Project name: Next-Day Stock Price Forecast

Subject: Deep Learning

Project Link: https://github.com/paolosilv/deep-learning2023

Student: Paolo Silvestri

Student ID: 521343

Project Goals

Next day stock closing price (short-term prediction)

Should I buy or sell for tomorrow?

Data

Stock Symbol: AAPL -> APPLE

Provider: Yahoo Finance

Data Period: 2010-01-01 -> 2023-09-21

Primary Data Structure:

	Open	High	Low	Close	Adj Close	Volume
Date						
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.487533	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656429	6.498750	601904800
2010-01-06	7.656429	7.686786	7.526786	7.534643	6.395380	552160000
2010-01-07	7.562500	7.571429	7.466071	7.520714	6.383558	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.425995	447610800
...	...	...	...	...	...	...
2023-09-15	176.479996	176.500000	173.820007	175.009995	175.009995	109205100
2023-09-18	176.479996	179.380005	176.169998	177.970001	177.970001	67257600
2023-09-19	177.520004	179.630005	177.130005	179.070007	179.070007	51826900
2023-09-20	179.259995	179.699997	175.399994	175.490005	175.490005	58436200
2023-09-21	174.550003	176.300003	173.860001	173.929993	173.929993	63047900

Data Pre-Processing

Features: «Close», «Volume», «RSI»

Data scaling -> MinMaxScaler

Training sequence -> 5 days

Data Splitting

Train size -> 85% -> 2930 rows

Test size -> 15% -> 518 rows

Batch size -> 64

Model Structure

LSTM -> 2 Layers | 64 Hidden Size

Fully Connected -> 1 Output

```
class StockForecastingLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(StockForecastingLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)


    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out
```

Loss and Optimizer

Loss Metric -> MSE (Mean Squared Error)
$= (1/n) * \Sigma(y_i - \hat{y}_i)^2$

Optimizer -> ADAM (Adaptive Moment Estimation)

