Time Series Prediction of Coca-Cola Stock Prices Using LSTM

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, mean_absolute_percentage_e
```

```
# Load the data
data = pd.read_csv('/content/Coca-Cola_stock_history.csv')
```

```
# Convert Date to datetime format, handling extra characters
data['Date'] = pd.to_datetime(data['Date'], format='%Y-%m-%d %H:%M:%S%z', errors='coerce')
```

```
# Set Date as index
data.set_index('Date', inplace=True)
```

```
# Select relevant columns
features = data[['Close', 'Volume', 'Dividends', 'Stock Splits']]
```

```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_prices = scaler.fit_transform(prices)
```

```python
# Create sequences of data
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

seq_length = 50
X, y = create_sequences(scaled_prices, seq_length)
```

```python
# Reshape X to fit the RNN input
X = X.reshape((X.shape[0], X.shape[1], 1))
```

```python
# Split data into train and test sets
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

```python
# Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(seq_length, X.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 50, 50)            10400

 dropout_16 (Dropout)        (None, 50, 50)            0
```

```
 lstm_7 (LSTM)               (None, 50)                20200

 dropout_17 (Dropout)        (None, 50)                0

 dense_8 (Dense)             (None, 1)                 51

=================================================================
Total params: 30651 (119.73 KB)
Trainable params: 30651 (119.73 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
```

```
382/382 [==============================] - 21s 54ms/step - loss: 8.2354e-05 - val_loss: 0.0022
Epoch 35/50
382/382 [==============================] - 21s 54ms/step - loss: 8.3430e-05 - val_loss: 0.0045
Epoch 36/50
382/382 [==============================] - 21s 55ms/step - loss: 8.2163e-05 - val_loss: 0.0036
Epoch 37/50
382/382 [==============================] - 20s 53ms/step - loss: 8.1604e-05 - val_loss: 0.0049
Epoch 38/50
382/382 [==============================] - 21s 56ms/step - loss: 8.0913e-05 - val_loss: 0.0023
Epoch 39/50
382/382 [==============================] - 20s 53ms/step - loss: 8.2749e-05 - val_loss: 0.0037
Epoch 40/50
382/382 [==============================] - 21s 55ms/step - loss: 7.8992e-05 - val_loss: 0.0033
Epoch 41/50
382/382 [==============================] - 20s 52ms/step - loss: 7.8135e-05 - val_loss: 0.0074
Epoch 42/50
382/382 [==============================] - 21s 56ms/step - loss: 8.1521e-05 - val_loss: 0.0039
Epoch 43/50
382/382 [==============================] - 20s 51ms/step - loss: 7.6379e-05 - val_loss: 0.0051
Epoch 44/50
382/382 [==============================] - 21s 56ms/step - loss: 7.8489e-05 - val_loss: 0.0057
Epoch 45/50
382/382 [==============================] - 20s 52ms/step - loss: 7.7944e-05 - val_loss: 0.0053
Epoch 46/50
382/382 [==============================] - 21s 56ms/step - loss: 8.1946e-05 - val_loss: 0.0055
Epoch 47/50
382/382 [==============================] - 20s 51ms/step - loss: 7.9614e-05 - val_loss: 0.0048
Epoch 48/50
382/382 [==============================] - 21s 56ms/step - loss: 8.0926e-05 - val_loss: 0.0087
Epoch 49/50
382/382 [==============================] - 20s 52ms/step - loss: 7.7035e-05 - val_loss: 0.0054
Epoch 50/50
382/382 [==============================] - 21s 55ms/step - loss: 7.7047e-05 - val_loss: 0.0059
```

