NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PROJECT REPORT

STATISTICAL ANALYSIS

TOPIC: FORECASTING GOLD PRICES USING TIME SERIES ANALYSIS

Teacher: PSG. Nguyễn Đình Thuân
KS. Nguyễn Minh Nhựt

Class: STAT3013.N11.CTTT

Group: 5

Nguyễn Nhất Thưởng	20522000
Lê Quang Hoà	20521331
Kiều Xuân Diệu Hương	20521381

City. Ho Chi Minh City, January 5, 2022

THANK YOU

I would like to sincerely thank the University of Information Technology National University of Ho Chi Minh City, as well as the teachers and teachers, for creating favorable conditions for us to study and complete the course in the past time. I would also like to thank Mr. Nguyen Dinh Thuan and Mr. Nguyen Minh Nhut who has always been dedicated to teaching and answering questions and helping us in the past time.

Thank the team members for their commitment and effort to complete the assigned tasks. Even though the knowledge is quite heavy, and new, leading to many difficulties in the exchange and discussion process, the members still actively contribute ideas and help each other.

Although our group's knowledge is still limited, the group's project is still not in-depth enough and there are many shortcomings, but this is the final result that the team has achieved. Therefore, we are looking forward to receiving your comments so that we can complete the study in the best way.

Once again I sincerely thank you.

Ho Chi Minh City, January 5, 2022

Group 5 performs

TEACHER'S COMMENTS

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION.

1.1 Reasons for choosing the topic:

2022 is a turbulent year for the world economy. From energy issues to rates are all discussed stories. Inflation in many countries skyrocketed. Another notable issue in 2022 is inflation. As with electric vehicles, this is directly related to energy prices. High energy prices have pushed up inflation rates in many countries. All of them affect the stock market.

International Monetary Fund (IMF) Director General Kristalina Georgieva said on January 1 that the world economy in 2023 will face many difficulties when the main drivers of global growth undergo a recession. weak. The OECD forecasts that the global economic growth rate will decrease from 6.1% in 2021 to 2.2 in 2023, but even so, the world economy will also escape the consecutive quarterly decline. In Vietnam Prime Minister Pham Minh Chinh at the opening session of the 4th session (the XV National Assembly) also commented, "The world situation in 2023 may change rapidly, complicatedly and unpredictably. economy, increasing the potential for production, business and other social activities.

Investment is also an activity that creates jobs for workers, improves people's lives in society, and develops production. In an economic situation facing a crisis, investing becomes even more important. So what to invest and when to invest is the question of many people. In the midst of the economic crisis, gold is considered a safe haven. However, this precious metal is sparkling but extremely elusive. According to the analysis of experts, gold acts as a store of value that protects the value of an investor's assets over time. Gold hoarding contributes to building a hedge against a sharp devaluation of fiat currencies. But to invest in gold, it is necessary to know the information and determine the investment goals.

From the above reasons, group 5 chose the topic "Forecasting gold prices using time series analysis"

CHAPTER 2: NON-LINEAR REGRESSION

2.1 Model definition

Nonlinear time-series analysis is a group of techniques used to gather dynamical data regarding a data set's progression of values [1]. This framework relies critically on the concept of reconstruction of the state space of the system from which the data are sampled, and the information regarding their temporal behaviours is accessible through a single variable’s time series. The aim of this method is to forecast the future direction of the time series, and in some cases, to reconstruct the motion equations. However, in practice, there are several

challenges that limit the effectiveness of this strategy, such as whether the signal completely and precisely captures the dynamics and whether it contains noise. Additionally, the numerical algorithms that we use to instantiate these ideas are not perfect; they involve approximations, scale parameters, and finite-precision arithmetic, among other things. Nevertheless, nonlinear time series analysis has been used for thousands of real data sets from a variety of systems. A simple non-linear regression model is expressed as follows:

$$y = f(x, \beta) + \varepsilon \quad [1]$$

Where:

X is vector of P predictors

β is vector of k parameters

F is the known regression function

ε is the error

2.2 Code execution

Step1: Import data and libraies

```
from google.colab import drive
drive.mount('/content/drive') 123
```

```
Mounted at /content/drive
```

```
[ ] !pip install pystan~=2.14
    !pip install fbprophet
```

Figure 1: Connect to drive and Install Prophet for time series analysis and forecasting.

```
#import libraries
import itertools
from fbprophet import Prophet
import pandas as pd
import numpy as np
```

Figure 2: Import libraries.

```
df=pd.read_csv('/content/drive/MyDrive/Team5-PAPERREPORT/Data/DATA(2002-2022).csv')
df
```

	Date	Prices
0	2002-01-01	4170449.50
1	2002-01-02	4198629.19
2	2002-01-03	4206113.25
3	2002-01-04	4190692.00
4	2002-01-07	4203514.59
...	...	...
5284	2022-04-04	44102530.37
5285	2022-04-05	44470144.87
5286	2022-04-06	44137702.89
5287	2022-04-07	44176594.17
5288	2022-04-08	44384284.57

5289 rows × 2 columns

Figure 3: Get the data from csv file.

Step 2: Data Visualization:

```
df['Prices'].plot(figsize=(12,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff17c0a3c40>
```

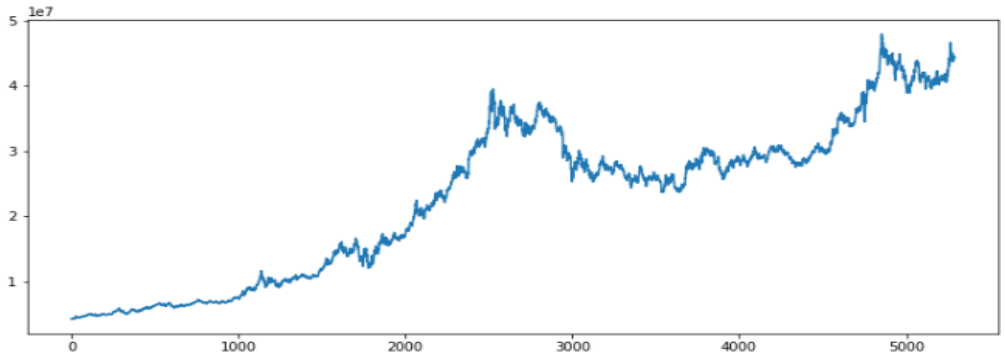


Figure 4: Data plot of actual data.

Step 3: Create model

```
X = np.array(df.index, dtype = float)
y = np.array(df['Prices'], dtype = float)
Z = []
```

```
no_of_predictions = 0
coefs = poly.polyfit(X, y,3)
X_new = np.linspace(X[0], X[-1]+no_of_predictions, num=len(X)+no_of_predictions)
ffit = poly.polyval(X_new, coefs)
pred = poly.polyval(Z, coefs)
predictions = pd.DataFrame(Z,pred)
```

Figure 5: Using X to create index for calculating and Y is the price.

Step 4: Fitting the model

```
print (predictions)
plt.plot( y, label="Original data")
plt.plot(ffit, label = "Fitted data")
plt.legend(loc='upper left')
plt.show()
```

```
Empty DataFrame
Columns: []
Index: []
```

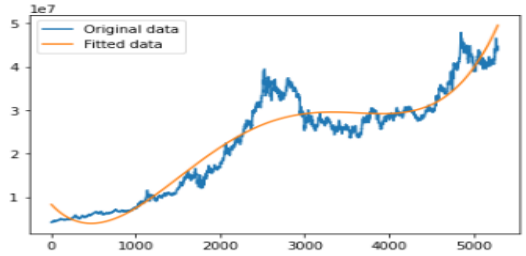


Figure 6: Fitting the model.

Step 5: Model Evaluation

The RMSE and MAPE is calculate by using actual data and predicted data from 2002-2022.

```
[ ] from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error
    mape = mean_absolute_percentage_error(y,ffit)
    mse = mean_squared_error(y,ffit)
    rmse = np.sqrt(mse)
    r = r2_score(y,ffit)
    print(f"R^2: {r:.2f}")
    print(f"MAPE: {mape * 100:.2f}%")
    print(f"RMSE: {rmse:.2f}")
```

```
R^2: 0.85
MAPE: 21.78%
RMSE: 4545507.81
```

Figure 7: Model Evaluation.

CHAPTER 3: LINEAR REGRESSION

3.1 Model definition

Linear regression is a statistical method for determining the value of a dependent variable from an independent variable. Linear regression is also a measure of the connection between two variables [2]. A dependent variable is predicted using this modeling approach based on one or more independent factors. Of all statistical methods, linear regression analysis is the one that is most frequently utilized.

3.2 Significance of Linear Regression:

a. Descriptive - It aids in evaluating the degree of correlation between the outcome (the dependent variable) and the predictor factors.

b. Adjustment - It adjusts the influence of variables or confounders.

c. Predictors - It aids in the estimation of significant risk variables that have an impact on the dependent variable.

d. Extent of prediction: - It aids in determining how much a change in the independent variable of one "unit" might impact the dependent variable.

e. Prediction - It assists in quantifying new cases.

Following the formation of the predictive network pattern, the topic will turn to linear regression, which is likewise a relatively straightforward technique for examining the relationship between a predictor variable and a response variable

$$y = \beta_0 + \beta_1 x + e$$

To create multiple linear regression equations, the variables were extracted from the dataset using correlation, and the parameters were taken from the training data [2]. The number r, also known as the linear correlation coefficient, quantifies the direction and intensity of a relationship between two variables.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

3.3 Code execution

Step 1: Import library

```
from google.colab import drive
drive.mount('/content/drive') 123
```

```
Mounted at /content/drive
```

```
[ ] !pip install pystan~=2.14
    !pip install fbprophet
```

Figure 8: Connect to drive and Install Prophet for time series analysis and forecasting.

```
import numpy.polynomial.polynomial as poly
from datetime import datetime
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LinearRegression
```

Figure 9: Import libraries.

```
df=pd.read_csv('/content/drive/MyDrive/Team5-PAPERREPORT/Data/DATA(2002-2022).csv',parse_dates = ['Date'],index_col = ['Date'])
df
```

Date	Prices
2002-01-01	4170449.50
2002-01-02	4198629.19
2002-01-03	4206113.25
2002-01-04	4190692.00
2002-01-07	4203514.59
...	...
2022-04-04	44102530.37
2022-04-05	44470144.87
2022-04-06	44137702.89
2022-04-07	44176594.17
2022-04-08	44384284.57

5289 rows × 1 columns

Figure 10: Get the data from csv file.

Step 2: Data Transformation

The Date's data type now is not date time so we have to transform the data type, We change the Date column in to datetime type. And set the column Datenew into index.

```
df['Date'] = pd.to_datetime(df['Date'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5289 entries, 0 to 5288
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5289 non-null   datetime64[ns]
 1   Prices  5289 non-null   float64
 2   Time    5289 non-null   int32
dtypes: datetime64[ns](1), float64(1), int32(1)
memory usage: 103.4 KB
```

```
df['Time'] = np.arange(len(df.index))
df
```

Date	Prices	Time
2002-01-01	4170449.50	0
2002-01-02	4198629.19	1
2002-01-03	4206113.25	2
2002-01-04	4190692.00	3
2002-01-07	4203514.59	4
...	...	...
2022-04-04	44102530.37	5284
2022-04-05	44470144.87	5285
2022-04-06	44137702.89	5286
2022-04-07	44176594.17	5287
2022-04-08	44384284.57	5288

5289 rows × 2 columns

Figure 11: Data transformation

Step 3: Create and fitting model

```
numberpredictions=0
date_predict = pd.date_range(start="2022-4-9",end="2022-5-9")
date_predict
```

```
DatetimeIndex(['2022-04-09', '2022-04-10', '2022-04-11', '2022-04-12',
               '2022-04-13', '2022-04-14', '2022-04-15', '2022-04-16',
               '2022-04-17', '2022-04-18', '2022-04-19', '2022-04-20',
               '2022-04-21', '2022-04-22', '2022-04-23', '2022-04-24',
               '2022-04-25', '2022-04-26', '2022-04-27', '2022-04-28',
               '2022-04-29', '2022-04-30', '2022-05-01', '2022-05-02',
               '2022-05-03', '2022-05-04', '2022-05-05', '2022-05-06',
               '2022-05-07', '2022-05-08', '2022-05-09'],
              dtype='datetime64[ns]', freq='D')
```

Figure 12: Create predicting dates.

```
coefs = poly.polyfit(range, value,1)
range_new = np.linspace(range[0], range[-1]+ numberpredictions, num=len(range)+numberpredictions)
range_new1 = np.linspace(range[0], range[-1]+ numberpredictions, num=len(range))
fit = poly.polyval(range_new, coefs)
pred = poly.polyval(predict_range, coefs)
prediction = pd.DataFrame(date_predict ,pred)
print (prediction)
```

	0
4.167637e+07	2022-04-09
4.168353e+07	2022-04-10
4.169069e+07	2022-04-11
4.169785e+07	2022-04-12
4.170502e+07	2022-04-13
4.171218e+07	2022-04-14
4.171934e+07	2022-04-15
4.172650e+07	2022-04-16
4.173366e+07	2022-04-17
4.174082e+07	2022-04-18
4.174798e+07	2022-04-19
4.175514e+07	2022-04-20
4.176230e+07	2022-04-21
4.176946e+07	2022-04-22
4.177662e+07	2022-04-23
4.178378e+07	2022-04-24
4.179094e+07	2022-04-25
4.179810e+07	2022-04-26
4.180527e+07	2022-04-27
4.181243e+07	2022-04-28
4.181959e+07	2022-04-29
4.182675e+07	2022-04-30
4.183391e+07	2022-05-01
4.184107e+07	2022-05-02
4.184823e+07	2022-05-03
4.185539e+07	2022-05-04
4.186255e+07	2022-05-05
4.186971e+07	2022-05-06
4.187687e+07	2022-05-07
4.188403e+07	2022-05-08
4.189119e+07	2022-05-09

Figure 13: 6 Predicting dates.

Step 4: Data visualiation

```
plt.plot(df.index,value, label="Original data")
plt.plot(df.index, fit, label = "Fitted data")
plt.plot(date_predict, pred, 'green', label="Prediction data")
plt.legend(loc='upper left')
plt.show()
```

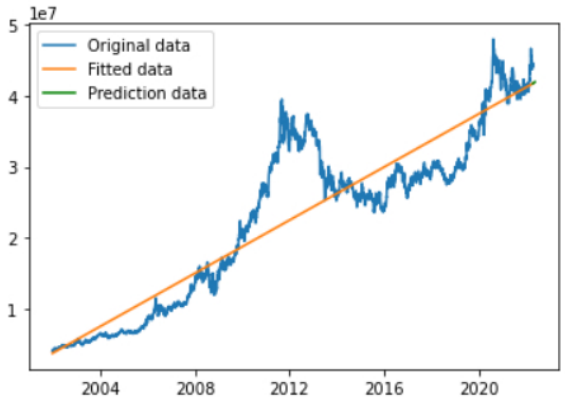


Figure 14: Predicted and actual data.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error
mape = mean_absolute_percentage_error(df['Prices'],fit)
mse = mean_squared_error(df['Prices'],fit)
rmse = np.sqrt(mse)

print(f"MAPE: {mape * 100:.2f}%")
print(f"RMSE: {rmse:.2f}")
```

```
MAPE: 17.65%
RMSE: 4777274.13
```

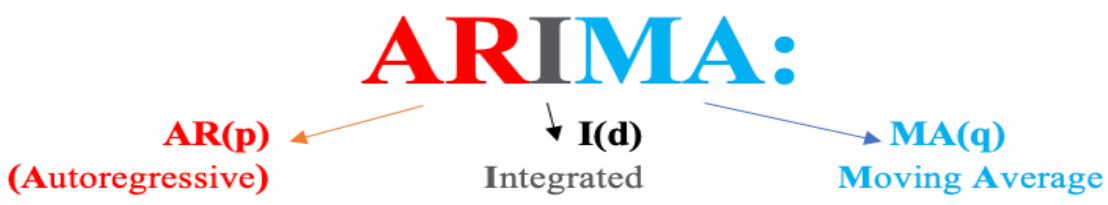
Figure 15: Model evaluate.

CHAPTER 4: ARIMA

4.1 Model definition:

ARIMA stands for **A**utoregressive **I**ntegrated **M**oving **A**verage. ARIMA has time-series data train and made future predictions. As it incorporates error terms and observations of lag components, ARIMA may capture complicated interactions. Therefore, ARIMA model uses linear regression to create predictions while attempting to explain data by using time series data on its previous values.

4.2 How ARIMA model work:



Auto Regressive (AR) model is is a stationary process which Y_t depends only on its own lags [3].

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \ldots + \beta_p Y_{t-p} + \epsilon_t$$

Moving Average (MA) model is a stationary process which is depends on the lagged forecast errors [3].

$$Y_t = \alpha + \epsilon_t + \theta_1 \epsilon_{t-1} + \emptyset_2 \epsilon_{t-2} + \cdots + \emptyset_q \epsilon_{t-q}$$

Intergrated (I): The difference (d) is used when the original series becomes stationary. And we will use the difference to make a non-stationary to be a stationary.

$$y'_t = y_t - y_{t-1} = (1 - B) y_t$$

ARIMA with order p, d, q can be estimated by the following formula [3]:

$$Y_T = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \ldots + \beta_p Y_{t-p} \epsilon_t + \emptyset_1 \epsilon_{t-1} + \emptyset_2 \epsilon_{t-2} + \cdots + \emptyset_q \epsilon_{t-q}$$

4.3 Advantage and disadvatage of ARIMA:

Advantage:

- The forecast can be generalized using only the prior data from a time series.
- Meets expectations for short-term forecasts.
- Non-stationary time series are modeled.

Disadvantage:

- Determining the model's (p, d, q) order can involve a lot of subjectivity.
- Quite expensive.
- Longer-term forecasts perform worse.

4.4 Code execution

Step1:Import library

- First of all, we will import drive which contains our dataset. And after that the command drive.mount will follow to the link (‘/content/drive’) with the aim to approach to our dataset.

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

Figure 16: Import drive from colab.

