

# Investing

Dự đoán giá chứng khoán của công ty Apple bằng  
LSTM

Machine Learning



About the Financial Expert

**Hi there! I'm Trune,  
an investor at VJU  
Financial.**

I've been in finance for more than **15 days**.

I help money-savvy individuals manage their money and start investing.





# Đặt vấn đề

What we'll do today

01

Thị trường chứng khoán biến động thường xuyên và phụ thuộc vào rất nhiều yếu tố ngoại cảnh

02

Ngoài các yếu tố ngoại cảnh, sự ảnh hưởng của giá chứng khoán trong quá khứ cũng tác động rất nhiều đến tương lai

03

Tìm ra sự tác động này giúp một phần trong việc dự đoán giá chứng khoán

# Thu nhập dữ liệu



Dữ liệu được sử dụng: Kaggle  
Sử dụng thư viện pandas-datareader  
1.16.0

`pip install pandas-datareader`

**Đọc dữ liệu +**

`df = pd.read_csv('Apple_Stock_Prices.csv')`

	Date	Open	High	Low	Close	Adj Close	Volume
2005-02-01	2005-02-01	1.375893	1.388750	1.367500	1.384464	1.180142	678395200
2005-02-02	2005-02-02	1.391964	1.426964	1.387321	1.421964	1.212108	1020062400
2005-02-03	2005-02-03	1.412500	1.418393	1.380893	1.389464	1.184404	731651200
2005-02-04	2005-02-04	1.390536	1.409464	1.384464	1.407857	1.200083	563556000
2005-02-07	2005-02-07	1.409464	1.416964	1.383929	1.409643	1.201605	524456800
	...	...	...	...	...	...	...
2023-01-23	2023-01-23	138.119995	143.320007	137.899994	141.110001	141.110001	81760300
2023-01-24	2023-01-24	140.309998	143.160004	140.300003	142.529999	142.529999	66435100
2023-01-25	2023-01-25	140.889999	142.429993	138.809998	141.860001	141.860001	65799300
2023-01-26	2023-01-26	143.169998	144.250000	141.899994	143.960007	143.960007	54105100
2023-01-27	2023-01-27	143.160004	147.229996	143.080002	145.929993	145.929993	70492800

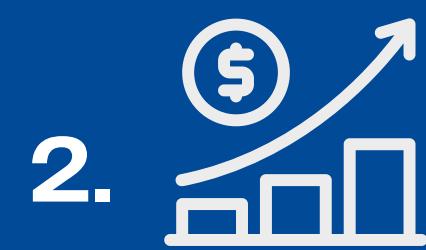
4529 rows × 6 columns

# Các dữ liệu



## 1. Open/close price

- Open price: Giá mở bán
- Close price: Giá cổ phiếu tại phiên giao dịch cuối cùng



## 2. High/low price

- High price: Giá cao nhất trong ngày
- Low price: Giá thấp nhất trong ngày



## 3. Adj Close price

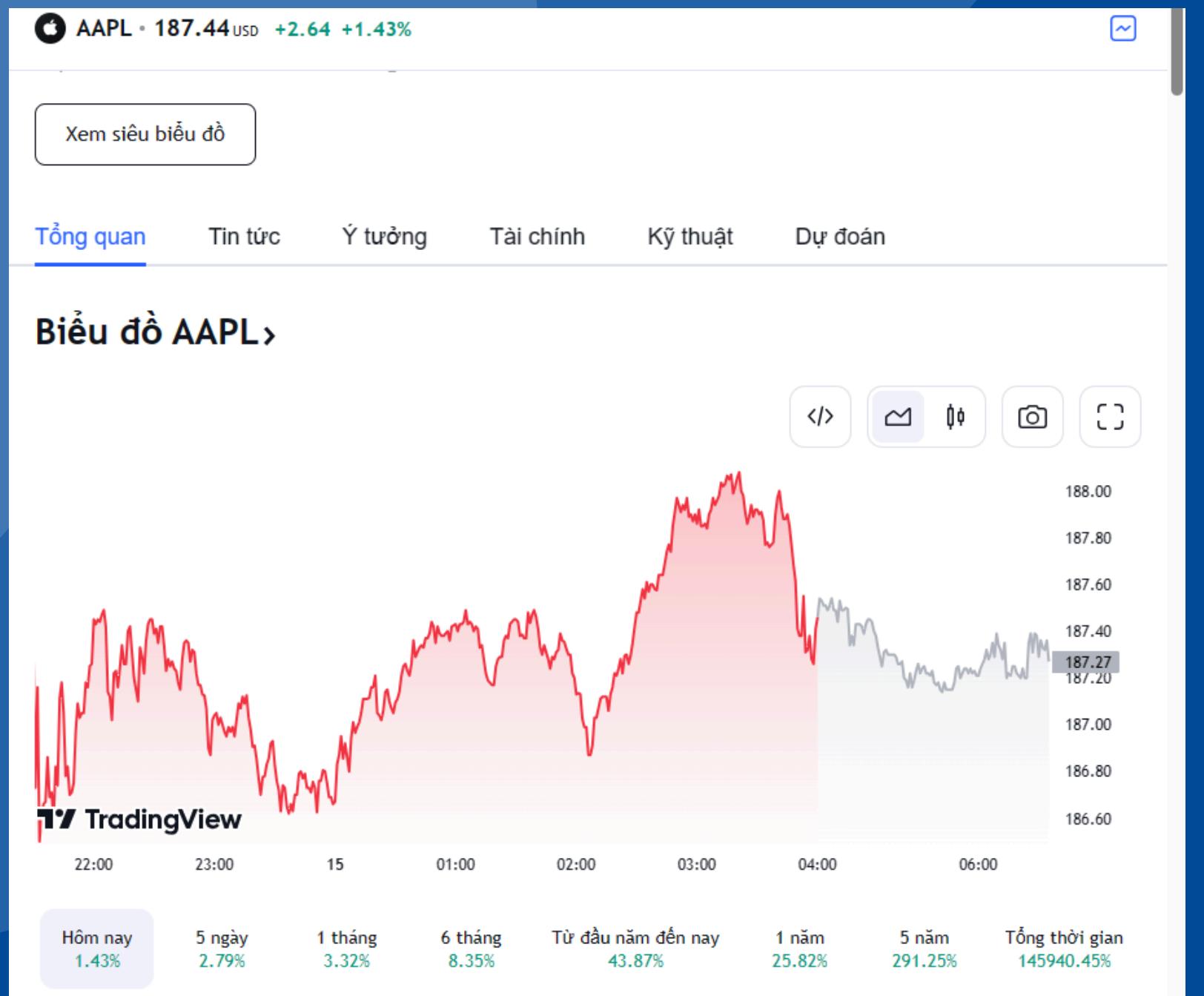
Giá điều chỉnh - được coi là giá thực sự của cổ phiếu, thường được dùng trong kiểm tra, phân tích chứng khoáng



