Trường Đại học Kinh tế-Luật Thành phố Hồ Chí Minh

BÁO CÁO CUỐI KỲ

Đề tài : Dự báo giá Polkadot bằng mô hình ARIMA

Họ và tên: Giáp Hoàng Long

MSSV: K194141728

Môn học: Ứng dụng Python trong tài chính

Giảng viên : Ngô Phú Thanh

GIỚI THIỆU

Thị trường cryptocurrency là một thị trường tương đối mới mẻ và nổi lên nhanh chóng trong những năm gần đây. Tốc độ tăng trưởng của thị trường này trong năm 2021 nhìn chung là tốt và mang đến nhiều triển vọng trong tương lai.

Theo nhận định trong năm 2022, xu hướng của thị trường tiền điện tử vẫn sẽ là sự hồi phục và bứt phá của các đồng coin nền tảng (top coin). Khi mà các nền tảng lớn như Etherium, Polkadot, Near,.. đang có những bước chuẩn bị cuối cùng để nâng cấp hệ thống cũng như khởi chạy những blockchain đầu tiên trong hệ sinh thái của mình.

Polkadot là một blockchain nền tảng, mã nguồn mở, được triển khai bởi Web3 Foundation dưới sự dẫn dắt của cựu co-founder Etherium Gavin Wood. DOT cũng sẽ được sử dụng làm đồng tiền chính để đấu giá và tham gia vào hệ thống.

Dự báo chuỗi thời gian là một trong những dự báo điển hình của môn thống kê,, machine learning. Với việc sử dụng những chuỗi mang yếu tố thời gian để xây dựng mô hình với giả định là những việc đã xảy ra trong quá khứ sẽ được lặp lại và từ đó dùng để đưa ra các dự báo về tương lai. Arima là mô hình dự báo chuỗi thời gian sẽ được áp dụng trong nghiên cứu này.

GIẢ THUYẾT

Mô hình Arima dựa trên giả thuyết chuỗi dừng và phương sai sai số không đổi. Mô hình sẽ dựa vào các tín hiệu của dữ liệu trong quá khứ đã xảy ra để đưa ra các dự báo về nó. Mô hình Arima gồm 3 phần chính là AR (auto regression), MA (moving average) và I (Integrated). Mô hình là sự kết hợp của hai yếu tố chính là tự hồi quy và trung bình trượt nhưng vì giá thường là chuyển động không dừng, khi đó ta dùng phương pháp đơn giản nhất là sai phân để giá mang tính dừng. Từ đó, có thể tìm ra 3 yếu tố chính *p,d,q* để cho ra mô hình tốt nhất.

QUY TRÌNH

1. **Gọi các thư viện hỗ trợ**

```
# !pip install binance

from binance import Client
import pandas as pd, numpy as np, matplotlib.pyplot as plt
import mplfinance as mpf
import math
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error

import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

import warnings
warnings.filterwarnings('ignore')

import scipy.stats as scs
import statsmodels.api as sm

# Set figsize of plot
plt.rcParams["figure.figsize"] = (20, 7)
```

Đầu tiên, chúng ta sẽ import một số thư viện cần sử dụng thực hiện mô hình. Dữ liệu sẽ được lấy thông qua API của Binance nên cần phải cài đặt package của binance.

Đồng thời, cũng sẽ thực hiện việc cố định kích thước của các plot sẽ vẽ sau này ngay từ đầu. Ở đây kích thước được sử dụng là (20,7).

2. Lấy dữ liệu quá khứ

```
# Get public key and private key from Binance (API Management)
api_key = "NT1J5dc36vJnNtzoqGK6JuJcJI7oqEYKzYYGsZcswFaYOA5SrxnDWwvaX33qsZmg"
secret ="IqN8nyddAUEcB10V4nzXFTjiEFPsK1PZUvN5C7XZhMJyIBDOPVPxmWuyfKpetLKh"
client = Client(api_key, secret)

#Get data
historical = client.get_historical_klines("DOTUSDT", Client.KLINE_INTERVAL_1DAY,"1 Jan 2013")
data = pd.DataFrame(historical)

data
```

✓ 0.5s

Để có thể lấy dữ liệu từ binance, trước tiên phải tạo tài khoản trên sàn binance, thực hiện các yêu cầu về e-KYC của sàn và chờ sàn xác nhận. Sau khi hoàn tất truy cập vào API Mangement (chỉ có trên trình duyệt web cho PC) khởi tạo api key cá nhân.

Tại biến *client gọi* hàm Client và đưa vào api_key và secret key.

Lưu trữ dữ liệu vào biến *historical* gọi hàm get_historical_klines, bên trong hàm chúng ta sẽ nhập ticker của cặp tiền tệ mà sẽ dùng để phân tích. Vd : BTCUSDT (giá bitcoin theo usd), ETHBTC (giá etherium theo bitcoin),.... Trong bài này sẽ sử dụng cặp tiền ảo là DOTUSDT (giá polkadot theo usd).

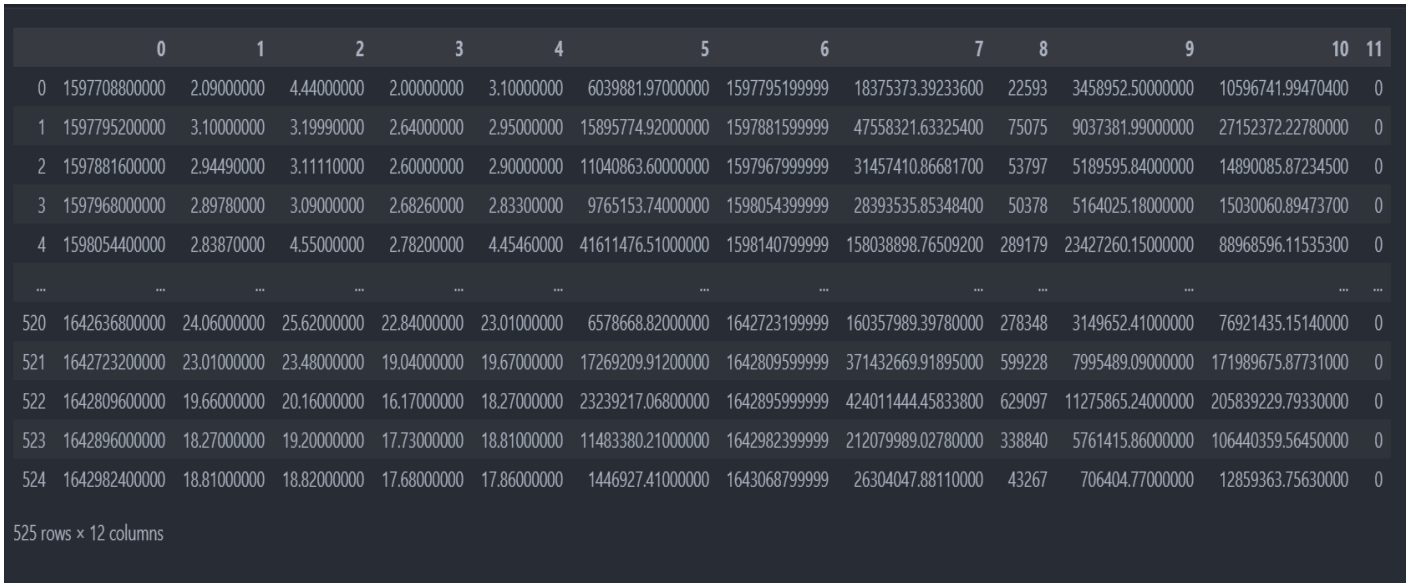
Dữ liệu được chọn dữ liệu ngày(ngoài ra còn có dữ liệu phút, giờ và tháng). Vì để có thể lấy tất cả dữ liệu ngày nên t sẽ chỉnh thời gian để lấy được lâu nhất.