```
import pandas as pd #read data file csv
import numpy as np #call the numpy library and related functions and data types.
import itertools #Functions creating iterators for efficient looping
import matplotlib as plt
import matplotlib.pyplot as plt #draw chart
import statsmodels.api as sm #Cross-sectional models and methods
#Index analysis
from statsmodels.tsa.stattools import adfuller #library for estimate the ADF test
from statsmodels.tsa.arima_model import ARIMA #library for estimate the ARIMA function
from sklearn.metrics import mean_absolute_error #measure mean absolute error
from sklearn.metrics import mean_squared_error #measure MSE and RMSE = sqrt(MSE)
from sklearn.metrics import mean_absolute_percentage_error #measure the mean percent absolute error
```

Figure 17: Import Library.

Step 2: Read Data

We use the library pandas (pd) to read the file csv "/content/drive/MyDrive/Team5-PAPERREPORT/Data/DATA(2002-2022).csv", and then we use the command index_col = "Date" to convert index column to Date column.

```
#Proceed to read csv data using pandas library with the command pd.read_csv
df=pd.read_csv('/content/drive/MyDrive/Team5-PAPERREPORT/Data/DATA(2002-2022).csv',parse_dates = ['Date'],index_col = ['Date'])
df
```

Figure 18: Read Data.

After that we use df command to display the dataset from file csv.

	Prices
Date	
2002-01-01	4170449.50
2002-01-02	4198629.19
2002-01-03	4206113.25
2002-01-04	4190692.00
2002-01-07	4203514.59
...	...
2022-04-04	44102530.37
2022-04-05	44470144.87
2022-04-06	44137702.89
2022-04-07	44176594.17
2022-04-08	44384284.57

5289 rows × 1 columns

Figure 19: The result after use df command.

To drawing the plot of ['Prices'] we use "df.plot()" so that we can see the change of the prices of golden from 2002 to 2022

```
df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3844ae2460>
```

Figure 20: The plot of Prices from 2002 to 2022.

In addition, to see the component of model or what are used to created our model we use the command "sm.tsa.seasonal_decompose(df,model = 'additive')"

We have two type of decompose:

The additive model is the terms can also refer to a particular model for time series data, where the model can be decomposed into four different components, related in an additive sense.

$$Y[t] = T[t] + S[t] + e[t]$$

The multiplicative model is when the components are related instead by multiplication, the model is a multiplicative model:

$$Y[t] = T[t] * S[t] * e[t]$$

Therefore, our dataset is additive so that our model have components include trend, seasonal, residual.

```
decomposition = sm.tsa.seasonal_decompose(df,model = 'additive')
fig = decomposition.plot()
fig.set_size_inches(10, 5)
```

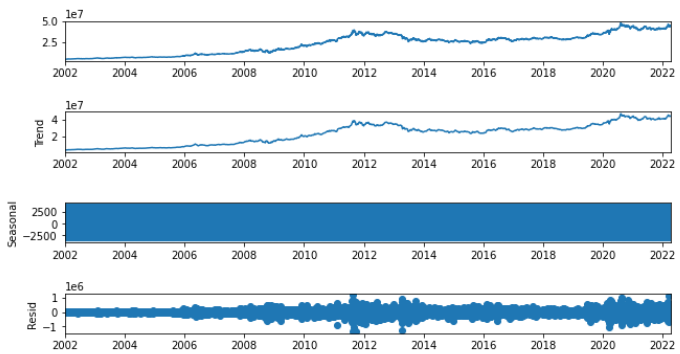


Figure 21: Component of model.

Step 3: Check stationary:

ADF TEST:

We will use ADF test is used to determine the presence of unit root in the series, and hence helps in understand if the series is stationary or not. The null and alternate hypothesis of this test are:

$$H_0: unit\ root, non-stationary$$
$$H_1: non-unit\ root, stationary$$

If the null hypothesis in failed to be rejected, this test may provide evidence that the series is non-stationary.

```
[75] def adf_test(timeseries):
        print("Results of Dickey-Fuller Test:")
        dftest = adfuller(timeseries, autolag="AIC")
        dfoutput = pd.Series(
            dftest[0:4],
            index=[
                "Test Statistic",
                "p-value",
                "#Lags Used",
                "Number of Observations Used",
            ],
        )
        for key, value in dftest[4].items():
            dfoutput["Critical Value (%s)" % key] = value
        print(dfoutput)

adf_test(df)

Results of Dickey-Fuller Test:
Test Statistic                  -0.370760
p-value                          0.914868
#Lags Used                      24.000000
Number of Observations Used    5264.000000
Critical Value (1%)              -3.431593
Critical Value (5%)              -2.862089
Critical Value (10%)             -2.567062
dtype: float64
```

Figure 22: The result of ADF test.

p-value is obtained is greater than significance level of 0.05 and the ADF statistic is higher than any of the critical values.

Therefore, we have to reject the null hypothesis. Our result is that the time series is non-stationary.

KPSS TEST:

KPSS is another test for checking the stationarity of a time series. The null and alternate hypothesis for the KPSS test are opposite that of the ADF test.

Null Hypothesis: The process is trend stationary.

Alternate Hypothesis: The series has a unit root (series is not stationary). A function is created to carry out the KPSS test on a time series

```
from statsmodels.tsa.stattools import kpss

def kpss_test(timeseries):
    print("Results of KPSS Test:")
    kpsstest = kpss(timeseries, regression="c")
    kpss_output = pd.Series(
        kpsstest[0:3], index=["Test Statistic", "p-value", "Lags Used"]
    )
    for key, value in kpsstest[3].items():
        kpss_output["Critical Value (%s)" % key] = value
    print(kpss_output)
```

```
kpss_test(df)
```

```
Results of KPSS Test:
Test Statistic           13.236847
p-value                   0.010000
Lags Used                33.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
```

Figure 23: The result of KPSS Test.

The p-value is obtained is smaller than significance level of 0.05

Following to the result of ADF test and KPSS test we can see that our model is non-stationary.

To change to an non-stationary to stationary model we can use difference

Detrending by Differencing

It is one of the simplest methods for detrending a time series. A new series is constructed where the value at the current time step is calculated as the difference between the original observation and the observation at the previous time step.

```
df['Prices_diff'] = df['Prices'] - df['Prices'].shift(12)
df[['Prices','Prices_diff']].head(13)
```

Date	Prices	Prices_diff
2002-01-01	4170449.50	NaN
2002-01-02	4198629.19	NaN
2002-01-03	4206113.25	NaN
2002-01-04	4190692.00	NaN
2002-01-07	4203514.59	NaN
2002-01-08	4204907.59	NaN
2002-01-09	4244045.59	NaN
2002-01-10	4330859.09	NaN
2002-01-11	4314110.63	NaN
2002-01-14	4324455.49	NaN
2002-01-15	4291987.66	NaN
2002-01-16	4298596.19	NaN
2002-01-17	4305210.00	134760.5

After that we will check the value of Prices after using difference

```
X=df['Prices_diff'].dropna()
adf_test(X)
```

```
Results of Dickey-Fuller Test:
Test Statistic                -1.323232e+01
p-value                        9.529576e-25
#Lags Used                     3.300000e+01
Number of Observations Used    5.243000e+03
Critical Value (1%)           -3.431598e+00
Critical Value (5%)           -2.862091e+00
Critical Value (10%)          -2.567064e+00
```

Figure 24: The result of ADF test in diff(1).

According to ADF test, the p-value < 0,05(1) then the null hypothesis can be rejected. Hence the model is stationary

```
kpss_test(X)
```

```
Results of KPSS Test:
Test Statistic            0.097289
p-value                   0.100000
Lags Used                33.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
```

Figure 25: The result of KPSS test in diff(1).

According to ADF test, the p-value >0.05(2) then the null hypothesis can be rejected. Hence the model is stationary

From (1) and (2) we can come to the following conclusion: Our model can be stationary when d=1.

DETERMINE "q" AND "p":

The common way to determine q and p is using ACF plot and PACF plot. However,we find it difficult because we don't have enough knowledge to make sure that we can find "q" and "p" by ACF plot and PACF plot.

Another way for us is using auto_arima. The auto.arima() function is useful, but anything automated can be a little dangerous, and it is worth understanding something of the behaviour of the models even when you rely on an automatic procedure to choose the model for you. Therefore, we have another way to estimate the "q" and "p" also "d" without using auto_arima, we will use the loop of the range of (q,d,p). That may be an auto_arima but using in manual way.

Step 4 : Split the train and test include :90-10%, 80-20%,70-30%

Step 5: Using the loop for choosing the best model for the train and test.

First of all, to choose the best model we have to import itertools and give y range of p d and q. P and q can have any value from 0 to 8 and d have value from 0 to 2.

Secondly we take all combination p d q by running our itertools.product and pdq have 128 combination

```
p = range(0,8)
q = range(0,8)
d = range(0,2)
pdq_combination = list(itertools.product(p,d,q))
len(pdq_combination)
```

```
128
```

Figure 26: The range of q,d,p.

Writing the for loop where (p,d,q) running though all possible combination (p,d,q) and it will pass (p,d,q) in the order of ARIMA.

Different model will be create and save. This model will be used to predict. we will calculate the RMSE and it will be save in error. Pdq will be save in order.

```
rmse=[]
order1=[]
```

```
for pdq in pdq_combination:
  try:
    model = ARIMA(train,order=pdq).fit()
    pred=model.predict(start=len(train),end=(len(df)-1))
    error = np.sqrt(mean_squared_error(test,pred))
    order1.append(pdq)
    rmse.append(error)
  except:
    continue
```

Figure 27: The loop for (p,d,q)

Finally we export a csv file with 2 columns: pdq and RMSE

```
results = pd.DataFrame(index = order1,data=rmse,columns = ['RMSE'])
```

```
results.to_csv('ARima_result.csv')
```

Figure 28; Create a file csv for the result of the loop

The file csv will be save in the folder on colab.

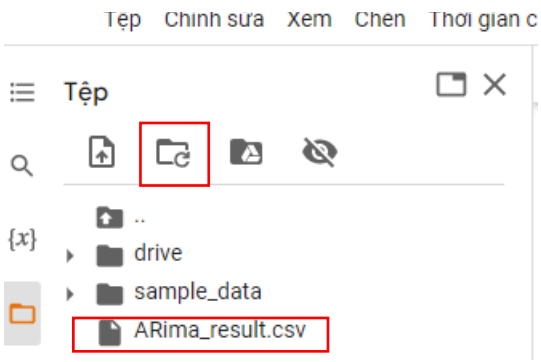


Figure 29: File csv after running the loop.

Step 6: Fit the model following to the train and test

```
# Setting up ARIMA model
model = ARIMA(train,order=(6,0,2)).fit()
```

Figure 30: Set ARIMA model.

```
print(model.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                 Prices   No. Observations:                 4760
Model:                     ARMA(6, 2)   Log Likelihood              -65849.718
Method:                       css-mle   S.D. of innovations         246109.654
Date:                Wed, 04 Jan 2023   AIC                         131719.436
Time:                        16:39:18   BIC                         131784.116
Sample:                    01-01-2002   HQIC                        131742.161
                         - 03-30-2020
==============================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         2.061e+07        nan        nan        nan         nan         nan
ar.L1.Prices    -0.8234        nan        nan        nan         nan         nan
ar.L2.Prices     0.9396   2.38e-05   3.94e+04      0.000       0.940       0.940
ar.L3.Prices     0.8837   3.07e-05   2.88e+04      0.000       0.884       0.884
ar.L4.Prices    -0.0183      0.000    -60.784      0.000      -0.019      -0.018
ar.L5.Prices     0.0112        nan        nan        nan         nan         nan
ar.L6.Prices     0.0071        nan        nan        nan         nan         nan
ma.L1.Prices     1.8326      0.007    278.826      0.000       1.820       1.846
ma.L2.Prices     0.8957      0.007    136.771      0.000       0.883       0.909
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0000           -0.0000j            1.0000           -0.0000
AR.2           -1.0099           -0.2782j            1.0475           -0.4572
AR.3           -1.0099           +0.2782j            1.0475            0.4572
AR.4            2.4155           -4.2433j            4.8826           -0.1676
AR.5            2.4155           +4.2433j            4.8826            0.1676
AR.6           -5.3978           -0.0000j            5.3978           -0.5000
MA.1           -1.0230           -0.2643j            1.0566           -0.4598
MA.2           -1.0230           +0.2643j            1.0566            0.4598
-----------------------------------------------------------------------------
```

Figure 31: ARIMA model result.

The above table is Summary's result from the Arima model [4]

- Coef : coefficient is estimated by the corresponding model with variables on the left

- Std : Standard deviation of the correcsponding Coef. From the estimated coefficient and standard deviation we can calculate the confidence interval. The lower and upper bound is [0.025 , 0.095]
- Z: critical value from the normal distribution. The value of z is calculated by:

$$z= (\beta- \mu)/(\mu\_\alpha/2\hat{}\sigma)$$

Where:

- B is estimated value
- μ is mean value
- μ_αis the 97.5% percentile of the normal distribution.

P > |z| is the probability of the value P(|X| > 0 | X ~ N(0,σ2). This is P-value of H0 hypothesis. If P-value < 0.05 mean that the estimated coefficient > 0 and it has statistical significance.

The index on the right is:

- No. Observation: number of observations
- Log Likelihood: Value of logarit fuction
- AIC: Akaike Information Criteria Index
- BIC: Bayesian Information Criteria index. This index also has the function of measuring the error of the model like AIC but according to the statistical inference

Step 7: Model evaluation:

```
mape = mean_absolute_percentage_error(test, pred)
mse = mean_squared_error(test, pred)
rmse = np.sqrt(mse)

print(f"MAPE: {mape * 100:.2f}%")
print(f"RMSE: {rmse:.2f}")
```

```
MAPE: 8.73%
RMSE: 4149475.21
```

Figure 32: The best result for the best model

Step 8 Prediction for train and test:

We use the model which was fitted in step 6 to predict the test set with named "pred". The start is the last day of train set til the end of the dataset.

```
pred = model.predict(start=len(train),end=(len(df)-1))
```

Figure 33: The pred for train and test.

After that we will draw the plot of the train, test and pred to compare the reality (test set) with the prediction we was predicted (pred)

```
plt.plot(train.index, train['Prices'],label = 'train')
plt.plot(test.index, test['Prices'], color='r',label = 'test')
pred.plot(label = 'predict',figsize = (20,12))
plt.legend()
```

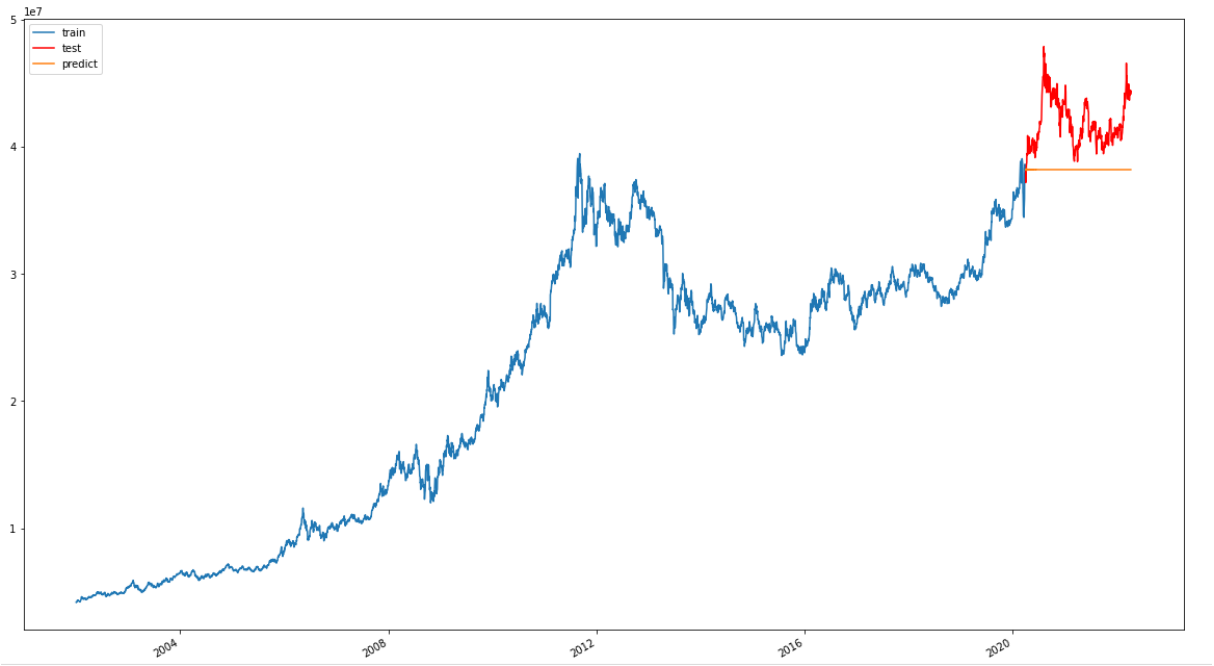


Figure 34: The best plot for the prediction of the best train test set.

Step 9: Prediction for the future

In this step we will use the model from the step 6 to predict the dataset in the next 30 days