## 4. Volume

Số lượng cổ phiếu dùng trong ngày

# Why choose?



## Câu hỏi

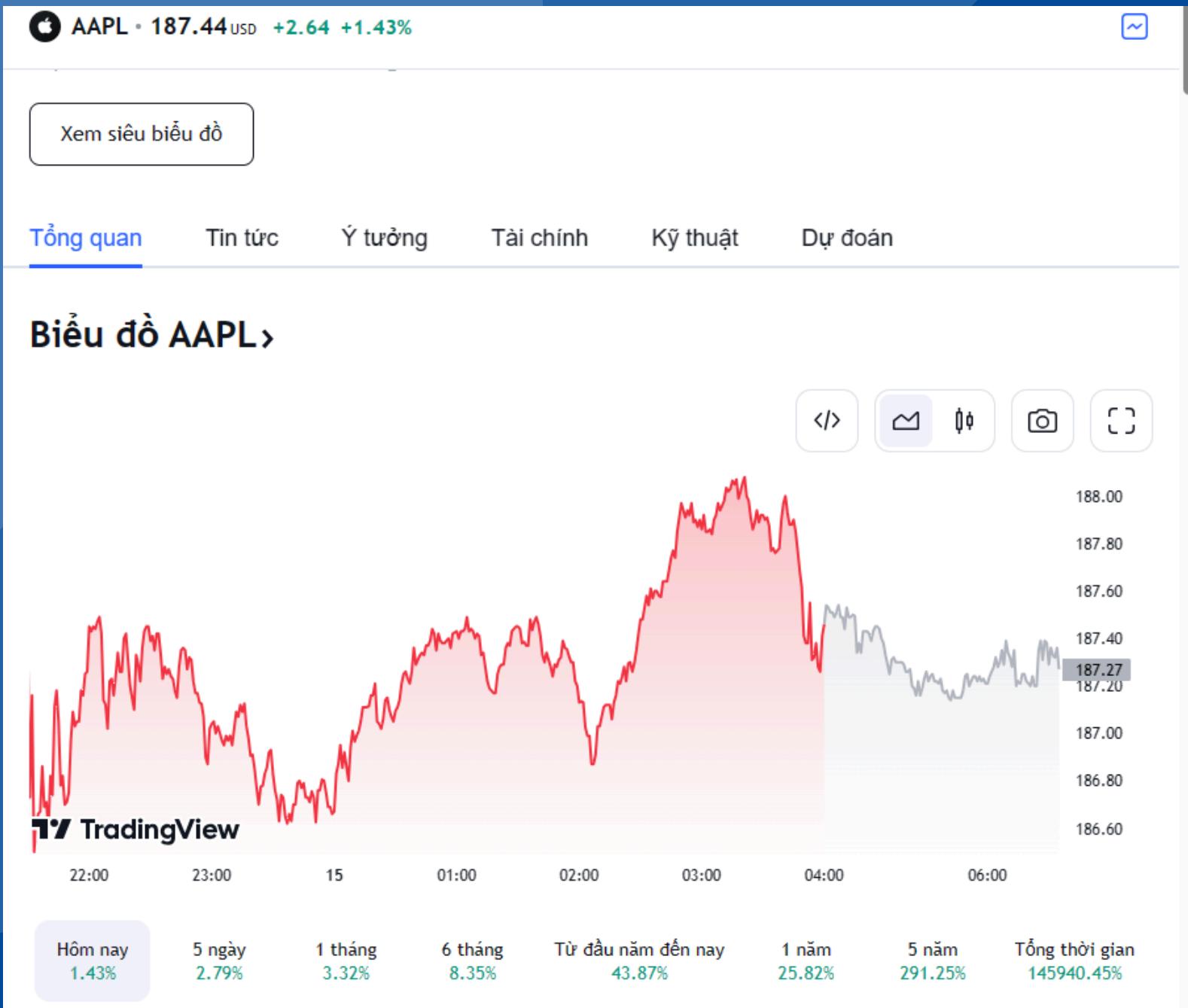
Thời điểm thích hợp để mua, bán cổ phiếu để thu lời về nhiều nhất?

## Solution:

- Mua vào lúc giá thấp
- Bán ra lúc giá cao

Cơ sở: Đồ thị biểu diễn giá cổ phiếu

# Why choose?



## Câu hỏi

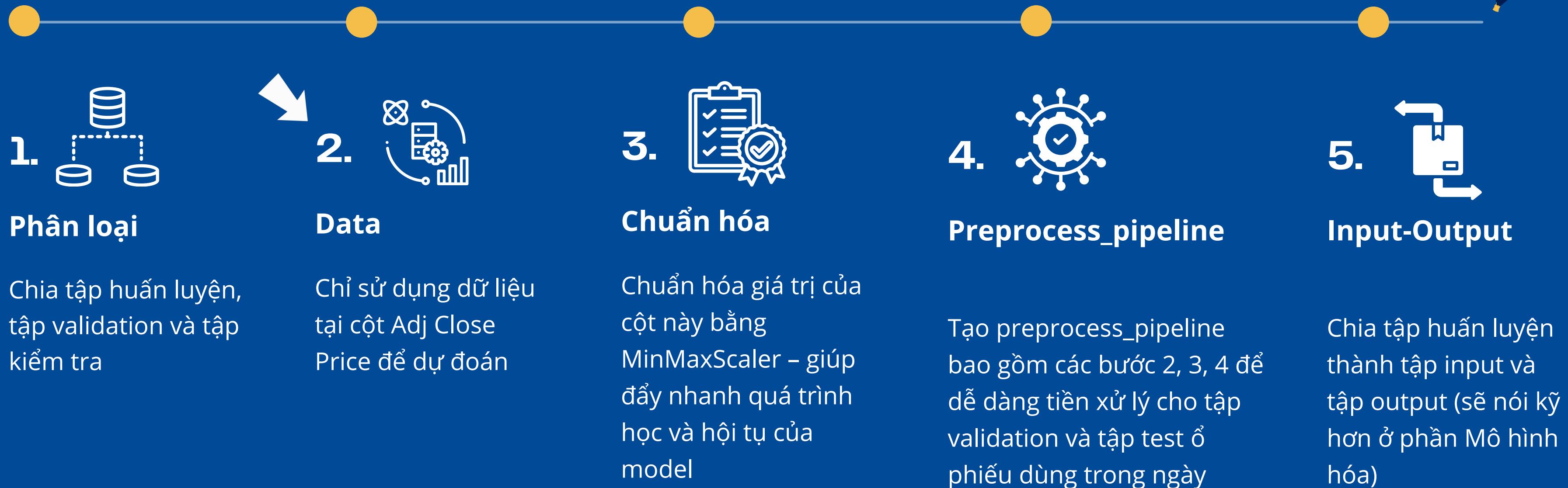
**Làm thế nào để biết được lúc nào giá thấp, lúc nào giá cao?**

**Dựa vào:**

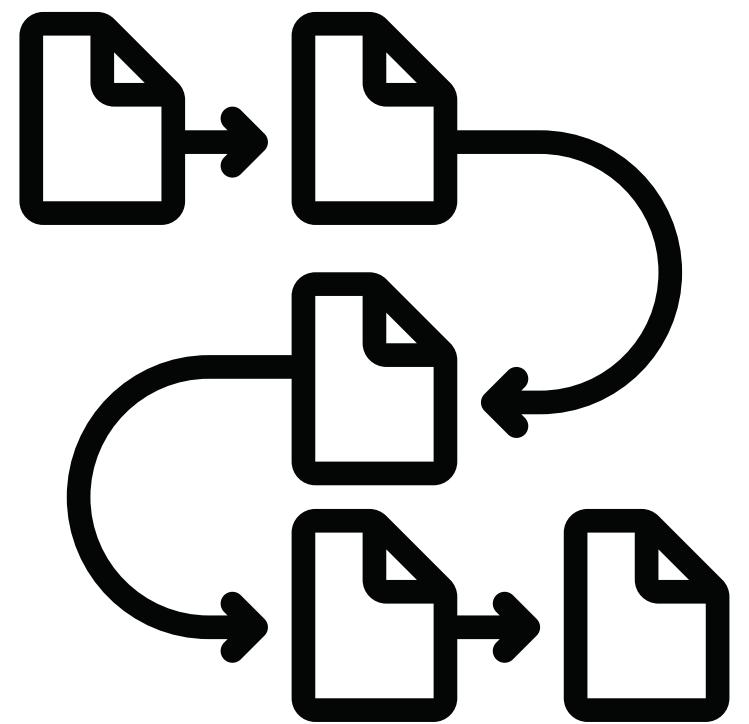
- Cảm tính
- **Ứng dụng LSTM trong việc dự đoán sự tăng giảm về giá cổ phiếu**

