VIETNAM NATIONAL UNIVERSITY

UNIVERSITY OF INFORMATION TECHNOLOGY

INFORMATION SYSTEMS FACULTY

REPORT

SUBJECT: Gold prices forecasting: A comparison of various forecasting models

LECTURER: Assoc. Prof. Dr. Nguyễn Đình Thuân

TA: Nguyễn Minh Nhựt

CLASS: STAT3013.N11.CTTT

Phạm Thành Đạt – 20521175

Thiều Huy Hoàng - 20521350

Nguyễn Quang Vy -20522181

INDEX

ACKNOWLEDGMENT

*Dear **Assoc. Prof. Dr. Nguyễn Đình Thuân** and **Mr. Nguyễn Minh Nhựt, teaching assistants** at the University of Information Technology!*

First and foremost, I would like to express my deepest gratitude to ***Assoc. Prof. Dr. Nguyễn Đình Thuân*** and ***Mr. Nguyễn Minh Nhựt*** who always cared for our team during this course. I appreciate your assistance and the sharing of your valuable experience, which will help us accomplish our project. I would sincerely like to thank ***Mr. Nhựt***, who teaches and supports our team throughout our course.

I would like to take this opportunity to thank ***Assoc. Prof. Dr. Nguyễn Đình Thuân*** for permitting us to carry out our project. Finally, I would like to express how honored when our team was able to learn from and attend your class.

Many thanks to ***Assoc. Prof. Dr. Nguyễn Đình Thuân*** and ***Mr. Nguyễn Minh Nhựt*** for your tireless efforts in guiding the team to success and encouraging our team to keep moving forward. My heartfelt gratitude goes to all of my classmates, especially my friends, for devoting their time to assisting and supporting our team in the fabrication of our project.

Hồ Chí Minh City, 31 December 2022

Yours Sincerely,

Team 6

Gold prices forecasting:
A comparison of various forecasting models

I. Literature review

In any unstable political and economic situation or crisis, gold is one of the most effective financial currencies for maintaining price and avoiding risk. From ancient times until now, gold has been an important currency for the Vietnamese people in general, so every year the Vietnamese people have a god of fortune, and on that day, everyone buys gold with the hope of luck. Most well-to-do families in Vietnam own some gold, and they are particularly interested in fluctuations in gold prices.

As a result, in this project, our group has chosen to predict the gold price using machine learning and deep learning models in the hope of assisting the people of Vietnam.

II. Dataset

The data set was gathered on Kaggle Web between January 2, 2012, and December 30, 2022 **[1]**. It has 2870 rows and 2 attribute columns in total. One column describes the timeline of data representation, while the others show the gold price in VND over time.

Date	US dollar (	Euro (EUR	Japanese y	Canadian (	Chinese re	Indonesiar	Thai baht 1	Vietnames	Korean wc	Russian ru	South Afric	Australian
1/2/2012	1531	1179.37	117795.1	1558.94	9636.11	13882343	48303.03	32202289	1763712	49180.29	12360.37	1493.37
1/3/2012	1598	1224.24	122646.5	1612.54	10057.81	14597730	50416.88	33607538	1838899	50657.38	12838.89	1539.57
1/4/2012	1613	1249.52	123781.6	1636.31	10153.19	14750885	50753.04	33923003	1853014	51401.13	13175.06	1561.25
1/5/2012	1599	1249.9	123298.9	1632.98	10076.42	14590875	50616.34	33628569	1843167	51220.92	13082.94	1560.61
1/6/2012	1616.5	1271.43	124656.4	1655.7	10199.31	14702068	51121.81	34000653	1879747	51664.95	13167.44	1580.93
1/9/2012	1615	1267.91	124112.7	1657.88	10198.08	14777250	51292.39	33970718	1879295	51549.66	13180.01	1579.39
1/10/2012	1637	1281.11	125713.4	1667.53	10337.65	14986735	51835.59	34406466	1893518	51672.71	13286.38	1584.78
1/11/2012	1634.5	1288.38	125725.7	1668.74	10322.68	14972020	51895.38	34374352	1893977	51926.9	13289.14	1589.29
1/12/2012	1661	1297.96	127490	1695.22	10493.87	15214760	52869.61	34939135	1923604	52517.82	13411.33	1611.06
1/13/2012	1635.5	1291.2	125900.8	1675.81	10314.44	14850340	52017.07	34400289	1877881	52203.76	13339.46	1590.26
1/16/2012	1641	1294.93	125881.1	1670.78	10365.38	14998740	52339.68	34490538	1895026	51984.89	13258.38	1588.58
1/17/2012	1656	1300.15	127197.4	1679.51	10457.64	15044760	52619.39	34805808	1897030	52378.1	13358.12	1593
1/18/2012	1647	1285.06	126489.6	1669.81	10395.86	14901233	52308.71	34471710	1880462	51908.17	13203.92	1585.87
1/19/2012	1655	1283.69	127550.8	1669.65	10454.14	14895000	52422.1	34589500	1881900	51863.55	13131.43	1588.83

Figure 1. Dataset

III. Related Work

The list of paper which have been review based on problems under consideration, the problems, algorithms, evaluation.

Table 1. Table on Related Works

SI No	Year	Title	Author(s)	Dataset	Problems	Purpose	Algorithm	Evaluation
1	2019	Gold and Diamond Price Prediction Using Enhanced Ensemble Learning	(Pandey et al., 2019) [1]	Previous data of the product.	Variation in price of gold market.	To analyze and examine the patterns of previous close prices.	linear regression on and random forest	Mean, Best and worst is calculated for preciseness.
2	2020	Gold Price Prediction and Modelling using Deep Learning Techniques	(Vidya and Hari, 2020) [2]	Data taken from World Gold Council for year 1987 to44013	Gold pricing nonlinearity.	To forecast gold price using LSTM	Long Shortterm Memory Network s (LSTM)	Root mean square error (RMSE)
3	2017	Forecasting Gold Price with Auto Regressive Integrated Moving Average Model	(Tripathy, 2017) [3]	Price of gold from July 1990 to February 2015	Forecasting models forecasting is inaccurate	To gain more accuracy using ARIMA (Auto regressive Integrated	Box-Jenkins' ARIMA (Auto regressive Integrated Moving Average)	Provides good results for the error measures used.

						Moving Average)		
4	2015	Gold Price Prediction Using Type-2 Neuro-Fuzzy Modeling and ARIMA	(Modeling et al., 2015)	Gold Price historical data	When time factor is included in dataset there will uncertainty in results which might occur in future	To predict accuracy in predicting price of gold	type-2 neurofuzzy modeling and ARIMA (Auto regressive Integrated Moving Average)	Root mean square error (RMSE)Mean Absolute Percentage error (MAPE)Mean Absolute Error (MAE)
5	2016	Gold Price Forecasting Using ARIMA Model	Banhi Guha and Gautam Bandyopadhyay [4]	secondary monthly data for Gold price, collected from Multi Commodity Exchange of India Ltd (MCX) ranging from November 2003 to January 2014.	Forecasting models forecasting is inaccurate	To forecast the price of Gold using time-series ARIMA Model.	ARIMA (Auto regressive Integrated Moving Average)	Provides good results for the error measures used.
6	2018	The Prediction of Gold Price Using ARIMA Model	Xiaohui Yang [5]	The data are collected from the World Gold Council, consisting of 1305 observations of daily gold price from July 1st 2013 to June 29 2018.	Variation in price of gold market.	investigate and carry out the prediction of the future international gold price	ARIMA (Auto regressive Integrated Moving Average)	Provides the most accurate and appropriate model for forecasting
7	2019	Gold Price Forecast based on LSTM-	Zhanhong He , Junhao Zhou , Hong-Ning Dai , Hao Wang[6]	World Gold Council [2](WGC) contains	Forecasting models forecasting is inaccurate	predict the tendency of daily gold price.	LSTM and CNN neural networks	Root mean square error (RMSE)

		CNN Model		10471 daily gold price transaction record from Dec. 29, 1978 to Feb. 15, 2019 (only on trading day)			with Attention Mechanism	Mean Absolute Percentage error (MAPE) Root mean absolute error (RMAE)

IV. Method

Although there are numerous time-series forecasting models available, this project presents an empirical evaluation of ***seven popular time-series forecasting models*** for ***forecasting gold prices***. In particular, six forecasting models are:

1. **Autoregressive integrated moving average (ARIMA)**
2. **Prophet**
3. **LSTM CNN model**
4. **Bidirectional LSTM**
5. **Linear regression.**
6. **Support Vector Regression**

Our project will be implementing multiple forecasting models and comparing their performance using error measures namely MAPE and RMSE for each model to determine which one is the most optimal for estimating the price.