Đưa dữ liệu vào dạng dataframe.

***Lưu ý:** Vì đây là API được lấy từ sàn Binance nên sẽ có những hạn chế như giá của các đồng tiền sẽ chỉ lấy được từ ngày đồng tiền đó được list trên sàn binance. Điểm lưu ý tiếp theo là, số lượng coin được list trên sàn binance là hạn

chế. Hiện tại chỉ khoảng hơn 500 coin được list so với 16000 của market.

Kết quả :



	0	1	2	3	4	5	6	7	8	9	10	11
0	1597708800000	2.09000000	4.44000000	2.00000000	3.10000000	6039881.97000000	1597795199999	18375373.39233600	22593	3458952.50000000	10596741.99470400	0
1	1597795200000	3.10000000	3.19990000	2.64000000	2.95000000	15895774.92000000	1597881599999	47558321.63325400	75075	9037381.99000000	27152372.22780000	0
2	1597881600000	2.94490000	3.11110000	2.60000000	2.90000000	11040863.60000000	1597967999999	31457410.86681700	53797	5189595.84000000	14890085.87234500	0
3	1597968000000	2.89780000	3.09000000	2.68260000	2.83300000	9765153.74000000	1598054399999	28393535.85348400	50378	5164025.18000000	15030060.89473700	0
4	1598054400000	2.83870000	4.55000000	2.78200000	4.45460000	41611476.51000000	1598140799999	158038898.76509200	289179	23427260.15000000	88968596.11535300	0
...	...	...	...	...	...	...	...	...	...	...	...	...
520	1642636800000	24.06000000	25.62000000	22.84000000	23.01000000	6578668.82000000	1642723199999	160357989.39780000	278348	3149652.41000000	76921435.15140000	0
521	1642723200000	23.01000000	23.48000000	19.04000000	19.67000000	17269209.91200000	1642809599999	371432669.91895000	599228	7995489.09000000	171989675.87731000	0
522	1642809600000	19.66000000	20.16000000	16.17000000	18.27000000	23239217.06800000	1642895999999	424011444.45833800	629097	11275865.24000000	205839229.79330000	0
523	1642896000000	18.27000000	19.20000000	17.73000000	18.81000000	11483380.21000000	1642982399999	212079989.02780000	338840	5761415.86000000	106440359.56450000	0
524	1642982400000	18.81000000	18.82000000	17.68000000	17.86000000	1446927.41000000	1643068799999	26304047.88110000	43267	706404.77000000	12859363.75630000	0

525 rows × 12 columns

Đây là dữ liệu thô ban đầu lấy được từ binance. 525 rows và 12 columns. Hoàn toàn chưa có tên cột

3. Xử lý dữ liệu thô

```
#Set index and name for columns
data.columns = ["Open time",'Open','High','Low','Close','Volume','Close time','Quote Asset Volume','Number of trade','tb base volume','tb quote volume','ignore']

#Convert open time and close time columns from num to datatime
data['Open time']= pd.to_datetime(data['Open time']/1000, unit ='s')
data['Close time']= pd.to_datetime(data['Close time']/1000, unit ='s')

#Tranform the others to numeric
numeric_columns = ['Open','High','Low','Close','Volume','Quote Asset Volume','tb base volume','tb quote volume']
data[numeric_columns] = data[numeric_columns].apply(pd.to_numeric, axis =1)

data
```

Đầu tiên là đặt tên cho các cột trong dataframe *data*.

Nhìn vào column 0 và 6 hay column ‘Open time’ và ‘Close time’ ban đầu ở dữ liệu thô chưa được đưa về dạng ngày tháng thông thường nên ta sẽ chuyển đổi 2 cột này. Các cột còn lại trừ 2 cột vừa chuyển đổi hiện vẫn đang ở dạng object, để có thể dùng tính toán cho các bước tiếp theo ta tiến hành chuyển đổi tất cả

sang dạng số.

Kết quả:

	Open time	Open	High	Low	Close	Volume	Close time	Quote Asset Volume	Number of trade	tb base volume	tb quote volume	ignore
0	2020-08-18	2.0900	4.4400	2.0000	3.1000	6.039882e+06	2020-08-18 23:59:59.999000064	1.837537e+07	22593	3458952.50	1.059674e+07	0
1	2020-08-19	3.1000	3.1999	2.6400	2.9500	1.589577e+07	2020-08-19 23:59:59.999000064	4.755832e+07	75075	9037381.99	2.715237e+07	0
2	2020-08-20	2.9449	3.1111	2.6000	2.9000	1.104086e+07	2020-08-20 23:59:59.999000064	3.145741e+07	53797	5189595.84	1.489009e+07	0
3	2020-08-21	2.8978	3.0900	2.6826	2.8330	9.765154e+06	2020-08-21 23:59:59.999000064	2.839354e+07	50378	5164025.18	1.503006e+07	0
4	2020-08-22	2.8387	4.5500	2.7820	4.4546	4.161148e+07	2020-08-22 23:59:59.999000064	1.580389e+08	289179	23427260.15	8.896860e+07	0
...	...	...	...	...	...	...	...	...	...	...	...	...
520	2022-01-20	24.0600	25.6200	22.8400	23.0100	6.578669e+06	2022-01-20 23:59:59.999000064	1.603580e+08	278348	3149652.41	7.692144e+07	0
521	2022-01-21	23.0100	23.4800	19.0400	19.6700	1.726921e+07	2022-01-21 23:59:59.999000064	3.714327e+08	599228	7995489.09	1.719897e+08	0
522	2022-01-22	19.6600	20.1600	16.1700	18.2700	2.323922e+07	2022-01-22 23:59:59.999000064	4.240114e+08	629097	11275865.24	2.058392e+08	0
523	2022-01-23	18.2700	19.2000	17.7300	18.8100	1.148338e+07	2022-01-23 23:59:59.999000064	2.120800e+08	338840	5761415.86	1.064404e+08	0
524	2022-01-24	18.8100	18.8200	17.0000	17.1800	3.830104e+06	2022-01-24 23:59:59.999000064	6.787446e+07	113378	1808926.27	3.207023e+07	0