**Ý nghĩa:** Thực hiện đầu tư thành công là thực hiện thành công hành vi tạo ra giá trị thặng dư cá nhân. Giá trị thặng dư này mang lại nguồn động lực dồi dào trong việc phát triển bản thân và đất nước

# Tiền xử lí dữ liệu

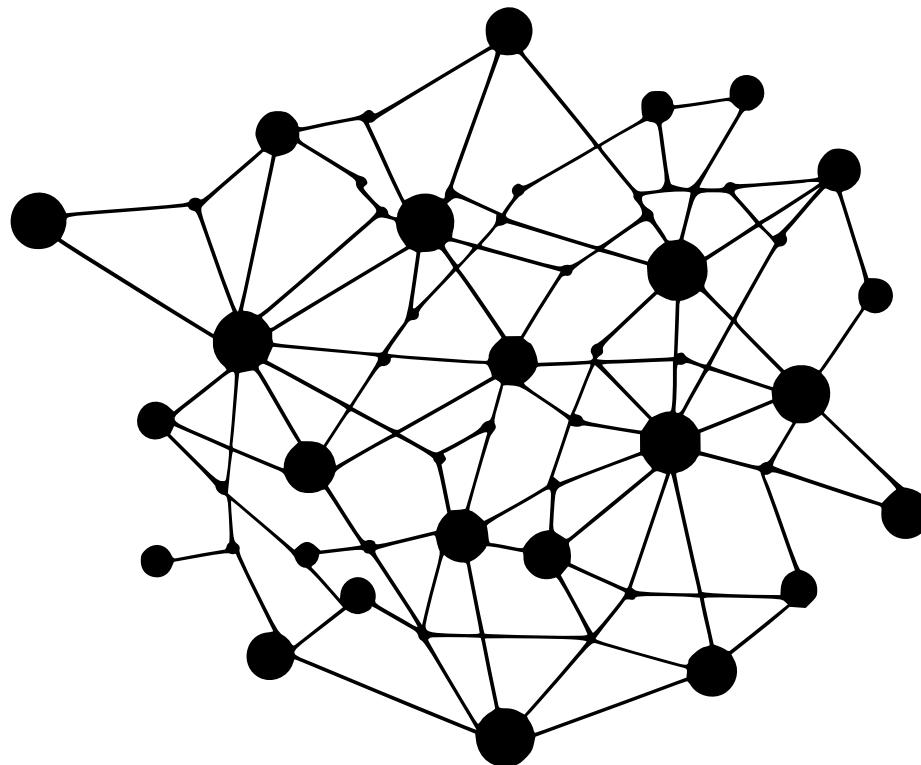


# Tại sao dùng LSTM?



## Tính tuần tự của tập dữ liệu

Tập dữ liệu mà trong đó, các mẫu dữ liệu xuất hiện theo một thứ tự (ví dụ như các từ trong 1 văn bản, các nốt nhạc trong 1 bản nhạc,...)



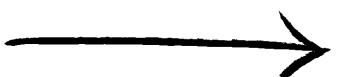
## Neural Network

Mạng NN truyền thống tách dữ liệu ra khỏi ngữ cảnh (các đầu vào độc lập với nhau), không tận dụng tối ưu sự có mặt của tính tuần tự trong tập dữ liệu.



## Ngữ nghĩa

Tính tuần tự mang trong nó 1 thông tin quan trọng ảnh hưởng đến “ngữ nghĩa” của dữ liệu. Cần phải tận dụng thông tin này



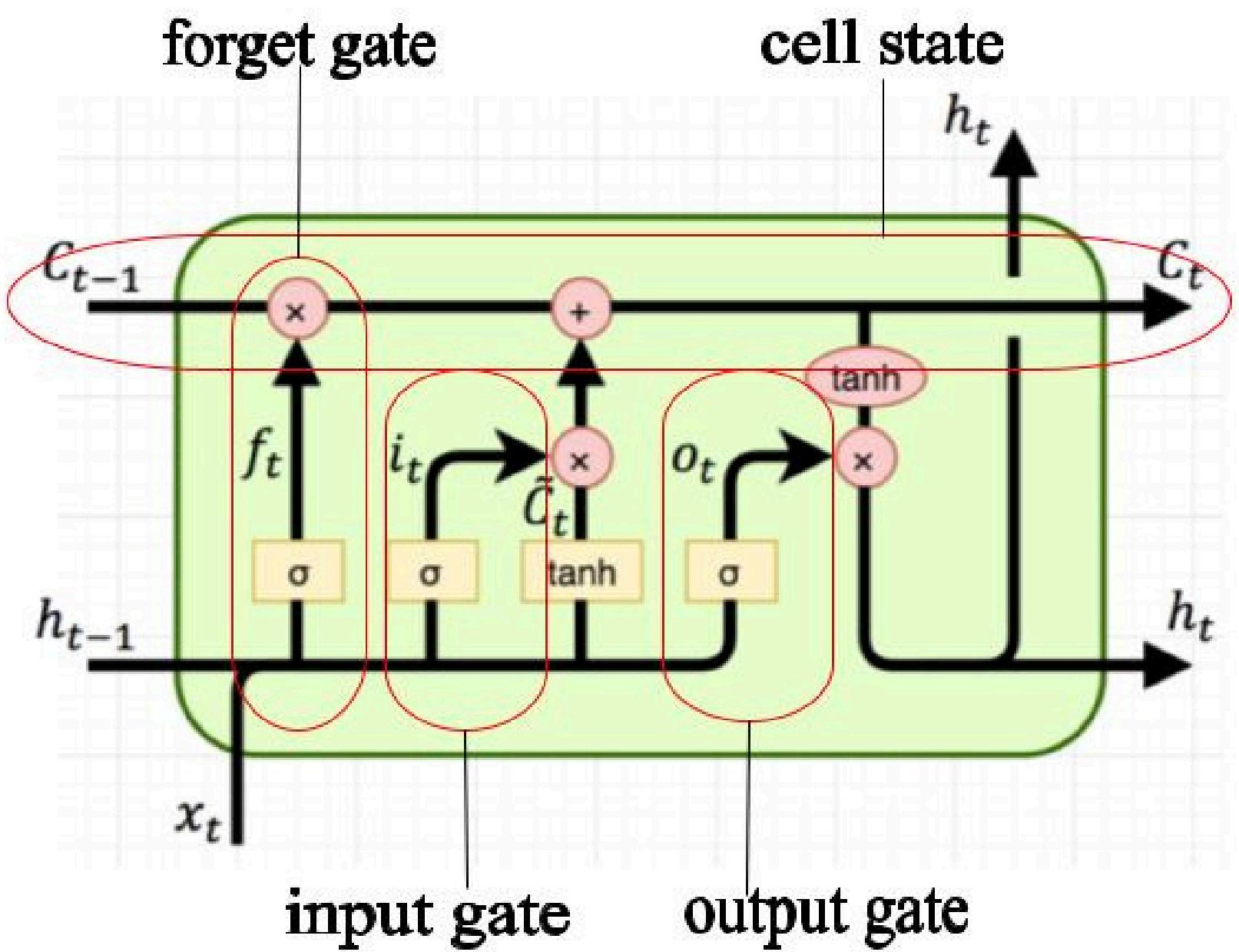
**LSTM được sinh ra để xử lý loại dữ liệu này**