❖ Performance measure

To assess the predictive power of our proposed models, we use two performance measures: the root means square error (RMSE) and the MAPE. When we train models, we use RMSE as a loss function, and MAPE is a statistical measure of prediction accuracy. The following are the equations:

$$RMSE = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\left(x_{1,i} - x_{2,i}\right)^2} \quad [2]$$

$$MAPE = \frac{100}{M}\sum_{i=1}^{M}\left|\frac{x_{2,i} - x_{1,i}}{x_{1,i}}\right| \quad [3]$$

Where M is the number of data points, $x_{1,i}$ is a predicted value and $x_{2,i}$ is a real value.

A. AUTOREGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA)

Definition

An autoregressive integrated moving average, or ARIMA, is a statistical analysis model that uses time series data to either better understand the data set or to predict future trends.

A statistical model is autoregressive if it predicts future values based on past values. For example, an ARIMA model might seek to predict a gold future prices based on its past performance or forecast a company's earnings based on past periods. **[4]**

The ARIMA model stands for Auto Regression (AR), Moving Average (MA) and Differential Integration Integrated - I.

Important point: The ARIMA model is not a perfect predictive model for any time series data.

The ARIMA model only works best if the data is highly time dependent. Randomized data usually do not work for ARIMA models.

The ARIMA model is only good at predicting time points.

- **Types of Models ARIMA:**

The ARIMA model is not seasonal
Seasonal ARIMA model (Seasonal ARIMA – SARIMA)

- **Stationary**

A stationary time series is a series of mean, variance, and autocorrelation values that do not change over time and it does not include the trend factor. With most statistical predictive methods, the calculation must be ensured. stationarity of the data series, so checking for stationarity is very important. To test the stationarity of data, we have two popular testing methods: Dickey (DF) test and Improved Dickey Fuller (ADF4). **[5]**

- **ARIMA (*p, d, q*)**

The parameter ***p*** is the number of autoregressive terms or the number of "lag observations." It is also called the "lag order," and it determines the outcome of the model by providing lagged data points.

The parameter ***d*** is known as the degree of differencing. it indicates the number of times the lagged indicators have been subtracted to make the data stationary.

The parameter ***q*** is the number of forecast errors in the model and is also referred to as the size of the moving average window.

The parameters take the value of integers and must be defined for the model to work. They can also take a value of 0, implying that they will not be used in the model. In such a way, the ARIMA model can be turned into:

ARMA model (no stationary data, ***d* = 0**)

AR model (no moving averages or stationary data, just an autoregression on past values, ***d* = 0, *q* = 0**)

MA model (a moving average model with no autoregression or stationary data, ***p* = 0, *d* = 0)**

Therefore, ARIMA models may be defined as:

1. ARIMA(1, 0, 0) – known as the **first-order autoregressive model**
2. ARIMA(0, 1, 0) – known as the **random walk model**
3. ARIMA(1, 1, 0) – known as the **differenced first-order autoregressive model**, and so on.

Once the parameters (p, d, q) have been defined, the ARIMA model aims to estimate the coefficients $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$, which is the result of using previous data points to forecast values. **[6]**

Implementation in Python language

1. Import libraries

```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib as plt
# Load specific forecasting tools
from statsmodels.tsa.arima_model import ARIMA,ARIMAResults
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # for determining (p,q) orders
from pmdarima import auto_arima # for determining ARIMA orders
import matplotlib.pyplot as plt
# Ignore harmless warnings
import warnings
warnings.filterwarnings("ignore")


import matplotlib.ticker as ticker
formatter = ticker.StrMethodFormatter('{x:,.0f}')
from matplotlib import pyplot
import math
from datetime import datetime
```

Figure 2. Import libraries

2. Import data and get Date column as Index

```
df = pd.read_csv("C:/Users/thieu/Downloads/Data-Gold.csv",index_col='Date',parse_dates=True)
df.tail(30)
```

Date	VND
2022-11-21	39976008.0
2022-11-22	39982902.0
2022-11-23	40113888.0
2022-11-24	40337943.0
2022-11-25	40306920.0
2022-11-28	40336794.0
2022-11-29	40529826.0
2022-11-30	40442502.0
2022-12-01	41713296.0
2022-12-02	41584608.0

Figure 3. Import data

3. Data visualization

```
title = 'Real Gold Price'
ylabel='VND'
xlabel='' # we don't really need a label here

ax = df['VND'].plot(figsize=(12,5),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```

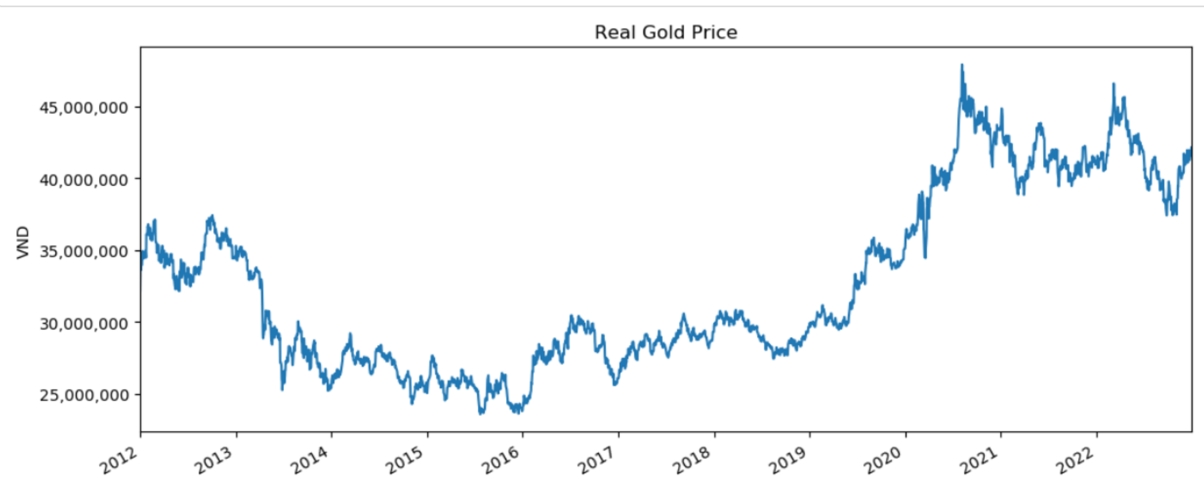

Figure 4. Visualize data

4. Check if data is stationary series using adf test

```python
from statsmodels.tsa.stattools import adfuller

def adf_test(series,title=''):
    """
    Pass in a time series and an optional title, returns an ADF report
    """
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles differenced data

    labels = ['ADF test statistic','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())          # .to_string() removes the line "dtype: float64"

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is non-stationary")
```

Figure 5. Code ADF test

```
: adf_test(df['VND'])

Augmented Dickey-Fuller Test:
ADF test statistic        -0.681541
p-value                    0.851473
# lags used               22.000000
# observations          2847.000000
critical value (1%)       -3.432649
critical value (5%)       -2.862556
critical value (10%)      -2.567311
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary
```

Figure 6. Test the stationary of the original data

Since the original series is not stationary, we take the first difference (d = 1) of the series to test for stationarity

```
from statsmodels.tsa.statespace.tools import diff
df['d1'] = diff(df['VND'],k_diff=1)

# Equivalent to:
# df1['d1'] = df1['Inventories'] - df1['Inventories'].shift(1)

adf_test(df['d1'],'Real Gold Price')
```

```
Augmented Dickey-Fuller Test: Real Gold Price
ADF test statistic     -1.256895e+01
p-value                 2.024078e-23
# lags used             2.100000e+01
# observations          2.847000e+03
critical value (1%)    -3.432649e+00
critical value (5%)    -2.862556e+00
critical value (10%)   -2.567311e+00
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary
```

Figure 7. Test stationary of the first difference (d=1)