525 rows × 12 columns

Tiếp theo:

```
# Remove cols not used
df = data.iloc[:,[0,4]]
# Set DATE to index
df.set_index("Open time", inplace = True)
df.index.name = "Date"
df.columns = ["Price"]
df
```

✓ 0.5s

Gọi một biến mới *df,* dataframe này sẽ chỉ lấy **Giá đóng cửa** của ngày làm giá đại diện cho ngày. Tiếp theo chọn ngày làm cột index và đổi tên thành “Date”, đổi tên cột cũ là “Close” thành “Price”.

Kết quả:

	Price
Date	
2020-08-18	3.1000
2020-08-19	2.9500
2020-08-20	2.9000
2020-08-21	2.8330
2020-08-22	4.4546
...	...
2022-01-20	23.0100
2022-01-21	19.6700
2022-01-22	18.2700
2022-01-23	18.8100
2022-01-24	17.1000

525 rows × 1 columns

4. Kiểm tra dữ liệu

Tiến hành thực hiện các bước kiểm tra dữ liệu cơ bản :

```
df.info()
✓ 0.4s
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 525 entries, 2020-08-18 to 2022-01-24
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Price   525 non-null    float64
dtypes: float64(1)
memory usage: 8.2 KB
```

Dữ liệu không có dữ liệu bị Na hay null ở trong dataframe và đang ở dưới dạng float64

```
df.describe()
✓ 0.5s
```

	Price
count	525.000000
mean	22.581282
std	13.730183
min	2.833000
25%	6.271800
50%	23.868000
75%	34.130000
max	53.820000

Sử dụng hàm describe() để xem xét nếu có các yếu tố bất thường của dữ liệu. Xem xét được các giá trị lớn nhất và nhỏ nhất của giá Polkadot. ATH là 53.82 usd và ATL 2.83.

Giá trị trung bình và trung vị khá tương đồng.

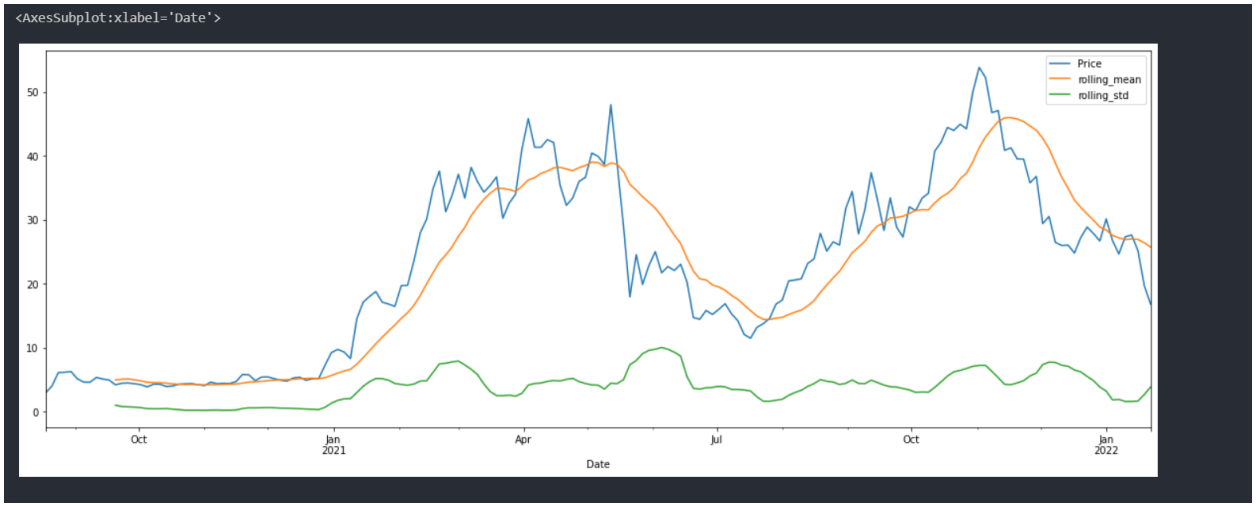
5. Giá trị trung bình cuộn và standard deviation

```
# Rolling mean and std regard 3 days
df_check = df.resample("3D").last()

window_size = 12
df_check["rolling_mean"] = df_check["Price"].rolling(window = window_size).mean()
df_check["rolling_std"] = df_check["Price"].rolling(window = window_size).std()

df_check.plot()
```

✓ 0.3s

Cuộn dữ liệu về dạng 3 ngày và lấy giá trị cuối cùng trong 3 ngày đó . Lý do phải đưa về dạng 3 ngày vì nếu để ở dạng ngày hình ảnh hiện thị trên plot sẽ rất chằng chịt và khó nhìn. Lý do thứ hai vì hạn chế của bộ dữ liệu là chỉ có 525 rows nên lấy theo tuần sẽ không đáp ứng được yêu cầu của hàm seasonal_decompose sẽ áp dụng ở phần 6 (yêu cầu tối thiểu 104 giá trị).

Kết quả:



Giá có xu hướng đi lên và tăng dần theo thời gian khi lần lượt đạt các giá trị cao hơn theo thời gian. Tuy nhiên, giá cả có sự bất ổn khi độ lệch chuẩn tăng đột biến ở nhiều thời điểm