```
prediction = model.predict(len(df),len(df) + 30)
prediction
```

```
2022-04-11    4.436569e+07
2022-04-12    4.438176e+07
2022-04-13    4.436444e+07
2022-04-14    4.436451e+07
2022-04-15    4.435674e+07
2022-04-18    4.435534e+07
2022-04-19    4.435103e+07
2022-04-20    4.434926e+07
2022-04-21    4.434625e+07
2022-04-22    4.434433e+07
2022-04-25    4.434184e+07
2022-04-26    4.433984e+07
2022-04-27    4.433758e+07
2022-04-28    4.433554e+07
2022-04-29    4.433338e+07
2022-05-02    4.433132e+07
2022-05-03    4.432920e+07
2022-05-04    4.432713e+07
2022-05-05    4.432503e+07
2022-05-06    4.432295e+07
2022-05-09    4.432086e+07
2022-05-10    4.431877e+07
2022-05-11    4.431669e+07
2022-05-12    4.431460e+07
2022-05-13    4.431252e+07
2022-05-16    4.431044e+07
2022-05-17    4.430835e+07
2022-05-18    4.430627e+07
2022-05-19    4.430419e+07
2022-05-20    4.430211e+07
2022-05-23    4.430002e+07
```

Figure 35: The data of the next 30 days

```
prediction.plot(legend = True,label = 'prediction')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3842de50d0>
```

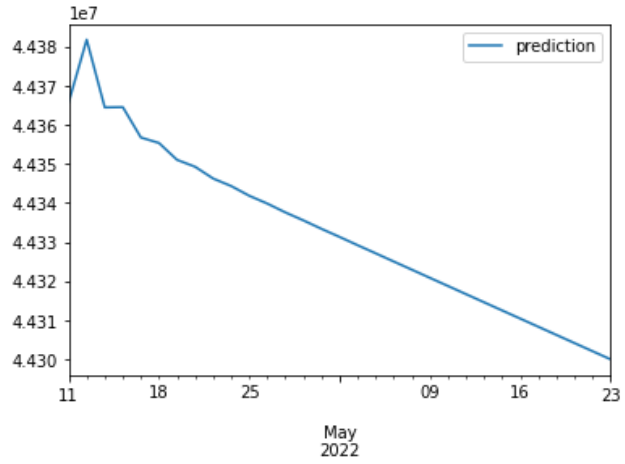


Figure 36: The best prediction for the best model.

4.4.1 Train 70% - Test 30%

First of all, we use len(df) to see the length of the datadet.

```
len(df)
```

```
5289
```

Figure 37: The lenght of the dataset

- With the train 70% and 30 % we use 5289*0.7 =3702 and the command ":3702" for the train and "3702:" for the test.

```
train = df[:3702]
test = df[3702:]
```

Figure 38: The commads for split train and test.

Step 6: Fit the model following to the train and test

```
model = ARIMA(df,order=(3,0,4)).fit()
```

```
print(model.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                 Prices   No. Observations:                 5289
Model:                     ARMA(3, 4)   Log Likelihood              -73667.724
Method:                       css-mle   S.D. of innovations         270694.849
Date:                Wed, 04 Jan 2023   AIC                         147353.447
Time:                        19:55:01   BIC                         147412.608
Sample:                    01-01-2002   HQIC                        147374.123
                         - 04-08-2022
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        2.274e+07    1.6e+07      1.425      0.154   -8.54e+06     5.4e+07
ar.L1.Prices     0.8611      0.001    741.601      0.000       0.859       0.863
ar.L2.Prices     0.5938      0.002    267.848      0.000       0.589       0.598
ar.L3.Prices    -0.4550      0.002   -240.305      0.000      -0.459      -0.451
ma.L1.Prices     0.1331      0.014      9.642      0.000       0.106       0.160
ma.L2.Prices    -0.4644      0.014    -33.526      0.000      -0.492      -0.437
ma.L3.Prices    -0.0024      0.014     -0.171      0.864      -0.030       0.025
ma.L4.Prices    -0.0411      0.013     -3.101      0.002      -0.067      -0.015
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -1.3378           +0.0000j            1.3378            0.5000
AR.2            1.0001           +0.0000j            1.0001            0.0000
AR.3            1.6428           +0.0000j            1.6428            0.0000
MA.1           -1.2576           -0.0000j            1.2576           -0.5000
MA.2            1.4658           -0.0000j            1.4658           -0.0000
MA.3           -0.1329           -3.6296j            3.6320           -0.2558
MA.4           -0.1329           +3.6296j            3.6320            0.2558
-----------------------------------------------------------------------------
```

Figure 39: The ARIMA model (3,0,4)

Step 7: Model evaluation:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error
mape = mean_absolute_percentage_error(test, pred)
mse = mean_squared_error(test, pred)
rmse = np.sqrt(mse)

print(f"MAPE: {mape * 100:.2f}%")
print(f"RMSE: {rmse:.2f}")
```

```
MAPE: 16.27%
RMSE: 8727545.10
```

Figure 40: The MAPE and RMSE.

Step 8: Prediction for train and test:

```
plt.plot(train.index, train['Prices'],label = 'train')
plt.plot(test.index, test['Prices'], color='r',label = 'test')
pred.plot(label = 'predict',figsize = (20,12))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f4c7860e280>
```

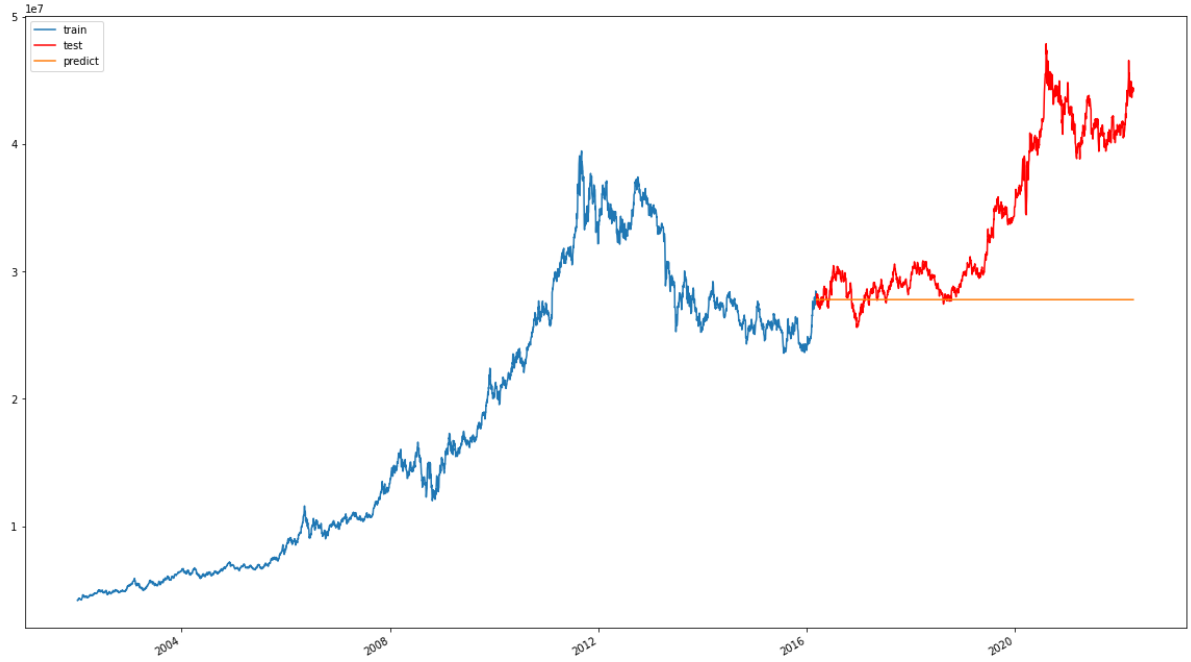


Figure 41: The plot for the 70% train and 30% test.

Step 9: Prediction for the future:

```
prediction = model.predict(len(df),len(df) + 30)
prediction
```

```
2022-04-11    4.436432e+07
2022-04-12    4.437891e+07
2022-04-13    4.436418e+07
2022-04-14    4.436301e+07
2022-04-15    4.435504e+07
2022-04-18    4.435419e+07
2022-04-19    4.434925e+07
2022-04-20    4.434812e+07
2022-04-21    4.434460e+07
2022-04-22    4.434315e+07
2022-04-25    4.434032e+07
2022-04-26    4.433862e+07
2022-04-27    4.433615e+07
2022-04-28    4.433429e+07
2022-04-29    4.433199e+07
2022-05-02    4.433004e+07
2022-05-03    4.432784e+07
2022-05-04    4.432583e+07
2022-05-05    4.432368e+07
2022-05-06    4.432164e+07
2022-05-09    4.431952e+07
2022-05-10    4.431746e+07
2022-05-11    4.431535e+07
2022-05-12    4.431328e+07
2022-05-13    4.431118e+07
2022-05-16    4.430911e+07
2022-05-17    4.430702e+07
2022-05-18    4.430494e+07
2022-05-19    4.430285e+07
2022-05-20    4.430077e+07
2022-05-23    4.429868e+07
Freq: B, dtype: float64
```

Figure 42: The data for model (3,0,4).

```
prediction.plot(legend = True,label = 'prediction')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c784a73a0>
```

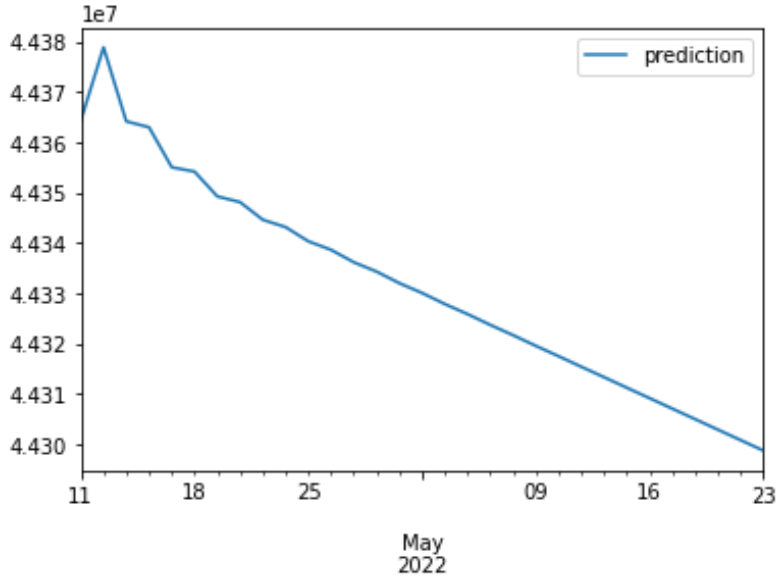


Figure 43: The plot of prediction for future.

4.4.2 Train 80% - Test 20%

First of all, we use len(df) to see the length of the datadet.

```
len(df)
```

```
5289
```

Figure 44: the len of the dataset.

With the train 80% and 20 % we use 5289*0.7 =4231 and the command ":4231" for the train and "4231:" for the test.

```
train = df[:4231]
test = df[4231:]
```

After train and test we can draw the plot by the command "plt".

Step 6: Fit the model following to the train and test

```
model = ARIMA(train,order=(5,0,4)).fit()
```

```
print(model.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                 Prices   No. Observations:                 5289
Model:                     ARMA(5, 4)   Log Likelihood              -73660.111
Method:                       css-mle   S.D. of innovations         270212.214
Date:                Wed, 04 Jan 2023   AIC                         147342.222
Time:                        20:11:11   BIC                         147414.529
Sample:                    01-01-2002   HQIC                        147367.492
                         - 04-08-2022
==============================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         2.274e+07        nan        nan        nan         nan         nan
ar.L1.Prices     0.7295        nan        nan        nan         nan         nan
ar.L2.Prices     1.5639   3.23e-05   4.84e+04      0.000       1.564       1.564
ar.L3.Prices    -1.3528   3.39e-05  -3.99e+04      0.000      -1.353      -1.353
ar.L4.Prices    -0.7733        nan        nan        nan         nan         nan
ar.L5.Prices     0.8327        nan        nan        nan         nan         nan
ma.L1.Prices     0.2756      0.008     35.362      0.000       0.260       0.291
ma.L2.Prices    -1.3108      0.009   -151.697      0.000      -1.328      -1.294
ma.L3.Prices     0.0434      0.009      5.013      0.000       0.026       0.060
ma.L4.Prices     0.8240      0.008    105.590      0.000       0.809       0.839
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -0.9861           -0.2484j            1.0169           -0.4607
AR.2           -0.9861           +0.2484j            1.0169            0.4607
AR.3            0.9504           -0.5081j            1.0777           -0.0781
AR.4            0.9504           +0.5081j            1.0777            0.0781
AR.5            1.0000           -0.0000j            1.0000           -0.0000
MA.1            0.9614           -0.5005j            1.0839           -0.0764
MA.2            0.9614           +0.5005j            1.0839            0.0764
MA.3           -0.9878           -0.2395j            1.0164           -0.4621
MA.4           -0.9878           +0.2395j            1.0164            0.4621
-----------------------------------------------------------------------------
```

Figure 45: The ARIMA model (5,0,4).

Step 7: Model evaluation:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error
mape = mean_absolute_percentage_error(test, pred)
mse = mean_squared_error(test, pred)
rmse = np.sqrt(mse)

print(f"MAPE: {mape * 100:.2f}%")
print(f"RMSE: {rmse:.2f}")
```

```
MAPE: 18.05%
RMSE: 8984086.99
```

Figure 46: The MAPE and RMSE.

Step 8: Prediction for train and test:

```
plt.plot(train.index, train['Prices'],label = 'train')
plt.plot(test.index, test['Prices'], color='r',label = 'test')
pred.plot(label = 'predict',figsize = (20,12))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f2297e3d3d0>
```

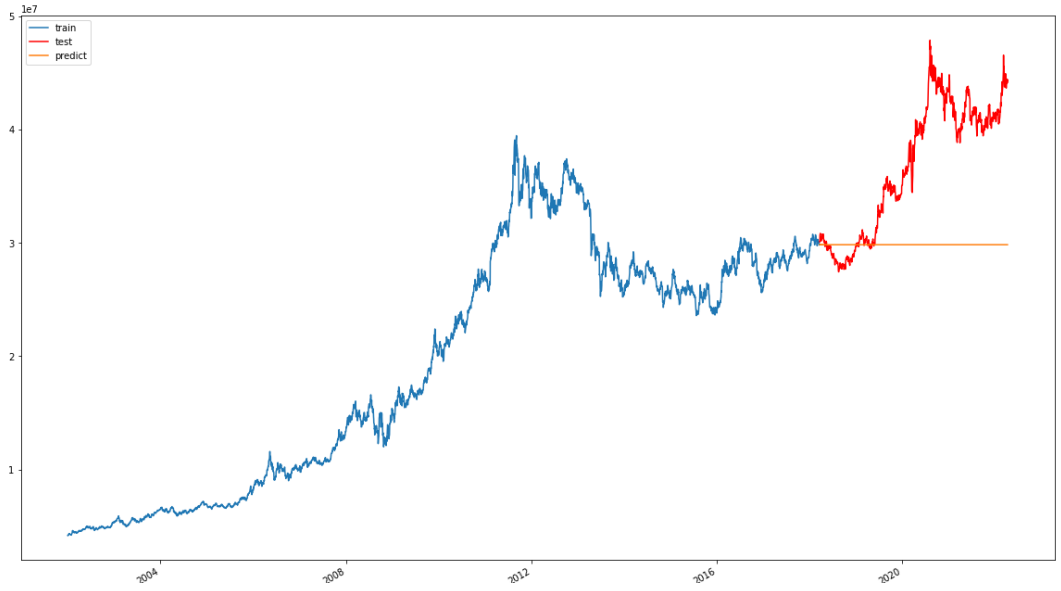


Figure 47: The plot for the 80% train and 20% test.

Step 9: Prediction for the future:

```
prediction = model.predict(len(df),len(df) + 30)
prediction
```

```
2022-04-11    4.437473e+07
2022-04-12    4.438436e+07
2022-04-13    4.435523e+07
2022-04-14    4.435888e+07
2022-04-15    4.432755e+07
2022-04-18    4.433440e+07
2022-04-19    4.431600e+07
2022-04-20    4.432860e+07
2022-04-21    4.432702e+07
2022-04-22    4.433908e+07
2022-04-25    4.434828e+07
2022-04-26    4.435093e+07
2022-04-27    4.436266e+07
2022-04-28    4.435226e+07
2022-04-29    4.436236e+07
2022-05-02    4.434321e+07
2022-05-03    4.435225e+07
2022-05-04    4.433305e+07
2022-05-05    4.434260e+07
2022-05-06    4.433052e+07
2022-05-09    4.433970e+07
2022-05-10    4.433696e+07
2022-05-11    4.434227e+07
2022-05-12    4.434675e+07
2022-05-13    4.434487e+07
2022-05-16    4.435307e+07
2022-05-17    4.434367e+07
2022-05-18    4.435315e+07
2022-05-19    4.433945e+07
2022-05-20    4.434908e+07
2022-05-23    4.433596e+07
```

Figure 48: The data for model (5,0,4)

```
prediction.plot(legend = True,label = 'prediction')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2297cabfd0>
```

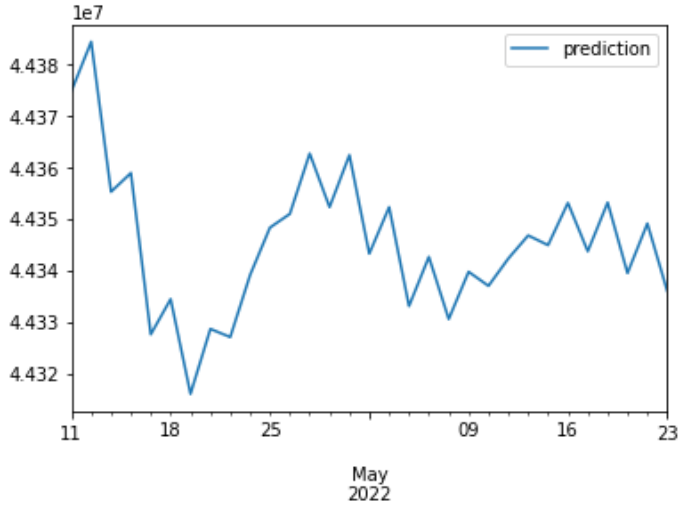


Figure 49: The plot of prediction for future.

4.4.3 Train 90% - Test 10%

First of all, we use len(df) to see the length of the datadet.

