# 📦 Mã nguồn: coin

## config.py

# ==== config.py ====  
CRYPTO = "BTC/USDT" # cặp coin  
START\_DATE = "2017-01-01"  
PRE\_DAY = 30  
TEST\_SIZE = 60  
EPOCHS = 50  
BATCH\_SIZE = 32  
UNITS = 60  
MODEL\_BASENAME = "crypto"

## crypto.py

import investpy  
import pandas as pd  
import datetime as dt  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Dropout, LSTM  
import matplotlib.pyplot as plt  
  
# Import config  
from config import CRYPTO, START\_DATE, PRE\_DAY, TEST\_SIZE, EPOCHS, BATCH\_SIZE, UNITS, MODEL\_BASENAME  
  
# ==== Load dữ liệu ====  
END = dt.datetime.now().strftime("%d/%m/%Y")  
df = investpy.get\_crypto\_historical\_data(  
 crypto=CRYPTO,  
 from\_date=START\_DATE,  
 to\_date=END  
)  
  
df = pd.DataFrame(df)  
  
# ==== Tạo feature ====  
df['H-L'] = df['High'] - df['Low']  
df['O-C'] = df['Open'] - df['Close']  
for ma in [7, 14, 21]:  
 df[f'SMA\_{ma}'] = df['Close'].rolling(window=ma).mean()  
df['SD\_7'] = df['Close'].rolling(window=7).std()  
df['SD\_21'] = df['Close'].rolling(window=21).std()  
df.dropna(inplace=True)  
  
# Lưu CSV để tham khảo  
df.to\_csv(f"{CRYPTO}.csv")  
print("✅ Done Loading Data")  
  
# ==== Chuẩn hóa dữ liệu ====  
cols\_x = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
cols\_y = ['Close']  
scala\_x = MinMaxScaler(feature\_range=(0, 1))  
scala\_y = MinMaxScaler(feature\_range=(0, 1))  
scaled\_x = scala\_x.fit\_transform(df[cols\_x].values)  
scaled\_y = scala\_y.fit\_transform(df[cols\_y].values)  
  
# ==== Tạo dữ liệu train/test ====  
x\_total, y\_total = [], []  
for i in range(PRE\_DAY, len(df)):  
 x\_total.append(scaled\_x[i-PRE\_DAY:i])  
 y\_total.append(scaled\_y[i])  
  
x\_train = np.array(x\_total[:-TEST\_SIZE])  
x\_test = np.array(x\_total[-TEST\_SIZE:])  
y\_train = np.array(y\_total[:-TEST\_SIZE])  
y\_test = np.array(y\_total[-TEST\_SIZE:])  
print(x\_train.shape, y\_train.shape, x\_test.shape, y\_test.shape)  
  
# ==== Build model ====  
model = Sequential([  
 LSTM(UNITS, return\_sequences=True, input\_shape=(x\_train.shape[1], x\_train.shape[2])),  
 Dropout(0.2),  
 LSTM(UNITS, return\_sequences=True),  
 Dropout(0.2),  
 LSTM(UNITS),  
 Dropout(0.2),  
 Dense(1)  
])  
model.compile(optimizer='adam', loss='mean\_squared\_error')  
model.fit(x\_train, y\_train, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1)  
  
model.save(f"{MODEL\_BASENAME}.h5")  
print("✅ Done Training Model")  
  
# ==== Test dự đoán ====  
predict\_prices = model.predict(x\_test)  
predict\_prices = scala\_y.inverse\_transform(predict\_prices)  
  
real\_price = df[-TEST\_SIZE:]['Close'].values.reshape(-1, 1)  
  
plt.plot(real\_price, color="red", label=f"Real {CRYPTO} Prices")  
plt.plot(predict\_prices, color="blue", label=f"Predicted {CRYPTO} Prices")  
plt.title(f"{CRYPTO} Prices Prediction")  
plt.xlabel("Time")  
plt.ylabel("Price")  
plt.legend()  
plt.show()  
  
# ==== Dự đoán ngày tiếp theo ====  
x\_predict = df[-PRE\_DAY:][cols\_x].values  
x\_predict = scala\_x.transform(x\_predict)  
x\_predict = np.array(x\_predict).reshape(1, PRE\_DAY, len(cols\_x))  
  
prediction = model.predict(x\_predict)  
prediction = scala\_y.inverse\_transform(prediction)  
print(f"📈 Next day predicted price: {prediction}")

