- **Mô tả thuộc tính:**

**Ngày:** Ngày giao dịch.
**Lần cuối:** Giá cuối cùng (giá đóng cửa) của cổ phiếu trong ngày giao dịch đó.
**Mở:** Giá mở cửa của cổ phiếu trong ngày giao dịch đó.
**Cao:** Giá cao nhất của cổ phiếu trong ngày giao dịch đó.
**Thấp:** Giá thấp nhất của cổ phiếu trong ngày giao dịch đó.
**KL:** Khối lượng giao dịch của cổ phiếu trong ngày giao dịch đó.
**% Thay đổi:** Phần trăm thay đổi của giá cổ phiếu so với ngày giao dịch trước đó.

- **Tiền xử lý dữ liệu:**
  **1. Kiểm tra dữ liệu**
  Kiểm tra kiểu dữ liệu

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 501 entries, 0 to 500
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Ngày        501 non-null    object
 1   Lần cuối    501 non-null    object
 2   Mở          501 non-null    object
 3   Cao         501 non-null    object
 4   Thấp        501 non-null    object
 5   KL          501 non-null    object
 6   % Thay đổi  501 non-null    object
dtypes: object(7)
memory usage: 27.5+ KB
```

  Kiểm tra 10 dòng đầu để thấy được tổng quan kiểu dữ liệu khi bắt đầu xử lý.

```
df.head(10)
```
✓ 0.0s

| | Ngày | Lần cuối | Mở | Cao | Thấp | KL | % Thay đổi |
|---|---|---|---|---|---|---|---|
| 0 | 20/06/2024 | 65,900.0 | 65,100.0 | 66,100.0 | 65,100.0 | 905.40K | 0.92% |
| 1 | 19/06/2024 | 65,300.0 | 66,000.0 | 66,000.0 | 65,100.0 | 6.16M | -0.91% |
| 2 | 18/06/2024 | 65,900.0 | 66,500.0 | 66,500.0 | 65,900.0 | 2.75M | -0.15% |
| 3 | 17/06/2024 | 66,000.0 | 66,300.0 | 66,600.0 | 66,000.0 | 3.62M | -0.30% |
| 4 | 14/06/2024 | 66,200.0 | 67,000.0 | 67,300.0 | 66,200.0 | 4.12M | -1.05% |
| 5 | 13/06/2024 | 66,900.0 | 67,000.0 | 67,400.0 | 66,800.0 | 2.80M | -0.15% |
| 6 | 12/06/2024 | 67,000.0 | 66,500.0 | 67,300.0 | 66,300.0 | 3.43M | 0.75% |
| 7 | 11/06/2024 | 66,500.0 | 67,400.0 | 67,500.0 | 66,400.0 | 6.18M | -1.04% |
| 8 | 10/06/2024 | 67,200.0 | 68,400.0 | 68,500.0 | 67,200.0 | 4.56M | -0.88% |
| 9 | 07/06/2024 | 67,800.0 | 67,600.0 | 68,600.0 | 67,500.0 | 4.13M | 0.89% |

=> Cần chỉnh lại các kiểu dữ liệu :

**Ngày** : Datetime

**Lần cuối, Mở, Cao, Thấp, % Thay đổi**: Float

**KL** : Int (Cần chuyển đổi khối lượng giao dịch từ dạng chuỗi có hậu tố sang số nguyên)\

**2. Xử lý dữ liệu và kiểm tra lại:**

```
# Chuyển đổi cột 'Ngày' sang kiểu datetime
df['Ngày'] = pd.to_datetime(df['Ngày'], format='%d/%m/%Y')

# Loại bỏ dấu phẩy và chuyển đổi các cột số sang kiểu float
df['Lần cuối'] = df['Lần cuối'].str.replace(',', '').astype(float)
df['Mở'] = df['Mở'].str.replace(',', '').astype(float)
df['Cao'] = df['Cao'].str.replace(',', '').astype(float)
df['Thấp'] = df['Thấp'].str.replace(',', '').astype(float)
```

✓ 0.0s