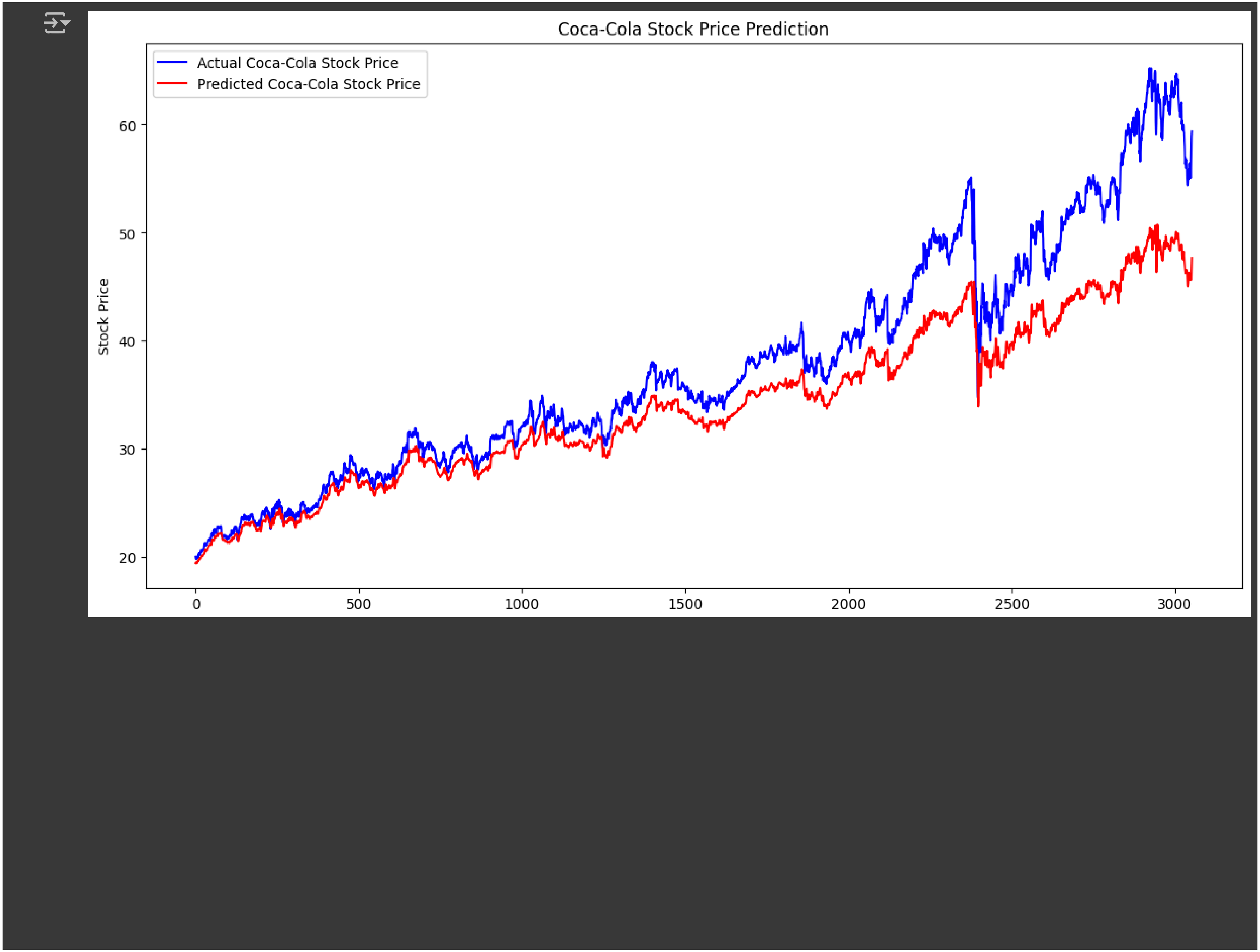
```
# Evaluate the model performance
predicted_prices = model.predict(X_test)
predicted_prices = scaler.inverse_transform(np.concatenate((predicted_prices, np.zeros((predicted_prices.shape[0], 3)))
actual_prices = scaler.inverse_transform(np.concatenate((y_test.reshape(-1, 1), np.zeros((y_test.shape[0], 3))), axis=1
```

```
96/96 [==============================] - 2s 14ms/step
```

```python
# Calculate Mean Squared Error (MSE) and Mean Absolute Error (MAE)
mse = mean_squared_error(actual_prices, predicted_prices)
mae = mean_absolute_error(actual_prices, predicted_prices)
r2 = r2_score(actual_prices, predicted_prices)
mape = mean_absolute_percentage_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R2): {r2}")
print(f"Mean Absolute Percentage Error (MAPE): {mape}")
```

```
Mean Squared Error (MSE): 25.244215153620342
Mean Absolute Error (MAE): 3.680703339364588
R-squared (R2): 0.7800762457145903
Mean Absolute Percentage Error (MAPE): 0.08267929786686115
```

```python
# Plotting the actual vs predicted prices
plt.figure(figsize=(14, 7))
plt.plot(actual_prices, color='blue', label='Actual Coca-Cola Stock Price')
plt.plot(predicted_prices, color='red', label='Predicted Coca-Cola Stock Price')
plt.title('Coca-Cola Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



Coca-Cola Stock Price Prediction
Actual Coca-Cola Stock Price
Predicted Coca-Cola Stock Price
Stock Price
60
50
40
30
20
0
500
1000
1500
2000
2500
3000