```
df
```

Date	VND	d1
2012-01-02	32202288.50	NaN
2012-01-03	33607538.00	1405249.50
2012-01-04	33923003.00	315465.00
2012-01-05	33628569.00	-294434.00
2012-01-06	34000652.75	372083.75
2012-01-09	33970717.50	-29935.25
2012-01-10	34406466.00	435748.50
2012-01-11	34374352.25	-32113.75
2012-01-12	34939135.00	564782.75
2012-01-13	34400289.25	-538845.75
2012-01-16	34490538.00	90248.75
2012-01-17	34805808.00	315270.00

Figure 8. Data after add d1

5. Split data

We will split the dataset into train and test sets, we take 90% train and 10% test

```
from sklearn.model_selection import train_test_split

# Splitting the dataset into 90% training data and 10% testing data.
train, test = train_test_split(df, test_size=.10, random_state=0,shuffle=False)
```

Figure 9. Split data

6. Find coefficients p,q,d

Build ARIMA model, find coefficients p,q,d using auto_arima . function

```
auto_arima(train['VND'],seasonal=False).summary()
```

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	2582
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-36338.203
Method:	css-mle	**S.D. of innovations**	313239.885
Date:	Mon, 02 Jan 2023	**AIC**	72684.405
Time:	08:16:59	**BIC**	72707.830
Sample:	1	**HQIC**	72692.896

	coef	std err	z	P>|z|	[0.025	0.975]
const	3188.3916	5162.252	0.618	0.537	-6929.437	1.33e+04
ar.L1.D.y	0.8876	0.082	10.856	0.000	0.727	1.048
ma.L1.D.y	-0.9059	0.075	-12.068	0.000	-1.053	-0.759

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.1267	+0.0000j	1.1267	0.0000
MA.1	1.1038	+0.0000j	1.1038	0.0000

Figure 10. Result of auto_arima function

After using auto arima, we find 3 coefficients p, d, q respectively 1,1,1

```
model = ARIMA(train['VND'],order=(1,1,1))
fitted = model.fit()
fitted.summary()
```

ARIMA Model Results

Dep. Variable:	D.VND	**No. Observations:**	2582
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-36338.203
Method:	css-mle	**S.D. of innovations**	313239.885
Date:	Mon, 02 Jan 2023	**AIC**	72684.405
Time:	08:16:59	**BIC**	72707.830
Sample:	01-03-2012	**HQIC**	72692.896
	- 11-24-2021		

	coef	std err	z	P>|z|	[0.025	0.975]
const	3188.3916	5162.252	0.618	0.537	-6929.437	1.33e+04
ar.L1.D.VND	0.8876	0.082	10.856	0.000	0.727	1.048
ma.L1.D.VND	-0.9059	0.075	-12.068	0.000	-1.053	-0.759

Figure 11. Use ARIMA to train model

7. Find predictive data and plot model

Find predictive data based on train data and compare with test set

```
# Obtain predicted values
start = len(train)
end  = len(train)+len(test)-1
pred = fitted.predict(start=start, end=end, dynamic=False, typ='levels')
```

Figure 12. Create predict data

```
# Compare predictions to expected values
for i in range(len(pred)):
    print(f"predicted={pred[i]}, expected={test['VND'][i]}")
```

```
predicted=40456017.093508005, expected=40546299.04
predicted=40475282.397533, expected=40840344.31
predicted=40492740.4187707, expected=40523204.39
predicted=40508594.322542086, expected=40948149.18
predicted=40523024.43538057, expected=40642813.75
predicted=40536190.81244958, expected=40127275.0
predicted=40548235.51634444, expected=40370843.12
predicted=40559284.63972318, expected=40953414.0
predicted=40569450.100563735, expected=41104653.5
predicted=40578831.23560768, expected=40902535.12
predicted=40587516.214677334, expected=40833686.25
predicted=40595583.29600265, expected=40889756.25
predicted=40603101.94043084, expected=41083645.12
predicted=40610133.80038234, expected=40886466.75
predicted=40616733.59763347, expected=40740850.5
predicted=40622949.902423404, expected=41287631.13
```

Figure 13. Compare predict data to test set

Plot predictions against known values

```
# Plot predictions against known values
title = 'Real Gold Price'
ylabel='Actual'
xlabel='Predict' # we don't really need a label here
pd=train['VND'].plot(legend=True,label='train')
ax = test['VND'].plot(legend=True,figsize=(12,6),title=title,label='test')
pred.plot(legend=True,label='predict')
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```

Figure 14. Model of train, test and predict data

8. Calculate RMSE

```
MSE = np.square(np.subtract(test['VND'],pred)).mean()

rsme = math.sqrt(MSE)
print("Root Mean Square Error:\n")
print(rsme)
```

```
Root Mean Square Error:

2074521.9026960835
```

Figure 15. Calculate RMSE

9. Calculate MAPE

```
def mape(y_test, pred):
    y_test, pred = np.array(test['VND']), np.array(pred)
    mape = np.mean(np.abs((test['VND'] - pred) / test['VND']*100))
    return mape
print(mape(test['VND'],pred))
```

```
3.9400921075393693
```

Figure 16. Calculate MAPE

10.Prediction for the next 30 days

```
model = ARIMA(df['VND'],order=(1,1,1))
fittin = model.fit()
fittin.summary()
forecast = fittin.predict(len(df),len(df)+30,typ='levels').rename('ARIMA(1,1,1) Forecast')
print(forecast)
```

```
2870    4.212931e+07
2871    4.212177e+07
2872    4.211562e+07
2873    4.211068e+07
2874    4.210680e+07
2875    4.210384e+07
2876    4.210169e+07
2877    4.210025e+07
2878    4.209942e+07
2879    4.209913e+07
2880    4.209932e+07
```

Figure 17. Predict the next 30 days

11.Plot predictions against known values

```
# Plot predictions against known values
title = 'Real Gold Price'
ylabel='VND'
xlabel='' # we don't really need a label here

ax = df['VND'].plot(legend=True,figsize=(12,6),title=title)
fcast.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```

Figure 18. Model after predict

Predicted data for the next 30 days

```
2870    4.212931e+07
2871    4.212177e+07
2872    4.211562e+07
2873    4.211068e+07
2874    4.210680e+07
2875    4.210384e+07
2876    4.210169e+07
2877    4.210025e+07
2878    4.209942e+07
2879    4.209913e+07
2880    4.209932e+07
2881    4.209992e+07
2882    4.210088e+07
2883    4.210215e+07
2884    4.210370e+07
2885    4.210549e+07
2886    4.210749e+07
2887    4.210967e+07
2888    4.211202e+07
2889    4.211451e+07
2890    4.211712e+07
2891    4.211983e+07
2892    4.212264e+07
2893    4.212554e+07
2894    4.212850e+07
2895    4.213153e+07
2896    4.213461e+07
2897    4.213774e+07
2898    4.214091e+07
2899    4.214412e+07
2900    4.214736e+07
Name: ARIMA(1,1,1) Forecast, dtype: float64
```

Figure 19. Result of the predict 30 days

We perform ARIMA model on 3 cases:

Table 2. Measuring the ARIMA model according to split data

Model	Train-Test	RMSE	MAPE
ARIMA	7-3	5578826.49	11.85%
	8-2	4697358.81	10.32%
	9-1	2074521.9	3.94%

After measuring the ARIMA model, the model with 90% data for training and 10% data for testing is the most optimal in all three cases. We see that the results are very good, with very low RMSE (2074521.9) and MAPE (3.94%), because there has been no strong fluctuation or sudden change over the years due to the characteristics of the gold data set.

B. PROPHET

Prophet is a free, open-source application developed by Facebook for forecasting time series data, which aids in understanding and potential market forecasts for organizations. It is based on a decomposable additive model, which also accounts for the effects of vacations, and fits non-linear trends with seasonality. [7]

Trend:

The trend shows the tendency of the data to increase or decrease over a long period of time and it filters out the seasonal variations.

Seasonality:

Seasonality is the variations that occur over a short period of time and is not prominent enough to be called a “trend”.

Understanding the Prophet Model

The general idea of the model is similar to a generalized additive model. The “Prophet Equation” fits, as mentioned above, trend, seasonality and holidays. This is given by,

$$\mathbf{y(t) = g(t) + s(t) + h(t) + e(t)}$$

where:

g(t) refers to trend (changes over a long period of time)

s(t) refers to seasonality (periodic or short term changes)

h(t) refers to effects of holidays to the forecast

e(t) refers to the unconditional changes that is specific to a business or a person or a circumstance. It is also called the error term.

y(t) is the forecast.

Implementation in Python language

1. Import libraries:

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

from prophet.plot import add_changepoints_to_plot
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error
```

```
import warnings
warnings.simplefilter('ignore')
```