```
len(df)
```

```
5289
```

Figure 50: the length of the dataset

With the train 90% and 10 % we use 5289*0.7 =4760 and the command ":4760" for the train and "4760:" for the test.

```
train = df[:4760]
test = df[4760:]
```

Figure 51: the command to split train and test

After train and test we can draw the plot by the command "plt"

Step 6: Fit the model following to the train and test

```
# Setting up ARIMA model
model = ARIMA(train,order=(6,0,2)).fit()
```

```
print(model.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                 Prices   No. Observations:                 4760
Model:                     ARMA(6, 2)   Log Likelihood              -65849.718
Method:                        css-mle   S.D. of innovations         246109.654
Date:                Wed, 04 Jan 2023   AIC                         131719.436
Time:                        16:39:18   BIC                         131784.116
Sample:                    01-01-2002   HQIC                        131742.161
                         - 03-30-2020
==============================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         2.061e+07        nan        nan        nan         nan         nan
ar.L1.Prices    -0.8234        nan        nan        nan         nan         nan
ar.L2.Prices     0.9396   2.38e-05   3.94e+04      0.000       0.940       0.940
ar.L3.Prices     0.8837   3.07e-05   2.88e+04      0.000       0.884       0.884
ar.L4.Prices    -0.0183      0.000    -60.784      0.000      -0.019      -0.018
ar.L5.Prices     0.0112        nan        nan        nan         nan         nan
ar.L6.Prices     0.0071        nan        nan        nan         nan         nan
ma.L1.Prices     1.8326      0.007    278.826      0.000       1.820       1.846
ma.L2.Prices     0.8957      0.007    136.771      0.000       0.883       0.909
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0000           -0.0000j            1.0000           -0.0000
AR.2           -1.0099           -0.2782j            1.0475           -0.4572
AR.3           -1.0099           +0.2782j            1.0475            0.4572
AR.4            2.4155           -4.2433j            4.8826           -0.1676
AR.5            2.4155           +4.2433j            4.8826            0.1676
AR.6           -5.3978           -0.0000j            5.3978           -0.5000
MA.1           -1.0230           -0.2643j            1.0566           -0.4598
MA.2           -1.0230           +0.2643j            1.0566            0.4598
-----------------------------------------------------------------------------
```

Figure 52: The ARIMA model (6,0,2)

Step 7: Model evaluation:

```
mape = mean_absolute_percentage_error(test, pred)
mse = mean_squared_error(test, pred)
rmse = np.sqrt(mse)

print(f"MAPE: {mape * 100:.2f}%")
print(f"RMSE: {rmse:.2f}")
```

```
MAPE: 8.73%
RMSE: 4149475.21
```

Figure 53: The MAPE and RMSE

Step 8: Prediction for train and test:

```
plt.plot(train.index, train['Prices'],label = 'train')
plt.plot(test.index, test['Prices'], color='r',label = 'test')
pred.plot(label = 'predict',figsize = (20,12))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f3842e63f40>
```

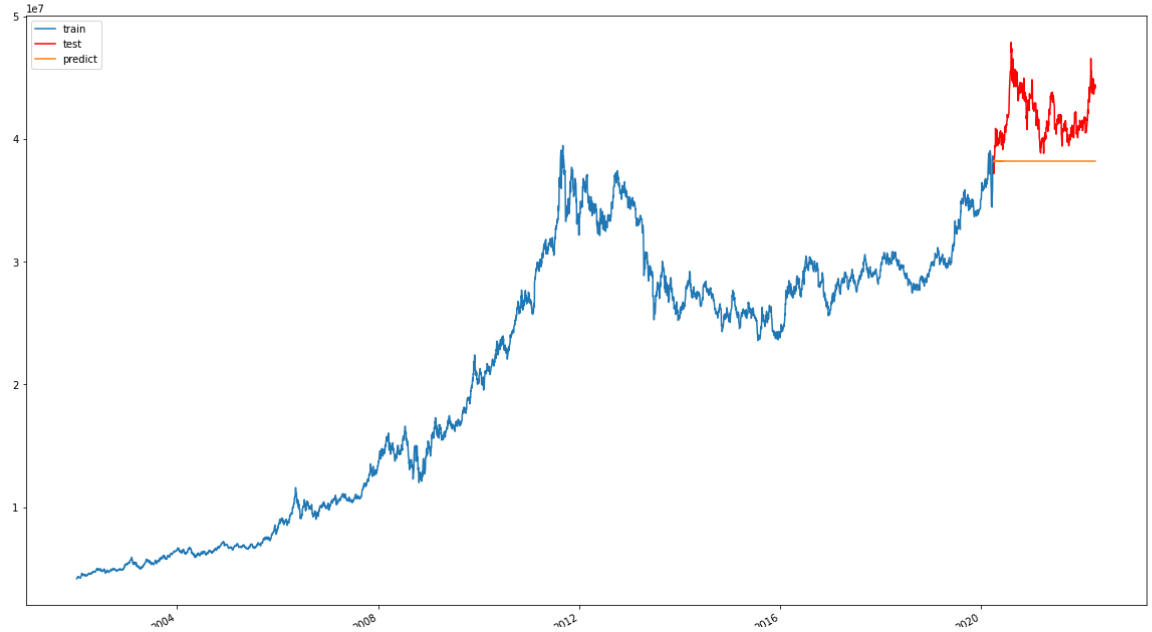


Figure 54: The plot for the 90% train and 10% test

Step 9: Prediction for the future:

```
prediction = model.predict(len(df),len(df) + 30)
prediction
```

```
2022-04-11    4.436569e+07
2022-04-12    4.438176e+07
2022-04-13    4.436444e+07
2022-04-14    4.436451e+07
2022-04-15    4.435674e+07
2022-04-18    4.435534e+07
2022-04-19    4.435103e+07
2022-04-20    4.434926e+07
2022-04-21    4.434625e+07
2022-04-22    4.434433e+07
2022-04-25    4.434184e+07
2022-04-26    4.433984e+07
2022-04-27    4.433758e+07
2022-04-28    4.433554e+07
2022-04-29    4.433338e+07
2022-05-02    4.433132e+07
2022-05-03    4.432920e+07
2022-05-04    4.432713e+07
2022-05-05    4.432503e+07
2022-05-06    4.432295e+07
2022-05-09    4.432086e+07
2022-05-10    4.431877e+07
2022-05-11    4.431669e+07
2022-05-12    4.431460e+07
2022-05-13    4.431252e+07
2022-05-16    4.431044e+07
2022-05-17    4.430835e+07
2022-05-18    4.430627e+07
2022-05-19    4.430419e+07
2022-05-20    4.430211e+07
2022-05-23    4.430002e+07
```

Figure 55: The data for model (6,0,2)

```
prediction.plot(legend = True,label = 'prediction')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3842de50d0>
```

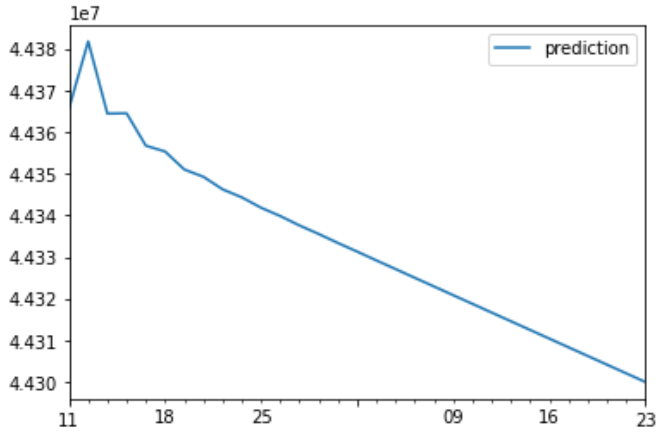


Figure 56: The plot of prediction for future

CHAPTER 5: FBPROPHET

5.1 Model definition

Facebook Prophet is an open-source program created by the company that analyzes time series data using a decomposable additive model. It takes holidays into account and often fits nonlinear data with seasonality on an annual, monthly, and daily basis. Prophet fits a variety of linear and nonlinear time functions as components, combining them into the following equation [5]:

$$y(t) = g(t) + s(t) + h(t) + e(t) \qquad (8)$$

where:

- y(t): predictions (forecast).
- g(t): trend alludes to changes throughout a significant stretch of time
- s(t): seasonality weekly, daily, yearly.
- h(t): holidays
- e(t): error term represents any surprising changes not obliged by the model

The Facebook prophet has multiple trends, seasonality, change points, and holiday parameters which needs to be tuned for better results. The goal of Prophet is to tease out the signal in a data set and forecast that signal into the future. The approach to getting at that signal of Prophet is breaking down the signal into three pieces trend, season, and holiday.

The logistic growth model [7]:

$$g(t) = \frac{C}{1 - e^{-k(t-m)}}$$

where:

- C: carry capacity
- k: growth rate
- m: an offset parameter

The linear growth model [8]:

$$y = \{\beta_0 + \beta_1 x\ \beta_0 - \beta_2 x + (\beta_1 + \beta_2) x\ \ x \leq c\ x > c$$

5.2Code execution

5.2.1 Train 70% - Test 30%

Step1: Import data and libraies

```
from google.colab import drive
drive.mount('/content/drive') 123
```

```
Mounted at /content/drive
```

```
[ ] !pip install pystan~=2.14
    !pip install fbprophet
```

Figure 57: Connect to drive and Install Prophet for time series analysis and forecasting.

```
#import libraries
import itertools
from fbprophet import Prophet
import pandas as pd
import numpy as np
```

Figure 58: Import libraries.

```
forecast.tail(5)
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	daily	daily_lower	daily_upper	multiplicative_terms	multiplicative_terms_lower	m
1581	2022-04-04	2.138664e+06	-9.056076e+07	1.212413e+08	-1.211023e+07	1.636378e+07	6.42677	6.42677	6.42677	6.42677	6.42677	
1582	2022-04-05	2.138137e+06	-9.021222e+07	1.223311e+08	-1.212058e+07	1.637610e+07	6.42677	6.42677	6.42677	6.42677	6.42677	
1583	2022-04-06	2.137610e+06	-8.988936e+07	1.219001e+08	-1.213094e+07	1.638842e+07	6.42677	6.42677	6.42677	6.42677	6.42677	
1584	2022-04-07	2.137083e+06	-8.965894e+07	1.216395e+08	-1.214129e+07	1.640074e+07	6.42677	6.42677	6.42677	6.42677	6.42677	
1585	2022-04-08	2.136556e+06	-9.093449e+07	1.226540e+08	-1.215164e+07	1.641305e+07	6.42677	6.42677	6.42677	6.42677	6.42677	

Figure 59: Get the data from csv file.

Step 2: Data Transformation

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5289 entries, 0 to 5288
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5289 non-null   object
 1   Prices  5289 non-null   float64
dtypes: float64(1), object(1)
memory usage: 82.8+ KB
```

Figure 60: Data structure information.

In the figure 4, the data has 2 columns the Date and the Prices column. The data type of Date column is object type so we need to transfrom the data type of the Date column to Datetime

```
#Date variable
df.Date = pd.to_datetime(df.Date,
                                    format = "%Y-%m-%d")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5289 entries, 0 to 5288
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5289 non-null   datetime64[ns]
 1   Prices  5289 non-null   float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 82.8 KB
```

Figure 61: Data Transformation.

Facebook Prophet is very delicate when is comes to names, The date variable must be name ‘ds’ and the time series is ‘Y’

Ds has date format, timestamps

Y pressents the quantitative value, which represents the measurement we predict

```
#renaming variable
dataset = dataset.rename(columns = {'Close' : 'y'})
dataset = dataset.rename(columns = {'Date' : 'ds'})
dataset.head(5)
```

	ds	y
0	2018-01-01	13657.200195
1	2018-01-02	14982.099609
2	2018-01-03	15201.000000
3	2018-01-04	15599.200195
4	2018-01-05	17429.500000

Figure 62: Renaming and data after transformation.

Step 3: Train test split

Data is splited in to 70% for training and 30% for testing. In this case there is 5289 rows so the first 3703 days is traning data and the next 1586 days is for testing

```
#Training and test set
test_days = 1586
training_set = df.iloc[:-test_days, :]
test_set = df.iloc[-test_days:, :]
training_set.tail(5)
```

	ds	y
3698	2016-03-04	28472281.25
3699	2016-03-07	28267828.32
3700	2016-03-08	28254100.00
3701	2016-03-09	27791601.82
3702	2016-03-10	28216353.50

Figure 63: Data split into 70% traning and 30% testing.

Step 4: Model Creation and forecasting on the tesing set

We are using test set dataframe of forecasting.

Including some parameters:

- m: model Prophet

- periods: predicted period
- freq: 'day', 'week', 'month', 'quarter', 'year', 1(1 sec), 60(1 minute) or 3600(1 hour).
- include_history : Boolean data type, check dataset containing dataframe about historical dates

```
#Facebook Prophet model
m = Prophet(growth = "linear",
            yearly_seasonality = False,
            weekly_seasonality = False,
            daily_seasonality = True,
            seasonality_mode = "multiplicative",
            seasonality_prior_scale = 10,
            holidays_prior_scale = 10,
            changepoint_prior_scale = 0.05)
m.fit(training_set)
forecast = m.predict(test_set)
```

Figure 64: Creating model.

```
forecast.tail(5)
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	daily	...
1053	2022-04-04	3.545664e+07	2.133417e+06	6.894995e+07	1.397835e+06	6.829206e+07	346169.067258	346169.067258	346169.067258	125034.658512	...
1054	2022-04-05	3.546054e+07	1.804784e+06	6.922671e+07	1.353685e+06	6.833273e+07	340339.582295	340339.582295	340339.582295	125034.658512	...
1055	2022-04-06	3.546445e+07	1.464003e+06	6.890218e+07	1.309534e+06	6.838562e+07	327189.319140	327189.319140	327189.319140	125034.658512	...
1056	2022-04-07	3.546835e+07	1.732767e+06	6.885414e+07	1.265383e+06	6.843851e+07	313743.536853	313743.536853	313743.536853	125034.658512	...
1057	2022-04-08	3.547225e+07	1.630248e+06	6.911375e+07	1.225754e+06	6.849140e+07	311559.380992	311559.380992	311559.380992	125034.658512	...

5 rows × 22 columns

Figure 65: The result.