# Investing Explained

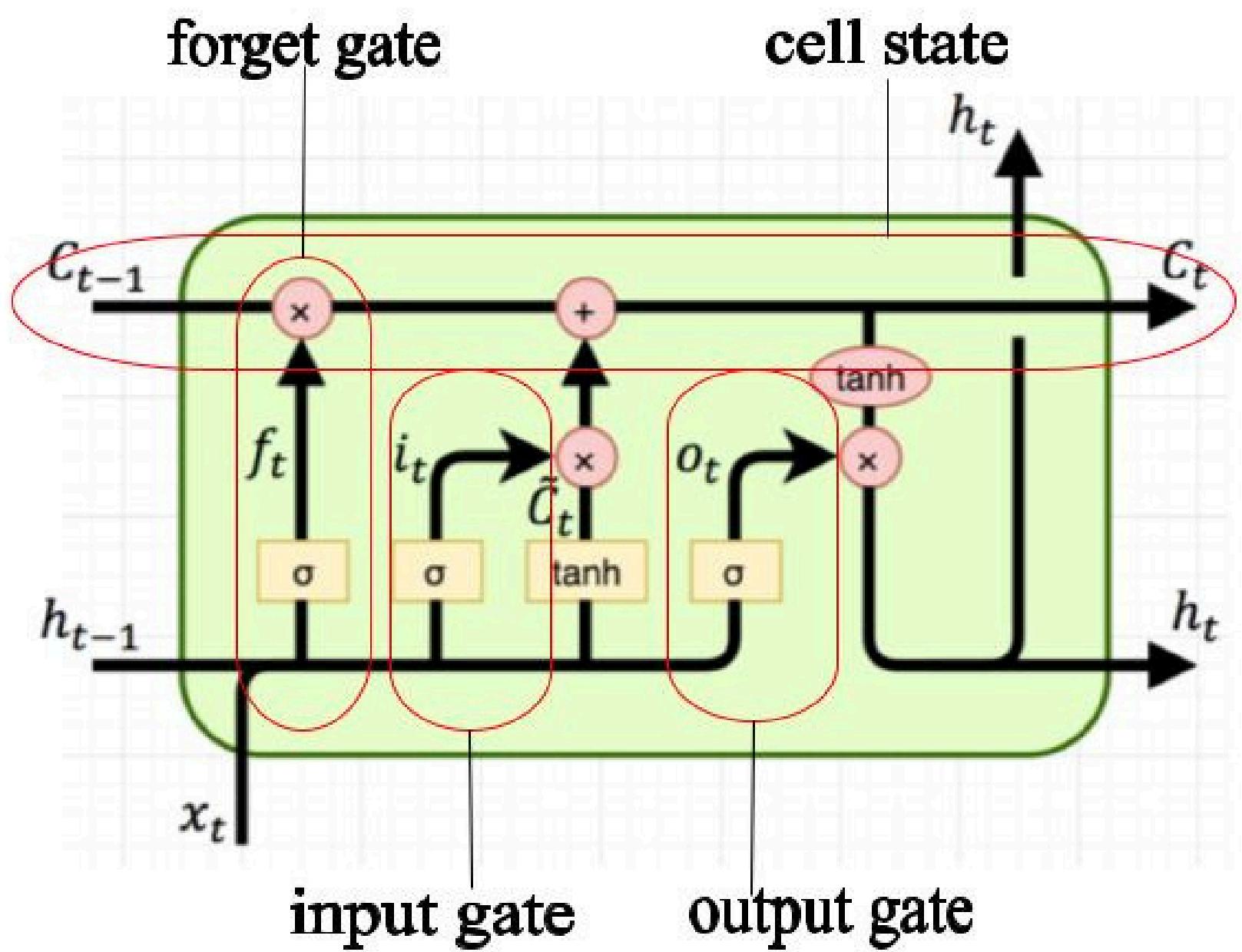
A brief guide to the basics of investing

# Mô hình hóa – Long Short Term Memory



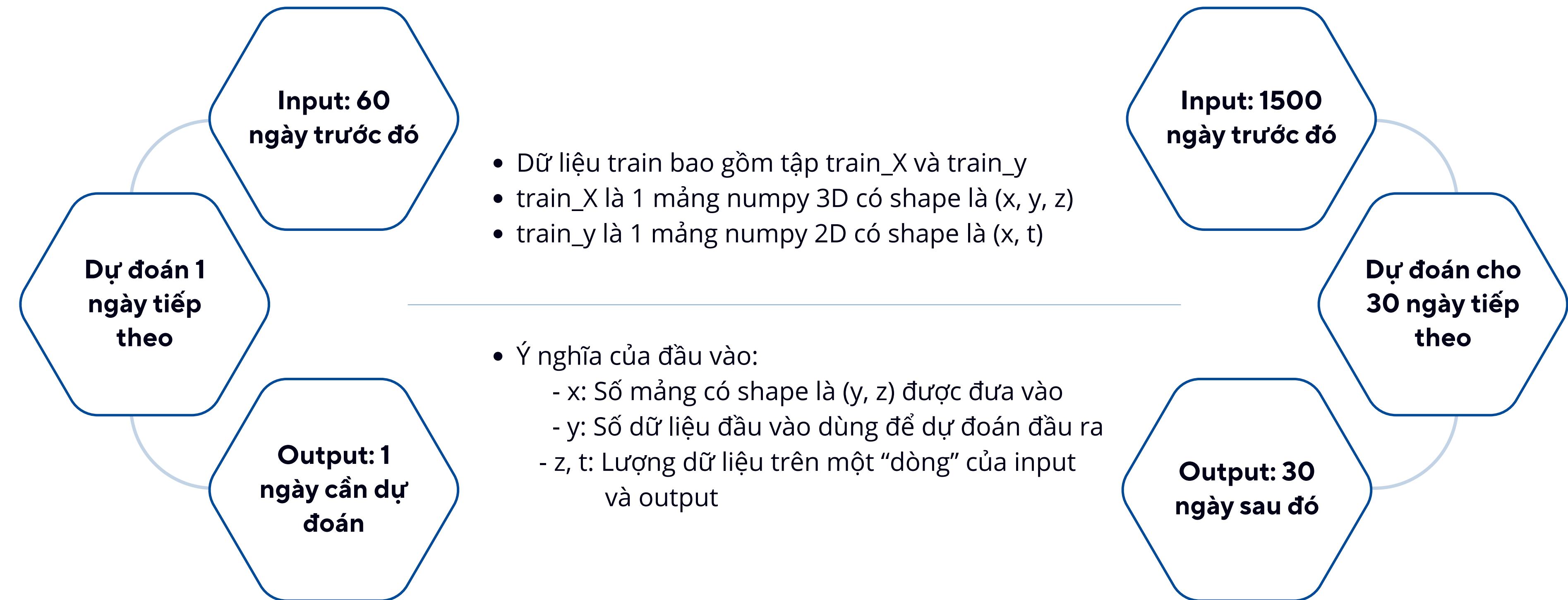
- Output:  $c_t, h_t$ , ta gọi  $c$  là cell state,  $h$  là hidden state.
- Input:  $c_{t-1}, h_{t-1}, x_t$ . Trong đó  $x_t$  là input ở state thứ  $t$  của model.  $c_{t-1}, h_{t-1}$  là output của layer trước.  $h$  đóng vai trò khá giống như  $s$  ở RNN, trong khi  $c$  là điểm mới của LSTM.
- Forget gate:  $f_t = \sigma(U_f * x_t + W_f * h_{t-1} + b_f)$
- Input gate:  $i_t = \sigma(U_i * x_t + W_i * h_{t-1} + b_i)$
- Output gate:  $o_t = \sigma(U_o * x_t + W_o * h_{t-1} + b_o)$
- Ma trận trọng số:  $W_f, W_i, W_o, W_c$   
 $W_f, W_i, W_o, W_c$   
 $b_f, b_i, b_o, b_c$

# Mô hình hóa – Long Short Term Memory



- **Cell state:** Đây là nơi một ô có thể lưu trữ thông tin (một phần liên quan đến thông tin được xử lý trước đó, một phần liên quan đến dữ liệu đầu vào tại mỗi ô).
- **Forget gate:** Quyết định thông tin nào sẽ bị loại bỏ hoặc giữ lại (đầu vào cho cổng này bao gồm dữ liệu đầu vào và thông tin từ ô trước đó).
- **Input gate:** Lọc thông tin quan trọng từ dữ liệu đầu vào và thông tin của ô trước đó, cập nhật thông tin này vào trạng thái ô.
- **Output gate:** Quyết định đầu vào cho ô tiếp theo bằng cách lọc thêm thông tin từ đầu vào, trạng thái ô và thông tin nhận được từ ô trước đó.

