# 📦 Mã nguồn: coin

## config.py

# Cấu hình dữ liệu  
CRYPTO = "BTC/USDT"   
START\_DATE = "2020-01-01"  
END\_DATE = "2025-9-13"   
PRE\_DAY = 30  
TEST\_RATIO = 0.1  
MIN\_TEST\_DAYS = 60   
PRINT\_LAST\_N = 5  
PREDICT\_DAYS = 5  
  
# Cấu hình training  
EPOCHS = 10  
BATCH\_SIZE = 32  
UNITS = 60  
  
# Đường dẫn lưu model  
MODELS\_DIR = "models"  
MODEL\_BASENAME = "crypto"

## main.py

import os  
import numpy as np  
import pandas as pd  
from sklearn.preprocessing import MinMaxScaler  
  
from config import (  
 CRYPTO, START\_DATE, END\_DATE, PRE\_DAY, TEST\_RATIO,  
 EPOCHS, BATCH\_SIZE, MODELS\_DIR, PREDICT\_DAYS  
)  
from utils import (  
 data\_utils, scaler\_utils, model\_utils,  
 train\_utils, plot\_utils, analytic\_utils,  
 forecast\_utils, log\_utils  
)  
  
FEATURE\_COLS = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
TARGET\_COL = 'Close'  
  
  
def load\_and\_prepare\_data():  
 print(f"📥 Loading {CRYPTO} from {START\_DATE} to {END\_DATE}...")  
 df = data\_utils.load\_crypto\_data(CRYPTO, START\_DATE, END\_DATE)  
 df = data\_utils.add\_features(df)  
 if df.empty:  
 print("❌ No data.")  
 return None  
 return df  
  
  
def get\_or\_create\_scalers(train\_df):  
 # Fit scaler trên tập train để tránh leakage  
 scaler\_x, scaler\_y = scaler\_utils.get\_fresh\_scalers(  
 train\_df[FEATURE\_COLS], train\_df[[TARGET\_COL]],  
 prefix="crypto", out\_dir=MODELS\_DIR  
 )  
 return scaler\_x, scaler\_y  
  
def scale\_data(df, scaler\_x, scaler\_y):  
 scaled\_x = scaler\_x.transform(df[FEATURE\_COLS])  
 scaled\_y = scaler\_y.transform(df[[TARGET\_COL]])  
 return scaled\_x, scaled\_y  
  
  
def get\_model(x\_train):  
 latest\_version = model\_utils.get\_latest\_version()  
 version = latest\_version + 1  
  
 if latest\_version > 0:  
 prev\_path = os.path.join(MODELS\_DIR, f"v{latest\_version}", "model.keras")  
  
 RESET\_MODEL = True  
 if RESET\_MODEL:  
 print(f"🔄 Loading kiến trúc từ v{latest\_version} và reset weight...")  
 model = model\_utils.load\_existing\_model\_with\_reset(prev\_path)  
 else:  
 print(f"🔄 Finetuning from v{latest\_version} ({prev\_path})")  
 model = model\_utils.load\_existing\_model(prev\_path)  
 model.compile(optimizer="adam", loss="mean\_squared\_error")  
 else:  
 print("🚀 Training new model...")  
 model = model\_utils.build\_new\_model((x\_train.shape[1], x\_train.shape[2]))  
  
 return model, version  
  
  
def train\_and\_save\_model(model, x\_train, y\_train, version, scaler\_x, scaler\_y):  
 callbacks = train\_utils.get\_callbacks(version)  
 history = model.fit(  
 x\_train, y\_train,  
 validation\_split=0.1,  
 epochs=EPOCHS,  
 batch\_size=BATCH\_SIZE,  
 verbose=1,  
 callbacks=callbacks  
 )  
 return model, history  
  
  
def predict\_and\_evaluate(model, x\_test, y\_test, scaler\_y, df, test\_size):  
 y\_pred\_test = model.predict(x\_test)  
 y\_pred\_test = scaler\_y.inverse\_transform(y\_pred\_test)  
 y\_true\_test = scaler\_y.inverse\_transform(y\_test)  
 dates\_test = df.index[-test\_size:]  
  
 metrics = analytic\_utils.evaluate\_predictions(y\_true\_test, y\_pred\_test)  
 analytic\_utils.print\_evaluation(metrics, prefix="Test")  
  