There are 3 features that we consider in this dataframe

- Yhat: The predicted value
- Yhat_lower: The lower bound of predicted value
- Yhat_upper The upper bound of predicted value

Step 5: Model Evaluation

```
[ ]  mae = mean_absolute_error(test_set['y'], forecast['yhat'])
     mape = mean_absolute_percentage_error(test_set['y'], forecast['yhat'])

     print(f"MAE: {mae:.2f}")
     print(f"MAPE: {mape * 100:.2f}%")
```

```
MAE: 4862758.69
MAPE: 12.21%
```

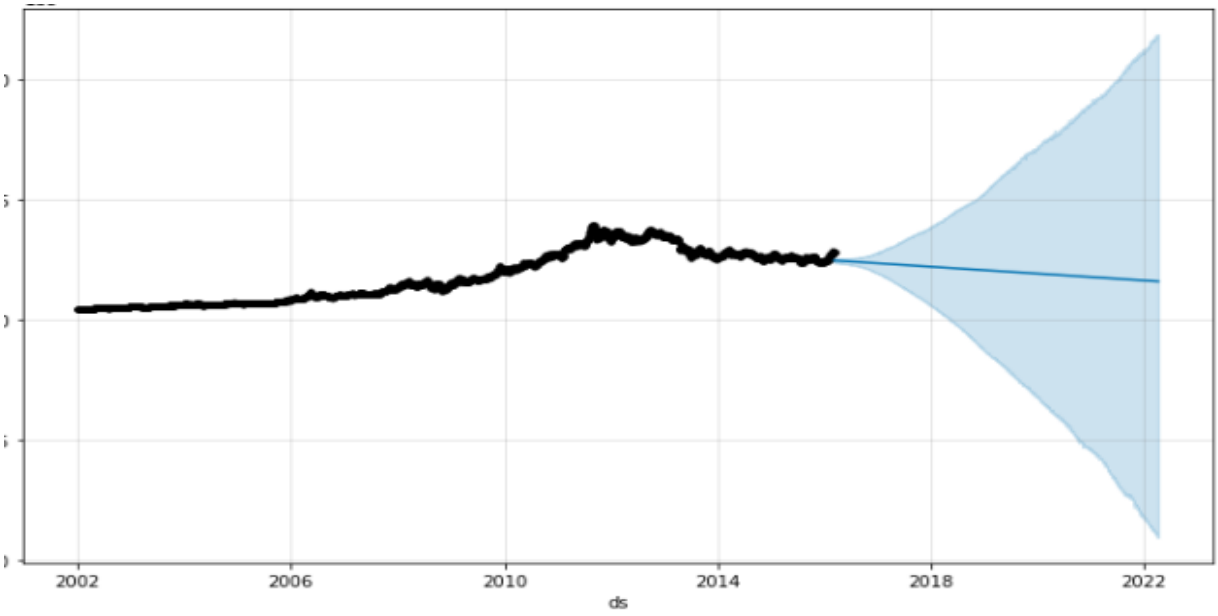


Figure 66: Predicted value on test set.

Step 6: Forecasting the future

Because we are predicting the future so we have to create the dataframe that is from the last day of training set till 8/5/2022 (30 days after real data)

```
#Create Future Dataframe
future = m.make_future_dataframe(periods = 2250, # Day until 30 days after 8/4/2022
                                 freq = "D")
future
```

	ds
0	2002-01-01
1	2002-01-02
2	2002-01-03
3	2002-01-04
4	2002-01-07
...	...
5948	2022-05-04
5949	2022-05-05
5950	2022-05-06
5951	2022-05-07
5952	2022-05-08

5953 rows × 1 columns

Merge columns

Figure 67: Creating future dataframe for predicting.

```
#forecast
forecast = m.predict(future)
```

Predict data

```
#predictions
predictions_prophet = forecast.yhat[-test_days:].rename("prophet")
predictions_prophet[:5]
```

```
4683    3.147732e+07
4684    3.149310e+07
4685    3.148555e+07
4686    3.147113e+07
4687    3.145693e+07
Name: prophet, dtype: float64
```

Figure 68: Forecasting and print the predictions.

Step 7: Forecast visualization

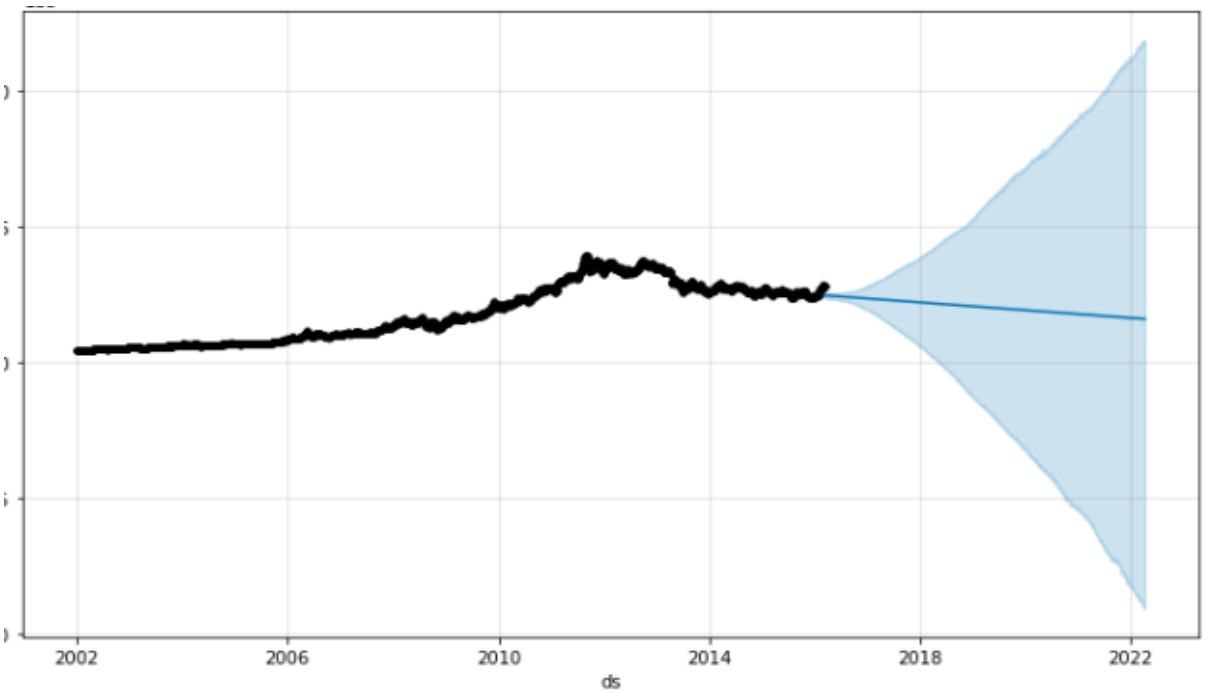


Figure 69: Forecasting.

```
from fbprophet.plot import plot_plotly
plot_plotly(m,forecast)
```

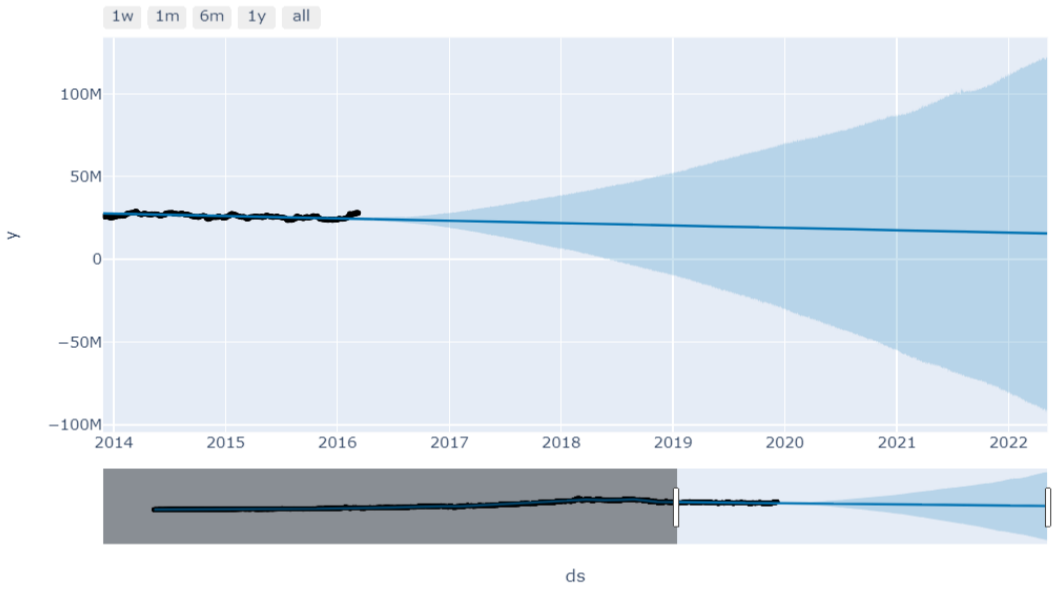


Figure 70: Forecasting in 30 days.

```
m.plot_components(forecast);
```

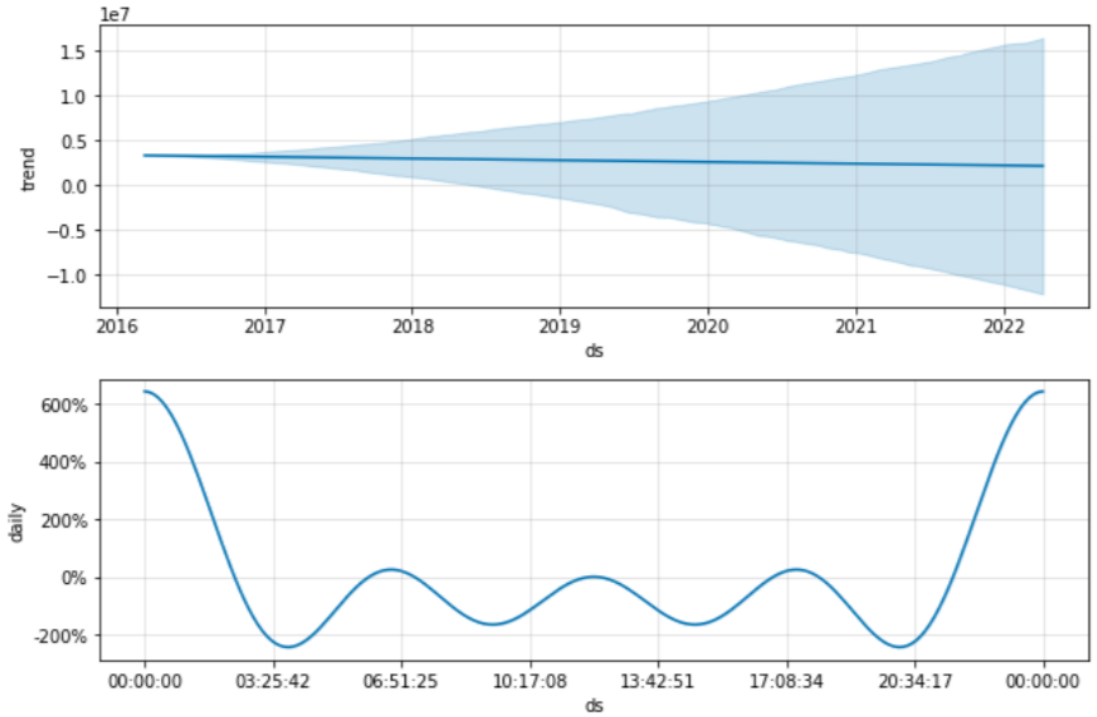


Figure 71: Trend and seasonality of year, week and day.

```
[73] #Print out predicted value of one day
     forecast[forecast.ds == '2022-04-08']['yhat']

     5922    1.586771e+07
     Name: yhat, dtype: float64
```

```
#Print out actual value of one day
test_set[test_set.ds=='2022-04-08']['y']

5288    44384284.57
Name: y, dtype: float64
```

```
#MAE and RMSE
from sklearn.metrics import mean_squared_error, mean_absolute_error
print(round(mean_absolute_error(test_set['y'], predictions_prophet),0))
print(round(np.sqrt(mean_squared_error(test_set['y'], predictions_prophet)), 0))

15204209.0
17056056.0
```

Figure 72: Model Evaluation.

5.2.2 Train 80% - Test 20%

Step1: Import data and libraies

```
from google.colab import drive
drive.mount('/content/drive') 123
```

```
Mounted at /content/drive
```

```
[ ] !pip install pystan~=2.14
    !pip install fbprophet
```

Figure 73: Connect to drive and Install Prophet for time series analysis and forecasting.

```
#import libraries
import itertools
from fbprophet import Prophet
import pandas as pd
import numpy as np
```

Figure 74: Import libraries.

```
#get the data
data = pd.read_csv("/content/drive/My Drive/Năm 4/Làm nhóm HK1 2022-2023/Facebook Prophet/BTC_TimeSeries.csv")
data.head(5)
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2018-01-01	14112.200195	14112.200195	13154.700195	13657.200195	10291200000	0.0	0.0
1	2018-01-02	13625.000000	15444.599609	13163.599609	14982.099609	16846600192	0.0	0.0
2	2018-01-03	14978.200195	15572.799805	14844.500000	15201.000000	16871900160	0.0	0.0
3	2018-01-04	15270.700195	15739.700195	14522.200195	15599.200195	21783199744	0.0	0.0
4	2018-01-05	15477.200195	17705.199219	15202.799805	17429.500000	23840899072	0.0	0.0

Figure 75: Get the data from csv file.

Step 2: Data Transformation

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5289 entries, 0 to 5288
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5289 non-null   object
 1   Prices  5289 non-null   float64
dtypes: float64(1), object(1)
memory usage: 82.8+ KB
```

Figure 76: Data structure information.

In the figure 4, the data has 2 columns the Date and the Prices column. The data type of Date column is object type so we need to transfrom the data type of the Date column to Datetime

```
#Date variable
df.Date = pd.to_datetime(df.Date,
                                 format = "%Y-%m-%d")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5289 entries, 0 to 5288
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5289 non-null   datetime64[ns]
 1   Prices  5289 non-null   float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 82.8 KB
```

Figure 77: Data Transformation.

Facebook Prophet is very delicate when is comes to names, The date variable must be name ‘ds’ and the time series is ‘Y’

Ds has date format, timestamps

Y pressents the quantitative value, which represents the measurement we predict

```
#renaming variable
dataset = dataset.rename(columns = {'Close' : 'y'})
dataset = dataset.rename(columns = {'Date' : 'ds'})
dataset.head(5)
```

	ds	y
0	2018-01-01	13657.200195
1	2018-01-02	14982.099609
2	2018-01-03	15201.000000
3	2018-01-04	15599.200195
4	2018-01-05	17429.500000

Figure 78: Renaming and data after transformation.

Step 3: Train test split

Data is splited in to 80% for training and 20% for testing. In this case there is 5289 rows so the first 4231 days is traning data and the next 1058 days is for testing

```
[ ] #Training and test set
    test_days = 1058
    training_set = df.iloc[:-test_days, :]
    test_set = df.iloc[-test_days:, :]
    training_set.tail(5)
```

	ds	y
4226	2018-03-14	30132596.63
4227	2018-03-15	30024640.63
4228	2018-03-16	29823768.12
4229	2018-03-19	29879408.58
4230	2018-03-20	29857369.50

Figure 79: Data split into 70% traning and 30% testing.

Step 4: Model Creation and forecasting on the tesing set

We are using test set dataframe of forecasting.

Including some parameters:

- m: model Prophet
- periods: predicted period
- freq: 'day', 'week', 'month', 'quarter', 'year', 1(1 sec), 60(1 minute) or 3600(1 hour).
- include_history : Boolean data type, check dataset containing dataframe about historical dates

```
#Facebook Prophet model
m = Prophet(growth = "linear",
            yearly_seasonality = False,
            weekly_seasonality = False,
            daily_seasonality = True,
            seasonality_mode = "multiplicative",
            seasonality_prior_scale = 10,
            holidays_prior_scale = 10,
            changepoint_prior_scale = 0.05)
m.fit(training_set)
forecast = m.predict(test_set)
```

Figure 80: Creating model.

```
forecast.tail(5)
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	daily	...
1053	2022-04-04	3.545664e+07	2.133417e+06	6.894995e+07	1.397835e+06	6.829206e+07	346169.067258	346169.067258	346169.067258	125034.658512	...
1054	2022-04-05	3.546054e+07	1.804784e+06	6.922671e+07	1.353685e+06	6.833273e+07	340339.582295	340339.582295	340339.582295	125034.658512	...
1055	2022-04-06	3.546445e+07	1.464003e+06	6.890218e+07	1.309534e+06	6.838562e+07	327189.319140	327189.319140	327189.319140	125034.658512	...
1056	2022-04-07	3.546835e+07	1.732767e+06	6.885414e+07	1.265383e+06	6.843851e+07	313743.536853	313743.536853	313743.536853	125034.658512	...
1057	2022-04-08	3.547225e+07	1.630248e+06	6.911375e+07	1.225754e+06	6.849140e+07	311559.380992	311559.380992	311559.380992	125034.658512	...

5 rows × 22 columns

Figure 81: The result.

There are 3 features that we consider in this dataframe

- Yhat: The predicted value
- Yhat_lower: The lower bound of predicted value
- Yhat_upper The upper bound of predicted value

Step 5: Model Evaluation

```
[ ] mae = mean_absolute_error(test_set['y'], forecast['yhat'])
    mape = mean_absolute_percentage_error(test_set['y'], forecast['yhat'])

    print(f"MAE: {mae:.2f}")
    print(f"MAPE: {mape * 100:.2f}%")
```

```
MAE: 4862758.69
MAPE: 12.21%
```

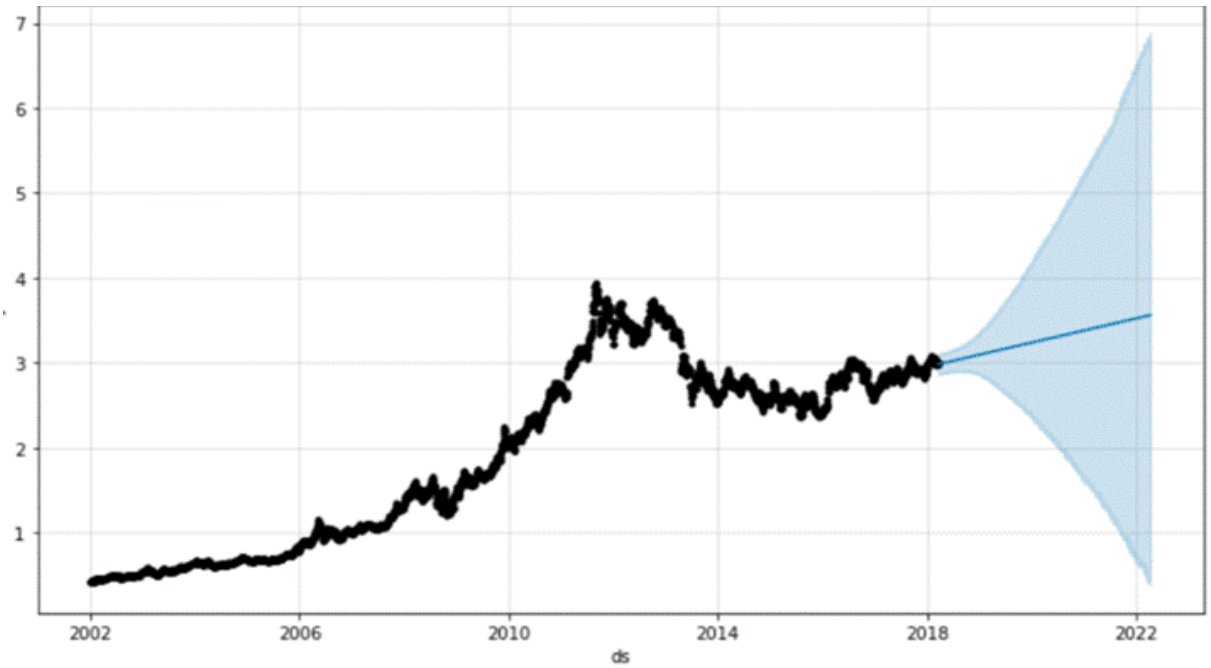


Figure 82: Predicted value on test set.

Step 6: Forecasting the future

Because we are predicting the future so we have to create the dataframe that is from the last day of training set till 8/5/2022 (30 days after real data)

```
#Create Future Dataframe
future = m.make_future_dataframe(periods = 1510, # Day until 30 days after 8/4/2022
                                 freq = "D")
future
```

	ds
0	2002-01-01
1	2002-01-02
2	2002-01-03
3	2002-01-04
4	2002-01-07
...	...
5736	2022-05-04
5737	2022-05-05
5738	2022-05-06
5739	2022-05-07
5740	2022-05-08

5741 rows × 1 columns

Figure 83: Creating future dataframe for predicting.