# Dữ liệu đầu vào của LSTM



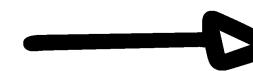
# Tối ưu trọng số của model

Giả sử dữ liệu đầu vào có shape (60, 1), đầu ra có shape (30, 1)

## Khi đó

- Các ma trận W phải có shape ( $x$ , 60) để có thể nhân được với input
- $h$  và  $c$  có shape là (30, 1) để có thể “nhân” với nhau (element wise multiplication)

## Tương tự với những cỗng còn lại



Thuật toán tối ưu? Khó

**Lúc này,  $f$ ,  $i$ ,  $c'$  sẽ có shape (30, 1)**

## Xét cỗng forget:

- $W_f$  phải có shape (30, 60) vì  $x$  có shape (60, 1)
- $U_f$  phải có shape (30, 30) vì  $h$  có shape (30, 1)
- $b_f$  phải có shape (30, 1)

## Ma trận trọng số cần phải tối ưu:

$W_f, W_i, W_o, W_c$

$W_f, W_i, W_o, W_c$

$b_f, b_i, b_o, b_c$

# Chia tập dữ liệu

Always think long-term.

```
# Chia tập dữ liệu để làm việc
def splitDataset(df):
    length = df.shape[0]
    sixtyPercent = length * 60 // 100
    eightyPercent = length * 80 // 100
    train_df = df[:sixtyPercent]
    val_df = df[sixtyPercent:eightyPercent]
    test_df = df[eightyPercent:]
    return train_df, val_df, test_df

train_df, val_df, test_df = splitDataset(df)
train_df.shape, val_df.shape, test_df.shape
```

**Investing regularly pays off in the long term.**

1. **Tính độ dài của DataFrame:** `length = df.shape[0]` tính số lượng hàng trong DataFrame `df`.
2. **Chia tỷ lệ phần trăm cho tập huấn luyện và tập validation:** Tại đây, được quy định rằng 60% dữ liệu sẽ được sử dụng cho tập huấn luyện (`sixtyPercent = length * 60 // 100`), và 20% dữ liệu tiếp theo sẽ được sử dụng cho tập validation (`eightyPercent = length * 80 // 100`). Phần còn lại sẽ được sử dụng cho tập kiểm tra.
3. **Tạo các DataFrame cho mỗi phần:** Dữ liệu được chia thành ba phần dựa trên tỷ lệ đã quy định ở bước trước. `train_df` chứa 60% dữ liệu đầu tiên, `val_df` chứa 20% dữ liệu tiếp theo và `test_df` chứa phần còn lại của dữ liệu.
4. **Trả về các DataFrame đã chia:** Cuối cùng, hàm trả về ba DataFrame: `train_df`, `val_df`, và `test_df`.

# Tiền xử lý & Preprocess\_pipeline

Always think long-term.

```
# Tiền xử lý tập train và tạo preprocess_pipeline
# tạo dataframe chỉ bao gồm cột index và cột Adj Close Price
def createDataFrame(df):
    return df.filter(['Adj Close'])

real_train_df = createDataFrame(train_df)
real_train_df.shape
```

**Investing regularly pays off in the long term.**

1. **Hàm createDataFrame(df):** Đây là một hàm được định nghĩa để tạo một DataFrame mới từ DataFrame đầu vào **df**.
2. **df.filter(['Adj Close']):** Dòng này sử dụng phương thức filter() của DataFrame để chọn chỉ cột '**Adj Close**'. Do đó, DataFrame mới sẽ chỉ chứa cột '**Adj Close**'.
3. **Trả về DataFrame mới:** Cuối cùng, hàm trả về DataFrame mới chỉ chứa cột '**Adj Close**'.
4. **Sử dụng hàm createDataFrame để tạo DataFrame cho tập huấn luyện:** DataFrame mới này được lưu vào biến **real\_train\_df**.
5. **real\_train\_df.shape:** Dòng này in ra kích thước của DataFrame **real\_train\_df** để kiểm tra số hàng và số cột của nó. Điều này giúp đảm bảo rằng tiến trình tiền xử lý đã hoạt động như mong đợi.

# Scale dữ liệu train

Always think long-term.

```
# scale dữ liệu train
scaler = MinMaxScaler()
scaled_train_data = scaler.fit_transform(real_train_df)

def reshapeForModel(scaled_train_data):
    return scaled_train_data.reshape((-1,))

scaled_train_data = reshapeForModel(scaled_train_data)
scaled_train_data.shape
```

**Investing regularly pays off in the long term.**

## 1. Scale dữ liệu huấn luyện:

- Đầu tiên, chúng ta tạo một trình chuyển đổi MinMaxScaler() để thực hiện việc scale dữ liệu.
- Tiếp theo, chúng ta sử dụng phương thức fit\_transform() của MinMaxScaler() để scale dữ liệu huấn luyện được chứa trong real\_train\_df. Dữ liệu được scale để nằm trong khoảng giá trị [0, 1], giúp mô hình học được hiệu quả hơn.
- Dữ liệu được gán lại cho biến scaled\_train\_data.

## 2. Reshape dữ liệu cho mô hình LSTM:

- Chúng ta cần định hình lại dữ liệu để phù hợp với đầu vào của mô hình LSTM. Mô hình LSTM yêu cầu đầu vào có ba chiều: số lượng mẫu, số lượng thời điểm (time steps), và số lượng đặc trưng (features).
- Trong trường hợp này, chúng ta chỉ có một đặc trưng (giá cổ phiếu điều chỉnh), do đó số lượng đặc trưng là 1.
- Phương thức reshapeForModel() được sử dụng để định hình lại dữ liệu thành dạng phù hợp với mô hình LSTM, bằng cách sử dụng -1 cho số lượng mẫu (số lượng hàng) và 1 cho số lượng đặc trưng.
- Kết quả sau khi định hình lại dữ liệu sẽ là một mảng một chiều.

