VIETNAM NATIONAL UNIVERSITY
UNIVERSITY OF INFORMATION TECHNOLOGY
INFORMATION SYSTEMS FACULTY

REPORT LAB 4
SUBJECT: DATA ANALYSIS

Lecturer : Assoc. Prof. Nguyen Dinh Thuan
Instructor : TA. Nguyen Minh Nhut
Class : STAT3013.O12.CTTT
Group 11 : Pham Thi Thuy Trang - 21522697
: Tran Ngoc Khoi Nguyen - 21522398
: Chau Nguyen Thanh Tai - 21522561

Ho Chi Minh City, December 2023

TABLE OF CONTENT

TEACHER'S COMMENTS

WORK DISTRIBUTION

Works \ Members		Pham Thi Thuy Trang (Leader)	Tran Ngoc Khoi Nguyen	Chau Nguyen Thanh Tai
Problem statement		✓	✓	✓
Build the report template		✓	✓	
Task 1	*ACF*			✓
	PACF		✓	
	ARIMA	✓		
Task 2	*ACF*			✓
	PACF		✓	
	ARIMA	✓		
Summarize and edit reports		✓	✓	✓
Completion (%)		100%	100%	100%

A. TASK 1:

I. ACF:

1. What is ACF?

Autocorrelation is a mathematical representation of the degree of similarity between a given time series and a lagged version of itself over successive time intervals. It's conceptually like the correlation between two different time series, but autocorrelation uses the same time series twice: once in its original form and once lagged one or more time periods.

Naturally, autocorrelation can be a useful tool for traders to utilize, particularly for technical analysts.[1]

2. How to find ACF?

This is a statistical method used to identify the presence of autocorrelation in time series data. It measures the correlation between a time series and a lagged version of itself. In the context of time series analysis, the ACF is often used to determine the order of an autoregressive (AR) model.

$$ACF(k) = \frac{\sum_{t=k+1}^{n}(x_t - \mu)(x_{t-k} - \mu)}{\sum_{t=1}^{n}(x_t - \mu)^2}$$

Where:

- **n**: the length of time series.
- **x(t)**: the value at time t.
- **μ**: the mean of time series.

3. Why do we need to use ACF?

Given that ACF is essential for modeling AR, MA, and other concepts, we shall outline a few advantages:

- Model Identification: the ACF plot dictates the order in which the AR model is presented. Observing the prominent autocorrelation patterns in the graphs can help determine which lags are appropriate to incorporate in the model...

- Diagnosis of the Model: The ACF plot can also be used to determine a time series model's quality of fit. Unusual patterns in ACF graphs may suggest that some significant autocorrelation in the data is missing from the model.

- Forecasting: By illuminating the underlying patterns in the time series data, ACF can help forecast future values.

- In summary, ACF is a valuable tool in time series analysis for understanding the autocorrelation structure of a time series, identifying appropriate model orders, and diagnosing model adequacy.

4. Example of ACF:

- In real life: [2]

Autocorrelation can be used to find repeating patterns, such as the presence of a periodic signal obscured by noise or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies.

In fact, ACF is applied in many fields:

In finance, stock traders and analysts use autocorrelation function to understand the degree of similarity, moving patterns, comprehend the impact of past prices, and predict future prices.

In meteorology, meteorologists and climate researchers use autocorrelation functions to study temperature rise in a weather report.

In physics and engineering, autocorrelation functions have various applications.

In health and medicine, autocorrelation functions have applications in medical technology and research.

- In mathematics:

ACF is an important tool for understanding and analyzing the quality of time series, especially when you are doing modeling and forecasting in the field of time forecasting.

To summary, the autocorrelation function is a statistical tool that can be used to examine time series data. It enables us to examine the correlation between samples within a single time series and their later variants. To comprehend the samples and the characteristics of the time series, as well as to model time series data, one can utilize the correlation function to ascertain whether latency has a substantial correlation. Finding repeating samples, such as periodic signals hidden by noise or the fundamental frequency absent from the suggested signal, can be accomplished by automatic correlation.

II. PACF:

1. What is PACF?

PACF (Partial Autocorrelation Function) which is a partial correlation between observations at time **t** and previous times. PACF shows the correlation between Y_t and Y_{t-k} (with k is lag), ignoring the independence $Y_{t-1}, Y_{t-2}, \ldots, Y_{t-k+1}$, so Y_t is considered as a constant, $Y_t = y_t$, t = $t+1, t+2, \ldots, t+k-1$. [3]

⇨ In the ACF model, we have variables that are correlated with each other in an indirect form. Additionally, with PACF, for each lag, we can find a direct correlation from the observation at time t to the observation at a specific time t-k. PACF basically eliminates the effects of other delays.

PACF is also defined for positive lag only, their value also lies between -1 and +1. [4]

2. How to find PACF?

The partial autocorrelation function (PACF), defined as:

$PACF(k) = r_{kk} = corr(Y_t, Y_{t-k} \mid Y_{t-1}, Y_{t-2}, \ldots, Y_{t-k+1})$ [5]

$$= \frac{\mathbf{Cov}(y_t, y_{t+k} | y_{t+1}, y_{t+2}, \cdots, y_{t+k-1})}{\sqrt{\boldsymbol{Var}(y_t | y_{t+1}, y_{t+2}, \cdots, y_{t+k-1}) \boldsymbol{Var}(y_{t+1} | y_{t+1}, y_{t+2}, \cdots, y_{t+k-1})}}$$

Where:

- Cov(X,Y|Z) is the covariance between the random variables X and Y conditioned on the random variable Z.
- Var(X|Z) is the variance of X conditioned on Z. [6]

We can concretize the above formula through four variables y, x_1, x_2, x_3. Consider a regression context in which y is the response variable and x_1, x_2, x_3 are predictor variables. The partial correlation between y and x_3 is the correlation between the variables determined taking into account how both y and x_3 are related to x_1 and x_2.
In regression, this partial correlation could be found by correlating the residuals from two different regressions:
1. Regression in which we predict y from x_1 and x_2.
2. Regression in which we predict x_3 from x_1 and x_2.
More formally, we can define the partial correlation just described as:

$$\frac{\mathbf{Cov}(y, x_3 \mid x_1, x_2)}{\sqrt{\boldsymbol{Var}(y \mid x_1, x_2)\boldsymbol{Var}(x_3 \mid x_1, x_2)}}$$

⇨ Going back, we can interpret the above mentioned formula based on time series:
The partial autocorrelation between Y_t and Y_{t-k} is defined as the conditional correlation between Y_t and Y_{t-k}, conditional on Y_{t-k+1}, ... , Y_{t-1}, the set of observations that come between the time points t and k.

- The 1st order partial autocorrelation will be defined to equal the 1st order autocorrelation.
- The 2nd order (lag) partial autocorrelation is:

$$\frac{\mathbf{Cov}(Y_t, Y_{t-2} \mid Y_{t-1})}{\sqrt{\boldsymbol{Var}(Y_t \mid Y_{t-1})\boldsymbol{Var}(Y_{t-2} \mid Y_{t-1})}}$$

This is the correlation between values two time periods apart conditional on knowledge of the value in between.

- The 3rd order (lag) partial autocorrelation is:

$$\frac{\mathbf{Cov}(Y_t, Y_{t-3} \mid Y_{t-1}, Y_{t-2})}{\sqrt{\boldsymbol{Var}(Y_t \mid Y_{t-1}, Y_{t-2})\boldsymbol{Var}(Y_{t-3} \mid Y_{t-1}, Y_{t-2})}}$$

- **And, so on, for any lag.** [7]

3. Why we need to use PACF?

We all already know that if the current value Y_t can be explained with a function of p past values, we can model the time series with an AR(p) model. [8]
With PACF, PACF is used specifically to diagnose autoregression models because PACF can find the order p for the AR model. Order p is estimated by selecting the last lag at which the PACF is nonzero. [9]
With an autoregressive model of order p can be written as:
$Y_t = \mathbf{c} + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-k} + \varepsilon_t$ [10]

Properties	AR(p)	MA(q)	ARMA(p,q)
ACF	Decay	Cuts after q lags	Decay
PACF	Cuts after p lags	Decay	Decay

Table 1. PROPERTIES OF ACF & PACF FOR AR, MA & ARMA [11]

⇨ For an AR(p) process, PACF will drop abruptly right after the lag p.
Example: A causal AR(1) process:

$$X_t = \phi X_{t-1} + Z_t$$

⇨ The PACF function measures the degree of correlation between the current value of a variable and its previous values, after removing the influence of other variables. If PACF has a large value at a certain lag, it can be concluded that the current value of the variable is correlated with the value at time *t* - *k*. This relationship can be positive (+) or negative (–).

- If PACF has a positive value at lag 1, it can be concluded that the current value of the variable tends to increase if the value at time t - 1 also increases.

From there, we can determine the lags at which the current value of the variable is correlated with its previous values.

- If the PACF has a value greater than the statistical significance threshold at lag 1, then the degrees of freedom of the AR model are 1. This means that the AR model has an autoregressive variable.

⇨ Therefore, the PACF function is an important tool in determining the degrees of freedom of the AR model. Correctly determining the degrees of freedom of the AR model will help the model have higher accuracy.

Summary: The PACF is a measure of the linear dependency between r_t and r_{t+k} when the dependencies of rt with the intermediate values $r_{t+1},\ldots,r_{t+l-1}$ have already been accounted for. The reason to measure the PACF is that their combined behaviors can be of great aid in identifying if a time series can be modeled as an autoregressive model (AR) or a mixture of both (ARMA). [12]

4. Example of PACF:

The Partial Autocorrelation Function (PACF) of a zero-mean stationary TS $\{X_t\}$t=0,1,... is defined as:

$$\phi_{11} = corr(X_1, X_0) = \boldsymbol{p(1)}$$

$$\phi_{11} = corr(X_\tau - f_{(\tau-1)}, X_0 - f_{(\tau-1)}), \qquad \tau \geq 2,$$

Where:

$$f_{(\tau-1)} = f(X_{\tau-1}, \ldots, X_1)$$

minimizes the mean square linear prediction error:

$$E(X_{\tau-1} - f_{(\tau-1)})^2$$

- The subscript at the f function denotes the number of variables the function depends on.
- By stationarity, ϕ_{11} is the correlation between variables X_t and $X_{t-\tau}$ with the linear effect:

$$f(X_{t-1}, \ldots, X_{t-\tau+1}) = \beta_1 X_{t-1} + \cdots + \beta_{\tau-1} X_{t-\tau+1}$$

on each variable removed.