```
# Chuyển đổi khối lượng giao dịch từ dạng chuỗi có hậu tố sang số nguyên
def convert_volume(volume):
    if 'K' in volume:
        return int(float(volume.replace('K', '')) * 1000)
    elif 'M' in volume:
        return int(float(volume.replace('M', '')) * 1000000)
    else:
        return int(volume)

df['KL'] = df['KL'].apply(convert_volume)

# Chuyển đổi phần trăm thay đổi từ dạng chuỗi sang số thực
df['% Thay đổi'] = df['% Thay đổi'].str.replace('%', '').astype(float)

# Kiểm tra lại kiểu dữ liệu
df.info()
```

✓ 0.0s

Các kiểu dữ liệu đã được thay đổi

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 501 entries, 0 to 500
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Ngày        501 non-null    datetime64[ns]
 1   Lần cuối    501 non-null    float64
 2   Mở          501 non-null    float64
 3   Cao         501 non-null    float64
 4   Thấp        501 non-null    float64
 5   KL          501 non-null    int64
 6   % Thay đổi  501 non-null    float64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 27.5 KB
```

Kiểm tra lại dữ liệu:

```
df.head(10)
```

0.0s

| | Ngày | Lần cuối | Mở | Cao | Thấp | KL | % Thay đổi |
|---|---|---|---|---|---|---|---|
| 0 | 2024-06-20 | 65900.0 | 65100.0 | 66100.0 | 65100.0 | 905400 | 0.92 |
| 1 | 2024-06-19 | 65300.0 | 66000.0 | 66000.0 | 65100.0 | 6160000 | -0.91 |
| 2 | 2024-06-18 | 65900.0 | 66500.0 | 66500.0 | 65900.0 | 2750000 | -0.15 |
| 3 | 2024-06-17 | 66000.0 | 66300.0 | 66600.0 | 66000.0 | 3620000 | -0.30 |
| 4 | 2024-06-14 | 66200.0 | 67000.0 | 67300.0 | 66200.0 | 4120000 | -1.05 |
| 5 | 2024-06-13 | 66900.0 | 67000.0 | 67400.0 | 66800.0 | 2800000 | -0.15 |
| 6 | 2024-06-12 | 67000.0 | 66500.0 | 67300.0 | 66300.0 | 3430000 | 0.75 |
| 7 | 2024-06-11 | 66500.0 | 67400.0 | 67500.0 | 66400.0 | 6180000 | -1.04 |
| 8 | 2024-06-10 | 67200.0 | 68400.0 | 68500.0 | 67200.0 | 4560000 | -0.88 |
| 9 | 2024-06-07 | 67800.0 | 67600.0 | 68600.0 | 67500.0 | 4130000 | 0.89 |

Dữ liệu đã được xử lý để phù hợp cho việc áp dụng mô hình.

**Tuần 3,4,5: Sau khi tiền xử lý dữ liệu, tiếp theo ta cần phân chia dữ liệu và xây dựng mô hình**

- # Phân chia dữ liệu

### 1. Sử dụng index

Ta sử dụng index để chọn ra 2 thuộc tính cần sử dụng để áp dụng cho mô hình, cũng một phần giúp truy xuất dữ liệu nhanh hơn.

```
df1_full = pd.DataFrame(df,columns=['Ngày','Lần cuối'])
df1_full.index = df1_full.Ngày
df1_full.drop('Ngày',axis=1,inplace=True)
df1_full.info
```
✓ 0.0s

Có các thuộc tính biểu thị về giá trị như Mở (Open), Lần Cuối (Close), Cao (High), và Thấp (Low), nhưng khi dự đoán giá cổ phiếu, thuộc tính Lần Cuối (Close) là thuộc tính được chọn để áp dụng cho mô hình. Lý do chính là:

- Giá đã thống nhất vào thời điêm đó, dựa trên thông tin mua và hoạt động mua bán trong ngày.

- Giá Lần Cuối ít bị ảnh hưởng bởi những biến động ngắn hạn trong ngày so với Giá Mở, Cao và Thấp. Nó thường là giá mà các báo cáo và phân tích tài chính sử dụng.