## test.py

import investpy  
import pandas as pd  
import datetime as dt  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
from tensorflow.keras import models  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Dropout, LSTM  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.metrics import Accuracy  
import matplotlib.pyplot as plt  
import os  
  
# Constants  
DEFAULT\_START\_DATE = '01/01/2024'  
COMPANY = 'PLC'  
COUNTRY = 'VietNam'  
MODEL\_PATH = f"{COMPANY}.h5"  
CSV\_PATH = f"{COMPANY}.csv"  
TEST\_SIZE = 60  
PRE\_DAY = 30  
MA\_WINDOWS = [7, 14, 21]  
  
def fetch\_stock\_data(stock, country, start, end):  
 print("[Debug] fetch\_stock\_data\n")  
 try:  
 df = investpy.get\_stock\_historical\_data(stock=stock, country=country, from\_date=start, to\_date=end)  
 return pd.DataFrame(df)  
 except Exception as e:  
 print(f"Error fetching data: {e}")  
 return None  
  
def add\_features(df):  
 print("[Debug] add\_features\n")  
 df['H-L'] = df['High'] - df['Low']  
 df['O-C'] = df['Open'] - df['Close']  
 for ma in MA\_WINDOWS:  
 df[f'SMA\_{ma}'] = df['Close'].rolling(window=ma).mean()  
 df[f'SD\_{MA\_WINDOWS[0]}'] = (((df['Close'] - df[f'SMA\_{MA\_WINDOWS[0]}'])\*\*2).rolling(window=MA\_WINDOWS[0]).sum()/MA\_WINDOWS[0])\*\*.5  
 df.dropna(inplace=True)  
 return df  
  
def scale\_data(df, feature\_cols, target\_col):  
 print("[Debug] scale\_data\n")  
 scaler\_x = MinMaxScaler(feature\_range=(0, 1))  
 scaler\_y = MinMaxScaler(feature\_range=(0, 1))  
 scaled\_x = scaler\_x.fit\_transform(df[feature\_cols].values)  
 scaled\_y = scaler\_y.fit\_transform(df[[target\_col]].values)  
 return scaled\_x, scaled\_y, scaler\_x, scaler\_y  
  
def create\_sequences(scaled\_x, scaled\_y, pre\_day):  
 print("[Debug] create\_sequences\n")  
 x, y = [], []  
 for i in range(pre\_day, len(scaled\_x)):  
 x.append(scaled\_x[i-pre\_day:i])  
 y.append(scaled\_y[i])  
 return np.array(x), np.array(y)  
  
def load\_trained\_model(model\_path):  
 print("[Debug] load\_trained\_model\n")  
 if not os.path.exists(model\_path):  
 raise FileNotFoundError(f"Model file {model\_path} not found.")  
 return models.load\_model(model\_path)  
  
def plot\_prices(real\_prices, predicted\_prices, label):  
 print("[Debug] plot\_prices\n")  
 plt.style.use('dark\_background')  
 plt.figure(figsize=(20, 16))  
 plt.plot(real\_prices, color="red", label=f"Real {label} Prices")  
 plt.plot(predicted\_prices, color="blue", label=f"Predicted {label} Prices", ls='--')  
 plt.title(f"{label} Prices")  
 plt.xlabel("Time")  
 plt.ylabel("Stock Prices")  
 plt.ylim(bottom=0)  
 plt.xlim(left=0)  
 plt.legend()  
 plt.show()  
  
# ...existing code...  
  
def fetch\_crypto\_data(crypto, start, end):  
 print("[Debug] fetch\_crypto\_data\n")  
 try:  
 df = investpy.get\_crypto\_historical\_data(crypto=crypto, from\_date=start, to\_date=end)  
 return pd.DataFrame(df)  
 except Exception as e:  
 print(f"Error fetching crypto data: {e}")  
 return None  
  