Training

Epochs -> 150

Learning Rate -> 0.001

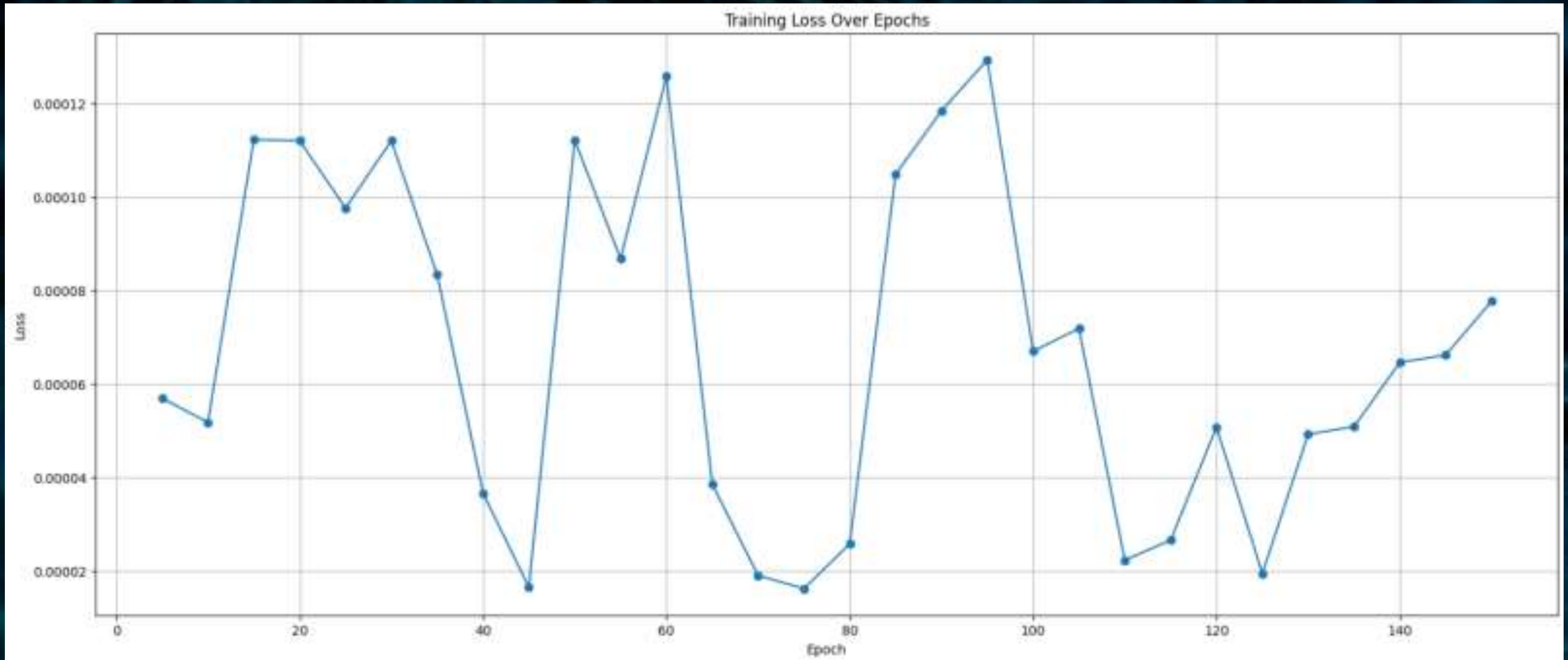
```python
# Training the model
losses = []
epochs = []
for epoch in range(num_epochs):
    for batch_seq, batch_target in train_loader:
        batch_seq = batch_seq.to(device)
        batch_target = batch_target.to(device)

        # Forward pass
        outputs = model(batch_seq)
        loss = criterion(outputs, batch_target)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 5 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.6f}')
        epochs.append(epoch+1)
        losses.append(loss.item())
```

```
Epoch [5/150], Loss: 0.000057
Epoch [10/150], Loss: 0.000052
Epoch [15/150], Loss: 0.000112
Epoch [20/150], Loss: 0.000112
Epoch [25/150], Loss: 0.000098
Epoch [30/150], Loss: 0.000112
Epoch [35/150], Loss: 0.000083
Epoch [40/150], Loss: 0.000037
Epoch [45/150], Loss: 0.000016
Epoch [50/150], Loss: 0.000112
Epoch [55/150], Loss: 0.000087
Epoch [60/150], Loss: 0.000126
Epoch [65/150], Loss: 0.000039
Epoch [70/150], Loss: 0.000019
Epoch [75/150], Loss: 0.000016
Epoch [80/150], Loss: 0.000026
Epoch [85/150], Loss: 0.000105
Epoch [90/150], Loss: 0.000118
Epoch [95/150], Loss: 0.000129
Epoch [100/150], Loss: 0.000067
Epoch [105/150], Loss: 0.000072
Epoch [110/150], Loss: 0.000022
Epoch [115/150], Loss: 0.000027
Epoch [120/150], Loss: 0.000051
Epoch [125/150], Loss: 0.000019
Epoch [130/150], Loss: 0.000049
Epoch [135/150], Loss: 0.000051
Epoch [140/150], Loss: 0.000065
Epoch [145/150], Loss: 0.000066
Epoch [150/150], Loss: 0.000078
```



Different approaches, average results

Epochs: 150, seq_length : 5 days, lr = 0.001, LSTM layers = 2, LSTM hidden size = 64

1) Features: «Close» | MSE: 12.244 | Accuracy: 51.14%

2) Features: «Close», «Volume», «RSI» | MSE: 11.764 | Accuracy: 54.66% -> the best one

3) Features: «Close», «Volume», «RSI», «MACD» | MSE: 18.338 | Accuracy: 52.53%

4) Features: «Close», «Volume», «RSI», «MACD», «ATR» | MSE: 20.048 | Accuracy: 51.32%

5) Features: «Close», «Volume», «RSI», «MACD», «ATR», «50-MA» | MSE: 13.261 | Accuracy: 52.54%