- Giá Lần Cuối có ảnh hưởng đến chiến lược giao dịch của nhà đầu tư cho ngày tiếp theo. Nếu giá đóng cửa cao hơn giá mở cửa, có thể có xu hướng tăng giá vào ngày hôm sau và ngược lại.

### 2. Tách dữ liệu.

Với ý tưởng là sẽ chỉ sử dụng 75% của bộ dữ liệu để train và test mô hình, và sử dụng mô hình để đoán 25% còn lại. Sau khi dự đoán, ta lấy 25% dữ liệu chưa được sử dụng trong mô hình để chứng thực độ chính xác của mô hình.

```
start_index = int(0.75 * len(df1_full))

# Trích xuất phần tử từ chỉ số đã tính
df1 = df1_full[:start_index]

# In kết quả để kiểm tra
print(df1)
```
✓ 0.0s

Lấy dữ liệu từ index trước đó để chia dữ liệu.

```
            Lần cuối
Ngày
2022-06-20   62169.9
2022-06-21   66404.7
2022-06-22   63611.5
2022-06-23   63701.6
2022-06-24   64422.4
...              ...
2023-12-07   67030.3
2023-12-08   67618.3
2023-12-11   68598.2
2023-12-12   68010.3
2023-12-13   66834.3

[375 rows x 1 columns]
```

Kết quả từ bộ dữ liệu với thời gian từ 20/6/2022 tới 20/6/2024 thì bây giờ chỉ còn từ 20/6/2022 tới 1/12/2023. Tức là ~75% bộ dữ liệu ban đầu với 375 dòng.

**3. Phân chia dữ liệu train, test.**

```
data = df1.values
train_data = data[:300]
test_data = data[300:]

✓ 0.0s
```

Chia dữ liệu thành 2 tập train và test với train là 300 dòng dữ liệu đầu và test là phần còn lại.

```
# Chuẩn hóa dữ liệu
sc = MinMaxScaler(feature_range=(0, 1))
sc_train = sc.fit_transform(data)
✓ 0.0s
```

Chuẩn hóa dữ liệu giúp đưa các giá trị về cùng một phạm vi để mô hình có thể hiệu quả hơn trong việc học và dự đoán. Trong trường hợp này, MinMaxScaler được sử dụng để chuyển đổi giá cổ phiếu về khoảng từ 0 đến 1 trước khi đưa vào mô hình LSTM. Điều này giúp cải thiện tính ổn định và hiệu suất của mô hình dự đoán.

```
a = 100
✓ 0.0s
```

Đặt biến **a** = 100

```
#tạo vòng lặp các giá trị
x_train,y_train=[],[]
for i in range(a,len(train_data)):
    x_train.append(sc_train[i-a:i,0]) #lấy a giá đóng cửa liên tục
    y_train.append(sc_train[i,0]) #lấy ra giá đóng cửa ngày hôm sau
✓ 0.0s
```

Mô hình LSTM yêu cầu đầu vào là các chuỗi liên tục để có thể học được các mẫu thời gian. Do đó,cần tạo ra các chuỗi liên tục từ dữ liệu lịch sử, tạo ra các chuỗi liên tục dài a ngày.

Cụ thể, vòng lặp tác dụng:

- Nếu a = 100 và dữ liệu lịch sử có độ dài n, thì

- Từ ngày 1 đến ngày 100 tạo thành một chuỗi và ngày 101 là giá trị cần dự đoán.

- Từ ngày 2 đến ngày 101 tạo thành một chuỗi khác và ngày 102 là giá trị cần dự đoán.