 return y\_true\_test, y\_pred\_test, dates\_test, metrics  
  
  
def predict\_future(model, df, scaler\_x, scaler\_y):  
 preds\_future = forecast\_utils.robust\_future\_predict(  
 model, df, scaler\_x, scaler\_y,  
 pre\_day=PRE\_DAY, predict\_days=PREDICT\_DAYS, use\_true\_ratio=0.3  
 )  
 future\_dates = pd.date\_range(  
 df.index[-1] + pd.Timedelta(days=1),  
 periods=PREDICT\_DAYS,  
 freq="D"  
 )  
  
 print("🔮 Future predictions:")  
 for d, p in zip(future\_dates, preds\_future):  
 print(f"{d.date()} | {p[0]:.2f} USD")  
 return future\_dates, preds\_future  
  
  
def main():  
 df = load\_and\_prepare\_data()  
 if df is None:  
 return  
  
 test\_size = max(60, int(len(df) \* TEST\_RATIO))  
 train\_df = df.iloc[:-test\_size]  
 scaler\_x, scaler\_y = get\_or\_create\_scalers(train\_df)  
 scaled\_x, scaled\_y = scale\_data(df, scaler\_x, scaler\_y)  
  
 x\_train, y\_train, x\_test, y\_test, test\_size = train\_utils.split\_train\_test(  
 df, scaled\_x, scaled\_y, PRE\_DAY, test\_ratio=TEST\_RATIO  
 )  
  
 print(f"📊 Dataset size: {len(df)} rows")  
 print(f"📊 Train: {x\_train.shape}, Test: {x\_test.shape}")  
  
 model, version = get\_model(x\_train)  
  
 # 1. Train model  
 model, history = train\_and\_save\_model(model, x\_train, y\_train, version, scaler\_x, scaler\_y)  
  
 # 2. Evaluate  
 y\_true\_test, y\_pred\_test, dates\_test, metrics = predict\_and\_evaluate(  
 model, x\_test, y\_test, scaler\_y, df, test\_size  
 )  
  
 # 3. Save model + scaler + metrics  
 model\_utils.save\_model\_with\_meta(  
 model, scaler\_x, scaler\_y, version,  
 history=history,  
 config={"EPOCHS": EPOCHS, "BATCH\_SIZE": BATCH\_SIZE, "PRE\_DAY": PRE\_DAY},  
 metrics=metrics  
 )  
 log\_utils.save\_metrics\_to\_csv(metrics, version, MODELS\_DIR)  
  
 # 4. Forecast tương lai  
 future\_dates, preds\_future = predict\_future(model, df, scaler\_x, scaler\_y)  
 forecast\_utils.save\_future\_forecast(preds\_future, future\_dates, version, out\_dir=MODELS\_DIR)  
   
 # 5. Plot  
 plot\_utils.plot\_forecast(  
 df, y\_true\_test, y\_pred\_test, dates\_test,  
 future\_dates, preds\_future, version, out\_dir=MODELS\_DIR  
 )  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 for \_ in range(5):  
 main()

## models\nope\_have.py

## utils\analytic\_utils.py

import numpy as np  
from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  
  
def evaluate\_predictions(y\_true, y\_pred):  
 y\_true = y\_true.reshape(-1)  
 y\_pred = y\_pred.reshape(-1)  
  
 rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))  
 mae = mean\_absolute\_error(y\_true, y\_pred)  
 mape = np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  
 r2 = r2\_score(y\_true, y\_pred)  
  
 return {  
 "RMSE": rmse,  
 "MAE": mae,  
 "MAPE": mape,  
 "R2": r2  
 }  
  
def print\_evaluation(metrics, prefix="Test"):  
 print(f"📊 {prefix} evaluation:")  
 for k, v in metrics.items():  
 if k in ["RMSE", "MAE"]:  
 print(f" {k}: {v:,.2f} USD")  
 elif k == "MAPE":  
 print(f" {k}: {v:.2f}%")  
 else:  
 print(f" {k}: {v:.4f}")

## utils\data\_utils.py

import ccxt  
import pandas as pd  
import numpy as np  
from config import START\_DATE, END\_DATE  
  
def load\_crypto\_data(symbol="BTC/USDT", start\_date=START\_DATE, end\_date= END\_DATE, timeframe="1d"):  
 exchange = ccxt.binance()  
 since = exchange.parse8601(start\_date + "T00:00:00Z")  
 all\_ohlcv = []  
 while True:  
 ohlcv = exchange.fetch\_ohlcv(symbol, timeframe=timeframe, since=since, limit=2000)  
 if not ohlcv:  
 break  
 all\_ohlcv += ohlcv  
 # next since = timestamp cuối + 1ms  
 since = ohlcv[-1][0] + 1  
 if end\_date and pd.to\_datetime(since, unit="ms") > pd.to\_datetime(end\_date):  
 break  
 df = pd.DataFrame(all\_ohlcv, columns=["Timestamp","Open","High","Low","Close","Volume"])  
 df["Date"] = pd.to\_datetime(df["Timestamp"], unit="ms")  
 df.set\_index("Date", inplace=True)  
 if end\_date:  
 df = df.loc[:end\_date]  
 return df  
  