The PACF of AR(1):

Consider a process:

$X_t = \phi X_{t-1} + Z_t$, $Zt \sim WN(0, \sigma^2)$

where | ϕ | < 1, i.e., a causal AR(1). Then by definition above.

To calculate ϕ_{22} we need to find the function $f_{(1)}$ which is of the form $f_{(1)} = \beta X_1$.

We choose β to minimize $E(X_2 - \beta X_1)^2 = \mathrm{E}(X_2^2 - 2\beta X_1 X_2 + \beta^2 X_1^2)$
$= \gamma(0) - 2\beta\gamma(1) + \beta^2\gamma(0)$
which is a polynomial in β. Taking the derivative with respect to β and setting it equal to zero, we obtain

$$-2\gamma(1) + -2\gamma(0)\beta = 0$$

Hence

$$\beta = \frac{\gamma(1)}{\gamma(0)} = \mathrm{p}(1) = \phi$$

and
$f_{(1)} = \beta X_1$
Then

$$\phi_{22} = corr(X_2 - \phi X_1, X_0 - \phi X_1) = \phi_{22} = corr(Z_2, X_0 - \phi X_1) = \mathbf{0}$$

as by causality X_0, X_1 do not depend on Z_2. Similarly we would obtain $\phi_{33} = 0$.
In fact:
$\phi_{\tau\tau} = 0 \; for \; \tau > 1.$ [13]

III. ARIMA:

1. What is ARIMA?

ARIMA stands for Autoregressive Integrated Moving Average. It is a statistical forecasting method widely used in time series analysis. This model incorporates autoregressive (AR), moving average (MA), and differencing (I) components to capture the relationship between current and past values of a time series. [14]

Autoregressive (AR)

AR is Auto Regression, and p is the number of autoregressive terms [2]. The equation for AR model is: [15]

$$y_t = \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \ldots + \varphi_p y_{t-p} + \delta + \varepsilon_t$$

[16]

In which:

- Y t is the current value
- μ is the constant term
- p is the number of orders
- φ is the autoregressive coefficient
- ε_t is the error

Moving Average (MA)

MA is the Moving Average, and q is the number of terms in the moving average. The equation for MA model is: [15]

$$y_t = \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_p \varepsilon_{t-p} + \mu + \varepsilon_t$$

[16]

Differencing (I)

Last, the I part is Integrated, and d is the number of differences (order) required to make it a stationary sequence. For example:

$$\text{If d} = 0: \ \Delta Y_t = Y_t$$
$$\text{If d} = 1: \ \Delta Y_t = Y_t - Y_{t-1}$$
$$\text{If d} = 2: \ \Delta Y_t = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2}) = Y_t - 2Y_{t-1} + Y_{t-2}$$

[16]

After combining them, we will have the ARIMA (p, d, q) express as follow:

$$\Delta Y_t = \mu + \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + .. + \varphi_p Y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + .. + \theta_p \epsilon_{t-p} + \epsilon_t$$

In summary, ARIMA models capture the relationship between current and past values of a time series, incorporating autoregressive, moving average, and differencing components. They are widely used for forecasting future values of various time series, including economic data, stock prices, and weather patterns.

2. How to know that dataset is suitable to use ARIMA model?

To determine whether such data is suitable for using the ARIMA model, we need to examine the following factors:

- **Stationarity:** Data should have distributed uniformity, meaning that the mean, variance, and self-correlation of the data do not change over time. If the data does not have distributed uniformity, we need to conduct differencing to transform the data into distributed homogeneity.

- **Periodicity:** Data can be periodic, meaning that data tends to repeat in certain cycles. If the data is periodic, we need to calculate the period of the data and add a seasonal component to the ARIMA model.

- **Correlation:** Data should have a correlation between present and past values. If the data is not correlated, the ARIMA model will be ineffective.

There are several methods for testing the above factors, which include:

- **Distributed uniformity testing:** Tests such as Dickey-Fuller test, Augmented Dickey-Fuller test, Phillips-Perron test... can be used.

- **Periodicity testing:** Methods such as Fourier analysis, wavelet analysis,... can be used

- **Correlation testing:** Methods such as self-correlation graph (ACF), redundant self-correlation graph (PACF) can be used, ...

If the data meets all of the above factors, then we can use the ARIMA model to forecast the data.

3. Why we need to use ARIMA?

There are several reasons why ARIMA models are widely used for time series forecasting:

- **Simple and interpretable:** ARIMA models are relatively straightforward to understand and interpret, making them accessible to users with varying levels of statistical expertise.

- **Effective for stationary data:** ARIMA models are particularly well-suited for forecasting stationary time series, which are characterized by consistent statistical properties over time.

- **Adaptability to different time series:** ARIMA models can be customized to accommodate various time series patterns, including seasonal patterns and trends.

- **Computational efficiency:** ARIMA models are computationally efficient, making them suitable for forecasting large datasets and handling real-time data streams.

- **Robustness to noise:** ARIMA models can handle moderate levels of noise in the data, making them suitable for real-world applications where data is often noisy.

- **Versatility in applications:** ARIMA models have been applied to a wide range of time series forecasting problems, including forecasting economic indicators, sales figures, stock prices, and weather patterns.

- **Ease of implementation:** ARIMA models are readily available in various statistical software packages, making them easy to implement and use.

- **Foundation for more advanced models:** ARIMA models serve as a foundation for more sophisticated time series forecasting techniques, such as exponential smoothing and neural network-based models.

Overall, ARIMA models offer a balance of simplicity, effectiveness, and adaptability, making them a valuable tool for time series forecasting. Their ease of use, computational efficiency, and versatility make them widely applicable across various domains.

4. Example of ARIMA:

Example 1: Forecasting Monthly Sales Data

Suppose you have monthly sales data for a retail store for the past five years. You want to use an ARIMA model to forecast sales for the next year.

- **Data Preparation:** Import the sales data into a statistical software package. Check for stationarity by examining the time series plot and performing unit root tests. If the data is not stationary, apply differencing to make it stationary.

- **Model Identification:** Use autocorrelation function (ACF) and partial autocorrelation function (PACF) plots to identify the appropriate order of the AR and MA components (p and q, respectively).

- **Model Estimation:** Estimate the ARIMA model parameters using the identified p, d, and q values. This involves minimizing a loss function such as the sum of squared errors (SSE).

- **Model Evaluation:** Evaluate the fitted ARIMA model by comparing its forecasts to the actual sales data for a period not used in model estimation. Use metrics like mean absolute error (MAE) and mean squared error (MSE) to assess the model's predictive performance.

Example 2: Forecasting Quarterly GDP Growth

Assume you have quarterly GDP growth data for a country for the past 20 years. You want to use an ARIMA model to forecast GDP growth for the next four quarters.

- **Data Preparation:** Import the GDP growth data into a statistical software package. Check for stationarity and seasonality. If the data is non-stationary, apply differencing. If the data is seasonal, incorporate a seasonal component (s) into the ARIMA model.

- **Model Identification:** Use ACF and PACF plots to identify the appropriate order of the AR, MA, and seasonal (p, q, s) components.

- **Model Estimation:** Estimate the ARIMA model parameters using the identified p, d, q, and s values. Minimize the SSE to find the optimal parameter values.

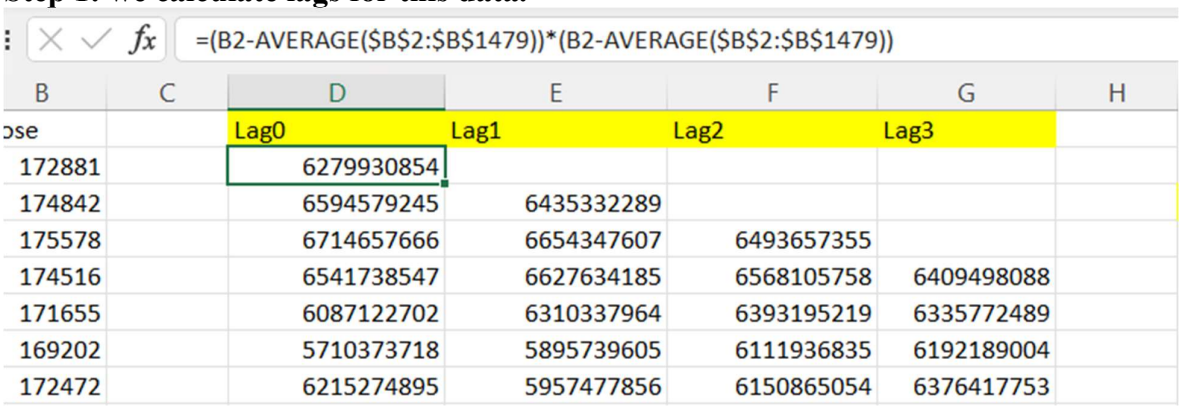
- **Model Evaluation:** Evaluate the fitted ARIMA model by comparing its forecasts to the actual GDP growth data for a period not used in model estimation. Use metrics like MAE and MSE to assess the model's predictive accuracy.

B. TASK 2:

1. ACF:

a) Using Excel:

Step 1: we calculate lags for this data.



B	C	D	E	F	G	H
ɔse		Lag0	Lag1	Lag2	Lag3	
172881		6279930854				
174842		6594579245	6435332289			
175578		6714657666	6654347607	6493657355		
174516		6541738547	6627634185	6568105758	6409498088	
171655		6087122702	6310337964	6393195219	6335772489	
169202		5710373718	5895739605	6111936835	6192189004	
172472		6215274895	5957477856	6150865054	6376417753	

Figure 1: Calculate Lags

Step 2: After these lags, we calculate Denominator.

