```
# Import necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns  # For loading the dataset
from sklearn.preprocessing import MinMaxScaler  # For normalizing data
from sklearn.metrics import mean_squared_error, mean_absolute_error
# import tensorflow as tf
# from tensorflow import keras
from keras.src.models.sequential import Sequential
from keras.src.layers.core.dense import Dense
from keras.src.layers.rnn.lstm import LSTM
# from k.models import Sequential  # For creating the LSTM model
# from k.layers import LSTM, Dense  # For adding LSTM and Dense layers
import matplotlib.pyplot as plt  # For plotting the results


# Load the 'tips' dataset from seaborn
data = sns.load_dataset('tips')


# Take the 'total_bill' column as a pseudo-stock price for the demo
prices = data['total_bill'].values.reshape(-1,1)  # Reshape to a 2D array for scaling
```

```
print("Prices : ", prices)


# Normalize the data using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(prices)  # Scale data between 0 and 1


print("Scaled Data : ",scaled_data)




# print("prices : ",type(prices))
# print("Scaled Data : ",type(scaled_data))
# df = pd.DataFrame(prices)
# print(df.head(5))




# Create the training data by taking past 'time_step' values to predict the next one
train_data = []
target_data = []
time_step = 10  # Using past 10 values to predict the next one


# Loop through the scaled data and prepare input-output sequences
```

```
for i in range(time_step, len(scaled_data)):

    train_data.append(scaled_data[i-time_step:i, 0])  # Last 10 days

    target_data.append(scaled_data[i, 0])  # Next day (target)



train_data, target_data = np.array(train_data), np.array(target_data)

train_data = np.reshape(train_data, (train_data.shape[0], train_data.shape[1], 1))  # Reshape for LSTM input



# Build the LSTM model

model = Sequential()



# Add the first LSTM layer with 50 units (neurons) and return sequences to feed into the next LSTM layer

model.add(LSTM(units=50, return_sequences=True, input_shape=(train_data.shape[1], 1)))



# Add another LSTM layer without returning sequences

model.add(LSTM(units=50))



# Add a Dense output layer with 1 unit (for predicting the next value)

model.add(Dense(1))
```

```
# Compile the model using Adam optimizer and mean squared error as the loss function
model.compile(optimizer='adam', loss='mean_squared_error')


# Train the model for 10 epochs with a batch size of 32
model.fit(train_data, target_data, epochs=10, batch_size=32)

# Predict the future values based on training data
predicted_prices = model.predict(train_data)

# Inverse transform the predicted prices to the original scale
predicted_prices = scaler.inverse_transform(predicted_prices)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(target_data, predicted_prices)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(target_data, predicted_prices)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
```

```
# Print the calculated metrics

print(f"Mean Squared Error (MSE): {mse}")

print(f"Mean Absolute Error (MAE): {mae}")

print(f"Root Mean Squared Error (RMSE): {rmse}")



# Plot the actual vs predicted values for visualization

plt.figure(figsize=(10,6))

plt.plot(prices, label='Actual Prices')  # Plot actual values

plt.plot(np.concatenate([np.zeros(time_step), predicted_prices.flatten()]), label='Predicted Prices')  # Plot predicted values

plt.legend()

plt.savefig('Deep_Learning/DL_Plots/LSTM_Plot.jpg')

plt.show()
```

Explain me each and every line of code and while explaining consider me as a freshers who is not aware of Deep Learning

Absolutely, let's break down the code line by line, explaining it for someone new to Deep Learning:

1. Import Libraries (Lines 1-8):

This section imports necessary libraries for data manipulation, model building, and visualization. Here's what each library does:

- numpy (np): Used for numerical computations (arrays, matrices, etc.).
- pandas (pd): Used for data analysis and manipulation (DataFrames, Series).
- seaborn (sns): Used for creating statistical data visualizations.
- sklearn.preprocessing.MinMaxScaler: Used to normalize data between 0 and 1.
- sklearn.metrics: Used to calculate performance metrics like Mean Squared Error (MSE).
- keras (part of TensorFlow): Used for building deep learning models (LSTM in this case).

- matplotlib.pyplot (plt): Used for creating visualizations like plots.

2. Load Dataset (Line 9):

This line uses seaborn to load a built-in dataset called "tips" which likely contains restaurant bill information.

3. Prepare Data (Lines 10-18):

- We extract the "total_bill" column as a pseudo-stock price (imaginary stock prices for demonstration).
- reshape(-1, 1) converts the data into a 2D array suitable for scaling.
- MinMaxScaler is used to normalize the data between 0 and 1 for better model training.

4. Create Training Data (Lines 20-30):

- We define a time_step variable (set to 10 here) which represents the number of past values used to predict the next one.
- We loop through the scaled data, creating sequences of past time_step values and their corresponding next values (targets for prediction).
- np.array converts the lists into NumPy arrays for easier manipulation.
- reshape reformats the training data into a specific format required by LSTMs (3D with samples, timesteps, and features).

5. Build the LSTM Model (Lines 32-40):

- Sequential creates a sequential model where data flows layer by layer.
- **First LSTM Layer:**
  - units=50 defines the number of neurons (processing units) in the layer (here, 50).
  - return_sequences=True enables the layer to pass its output to the next layer (as a sequence).
  - input_shape=(train_data.shape[1], 1) specifies the expected input shape (number of timesteps and features).
- **Second LSTM Layer:**
  - units=50 defines another layer with 50 neurons.
  - This layer doesn't return sequences as it's the final layer before prediction.
- **Output Layer:**
  - Dense(1) creates a dense layer with 1 neuron for single-value prediction (next price).

6. Compile the Model (Lines 42-43):

- model.compile configures the model for training.
- optimizer='adam' sets the optimization algorithm used for training (Adam is a popular choice).
- loss='mean_squared_error' defines the loss function to be minimized during training (measures prediction error).

7. Train the Model (Line 45):

- model.fit trains the model on the prepared training data.

- epochs=10 specifies the number of training cycles (iterations).
- batch_size=32 defines the number of samples processed together in each training step.

8. Predict Future Values (Line 47):

- model.predict uses the trained model to predict future values based on the training data.

9. Reverse Normalization (Line 48):

- We transform the predicted values back to the original scale using the scaler we created earlier.

10. Calculate Performance Metrics (Lines 50-53):

- These lines calculate various metrics to evaluate the model's performance:
  - mean_squared_error (MSE): Measures the average squared difference between predicted and actual values.
  - mean_absolute_error (MAE): Measures the average absolute difference between predicted and actual values.
  - root_mean_squared_error (RMSE): Square root of MSE (often easier to interpret).

11. Visualization (Lines 55-62):

- This section creates a plot to