  
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 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)

## utils\export\_code.py

import os  
from docx import Document  
  
VERSION = "1.7"  
SERVICE\_NAME = "coin"  
OUTPUT\_NAME = f"{SERVICE\_NAME}\_export\_{VERSION}.docx"  
  
def export\_code():  
 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()

## utils\forecast\_utils.py

import numpy as np  
from tensorflow.keras.models import load\_model  
import joblib  
import os  
import csv  
  
def robust\_future\_predict(model, df, scaler\_x, scaler\_y, pre\_day, predict\_days=5, use\_true\_ratio=0.5):  
 FEATURE\_COLS = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
 scaled\_x = scaler\_x.transform(df[FEATURE\_COLS])  
 last\_window = scaled\_x[-pre\_day:]  
 preds\_future = []  
 current = last\_window.copy()  
  
 for i in range(predict\_days):  
 pred = model.predict(current.reshape(1, pre\_day, current.shape[1]))  
 preds\_future.append(pred[0,0])  
 new\_row = current[-1].copy()  
  
 # Nếu còn dữ liệu thật thì lấy ra trộn  
 if i < len(df):  
 true\_val = scaler\_y.transform(df[['Close']].iloc[[-(i+1)]]).ravel()[0]  
 new\_row[-1] = use\_true\_ratio \* true\_val + (1 - use\_true\_ratio) \* pred  
 else:  
 new\_row[-1] = pred  
  
 current = np.vstack([current[1:], new\_row])  
  
 preds\_future = scaler\_y.inverse\_transform(np.array(preds\_future).reshape(-1,1))  
 return preds\_future  
  
def ensemble\_predict(model\_entries, df, pre\_day, predict\_days=5):  
 preds\_all = []  
 for entry in model\_entries:  
 model = load\_model(entry["model\_path"])  
 scaler\_x = joblib.load(entry["scaler\_x"])  
 scaler\_y = joblib.load(entry["scaler\_y"])  
  
 # chuẩn hóa dữ liệu  
 FEATURE\_COLS = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
 scaled\_x = scaler\_x.transform(df[FEATURE\_COLS])  
 last\_window = scaled\_x[-pre\_day:]  
  
 preds\_future = []  
 current = last\_window.copy()  
 for \_ in range(predict\_days):  
 pred = model.predict(current.reshape(1, pre\_day, len(FEATURE\_COLS)))  
 preds\_future.append(pred[0, 0])  
 new\_row = current[-1].copy()  
 new\_row[-1] = pred  
 current = np.vstack([current[1:], new\_row])  
  
 preds\_future = scaler\_y.inverse\_transform(np.array(preds\_future).reshape(-1, 1))  
 preds\_all.append(preds\_future)  
  
 # Lấy trung bình của tất cả model  
 preds\_mean = np.mean(np.array(preds\_all), axis=0)  
 return preds\_mean  
  
def save\_future\_forecast(preds\_future, future\_dates, version, out\_dir="models"):  
 path = os.path.join(out\_dir, "future\_forecast.csv")  
 file\_exists = os.path.isfile(path)  
  
 with open(path, "a", newline="") as f:  
 writer = csv.writer(f)  
 if not file\_exists:  
 writer.writerow(["version", "date", "predicted\_close"])  
 for d, p in zip(future\_dates, preds\_future):  
 writer.writerow([version, d.strftime("%Y-%m-%d"), round(float(p[0]), 2)])  
  
 print(f"📑 Đã ghi future forecast vào {path}")

## utils\log\_utils.py

import os  
import csv  
  
def save\_metrics\_to\_csv(metrics, version, out\_dir="models"):  
 """Ghi metrics vào file CSV để tiện tra cứu."""  
 path = os.path.join(out\_dir, "metrics\_log.csv")  
 file\_exists = os.path.isfile(path)  
  
 with open(path, "a", newline="") as f:  
 writer = csv.writer(f)  
 if not file\_exists:  
 writer.writerow(["version", "RMSE", "MAE", "MAPE", "R2"])  
 writer.writerow([  
 version,  
 round(metrics["RMSE"], 2),  
 round(metrics["MAE"], 2),  
 round(metrics["MAPE"], 2),  
 round(metrics["R2"], 4)  
 ])  
 print(f"📑 Đã ghi metrics vào {path}")