=DEVSQ($B$2:$B$1479)

	D	E	F	G	H	I
	Lag0	Lag1	Lag2	Lag3		
	6279930854					
	6594579245	6435332289				Denominator
	6714657666	6654347607	6493657355			8.59036E+11
	6541738547	6627634185	6568105758	6409498088		8.59036E+11
	6087122702	6310337964	6393195219	6335772489		8.59036E+11
	5710373718	5895739605	6111936835	6192189004		8.59036E+11
	6215274895	5957477856	6150865054	6376417753		

Figure 2: Calculate the Dnominator

Step 3: Calculate the Numberator.

=SUM(D2:D1479)

	D	E	F	G	H	I	J
	Lag0	Lag1	Lag2	Lag3			
	6279930854						
	6594579245	6435332289				Denominator	Numberator
	6714657666	6654347607	6493657355			8.59036E+11	8.59036E+11
	6541738547	6627634185	6568105758	6409498088		8.59036E+11	8.53384E+11
	6087122702	6310337964	6393195219	6335772489		8.59036E+11	8.47563E+11
	5710373718	5895739605	6111936835	6192189004		8.59036E+11	8.41731E+11
	6215274895	5957477856	6150865054	6376417753			

Figure 3: Calculate the Numberator

Step 4: Calculate ACF by using the value of Numberator divided by the value of Denominator.

=J4/I4

	D	E	F	G	H	I	J	K
	Lag0	Lag1	Lag2	Lag3				
	6279930854							
	6594579245	6435332289				Denominator	Numberator	ACF
	6714657666	6654347607	6493657355			8.59036E+11	8.59036E+11	1
	6541738547	6627634185	6568105758	6409498088		8.59036E+11	8.53384E+11	0.9934202
	6087122702	6310337964	6393195219	6335772489		8.59036E+11	8.47563E+11	0.9866442
	5710373718	5895739605	6111936835	6192189004		8.59036E+11	8.41731E+11	0.9798545
	6215274895	5957477856	6150865054	6376417753				

Figure 4: Calculate ACF

b) Using R Language:

Step 1: Choose file and attach.

```
> df <- read.csv(file = ("C:/Users/User/Downloads/vnmvn.csv"))
> df
         date  close
1     2/1/2018 172881
2     3/1/2018 174842
3     4/1/2018 175578
4     5/1/2018 174516
5     8/1/2018 171655
6     9/1/2018 169202
7    10/1/2018 172472
```

Figure 5: Import file.

Step 2: We perform the Augmented Dickey-Fuller (ADF) test on the 'close' column of the data.

```
> adf.test(df$close)

        Augmented Dickey-Fuller Test

data:  df$close
Dickey-Fuller = -3.0937, Lag order = 11, p-value = 0.1153
alternative hypothesis: stationary
```

Figure 6: Performing the ADF test.

Step 3: We perform the Augmented Dickey-Fuller (ADF) test on the differenced series (adf1). This is a common approach to check for stationarity in time series data.

```
        Augmented Dickey-Fuller Test

data:  adf1
Dickey-Fuller = -12.656, Lag order = 11, p-value = 0.01
alternative hypothesis: stationary
```

Figure 7: Performing the ADF test on differenced series(adf1)

Step 4: We use the acf function in R to plot the autocorrelation function (ACF) for the difference series adf1.

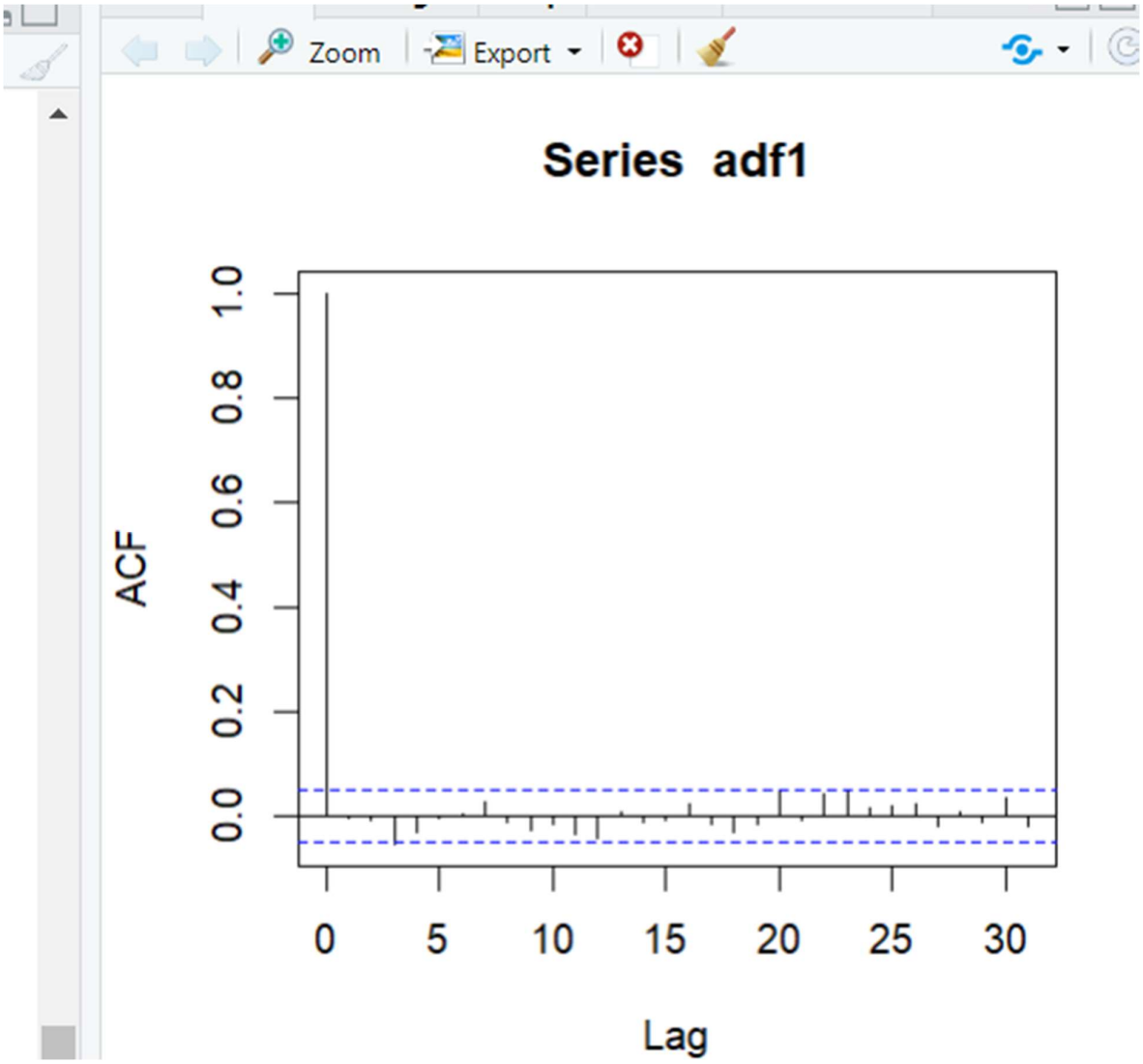


Figure 8: ACF plot

c) Using Python Language:

Step 1: Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import register_matplotlib_converters
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
register_matplotlib_converters()
```

✓ 4.1s

Figure 9: Import libraries.

Step 2: We read file csv.

```
df = pd.read_csv('vnmvn.csv')
df.head()
```

	date	close
0	2/1/2018	172881.0
1	3/1/2018	174842.0
2	4/1/2018	175578.0
3	5/1/2018	174516.0
4	8/1/2018	171655.0

Figure 10: Read the data.

Step 3: We use Matplotlib to create charts and graphs.

```
plt.figure(figsize=(10,4))
plt.plot(df.close)
plt.title('Close datetime', fontsize=20)
plt.ylabel('close', fontsize=16)
for year in range(2018,2023):
    plt.axvline(pd.to_datetime(str(year)+'-01-01'), color='k', linestyle='--', alpha=0.2)
```

✓ 0.1s

Figure 11: Create chart by using Matplotlib.

Here is the chart:

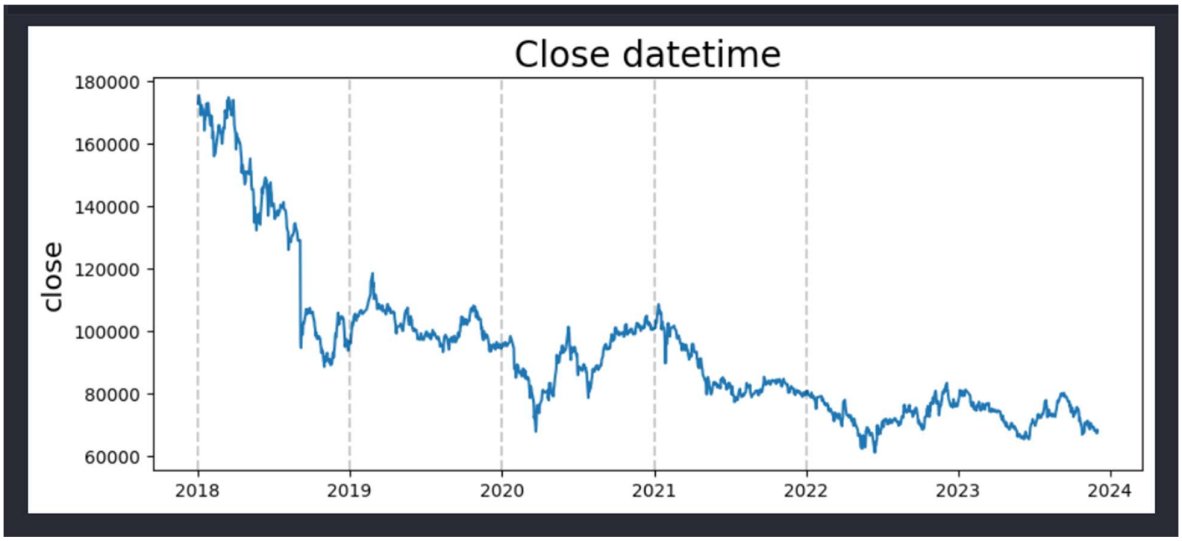


Figure 12: Close datetime chart

Step 4: We create an autocorrelation plot using the plot_acf function from the statsmodels library.

