Long Short-Term Memory (LSTM) Models

Time-Series Forecasting in Financial Markets Many financial institutions use LSTM models for predicting stock prices, cryptocurrency trends, and market volatility. One common approach is using LSTMs to predict the next day's price based on the last 30 days of stock prices.

Problem: Predict the next day's closing price for a stock using 30 days of historical data.

Steps:

1. Data Preparation: Stock market data (daily closing prices) is preprocessed to create time-series datasets.
2. Model Implementation: An LSTM model is trained on the prepared data to predict future stock prices.
3. Evaluation: The model is evaluated based on mean squared error (MSE) or mean absolute error (MAE).

Using the NSE data provided.

Import Libraries and Load Stock Data

```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

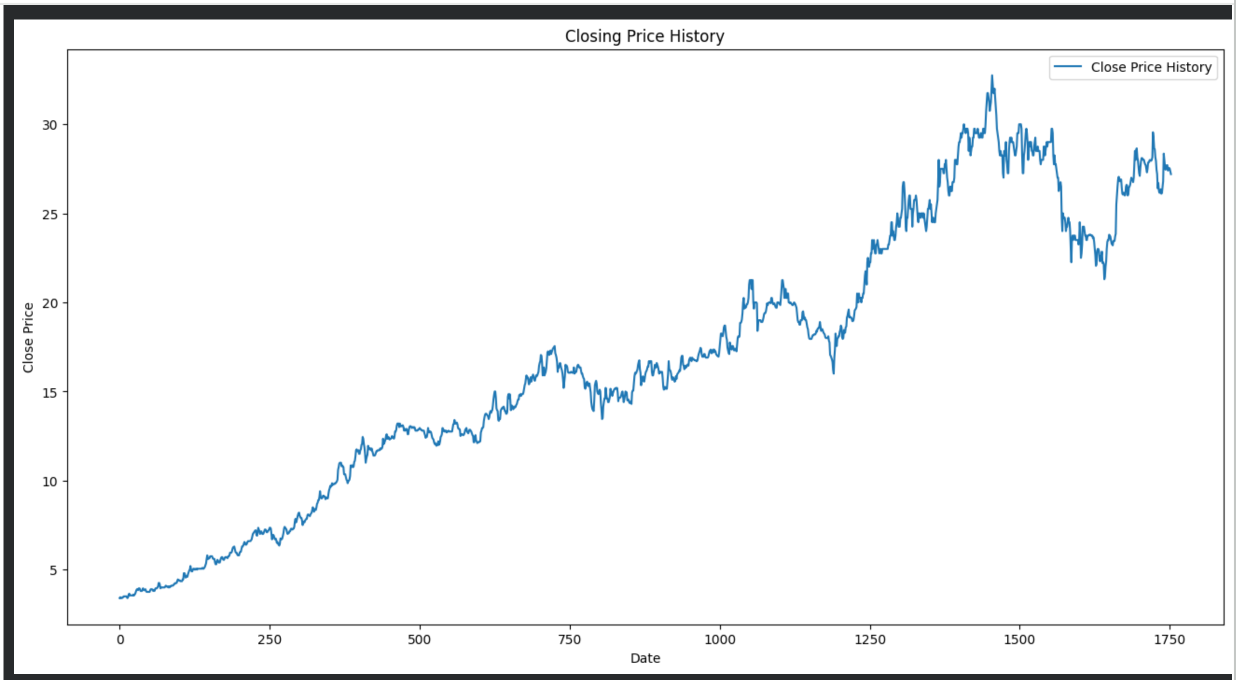
```
# load the dataset provided
df = pd.read_csv('/content/NSE_SCOM_Safaricom.csv')
print(df.head())
```

```
         Date  Open  High   Low  Close    Vol. Change %
0  11-06-2012  3.40  3.45  3.35   3.40   3.24M    0.00%
1  12-06-2012  3.40  3.50  3.40   3.45   6.09M    1.47%
2  13-06-2012  3.45  3.45  3.35   3.40   7.29M   -1.45%
3  14-06-2012  3.40  3.45  3.35   3.40  31.81M    0.00%
4  15-06-2012  3.40  3.50  3.40   3.40   7.57M    0.00%
```

Data Preprocessing

```
# Close' price because it is commonly used for stock price predictio
data = df[['Close']]

# visualize the closing price history
plt.figure(figsize=(16,8))
plt.plot(data['Close'], label='Close Price History')
plt.title('Closing Price History')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



```
# use MinMaxScaler to scale the data to be between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# prepare the data for LSTM
X = []
y = []
sequence_length = 60  # use past 60 days to predict the next day
for i in range(sequence_length, len(scaled_data)):
    X.append(scaled_data[i-sequence_length:i, 0])
    y.append(scaled_data[i, 0])
```

```
# Convert to numpy arrays
X, y = np.array(X), np.array(y)
```

```
# Reshape X for LSTM input
X = np.reshape(X, (X.shape[0], X.shape[1], 1)) # LSTM expects 3D ir
```