## utils\model\_utils.py

import os, json, time, joblib  
from tensorflow.keras.models import Sequential, load\_model,clone\_model  
from tensorflow.keras.layers import LSTM, Dropout, Dense  
from config import UNITS, MODELS\_DIR  
  
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  
  
def get\_latest\_version():  
 if not os.path.exists(MODELS\_DIR):  
 return 0  
 versions = [int(d[1:]) for d in os.listdir(MODELS\_DIR) if d.startswith("v") and d[1:].isdigit()]  
 return max(versions) if versions else 0  
  
def load\_existing\_model(path):  
 return load\_model(path, compile=False)  
  
def save\_model\_with\_meta(model, scaler\_x, scaler\_y, version, history=None, config=None, metrics=None):  
 version\_dir = os.path.join(MODELS\_DIR, f"v{version}")  
 os.makedirs(version\_dir, exist\_ok=True)  
  
 # --- Save model  
 model\_path = os.path.join(version\_dir, "model.keras")  
 model.save(model\_path, save\_format="keras")  
  
  
 # --- Save scalers  
 joblib.dump(scaler\_x, os.path.join(version\_dir, "scaler\_x.pkl"))  
 joblib.dump(scaler\_y, os.path.join(version\_dir, "scaler\_y.pkl"))  
  
 # --- Save training history  
 if history is not None:  
 with open(os.path.join(version\_dir, "history.json"), "w") as f:  
 json.dump(history.history, f, indent=2)  
  
 # --- Save metadata  
 meta = {  
 "version": version,  
 "model\_path": model\_path,  
 "scaler\_x": os.path.join(version\_dir, "scaler\_x.pkl"),  
 "scaler\_y": os.path.join(version\_dir, "scaler\_y.pkl"),  
 "timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),  
 "config": config or {},  
 "metrics": metrics or {} # 👈 thêm vào đây  
 }  
 with open(os.path.join(version\_dir, "meta.json"), "w") as f:  
 json.dump(meta, f, indent=2)  
  
 # --- Update registry.json  
 registry\_path = os.path.join(MODELS\_DIR, "registry.json")  
 registry = []  
 if os.path.exists(registry\_path):  
 with open(registry\_path, "r") as f:  
 registry = json.load(f)  
 registry.append(meta)  
 with open(registry\_path, "w") as f:  
 json.dump(registry, f, indent=2)  
  
 # --- Update latest.json  
 latest\_path = os.path.join(MODELS\_DIR, "latest.json")  
 with open(latest\_path, "w") as f:  
 json.dump(meta, f, indent=2)  
  
 return model\_path  
  
def load\_existing\_model\_with\_reset(path, input\_shape=None):  
 old\_model = load\_model(path, compile=False)   
 model = clone\_model(old\_model)   
 if input\_shape is not None:  
 try:  
 model.build((None, input\_shape[0], input\_shape[1]))  
 except Exception:  
 pass   
  
 model.compile(optimizer="adam", loss="mean\_squared\_error")  
 print("🔄 Loaded architecture từ model cũ, reset weights.")  
 return model

## utils\plot\_utils.py

import os  
import matplotlib.pyplot as plt  
import matplotlib.ticker as mtick  
  
def plot\_forecast(df, y\_true\_test, y\_pred\_test, dates\_test,  
 future\_dates, preds\_future, version, out\_dir="models"):  
  
 plots\_dir = os.path.join(out\_dir, "plots")  
 os.makedirs(plots\_dir, exist\_ok=True)  
  
 plt.figure(figsize=(12, 6))  
  
 # Thực tế toàn bộ  
 plt.plot(df.index, df["Close"], color="blue", label="Thực tế")  
  
 # Dự đoán test  
 plt.plot(dates\_test, y\_pred\_test, color="orange", linestyle="--", label="Dự đoán (test)")  
  
 # Dự đoán tương lai  
 plt.plot(future\_dates, preds\_future, color="red", linestyle="--", marker="o", label="Dự đoán (tương lai)")  
  