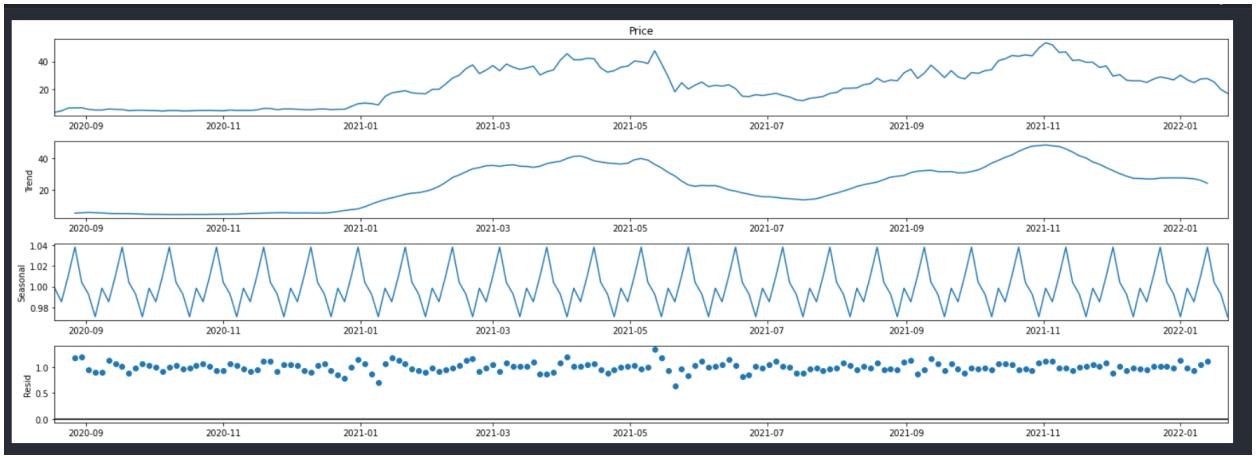
6. Phân rã chuỗi thời gian

```
# Decompose data to check seasonal by using multiplicative model
decompose_results = seasonal_decompose(df_check["Price"], model = "multiplicative") #addictive
decompose_results.plot()
plt.tight_layout()
plt.show()
```

✓ 0.5s

Gọi hàm seasonal_decompose đưa vào series về giá của dữ liệu, chọn mô hình multiplicative (mô hình tổng hợp).

Hiển thị kết quả:



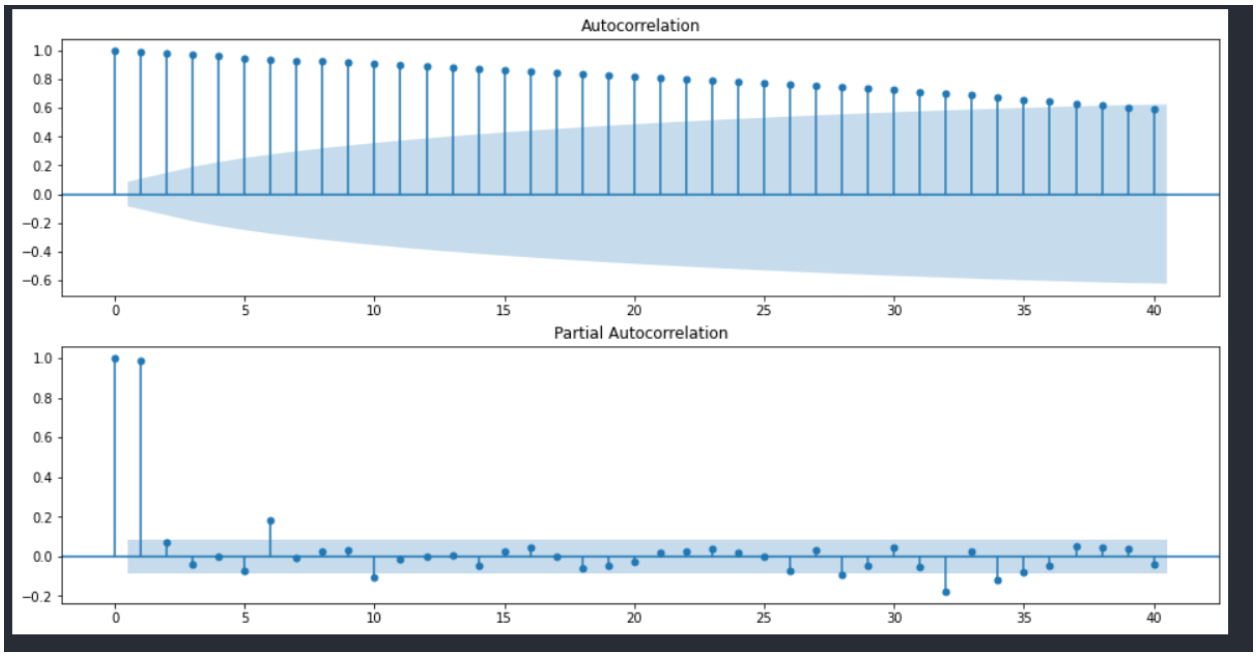
Đường trend thể hiện xu hướng tổng quát là giá trị tăng theo thời gian. Biểu đồ Seasonal Dữ liệu có tình thời vụ và chu kì lặp lại theo tháng là rất đúng và rõ ràng..

7. Kiểm tra dữ liệu test autocorrelation

```python
def test_autocorrelation(x, n_lags=40, alpha=0.05, h0_type='c'):
    '''
    Function for testing the stationarity of a series by using:
    * the ADF test
    * the KPSS test
    * ACF/PACF plots

    Parameters
    ----------
    x: pd.Series / np.array
        The time series to be checked for stationarity
    n_lags : int
        The number of lags for the ACF/PACF plots
    alpha : float
        Significance level for the ACF/PACF plots
    h0_type: str{'c', 'ct'}
        Indicates the null hypothesis of the KPSS test:
            * 'c': The data is stationary around a constant(default)
            * 'ct': The data is stationary around a trend

    Returns
    -------
    fig : matplotlib.figure.Figure
        Figure containing the ACF/PACF plot
    '''
    fig, ax = plt.subplots(2, figsize=(16, 8))
    plot_acf(x, ax=ax[0], lags=n_lags, alpha=alpha)
    plot_pacf(x, ax=ax[1], lags=n_lags, alpha=alpha)

    return fig
```

Sử dụng hàm auto_correlation để kiểm tra tự tương quan trong dữ liệu

Kết quả:



Kết quả từ hình ACF cho ta thấy dữ liệu có sự tự tương quan. Cùng với đó hình PACF cũng cho thấy sự tương quan với độ trễ.

8. Sai phân để chuỗi dữ liệu có tính dừng

```python
# Sai phan du lieu bac 1

df_diff = df_check.diff(1)
df_diff.dropna(inplace =True)
fig, ax = plt.subplots(2, sharex =True)
df_check["Price"].plot(ax =ax[0], title ="DOT")
df_check['Price'].diff().plot(ax= ax[1], title ="Sai phan bac 1")
```

✓ 0.2s Python

Sai phân dữ liệu đã được từ biến df_check. Chọn hàm diff(1) để sai phân bậc 1. Loại bỏ hàng thừa sau khi biến đổi vẽ hình kết quả.