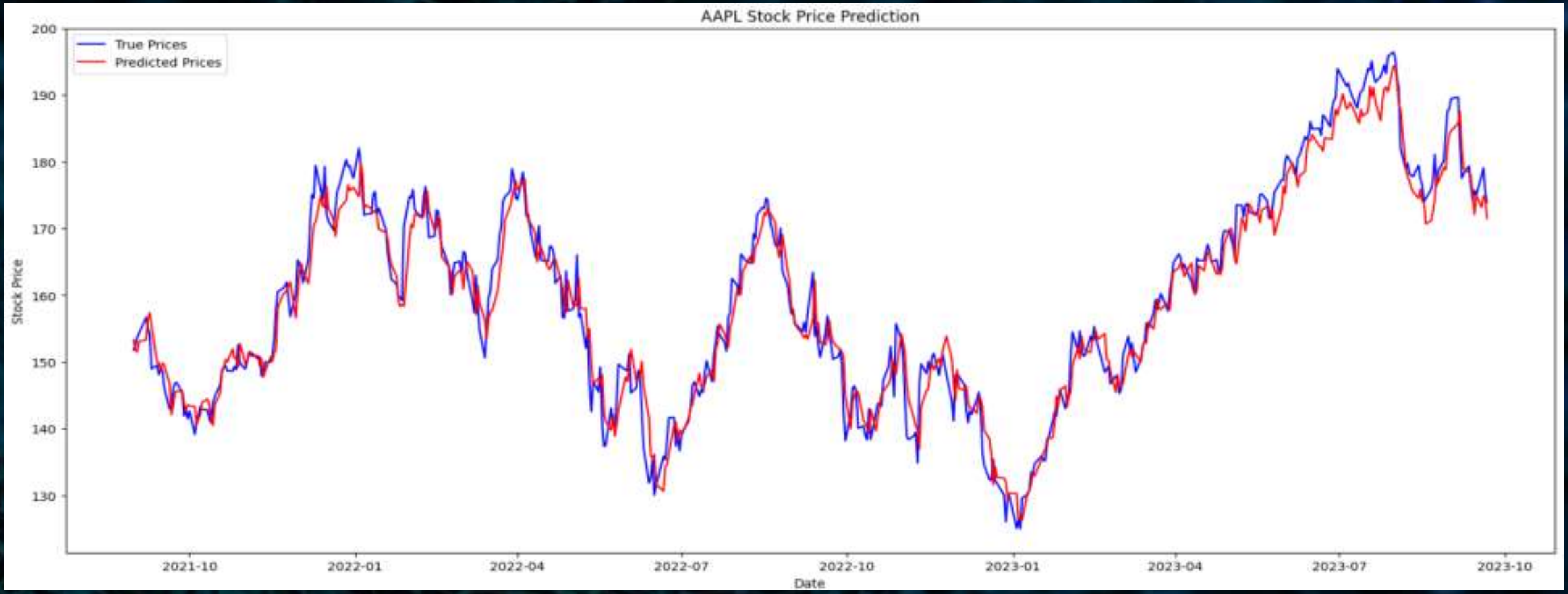
Testing

```
# Evaluate the model
model.eval()
test_seq = Variable(test_seq)
with torch.no_grad():
    predicted = model(test_seq).cpu().numpy()
    predicted = scaler.inverse_transform(predicted)
    true = scaler.inverse_transform(test_target.cpu().numpy())
```

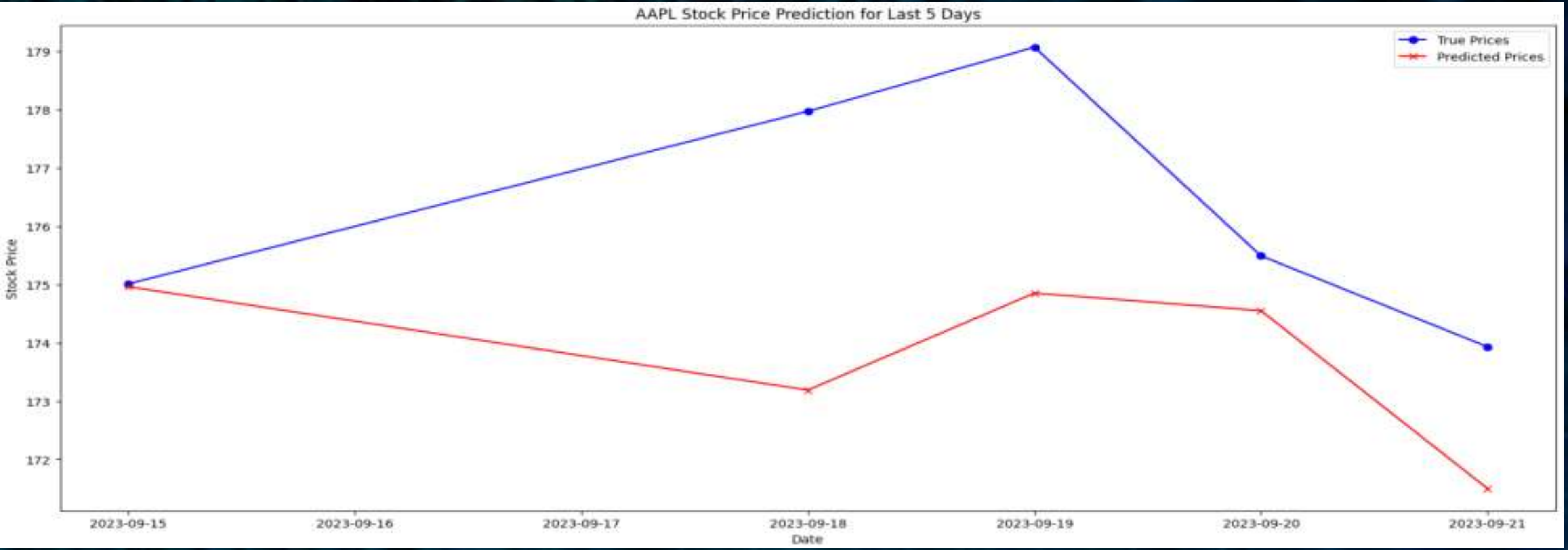
MSE -> 10.7306

```
Mean Squared Error (MSE): 10.7306
Root Mean Squared Error (RMSE): 3.2758
Mean Absolute Error (MAE): 2.5379
R-squared (R^2): 0.9583
```