- Tiếp tục như vậy cho đến khi đạt đến cuối bộ dữ liệu.

```
#xếp dữ liệu thành 1 mảng 2 chiều
x_train = np.array(x_train)
y_train = np.array(y_train)

#xếp lại dữ liệu thành mảng 1 chiều
x_train = np.reshape(x_train,(x_train.shape[0],x_train.shape[1],1))
y_train = np.reshape(y_train,(y_train.shape[0],1))
✓ 0.0s
```

- Sau khi thu thập dữ liệu trong vòng lặp**, x_train và y_train** được chuyển đổi thành các **numpy array.**

- **x_train** được **reshape** từ mảng 2 chiều thành **tensor** 3 chiều (số mẫu, số thời điểm quan sát, số đặc trưng), trong đó số mẫu là số lượng mẫu dữ liệu huấn luyện, số thời điểm quan sát là 100 (do đã chọn lấy 100 ngày liền kề), và số đặc trưng là 1 (do chỉ sử dụng giá đóng cửa làm đặc trưng).

- **y_train** được reshape thành mảng 2 chiều (số mẫu, 1), mỗi hàng chứa một giá trị giá cổ phiếu tương ứng với mỗi mẫu trong **x_train**.

- # Xây dựng mô hình

```
#xây dựng mô hình
model = Sequential() #tạo lớp mạng cho dữ liệu đầu vào
#2 lớp LSTM
model.add(LSTM(units=128,input_shape=(x_train.shape[1],1),return_sequences=True))
model.add(LSTM(units=64))
model.add(Dropout(0.15)) #loại bỏ 1 số đơn vị tránh học tủ (overfitting)
model.add(Dense(1)) #output đầu ra 1 chiều
#đo sai số tuyệt đối trung bình có sử dụng trình tối ưu hóa adam
model.compile(loss='mean_absolute_error',optimizer='adam')
✓ 0.0s
```

Khởi tạo mô hình **Sequential** để xây dựng các lớp theo thứ tự (phù hợp cho mô hình có chuỗi thời gian như dư đoán giá cổ phiếu.
- Tạo các lớp **LSTM** có **128 đơn vị** để học và ghi nhớ các chuỗi thời gian dài và **64 đơn vị** tiếp tục học từ các mẫu đã đc lớp **128 đơn vị** trích xuất.
- Sử dụng **Dropout** để loại bỏ ngẫu nhiên **15%** (có thể điều chỉnh cho phù hợp) để giảm overfitting.
- Sau khi qua các lớp LSTM, thông thường ta cần dùng **Dense** để giảm số chiều về kết quả cuối cùng mong đợi (ở đây dense là 1 vì chỉ dự đoán 1 thuộc tính là “Lần cuối”)
- Đánh giá lỗi dựa trên sai số tuyệt đối trung bình **('mean_absolute_error')** và sử dụng thuật toán tối ưu **adam** để cải thiện quá trình học **(optimizer='adam')**.

```python
#huấn luyện mô hình
save_model = "D:\TÀI LIỆU HỌC TẬP\Đề Tài Tốt Nghiệp\save_model.keras"
best_model = ModelCheckpoint(save_model,monitor='loss',verbose=2,save_best_only=True,mode='auto')
model.fit(x_train,y_train,epochs=100,batch_size=a,verbose=2,callbacks=[best_model])
```
✓ 51.1s

- Mô hình sẽ được lưu vào thư mục "D:\TÀI LIỆU HỌC TẬP\Đề Tài Tốt Nghiệp" với tên tệp là "save_model.keras". Sau đó, đo lường để theo dõi và quyết định mô hình nào là tốt nhất để lưu lại. Trong trường hợp này, mô hình sẽ theo dõi giá trị mất mát (loss) và chỉ lưu lại mô hình tốt nhất dựa trên giá trị được theo dõi (trong trường hợp này là loss). Nếu được đặt thành True, mô hình sẽ chỉ lưu lại khi giá trị giảm.
- Huấn luyện mô hình với dữ liệu huấn luyện và các tham số cấu hình có thể tùy chỉnh như **epochs=100** (Số lượng lần lặp lại để huấn luyện mô hình trên toàn bộ dữ liệu), **batch_size=a(100)** (Số lượng mẫu dữ liệu được sử dụng trong mỗi lần cập nhật gradient).

```python
#dữ liệu train
y_train = sc.inverse_transform(y_train) #giá thực
final_model = load_model("D:\TÀI LIỆU HỌC TẬP\Đề Tài Tốt Nghiệp\save_model.keras")
y_train_predict = final_model.predict(x_train) #dự đoán giá đóng cửa trên tập đã train
y_train_predict = sc.inverse_transform(y_train_predict) #giá dự đoán
```
✓ 1.4s