Kết quả:

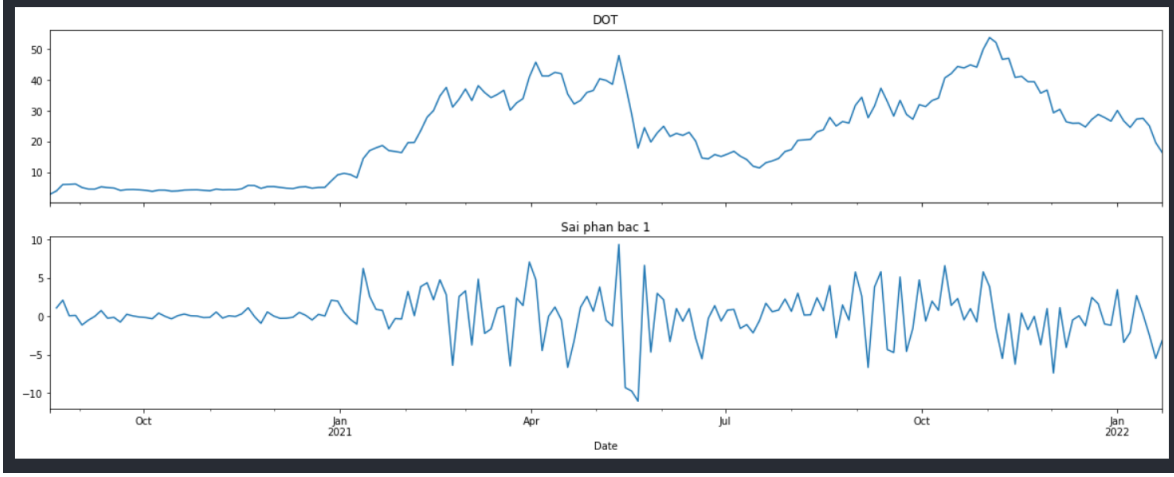
```
df_diff
```

✓ 0.4s

Date	Price	rolling_mean	rolling_std
2020-09-23	0.2514	0.126158	-0.209529
2020-09-26	0.0362	0.039350	-0.049349
2020-09-29	-0.1033	-0.143667	-0.043183
2020-10-02	-0.1698	-0.163025	-0.081752
2020-10-05	-0.3253	-0.198942	-0.152117
...	...	...	...
2022-01-10	2.6900	-0.265000	-0.319189
2022-01-13	0.2800	0.096667	0.006348
2022-01-16	-2.4500	-0.067500	0.063781
2022-01-19	-5.4800	-0.529167	1.022923
2022-01-22	-3.1700	-0.690000	1.239732

163 rows × 3 columns

```
<AxesSubplot:title={'center':'Sai phan bac 1'}, xlabel='Date'>
```



Kết quả cho thấy sau khi sai phân dữ liệu có vẻ có tính dừng. Khi chuyển động xung quanh trục 0. Tuy nhiên vẫn cần phải có các bước kiểm định ý nghĩa thống kê để chắc chắn.

9. Kiểm định tính dừng của chuỗi dữ liệu

```python
# Ham Kiem dinh ADF
def adf_test(x):
    indices = ["Test Statistics", "p-value",
               "# of Lags Used","of Observations Used"]
    adf_test = adfuller(x)
    results = pd.Series(adf_test[0:4], index = indices)
    for key, value in adf_test[4].items():
        results[f"Critical Value ({key})"] = value
    return results

# Ham Kiem dinh KPSS
def kpss_test(x):
    indices = ["Test Statistics", "p-value",
               "# of Lags"]
    kpss_test = kpss(x)
    results = pd.Series(kpss_test[0:3], index = indices)
    for key, value in kpss_test[3].items():
        results[f"Critical Value ({key})"] = value
    return results


#Kiem dinh ADF sai phan bac 1
print(f"Kiem dinh ADF:\n{adf_test(df_diff['Price'])}")

print("---------------------------------------")

#Kiem dinh KPSS sai phan bac 1
print(f"Kiem dinh KPSS:\n{kpss_test(df_diff['Price'])}")
```

Viết 2 hàm để trực quan hóa hơn kết quả từ 2 hàm được viết sẵn là adf_test và kpss_test.

In kết quả kiểm định:

```
Kiem dinh ADF:
Test Statistics        -1.276837e+01
p-value                 7.867691e-24
# of Lags Used          0.000000e+00
of Observations Used    1.620000e+02
Critical Value (1%)    -3.471374e+00
Critical Value (5%)    -2.879552e+00
Critical Value (10%)   -2.576373e+00
dtype: float64
---------------------------------------
Kiem dinh KPSS:
Test Statistics           0.172421
p-value                   0.100000
# of Lags                14.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

Kết quả từ ADF test:

p-value < 0.05 => Bác bỏ H0 : Dữ liệu sai phân bậc 1 là không dừng

Kết quả từ KPSS test :

p-value > 0.05 ⇒ Chấp nhận H0: Dữ liệu sai phân bậc 1 có tính dừng

Từ 2 kiểm định trên ta có thể nói dữ liệu sai phân bậc nhất giá polkadot có tính dừng ⇒ **d = 1**

10. Chọn p d q

```
# Train & test split
to_row = int(len(df)*0.9)

train =list(df[0:to_row]["Price"])
test =list(df[to_row:]["Price"])


# Using autoarima to choose p d q with AIC smallest

import pmdarima as pm
auto_arima = pm.auto_arima(df[0:to_row]["Price"], trace =1,
                           error_action ="ignore",
                           suppress_warnings =True,
                           seasonal = False,
                           stepwise =True,
                           approximation =False,
                           n_jobs =-1,
                           seasonal_test =None)
```

0.6s Python

```
# Using plotly.express
import plotly.graph_objects as go
import plotly.express as px


# Visualization
fig = go.Figure()
fig.add_trace(go.Scatter(x= df[0:to_row]["Price"].index, y= df[0:to_row]["Price"],
                         mode='lines+markers',
                         name='Train data'))
fig.add_trace(go.Scatter(x=df[to_row:]["Price"].index, y=df[to_row:]["Price"],
                         mode='lines+markers',
                         name='Test data'))
# Edit the layout
fig.update_layout(title='DOT ACTUAL PRICES', title_x =0.5,
                  xaxis_title='Date',
                  yaxis_title='CLOSING PRICES')