Figure 20. Import libraries

2. Import data

```
#get the data
df = pd.read_csv('C:/Users/thieu/Downloads/Data-Gold.csv',parse_dates=True, index_col=0)
df.tail(5)
```

	VND
Date	
2022-12-26	41586906.0
2022-12-27	41894838.0
2022-12-28	41727084.0
2022-12-29	41961480.0
2022-12-30	42138426.0

Figure 21. Import data

3. Split data

We will split the dataset into train and test sets, we take 70% train and 30% test

```
from sklearn.model_selection import train_test_split
# Splitting the dataset into 70% training data and 30% testing data.
train, test = train_test_split(df, test_size=.30, random_state=0,shuffle=False)
```

Figure 22. Spit data

4. Reading data

Prophet's input is always a dataset with two attributes ‘ds’ and ‘y’. Where ‘ds’ has date format, timestamp. And the ‘y’ column represents the quantitative value, which represents the measurement we predict.

```
train = train.reset_index(level=0)
train.columns = ['ds','y']
train.head(5)
```

	ds	y
0	2012-01-02	32202288.50
1	2012-01-03	33607538.00
2	2012-01-04	33923003.00
3	2012-01-05	33628569.00
4	2012-01-06	34000652.75

```
test = test.reset_index(level=0)
test.columns = ['ds','y']
test.tail(5)
```

	ds	y
856	2022-12-26	41586906.0
857	2022-12-27	41894838.0
858	2022-12-28	41727084.0
859	2022-12-29	41961480.0
860	2022-12-30	42138426.0

Figure 23. Read data train and test

```
df = pd.concat([train, test],    # Combine vertically
                          ignore_index = True,
                          sort = False)
df.head(5)
```

	ds	y
0	2012-01-02	32202288.50
1	2012-01-03	33607538.00
2	2012-01-04	33923003.00
3	2012-01-05	33628569.00
4	2012-01-06	34000652.75

Figure 24. Combine data train and test

5. Data visualization

```
df.plot(x="ds", y='y',figsize=(18,6), label="Real Gold Price")
```

```
<AxesSubplot:xlabel='ds'>
```

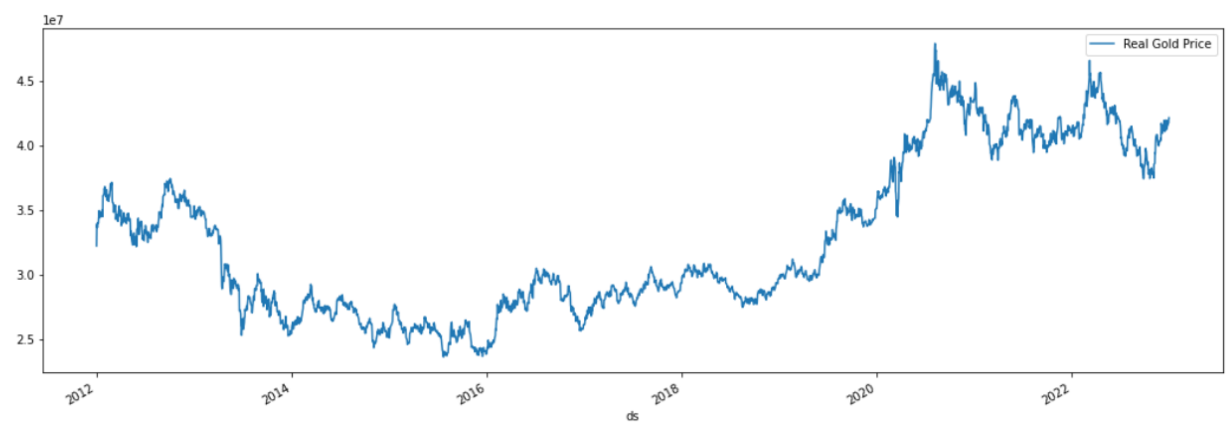

Figure 25. Visualize data

6. Check the train test set again

```
interrupt = len(train)
interrupt
```

```
2009
```

```
# Check size
print(train.shape)
print(test.shape)
```

```
(2009, 2)
(861, 2)
```

Figure 26. Check size

7. Create data to predict and compare with test set

Create data future with data test to predict and compare the results with the test set, using prophet and print out the model

```
future = test.copy()
```

```
m = Prophet(changepoint_prior_scale=0.099)
predict = m.fit(df).predict(future)
fig = m.plot(predict)
a = add_changepoints_to_plot(fig.gca(), m, predict)
```

Figure 27. Create predict data

Figure 28. Plot predict data

```
ax = predict.plot(x='ds',y='yhat',label='Predictions',legend=True,figsize=(16,8))
test.plot(x='ds',y='y',label='Test Gold Price',legend=True,ax=ax)
train.plot(x='ds',y='y',label='Train Gold Price',legend=True,ax=ax)
```

```
<AxesSubplot:xlabel='ds'>
```

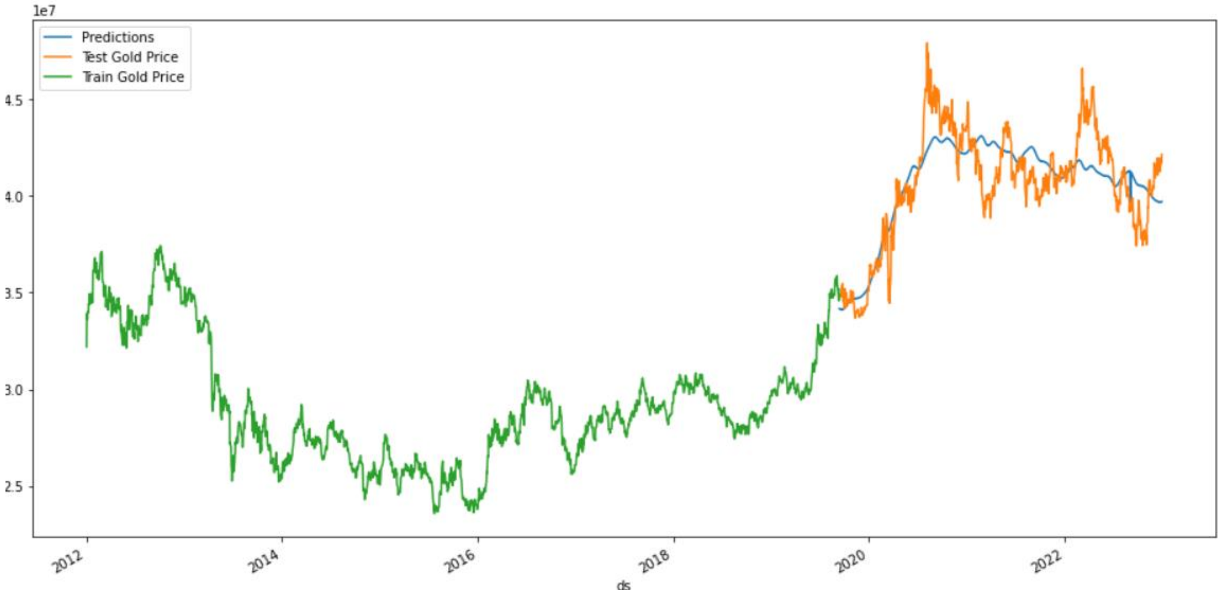

Figure 29. Plot predict data

8. Predict 1 year

Predictions are then made on a dataframe with a column ds containing the dates for which a prediction is to be made. You can get a suitable dataframe that extends into the future a specified number of days using the helper method "Prophet.make_future_dataframe". By default it will also include the dates from the history, so we will see the model fit as well.

```
]: future = m.make_future_dataframe(periods=365)
   future.tail()
```

]:

	ds
3230	2023-12-26
3231	2023-12-27
3232	2023-12-28
3233	2023-12-29
3234	2023-12-30

Figure 30. Create future data

Then we use predict function to predict 1 year later

```
: # Python
  forecast = m.predict(future)
  forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

:

	ds	yhat	yhat_lower	yhat_upper
3230	2023-12-26	3.844256e+07	3.210738e+07	4.512200e+07
3231	2023-12-27	3.843644e+07	3.179969e+07	4.543467e+07
3232	2023-12-28	3.845308e+07	3.230995e+07	4.541940e+07
3233	2023-12-29	3.845221e+07	3.228633e+07	4.508400e+07
3234	2023-12-30	3.634457e+07	2.999436e+07	4.326132e+07

Figure 31. Result of future data

9. Visualize the future model

After having predictive future data, we visualize the model

