```
#Lê Hoài Linh
from pandas import read_csv, DataFrame
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
from tensorflow.keras.callbacks import History
from tensorflow.keras.optimizers import Adam, SGD
from matplotlib import pyplot
from numpy import array
from google.colab.drive import mount
from numpy import concatenate



from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df = read_csv('/content/drive/MyDrive/CPI.csv', index_col=0)
df.head()
# Đọc dữ liệu từ file CSV và chỉ chọn các cột từ 'C1' đến 'FX1'
```

Chỉ tiêu	Đơn vị tính	Tháng 1/2009	Tháng 2/2009	Tháng 3/2009	Tháng 4/2009	Tháng 5/2009	Tháng 6/2009	Tháng 7/2009	Tháng 8/2009	Tháng 9/2009	...	Tháng 2/2023	Tháng 3/2023	Tháng 4/2023	Tháng 5/2023	Tháng 6/2023	Thá 7/2(
Chỉ số giá tiêu dùng	%	0.32	1.17	-0.17	0.35	0.44	0.55	0.52	0.24	0.62	...	0.45	-0.23	-0.34	0.01	0.27	0
Hàng ăn và dịch vụ ăn uống	%	0.39	1.67	-0.46	0.43	0.18	0.28	-0.05	-0.08	0.05	...	-0.17	-0.58	-0.38	0.24	0.57	0
Lương thực	%	-0.04	0.82	1.27	0.03	-0.37	-1.10	-0.92	-0.42	-0.85	...	0.26	0.28	0.30	0.29	0.09	0
Thực phẩm	%	0.55	1.72	-1.55	0.46	0.36	0.67	-0.05	-0.09	0.17	...	-0.49	-1.00	-0.71	0.22	0.72	0
Ăn uống ngoài	%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.77		0.45	0.11	0.13	0.26	0.42	0

```
import pandas as pd

selected_columns = df.columns[df.columns.get_loc('Tháng 1/2009'): df.columns.get_loc('Tháng 10/2023') + 1]

# Lấy giá trị của dòng đầu tiên trong mỗi cột
date_values = df[selected_columns].iloc[0].values

# Tạo DataFrame mới với 2 cột Date và CPI
df_new = pd.DataFrame({'Date': selected_columns, 'CPI': date_values})

# Kiểm tra và chuyển đổi cột "CPI" sang kiểu số
df_new['CPI'] = pd.to_numeric(df_new['CPI'], errors='coerce')

# Nhân giá trị của cột "CPI" với 100
df_new['CPI'] *= 100

# Hiển thị DataFrame mới
print(df_new)
```

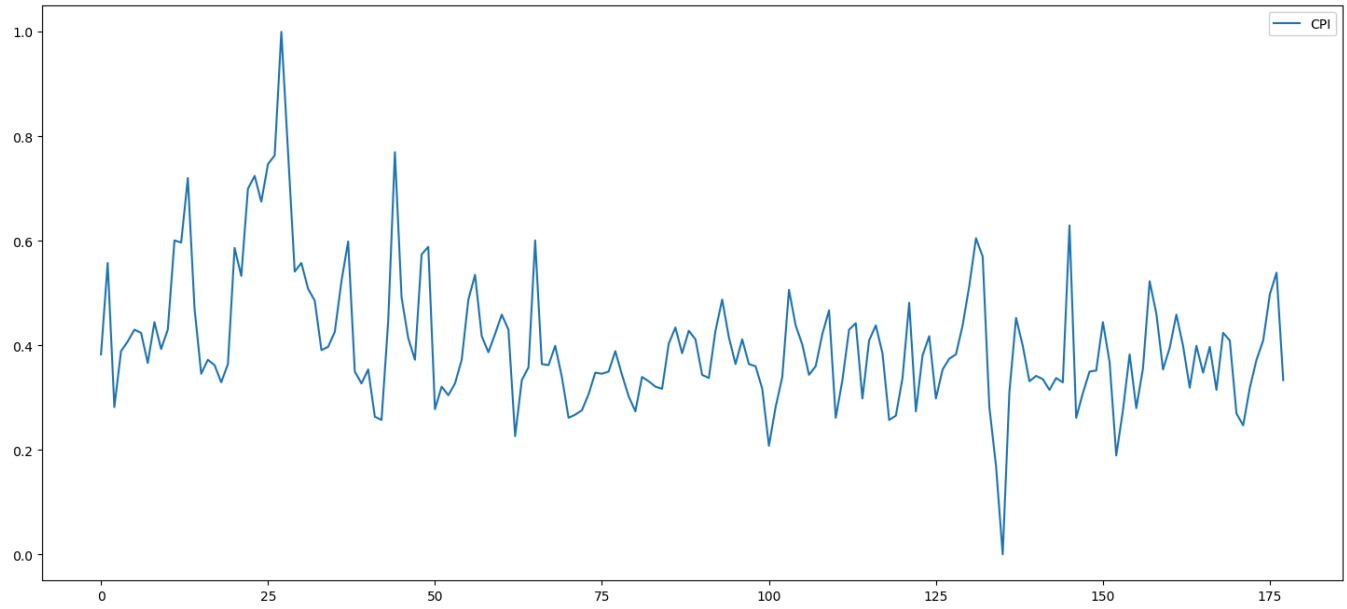
```
                Date    CPI
0     Tháng 1/2009   32.0
1     Tháng 2/2009  117.0
2     Tháng 3/2009  -17.0
3     Tháng 4/2009   35.0
4     Tháng 5/2009   44.0
..             ...    ...
173   Tháng 6/2023   27.0
```

```
      174   Tháng 7/2023   45.0
      175   Tháng 8/2023   88.0
      176   Tháng 9/2023  108.0
      177  Tháng 10/2023    8.0

      [178 rows x 2 columns]
```

```