fig.show()
```

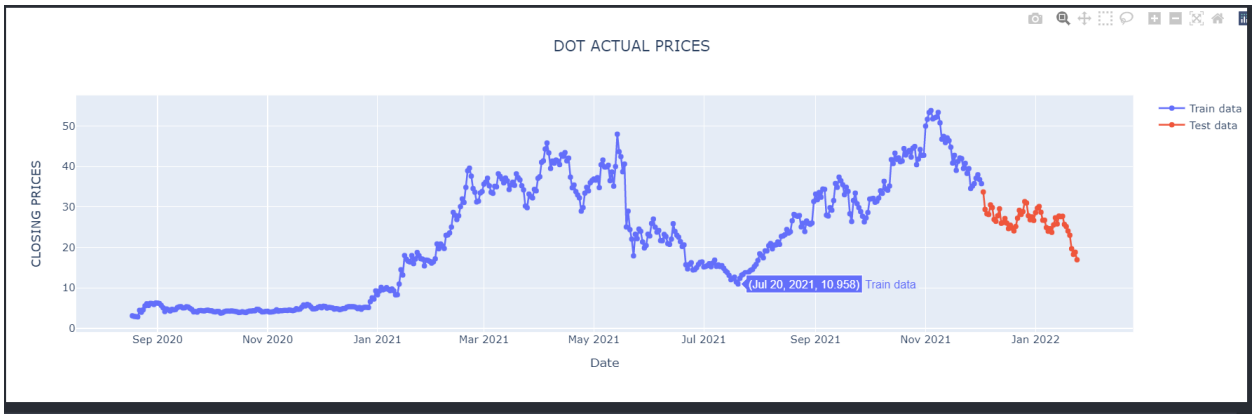
0.1s Python

Ở phần này, ta sẽ chia tập dữ liệu theo tỉ lệ 90% train và 10% test.

Sử dụng hàm auto_arima từ pmdarima để chọn được *p,d,q* với AIC là bé nhất và phù hợp nhất

Kết quả:

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=1923.570, Time=0.42 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1926.375, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1923.816, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1924.184, Time=0.04 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1925.027, Time=0.01 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=1927.026, Time=0.27 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=1921.877, Time=0.17 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1919.878, Time=0.10 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=1925.153, Time=0.05 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=1925.078, Time=0.05 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=1918.613, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=1922.983, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=1922.614, Time=0.03 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=1920.613, Time=0.10 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=1925.725, Time=0.09 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=1923.882, Time=0.04 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=1923.810, Time=0.02 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=1922.444, Time=0.10 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0]
Total fit time: 1.626 seconds
```



Kết quả lựa chọn cho ra mô hình ARIMA(1,1,1). Trong auto Arima ta tạm thời chấp nhận việc sử dụng nó để trả về các thông số cho mô hình ARIMA.

11.Xây dựng mô hình dự báo

```python
# Build model prediction

model_predictions = []
test_obser = len(test)

for i in range(test_obser):
    model = sm.tsa.arima.ARIMA(train, order=(1, 1, 1))
    model_fit = model.fit()
    output = model_fit.forecast()
    model_predictions.append(output[0])
    train.append(test[i])
```

✓ 3.5s Python

Đây là mô hình được xây dựng dựa trên tập dữ liệu *train* với mô hình ARIMA(1,1,1).

Khởi tạo list *model predictions* để lưu trữ các giá trị được dự báo. Số quan sát được dự báo sẽ là số quan sát của tập *test.*

Sử dụng vòng for để mỗi giá trị được dự báo sẽ được lưu trữ vào trong list *model predictions*, cùng với đó qua mỗi vòng dữ liệu từ tập test cũng sẽ được thêm vào tập *train* để dự báo giá tiếp theo.

Kết quả:

```
#Print summary
print(model_fit.summary())
```

✓ 0.4s

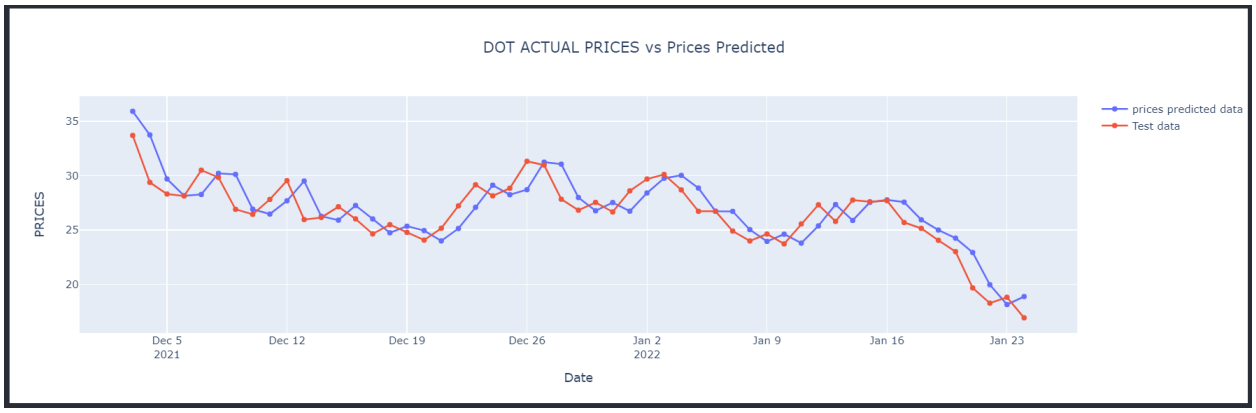
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  524
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -1034.498
Date:                Mon, 24 Jan 2022   AIC                           2074.996
Time:                        23:39:13   BIC                           2087.775
Sample:                             0   HQIC                          2080.001
                                - 524
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8445      0.083    -10.210      0.000      -1.007      -0.682
ma.L1          0.7646      0.097      7.905      0.000       0.575       0.954
sigma2         3.0588      0.074     41.358      0.000       2.914       3.204
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              3835.95
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):               4.98   Skew:                            -0.87
Prob(H) (two-sided):                  0.00   Kurtosis:                        16.15
===================================================================================
```

Kết quả cho thấy với 524 quan sát được sử dụng thì cho ra mô hình có AIC là 2074. Các biến số đều có p-value <0.05 ⇒ đều có ý nghĩa thống kê.

Trực quan giá trị dự báo:

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=df[to_row:]["Price"].index, y= model_predictions,
                         mode='lines+markers',
                         name='prices predicted data'))
fig.add_trace(go.Scatter(x=df[to_row:]["Price"].index, y=df[to_row:]["Price"],
                         mode='lines+markers',
                         name='Test data'))
# fig.add_trace(go.Scatter(x=df[to_row:]["Close"].index, y=df[to_row:]["Close"],
#                          mode='lines+markers',
#                          name='Test data'))
# Edit the layout
fig.update_layout(title='DOT ACTUAL PRICES vs Prices Predicted', title_x =0.5,
                  xaxis_title='Date',
                  yaxis_title='PRICES')

fig.show()
```

✓ 0.6s Python



Giá trị dự báo là tương đối sát nhưng vẫn có độ trễ.