```
# Pythons
fig1 = m.plot(forecast)
a = add_changepoints_to_plot(fig1.gca(), m, forecast)
```

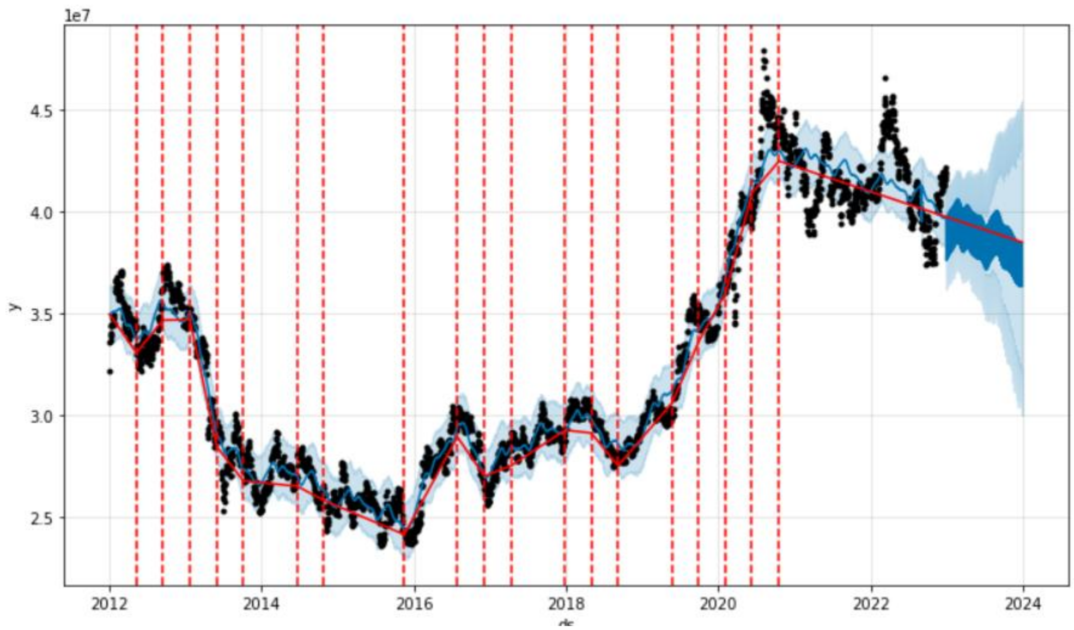

Figure 32. Plot future data

```
ax = forecast.plot(x='ds',y='yhat',label='Predictions',legend=True,figsize=(16,8))
test.plot(x='ds',y='y',label='Test Gold Price',legend=True,ax=ax)
train.plot(x='ds',y='y',label='Train Gold Price',legend=True,ax=ax)
```

Figure 33. Plot future data

We use the Prophet.plot_components method to see the forecast components. By default you'll see the trend, yearly seasonality, and weekly seasonality of the time series.

```
# Python
fig2 = m.plot_components(forecast)
```

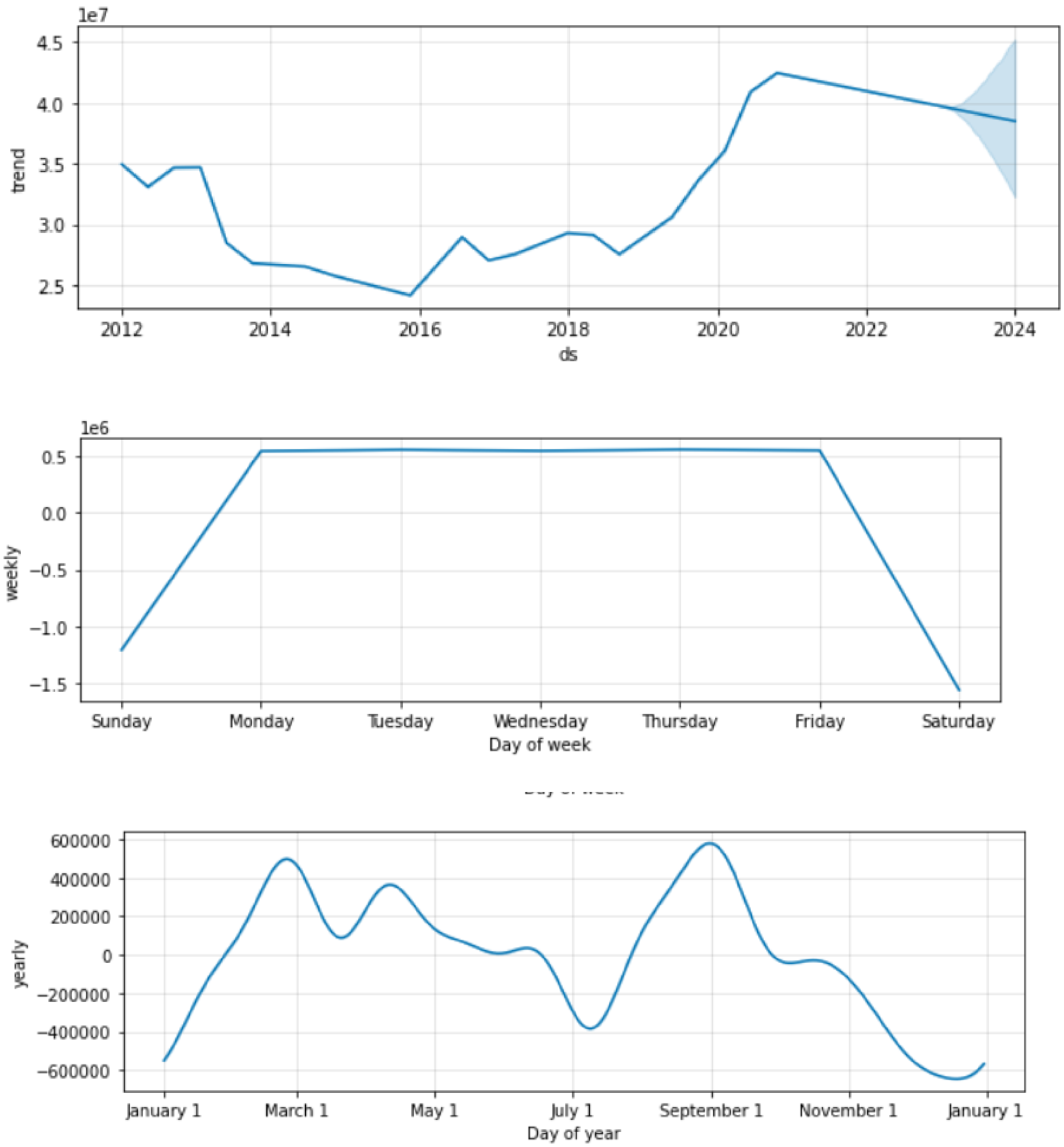

Figure 34. Components of data

10. Calculate MAPE, RMSE

```
mae = mean_absolute_error(test.y, predict[:interrupt].yhat)
mape = mean_absolute_percentage_error(test.y, predict[:interrupt].yhat)
mse = mean_squared_error(test.y, predict[:interrupt].yhat)
rmse = np.sqrt(mse)

# print(f"MAE: {mae:.2f}")
print(f"MAPE: {mape*100:.2f}%")
# print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse:.2f}")
```