```
acf_plot = plot_acf(df.close, lags=4)
```

✓ 0.1s

Figure 13: create an autocorrelation plot using the plot_acf function.

Here is the chart:

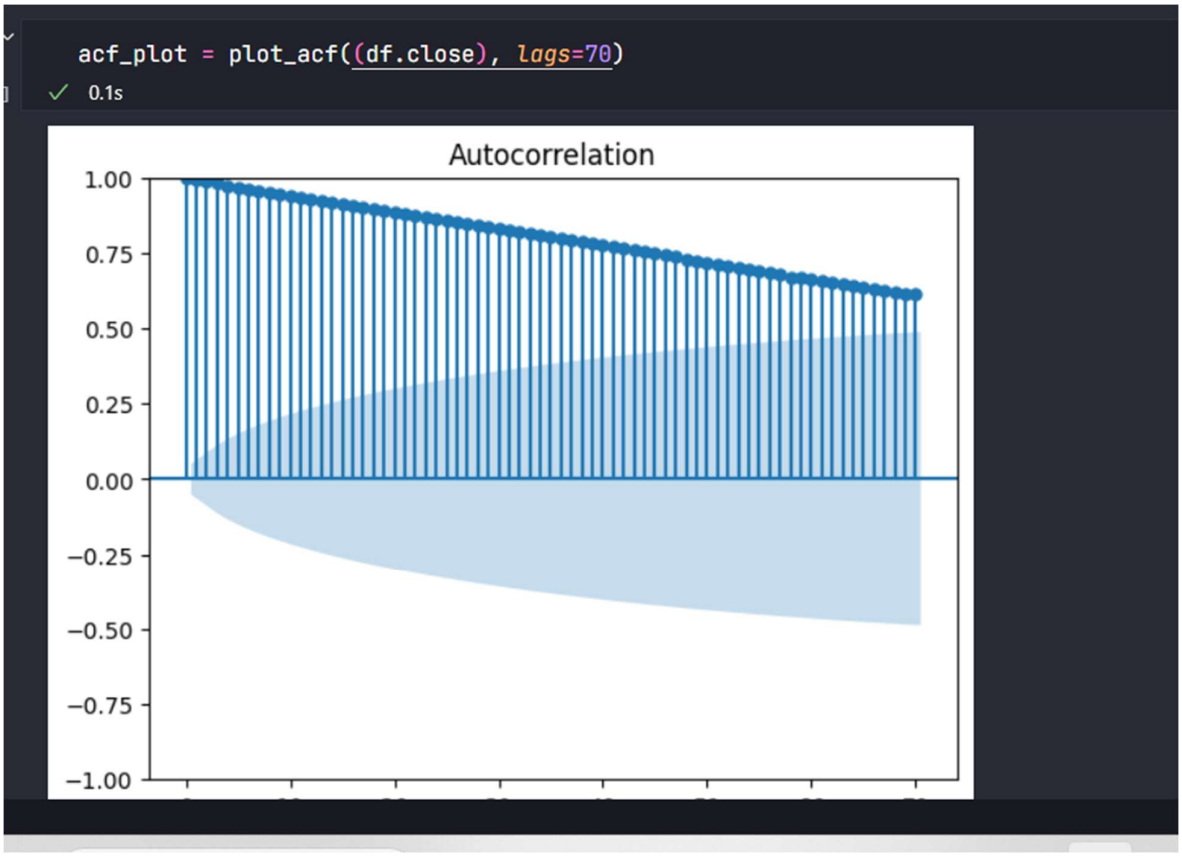


Figure 14: ACF plot

Explain:
In the chart above, the chart gets lower and tends to reach 0. That mean it is a stationary data and we use it to compare the hypothesis with PACF.

2. PACF:

a) Using Excel:

We have rules to calculate the PACF(p):

- The 1st order partial autocorrelation will be defined to equal the 1st order autocorrelation.
- The 2nd order (lag) partial autocorrelation is:

$$\frac{\mathbf{Cov}(Y_t, Y_{t-2} \mid Y_{t-1})}{\sqrt{\boldsymbol{Var}(Y_t \mid Y_{t-1})\boldsymbol{Var}(Y_{t-2} \mid Y_{t-1})}}$$

This is the correlation between values two time periods apart conditional on knowledge of the value in between.

- The 3rd order (lag) partial autocorrelation is:

$$\frac{\mathbf{Cov}(Y_t, Y_{t-3} \mid Y_{t-1}, Y_{t-2})}{\sqrt{\boldsymbol{Var}(Y_t \mid Y_{t-1}, Y_{t-2})\boldsymbol{Var}(Y_{t-3} \mid Y_{t-1}, Y_{t-2})}}$$

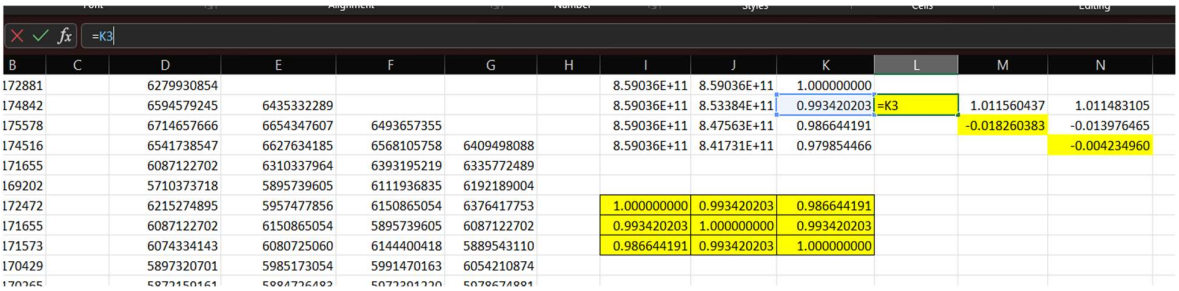


Figure 15 1. Choosing PACF(1) based on ACF

- We have PACF(1) is the same as ACF as we have mentioned above.

Continuous to calculate PACF(2) and PACF(3).

- With PACF(2), we drag from M3:M4 to calculate PACF(p) with the command: =MMULT(MMULT(MINVERSE(MMULT(matrix_range, matrix_range)), matrix_range), range of value)

Which is:

- MINVERSE function calculates the inverse of a square matrix.
- MMULT function calculates the matrix product of two arrays.

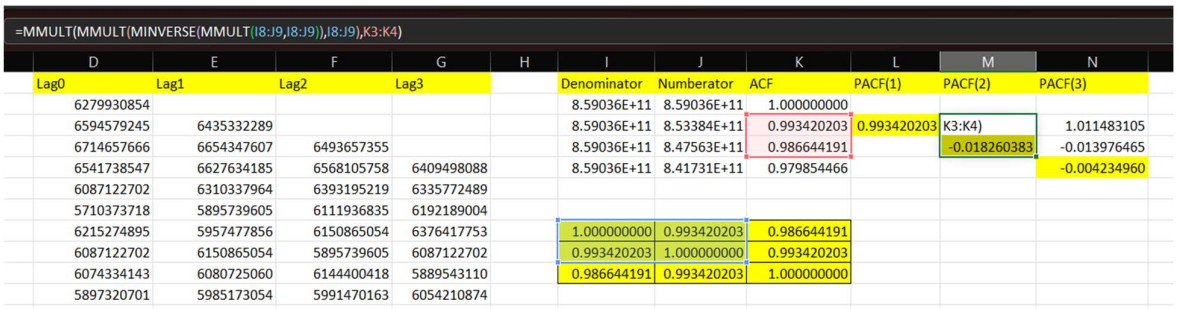


Figure 16 2. Calculating PACF(2) by command

- Do the same with PACF(3) and range is the entire matrix table and the values ACF(1) to ACF(3).

After calculation, we have the following result:

I	J	K	L	M	N
Denominator	Numberator	ACF	PACF(1)	PACF(2)	PACF(3)
8.59036E+11	8.59036E+11	1.000000000			
8.59036E+11	8.53384E+11	0.993420203	0.993420203	1.011560437	1.011483105
8.59036E+11	8.47563E+11	0.986644191		-0.018260383	-0.013976465
8.59036E+11	8.41731E+11	0.979854466			-0.004234960
1.000000000	0.993420203	0.986644191			
0.993420203	1.000000000	0.993420203			
0.986644191	0.993420203	1.000000000			

Figure 173. The result of PACF(p)

=> Based on the result, we choosing PACF(1) because this value is the only value of lags that is greater than significant point (0,05).

⇨ $\boldsymbol{AR(1)} = X_t = \phi X_{t-1} + Z_t$

b) Using R Language:

Step 1: This code prompts the user to select a CSV file using the file.choose() function. The selected file is then read into a data frame named df using the read.csv() function. "df" command prints the contents of the data frame df to the console. This is useful for checking that the file was read correctly and to get a general overview of the data.

```
> df<-read.csv(file.choose())
> df
        date  close  X
1 01/02/2018 172881 NA
2 01/03/2018 174842 NA
3 01/04/2018 175578 NA
4 01/05/2018 174516 NA
5 01/08/2018 171655 NA
6 01/09/2018 169202 NA
```

Figure 184. Reading and printing file

Step 2: This step attaches the data frame df to the global environment. This means that you can now access the variables in the data frame without having to prefix them with df$.
For example, instead of writing df$close, you can simply write close.

```
attach(df)
```

Figure 195. Attach command

Step 3: This code prints the first few rows of the close column of the data frame df. This is useful for getting a quick look at the time series data.