Build and Train the LSTM Model

```
# load libraries for LSTM
import tensorflow as tf
# Use Keras bundled with TensorFlow for compatibility
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
# build the LSTM model
model = Sequential()
# First LSTM layer with Dropout regularisation
model.add(LSTM(units=50, return_sequences=True, input_shape=(X.shape
model.add(Dropout(0.2))
# Second LSTM layer
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))

# fully connected output layer
model.add(Dense(units=1))

# compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# train the model
model.fit(X, y, epochs=50, batch_size=32)
```

```
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0021
Epoch 22/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0020
Epoch 23/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - loss: 0.0017
Epoch 24/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0018
Epoch 25/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0020
Epoch 26/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0018
Epoch 27/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 0.0018
Epoch 28/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 4s 79ms/step - loss: 0.0014
Epoch 29/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - loss: 0.0017
Epoch 30/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 5s 46ms/step - loss: 0.0015
Epoch 31/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 3s 56ms/step - loss: 0.0016
Epoch 32/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - loss: 0.0015
Epoch 33/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0014
Epoch 34/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0017
Epoch 35/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0014
Epoch 36/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - loss: 0.0015
Epoch 37/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - loss: 0.0017
Epoch 38/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 5s 46ms/step - loss: 0.0016
Epoch 39/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 3s 45ms/step - loss: 0.0013
Epoch 40/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - loss: 0.0014
Epoch 41/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 0.0014
Epoch 42/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 0.0014
Epoch 43/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0014
Epoch 44/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 0.0012
Epoch 45/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 4s 76ms/step - loss: 0.0013
Epoch 46/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 4s 45ms/step - loss: 0.0011
Epoch 47/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 0.0012
```

```
Epoch 48/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - loss: 0.0013
Epoch 49/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - loss: 0.0012
Epoch 50/50
53/53                      4s 46ms/step - loss: 0.0015
```

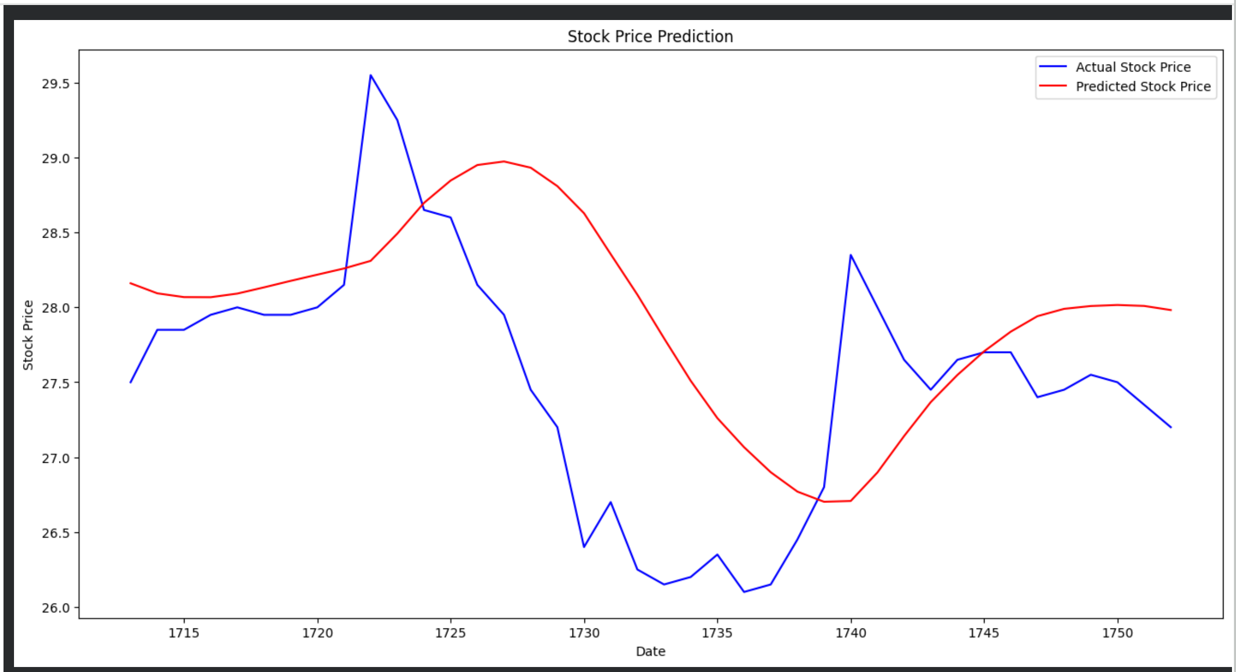
Predictions and Visualizations

```
# predict stock prices
test_data = data.tail(100)  # last 100 days for testing
scaled_test_data = scaler.transform(test_data)
X_test = []
for i in range(sequence_length, len(scaled_test_data)):
    X_test.append(scaled_test_data[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_prices = model.predict(X_test) # make predictions
predicted_prices = scaler.inverse_transform(predicted_prices) # res
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 404ms/step
```

```
# visualize the results
plt.figure(figsize=(16,8))
plt.plot(test_data.index[sequence_length:], test_data['Close'][sequ
plt.plot(test_data.index[sequence_length:], predicted_prices, color
plt.title('Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



Model Evaluation

Root Mean Squared Error (RMSE) is used to assess the accuracy of our predictions. A lower RMSE indicates better performance.

```
from sklearn.metrics import mean_squared_error

# calculate rmse
rmse = np.sqrt(mean_squared_error(test_data['Close'][sequence_length
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
Root Mean Squared Error (RMSE): 0.9165
```