```
MAPE: 2.99%
RMSE: 1573796.94
```

Figure 35. Result of MAPE, RMSE

We perform PROPHET model on 3 cases:

Table 3. Measuring the PROPHET model according to split data

Model	Train-Test	RMSE	MAPE
PROPHET	7-3	1573796.94	2.99%
	8-2	1664335.79	3.28%
	9-1	1753647.79	3.39%

After measuring the PROPHET model, the model with 70% data for training and 30% data for testing is the most optimal in all three cases. We see that the results are very good, with very low RMSE (1573796.94) and MAPE (2.99%), because there has been no strong fluctuation or sudden change over the years due to the characteristics of the gold data set.

C. LSTM CNN Model

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network that can learn order dependence in sequence prediction problems. This is a necessary characteristic in complex problem domains such as machine translation, speech recognition, and others. [7]

An LSTM layer is made up of a collection of recurrently connected memory blocks. Each block contains one or more recurrently connected memory cells through three multiplicative units - the input, output, and forget gates. These provide continuous analogs of the cells' write, read, and reset operations.

The advent of LSTM networks minimizes the drawback of gradient vanishing in part by allowing information to propagate more directly through the cell state.

LSTM cell:

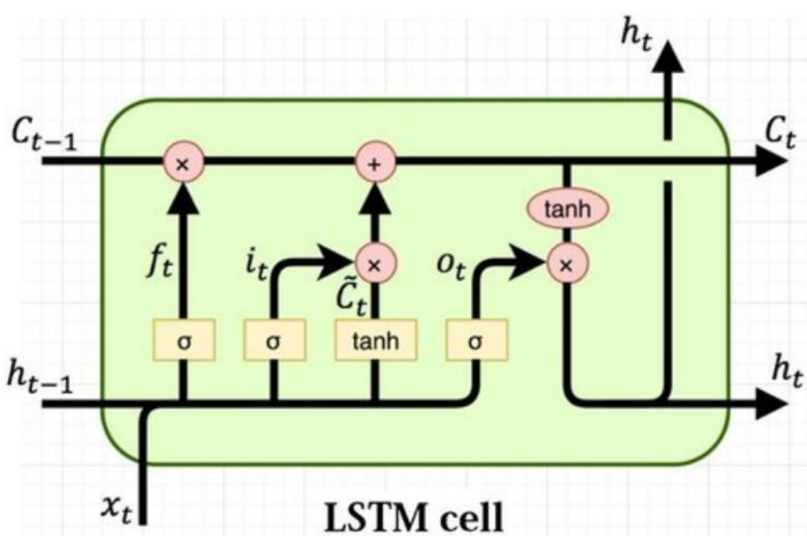

Figure 36. LSTM cell [8]

Calculate in LSTM cell:

$$Forget\ gate: f_t = \sigma(W_f x_t + U_f h_{t-1})$$

$$Input\ gate: i_t = \sigma(W_i x_t + U_i h_{t-1})$$

$$Cell\ gate: c_t = \tanh(W_c x_t + U_c h_{t-1})$$

$$Output\ gate: o_t = \phi h(W_o x_t + U_o h_{t-1})$$ [9]

$$Cell\ state: c_t = f_t \times c_{t-1} + i_t \times c_t$$

Implementation in Python language

1. Import libraries

```python
import pandas as pd
import datetime as dt
import numpy as np
import math

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM

from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error, r2_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

Figure 37. Import libraries

2. Import data

```python
df = pd.read_csv('Data-Gold.csv')
df['Date'] = pd.to_datetime(df['Date'])
df.set_index("Date",inplace=True)
df_VND = df['VND'].values.reshape(-1, 1)

df
```

	VND
Date	
2012-01-02	32202288.50
2012-01-03	33607538.00
2012-01-04	33923003.00
2012-01-05	33628569.00
2012-01-06	34000652.75
...	...
2022-12-26	41586906.00
2022-12-27	41894838.00
2022-12-28	41727084.00
2022-12-29	41961480.00

Figure 38. Import data

3. Visualization Gold Price in the past

```python
formatter = ticker.StrMethodFormatter('{x:,.0f} VND Dong')

title = 'Gold Price History'
ylabel = 'VND'
xlabel = 'Date'

ax = df['VND'].plot(figsize=(16, 9), title=title)
ax.autoscale(axis='x', tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter)
ax.grid(True)
```

Figure 39. Visualize data

4. Train Split

```
gold_price=df['VND']
values=gold_price.values
Train_len=math.ceil(len(values)*0.9)
Train_len
```

Figure 40. Split data

5. Scale data

```
sc=MinMaxScaler(feature_range=(0,1))
scaled_data=sc.fit_transform(df)
scaled_data
```

```
array([[0.35425019],
       [0.4120421 ],
       [0.42501583],
       ...,
       [0.74596438],
       [0.75560409],
       [0.76288112]])
```

Figure 41. Scale data

6. Create Training Data

```
[ ]
train_data=scaled_data[0:Train_len,:]

train_x=[]
train_y=[]

for i in range(365, len(train_data)):  # 1 years
  train_x.append(train_data[i-365:i,0])
  train_y.append(train_data[i,0])

train_x,train_y=np.array(train_x), np.array(train_y)

train_x=np.reshape(train_x,(train_x.shape[0],train_x.shape[1],1))

```

Figure 42. Create Training Data

7. Build Model

```
# define model
model = Sequential()
model.add(LSTM(100, return_sequences=True,input_shape=(train_x.shape[1],1)))
model.add(LSTM(100, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
# compile model
model.compile(loss='mse', optimizer='adam')
# fit model
m=model.fit(train_x, train_y,batch_size= 32, epochs=50)
```

```
Epoch 1/50
70/70 [==============================] - 54s 644ms/step - loss: 0.0063
Epoch 2/50
70/70 [==============================] - 38s 548ms/step - loss: 7.3701e-04
Epoch 3/50
70/70 [==============================] - 39s 558ms/step - loss: 6.5130e-04
Epoch 4/50
70/70 [==============================] - 37s 531ms/step - loss: 6.4069e-04
Epoch 5/50
70/70 [==============================] - 37s 530ms/step - loss: 5.6707e-04
Epoch 6/50
70/70 [==============================] - 39s 551ms/step - loss: 5.2777e-04
```

Figure 43. Build model

8. Save Model

```python
model.save("GoldPrice_lstm_9_1.h5")
```

Figure 44. Save model

9. Create Testing Data

```python
# Create the testing data
test_data=scaled_data[Train_len-365:,:]
test_x=[]
test_y=df[Train_len:]
for i in range (365, len(test_data)):
  test_x.append(test_data[i-365:i,0])
test_x=np.array(test_x)
test_x=np.reshape(test_x,(test_x.shape[0],test_x.shape[1],1))
```

Figure 45. Create Test data

10.Evaluate On Test Data

```python
Predictions= model.predict(test_x)
Predictions=sc.inverse_transform(Predictions)
```

```
9/9 [==============================] - 2s 163ms/step
```

Figure 46. Evaluate

11.Plot and visualize data

```
#plot data
train=df[:Train_len]
validation=df[Train_len:]
validation['Predictions']= Predictions
#visualize data
plt.figure(figsize=(16,8))
plt.title('Forcasting Gold Price According To The LSTM Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Vietnamese Dong',fontsize=18)
plt.plot(train['VND'])
plt.plot(validation[['VND','Predictions']])
plt.legend(['Training','Validation','Predictions'])
plt.show()
```