```
head(df$close)
1] 172881 174842 175578 174516 171655 169202
```

Figure 206. Getting a quick look at the data

Step 4:
Method 1: This code calculates the partial autocorrelation function (PACF) of the close column of the data frame df using the acf() function. The type = "partial" option specifies that only the PACF lags up to and including the maximum lag should be calculated. The PACF is then stored in the variable Low_PACF1. The $acf component of Low_PACF1 contains the PACF values.

```
> Low_PACF1 <-acf(df$close,type = "partial")
> Low_PACF1$acf
, , 1

              [,1]
 [1,]  0.9934202029
 [2,] -0.0182603834
 [3,] -0.0042349600
 [4,]  0.0246476122
```

Figure 217. Method 1 of calculating PACF

Method 2: This code calculates the PACF of the close column of the data frame df using the pacf() function. The pacf() function calculates the PACF for all lags. The PACF is then stored in the variable Low_PACF2. The $acf component of Low_PACF2 contains the PACF values.

```
> Low_PACF2 <- pacf(df$close)
> Low_PACF2$acf
, , 1

              [,1]
 [1,]  0.9934202029
 [2,] -0.0182603834
 [3,] -0.0042349600
 [4,]  0.0246476122
```

Figure 228. Method 2 of calculating PACF

In addition, R studio also displays a graph of PACF when we run the pacf function:

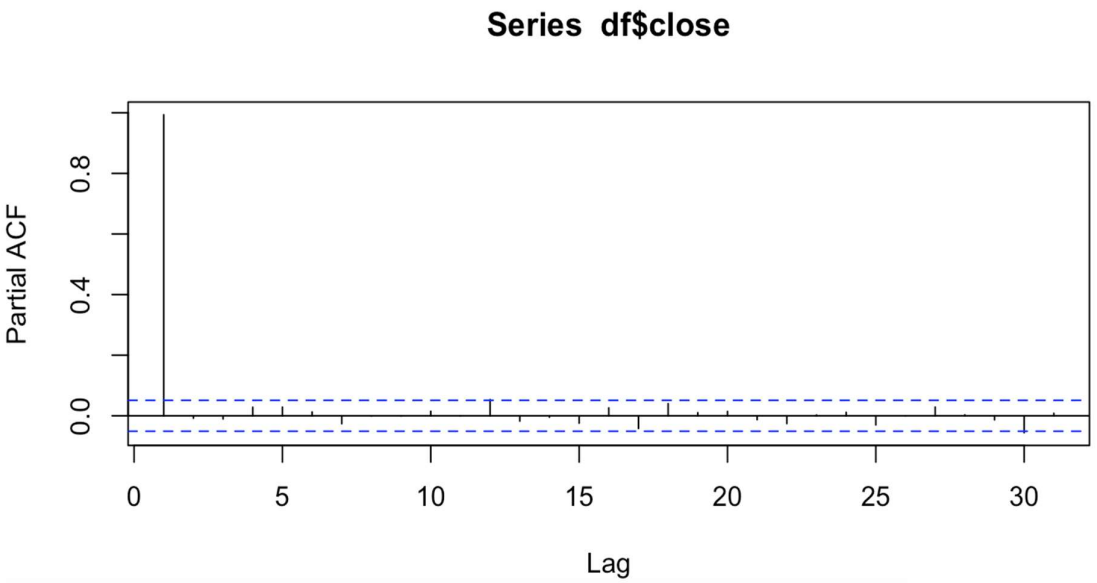


Figure 23. PACF plot in R after entering the command.

Conclusion: After comparing the results, we can see that the calculation results from R's pacf function are completely similar to the results calculated in Excel above. We can see that it only accepts the value PACF(1) and has AR(1).

c) Using Python Language:

Step 1: Imports three libraries that will be used in the analysis.
Step 2: Reads a CSV file named vnmvn.csv into a Pandas DataFrame named df and then prints the DataFrame df to the console. This is useful for checking that the file was read correctly and to get a general overview of the data.

```
In [1]: import pandas as pd
        import statsmodels.graphics.tsaplots as tsaplots
        import statsmodels.tsa.stattools as stattools

In [2]: df=pd.read_csv('vnmvn.csv')

