Can you explain why there are 4 layers in the NN? from sklearn.preprocessing import MinMaxScaler sc = MinMaxScaler(feature\_range = (0, 1)) training\_set\_scaled = sc.fit\_transform(training\_set) Now, we create a data structure with 60 timesteps and one output as an Array of x\_train and y\_train. 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) Here we have done reshaping of x\_train data. X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)) Now, the following libraries are required for building the RNN model and perform its operations. We have imported the Keras library and its packages. from keras.models import Sequential from keras.layers import Dense from keras.layers import LSTM from keras.layers import Dropout Let’s initialize our RNN. regressor = Sequential() Now, add the first layer of LSTM and some Dropout regularisation regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1))) regressor.add(Dropout(0.2)) Now, add the second layer of LSTM and some Dropout regularisation regressor.add(LSTM(units = 50, return\_sequences = True)) regressor.add(Dropout(0.2)) Now, add the third layer of LSTM and some Dropout regularisation regressor.add(LSTM(units = 50, return\_sequences = True)) regressor.add(Dropout(0.2)) Now, add the fourth layer of LSTM and some Dropout regularisation regressor.add(LSTM(units = 50)) regressor.add(Dropout(0.2)) Let’s add an output layer. regressor.add(Dense(units = 1)) Next, we will compile our RNN model here. regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error') We are using a training dataset to fit the RNN model. regressor.fit(X\_train, y\_train, epochs = 100, batch\_size = 32) Epoch 1/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0419 0s - loss: Epoch 2/100 38/38 [==============================] - 4s 104ms/step - loss: 0.0058 Epoch 3/100 38/38 [==============================] - 4s 99ms/step - loss: 0.0060 Epoch 4/100 38/38 [==============================] - 4s 98ms/step - loss: 0.0051 Epoch 5/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0050 Epoch 6/100 38/38 [==============================] - 4s 99ms/step - loss: 0.0045 Epoch 7/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0047 Epoch 8/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0046 Epoch 9/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0044 Epoch 10/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0046 Epoch 11/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0043 Epoch 12/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0041 Epoch 13/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0047 Epoch 14/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0035 Epoch 15/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0039 Epoch 16/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0038 Epoch 17/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0035 Epoch 18/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0035 Epoch 19/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0034 Epoch 20/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0036 Epoch 21/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0038 Epoch 22/100 38/38 [==============================] - 4s 100ms/step - loss: 0.0034 Epoch 23/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0033 Epoch 24/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0036 Epoch 25/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0035 Epoch 26/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0036 Epoch 27/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0031 Epoch 28/100 38/38 [==============================] - 4s 106ms/step - loss: 0.0032 Epoch 29/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0030 Epoch 30/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0030 Epoch 31/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0031 Epoch 32/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0030 0s - lo Epoch 33/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0028 Epoch 34/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0030 Epoch 35/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0025 Epoch 36/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0028 Epoch 37/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0032 Epoch 38/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0028 Epoch 39/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0031 Epoch 40/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0026 Epoch 41/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0026 Epoch 42/100 38/38 [==============================] - 4s 106ms/step - loss: 0.0027 Epoch 43/100 38/38 [==============================] - 4s 105ms/step - loss: 0.0027 Epoch 44/100 38/38 [==============================] - 4s 104ms/step - loss: 0.0023 Epoch 45/100 38/38 [==============================] - 4s 106ms/step - loss: 0.0023 Epoch 46/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0024 Epoch 47/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0024 0s - loss: 0.002 Epoch 48/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0025 Epoch 49/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0025 0s - lo Epoch 50/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0023 Epoch 51/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0023 Epoch 52/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0024 Epoch 53/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0022 Epoch 54/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0023 Epoch 55/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0022 Epoch 56/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0025 0s - loss Epoch 57/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0023 Epoch 58/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0022 Epoch 59/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0022 Epoch 60/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0021 Epoch 61/100 38/38 [==============================] - 4s 109ms/step - loss: 0.0021 Epoch 62/100 38/38 [==============================] - 4s 105ms/step - loss: 0.0020 Epoch 63/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0020 Epoch 64/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0022 0s - loss: 0.0 - ETA: 0s - loss: 0.002 Epoch 65/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0024 Epoch 66/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0021 Epoch 67/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0020 Epoch 68/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0019 Epoch 69/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0020 Epoch 70/100 38/38 [==============================] - 4s 106ms/step - loss: 0.0022 Epoch 71/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0020 Epoch 72/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0018 Epoch 73/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0020 Epoch 74/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0016 Epoch 75/100 38/38 [==============================] - 4s 104ms/step - loss: 0.0018 Epoch 76/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0018 Epoch 77/100 38/38 [==============================] - 4s 106ms/step - loss: 0.0019 Epoch 78/100 38/38 [==============================] - 4s 105ms/step - loss: 0.0017 Epoch 79/100 38/38 [==============================] - 4s 104ms/step - loss: 0.0019 Epoch 80/100 38/38 [==============================] - 4s 111ms/step - loss: 0.0018 Epoch 81/100 38/38 [==============================] - 5s 123ms/step - loss: 0.0017 Epoch 82/100 38/38 [==============================] - 4s 104ms/step - loss: 0.0017 Epoch 83/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0017 Epoch 84/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0015 Epoch 85/100 38/38 [==============================] - 4s 104ms/step - loss: 0.0015 Epoch 86/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0015 0s - los Epoch 87/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0018 Epoch 88/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0015 Epoch 89/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0015 1s Epoch 90/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0017 Epoch 91/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0017 Epoch 92/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0017 Epoch 93/100 38/38 [==============================] - 4s 103ms/step - loss: 0.0014 Epoch 94/100 38/38 [==============================] - 4s 104ms/step - loss: 0.0017 Epoch 95/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0016 Epoch 96/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0013 Epoch 97/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0014 Epoch 98/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0015 Epoch 99/100 38/38 [==============================] - 4s 102ms/step - loss: 0.0014 Epoch 100/100 38/38 [==============================] - 4s 101ms/step - loss: 0.0014 Now our next part is to predict stock prices and visualize their results. Here we have used the real stock price of 2017 with data of google\_stock\_price.csv. 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) The final step is to visualize our data results using the matplotlib library. 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()