```
[ ] #forecast
    forecast = m.predict(future)
```

Predict data

```
[ ] #predictions
    predictions_prophet = forecast.yhat[-test_days:].rename("prophet")
    predictions_prophet[:5]
```

```
4683    3.147732e+07
4684    3.149310e+07
4685    3.148555e+07
4686    3.147113e+07
4687    3.145693e+07
Name: prophet, dtype: float64
```

Figure 84: Forecasting and print the predictions.

Step 7: Forecast visualization

```
#visualization forecast
m.plot(forecast);
```

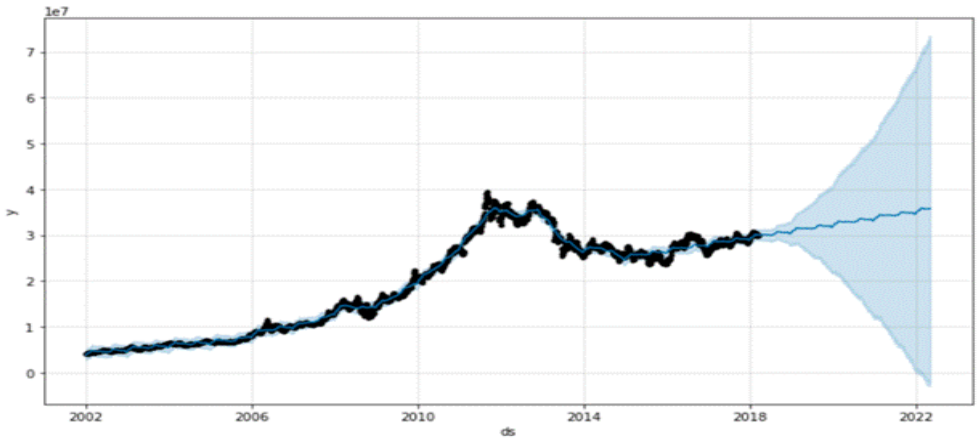


Figure 85: Forecasting.

```
from fbprophet.plot import plot_plotly
plot_plotly(m,forecast)
```

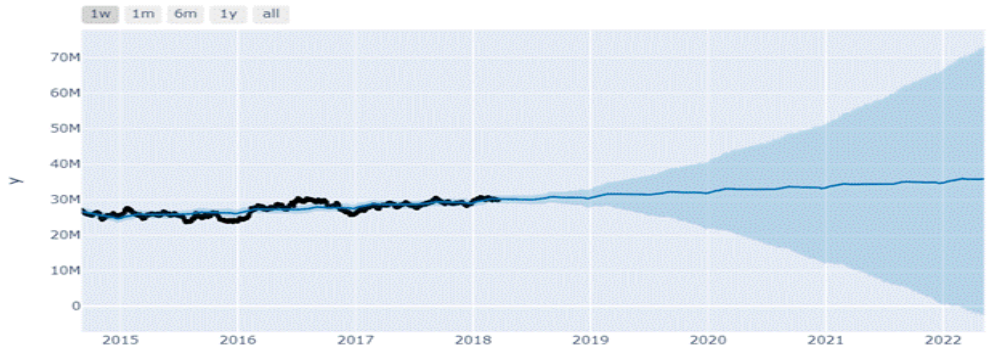


Figure 86: Forecasting in 30 days.

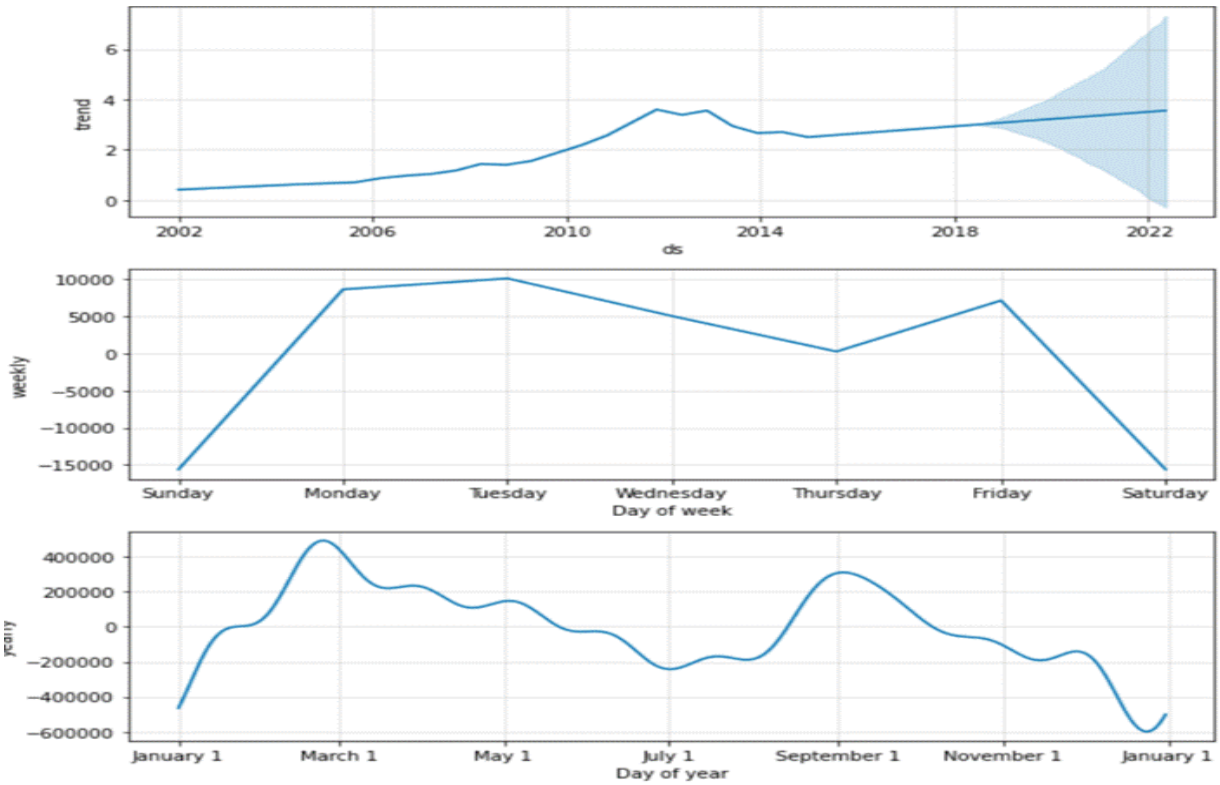


Figure 87: Trend and seasonality of year, week and day.

▾ Model Evaluation

```
[ ] #Print out predicted value of one day
    forecast[forecast.ds == '2022-04-08']['yhat']
```

```
5710    3.578381e+07
Name: yhat, dtype: float64
```

```
[ ] #Print out actual value of one day
    test_set[test_set.ds=='2022-04-08']['y']
```

```
5288    44384284.57
Name: y, dtype: float64
```

```
#MAE and RMSE
from sklearn.metrics import mean_squared_error, mean_absolute_error
print(round(mean_absolute_error(test_set['y'], predictions_prophet),0))
print(round(np.sqrt(mean_squared_error(test_set['y'], predictions_prophet)), 0))
```

```
4843316.0
5625390.0
```

5.2.3 Train 90% - Test 10%

Step 1: Import data and libraies

```
from google.colab import drive
drive.mount('/content/drive') 123
```

```
Mounted at /content/drive
```

```
[ ] !pip install pystan~=2.14
    !pip install fbprophet
```

Figure 88: Connect to drive and Install Prophet for time series analysis and forecasting.

```
#import libraries
import itertools
from fbprophet import Prophet
import pandas as pd
import numpy as np
```

Figure 89: Import libraries.

```
#get the data
data = pd.read_csv("/content/drive/My Drive/Năm 4/Làm nhóm HK1 2022-2023/Facebook Prophet/BTC_TimeSeries.csv")
data.head(5)
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2018-01-01	14112.200195	14112.200195	13154.700195	13657.200195	10291200000	0.0	0.0
1	2018-01-02	13625.000000	15444.599609	13163.599609	14982.099609	16846600192	0.0	0.0
2	2018-01-03	14978.200195	15572.799805	14844.500000	15201.000000	16871900160	0.0	0.0
3	2018-01-04	15270.700195	15739.700195	14522.200195	15599.200195	21783199744	0.0	0.0
4	2018-01-05	15477.200195	17705.199219	15202.799805	17429.500000	23840899072	0.0	0.0

Figure 90: Get the data from csv file.

Step 2: Data Transformation

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5289 entries, 0 to 5288
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5289 non-null   object
 1   Prices  5289 non-null   float64
dtypes: float64(1), object(1)
memory usage: 82.8+ KB
```

Figure 91: Data structure information.

In the figure 4, the data has 2 columns the Date and the Prices column. The data type of Date column is object type so we need to transfrom the data type of the Date column to Datetime

```
#Date variable
df.Date = pd.to_datetime(df.Date,
                                    format = "%Y-%m-%d")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5289 entries, 0 to 5288
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5289 non-null   datetime64[ns]
 1   Prices  5289 non-null   float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 82.8 KB
```

Figure 92: Data Transformation.

Facebook Prophet is very delicate when is comes to names, The date variable must be name ‘ds’ and the time series is ‘Y’

Ds has date format, timestamps

Y pressents the quantitative value, which represents the measurement we predict

```
#renaming variable
dataset = dataset.rename(columns = {'Close' : 'y'})
dataset = dataset.rename(columns = {'Date' : 'ds'})
dataset.head(5)
```

	ds	y
0	2018-01-01	13657.200195
1	2018-01-02	14982.099609
2	2018-01-03	15201.000000
3	2018-01-04	15599.200195
4	2018-01-05	17429.500000

Figure 93: Renaming and data after transformation.

Step 3: Train test split

Data is splited in to 90% for training and 10% for testing. In this case there is 5289 rows so the first 4760 days is traning data and the next 528 days is for testing

```
[ ] #Training and test set
    test_days = 1058
    training_set = df.iloc[:-test_days, :]
    test_set = df.iloc[-test_days:, :]
    training_set.tail(5)
```

	ds	y
4226	2018-03-14	30132596.63
4227	2018-03-15	30024640.63
4228	2018-03-16	29823768.12
4229	2018-03-19	29879408.58
4230	2018-03-20	29857369.50

Figure 94: Data split into 90% traning and 10% testing.

Step 4: Model Creation and forecasting on the tesing set

We are using test set dataframe of forecasting.

Including some parameters:

- m: model Prophet
- periods: predicted period
- freq: 'day', 'week', 'month', 'quarter', 'year', 1(1 sec), 60(1 minute) or 3600(1 hour).
- include_history : Boolean data type, check dataset containing dataframe about historical dates

```
#Facebook Prophet model
m = Prophet(growth = "linear",
            yearly_seasonality = False,
            weekly_seasonality = False,
            daily_seasonality = True,
            seasonality_mode = "multiplicative",
            seasonality_prior_scale = 10,
            holidays_prior_scale = 10,
            changepoint_prior_scale = 0.05)
m.fit(training_set)
forecast = m.predict(test_set)
```

Figure 95: Creating model.

```
forecast.tail(5)
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	daily	daily_lower	daily_upper	multiplicative_terms
523	2022-04-04	4.758465e+06	2.587985e+07	4.946580e+07	3.299330e+06	6.254176e+06	6.847959	6.847959	6.847959	6.847959
524	2022-04-05	4.759093e+06	2.606254e+07	4.889824e+07	3.296172e+06	6.257885e+06	6.847959	6.847959	6.847959	6.847959
525	2022-04-06	4.759721e+06	2.575146e+07	4.904676e+07	3.293014e+06	6.261594e+06	6.847959	6.847959	6.847959	6.847959
526	2022-04-07	4.760349e+06	2.541160e+07	4.983594e+07	3.289856e+06	6.265304e+06	6.847959	6.847959	6.847959	6.847959
527	2022-04-08	4.760977e+06	2.595160e+07	4.990271e+07	3.286698e+06	6.269013e+06	6.847959	6.847959	6.847959	6.847959

Figure 96: The result.

There are 3 features that we consider in this dataframe

- Yhat: The predicted value
- Yhat_lower: The lower bound of predicted value
- Yhat_upper The upper bound of predicted value

Step 5: Model Evaluation

```
mae = mean_absolute_error(test_set['y'], forecast['yhat'])
mape = mean_absolute_percentage_error(test_set['y'], forecast['yhat'])

print(f"MAE: {mae:.2f}")
print(f"MAPE: {mape * 100:.2f}%")
```

```
MAE: 2177625.27
MAPE: 5.16%
```

Figure 97: Predicted value on test set.

Step 6: Forecasting the future

Because we are predicting the future so we have to create the dataframe that is from the last day of training set till 8/5/2022 (30 days after real data)

```
#Create Future Dataframe
future = m.make_future_dataframe(periods = 1510, # Day until 30 days after 8/4/2022
                                 freq = "D")
future
```

	ds
0	2002-01-01
1	2002-01-02
2	2002-01-03
3	2002-01-04
4	2002-01-07
...	...
5736	2022-05-04
5737	2022-05-05
5738	2022-05-06
5739	2022-05-07
5740	2022-05-08

5741 rows × 1 columns

Figure 98: Creating future dataframe for predicting.

```
[ ] #forecast
    forecast = m.predict(future)
```

Predict data

```
[ ] #predictions
    predictions_prophet = forecast.yhat[-test_days:].rename("prophet")
    predictions_prophet[:5]
```

```
4683    3.147732e+07
4684    3.149310e+07
4685    3.148555e+07
4686    3.147113e+07
4687    3.145693e+07
Name: prophet, dtype: float64
```

Figure 99: Forecasting and print the predictions.