Buy or Sell Accuracy -> 55.32%



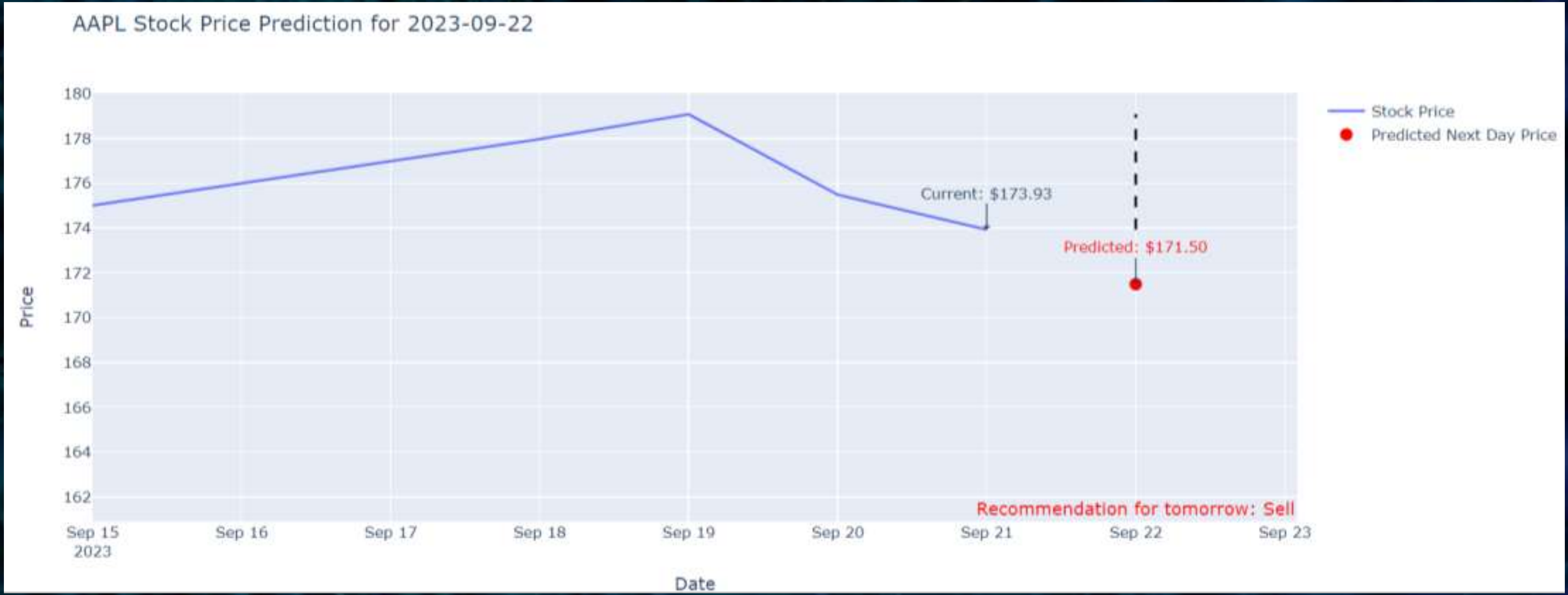
AAPL Stock Price Prediction
True Prices
Predicted Prices
Stock Price
Date
200
190
180
170
160
150
140
130
2021-10
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04
2023-07
2023-10



AAPL Stock Price Prediction for Last 5 Days
True Prices
Predicted Prices
Stock Price
Date
179
178
177
176
175
174
173
172
2023-09-15
2023-09-16
2023-09-17
2023-09-18
2023-09-19
2023-09-20
2023-09-21

Next Day Prediction -> 2023-09-22

Buy or Sell for tomorrow?



Thank You