In [3]: df
```

Out[3]:

	date	close	Unnamed: 2
0	01/02/2018	172881.0	NaN
1	01/03/2018	174842.0	NaN
2	01/04/2018	175578.0	NaN
3	01/05/2018	174516.0	NaN
4	01/08/2018	171655.0	NaN
...	...	...	...
1473	27/11/2023	68100.0	NaN
1474	28/11/2023	68500.0	NaN
1475	29/11/2023	68400.0	NaN
1476	30/11/2023	67400.0	NaN
1477	12/01/2023	68400.0	NaN

1478 rows × 3 columns

Figure 249. Import and read the data

Step 3:
Imports the statsmodels.tsa.stattools library again, but this time it is given the alias tsatools. This is done to make the code more readable.

Step 4:
Calculates the partial autocorrelation function (PACF) of the close column of the DataFrame df using the pacf() function from the tsatools library. The nlags parameter specifies the number of lags to calculate the PACF for, and the method parameter specifies the method to use to calculate the PACF. In this case, the ywm method is used, which is a method that is robust to non-stationarity. The PACF values are stored in the variable pacf_values.

Step 5:
Creates a plot of the PACF using the plot_pacf() function from the tsaplots library. The markersize parameter specifies the size of the markers in the plot. The plot is stored in the variable pacf_plot and then displays the plot of the PACF by pacf_plot.show().

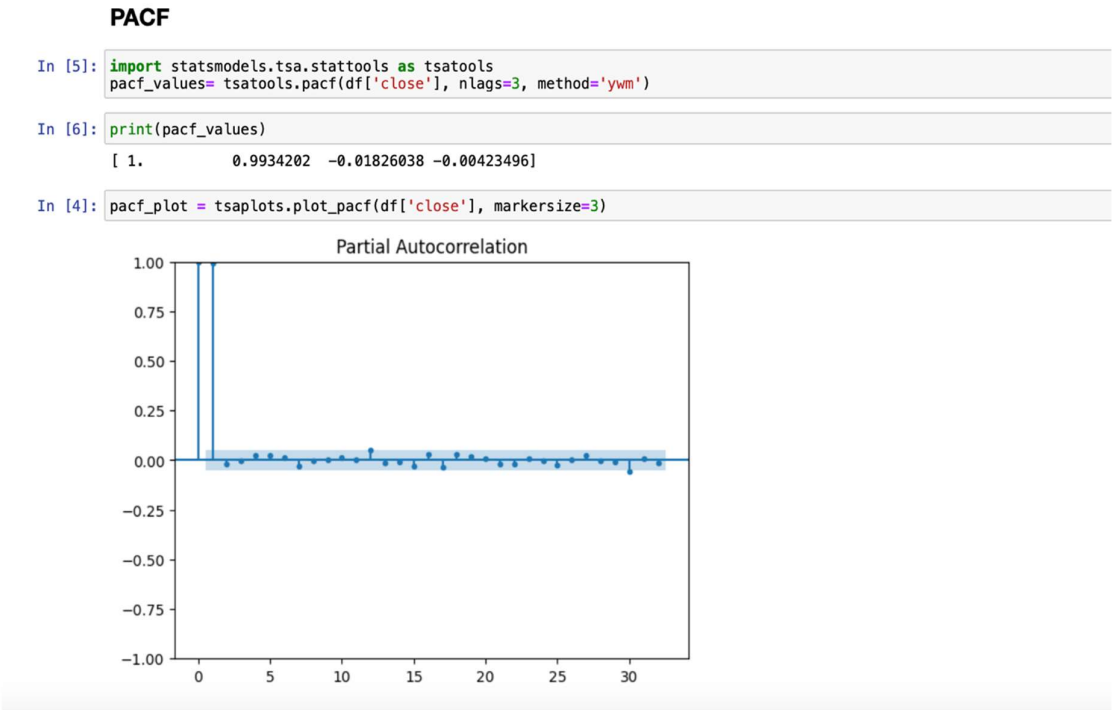


Figure 2510. Perform the process of calculating PACF and drawing a chart

Conclusion: After comparing the results, we can see that the calculation results and graphs from Python's pacf function are completely similar to the results calculated in Excel and R above. We can see that it only accepts the value PACF(1) and has AR(1).

3. ARIMA:

a) Using Excel:

We have ARIMA (3, 0, 0) model. Means that:

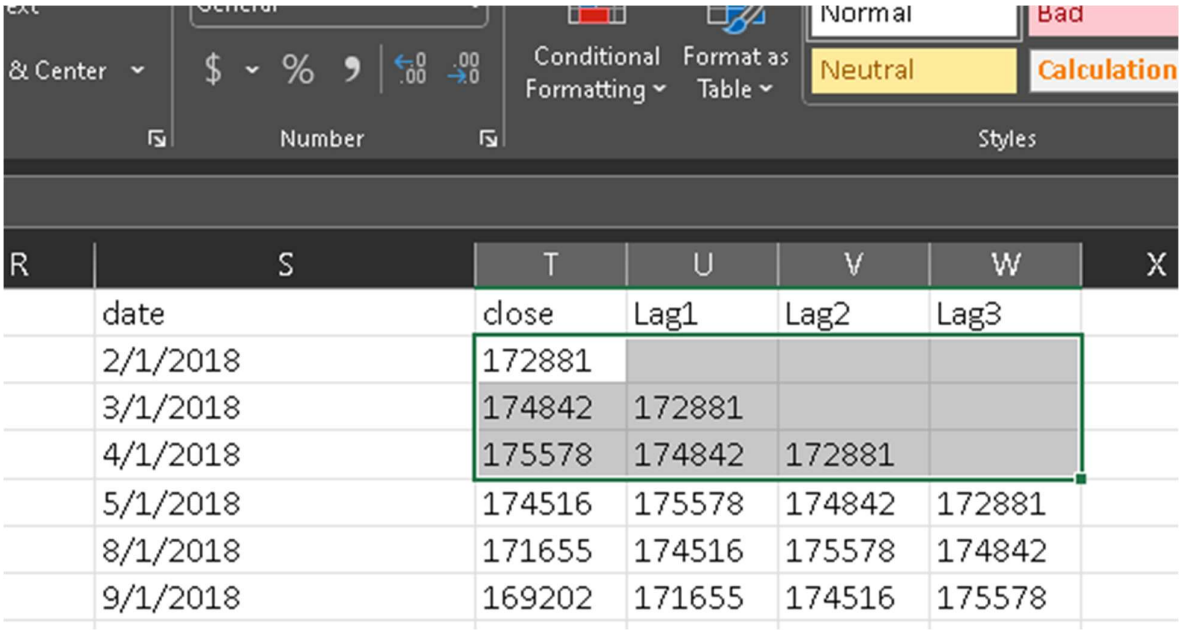
$$Y_t = \mu + \varphi_1(Y_{t-1}) + \varphi_2(Y_{t-2}) + \varphi_3(Y_{t-3})$$

Step 1: Calculate the lag to find the AR(q).

S	T	U	V	W	X
date	close	Lag1	Lag2	Lag3	
2/1/2018	172881				
3/1/2018	174842	172881			
4/1/2018	175578	174842	172881		
5/1/2018	174516	175578	174842	172881	
8/1/2018	171655	174516	175578	174842	
9/1/2018	169202	171655	174516	175578	
10/1/2018	172472	169202	171655	174516	
11/1/2018	171655	172472	169202	171655	

Figure 2611: Calculate Lag 1,2,3

Step 2: Delete all rows which have empty value.



R	S	T	U	V	W	X
	date	close	Lag1	Lag2	Lag3	
	2/1/2018	172881				
	3/1/2018	174842	172881			
	4/1/2018	175578	174842	172881		
	5/1/2018	174516	175578	174842	172881	
	8/1/2018	171655	174516	175578	174842	
	9/1/2018	169202	171655	174516	175578	

Figure 2712: Delete row which has empty value

Step 3: After delete rows which have empty value, we have the final dataset.

S	T	U	V	W
date	close	Lag1	Lag2	Lag3
5/1/2018	174516	175578	174842	172881
8/1/2018	171655	174516	175578	174842
9/1/2018	169202	171655	174516	175578
10/1/2018	172472	169202	171655	174516
11/1/2018	171655	172472	169202	171655
12/1/2018	171573	171655	172472	169202

Figure 2813: Data after delete empty values

Step 4: Choose Data Analysis -> Click Regression -> Click OK.

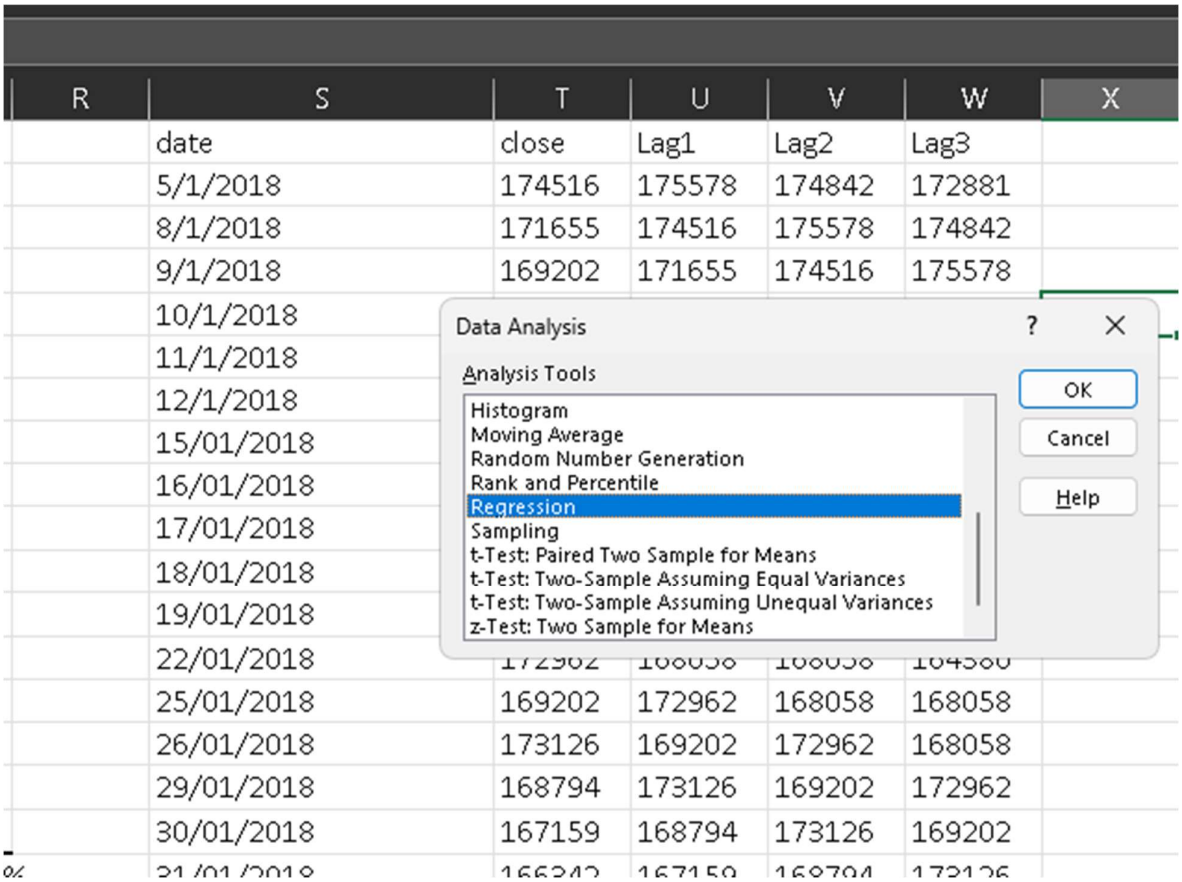


Figure 2914: Dialog of regression

Step 5: Then, we choose 'close' column to be the input Y range.

Lag1, Lag2, Lag3 columns to be the input X range.

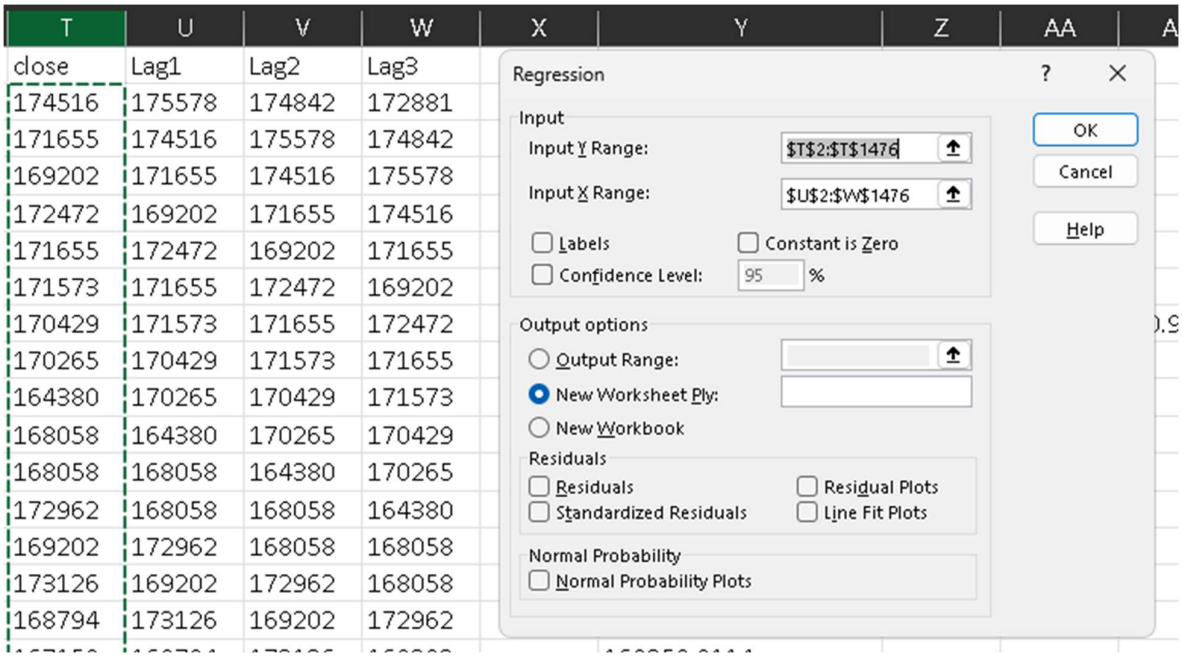


Figure 3015: Enter X & Y values to regression.

Step 6: Click "OK" then we have the coefficients table.

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.99741
R Square	0.99483
Adjusted F	0.99482
Standard E	1718.21
Observati	1475

ANOVA

	df	*SS*	*MS*	*F*	*gnificance F*
Regressior	3	8.4E+11	2.78355E+11	94285.4	0
Residual	1471	4.3E+09	2952259.587		
Total	1474	8.4E+11			

	Coefficients	*tandard Err*	*t Stat*	*P-value*	*Lower 95%*	*Jpper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	507.2	180.43	2.811057742	0.005	153.272	861.128	153.272	861.128
Lag 1	0.98779	0.02607	37.8903196	8E-220	0.93665	1.03893	0.93665	1.03893
Lag 2	-0.0015	0.03665	-0.042150597	0.96638	-0.0734	0.07035	-0.0734	0.07035
Lag3	0.00755	0.02597	0.290641491	0.77137	-0.0434	0.05849	-0.0434	0.05849

Figure 3116: Result of regression

Step 7: After having all necessary values, we start to predict based on this equation:

$$Y_t = 507,2 + 0.98779Y_{t-1} - 0.0015Y_{t-2} + 0.00755Y_{t-3}$$

Figure 3217: Equation of ARIMA

S	T	U	V	W	X	Y
date	close	Lag1	Lag2	Lag3		Predict
5/1/2018	174516	175578	174842	172881		174984.3812
8/1/2018	171655	174516	175578	174842		173949.0497
9/1/2018	169202	171655	174516	175578		171130.1324
10/1/2018	172472	169202	171655	174516		168703.3569
11/1/2018	171655	172472	169202	171655		171915.5091
12/1/2018	171573	171655	172472	169202		171085.0596
15/01/2018	170429	171573	171655	172472		171029.9748
16/01/2018	170265	170429	171573	171655		169893.8977

Figure 3318: Result of prediction

b) Using R Language:

Step: 1: Import the dataset.