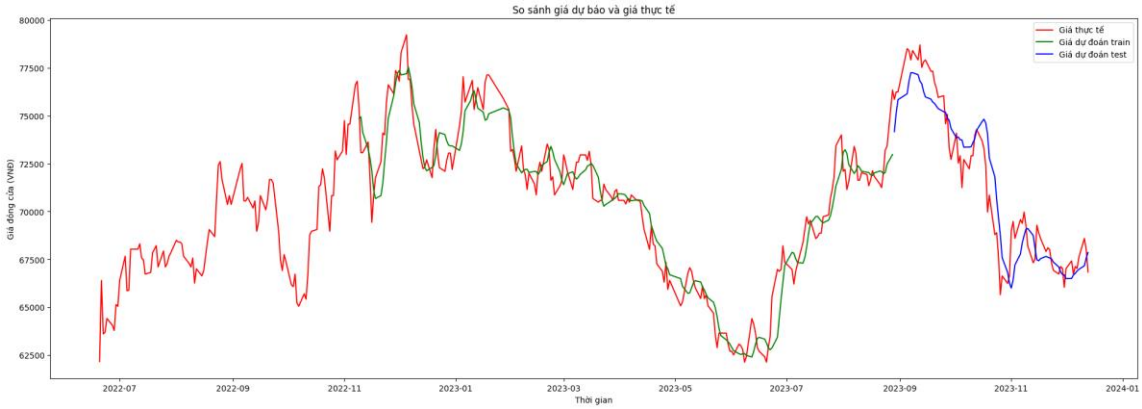
Sử dụng mô hình đã huấn luyện để dự đoán giá train.

```python
#xử lý dữ liệu test
test = df1[len(train_data)-a:].values
test = test.reshape(-1,1)
sc_test = sc.transform(test)

x_test = []
for i in range(a,test.shape[0]):
  x_test.append(sc_test[i-a:i,0])
x_test = np.array(x_test)
x_test = np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))

#dữ liệu test
y_test = data[300:] #giá thực
y_test_predict = final_model.predict(x_test)
y_test_predict = sc.inverse_transform(y_test_predict) #giá dự đoán
```
✓ 0.2s

Sử dụng mô hình đã huấn luyện để dự đoán tập test.

```python
#lập biểu đồ so sánh
train_data1 = df1[a:300]
test_data1 = df1[300:]

plt.figure(figsize=(24,8))
plt.plot(df1,label='Giá thực tế',color='red') #đường giá thực
train_data1['Dự đoán'] = y_train_predict #thêm dữ liệu
plt.plot(train_data1['Dự đoán'],label='Giá dự đoán train',color='green') #đường giá dự báo train
test_data1['Dự đoán'] = y_test_predict #thêm dữ liệu
plt.plot(test_data1['Dự đoán'],label='Giá dự đoán test',color='blue') #đường giá dự báo test
plt.title('So sánh giá dự báo và giá thực tế') #đặt tên biểu đồ
plt.xlabel('Thời gian') #đặt tên hàm x
plt.ylabel('Giá đóng cửa (VNĐ)') #đặt tên hàm y
plt.legend() #chú thích
plt.show()
```
✓ 0.4s

Xây dựng biểu đồ thể hiện giá dự đoán của train và test trên dữ liệu thực tế.