def main():  
 choice = input("Bạn muốn dự đoán (1) Cổ phiếu hay (2) Coin? Nhập 1 hoặc 2: ").strip()  
 end\_date = dt.datetime.now().strftime("%d/%m/%Y")  
 if choice == "1":  
 company = input("Nhập mã cổ phiếu (ví dụ: PLC): ").strip().upper()  
 country = input("Nhập tên quốc gia (ví dụ: VietNam): ").strip()  
 model\_path = f"{company}.h5"  
 csv\_path = f"{company}.csv"  
 df = fetch\_stock\_data(company, country, DEFAULT\_START\_DATE, end\_date)  
 label = company  
 elif choice == "2":  
 crypto = input("Nhập mã coin (ví dụ: bitcoin): ").strip().lower()  
 model\_path = f"{crypto}.h5"  
 csv\_path = f"{crypto}.csv"  
 df = fetch\_crypto\_data(crypto, DEFAULT\_START\_DATE, end\_date)  
 label = crypto  
 else:  
 print("Lựa chọn không hợp lệ.")  
 return  
  
 if df is None or df.empty:  
 print("No data to process.")  
 return  
  
 df = add\_features(df)  
 df.to\_csv(csv\_path)  
 print("Done Loading Data")  
  
 feature\_cols = ['H-L', 'O-C'] + [f'SMA\_{ma}' for ma in MA\_WINDOWS] + [f'SD\_{MA\_WINDOWS[0]}']  
 target\_col = 'Close'  
 scaled\_x, scaled\_y, scaler\_x, scaler\_y = scale\_data(df, feature\_cols, target\_col)  
  
 x\_total, y\_total = create\_sequences(scaled\_x, scaled\_y, PRE\_DAY)  
 if len(x\_total) == 0:  
 print("Not enough data for sequence creation.")  
 return  
  
 try:  
 model = load\_trained\_model(model\_path)  
 except FileNotFoundError as e:  
 print(e)  
 return  
  
 predicted\_prices = model.predict(x\_total)  
 predicted\_prices = scaler\_y.inverse\_transform(predicted\_prices)  
 real\_prices = df[PRE\_DAY:][target\_col].values.reshape(-1, 1)  
  
 print(f"Real prices shape: {real\_prices.shape}, Predicted prices shape: {predicted\_prices.shape}")  
  
 plot\_prices(real\_prices, predicted\_prices, label)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## train\_crypto.py

import os  
import re  
import numpy as np  
import pandas as pd  
import datetime as dt  
import investpy  
from sklearn.preprocessing import MinMaxScaler  
from tensorflow.keras.models import Sequential, load\_model  
from tensorflow.keras.layers import Dense, Dropout, LSTM  
  
# Import cấu hình chung  
from config import CRYPTO, START\_DATE, PRE\_DAY, TEST\_SIZE, EPOCHS, BATCH\_SIZE, UNITS, MODEL\_BASENAME  
  
  
# ==== Hàm tiện ích ====  
  
def add\_features(df):  
 df['H-L'] = df['High'] - df['Low']  
 df['O-C'] = df['Open'] - df['Close']  
 for ma in [7, 14, 21]:  
 df[f'SMA\_{ma}'] = df['Close'].rolling(window=ma).mean()  
 df['SD\_7'] = df['Close'].rolling(window=7).std()  
 df['SD\_21'] = df['Close'].rolling(window=21).std()  
 df.dropna(inplace=True)  
 return df  
  
def scale\_data(df, feature\_cols, target\_col):  
 scaler\_x = MinMaxScaler()  
 scaler\_y = MinMaxScaler()  
 scaled\_x = scaler\_x.fit\_transform(df[feature\_cols].values)  
 scaled\_y = scaler\_y.fit\_transform(df[[target\_col]].values)  
 return scaled\_x, scaled\_y  
  
def create\_sequences(scaled\_x, scaled\_y, pre\_day):  
 x, y = [], []  
 for i in range(pre\_day, len(scaled\_x)):  
 x.append(scaled\_x[i-pre\_day:i])  
 y.append(scaled\_y[i])  
 return np.array(x), np.array(y)  
  
def get\_latest\_model():  
 models = [f for f in os.listdir(".") if f.startswith("model\_") and f.endswith(".h5")]  
 if not models:  
 return None, 0  
 numbers = [int(re.findall(r"model\_(\d+)\.h5", f)[0]) for f in models]  
 latest\_num = max(numbers)  
 return f"model\_{latest\_num}.h5", latest\_num  
  
def build\_new\_model(input\_shape):  
 model = Sequential([  
 LSTM(UNITS, return\_sequences=True, input\_shape=input\_shape),  
 Dropout(0.2),  
 LSTM(UNITS, return\_sequences=True),  
 Dropout(0.2),  
 LSTM(UNITS),  
 Dropout(0.2),  
 Dense(1)  
 ])  
 model.compile(optimizer='adam', loss='mean\_squared\_error')  
 return model  
  
