the RNN

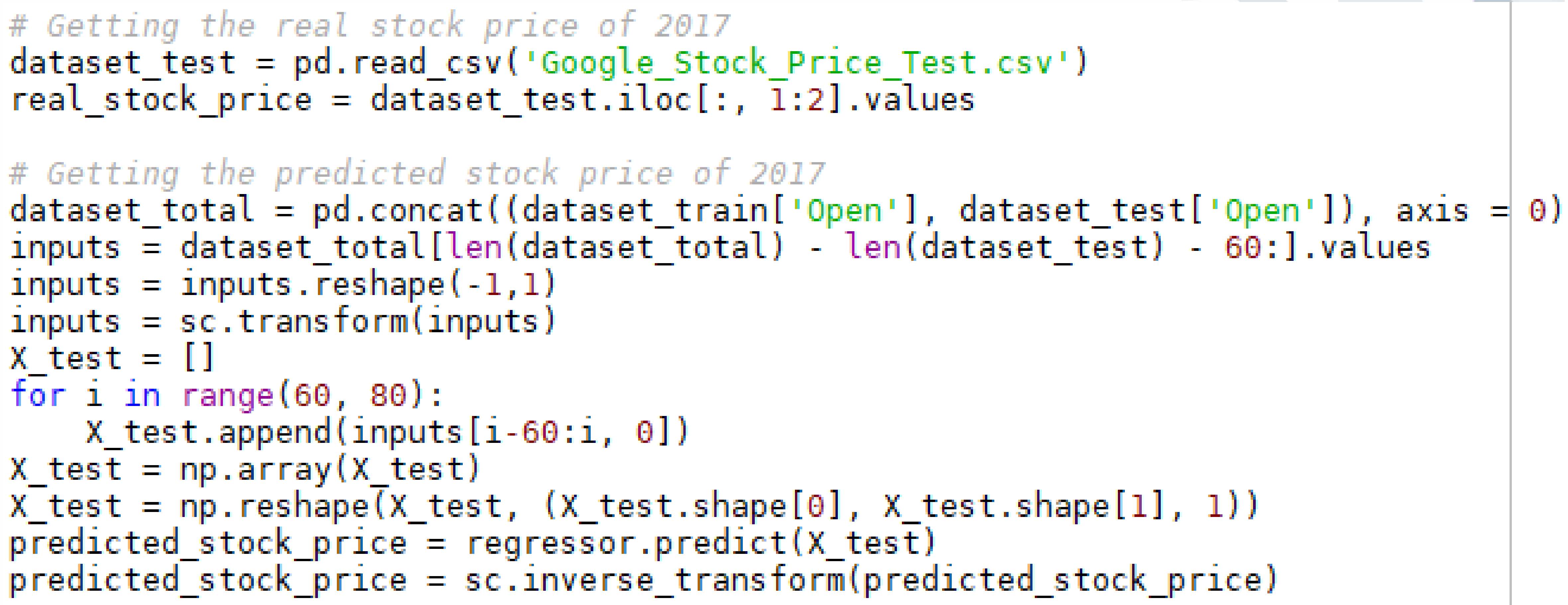


```python
# Importing the training set
dataset_train = pd.read_csv('Google_Stock_Price_Train.csv')
training_set = dataset_train.iloc[:, 1:2].values

# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)

# Creating a data structure with 60 timesteps and 1 output
X_train = []
y_train = []
for i in range(60, 1258):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

# Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```



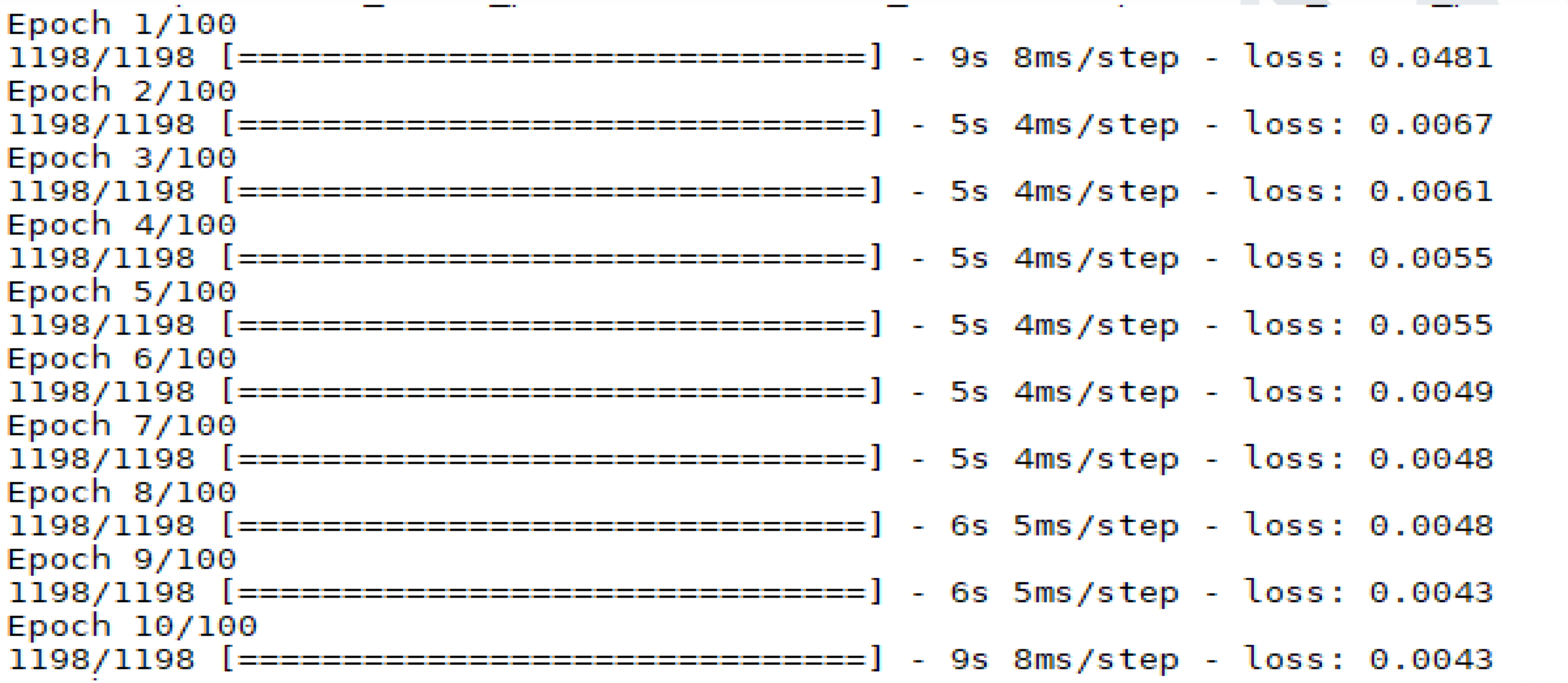
```python
# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

Predicting on the test set



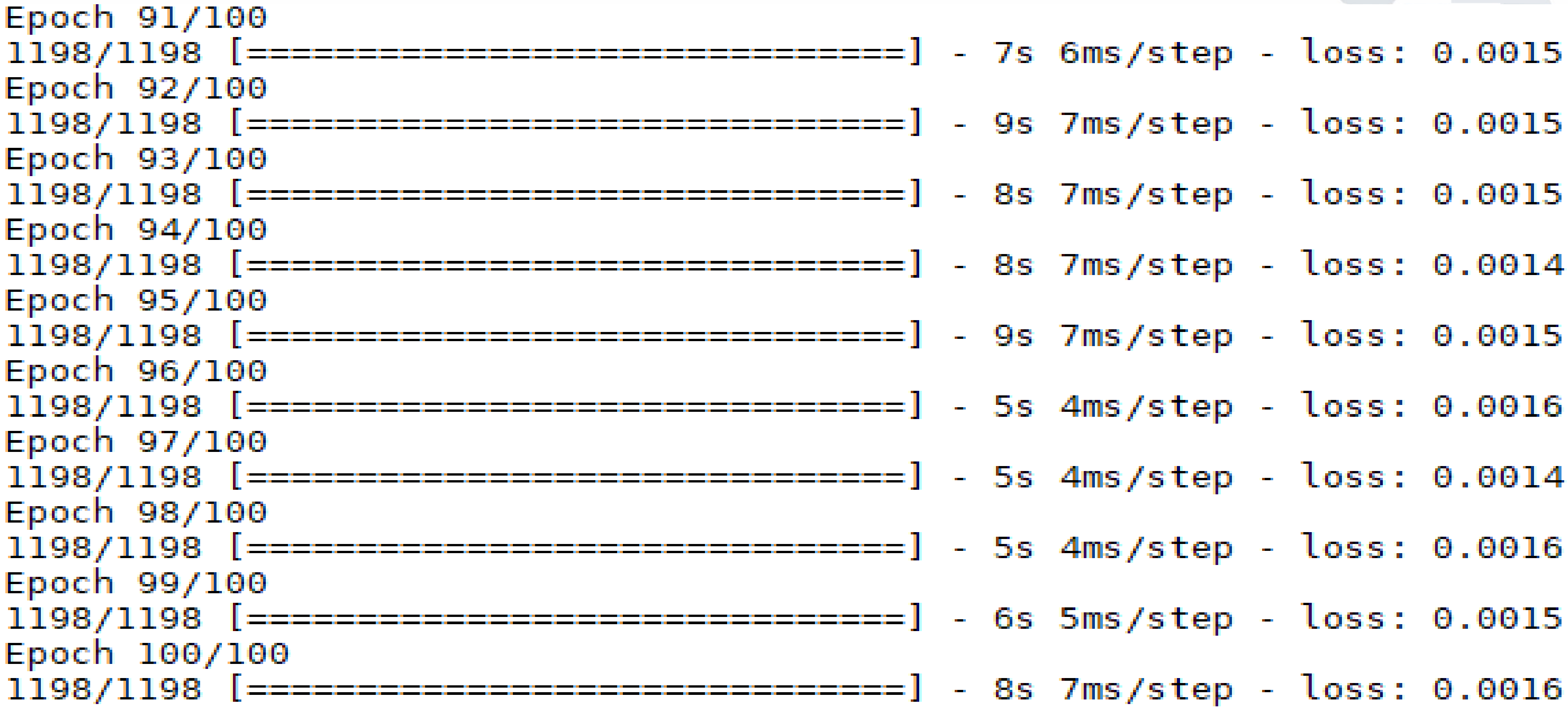
```