 # Vạch đánh dấu END\_DATE  
 plt.axvline(df.index[-1], color="gray", linestyle=":", label="END\_DATE")  
  
 # Format trục Y thành USD  
 ax = plt.gca()  
 ax.yaxis.set\_major\_formatter(mtick.StrMethodFormatter('${x:,.0f}'))  
  
 plt.title(f"Dự báo giá {df.columns.name or 'crypto'} - Model v{version}")  
 plt.xlabel("Ngày")  
 plt.ylabel("Giá (USD)")  
 plt.legend()  
 plt.tight\_layout()  
  
 path = os.path.join(plots\_dir, f"forecast\_v{version}.png")  
 plt.savefig(path)  
 plt.close()  
 print(f"📊 Đã lưu biểu đồ dự báo: {path}")

## utils\scaler\_utils.py

import joblib  
from sklearn.preprocessing import MinMaxScaler  
  
def save\_scalers(scaler\_x, scaler\_y, prefix="scaler", out\_dir="."):  
 joblib.dump(scaler\_x, f"{out\_dir}/{prefix}\_x.pkl")  
 joblib.dump(scaler\_y, f"{out\_dir}/{prefix}\_y.pkl")  
 print(f"💾 Đã lưu scaler vào {out\_dir}")  
  
def load\_scalers(prefix="scaler", in\_dir="."):  
 try:  
 scaler\_x = joblib.load(f"{in\_dir}/{prefix}\_x.pkl")  
 scaler\_y = joblib.load(f"{in\_dir}/{prefix}\_y.pkl")  
 print("📂 Đã load scaler từ file.")  
 return scaler\_x, scaler\_y  
 except FileNotFoundError:  
 print("⚠️ Không tìm thấy scaler cũ, sẽ tạo mới.")  
 return None, None  
  
def get\_fresh\_scalers(train\_X, train\_y, prefix="scaler", out\_dir="."):  
 scaler\_x = MinMaxScaler().fit(train\_X)   
 scaler\_y = MinMaxScaler().fit(train\_y)   
 joblib.dump(scaler\_x, f"{out\_dir}/{prefix}\_x.pkl")  
 joblib.dump(scaler\_y, f"{out\_dir}/{prefix}\_y.pkl")  
 print(f"💾 Đã cập nhật scaler (reset) vào {out\_dir}")  
 return scaler\_x, scaler\_y

## utils\selector\_utils.py

import os, json  
  
def get\_best\_model(registry\_path="models/registry.json", criterion="R2", mode="max"):  
 if not os.path.exists(registry\_path):  
 raise FileNotFoundError("❌ registry.json không tồn tại")  
  
 with open(registry\_path, "r") as f:  
 registry = json.load(f)  
  
 best = None  
 for entry in registry:  
 metrics = entry.get("metrics", {})  
 if criterion not in metrics:  
 continue  
 value = metrics[criterion]  
 if best is None:  
 best = entry  
 else:  
 if mode == "max" and value > best["metrics"][criterion]:  
 best = entry  
 elif mode == "min" and value < best["metrics"][criterion]:  
 best = entry  
 return best

## utils\train\_utils.py

import os, random, numpy as np, tensorflow as tf  
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint  
from config import MODELS\_DIR  
  
SEED = 42  
random.seed(SEED)  
np.random.seed(SEED)  
tf.random.set\_seed(SEED)  
  
def get\_callbacks(version):  
 version\_dir = os.path.join(MODELS\_DIR, f"v{version}")  
 os.makedirs(version\_dir, exist\_ok=True)  
  
 callbacks = [  
 EarlyStopping(monitor="val\_loss", patience=3, restore\_best\_weights=True), # giảm patience  
 ModelCheckpoint(  
 os.path.join(version\_dir, "best\_model\_tmp.keras"),  
 monitor="val\_loss",  
 save\_best\_only=True  
 )  
 ]  
 return callbacks  
  
def split\_train\_test(df, scaled\_x, scaled\_y, pre\_day, test\_ratio=0.2, min\_test\_days=60):  
 n = len(df)  
 test\_size = max(min\_test\_days, int(n \* test\_ratio))  
 if test\_size >= n - pre\_day:  
 test\_size = max(1, n - pre\_day - 1)  
  
 x, y = [], []  
 for i in range(pre\_day, len(scaled\_x)):  
 x.append(scaled\_x[i-pre\_day:i])  
 y.append(scaled\_y[i])  
 x, y = np.array(x), np.array(y)  
  
 x\_train, y\_train = x[:-test\_size], y[:-test\_size]  
 x\_test, y\_test = x[-test\_size:], y[-test\_size:]  
  
 return x\_train, y\_train, x\_test, y\_test, test\_size