min_val = df_new['CPI'].min()
max_val = df_new['CPI'].max()
df_new['CPI'] = (df_new['CPI'] - min_val) / (max_val - min_val)
```

```
df_new.plot(figsize = (18, 8))
pyplot.show()
```




```
def split_sequence(arr, n_steps):
  X = []
  for i in range(len(arr) - n_steps):
    X.append(arr[i:i + n_steps])
  return array(X), array(arr[n_steps:])
```

```
n_steps = 6
X, y = split_sequence(df_new['CPI'].values, n_steps)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, shuffle=False)
```

```
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
```

```
model = Sequential()
model.add(LSTM(32, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True))
model.add(Dense(1))
model.compile(optimizer = Adam(learning_rate = 0.001),loss = 'mse', metrics=['mse'])
#model.compile(optimizer = 'adam',loss = 'mse', metrics=['mse'])
model.summary()
```

```
      Model: "sequential"
      _________________________________________________________________
       Layer (type)                Output Shape              Param #
      =================================================================
       lstm (LSTM)                 (None, 1, 32)             4992

       dense (Dense)               (None, 1, 1)              33

      =================================================================
```

```
Total params: 5025 (19.63 KB)
Trainable params: 5025 (19.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
result = model.fit(X_train, y_train, epochs = 1000, batch_size=1, validation_data=(X_test, y_test))
```

```
Epoch 972/1000
129/129 [==============================] - 0s 4ms/step - loss: 0.0076 - mse: 0.0076 - val_loss: 0.0144 - val_mse: 0.0144
Epoch 973/1000
129/129 [==============================] - 0s 4ms/step - loss: 0.0069 - mse: 0.0069 - val_loss: 0.0147 - val_mse: 0.0147
Epoch 974/1000
129/129 [==============================] - 0s 4ms/step - loss: 0.0071 - mse: 0.0071 - val_loss: 0.0137 - val_mse: 0.0137
Epoch 975/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0073 - mse: 0.0073 - val_loss: 0.0134 - val_mse: 0.0134
Epoch 976/1000
129/129 [==============================] - 0s 4ms/step - loss: 0.0070 - mse: 0.0070 - val_loss: 0.0143 - val_mse: 0.0143
Epoch 977/1000