# Getting the real stock price of 2017
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
real_stock_price = dataset_test.iloc[:, 1:2].values

# Getting the predicted stock price of 2017
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 80):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Epochs 1 to 10



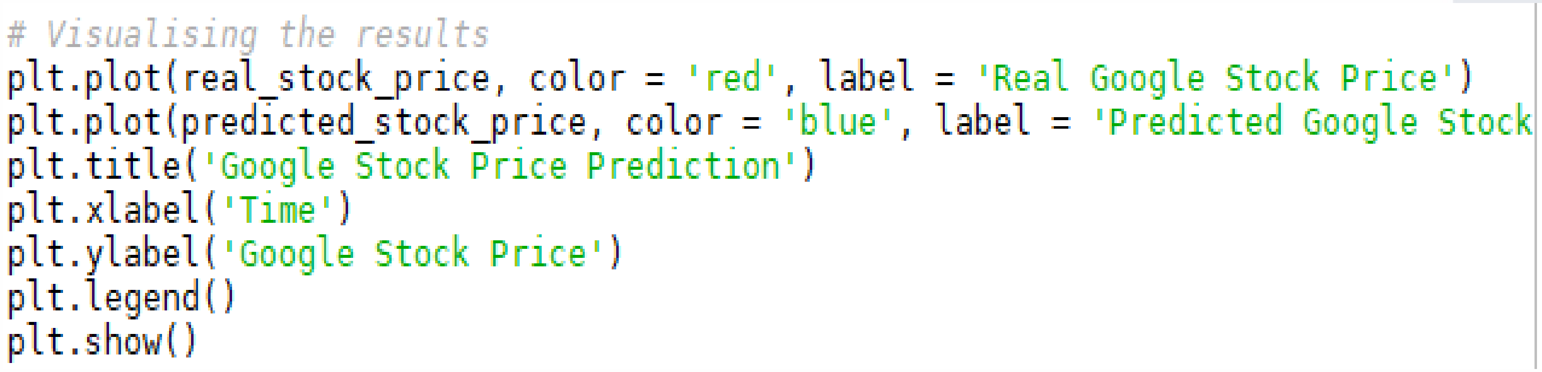
```
Epoch 1/100
1198/1198 [==============================] - 9s 8ms/step - loss: 0.0481
Epoch 2/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0067
Epoch 3/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0061
Epoch 4/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0055
Epoch 5/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0055