Step 7: Forecast visualization

Figure 100: Forecasting.

```
from fbprophet.plot import plot_plotly
plot_plotly(m,forecast)
```

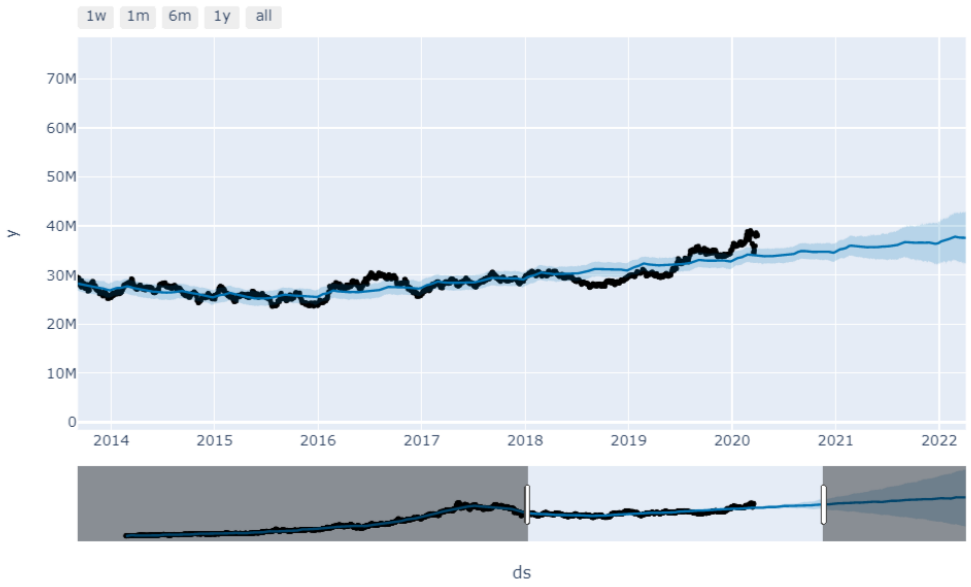


Figure 101: Forecasting in 30 days.

```
m.plot_components(forecast);
```

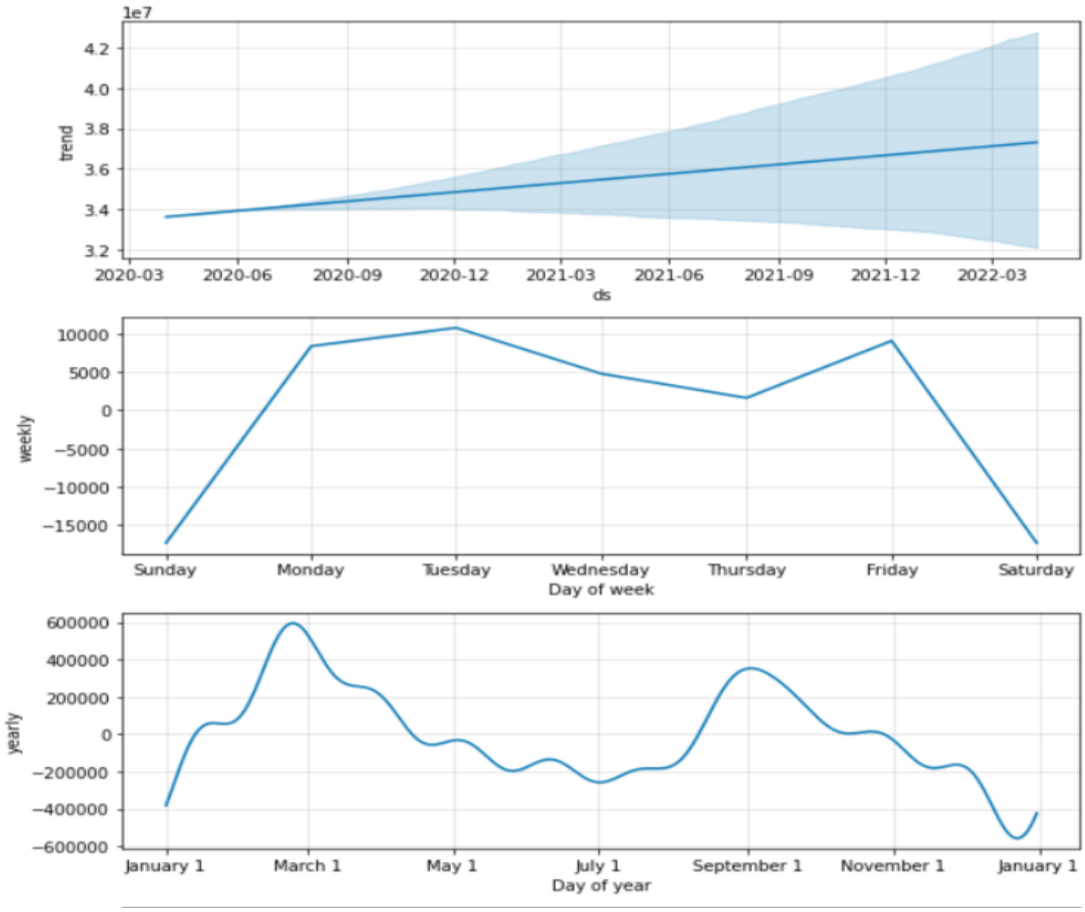


Figure 102: Trend and seasonality of year, week and day.

```
[139] #Print out predicted value of one day
      forecast[forecast.ds == '2022-04-08']['yhat']
```

```
5312    4.567282e+07
Name: yhat, dtype: float64
```

```
[140] #Print out actual value of one day
      test_set[test_set.ds=='2022-04-08']['y']
```

```
5288    44384284.57
Name: y, dtype: float64
```

```
#MAE and RMSE
from sklearn.metrics import mean_squared_error, mean_absolute_error
print(round(mean_absolute_error(test_set['y'], predictions_prophet),0))
print(round(np.sqrt(mean_squared_error(test_set['y'], predictions_prophet)), 0))
```

```
26996473.0
27025778.0
```

CHAPTER 6: LONG SHORT – TERM MEMORY (LSTM)

6.1 Model definition

6.1.1 LSTM network concept

Long Short-Term Memory (LSTM) is an improvement of RNN that allows learning of long correlations for which the recurrent neural network is constrained. means the influence of the input values at the beginning times on the output values at later times.

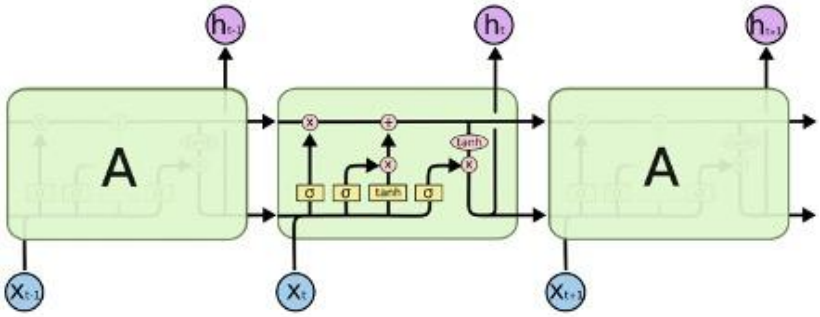


Figure 103: LSTM network.

6.1.2 Internal architecture of an LSTM cell

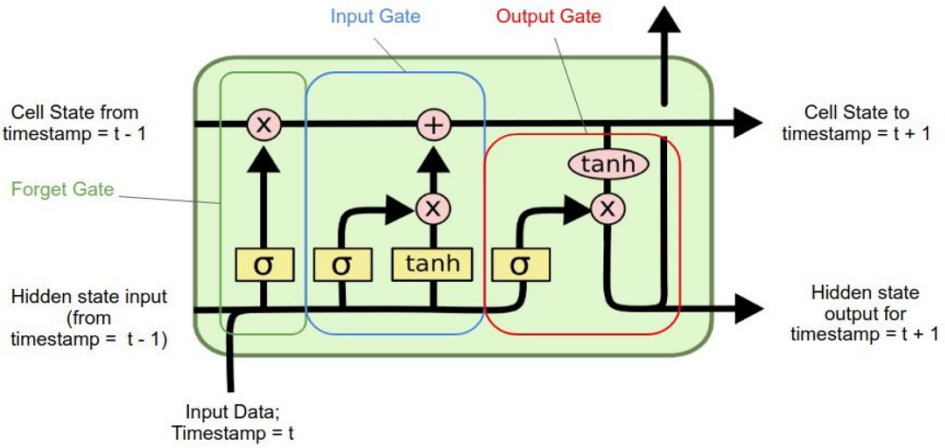


Figure 104: LSTM area.

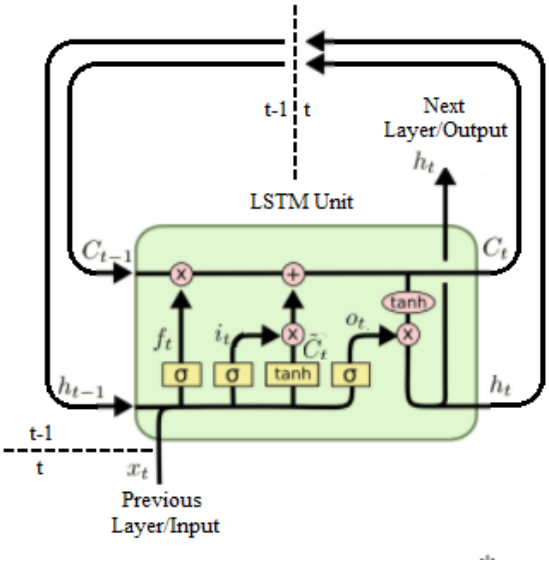


Figure 105: LSTM Cell

Where:

- C_t is cell's state
- h_t is a hidden state
- x_t is input of the cell at time t
- $C_{(t-1)}$ and $h_{(t-1)}$ is the state of the cell and the hidden state at time t-1
- f_t is forget gate
- i_t is input gate
- o_t is output gate

6.1.3 Execution of a cell LSTM

The first step is to determine what input will go to the cell states. The forget gates determine the value of the previous states, how much h(t-1) is passed. The forget gate value of (0,1) is determined by the sigmoid function [6].

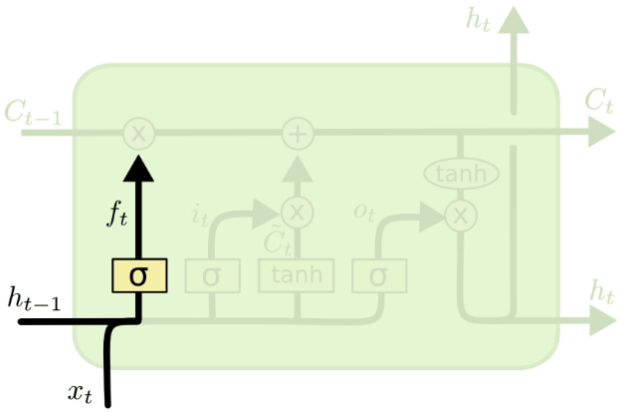


Figure 106: Forget gates.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

In the second step the input gates will determine the effect of the current input values.

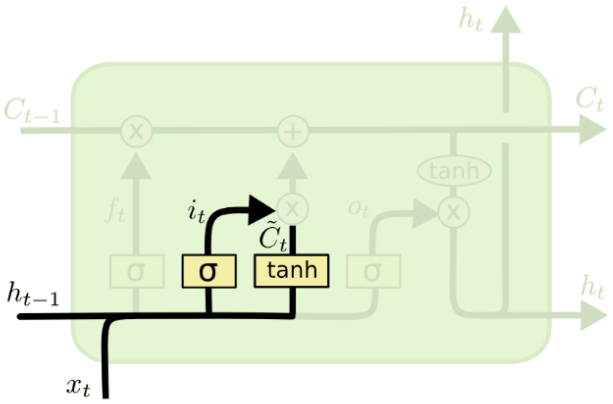


Figure 107: Input gates.

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \qquad i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Next, calculate the value for cell state c_t

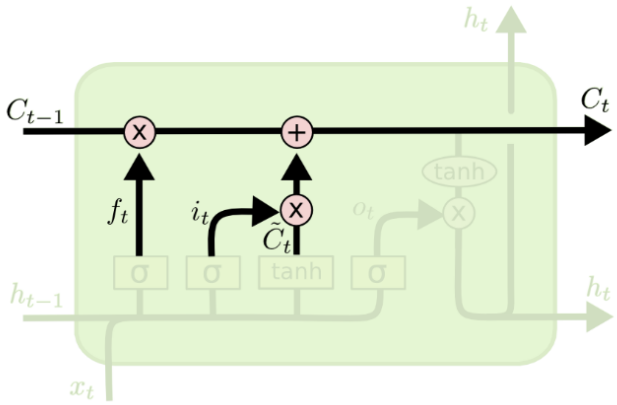


Figure 108: Calculate the value for cell state c_t.

Finally, calculate the value for the emergency layer state and the output gates.

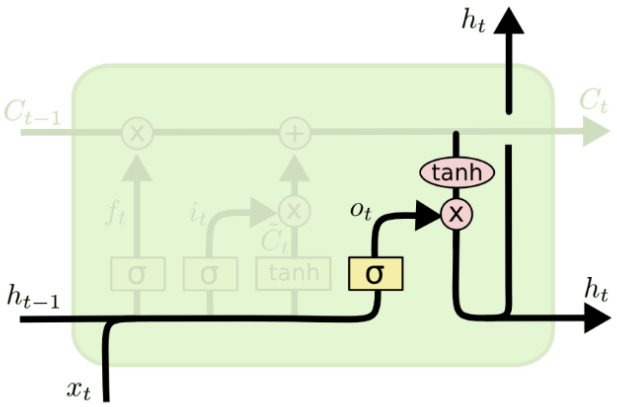


Figure 109: Output gates.

The application of LSTM that we will use to predict the future gold price is to predict the series of numbers. This is the process of predicting the next value for

a given series of numbers, this application is often used to forecast the stock market. , weather, Bitcoin price...

Below is an example of a series forecast:

6.2 Code execution with the ratio (70%-30%, 80%-20%, 90%-10%)

Step 1: Import library.

```
[ ] #Prepare and process data
    import math
    import pandas as pd #read data file csv
    import matplotlib.pyplot as plt #draw chart
    import matplotlib.ticker as ticker #Format
    import numpy as np #data processing
    from keras.callbacks import ModelCheckpoint #keep good training
    from tensorflow.keras.models import load_model #download model
    from sklearn.preprocessing import MinMaxScaler #normalize data 0->1
    #Training, model building
    from keras.models import Sequential #input
    from keras.layers import Dropout #avoid overfitting
    from keras.layers import LSTM #dependent learning
    from keras.layers import Dense #output
    #Phân tích chỉ số
    from sklearn.metrics import r2_score #measure suitability
    from sklearn.metrics import mean_squared_error #measure MSE and RMSE = sqrt(MSE)
    from sklearn. metrics import mean_absolute_error #measure mean absolute error
    from sklearn.metrics import mean_absolute_percentage_error #measure the mean percent absolute error
```

Figure 110: Library Import.

Step 2: Read data.

```
[ ] #Proceed to read csv data using pandas library with the command pd.read_csv
    df=pd.read_csv('/content/drive/MyDrive/Probability-Statistic/LAB3/Team5-PAPERREPORT/Data/DATA(2002-2022).csv')
    df
```

Figure 111: Read data using pandas library.

Checking the data we see that there are 2 columns totaling 5289 rows each, and no rows are left blank, with the column format Prices as "float" and date as '"object" for the malformed data we need. So, we're going to reformat the date for the date column

```
[ ] df.info() #check formatted data

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 5289 entries, 0 to 5288
    Data columns (total 2 columns):
     #   Column  Non-Null Count  Dtype
    ---  ------  --------------  -----
     0   Date    5289 non-null   object
     1   Prices  5289 non-null   float64
    dtypes: float64(1), object(1)
    memory usage: 82.8+ KB
```

Figure 112: Information of data.

```
[ ] #Reformat the structure of the Date column (year-month-day format)
    df["Date"]=pd.to_datetime(df.Date,format="%Y/%m/%d")
```

Figure 113: Date column data format.

Make the date column the DateFrame df's index. Each row in the DataFrame has a label called an index, which is used to specify the row's location.

```
[ ] df=pd.DataFrame(df,columns=['Date','Prices'])
    df.index=df.Date
    df.drop("Date",axis=1, inplace=True)
```

Data visualization of the gold prices from January 2002 to April 2022

```
[ ] #Draw plot
    format = ticker.StrMethodFormatter('{x:,.0f}')
    title = 'Gold Price'
    ylabel = 'Prices (VND)'
    xlabel = 'Date'
    plot = df['Prices'].plot(figsize=(20, 5), title=title)
    plot.autoscale(axis='x', tight=True)
    plot.set(xlabel=xlabel, ylabel=ylabel)
    plot.yaxis.set_major_formatter(format)
    plot.grid(True)
    plt.show()
```

Figure 114: Draw plot.

Figure 115: Gold price chart from January 2002 to April 2022.

Step 3: Split dataset.

We proceed to divide the dataset to conduct model training, here I will divide the data into 3 rates, the first rate is 70% train and 30% test, the second type is 80% train 20% test, the third type is 90% train and 10% test. For convenience in division, use the math.ceil() function to divide the data with the result rounded.

```
[ ] # divided dataste 70% train - 30% test
    training_data_len73 = math.ceil(len(data)* 0.7)
    train_data73 = data[0: training_data_len73, :]
    train_data73.shape
    test_data73=data[training_data_len73:]
    test_data73.shape

    (1586, 1)
```

Figure 116: : Validation test with 70% train and 30% test.

```
[ ] # divided dataste 80% train - 20% test
    data=df.values
    training_data_len = math.ceil(len(data)* 0.8)
    train_data = data[0: training_data_len, :]
    train_data.shape
    test_data=data[training_data_len:]
    test_data.shape

    (1057, 1)
```

Figure 117: Validation test with 80% train and 20% test.

```
[ ] # divided dataste 90% train - 10% test
    training_data_len91 = math.ceil(len(data)* 0.9)
    train_data91 = data[0: training_data_len91, :]
    train_data91.shape
    test_data91=data[training_data_len91:]
    test_data91.shape

    (528, 1)
```

Figure 118: Validation test with 90% train and 10% test.

Step 4: Data normalization.

The next step after we split the dataset is to normalize the data to 0 to 1 to conduct model training.

```
[ ] #Re-normalize the data with the values returned from 0-1
    sc=MinMaxScaler(feature_range=(0,1))
    #Normalize data named data
    sc_train=sc.fit_transform(data)
```

```
[ ] #Check data after normalization
    sc_train

    array([[0.00000000e+00],
           [6.44347526e-04],
           [8.15475581e-04],
           ...,
           [9.13878074e-01],
           [9.14767349e-01],
           [9.19516330e-01]])
```

Figure 119: Data normalization process.