The neural network (NN) in the code snippet you provided has four layers of Long Short-Term Memory (LSTM) cells, which are a type of recurrent neural network (RNN) architecture. Each LSTM layer is followed by a Dropout layer, which helps to regularize the model by randomly setting a fraction of input units to 0 at each update during training, thus preventing overfitting.

The reason for having four LSTM layers in this particular code snippet may be to capture complex temporal dependencies in the input data. Each LSTM layer learns to extract relevant features from the input data at different levels of abstraction. The first LSTM layer takes the input sequence and returns a sequence output, which is then passed to the second LSTM layer, and so on. The last LSTM layer produces the final output sequence, which is then passed to a Dense layer with one output unit to make a regression prediction.

The use of Dropout layers after each LSTM layer helps to prevent overfitting by randomly dropping out some connections during training, which can improve the generalization ability of the model.

It's important to note that the choice of the number of LSTM layers and other hyperparameters in a neural network model depends on the specific problem and dataset being used. It may require experimentation and tuning to determine the optimal architecture for a given task.

does one layer take care of simulating dependencies?

Yes, each layer in a neural network, including LSTM layers, is responsible for capturing and simulating different types of dependencies in the input data. In the case of LSTM layers, they are designed specifically to capture and model sequential dependencies in time series data or sequences.

In the provided code snippet, multiple LSTM layers are stacked on top of each other to create a deep recurrent neural network (RNN) model. Each LSTM layer learns to capture different levels of dependencies in the input data. The first LSTM layer takes input sequences of 60 timesteps and learns to capture short-term dependencies within those sequences. The subsequent LSTM layers then build upon the representations learned by the previous layers and capture longer-term dependencies in the input data.

By stacking multiple LSTM layers, the model can learn to capture complex temporal patterns and dependencies in the time series data, which can be useful for making accurate predictions in time series forecasting tasks. The Dropout regularization is also applied after each LSTM layer to prevent overfitting and improve the generalization capability of the model.

What do the subsequent dependencies take care of?

The subsequent LSTM layers in a deep LSTM-based neural network model are designed to capture longer-term dependencies in the input data. As information flows through the layers, each subsequent LSTM layer can learn to capture more complex and abstract representations of the input data, building upon the representations learned by the previous layers.

In time series data, dependencies can exist at various temporal scales. Some patterns may be short-term and captured by the first few LSTM layers, while others may be longer-term and require deeper LSTM layers to capture effectively. By stacking multiple LSTM layers, the model can learn to capture these longer-term dependencies, which may involve more complex interactions between different time steps in the input sequence.

The subsequent LSTM layers can learn to model more abstract and higher-level features, as the information from the previous layers is passed on to the next layers. This can help the model capture complex patterns in the time series data, such as trends, seasonality, and non-linear relationships between variables, which are important for accurate predictions in time series forecasting tasks.

Overall, the subsequent LSTM layers in a deep LSTM-based model take care of capturing longer-term dependencies in the input data and learning more abstract representations, which can help improve the model's ability to make accurate predictions in complex time series data.

Can you give an example for difference between short term and long term dependencies

Sure! Let's consider an example of a stock price prediction task using time series data. In this case, a short-term dependency might be the stock price at time step t being influenced by the stock prices at time steps t-1 and t-2, as stock prices often exhibit immediate trends or reactions to recent market events. On the other hand, a long-term dependency might be the stock price at time step t being influenced by the stock prices at time steps t-7 and t-14, as stock prices can also exhibit weekly or monthly patterns.

Short-term dependencies are typically captured by the early layers of an LSTM-based model, which have smaller receptive fields and can quickly respond to recent changes in the input data. These layers can learn to capture local patterns and short-term fluctuations in the time series data, such as sudden spikes or drops in stock prices.

Long-term dependencies, on the other hand, require capturing patterns that occur over longer time spans. These dependencies may involve more complex interactions between different time steps in the input sequence and may require higher-level abstractions. Deeper LSTM layers in a deep LSTM-based model can capture these longer-term dependencies by processing the input data over larger receptive fields and learning more abstract representations of the input data. For example, they can learn to capture weekly or monthly trends in stock prices, which may be influenced by factors such as economic indicators or seasonal patterns.

By combining the information captured by both short-term and long-term dependencies, a deep LSTM-based model can make more accurate predictions in time series data, capturing both immediate fluctuations and longer-term trends in the input data.