Nhóm 4

1. K214140940, Nguyễn Văn Tuấn Kiệt
2. K214140942, Nguyễn Khánh Linh
3. K214140957, Võ Minh Thư
4. K214140959, Phạm Hoàng Thủy Tiên
5. K214142104, Lê Nguyễn Kim Trinh

```
!pip install streamlit pyngrok pyspark tensorflow pycoingecko pandas numpy scikit-learn

!ngrok authtoken 2oNBajSooRk2Lkdz6OAOIg5ze8P_3yNGjJemssBoYazXjxRQf
```

Hiện kết quả đã ẩn

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
%%writefile app.py

import streamlit as st
from tensorflow.keras.models import load_model
import pandas as pd
import numpy as np
import datetime
import time
import plotly.graph_objects as go
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, TimestampType, DoubleType, FloatType
from pycoingecko import CoinGeckoAPI
from pyspark.sql.functions import asc
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

cg = CoinGeckoAPI()
spark = SparkSession.builder.appName("CryptoData").getOrCreate()

coins_to_fetch = {
    'Bitcoin': 'bitcoin',
    'Coinbase Wrapped BTC': 'coinbase-wrapped-btc',
    'Wrapped Bitcoin': 'wrapped-bitcoin',
    'Solv Protocol SolvBTC': 'solv-btc',
    'Wrapped stETH': 'wrapped-steth',
    'Rocket Pool ETH': 'rocket-pool-eth',
    'Wrapped eETH': 'wrapped-eeth',
    'Mantle Staked Ether': 'mantle-staked-ether',
    'Renzo Restaked ETH': 'renzo-restaked-eth',
    'Ethereum': 'ethereum'
}

def fetch_data(coin_id, vs_currency="usd", hours_to_fetch=1439, days_per_request=30):
    chunks_needed = hours_to_fetch // (days_per_request * 24) + 1
    end_date = datetime.datetime.now()
    all_data_rdd = spark.sparkContext.emptyRDD()

    for _ in range(chunks_needed):
        start_date = end_date - datetime.timedelta(days=days_per_request)

        data = cg.get_coin_market_chart_range_by_id(
            id=coin_id,
            vs_currency=vs_currency,
            from_timestamp=int(start_date.timestamp()),
            to_timestamp=int(end_date.timestamp())
        )

        current_data = [(price_data[0], price_data[1], volume_data[1])
                        for price_data, volume_data in zip(data['prices'], data['total_volumes'])]

        current_rdd = spark.sparkContext.parallelize(current_data)
        all_data_rdd = all_data_rdd.union(current_rdd)
```

```
            all_data_rdd = all_data_rdd.union(current_rdd)
            end_date = start_date
            time.sleep(0.5)

        all_data_rdd = all_data_rdd.take(1439)
        all_data_rdd = spark.sparkContext.parallelize(all_data_rdd)

        def convert_timestamp(row):
            timestamp_ms = row[0]
            dt_object = datetime.datetime.fromtimestamp(timestamp_ms / 1000)
            return (dt_object, row[1], row[2])

        converted_rdd = all_data_rdd.map(convert_timestamp)

        new_schema = StructType([
            StructField("timestamp", TimestampType(), True),
            StructField("price", DoubleType(), True),
            StructField("volume", FloatType(), True)
        ])

        df_spark = spark.createDataFrame(converted_rdd, schema=new_schema)
        df_spark = df_spark.orderBy("timestamp")
        df = df_spark.toPandas()
        df['timestamp'] = pd.to_datetime(df['timestamp'])
        df.set_index("timestamp", inplace=True)
        return df

    def calculate_macd(df, short_window=12, long_window=26, signal_window=9):
        df['ema_short'] = df['price'].ewm(span=short_window, adjust=False).mean()
        df['ema_long'] = df['price'].ewm(span=long_window, adjust=False).mean()
        df['macd'] = df['ema_short'] - df['ema_long']
        df['signal'] = df['macd'].ewm(span=signal_window, adjust=False).mean()
        df['change'] = df['macd'] - df['signal']
        return df

    def add_indicators(df, window=20):
        df['moving_avg'] = df['price'].rolling(window=window).mean()
        df['std_dev'] = df['price'].rolling(window=window).std()
        df['upper_band'] = df['moving_avg'] + (df['price'].rolling(window=window).std() * 2)
        df['lower_band'] = df['moving_avg'] - (df['price'].rolling(window=window).std() * 2)
        df['sma20'] = df['price'].rolling(window=window).mean()
        df['sma50'] = df['price'].rolling(window=50).mean()
        return df
    def calculate_signals(df_coin):
        short_window = int(0.025 * len(df_coin))
        long_window = int(0.05 * len(df_coin))
        signals = pd.DataFrame(index=df_coin.index)
        signals['signal'] = 0.0
        signals['short_ma'] = df_coin['price'].rolling(window=short_window, min_periods=1, center=False).mean()
        signals['long_ma'] = df_coin['price'].rolling(window=long_window, min_periods=1, center=False).mean()
        signals['signal'][short_window:] = np.where(signals['short_ma'][short_window:] > signals['long_ma'][short_window:], 1.0, 0.0)
        signals['positions'] = signals['signal'].diff().fillna(0)
        return signals