Kết quả của dữ liệu được dự đoán của train và test trên dữ liệu thực tế.

```
#  ĐÁNH GIÁ ĐỘ PHÙ HỢP, SAI SỐ, ...
print('Độ phù hợp tập train:',r2_score(y_train,y_train_predict))
print('Sai số tuyệt đối trung bình trên tập train (VNĐ):',mean_absolute_error(y_train,y_train_predict))
print('Phần trăm sai số tuyệt đối trung bình tập train:',mean_absolute_percentage_error(y_train,y_train_predict))
```

✓ 0.0s

```
Độ phù hợp tập train: 0.9246046295525665
Sai số tuyệt đối trung bình trên tập train (VNĐ): 854.1834843750004
Phần trăm sai số tuyệt đối trung bình tập train: 0.012067915209445586
```

```
print('Độ phù hợp tập test:',r2_score(y_test,y_test_predict))
print('Sai số tuyệt đối trung bình trên tập test (VNĐ):',mean_absolute_error(y_test,y_test_predict))
print('Phần trăm sai số tuyệt đối trung bình tập test:',mean_absolute_percentage_error(y_test,y_test_predict))
```

✓ 0.0s

```
Độ phù hợp tập test: 0.8818366763628864
Sai số tuyệt đối trung bình trên tập test (VNĐ): 1130.5776249999994
Phần trăm sai số tuyệt đối trung bình tập test: 0.01582050964820141
```

Đánh giá độ phù hợp, sai số… của dữ liệu dự đoán với dữ liệu thực tế.

- **Dự đoán giá cổ phiếu của 25% bộ dữ liệu chưa được sử dụng (Cuối 2023 – 20/6/2024)**

```
import datetime

# Nạp mô hình đã huấn luyện
model = load_model("D:\TÀI LIỆU HỌC TẬP\Đề Tài Tốt Nghiệp\save_model.keras")

# Đặt ngày kết thúc dự đoán
end_date = datetime.datetime(2024, 6, 20)

# Lấy ngày cuối cùng trong tập dữ liệu hiện tại
last_date = df1.index[-1]

# Tạo tập dữ liệu cho dự đoán
future_dates = []
predicted_prices = []
```

Chuẩn bị dữ liệu và mô hình đã huấn luyện cho việc dự đoán.

```
# Chuẩn bị dữ liệu dự đoán ban đầu
last_a_days = df1['Lần cuối'].values[-a:]
scaled_last_a_days = sc.transform(last_a_days.reshape(-1, 1))

while last_date < end_date:
    last_date += datetime.timedelta(days=1)
    # Bỏ qua các ngày thứ Hai và thứ Ba
    if last_date.weekday() in [0, 1]:
        continue
    future_dates.append(last_date)

    # Chuẩn bị dữ liệu đầu vào cho mô hình
    x_predict = np.array([scaled_last_a_days])
    x_predict = np.reshape(x_predict, (x_predict.shape[0], x_predict.shape[1], 1))

    # Dự đoán giá
    predicted_price = model.predict(x_predict)
    predicted_prices.append(predicted_price[0, 0])

    # Cập nhật dữ liệu dự đoán
    scaled_last_a_days = np.append(scaled_last_a_days[1:], predicted_price)
    scaled_last_a_days = scaled_last_a_days.reshape(-1, 1)
```

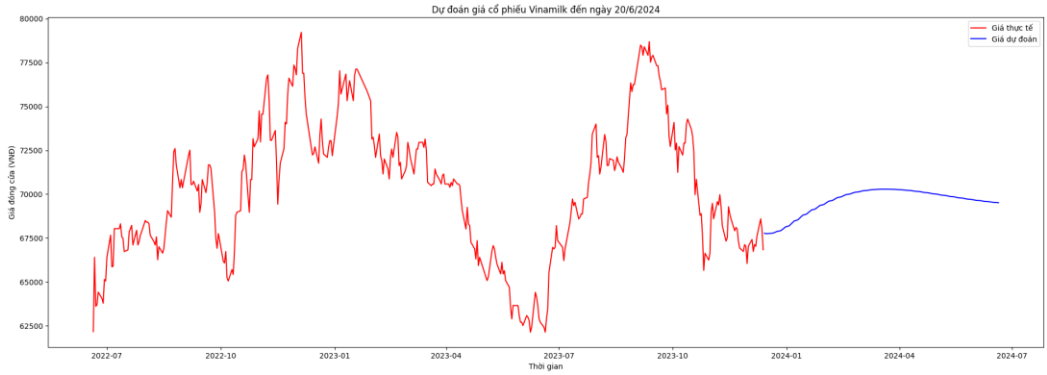
- Bỏ qua ngày thứ 2 và thứ 3 trong tuần vì 2 ngày đó không có các cuộc giao dịch, giảm thiểu sai số khi dự đoán.
- Chuyển đổi dữ liệu thành mảng và áp dụng mô hình đã huấn luyện để dự đoán.
- Cập nhật dữ liệu bằng cách cắt bớt dữ liệu cũ bằng cách loại bỏ phần tử đầu tiên **(scaled_last_a_days[1])** và thêm **predicted_price** vào cuối mảng.
- Sau khi đã cập nhật dữ liệu mới vào **scaled_last_a_days**, dòng **scaled_last_a_days = scaled_last_a_days.reshape(-1, 1)** thực hiện **reshape** lại thành một mảng 2 chiều với một cột. Điều này đảm bảo rằng dữ liệu sẽ có định dạng phù hợp để tiếp tục sử dụng trong các vòng lặp tiếp theo của quá trình dự đoán.

```
# Chuyển đổi giá trị dự đoán về dạng ban đầu
predicted_prices = sc.inverse_transform(np.array(predicted_prices).reshape(-1, 1))
```

