**LSTM Model Building and Training**

1. **Building the LSTM Model**:
   * **Sequential Model**: The Sequential model from Keras is a linear stack of layers. It's used here to create the LSTM model.
   * **Adding LSTM Layers**:
     + The first LSTM layer has 50 units (neurons). The return\_sequences=True parameter is set, which means this layer outputs a sequence, providing a three-dimensional output required for stacked LSTM layers.
     + The second LSTM layer also has 50 units but with return\_sequences=False, indicating that this layer outputs a two-dimensional array (batch size, units), suitable for the Dense layer that follows.
   * **Adding Dense Layers**:
     + A Dense layer with 25 units is added. It's a fully connected layer.
     + Another Dense layer with 1 unit is added as the output layer, which gives the final prediction.
2. **Compiling the Model**:
   * The model is compiled with the adam optimizer, a popular choice for deep learning applications.
   * The loss function used is 'mean\_squared\_error', which is common for regression problems like stock price prediction.
3. **Training the Model**:
   * The model is trained using the fit method, with train\_data (input features) and train\_target (target stock prices).
   * batch\_size=64 means the model will use 64 samples at a time to update the model weights.
   * epochs=20 means the entire dataset will be passed through the LSTM network 20 times.

**LSTM in the Context of Stock Prediction**

* **Purpose**: LSTM is particularly suited for predictions based on time series data, like stock prices, because it can capture long-term dependencies and patterns in data sequences.
* **Data Preparation**: Before this step, the stock data is transformed into sequences that the LSTM model can process. Each input sequence contains a set of consecutive stock prices, and the model learns to predict the next price in the sequence.
* **Output**: The model outputs a sequence of predicted stock prices. These predictions are then scaled back to the original price range using the MinMaxScaler's inverse transformation.