```
> df <- read.csv(file.choose(), sep = ",")
> df
         date  close   open   high    low     vol  change
1    2/1/2018 172881 172472 173208 170510 447.31K  0.0139
2    3/1/2018 174842 174516 175742 173371 668.84K  0.0113
3    4/1/2018 175578 174189 175742 174189 845.22K  0.0042
4    5/1/2018 174516 174924 175742 173698 622.63K -0.0060
5    8/1/2018 171655 174107 174107 170429 809.76K -0.0164
```

Figure 3419: Import dataset

Step 2: Set row names using "date":

```
> rownames(df) <-df$date
> df
                date  close   open   high    low     vol  change
2/1/2018    2/1/2018 172881 172472 173208 170510 447.31K  0.0139
3/1/2018    3/1/2018 174842 174516 175742 173371 668.84K  0.0113
4/1/2018    4/1/2018 175578 174189 175742 174189 845.22K  0.0042
5/1/2018    5/1/2018 174516 174924 175742 173698 622.63K -0.0060
```

Figure 3520: Set rows name

Step 3: Check for missing or infinite values:

```
> any(is.na(df$close))
[1] FALSE
>
> any(!is.finite(df$close))
[1] FALSE
>
> which(!is.finite(df$close))
integer(0)
```

Figure 3621: Check the missing or infinite values

Step 4: Replace Commas and convert to numeric:

```
> df$close <- as.numeric(gsub(",", ".", df$close))
```

Figure 3722: Replace commas & convert to numeric

Step 5: Clean datas

```
> df$close[is.na(df$close) | !is.finite(df$close)]
numeric(0)
```

Figure3823: Clean dataset

Step 6: Create time series object and plot:

```
> data <- ts(na.omit(df$close), frequency = 1, start = c(02,01,2018))
>
> plot(data)
```

Figure 3924: Create time series

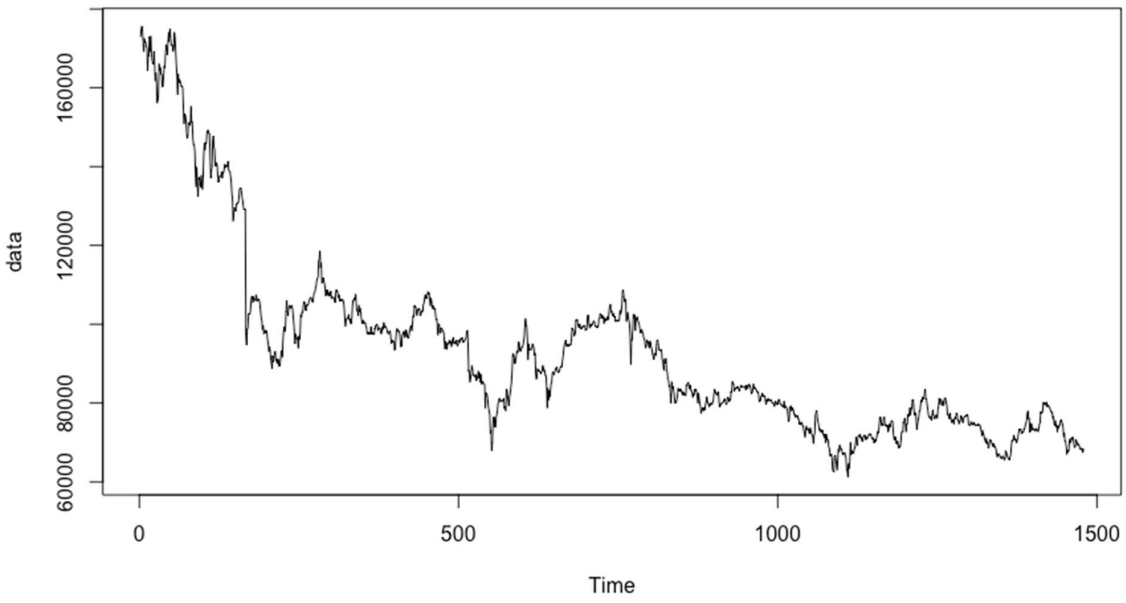


Figure 325: Result of plot data

Step 7: Check for stationarity using ADF Test:

```
> library(tseries)
> adf.test(data)

        Augmented Dickey-Fuller Test

data:  data
Dickey-Fuller = -3.0937, Lag order = 11, p-value = 0.1153
alternative hypothesis: stationary
```

Figure 3426: Check ADF Test

Step 8: Split data into Train, Test, and Validation:

```
> train <- head(df, 1034)
> test <- tail(df, 294)
> val <- tail(df, 194)
```

Figure 3927: Split data into 7:2:1 (Train, Test, Validation)

Step 9: Create time series objects for Train, Test, and Validation:

```
> train_ts <- ts(train$close, frequency = 1, start = c(02, 01, 2018))
> test_ts <- ts(test$close, frequency = 1, start = c(25, 02, 2022))
> val_ts <- ts(val$close, frequency = 1, start = c(28, 04, 2023))
```

Figure 4028: Create time series for Train, Test & Validation

Step 10: Fit ARIMA Model using "auto.arima":

```
> arima_model <- auto.arima(train_ts, trace = TRUE)

 Fitting models using approximations to speed things up...

 ARIMA(2,1,2) with drift         : Inf
 ARIMA(0,1,0) with drift         : 18545.04
 ARIMA(1,1,0) with drift         : 18546.91
 ARIMA(0,1,1) with drift         : 18547.05
 ARIMA(0,1,0)                    : 18545.53
 ARIMA(1,1,1) with drift         : 18548.88

 Now re-fitting the best model(s) without approximations...

 ARIMA(0,1,0) with drift         : 18560.16

 Best model: ARIMA(0,1,0) with drift
```

Figure 4129: Find the fit ARIMA model

Step 11: Fit ARIMA model and forecast for Train, Test, and Validation Sets:

```
> fitARIMAtrain <- arima(train_ts, order=c(0, 1, 0))
> predict <- forecast(fitARIMAtrain, 1034)
> predict
     Point Forecast    Lo 80     Hi 80    Lo 95     Hi 95
1036        74936.8 72466.67  77406.93 71159.07  78714.53
1037        74936.8 71443.51  78430.09 69594.28  80279.32
1038        74936.8 70658.41  79215.19 68393.57  81480.03
1039        74936.8 69996.54  79877.06 67381.33  82492.27
1040        74936.8 69413.43  80460.17 66489.53  83384.07
1041        74936.8 68886.25  80987.35 65683.28  84190.32
1042        74936.8 68401.46  81472.14 64941.85  84931.75
```

Figure 4230: Calculate fit ARIMA & forecast for Train

```
> fitARIMAtest <- arima(test_ts, order=c(0, 1, 0))
> predict2 <- forecast(fitARIMAtest, 295)
> predict2
    Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
320          68400 67089.46 69710.54 66395.70 70404.30
321          68400 66546.62 70253.38 65565.49 71234.51
322          68400 66130.08 70669.92 64928.45 71871.55
```

Figure 4331: Forecast for Test

```
> fitARIMAval <- arima(val_ts, order=c(0, 1, 0))
> predict3 <- forecast(fitARIMAtest, 149)
> predict3
    Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
320          68400 67089.46 69710.54 66395.70 70404.30
321          68400 66546.62 70253.38 65565.49 71234.51
322          68400 66130.08 70669.92 64928.45 71871.55
323          68400 65778.92 71021.08 64391.40 72408.60
324          68400 65469.54 71330.46 63918.25 72881.75
```

Figure 4432: Forecast for Validation

Step 12: Additional Forecasting for Validation Set:

```
> forecast(fitARIMAval, 30)
    Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
225          68400 67236.17 69563.83 66620.08 70179.92
226          68400 66754.10 70045.90 65882.82 70917.18
227          68400 66384.19 70415.81 65317.09 71482.91
228          68400 66072.35 70727.65 64840.16 71959.84
229          68400 65797.61 71002.39 64419.98 72380.02
```

Figure 4533: Result of additional forecasting for Validation set

Step 13: Plot Forecast for Validation Set:

```
> plot(forecast(fitARIMAval, 30))
```

Figure 4634: Plot the result

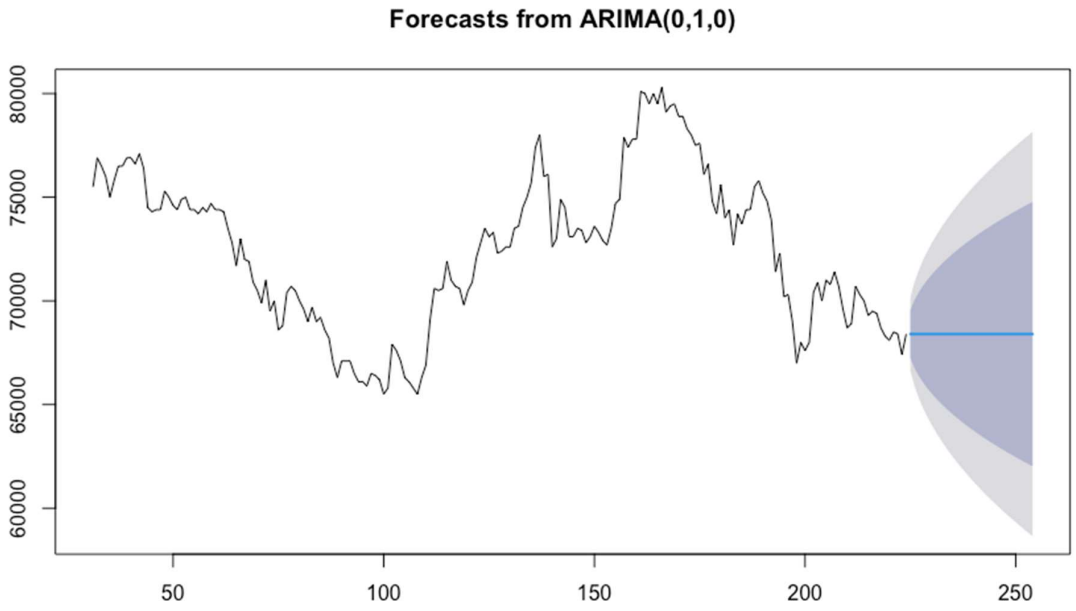


Figure 4735: Image of result forecasting

c) Using Python Language:

Step 1: Import necessary library:

```
[1]: # Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
```

Figure 4836: Import libraries to use

Step 2: We load data from CSV file into Jupyter notebook and split data into 3 parts: train data, test data and validate data with 7:2:1 scale.