Epoch 6/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0049
Epoch 7/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0048
Epoch 8/100
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0048
Epoch 9/100
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0043
Epoch 10/100
1198/1198 [==============================] - 9s 8ms/step - loss: 0.0043
```

Epochs 90 to 100

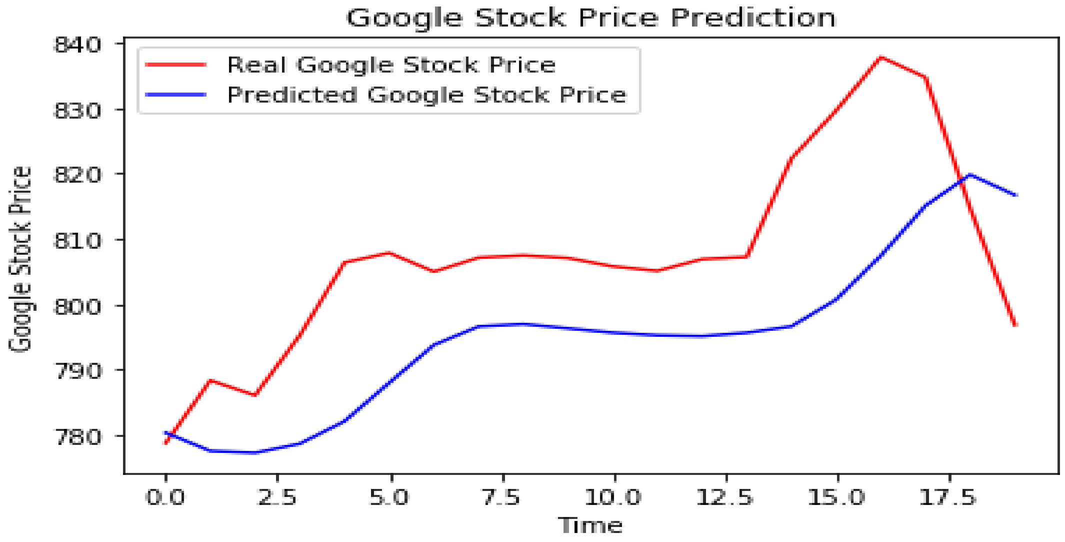


```
Epoch 91/100
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0015
Epoch 92/100
1198/1198 [==============================] - 9s 7ms/step - loss: 0.0015
Epoch 93/100
1198/1198 [==============================] - 8s 7ms/step - loss: 0.0015
Epoch 94/100
1198/1198 [==============================] - 8s 7ms/step - loss: 0.0014
Epoch 95/100
1198/1198 [==============================] - 9s 7ms/step - loss: 0.0015
Epoch 96/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0016
Epoch 97/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0014
Epoch 98/100
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0016
Epoch 99/100
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0015
Epoch 100/100
1198/1198 [==============================] - 8s 7ms/step - loss: 0.0016
```

Visualizing the Result



```
# Visualising the results
plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```


Google Stock Price Prediction
Real Google Stock Price
Predicted Google Stock Price
Google Stock Price
Time