**Investing regularly pays off in the long term.**

# Tạo dữ liệu

Always think long-term.

```
# tạo dữ liệu train_X và train_y
def createTrainX_y(scaled_train_data, time_steps=60, out_length=1):
    train_X = []
    train_y = []
    for i in range(len(scaled_train_data) - time_steps - out_length + 1):
        end_X = i + time_steps
        end_y = end_X + out_length
        train_X.append(scaled_train_data[i:end_X])
        train_y.append(scaled_train_data[end_X:end_y])

    train_X, train_y = np.array(train_X), np.array(train_y)
    return train_X.reshape((train_X.shape[0], train_X.shape[1], 1)), train_y

train_X, train_y = createTrainX_y(scaled_train_data)
train_X.shape, train_y.shape
```

## 1. Hàm createTrainX\_y:

- Đây là một hàm được định nghĩa để tạo ra dữ liệu huấn luyện từ dữ liệu đã được chuẩn hóa (scaled\_train\_data).
- Tham số time\_steps xác định số lượng bước thời gian quan sát trong quá khứ mà mô hình sẽ sử dụng để dự đoán. Trong trường hợp này, time\_steps được đặt là 60.
- Tham số out\_length là số lượng bước thời gian mà mô hình sẽ dự đoán tiếp theo. Trong trường hợp này, out\_length được đặt là 1, nghĩa là mô hình sẽ dự đoán giá trị tiếp theo sau 60 bước thời gian.
- Hàm này tạo ra các mẫu dữ liệu huấn luyện train\_X và train\_y, trong đó:
- train\_X là các mảng 3 chiều chứa các mẫu dữ liệu đầu vào (features) cho mô hình LSTM. Mỗi mẫu có kích thước (time\_steps, 1), trong đó 1 là số lượng biến độc lập (trong trường hợp này, chỉ có giá đóng cửa của cổ phiếu).
- train\_y là các mảng 2 chiều chứa các mẫu dữ liệu đầu ra (target) cho mô hình LSTM. Mỗi mẫu có kích thước (out\_length, 1), tức là mỗi mẫu chỉ chứa một giá trị dự đoán tiếp theo sau time\_steps bước thời gian.

## 2. Gọi hàm createTrainX\_y:

- Chúng ta gọi hàm createTrainX\_y với dữ liệu đã được chuẩn hóa scaled\_train\_data để tạo ra train\_X và train\_y.
- Kết quả trả về là train\_X và train\_y, là các mảng dữ liệu được chuẩn bị sẵn để huấn luyện mô hình LSTM.
- Cuối cùng, chúng ta kiểm tra kích thước của train\_X và train\_y để đảm bảo chúng có đúng kích thước cho quá trình huấn luyện.

# Pipeline

Always think long-term.

```
preprocess_pipeline = Pipeline([
    ('dropper', FunctionTransformer(createDataFrame)),
    ('scaler', MinMaxScaler()),
    ('reshape', FunctionTransformer(reshapeForModel)),
    ('createTrain', FunctionTransformer(createTrainX_y)),
])

train_X, train_y = preprocess_pipeline.fit_transform(train_df)
train_X.shape, train_y.shape
```

**Pipeline tiền xử lý dữ liệu để tự động hóa quá trình tiền xử lý từ dữ liệu gốc tới dữ liệu huấn luyện cho mô hình LSTM**

## 1. **Bước 1: 'dropper' - Loại bỏ các cột không cần thiết:**

- Chúng ta sử dụng FunctionTransformer để áp dụng hàm createDataFrame lên dữ liệu đầu vào để tạo một DataFrame chỉ chứa cột 'Adj Close'.
- Điều này sẽ loại bỏ các cột không cần thiết khác và chỉ giữ lại cột 'Adj Close'.

## 2. **Bước 2: 'scaler' - Scale dữ liệu:**

- Trong bước này, chúng ta sử dụng MinMaxScaler() để scale dữ liệu cột 'Adj Close' trong DataFrame.

## 3. **Bước 3: 'reshape' - Định hình lại dữ liệu cho mô hình LSTM:**

- Chúng ta sử dụng FunctionTransformer để áp dụng hàm reshapeForModel lên dữ liệu đã được scale để định hình lại cho phù hợp với đầu vào của mô hình LSTM.

## 4. **Bước 4: 'createTrain' - Tạo dữ liệu huấn luyện:**

- Trong bước này, chúng ta sử dụng FunctionTransformer để áp dụng hàm createTrainX\_y lên dữ liệu đã được định hình lại để tạo ra các cặp dữ liệu huấn luyện train\_X và train\_y.

# Tạo model

Always think long-term.

```
# Tạo model
def createModel(train_X, time_steps=60, out_length=1):
    model = Sequential()
    model.add(LSTM(60, activation='tanh', return_sequences=False, input_shape=(train_X.shape[1], 1)))
    model.add(Dense(out_length))
    model.compile(optimizer='adam', loss='mse')

    return model
```

```
# xử lý tập validation
# tạo tập inputs
val_inputs = df[len(df) - len(val_df) - 60:]
val_X, val_y = preprocess_pipeline.transform(val_inputs)
val_X.shape, val_y.shape
```

Sau đó, chúng ta tiến hành xử lý tập validation để chuẩn bị dữ liệu đầu vào cho việc đánh giá mô hình. Dữ liệu validation được chuẩn bị bằng cách sử dụng pipeline đã được định nghĩa trước đó (`preprocess_pipeline.transform()`) để scale và định hình lại dữ liệu dựa trên `val_inputs`, sau đó kích thước của `val_X` và `val_y` được in ra để kiểm tra.

## 1. Hàm `createModel(train_X, time_steps=60, out_length=1)`:

- Đây là một hàm được định nghĩa để tạo một mô hình mạng nơ-ron sử dụng kiến trúc LSTM.
- Tham số `train_X` được sử dụng để xác định kích thước của đầu vào mô hình.
- `time_steps` là số lượng bước thời gian mà mô hình sẽ xem xét khi dự đoán mỗi điểm dữ liệu.
- `out_length` là số lượng điểm dữ liệu mà mô hình sẽ dự đoán cho mỗi lần dự đoán.

## 2. Thêm các layer vào mô hình:

- Chúng ta sử dụng `Sequential()` để khởi tạo một mô hình tuần tự.
- Sau đó, chúng ta thêm một layer LSTM vào mô hình với các đặc điểm sau:
  - 60 units: Số lượng units trong layer LSTM.
  - activation='tanh': Hàm kích hoạt được sử dụng là tanh.
  - return\_sequences=False: Chỉ cần output ở layer cuối cùng của LSTM.
  - input\_shape=(train\_X.shape[1], 1): Định dạng đầu vào cho layer LSTM, với số lượng thời điểm và số lượng đặc trưng.
- Tiếp theo, chúng ta thêm một layer Dense với số lượng units là `out_length`, là layer đầu ra của mô hình.

## 3. Compile mô hình:

- Chúng ta sử dụng phương thức `compile()` để biên dịch mô hình với các tham số sau:
  - optimizer='adam': Sử dụng thuật toán tối ưu Adam.
  - loss='mse': Sử dụng hàm mất mát là Mean Squared Error (MSE) cho bài toán dự đoán giá trị liên tục.

## 4. Trả về mô hình đã tạo:

- Cuối cùng, hàm trả về mô hình mà chúng ta đã tạo.

# Thử nhiều tham số epochs

Always think long-term.

```
# thử nhiều tham số epochs để tìm ra model tốt nhất
epochs_ = [1,2,3,4,5,6]
best_val_err = float('inf')
best_epoch = None

for e in epochs_:
    # build model
    model = createModel(train_X)
    model.fit(train_X, train_y, batch_size=1, epochs=e)

    # dự đoán tập inputs
    pred_val = model.predict(val_X)
    pred_val = preprocess_pipeline.inverse_transform(pred_val)

    val_df['Predict'] = pred_val
    mse = np.sqrt(np.mean((val_df['Predict'] - val_df['Adj Close']) ** 2)))
    if mse < best_val_err:
        best_val_err = mse
        best_epoch = e
        print(e, mse)
```

## 1. Biến epochs\_:

- Biến này chứa danh sách các giá trị epochs mà chúng ta muốn thử nghiệm để tìm ra mô hình tốt nhất. Trong trường hợp này, chúng ta thử nghiệm các giá trị từ 1 đến 6.