```
[2]: # Read dataset file
df = pd.read_csv('vnmvn.csv')
df = df[['close']]
df = df.dropna() # Drop missing values
df = df.reset_index(drop=True) # Reset the index

# Split the data into training, testing, and validation sets by 7:2:1
train_size = int(0.7 * len(df))
test_size = int(0.2 * len(df))
val_size = len(df) - train_size - test_size

train_data = df[:train_size]
test_data = df[train_size:train_size+test_size]
val_data = df[train_size+test_size:]
```

Figure 4937: Import dataset & split dataset

Step 3: Find the best ARI function using “auto_arima” support function in python and then fit the data with the model. The best ARIMA model is ARIMA (0,1,0).

```
[3]: # Training process
x_train = np.array(train_data.index).reshape(-1, 1)
y_train = np.array(train_data['close'])

# Find the best ARIMA model using auto_arima
from pmdarima.arima import auto_arima
model = auto_arima(y_train, trace=True, error_action='ignore', suppress_warnings=True)

# Fit the model
model.fit(y_train)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.69 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=18560.152, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=18561.221, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=18561.227, Time=0.03 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=18560.656, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=18563.234, Time=0.07 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.837 seconds
```

```
[3]: ARIMA
ARIMA(0,1,0)(0,0,0)[0] intercept
```

Figure 5038: Finding the fit ARIMA model

Step 4: Assign testing data and validating data to some variables to calculate.

```
[4]: # make predictions on the testing set
x_test = np.array(test_data.index).reshape(-1, 1)
y_test = np.array(test_data['close'])
y_pred = model.predict(n_periods=len(y_test))

# make predictions on the validate set
x_val= np.array(val_data.index).reshape(-1, 1)
y_val = np.array(val_data['close'])
y_pred_val =  model.predict(n_periods=len(y_val))
```

Figure 5139: Prediction on testing & validation set

Step 5: After finishing testing and validating process, continue to predict the next 30 days.

```
[5]: # 6. Quá trình tạo index predict 30 ngày tiếp theo
last_index =  df.index[-1]
last_data = pd.RangeIndex(start=last_index, stop=last_index+30, step=1)

# Create an array of 30 consecutive integers starting from last_index
x_next_30_days = np.array(range(last_index+1, last_index+31)).reshape(-1, 1)

# Predict the closing prices for the next 30 days
y_next_30_days = model.predict(n_periods=len(x_next_30_days))

# Print the predicted closing prices for the next 30 days
print('Predicted closing prices for the next 30 days:')
print(y_next_30_days)
```

```
Predicted closing prices for the next 30 days:
[74841.98470474 74747.16940949 74652.35411423 74557.53881897
 74462.72352372 74367.90822846 74273.0929332  74178.27763795
 74083.46234269 73988.64704743 73893.83175218 73799.01645692
 73704.20116167 73609.38586641 73514.57057115 73419.7552759
 73324.93998064 73230.12468538 73135.30939013 73040.49409487
 72945.67879961 72850.86350436 72756.0482091  72661.23291384
 72566.41761859 72471.60232333 72376.78702807 72281.97173282
 72187.15643756 72092.3411423 ]
```

Figure 5240: Prediction for next 30 days

Step 6: Calculate RMSE value.

```
[6]: # calculate the RMSE
valid_rmse = np.sqrt(np.mean((y_pred_val - y_val)**2))
test_rmse = np.sqrt(np.mean((y_pred - y_test)**2))
print('Validation RMSE:', valid_rmse)
print('Testing RMSE:', test_rmse)

# calculate the MAPE
valid_mape = np.mean(np.abs((y_val - y_pred_val) / y_val)) * 100
test_mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
print("Validation MAPE:", valid_mape)
print("Testing MAPE:", test_mape)

# calculate the MAE
valid_mae = np.mean(np.abs(y_pred_val - y_val))
test_mae = np.mean(np.abs(y_pred - y_test))
print("Validation MAE:", valid_mae)
print("Testing MAE:", test_mae)
```

```
Validation RMSE: 7605.896707294758
Testing RMSE: 16573.456237169452
Validation MAPE: 9.42634997618378
Testing MAPE: 17.802781729100502
Validation MAE: 6855.259643833731
Testing MAE: 13419.157165405752
```

Figure 5341: Calculate RMSE values.

Step 7: Use plot() function to draw chart.

[16]:
```
# ploting the actual values and predicted values
plt.plot(train_data.index, train_data['close'])
plt.plot(test_data.index, test_data['close'])
plt.plot(val_data.index, val_data['close'])
plt.plot(test_data.index, y_pred)
plt.plot(val_data.index, y_pred_val)
plt.plot(last_data, y_next_30_days)
plt.legend(['Train', 'Test','Validate', 'Predictions_test','Predictions_validate', 'Next 30days'])
plt.show()
```

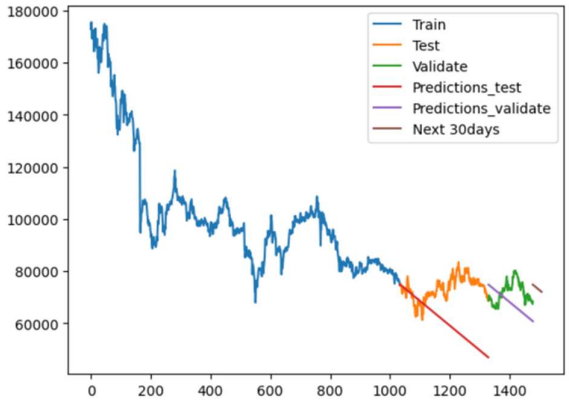


Figure 5442: Plot the result.

REFERENCE

[1] Autocorrelation: What It Is, How It Works, Tests - Tim Smith, March 19, 2023.
[2] Autocorrelation: Meaning, test, characteristics, examples -Wallstreetmojo Team.
[3], [5] Usman Abbas Yakubu, Moch Panji Agung Saputra. (2022). Time Series Model Analysis Using Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) for E-wallet Transactions during a Pandemic. *International Journal of Global Operations Research (Vol. 3, No. 3, pp. 80-85, 2022).* e-ISSN: 2722-1016, p-ISSN: 2723-1739.
[4] Banhi Guha, Gautam Bandyopadhyay. (2016). Gold Price Forecasting Using ARIMA Model (page 2). *Journal of Advanced Management Science*. DOI: 10.12720/joams.4.2.117-121.
[6], [12] Roberto Mota Navarro, Francois Leyvraz, Hernán Larralde. (2023). 3. Methods and metrics. *Dynamical properties of volume at the spread in the Bitcoin/USD market.* Instituto de Ciencias Físicas - Universidad Nacional Autónoma de México, Cuernavaca, 62210, Morelos, México.
[7] Eberly College of Science. (2023). 2.2 Partial Autocorrelation Function (PACF). *Lesson 2: MA Models, Partial Autocorrelation, Notational Conventions*. The Pennsylvania State University. https://online.stat.psu.edu/stat510/lesson/2/2.2
[8], [9] Ali Eshragh, Oliver Di Pietro, Michael A. Saunders. (2021). 3.Toeplitz OLS Problems for Time-series Data. *Toeplitz Least Squares Problems, Fast Algorithms and Big Data (Version 1)* (page 7, 8).
[10] Rob J Hyndman and George Athanasopoulos. (2018). 8.3 Autoregressive models. *Forecasting: Principles and Practice (Second Edition)*. Otexts.
[11] Banhi Guha, Gautam Bandyopadhyay. (2016). Gold Price Forecasting Using ARIMA Model (page 2). *Journal of Advanced Management Science*. DOI: 10.12720/joams.4.2.117-121.
[13] 6.2. ACF AND PACF OF ARMA(P,Q), CHAPTER 6. ARMA MODELS. *TimeSeries*. Queen Mary University of London. https://webspace.maths.qmul.ac.uk/b.bogacka/TimeSeries/TS_Chapter6_2_2.pdf?fbclid=IwAR0_w-7TNRHWSX-fZr3yAlas8YpAH1jS7WsQQmGXkYW1lZOqTz7hDhimCVo
[14] Brownlee, J. (2022, June 21). ARIMA for Time Series Forecasting with Python. Machine Learning Mastery.
[15] "Forecasting ethanol demand in India to meet future blending targets: A comparison of ARIMA and various regression models," Energy Rep., vol. 9, pp. 411–418, Mar. 2023, doi: 10.1016/j.egyr.2022.11.038.
[16] Y. Zhao, "Comparison of Stock Price Prediction in Context of ARIMA and Random Forest Models," *BCP Bus. Manag.*, vol. 38, pp. 1880–1885, Mar. 2023, doi: 10.54691/bcpbm.v38i.3996.

- **Link of documents (Excel, R, Python):** Files of Excel, R and Python
- **Raw dataset:** https://vn.investing.com/equities/vietnam-dairy-products-jsc-historical-data