  
# ==== Main ====  
  
def main():  
 try:  
 # 1. Load dữ liệu crypto  
 END = dt.datetime.now().strftime("%d/%m/%Y")  
 df = investpy.get\_crypto\_historical\_data(crypto=CRYPTO, from\_date=START\_DATE, to\_date=END)  
 df = pd.DataFrame(df)  
 df = add\_features(df)  
  
 if df.empty:  
 print("❌ Lỗi: Không có dữ liệu crypto để train.")  
 return  
  
 # 2. Chuẩn hóa  
 feature\_cols = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
 target\_col = 'Close'  
 scaled\_x, scaled\_y = scale\_data(df, feature\_cols, target\_col)  
 x, y = create\_sequences(scaled\_x, scaled\_y, PRE\_DAY)  
  
 if len(x) == 0:  
 print("❌ Lỗi: Không đủ dữ liệu để tạo chuỗi train/test.")  
 return  
  
 # 3. Tự động chọn train mới hoặc finetune  
 latest\_model, latest\_num = get\_latest\_model()  
  
 if latest\_model:  
 print(f"🔄 Phát hiện model cũ: {latest\_model} → Train tiếp...")  
 model = load\_model(latest\_model)  
 model.fit(x, y, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1)  
 new\_model\_path = f"model\_{latest\_num+1}.h5"  
 model.save(new\_model\_path)  
 print(f"✅ Đã lưu model mới: {new\_model\_path}")  
 else:  
 print("🚀 Không có model cũ → Train mới từ đầu...")  
 model = build\_new\_model((x.shape[1], x.shape[2]))  
 model.fit(x, y, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1)  
 model.save(f"{MODEL\_BASENAME}.h5")  
 print(f"✅ Đã lưu model: {MODEL\_BASENAME}.h5")  
  
 except Exception as e:  
 print(f"❌ Lỗi không mong muốn: {type(e).\_\_name\_\_} → {str(e)}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## model\nope\_have.py

## utils\export\_code.py

import os  
from docx import Document  
  
# ✅ THIẾT LẬP BIẾN TOÀN CỤC  
VERSION = "0.1"  
SERVICE\_NAME = "coin"  
OUTPUT\_NAME = f"{SERVICE\_NAME}\_export\_{VERSION}.docx"  
  
def export\_code():  
 # 🧭 Xác định vị trí file script đang chạy  
 current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
 print("[DEBUG] Đường dẫn file export\_code.py:", current\_dir)  
  
 # 📁 Đường dẫn đến thư mục service (cùng cấp)  
 service\_path = os.path.abspath(os.path.join(current\_dir, ".."))  
 print("[DEBUG] Đường dẫn đến service:", service\_path)  
  
 if not os.path.exists(service\_path):  
 print("[❌] Không tìm thấy thư mục:", service\_path)  
 return  
  
 # 📝 Tạo file docx  
 doc = Document()  
 doc.add\_heading(f"📦 Mã nguồn: {SERVICE\_NAME}", level=1)  
  
 file\_count = 0  
  
 for root, dirs, files in os.walk(service\_path):  
 print("[DEBUG] Đang đọc thư mục:", root)  
  
 for file in files:  
 if not file.endswith(".py"):  
 continue  
  
 file\_path = os.path.join(root, file)  
 rel\_path = os.path.relpath(file\_path, service\_path)  
  
 print(f" 📄 Đọc file: {rel\_path}")  
  
 try:  
 with open(file\_path, 'r', encoding='utf-8') as f:  
 content = f.read()  
  
 doc.add\_heading(rel\_path, level=2)  
 doc.add\_paragraph(content, style='Normal')  
 file\_count += 1  
  
 except Exception as e:  
 print(f"[⚠️] Không đọc được file: {file\_path} → {type(e).\_\_name\_\_}: {str(e)}")  
  
 # 📤 Lưu file  
 output\_path = os.path.abspath(os.path.join(current\_dir, "../doc", OUTPUT\_NAME))  
 doc.save(output\_path)  
 print(f"✅ Hoàn tất. Đã ghi {file\_count} file vào: {output\_path}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 export\_code()