12. Đánh giá mô hình

```
#Evaluate model

mape = np.mean(np.abs((np.array(model_predictions) - np.array(test))/np.abs(test)))
print("MAPE:"+ str(mape))
```

✓ 0.4s Python

```
MAPE:0.05369894102521619
```

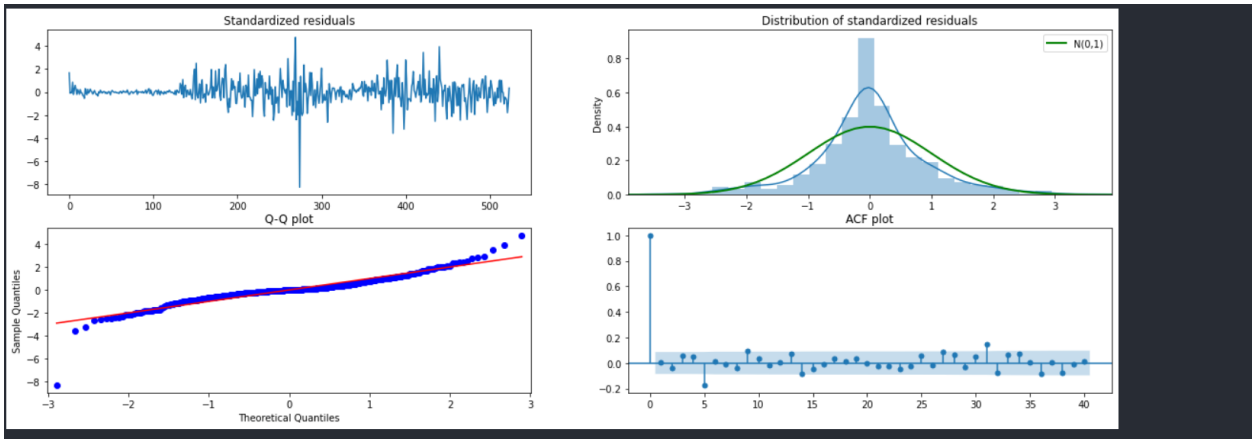
Sử dụng chỉ số MAPE (Mean Absolute Percentage Error) để đánh giá khả năng dự đoán.

MAPE cho ra kết quả dự báo có sai số 5,3% so với kết quả thực tế.

15. Kiểm định độ tin cậy bằng phân tích phần dư

```python
# Kiểm định độ tin cậy bằng phân tích phần dư
def arima_diagnostics(resids, n_lags=40):
 # create placeholder subplots
 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
 r = resids
 resids = (r - np.nanmean(r)) / np.nanstd(r)
 resids_nonmissing = resids[~(np.isnan(resids))]
 # residuals over time
 sns.lineplot(x=np.arange(len(resids)), y=resids, ax=ax1)
 ax1.set_title('Standardized residuals')
 # distribution of residuals
 x_lim = (-1.96 * 2, 1.96 * 2)
 r_range = np.linspace(x_lim[0], x_lim[1])
 norm_pdf = scs.norm.pdf(r_range)
 sns.distplot(resids_nonmissing, hist=True, kde=True,
 norm_hist=True, ax=ax2)
 ax2.plot(r_range, norm_pdf, 'g', lw=2, label='N(0,1)')
 ax2.set_title('Distribution of standardized residuals')
 ax2.set_xlim(x_lim)
 ax2.legend()
 # Q-Q plot
 qq = sm.qqplot(resids_nonmissing, line='s', ax=ax3)
 ax3.set_title('Q-Q plot')
# ACF plot
 plot_acf(resids, ax=ax4, lags=n_lags, alpha=0.05)
 ax4.set_title('ACF plot')
 return plt.show()


arima_diagnostics(model_fit.resid,40)
```

Kết quả:



Phần dư có tính dừng quay quanh 0. Giá trị kì vọng của phần dư là 0

QQplot có các điểm đều nằm tương đối trên đường thẳng => có phân phối chuẩn. Phần dư có phân phối là phân phối chuẩn N(0,1)

Mô hình tương đối ổn.

KẾT LUẬN

ARIMA là một mô hình phổ biến và có khả năng dự đoán chính xác khá cao.

Tuy nhiên ARIMA chỉ dựa trên tín hiệu của quá khứ của chuỗi dữ liệu vậy nên không thể đo lường được các yếu tố khác trên thị trường crypto.