129/129 [==============================] - 1s 6ms/step - loss: 0.0071 - mse: 0.0071 - val_loss: 0.0148 - val_mse: 0.0148
Epoch 978/1000
129/129 [==============================] - 1s 6ms/step - loss: 0.0072 - mse: 0.0072 - val_loss: 0.0132 - val_mse: 0.0132
Epoch 979/1000
129/129 [==============================] - 1s 7ms/step - loss: 0.0070 - mse: 0.0070 - val_loss: 0.0132 - val_mse: 0.0132
Epoch 980/1000
129/129 [==============================] - 1s 5ms/step - loss: 0.0074 - mse: 0.0074 - val_loss: 0.0164 - val_mse: 0.0164
Epoch 981/1000
129/129 [==============================] - 1s 6ms/step - loss: 0.0072 - mse: 0.0072 - val_loss: 0.0136 - val_mse: 0.0136
Epoch 982/1000
129/129 [==============================] - 1s 7ms/step - loss: 0.0072 - mse: 0.0072 - val_loss: 0.0151 - val_mse: 0.0151
Epoch 983/1000
129/129 [==============================] - 1s 7ms/step - loss: 0.0073 - mse: 0.0073 - val_loss: 0.0135 - val_mse: 0.0135
Epoch 984/1000
129/129 [==============================] - 1s 7ms/step - loss: 0.0070 - mse: 0.0070 - val_loss: 0.0132 - val_mse: 0.0132
Epoch 985/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0071 - mse: 0.0071 - val_loss: 0.0138 - val_mse: 0.0138
Epoch 986/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0072 - mse: 0.0072 - val_loss: 0.0168 - val_mse: 0.0168
Epoch 987/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0074 - mse: 0.0074 - val_loss: 0.0139 - val_mse: 0.0139
Epoch 988/1000
129/129 [==============================] - 1s 5ms/step - loss: 0.0074 - mse: 0.0074 - val_loss: 0.0139 - val_mse: 0.0139
Epoch 989/1000
129/129 [==============================] - 0s 4ms/step - loss: 0.0073 - mse: 0.0073 - val_loss: 0.0135 - val_mse: 0.0135
Epoch 990/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0073 - mse: 0.0073 - val_loss: 0.0141 - val_mse: 0.0141
Epoch 991/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0070 - mse: 0.0070 - val_loss: 0.0152 - val_mse: 0.0152
Epoch 992/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0071 - mse: 0.0071 - val_loss: 0.0150 - val_mse: 0.0150
Epoch 993/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0070 - mse: 0.0070 - val_loss: 0.0136 - val_mse: 0.0136
Epoch 994/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0072 - mse: 0.0072 - val_loss: 0.0163 - val_mse: 0.0163
