StockBot Manuel

1. Import Libraries and Set Up Environment

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
import datetime as dt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from pandas_datareader import data as pdr
import yfinance as yfin
yfin.pdr_override()
```

In this section:

- The required libraries are imported, including numerical computing with `numpy`, plotting with `matplotlib`, data manipulation with `pandas`, and machine learning tools from `tensorflow.keras`.
- `pandas_datareader` and `yfinance` are used to fetch financial data.

2. User Input for Ticker and Date Range

```
tckr = input('Enter company ticker: ')
company = tckr

inStart = input('Enter start date for historical data in YYYYMMDD or type
"S" for the standard date setting of 2015,1,1 ')
if (inStart == 'S'):
    start = dt.datetime(2015, 1, 1)
else:
    start = dt.datetime.strptime(inStart, '%Y%m%d')

inEnd = input('Enter end date for historical data in YYYYMMDD or type "S"
for the standard date setting of now ')
if (inEnd == 'S'):
    end = dt.datetime.now()
else:
    end = dt.datetime.strptime(inEnd, '%Y%m%d')
```

In this section:

- The user is prompted to input the company ticker symbol, start date, and end date for historical data.
- If 'S' is entered for the start or end date, the script uses standard date settings.

3. Retrieve Historical Data

```
data = pdr.get_data_yahoo(company, start, end)
```

In this section:

- The `pandas_datareader` library is used to fetch historical stock price data from Yahoo Finance using the specified ticker symbol and date range.

4. User Input for Scaling and Training Parameters

```
YoN = input('Select Y or N if you want scaling range to depend on the default, Close. The other options are the columns in the data chart ')
if(YoN=='Y'):
    choose = input('Enter your choice: ')
else:
    choose = 'Close'

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(data[choose].values.reshape(-1,1))

prediction_days = int(input('Enter the desired number of training days. 60 days recommended '))
```

In this section:

- The user is prompted to choose whether to customize the scaling range based on a specific column or use the default ('Close').
- The `MinMaxScaler` is applied to scale the chosen column.
- The user is prompted to input the desired number of training days.

5. Prepare Training Data

```
x_train = []
y_train = []

for x in range(prediction_days, len(scaled_data)):
    x_train.append(scaled_data[x-prediction_days:x, 0])
    y_train.append(scaled_data[x, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

In this section:

- Lists `x_train` and `y_train` are populated to create training data sequences for the LSTM model.
- The data is reshaped into the required format for the LSTM model.

6. Build LSTM Model with User Input

```
model = Sequential()

YoN = input('Do you want to change standard amounts of input layers and settings. "Y" or "N" ')
if (YoN=='Y'):
    # User input for customizing the LSTM layers and dropout rates
else:
    # Default settings for LSTM layers and dropout rates
```

In this section:

- The user is prompted to decide whether to customize the LSTM layers and dropout rates.
- If 'Y' is selected, the user is prompted for the number of layers, units, and dropout rates for each layer.

7. Compile and Train Model

```
optmzr = input('Enter adam or SGD optimizer ')
model.compile(optimizer= optmzr , loss='mean_squared_error')

opt = int(input("Enter epochs size. Recommendation is 25 "))
opt2 = int(input("Enter batch size. Recommendation is 32 "))
model.fit(x_train, y_train, epochs=opt, batch_size=opt2)
```

In this section:

- The user is prompted to choose the optimizer ('adam' or 'SGD'), number of epochs, and batch size for training the LSTM model.

8. Test Model on Unseen Data

```
test_start = dt.datetime(2022, 1, 1)
test_end = dt.datetime.now()

test_data = pdr.get_data_yahoo(company, test_start, test_end)
```

In this section:

- A new date range (`test_start` to `test_end`) is defined for testing the model on unseen data.

9. Prepare Test Data and Make Predictions

```
actual_prices = test_data[choose].values

total_dataset = pd.concat((data[choose], test_data[choose]), axis=0)

model_input = total_dataset[len(total_dataset)- len(test_data)-
prediction_days:].values
model_input = model_input.reshape(-1,1)
model_input= scaler.transform(model_input)

x_test = []
for x in range(prediction_days, len(model_input)):
    x_test.append(model_input[x-prediction_days:x, 0])

x_test = np.array(x_test)
x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

prediction_prices = model.predict(x_test)
prediction_prices = scaler.inverse_transform(prediction_prices)
```

In this section:

- The actual prices for the test data are extracted.
- The model input is prepared by combining historical and test data, scaling it, and creating sequences for prediction.
- The model predicts stock prices for the test data.

10. Plot Actual vs Predicted Prices

```
plt.plot(actual_prices, color='black', label=f"Actual {company} Price")
plt.plot(prediction_prices, color='green', label=f"Predicted {company} Price")
plt.title(f"{company} Share Price")
plt.xlabel('Date')
plt.ylabel(f"{company} Share Price")
plt.legend()
plt.show()
```

In this section:

- The actual and predicted prices are plotted using `matplotlib` for visual comparison.

11. Choose Prediction Type

- You will be prompted to choose the prediction type:
- Enter 'Y' if you want to predict the next day.
- Enter 'N' if you want to predict more than one day.

```
YoN = input('Y if you want to predict next day or N if you want to predict
more than one day: ')
```

12. Next Day Prediction

- If you choose to predict the next day ('Y'), the script will generate a prediction for the following day based on the trained model.

```
  real_data = [model_input[len(model_input) + 1 -
prediction_days:len(model_input+1), 0]]
  real_data = np.array(real_data)
  real_data = np.reshape(real_data, (real_data.shape[0],
real_data.shape[1], 1))

  prediction = model.predict(real_data)
  prediction = scaler.inverse_transform(prediction)
  print(f"Prediction: {prediction}")
```

13. Consecutive Days Prediction

- If you choose to predict more than one day ('N'), you will be asked to input the number of days you want to predict.

```
NoD = int(input('How many days do you want to predict? '))
num_days_to_predict = NoD
```

- The script will then generate predictions for the specified number of consecutive days.

```
predictions = []

for day in range(num_days_to_predict):
    # Prepare data for the next day
    # Make prediction for the next day
    # Append the prediction to the list
```

14. Visualization Option

- After predictions are generated, you will be asked whether you want to plot the data or just see a readout of numbers.
- Enter 'P' if you want to plot the actual and predicted prices, or 'N' for a numerical readout.

```
PoN = input('Do you want to plot the data or just a readout of numbers?
"P" for plot. "N" for numbers. ')
```

- If 'P' is chosen, a plot will be displayed showing actual and predicted prices for the specified number of days.

```
  plt.plot(actual_prices, color='black', label=f"Actual {company} Prices")
  x_extended = range(len(actual_prices), len(actual_prices) +
num_days_to_predict)
  plt.plot(x_extended, predictions, color='red', label=f"Predicted {company}
Prices")
  plt.title(f"{company} Share Prices - Actual vs Predicted")
  plt.xlabel('Date')
  plt.ylabel(f"{company} Share Price")
  plt.legend()
  plt.show()
```

- If 'N' is chosen, a readout of predictions for each day will be displayed.

```
  for day, prediction in enumerate(predictions, 1):
      print(f"Prediction for Day {day}: {prediction}")
```