SOURCE CODE

```
## Import LIB

# !pip install binance


from binance import Client

import pandas as pd, numpy as np, matplotlib.pyplot as plt

import mplfinance as mpf

import math

from statsmodels.tsa.arima_model import ARIMA

from sklearn.metrics import mean_squared_error, mean_absolute_error


import seaborn as sns

from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.tsa.stattools import adfuller, kpss

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf


import warnings

warnings.filterwarnings('ignore')
```

```
import scipy.stats as scs
import statsmodels.api as sm


# Set figsize of plot
plt.rcParams["figure.figsize"] = (20, 7)
# Get public key and private key from Binance (API Management)
##GET historical data
api_key =
"NT1J5dc36vJnNtzoqGK6JuJcJI7oqEYKzYYGsZcswFaYOA5SrxnDWwvaX33qsZmg"
secret ="IqN8nyddAUEcB10V4nzXFTjiEFPsK1PZUvN5C7XZhMJyIBDOPVPxmWuyfKpetLKh"
client = Client(api_key, secret)


#Get data
historical = client.get_historical_klines("DOTUSDT",
Client.KLINE_INTERVAL_1DAY,"1 Jan 2013")
data = pd.DataFrame(historical)


data
## Tranform data
#Set index and name for columns
data.columns = ["Open time",'Open','High','Low','Close','Volume','Close
time','Quote Asset Volume','Number of trade','tb base volume','tb quote
volume','ignore']
```

```
#Convert open time and close time columns from num to datatime

data['Open time']= pd.to_datetime(data['Open time']/1000, unit ='s')

data['Close time']= pd.to_datetime(data['Close time']/1000, unit ='s')


#Tranform the others to numeric

numeric_columns = ['Open','High','Low','Close','Volume','Quote Asset
Volume','tb base volume','tb quote volume']

data[numeric_columns] = data[numeric_columns].apply(pd.to_numeric, axis
=1)


data


# Remove cols not used

df = data.iloc[:,[0,4]]

# Set DATE to index

df.set_index("Open time", inplace = True)

df.index.name = "Date"

df.columns = ["Price"]

df

##Check data

df.info()


df.describe()
```

```
## Rolling mean and std

# Rolling mean and std regard 3 days

df_check = df.resample("3D").last()


window_size = 12

df_check["rolling_mean"] = df_check["Price"].rolling(window =
window_size).mean()

df_check["rolling_std"] = df_check["Price"].rolling(window =
window_size).std()


df_check.plot()


##Decompose time series data

# Decompose data to check seasonal by using multiplicative model

decompose_results = seasonal_decompose(df_check["Price"], model =
"multiplicative") #addictive

decompose_results.plot()

plt.tight_layout()

plt.show()


# Sai phan du lieu bac 1


df_diff = df_check.diff(1)

df_diff.dropna(inplace =True)

fig, ax = plt.subplots(2, sharex =True)
```

```
df_check["Price"].plot(ax =ax[0], title ="DOT")

df_check['Price'].diff().plot(ax= ax[1], title ="Sai phan bac 1")

df_diff

### Correlation test

def test_autocorrelation(x, n_lags=40, alpha=0.05, h0_type='c'):

    '''

    Function for testing the stationarity of a series by using:

    * the ADF test

    * the KPSS test

    * ACF/PACF plots



    Parameters

    ----------

    x: pd.Series / np.array

        The time series to be checked for stationarity

    n_lags : int

        The number of lags for the ACF/PACF plots

    alpha : float

        Significance level for the ACF/PACF plots

    h0_type: str{'c', 'ct'}

        Indicates the null hypothesis of the KPSS test:

            * 'c': The data is stationary around a constant(default)

            * 'ct': The data is stationary around a trend
```

```
    Returns
    -------
    fig : matplotlib.figure.Figure
        Figure containing the ACF/PACF plot
    '''
    fig, ax = plt.subplots(2, figsize=(16, 8))
    plot_acf(x, ax=ax[0], lags=n_lags, alpha=alpha)
    plot_pacf(x, ax=ax[1], lags=n_lags, alpha=alpha)

    return fig



# using function test_autocorrelation to check autocorrelation
fig = test_autocorrelation(df_check["Price"])


### ADF and KPSS test
# Ham Kiem dinh ADF
def adf_test(x):
    indices = ["Test Statistics", "p-value",
               "# of Lags Used","of Observations Used"]
    adf_test = adfuller(x)
    results = pd.Series(adf_test[0:4], index = indices)
    for key, value in adf_test[4].items():
        results[f"Critical Value ({key})"] = value
```

```
    return results


# Ham Kiem dinh KPSS
def kpss_test(x):
    indices = ["Test Statistics", "p-value",
                "# of Lags"]
    kpss_test = kpss(x)
    results = pd.Series(kpss_test[0:3], index = indices)
    for key, value in kpss_test[3].items():
        results[f"Critical Value ({key})"] = value
    return results



#Kiem dinh ADF sai phan bac 1
print(f"Kiem dinh ADF:\n{adf_test(df_diff['Price'])}")


print("---------------------------------------")


#Kiem dinh KPSS sai phan bac 1
print(f"Kiem dinh KPSS:\n{kpss_test(df_diff['Price'])}")
```

```
### Choose p d q

# Train & test split

to_row = int(len(df)*0.9)


train =list(df[0:to_row]["Price"])

test =list(df[to_row:]["Price"])




# Using autoarima to choose p d q with AIC smallest


import pmdarima as pm

auto_arima = pm.auto_arima(df[0:to_row]["Price"], trace =1,

                           error_action ="ignore",

                           suppress_warnings =True,

                           seasonal = False,

                           stepwise =True,

                           approximation =False,

                           n_jobs =-1,

                           seasonal_test =None)



# Using plotly.express

import plotly.graph_objects as go

import plotly.express as px
```

```
# Visualization

fig = go.Figure()

fig.add_trace(go.Scatter(x= df[0:to_row]["Price"].index, y=
df[0:to_row]["Price"],

                    mode='lines+markers',

                    name='Train data'))

fig.add_trace(go.Scatter(x=df[to_row:]["Price"].index,
y=df[to_row:]["Price"],

                    mode='lines+markers',

                    name='Test data'))

# Edit the layout

fig.update_layout(title='DOT ACTUAL PRICES', title_x =0.5,

                  xaxis_title='Date',

                  yaxis_title='CLOSING PRICES')


fig.show()



### Build model prediction


model_predictions = []

test_obser = len(test)


for i in range(test_obser):
```

```
    model = sm.tsa.arima.ARIMA(train, order=(1, 1, 1))
    model_fit = model.fit()
    output = model_fit.forecast()
    model_predictions.append(output[0])
    train.append(test[i])


#Print summary
print(model_fit.summary())


# Visualization
fig = go.Figure()
fig.add_trace(go.Scatter(x=df[to_row:]["Price"].index, y=
model_predictions,
                    mode='lines+markers',
                    name='prices predicted data'))
fig.add_trace(go.Scatter(x=df[to_row:]["Price"].index,
y=df[to_row:]["Price"],
                    mode='lines+markers',
                    name='Test data'))
fig.update_layout(title='DOT ACTUAL PRICES vs Prices Predicted', title_x
=0.5,
                  xaxis_title='Date',
                  yaxis_title='PRICES')


fig.show()
```

```
###Evaluate model

mape = np.mean(np.abs((np.array(model_predictions) -
np.array(test))/np.abs(test)))

print("MAPE:"+ str(mape))


### Kiểm định độ tin cậy bằng phân tích phần dư

def arima_diagnostics(resids, n_lags=40):

 # create placeholder subplots

 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)

 r = resids

 resids = (r - np.nanmean(r)) / np.nanstd(r)

 resids_nonmissing = resids[~(np.isnan(resids))]

 # residuals over time

 sns.lineplot(x=np.arange(len(resids)), y=resids, ax=ax1)

 ax1.set_title('Standardized residuals')

 # distribution of residuals

 x_lim = (-1.96 * 2, 1.96 * 2)

 r_range = np.linspace(x_lim[0], x_lim[1])

 norm_pdf = scs.norm.pdf(r_range)

 sns.distplot(resids_nonmissing, hist=True, kde=True,

 norm_hist=True, ax=ax2)

 ax2.plot(r_range, norm_pdf, 'g', lw=2, label='N(0,1)')

 ax2.set_title('Distribution of standardized residuals')

 ax2.set_xlim(x_lim)
```

```
 ax2.legend()

 # Q-Q plot

 qq = sm.qqplot(resids_nonmissing, line='s', ax=ax3)

 ax3.set_title('Q-Q plot')

# ACF plot

 plot_acf(resids, ax=ax4, lags=n_lags, alpha=0.05)

 ax4.set_title('ACF plot')

 return plt.show()


arima_diagnostics(model_fit.resid,40)
```

TÀI LIỆU THAM KHẢO

Phong, N.A và cộng sự (2020). Sách tham khảo "Ứng dụng Python trong tài chính", NXB Đại học Quốc Gia Tp.HCM

Francesco Rundo at el (2019), Machine Learning for Quantitative Finance Applications: A Survey, Applied Sciences 9(24):1-20

Khoa học dữ liệu (phamdinhkhanh.github.io)

Exploratory Data Analysis with the Binance API using Python and Pandas | MLTrader EP1 - YouTube

Using ARIMA to Predict Bitcoin Prices in Python in 2022🔴 - YouTube