    def buy_coin(real_movement, signal, df, initial_money=40000, max_buy=1, max_sell=1):
        starting_money = initial_money
        states_sell = []
        states_buy = []
        current_inventory = 0

        def buy(i, initial_money, current_inventory):
            shares = initial_money // real_movement[i]
            if shares < 1:
                print(
                    'day %d: total balances %f, not enough money to buy a unit price %f'
                    % (i, initial_money, real_movement[i])
                )
            else:
                if shares > max_buy:
                    buy_units = max_buy
                else:
                    buy_units = shares
                initial_money -= buy_units * real_movement[i]
                current_inventory += buy_units
                print(
                    'day %d: buy %d units at price %f, total balance %f'
                    % (i, buy_units, buy_units * real_movement[i], initial_money)
                )
                states_buy.append(0)
            return initial_money, current_inventory

        for i in range(real_movement.shape[0] - int(0.025 * len(df))):
            state = signal[i]
            if state == 1:
                initial_money, current_inventory = buy(i, initial_money, current_inventory)
                states_buy.append(i)
            elif state == -1:
                if current_inventory == 0:
                    print('day %d: cannot sell anything, inventory 0' % (i))
                else:
                    if current_inventory > max_sell:
                        sell_units = max_sell
                    else:
                        sell_units = current_inventory
                    current_inventory -= sell_units
                    total_sell = sell_units * real_movement[i]
                    initial_money += total_sell
                    try:
                        invest = (
                            (real_movement[i] - real_movement[states_buy[-1]])
                            / real_movement[states_buy[-1]]
                        ) * 100
                    except:
                        invest = 0
                    print(
                        'day %d, sell %d units at price %f, investment %f %%, total balance %f,'
                        % (i, sell_units, total_sell, invest, initial_money)
                    )
                states_sell.append(i)
        invest = ((initial_money - starting_money) / starting_money) * 100
        total_gains = initial_money - starting_money
        return states_buy, states_sell, total_gains, invest
    def plot_signals(df_coin, Signals, states_buy, states_sell, coin_selection):
        close = df_coin['price']
        fig2 = go.Figure()

        fig2.add_trace(go.Scatter(
            x=df_coin.index,
            y=close,
            mode='lines',
            line=dict(color='steelblue', width=1),
            name='Price'
        ))

        fig2.add_trace(go.Scatter(
            x=df_coin.index[states_buy],
            y=close[states_buy],
            mode='markers',
            marker=dict(symbol='triangle-up', size=10, color='green'),
            name='Buying Signal'
        ))

        fig2.add_trace(go.Scatter(
            x=df_coin.index[states_sell],
            y=close[states_sell],
            mode='markers',
            marker=dict(symbol='triangle-down', size=10, color='red'),
            name='Selling Signal'
        ))

        fig2.update_layout(
            title=f'BUY / SELL INDICATORS for {selected_coin}',
            title_x=0.5,
            xaxis_title='Year',
            yaxis_title='Price',
            width=1200,
            height=500,
            legend=dict(
                x=0.01,
                y=0.99,
                bgcolor='rgba(255, 255, 255, 0.5)',
                bordercolor='rgba(0, 0, 0, 0.1)',
                borderwidth=1
            )
        )
```

```
        return fig2
    def lstm(df_coin, coin_id):
        split_date = int(len(df_coin) * 0.8)
        train = df_coin.iloc[:split_date]
        test = df_coin.iloc[split_date:]
        train_processed = df_coin.iloc[:, 0:1].values
        train_processed = train_processed[0:len(train):1]
        test_processed = df_coin.iloc[:, 0:1].values
        test_processed = test_processed[len(train)-1:len(df_coin)-1:1]

        scaler = MinMaxScaler(feature_range=(-1, 1))
        train_sc = scaler.fit_transform(train_processed)
        test_sc = scaler.transform(test_processed)
        X_train = train_sc[:-1]
        y_train = train_sc[1:]
        X_test = test_sc[:-1]
        y_test = test_sc[1:]
        X_train_lmse = X_train.reshape(X_train.shape[0], 1, 1)
        X_test_lmse = X_test.reshape(X_test.shape[0], 1, 1)
        model_path = f'/content/drive/Shareddrives/Bigdata/{coin_id}_model.h5'
        lstm_model = load_model(model_path)
        y_pred_test_lstm = lstm_model.predict(X_test_lmse)
        y_train_pred_lstm = lstm_model.predict(X_train_lmse)
        lstm_y_pred_test = lstm_model.predict(X_test_lmse)
        lstm_y_pred_test_original = scaler.inverse_transform(lstm_y_pred_test)
        lstm_mae = mean_absolute_error(y_test, lstm_y_pred_test)
        lstm_rmse = np.sqrt(mean_squared_error(y_test, lstm_y_pred_test))
        last_price = cg.get_price(ids=coin_id, vs_currencies='usd')[coin_id]['usd']
        last_price_scaled = scaler.transform(np.array([[last_price]]))
        X_last = last_price_scaled.reshape(1, 1, 1)
        next_price_scaled = lstm_model.predict(X_last)
        next_price = scaler.inverse_transform(next_price_scaled)[0][0]