Epoch 995/1000
129/129 [==============================] - 0s 4ms/step - loss: 0.0073 - mse: 0.0073 - val_loss: 0.0138 - val_mse: 0.0138
Epoch 996/1000
129/129 [==============================] - 0s 4ms/step - loss: 0.0072 - mse: 0.0072 - val_loss: 0.0135 - val_mse: 0.0135
Epoch 997/1000
129/129 [==============================] - 0s 4ms/step - loss: 0.0070 - mse: 0.0070 - val_loss: 0.0143 - val_mse: 0.0143
Epoch 998/1000
129/129 [==============================] - 1s 4ms/step - loss: 0.0071 - mse: 0.0071 - val_loss: 0.0132 - val_mse: 0.0132
Epoch 999/1000
129/129 [==============================] - 1s 5ms/step - loss: 0.0074 - mse: 0.0074 - val_loss: 0.0134 - val_mse: 0.0134
Epoch 1000/1000
129/129 [==============================] - 1s 5ms/step - loss: 0.0073 - mse: 0.0073 - val_loss: 0.0134 - val_mse: 0.0134
```

```
y_pred = model.predict(X_test)
```

```
2/2 [==============================] - 1s 7ms/step
```

```
y_pred = y_pred.flatten()
```

```
mean_squared_error(y_test, y_pred)
```

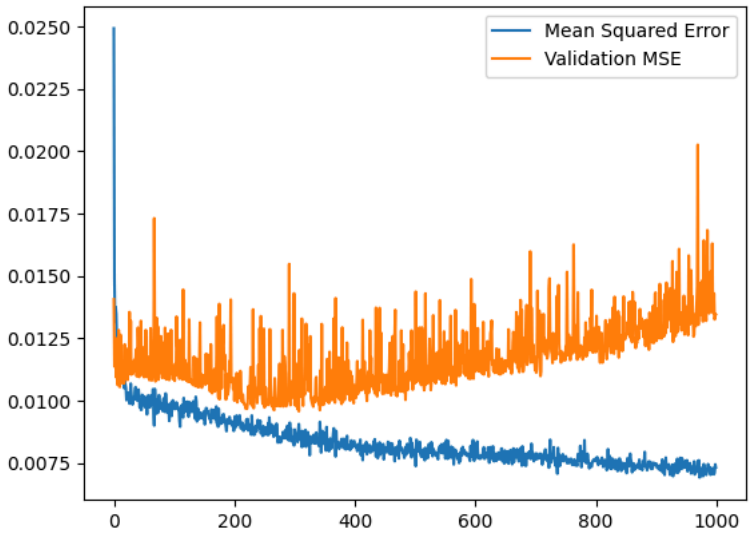
```
0.013446782308659524
```

```
r2_score(y_test, y_pred)
```

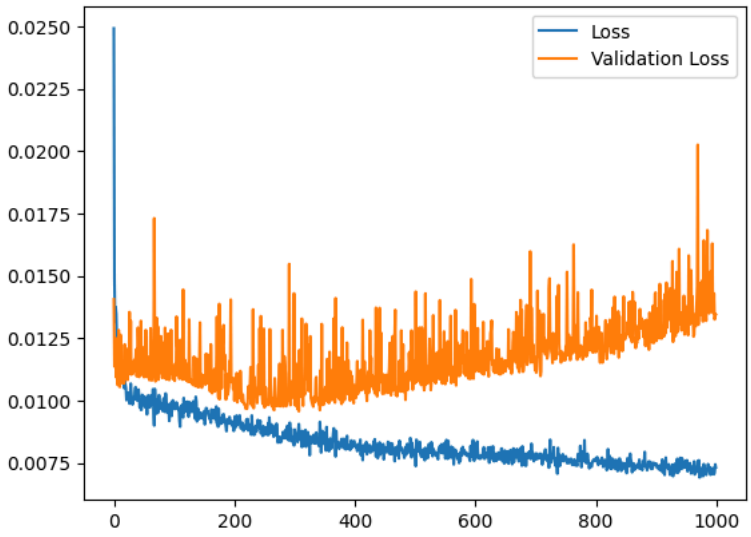
```
-0.3682845019480083
```

```
import numpy as np
```

```
pyplot.plot(result.history['mse'], label='Mean Squared Error')
pyplot.plot(result.history['val_mse'], label='Validation MSE')
pyplot.legend()
pyplot.show()
```




```
pyplot.plot(result.history['loss'], label='Loss')
pyplot.plot(result.history['val_loss'], label='Validation Loss')
pyplot.legend()
pyplot.show()
```



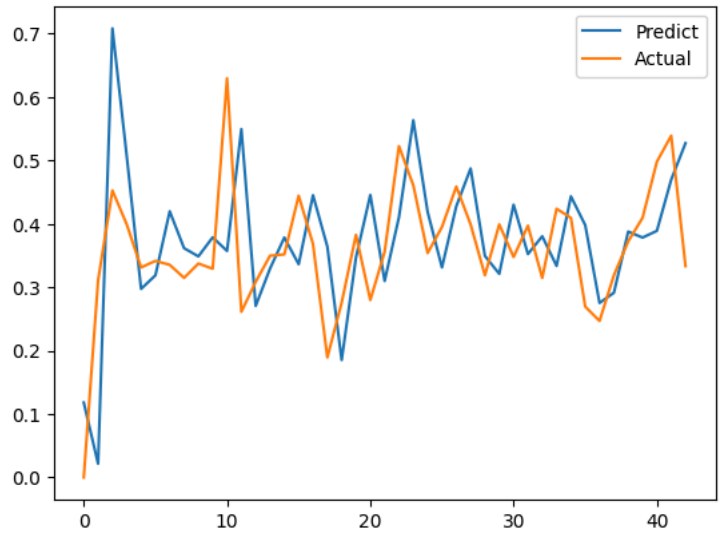

```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
#y_train_pred.flatten()
# Assuming max and min are functions that take an iterable as input
pred = concatenate((y_train_pred.flatten(), y_test_pred.flatten())) * (max(pred) - min(pred)) + min(pred)

y_scaled = y.copy()  # Create a copy of y to avoid modifying the original data
dic = {'Date': df.index[n_steps:], 'Actual': y_scaled * (max(y) - min(y)) + min(y), 'Pred': pred}


for key, value in dic.items():
    print(f"Length of {key}: {len(value)}")
```

```
5/5 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 10ms/step
Length of Date: 9
Length of Actual: 172
Length of Pred: 172