## 2. Biến best\_val\_err và best\_epoch:

- **best\_val\_err**: Biến này được sử dụng để lưu giữ giá trị lỗi tốt nhất mà chúng ta đã tìm thấy trong quá trình thử nghiệm.
- **best\_epoch**: Biến này lưu trữ số lượng epochs tương ứng với giá trị lỗi tốt nhất.

## 3. Vòng lặp for:

- Trong vòng lặp này, chúng ta lặp qua từng giá trị của biến **epochs\_**.
- Đối với mỗi giá trị của **epochs\_**, chúng ta xây dựng một mô hình mới bằng cách sử dụng hàm **createModel(train\_X)** và sau đó huấn luyện mô hình trên tập huấn luyện (**train\_X, train\_y**) với số lượng epochs tương ứng.
- Sau khi huấn luyện xong, chúng ta sử dụng mô hình để dự đoán trên tập validation (**val\_X**) và chuyển ngược lại để có giá trị dự đoán ban đầu bằng cách sử dụng **preprocess\_pipeline.inverse\_transform()**.
- Sau đó, chúng ta tính toán lỗi trung bình bình phương (MSE) giữa giá trị dự đoán và giá trị thực tế trên tập validation.
- Nếu MSE này tốt hơn MSE tốt nhất đã tìm thấy trước đó (**best\_val\_err**), chúng ta cập nhật **best\_val\_err** và **best\_epoch** với giá trị mới và in ra giá trị MSE và số lượng epochs tương ứng.

# Fit model với tập train và predict

Always think long-term.

```
# fit model với tập train và predict tập test
# tạo dữ liệu train để fit (bao gồm train_df và valid_df)
train_inputs = df[:len(df) - len(test_df) - 60]
train_X, train_y = preprocess_pipeline.fit_transform(train_inputs)

# build model
model = createModel(train_X)
model.fit(train_X, train_y, batch_size=1, epochs=best_epoch)
```

## 1. Tạo dữ liệu huấn luyện (train\_inputs) từ dữ liệu gốc:

- Chúng ta bắt đầu bằng việc chọn một phần của dữ liệu gốc để tạo tập dữ liệu huấn luyện. Đoạn mã này chọn ra các dòng từ đầu tới một số lượng dòng nhất định trước khi bắt đầu tập dữ liệu kiểm tra (**test\_df**) và thêm 60 dòng nữa. Điều này nhằm mục đích làm cho mô hình có đủ dữ liệu trong quá trình huấn luyện và dự đoán.

## 2. Chuẩn bị dữ liệu huấn luyện (train\_X, train\_y) sử dụng preprocess\_pipeline:

- Chúng ta sử dụng **preprocess\_pipeline.fit\_transform()** để tiến hành tiền xử lý dữ liệu huấn luyện từ **train\_inputs**, bao gồm cả tạo ra **train\_X** và **train\_y** từ dữ liệu huấn luyện.

## 3. Xây dựng mô hình:

- Sau khi tiền xử lý dữ liệu huấn luyện, chúng ta xây dựng một mô hình LSTM bằng cách sử dụng hàm **createModel(train\_X)** đã được định nghĩa trước đó. Điều này sẽ tạo ra một mô hình dự đoán dựa trên cấu trúc của dữ liệu huấn luyện.

## 4. Huấn luyện mô hình:

- Chúng ta sử dụng phương thức **fit()** để huấn luyện mô hình trên dữ liệu huấn luyện (**train\_X, train\_y**).
- Trong quá trình huấn luyện, chúng ta sử dụng số lượng epochs tốt nhất đã được tìm thấy trước đó (**best\_epoch**). Điều này đảm bảo rằng chúng ta sử dụng số lượng epochs phù hợp nhất để huấn luyện mô hình.

# Dự đoán và chuẩn hóa ngược

Always think long-term.

```
# tạo dữ liệu test
test_inputs = df[len(df) - len(test_df) - 60:]
test_X, test_y = preprocess_pipeline.transform(test_inputs)

# dự đoán tập inputs và chuẩn hóa ngược Lại về dạng giá ban đầu
pred_test = model.predict(test_X)
pred_test = preprocess_pipeline.inverse_transform(pred_test)
```

```
# dự đoán tập inputs và chuẩn hóa ngược Lại về dạng giá ban đầu
pred_test = model.predict(test_X)
pred_test = preprocess_pipeline.inverse_transform(pred_test)
pred_test.shape
```

## 1. Chuẩn bị dữ liệu kiểm tra (test\_inputs) từ dữ liệu gốc:

- Chúng ta bắt đầu bằng cách chọn một phần của dữ liệu gốc để tạo ra tập dữ liệu kiểm tra. Đoạn mã này chọn ra các dòng từ `len(df) - len(test_df) - 60` đến cuối cùng của dữ liệu gốc, và thêm 60 dòng nữa. Điều này giúp chúng ta có đủ dữ liệu để dự đoán trên tập kiểm tra.

## 2. Chuẩn bị dữ liệu kiểm tra (test\_X, test\_y) sử dụng preprocess\_pipeline:

- Chúng ta sử dụng `preprocess_pipeline.transform()` để tiến hành tiền xử lý dữ liệu kiểm tra từ `test_inputs`, tạo ra `test_X` và `test_y`. Điều này bao gồm việc chia dữ liệu thành các mẫu có kích thước phù hợp cho mô hình LSTM.

## 3. Dự đoán trên tập dữ liệu kiểm tra và chuẩn hóa ngược lại về dạng giá ban đầu:

- Chúng ta sử dụng mô hình đã huấn luyện để dự đoán giá trị của `test_X`, thu được `pred_test`.
- Tiếp theo, chúng ta sử dụng `preprocess_pipeline.inverse_transform()` để chuyển đổi ngược lại từ dạng dự đoán đã chuẩn hóa về dạng giá ban đầu của cổ phiếu.
- Kết quả cuối cùng là `pred_test`, là giá trị dự đoán của cổ phiếu trên tập dữ liệu kiểm tra, đã được chuyển đổi về đơn vị giá ban đầu.

# MSE

Always think long-term.

```
mse = np.sqrt(np.mean((test['Adj Close'] - test['Predictions']) ** 2))
print(mse)
```

