Detailed Explanation of LSTM Stock Price Prediction Code

LSTM Stock Price Prediction: Complete Explanation

1. Data Loading and Reset Index

The data is loaded from a CSV file containing historical stock prices. The `reset_index()` method is used to reset the DataFrame index to the default integer index. This makes the data more accessible for row-wise operations, especially when working with time-series data. It moves the existing index (e.g., date) into a separate column.

```
df1 = df.reset_index()['close']
```

2. Data Visualization

A simple line plot is created to visualize the stock's closing price trend over time using Matplotlib. This gives an overview of the data patterns, such as trends and seasonality.

```
plt.plot(df1)
```

3. Data Preprocessing with MinMaxScaler

LSTMs are sensitive to the scale of the data, so the MinMaxScaler is used to scale all values to a range of 0 to 1. This improves model convergence during training and ensures consistent input to the network. `fit_transform` scales the entire dataset.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
df1 = scaler.fit_transform(np.array(df1).reshape(-1,1))
```

4. Train-Test Split

The data is split into training and testing sets. Typically, 70% of the data is used for training, and the remaining 30% is reserved for testing. This ensures the model can be validated on unseen data.

```
train_size = int(len(df1)*0.7)
test_size = len(df1) - train_size
train_data, test_data = df1[0:train_size, :], df1[train_size:len(df1), :]
```

5. Create Dataset Function

The `create_dataset` function generates input-output pairs from the time-series data for supervised learning. Each input is a sequence of length `time_step` (e.g., 100 steps), and the output is the next value in the sequence.

```
def create_dataset(dataset, time_step=1):
    data_X, data_y = [], []
    for i in range(len(dataset) - time_step - 1):
        a = dataset[i:(i + time_step), 0]
        data_X.append(a)
        data_y.append(dataset[i + time_step, 0])
    return np.array(data_X), np.array(data_y)
```

6. Reshaping Input for LSTM

The input data is reshaped to match the LSTM's expected input format: [samples, time steps, features]. Here, `samples` refers to the number of sequences, `time steps` is the sequence length, and `features` represents the number of features at each time step (1 in this case).

```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

7. Model Building and Training

The Sequential model consists of three stacked LSTM layers and one Dense output layer. `return_sequences=True` is used in the first two LSTM layers to ensure they return the entire sequence for the next layer. The model is compiled with the Adam optimizer, which is adaptive and efficient for training deep learning models. Validation data helps monitor the model's performance on unseen data during training.

```
model = Sequential([
    Input(shape=(100, 1)),
    LSTM(50, return_sequences=True),
    LSTM(50, return_sequences=True),
    LSTM(50),
    Dense(1)
])
model.compile(loss='mean_squared_error', optimizer='adam')
```

8. Validation Data

Validation data is used to monitor the model's performance on unseen data during training. This helps to detect overfitting early. The `validation_data` argument in `model.fit` specifies the test set as the validation dataset.

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, batch_size=64, verbose=1)
```

9. Inverse Transform

The predictions made by the model are in the scaled range (0 to 1) because the data was preprocessed using MinMaxScaler. To interpret the predictions in the original scale, `scaler.inverse_transform` is used. This reverts the scaled values back to their original range.

```
y_train_predict = scaler.inverse_transform(y_train_predict)
y_test_predict = scaler.inverse_transform(y_test_predict)
```

10. Shift Prediction and Look Back

Predictions for both training and test sets are shifted to align with the original data for proper visualization. The `look_back` parameter determines the number of past time steps the model uses for making predictions. This shifting ensures that the predicted values are plotted at the correct time points.

```
look_back = 100
trainPredictPlot = np.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(y_train_predict) + look_back, :] = y_train_predict

testPredictPlot = np.empty_like(df1)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(y_train_predict) + (look_back * 2) + 1:len(df1) - 1, :] = y_test_predict
```

11. Prediction for Next 10 Days (While Loop)

The while loop predicts the next 30 values by iteratively using the last 100 time steps. Each iteration:

1. Extracts the last `n_steps` elements to create an input sequence.

2. Makes a prediction for the next time step using the LSTM model.

3. Updates the `temp_input` list to include the new prediction and exclude the oldest value.

4. Adds the prediction to `lst_output`, which stores all the predicted values.

```
lst_output = []
n_steps = 100
i = 0
while i < 30:
    if len(temp_input) > 100:
        x_input = np.array(temp_input[-100:]).reshape((1, n_steps, 1))
        y_hat = model.predict(x_input, verbose=0)
        temp_input.extend(y_hat[0].tolist())
        temp_input = temp_input[1:]
        lst_output.extend(y_hat.tolist())
        i += 1
    else:
        print("Insufficient data for prediction")
        break
```

12. np.arange for Plotting Future Predictions

`np.arange` generates evenly spaced values within a specified range. It is used here to create x-axis values for plotting the new predictions and extending the plot beyond the original dataset.

```
day_new = np.arange(1, 101)
day_pred = np.arange(101, 131)
```

13. Plotting Predicted and Original Data

The original data, the extended data with predictions, and the future predicted values are plotted together. This provides a comprehensive view of the model's performance and its ability to predict future trends.

```
plt.plot(day_new, scaler.inverse_transform(df1[-100:]))
plt.plot(day_pred, scaler.inverse_transform(lst_output))
df3 = df1.tolist()
df3.extend(lst_output)
plt.plot(scaler.inverse_transform(df3))
plt.show()
```