```

```
pyplot.plot(y_pred, label='Predict')
pyplot.plot(y_test, label='Actual')
pyplot.legend()
pyplot.show()
```




```
class WindowGenerator():
    def __init__(self, input_width, label_width, shift, train_df, val_df=None, test_df=None, label_columns=None):
        # Store the raw data.
        self.train_df = train_df
        self.val_df = val_df
        self.test_df = test_df
        # ... rest of your code ...

        # Work out the label column indices (if label_columns is provided)
        self.label_columns = label_columns  # Set default value as None
        if label_columns is not None:
            self.label_columns_indices = {name: i for i, name in enumerate(label_columns)}
        # ... rest of your code ...


    self.column_indices = {name: i for i, name in enumerate(train_df.columns)}
    # Work out the window parameters.
    self.input_width = input_width
    self.label_width = label_width
    self.shift = shift
    self.total_window_size = input_width + shift
    self.input_slice = slice(0, input_width)
    self.input_indices = np.arange(self.total_window_size)[self.input_slice]
    self.label_start = self.total_window_size - self.label_width
    self.labels_slice = slice(self.label_start, None)
    self.label_indices = np.arange(self.total_window_size)[self.labels_slice]

def __repr__(self):
    return '\n'.join([
      f'Total window size: {self.total_window_size}',
      f'Input indices: {self.input_indices}',
      f'Label indices: {self.label_indices}',
      f'Label column name(s): {self.label_columns}'])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-51-32cd722d2805> in <cell line: 1>()
----> 1 class WindowGenerator():
      2     def __init__(self, input_width, label_width, shift, train_df,
val_df=None, test_df=None, label_columns=None):
      3         # Store the raw data.
      4         self.train_df = train_df
      5         self.val_df = val_df

<ipython-input-51-32cd722d2805> in WindowGenerator()
     14
     15
---> 16     self.column_indices = {name: i for i, name in
enumerate(train_df.columns)}
     17     # Work out the window parameters.
     18     self.input_width = input_width

NameError: name 'train_df' is not defined
```

Next steps: Explain error