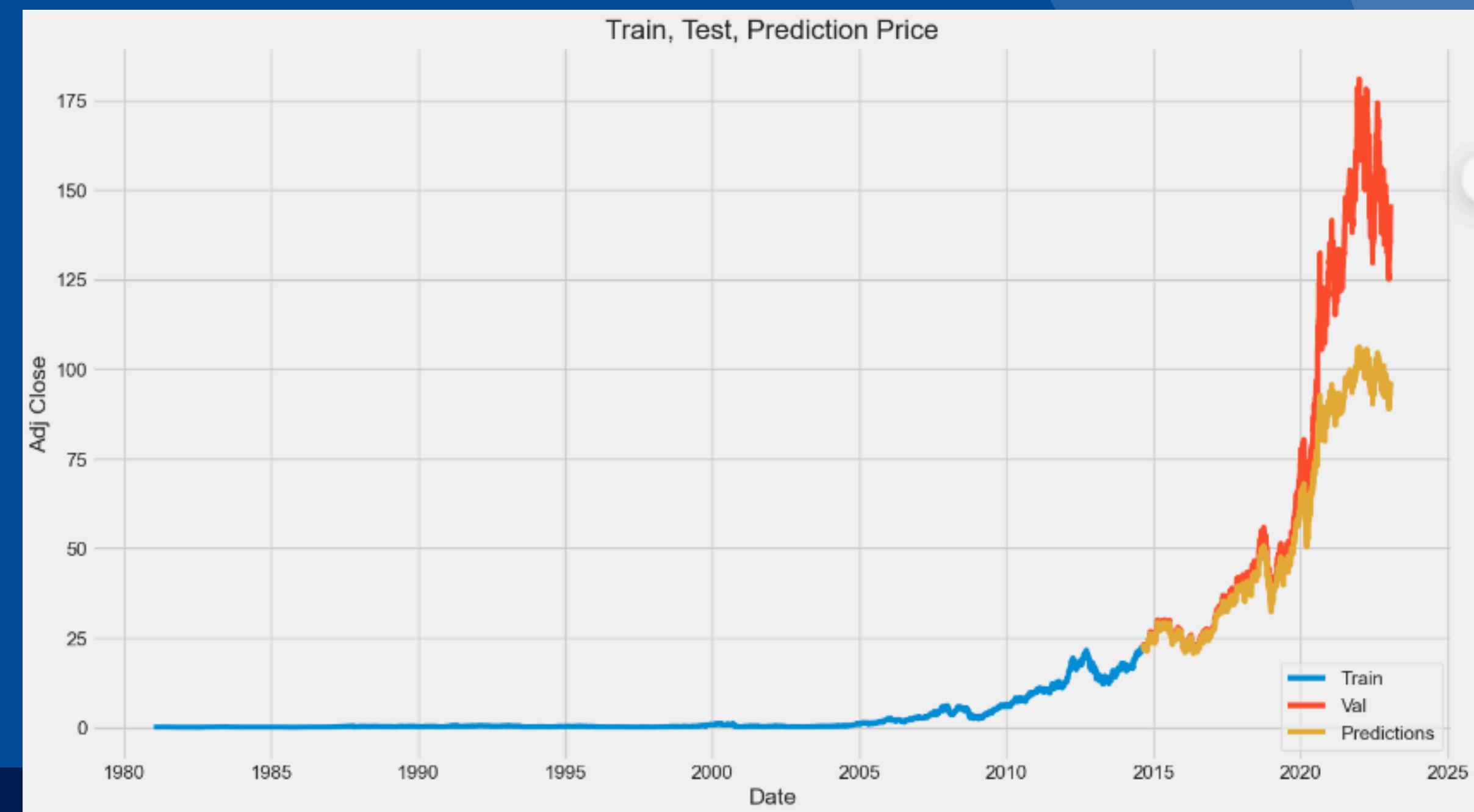
## Investing regularly pays off in the long term.

Đoạn mã này tính toán sai số trung bình bình phương (Mean Squared Error - MSE) giữa giá đóng cửa thực tế (Adj Close) và giá đóng cửa được dự đoán bởi mô hình (Predictions) trên tập dữ liệu kiểm tra. MSE là một phép đo đánh giá hiệu suất của mô hình dự đoán, và nó được tính bằng cách lấy trung bình của bình phương của sai số (khác biệt) giữa dự đoán và giá trị thực tế.



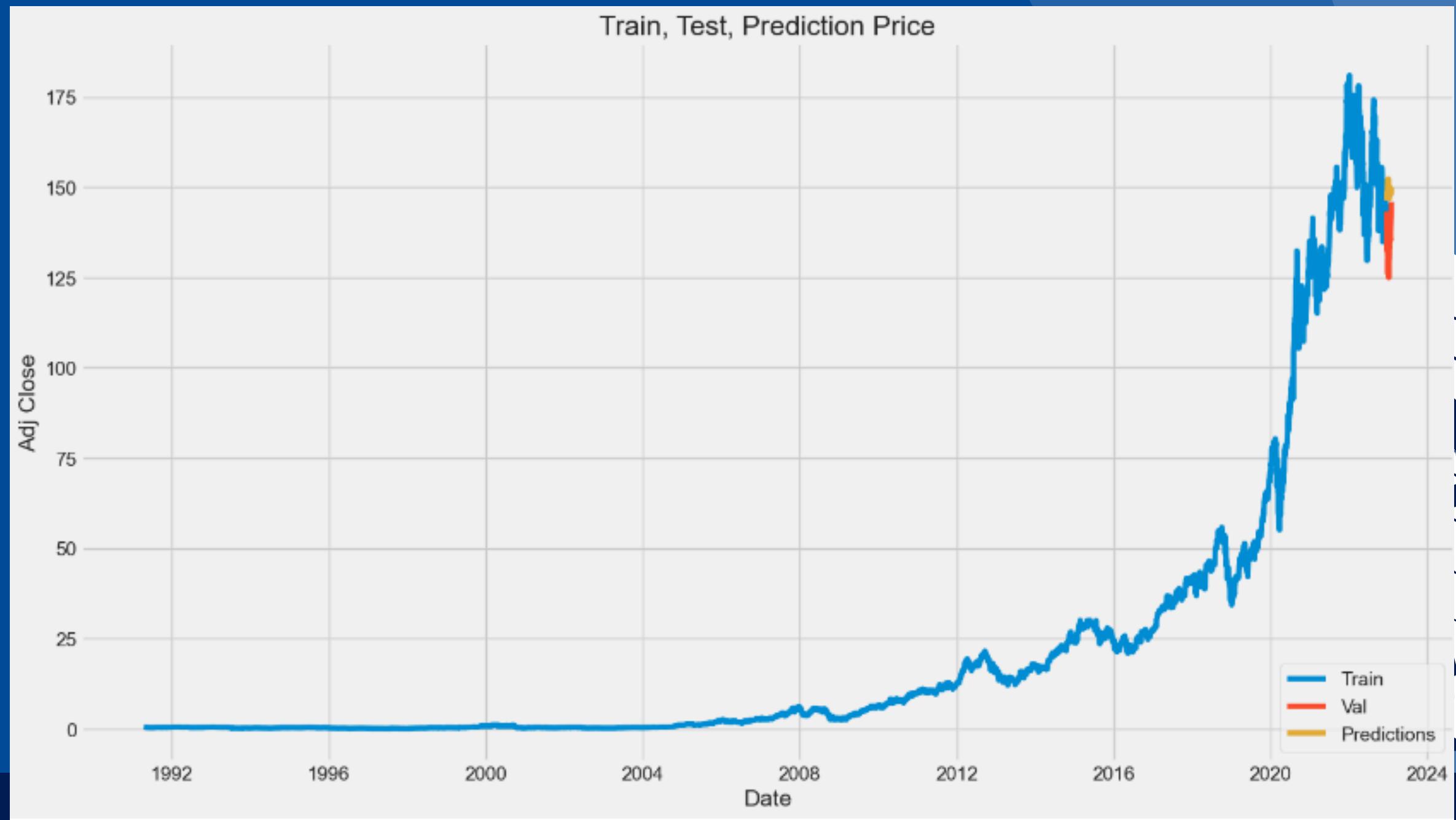
# Know Your Risk

Dự đoán 1 ngày với 2122 giá trị test  
MSE = 26.652150430735873



# Know Your Risk

Dự đoán 30 ngày với 30 giá trị test  
MSE = 16.084394047657305



# Khó khăn gấp phải huhu...

What every investor needs to know

01

Hậu xử lý dữ liệu

02

Quá trình tìm hiểu cơ sở tối ưu độ lỗi  
của model

03

Quá trình tìm hiểu cơ sở toán của  
model



# Contact

Make your money work for you  
with Sitwell Financial

## Phone Number

+84 37 3104 304

## Website

<https://github.com/Trungnef/Machine-Learning>

## Email Address

21110108@st.vju.ac.vn



# Resource Page

Use these icons and illustrations in your Canva Presentation. Happy designing!



# Resource Page

Use these icons and illustrations in your Canva Presentation. Happy designing!



# Resource Page

Find the magic and fun in presenting with Canva Presentations. Press the following keys while on Present mode!

**B** for blur

**C** for confetti

**D** for a drumroll

**O** for bubbles

**Q** for quiet

**X** to close

Any number from **0–9** for a timer