Step 5: Training set data processing.

The model will use 15 consecutive days to predict the next days, in this step we proceed to create a loop for the values.

```
# 8-2
#Declare 2 data arrays x_train and y_train
x_train,y_train=[],[]
for i in range(15,len(train_data)):
  x_train.append(sc_train[i-15:i,0]) #.append assign the element to the end of the array,
  #because the number of days it will learn to predict the next 1 day is 15 so -15
  #(each array includes 6 days price)
  y_train.append(sc_train[i,0])
```

When creating a loop, we arrange the data into an array and arrange them in one dimension.

```
[ ] x_train=np.array(x_train) # sort data into an array
    y_train=np.array(y_train)
    x_train=np.reshape(x_train,(x_train.shape[0], x_train.shape[1],1)) # Process data into a one-dimensional array
    y_train=np.reshape(y_train,(y_train.shape[0],1))
```

The data with other ratios are similar

```
[ ] # 7-3
    x_train73,y_train73=[],[]
    for i in range(15,len(train_data73)):
      x_train73.append(sc_train[i-15:i,0])
      y_train73.append(sc_train[i,0])
```

```
[ ] x_train73=np.array(x_train73)
    y_train73=np.array(y_train73)

    x_train73=np.reshape(x_train73,(x_train73.shape[0], x_train73.shape[1],1))
    y_train73=np.reshape(y_train73,(y_train73.shape[0],1))
```

```
[ ] # 9-1
    x_train91,y_train91=[],[]
    for i in range(15,len(train_data91)):
      x_train91.append(sc_train[i-15:i,0])
      y_train91.append(sc_train[i,0])
```

```
[ ] x_train91=np.array(x_train91)
    y_train91=np.array(y_train91)

    x_train91=np.reshape(x_train91,(x_train91.shape[0], x_train91.shape[1],1))
    y_train91=np.reshape(y_train91,(y_train91.shape[0],1))
```

Step 6: Build model.

In this step we proceed to build a model of 9 layers, the first layer is serial() to create the network layer for input data, the next is the lstm layer whose unit is 64 and in this layer we have to describe the input information. . input and return_sequence = True to notify the next lstm layer. Next, we come to the Dropout class, which is used to assume that part of the units is hidden during training, thus reducing the mixing product. Next is the lstm class with a unit number of 64 and after that is the dropout class with an index of 0.5. The last lstm layer has a unit number of 32 followed by 2 Dense composite layers with the last 25 Dense 1 is an output layer with 1-way output to predict a value.

With the .compile function, we have loss index = 'mean_absolute_error' to measure the average error index and optimizer = 'adam' the optimizer helps the learning process of the model.

```
# 8-2
#Xây dựng mô hình
model=Sequential() #input, create a list that stores lists (list)
model.add(LSTM(units=64, input_shape=(x_train.shape[1],1) ,return_sequences=True))
# create LSTM layer - describe input information
model.add(Dropout(0.5))
model.add(LSTM(units=64, return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(units=32))
model.add(Dropout(0.5))
model.add(Dense(25))
# model.add(Dropout(0.2))
model.add(Dropout(0.5)) #Avoid memorization
#(used to assume that part of the units is hidden during training, thereby reducing the blending product)
model.add(Dense(1)) #One-way output for forecasting a value (output direction)
model.compile(loss='mean_absolute_error', optimizer='adam')
# mean_absolute_error: used to evaluate the difference between the predictive model and the testing dataset.
# adam: used to optimize the learning speed
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 15, 64)            16896

 dropout (Dropout)           (None, 15, 64)            0

 lstm_1 (LSTM)               (None, 15, 64)            33024

 dropout_1 (Dropout)         (None, 15, 64)            0

 lstm_2 (LSTM)               (None, 32)                12416

 dropout_2 (Dropout)         (None, 32)                0

 dense (Dense)               (None, 25)                825

 dropout_3 (Dropout)         (None, 25)                0

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 63,187
Trainable params: 63,187
Non-trainable params: 0
_________________________________________________________________
```

Figure 120: Description of the number of Layers,Ooutput Shape and Param in the model.

Step 7: Model training.

In the 7th step, we train the model with the epochs index of 100. The model after training will be saved in the save_model.h5 file, above all, only the best model is saved when training.

```
# model model training
save_model="save_model.h5" #save the model as file .h5
best_model=ModelCheckpoint(save_model,monitor='loss',verbose=2,save_best_only=True, mode='auto')
# find the best model, save_best_only: save the best model
model.fit(x_train,y_train, epochs=100,verbose=2,callbacks=best_model)
```

Step 8: Data processing of test sets and train sets.

Re-upload the saved model and apply it to the x_train prediction and format the data back to the original format.

```
[ ] y_train=sc.inverse_transform(y_train) #Proceed to process the normalized number 0-1 into the price of gold VND
    final_model=load_model('save_model.h5') #reload the best model saved
    y_train_predict=final_model.predict(x_train) #predicted price
    y_train_predict=sc.inverse_transform(y_train_predict)
    #proceed to process the normalized number 0-1 into the predicted gold price VND
```

In the test set, we process the data as in the training set

```
[ ] test=df[len(train_data)-15:].values
    test=test.reshape(-1,1)
    sc_test=sc.transform(test)

    x_test=[]
    for i in range(15,test.shape[0]):
      x_test.append(sc_test[i-15:i,0])
    x_test=np.array(x_test)
    x_test=np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))

    y_test=data[training_data_len:]
    y_test_predict=final_model.predict(x_test)
    y_test_predict=sc.inverse_transform(y_test_predict)
```

Step 9: Data histogram between the set of train – test and predict

In the 9th step, we proceed to draw a graph to be able to observe the train - test data with the predicted data.

```
train_data1=df[15:training_data_len]
test_data1=df[training_data_len:]
format = ticker.StrMethodFormatter('{x:,.0f}')
title = 'Gold Price'
ylabel = 'Prices (VND)'
xlabel = 'Date'
plt.figure(figsize=(21,5))
plot = df['Prices'].plot(label=f"Prices Data",color='red')
train_data1['prediction']=y_train_predict
plot_pdtrain = train_data1['prediction'].plot(label=f"Prediction train", color='green')
test_data1['prediction']=y_test_predict
plot_pdtest = test_data1['prediction'].plot(label=f"Prediction test", color='blue')

plot.autoscale(axis='x', tight=True)
plot.set(xlabel=xlabel, ylabel=ylabel)
plot.yaxis.set_major_formatter(format)
plt.legend()
plt.grid(True)
plt.show()
```

After plotting we see the model works quite well, but to be more precise we proceed with the next step which is data analysis step.

Figure 121: Train 70% - Test 30%.

Figure 122: Train 80% - Test 20%.

Figure 123: Train 90% - Test 10%

Step 10: Evaluation

At first data rate of 70% train and 30% test, MAPE reuse is at 4.83% and RMSE is 2.149.828 VND with data after analysis of data set 7-3 is good.

Date	Prices	prediction
2016-03-11	28187483.25	26967356.0
2016-03-14	27707111.25	26996702.0
2016-03-15	27457584.00	26985362.0
2016-03-16	27389407.50	26921162.0
2016-03-17	28220786.25	26827656.0
...	...	...
2022-04-04	44102530.37	39973836.0
2022-04-05	44470144.87	39958364.0
2022-04-06	44137702.89	39970640.0
2022-04-07	44176594.17	39980868.0
2022-04-08	44384284.57	39977812.0

```
[ ] #7-3 test data
    mape_t73 = mean_absolute_percentage_error(test_data173['Prices'],test_data173['prediction'])
    print(f"MAPE: {mape_t73 * 100:.2f}%")
    print('RMSE: ',np.sqrt(mean_squared_error(test_data173['Prices'],test_data173['prediction'])))

    MAPE: 5.77%
    RMSE:  2452811.3028311552
```

At the second data rate of 80% train and 20% test, the MAPE reuse is 5.77% and RMSE is 2,452,811 VND with the data after analysis of data set 8-2 as good.

Date	Prices	prediction
2018-03-22	30297972.02	29167086.0
2018-03-23	30706516.46	29203758.0
2018-03-26	30849594.17	29300968.0
2018-03-27	30605851.36	29439882.0
2018-03-28	30405175.44	29556280.0
...	...	...
2022-04-04	44102530.37	41033744.0
2022-04-05	44470144.87	41015396.0
2022-04-06	44137702.89	41039464.0
2022-04-07	44176594.17	41057120.0
2022-04-08	44384284.57	41059368.0

```
[ ] #8-2 test data
    mape_t82 = mean_absolute_percentage_error(test_data1['Prices'],test_data1['prediction'])
    print(f"MAPE: {mape_t82 * 100:.2f}%")
    print('RMSE: ',np.sqrt(mean_squared_error(test_data1['Prices'],test_data1['prediction'])))

    MAPE: 4.83%
    RMSE:  2149828.17438557
```

At the final data rate of 90% train and 10% test, MAPE reuse at 5.52% and RMSE is 2,438,809 VND with data after analysis of dataset 9-1 as good.

Date	Prices	prediction
2020-04-01	37198698.40	36631136.0
2020-04-02	38148397.15	36504680.0
2020-04-03	38012703.80	36454420.0
2020-04-06	38652636.15	36477776.0
2020-04-07	38687281.88	36621532.0
...	...	...
2022-04-04	44102530.37	41288784.0
2022-04-05	44470144.87	41279376.0
2022-04-06	44137702.89	41320560.0
2022-04-07	44176594.17	41339600.0
2022-04-08	44384284.57	41334248.0

```
[ ] #9-1 test data
    # print('R_square train: ', r2_score(test_data191['Prices'],test_data191['prediction']))
    # print('MAE: ',mean_absolute_error(test_data191['Prices'],test_data191['prediction']))
    mape91 = mean_absolute_percentage_error(test_data191['Prices'],test_data191['prediction'])
    print(f"MAPE: {mape91 * 100:.2f}%")
    # print('MSE: ',mean_squared_error(test_data191['Prices'],test_data191['prediction']))
    print('RMSE: ',np.sqrt(mean_squared_error(test_data191['Prices'],test_data191['prediction'])))

    MAPE: 5.52%
    RMSE:  2438809.0023021423
```

The fact that the ratio between the trains - tests is not different, as well as the MAPE at ~5%, we can see that the LSTM model works very well, after many times of training the model, we find that the model runs at the ratio. 70% train 30% test rate is giving the best results so in the next part we will proceed to predict the next 15 days of future gold prices.

6.3 Pridiction gold prices next 15 days.

```
# demonstrate prediction for next 15 days
from numpy import array

lst_output=[]
n_steps=15
i=0
while(i<15):
    if(len(temp_input)>15):
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i+1,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        yhat = model.predict(x_input, verbose=1)
        print("{} day output {}".format(i+1,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=2)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1
```

Figure 124: Prediction for the next 15 days.

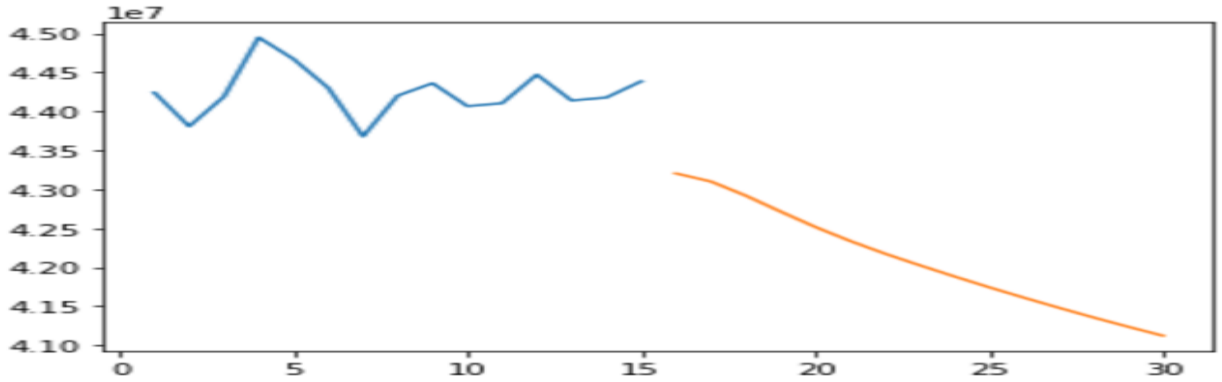


Figure 125: Gold price chart next 15 days orange line.

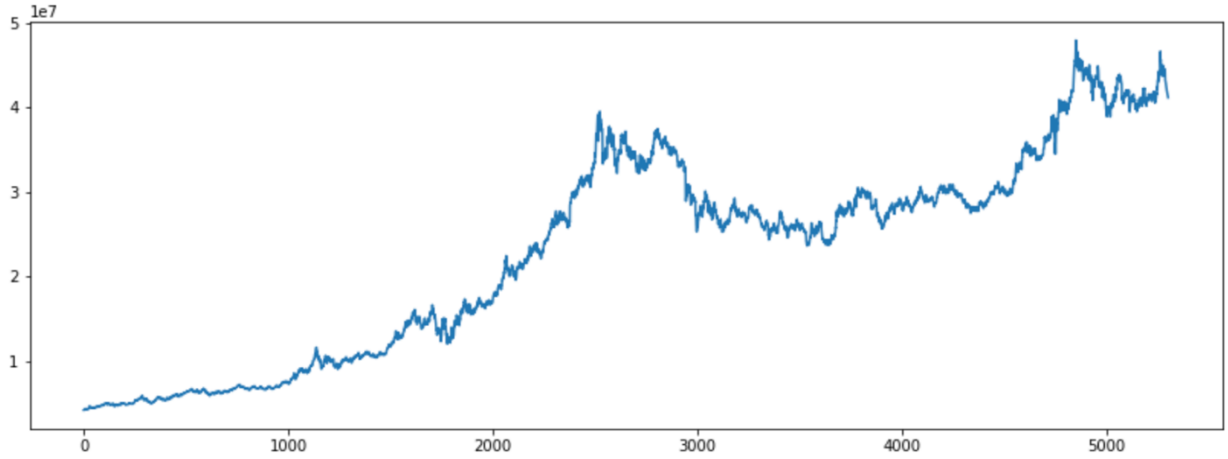


Figure 126: The predicted price data chart has been appended to the data set.

```
[70] lst_output #pridictions gofld prices

array([[43203146.61382373],
       [43100297.61558833],
       [42920158.55650423],
       [42715862.61965919],
       [42518011.28036357],
       [42338765.88189314],
       [42176398.84295245],
       [42026914.90319921],
       [41883197.69564589],
       [41744366.97068397],
       [41611062.79901217],
       [41482291.54706962],
       [41357205.88316942],
       [41237849.15681837],
       [41122655.93796472]])
```

Figure 127: Details of gold price of the next 15 days.

Thus, it is predicted that in the next 15 days the gold price will decrease

CHAPTER 7: CONCLUSION

1. Result

Model	Train-test	RMSE	MAPE (%)
Linear Regression	**10-0**	4777274.13	16.65
Non-Linear Regression	**10-0**	4545507.81	21.78
ARIMA	**7-3**	8727545.1	16.27
	8-2	8984086.99	18.05
	9-1	4149475.21	8.73
FBProphet	**7-3**	14211069.53	38.73
	8-2	5900284.21	12.19
	9-1	2509956.14	5.16
LSTM	**7-3**	2452811.30	5.77
	8-2	2149828.17	4.83
	9-1	2438809	5.52

From the outcomes, one can without much of a stretch see that LSTM has better execution compared to different models for all evaluation metrics. LSTM has performed well in all train-test set. The RMSE values are lower than ARIMA and FBProphet. The best MAPE score is 5.83% which is for LSTM by the data at the rate 8-2 train test. Best RMSE is 2149828.17 for 8-2 train-test split by LSTM. To assess overall model performance. The deep learning-based model LSTM outperform ARIMA, and FBProphet in term of mean absolute percentage error. At the same time LSTM has the lowest error in root mean squared error metrics compared with other models.

2. Future Work

This study used ARIMA, FBProphet, LSTM, LR, and NLR models to analyse and predict the price of gold. In terms of MAPE and RMSE assessment metrics, LSTM outperforms all other models for most of the train test set. The trend analysis demonstrates that ARIMA and FBProphet have significant MAPE errors because they were unable to accurately anticipate lower values. The techniques used produced positive outcomes. However, it limits the scope of our analysis to the model's suitability, which may also be enhanced by spending more time examining hyper-optimization strategies.

CHAPTER 8: DUTY ROSTER

Member	Name	Student ID	Works
1	Nguyễn Nhất Thưởng	20522000	LSTM
2	Lê Quang Hoà	20521331	NLR, FBProphet
3	Kiều Xuân Diệu Hương	20521381	LR, ARIMA

References

[1] What Is Nonlinear Regression? Comparison to Linear Regression

[2] What is linear regression?

[3] ARIMA Model – Complete Guide to Time Series Forecasting in Python

[4] ARIMA Model In Time Series

[5] Time Series Analysis using Facebook Prophet

[6] Understanding LSTM Networks

[7] Explain Prophet

[8] Facebook Prophet Tutorial: How to Use Time Series Forecasting_