Chuyển đổi giá trị dự đoán từ dạng chuẩn hóa về đơn vị ban đầu của giá chứng khoán.

```
# Hiển thị kết quả dự đoán
plt.figure(figsize=(24, 8))
plt.plot(df1.index, df1['Lần cuối'], color='red', label='Giá thực tế')
plt.plot(future_dates, predicted_prices, color='blue', label='Giá dự đoán')
plt.title('Dự đoán giá cổ phiếu Vinamilk đến ngày 20/6/2024')
plt.xlabel('Thời gian')
plt.ylabel('Giá đóng cửa (VNĐ)')
plt.legend()
plt.show()
✓ 13.2s
```

Hiển thị kết quả dự đoán.

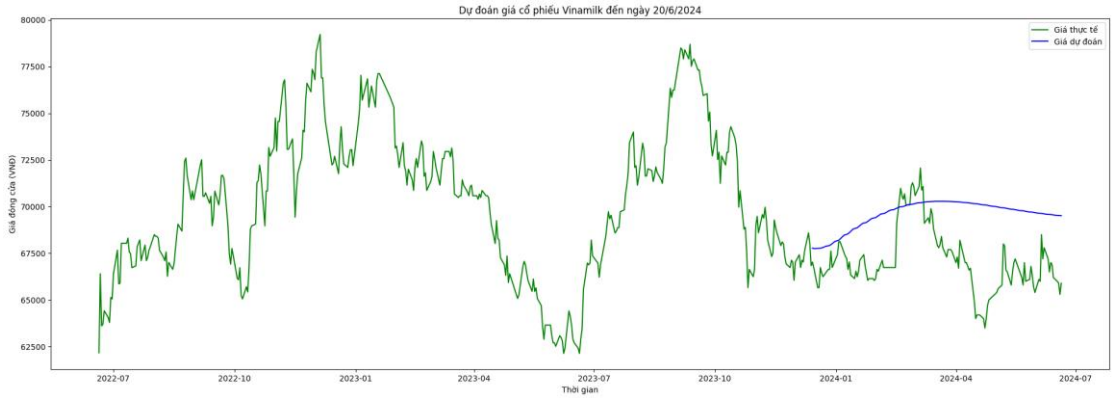



Kết quả dự đoán.

```
# Hiển thị kết quả dự đoán
plt.figure(figsize=(24, 8))
plt.plot(df1_full.index, df1_full['Lần cuối'], color='green', label='Giá thực tế')
plt.plot(future_dates, predicted_prices, color='blue', label='Giá dự đoán')
plt.title('Dự đoán giá cổ phiếu Vinamilk đến ngày 20/6/2024')
plt.xlabel('Thời gian')
plt.ylabel('Giá đóng cửa (VNĐ)')
plt.legend()
plt.show()
```

✓ 0.4s

So sánh kết quả dự đoán với dữ liệu thực tế.




Kết quả dự đoán so sánh với dữ liệu thực tế.