        symbol = 'triangle-up' if next_price > last_price else 'triangle-down'
        return y_test, test_processed, lstm_y_pred_test_original, next_price, symbol

    st.title('Crypto Tracking')

    selected_coin = st.selectbox('Select a coin:', list(coins_to_fetch.keys()))

    indicator_options = {
        'Moving Average': 'moving_avg',
        'Upper Band': 'upper_band',
        'Lower Band': 'lower_band',
        'SMA 20': 'sma20',
        'SMA 50': 'sma50',
        'MACD': 'MACD',
        'Signal': 'Signal',
        'LSTM': 'LSTM'
    }
    selected_indicators = st.multiselect('Select indicators to display:', list(indicator_options.keys()))

    if selected_coin:
        coin_id = coins_to_fetch[selected_coin]
        df = fetch_data(coin_id)
        df = add_indicators(df)

        if 'MACD' in selected_indicators:
            df = calculate_macd(df)

        fig = go.Figure()

        fig.add_trace(go.Bar(x=df.index, y=df['volume'] / 1e9, name='Volume', marker_color='silver', yaxis='y1'))
        fig.add_trace(go.Scatter(x=df.index, y=df['price'], mode='lines', name='Price', line=dict(width=1), yaxis='y2'))

        for indicator in selected_indicators:
            if indicator in ['MACD', 'Signal', 'LSTM']:
                continue
            fig.add_trace(go.Scatter(
                x=df.index, y=df[indicator_options[indicator]], mode='lines', name=indicator, line=dict(width=1), yaxis='y2'))

        fig.update_layout(
            title=f'{selected_coin} Price and Selected Indicators', title_x=0.5,
            xaxis_title="Date", yaxis=dict(title="Volume"),
            yaxis2=dict(title="Price", overlaying='y', side='right'),
            legend=dict(x=1, y=1, bgcolor='rgba(255, 255, 255, 0.5)'),
            plot_bgcolor='rgba(0, 0, 0, 0)')
        st.plotly_chart(fig)
        st.plotly_chart(fig)

        if 'MACD' in selected_indicators:
            fig1 = go.Figure()
            fig1.add_trace(go.Scatter(x=df.index, y=df['macd'], mode='lines', name='MACD', line=dict(color='blue', width=1)))
            fig1.add_trace(go.Scatter(x=df.index, y=df['signal'], mode='lines', name='Signal Line', line=dict(color='orange', width=1)))
            df['positive'] = df['change'].where(df['change'] > 0, 0)
            df['negative'] = df['change'].where(df['change'] < 0, 0)
            fig1.add_trace(go.Bar(x=df.index, y=df['positive'], name='Positive', marker=dict(color='green')))
            fig1.add_trace(go.Bar(x=df.index, y=df['negative'], name='Negative', marker=dict(color='red')))
            fig1.update_layout(
                title='MACD',
                title_x=0.5,
                xaxis_title='Date',
                yaxis_title='Value',
                legend_title='Indicators',
                legend=dict(bgcolor='rgba(255, 255, 255, 0.5)'),
                plot_bgcolor='rgba(0, 0, 0, 0)'
            )
            st.plotly_chart(fig1)

        if 'Signal' in selected_indicators:
            signals = calculate_signals(df)
            states_buy, states_sell, total_gains, invest = buy_coin(df['price'].values, signals['positions'].values, df)
            fig2 = plot_signals(df, signals, states_buy, states_sell, selected_coin)
            st.plotly_chart(fig2)

        if 'LSTM' in selected_indicators:
            y_test, test_processed, lstm_y_pred_test_original, next_price, symbol = lstm(df, coin_id)
            fig3 = go.Figure()
            fig3.add_trace(go.Scatter(x=np.arange(len(test_processed)), y=test_processed.flatten(), mode='lines', name='Actual Price'))
            fig3.add_trace(go.Scatter(x=np.arange(len(lstm_y_pred_test_original)), y=lstm_y_pred_test_original.flatten(), mode='lines', name='Pre
            fig3.add_trace(go.Scatter(x=[len(y_test)], y=[next_price], mode='markers', name='Next Price', marker_symbol=symbol, marker_size=10))
            st.plotly_chart(fig3)

    else:
        st.warning("Please select a coin.")
```

```
Overwriting app.py
```

```
from pyngrok import ngrok
import os

public_url = ngrok.connect(8501)
print("Public URL:", public_url)

os.system("streamlit run Crypto.py &")
```

```
Public URL: NgrokTunnel: "https://bf78-34-173-144-185.ngrok-free.app" -> "http://localhost:8501"
0
```