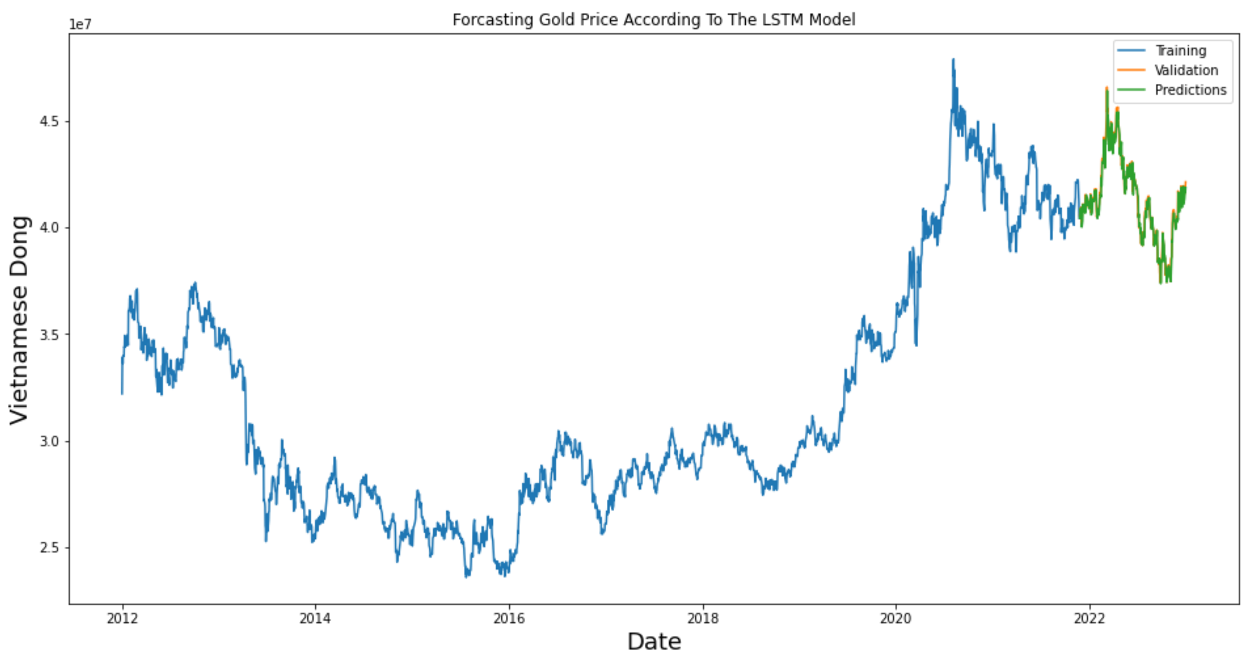

Figure 47. Visualize data

12.Compare the actual and prediction values

```
1 validation
```

Date	VND	Predictions
2021-11-25	40546299.04	40436168.0
2021-11-26	40840344.31	40606868.0
2021-11-29	40523204.39	40902272.0
2021-11-30	40948149.18	40516852.0
2021-12-01	40642813.75	40915240.0
...	...	...
2022-12-26	41586906.00	41354580.0
2022-12-27	41894838.00	41535848.0
2022-12-28	41727084.00	41851036.0
2022-12-29	41961480.00	41644092.0
2022-12-30	42138426.00	41872344.0

287 rows × 2 columns

Figure 48. Compare

13.Measure model by using MAPE and RMSE

```
mae = mean_absolute_error(data, Predictions)
mape = mean_absolute_percentage_error(data, Predictions)
mse = mean_squared_error(data, Predictions)
rmse = np.sqrt(mse)
r2 = r2_score(data, Predictions)
print(f"MAPE: {mape * 100:.2f}%")
print(f"RMSE: {rmse:.0f}")
```

```
MAPE: 0.74%
RMSE: 404883
```

Figure 49. Calculate MAPE, RMSE

14.Predict the next 30 days

1. Take records and create list

```
#Getting the last 100 days records
future=test_data[552:]
```

```
future=future.reshape(1,-1)
temp=list(future)
future.shape
```

```
(1, 100)
```

```
#Creating list of the last 100 data
temp=temp[0].tolist()
```

Figure 50. Create List

2. Predict next 30 days price using the current data

```python
#Predicting next 30 days price using the current data
lst_output=[]
n_steps=100
i=0
while(i<30):

    if(len(temp)>100):
        future = np.array(temp[1:])
        future=future.reshape(1,-1)
        future = future.reshape((1, n_steps, 1))
        yhat = model.predict(future, verbose=0)
        temp.extend(yhat[0].tolist())
        temp = temp[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        future = future.reshape((1, n_steps,1))
        yhat = model.predict(future, verbose=0)
        temp.extend(yhat[0].tolist())
        lst_output.extend(yhat.tolist())
        i=i+1

print(lst_output)
```

```
[[0.7594186663627625], [0.7549229860305786], [0.7500720620155334], [0.7458367943763 7
```

Figure 51. Predict

3. Inverse transformations and print out

```
tam=sc.inverse_transform(lst_output)
print(tam)
```

```
[[42054234.05557089]
 [41944918.54876437]
 [41826965.05162882]
 [41723981.6555182 ]
 [41642517.87021384]
 [41579905.48427352]
 [41529213.81485271]
 [41483252.74953154]
 [41436809.05824066]
 [41387329.02638949]
 [41334531.48449416]
 [41279551.2557808 ]
 [41224053.61724404]
 [41169563.26113658]
 [41117117.90575971]
 [41067171.19053841]
 [41019715.86883651]
 [40974441.7846254 ]
 [40930924.28502484]
 [40888743.06513625]
 [40847557.53305908]
 [40807134.34710825]
 [40767332.92254185]
 [40728067.74905293]
 [40689292.44816991]
 [40650972.23603912]
 [40613075.22746137]
 [40575556.49329232]
 [40538366.75640594]
 [40501469.78362131]]
```

Figure 52. Inverse and print out

4. Visualization the next 30 days price of gold

```
#Creating a dummy plane to plot graph one after another
plot_new=np.arange(1,101)
plot_pred=np.arange(101,131)
plt.plot(plot_new, sc.inverse_transform(scaled_data[2770:]))
plt.plot(plot_pred, sc.inverse_transform(lst_output),c='pink')
```

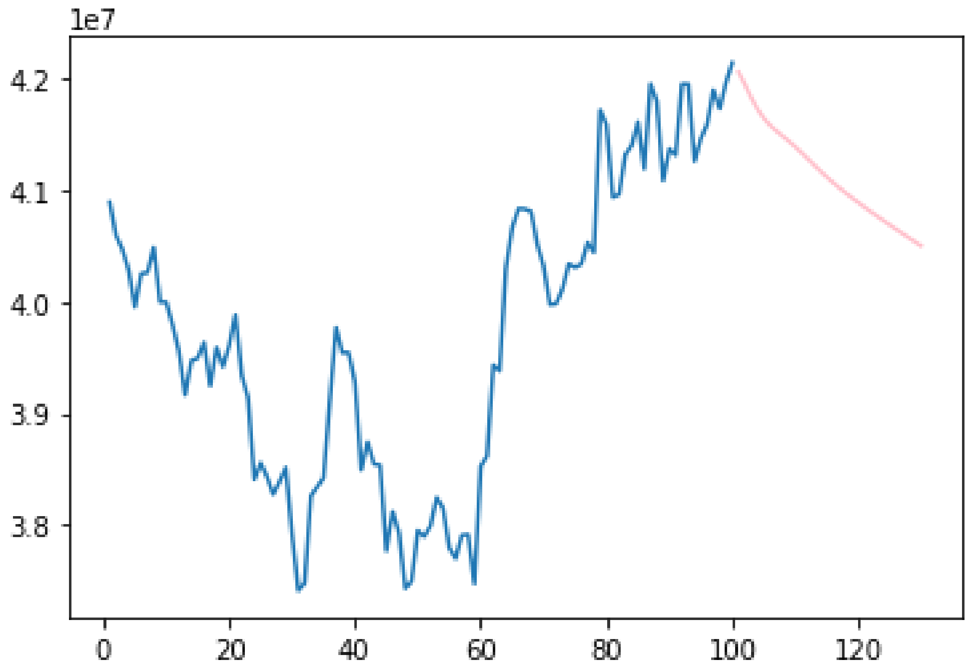

Figure 53. Visualize predict data

```
dataset_new=scaled_data.tolist()
```

```
len(dataset_new)
```

```
2870
```

```
#Entends helps us to fill the missing value with approx value
dataset_new.extend(lst_output)
```

```
final=sc.inverse_transform(dataset_new).tolist()

plt.ylabel("Gold Price in Future")
plt.xlabel("Time")
plt.title("Prediction of next 30 days for Gold Price")
plt.plot(final,)
plt.show()
```

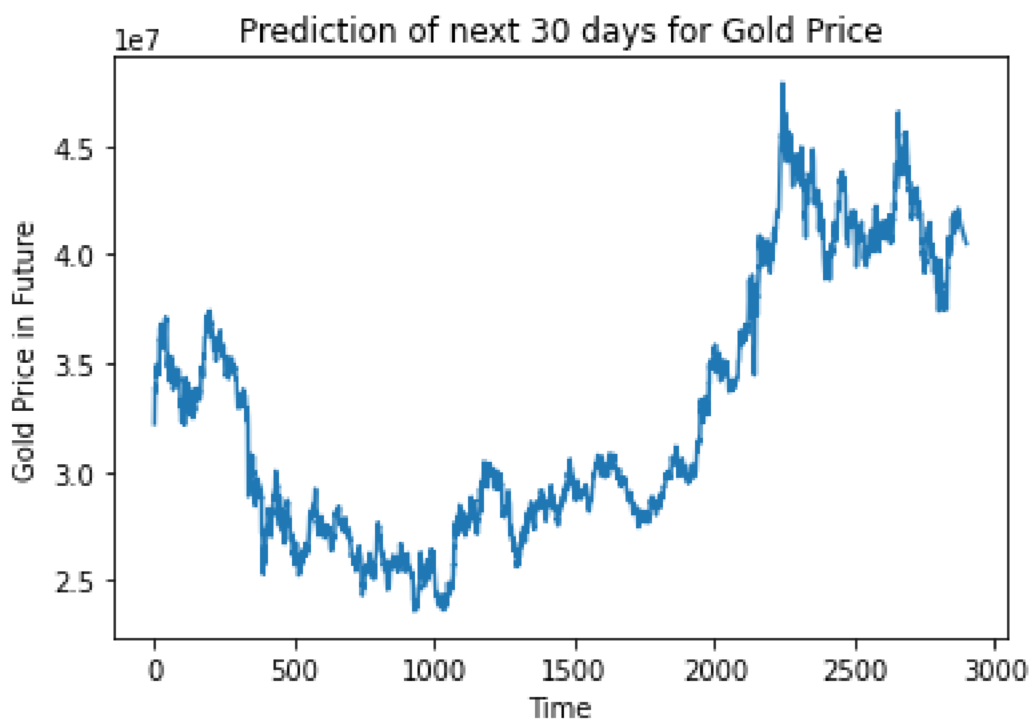

Figure 54. Visualize predict data 30 days

- **Conclusion:** After train and test split for three cases such as 90% train with 10% test, 80% train with 20% test, and 70% train with 30% test. I had the table of results below:

Table 4. Measuring the LSTM model according to split data

Model	Train-Test	RMSE	MAPE
LSTM	7-3	465100	0.81%
	8-2	512538	1.01%
	9-1	404883	0.74%

After measuring the LSTM model, the model with 90% data for training and 10% data for testing is the most optimal in all three cases. We see that the results are very good, with very low RMSE (404883) and MAPE (0.74%), because there has been no strong fluctuation or sudden change over the years due to the characteristics of the gold data set.

D. Bidirectional LSTM

Bidirectional long-short term memory (BiLSTM) is a technique that allows any neural network to store sequence information both forward and backward. BiLSTM allows input flow in both directions, whereas normal LSTM only allows input flow in one direction. **[10]**

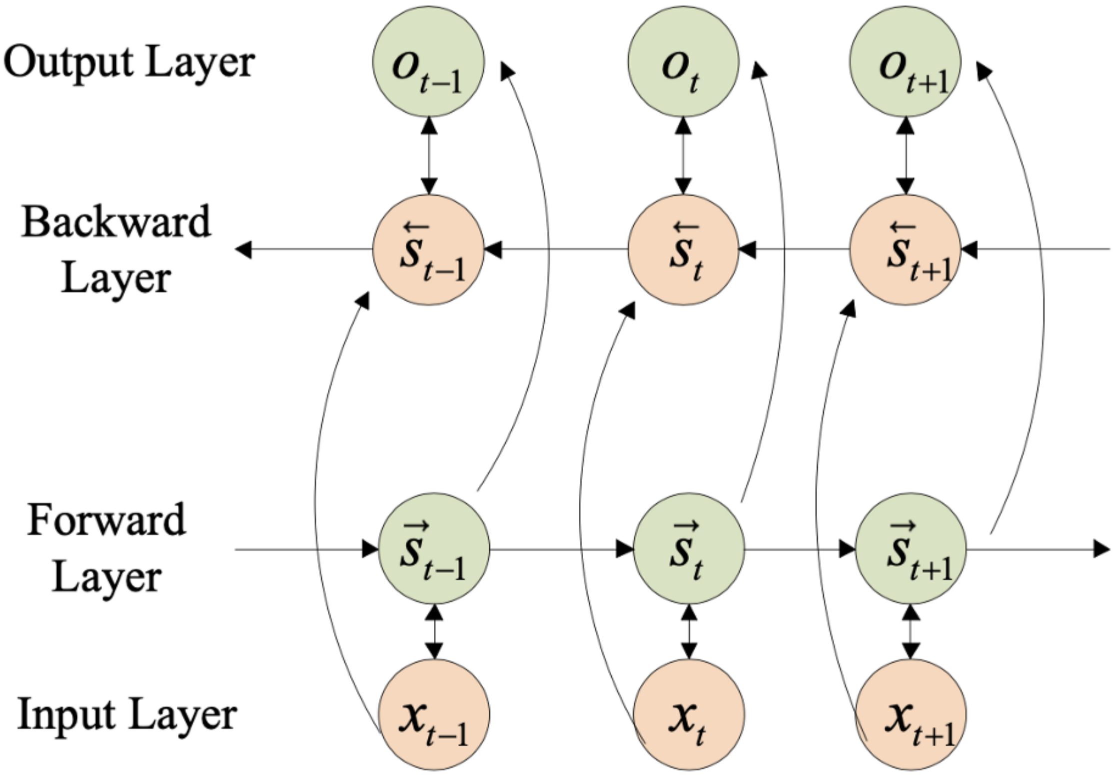

Figure 55. The basic structure of bidirectional LSTM **[11]**

Implementation in Python language

1. Import libraries

```
import pandas as pd
import datetime as dt
import numpy as np
import math

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Bidirectional


from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error, r2_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

Figure 56. Import libraries

2. Import data

```
df = pd.read_csv('Data-Gold.csv')
df['Date'] = pd.to_datetime(df['Date'])
df.set_index("Date",inplace=True)
df_VND = df['VND'].values.reshape(-1, 1)

df
```

Date	VND
2012-01-02	32202288.50
2012-01-03	33607538.00
2012-01-04	33923003.00
2012-01-05	33628569.00
2012-01-06	34000652.75
...	...
2022-12-26	41586906.00
2022-12-27	41894838.00
2022-12-28	41727084.00
2022-12-29	41961480.00
2022-12-30	42138426.00

Figure 57. Import data

3. Visualization Gold Price in the past

```
formatter = ticker.StrMethodFormatter('{x:,.0f} VND Dong')

title = 'Gold Price History'
ylabel = 'VND'
xlabel = 'Date'

ax = df['VND'].plot(figsize=(16, 9), title=title)
ax.autoscale(axis='x', tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter)
ax.grid(True)
```

Figure 58. Visualize data

4. Train Split

```
gold_price=df['VND']
values=gold_price.values
Train_len=math.ceil(len(values)*0.9)
Train_len
```

Figure 59. Split data

5. Scale data

```
sc=MinMaxScaler(feature_range=(0,1))
scaled_data=sc.fit_transform(df)
scaled_data
```

```
array([[0.35425019],
       [0.4120421 ],
       [0.42501583],
       ...,
       [0.74596438],
       [0.75560409],
       [0.76288112]])
```

Figure 60. Scale data

6. Create training data

```
[ ]
train_data=scaled_data[0:Train_len,:]

train_x=[]
train_y=[]

for i in range(365, len(train_data)):  # 1 years
  train_x.append(train_data[i-365:i,0])
  train_y.append(train_data[i,0])

train_x,train_y=np.array(train_x), np.array(train_y)

train_x=np.reshape(train_x,(train_x.shape[0],train_x.shape[1],1))

```

Figure 61. Create Train data

7. Build Model

```
# define model
model = Sequential()
model.add(Bidirectional(LSTM(100,return_sequences=True,input_shape=(train_x.shape[1],1))))
model.add(Bidirectional(LSTM(100,return_sequences=False)))
model.add(Dense(25))
model.add(Dense(1))
```

```
[ ]
# compile model
model.compile(loss='mse', optimizer='adam', metrics=['acc'])
# fit model
m=model.fit(train_x, train_y,batch_size= 32, epochs=50)
```

```
Epoch 1/50
70/70 [==============================] - 110s 1s/step - loss: 0.0050 - acc: 9.0171e-04
Epoch 2/50
70/70 [==============================] - 93s 1s/step - loss: 5.3109e-04 - acc: 9.0171e-04
Epoch 3/50
70/70 [==============================] - 92s 1s/step - loss: 4.9189e-04 - acc: 9.0171e-04
Epoch 4/50
70/70 [==============================] - 92s 1s/step - loss: 5.2727e-04 - acc: 9.0171e-04
Epoch 5/50
70/70 [==============================] - 93s 1s/step - loss: 4.4545e-04 - acc: 9.0171e-04
Epoch 6/50
70/70 [==============================] - 93s 1s/step - loss: 3.7986e-04 - acc: 9.0171e-04
```

Figure 62. Build model

8. Save Model

```
[ ]
model.save("GoldPrice_bilstm_7_3.h5")
```

Figure 63. Save model

9. Create testing data

```
# Create the testing data
test_data=scaled_data[Train_len-365:,:]
test_x=[]
test_y=df[Train_len:]
for i in range (365, len(test_data)):
  test_x.append(test_data[i-365:i,0])
test_x=np.array(test_x)
test_x=np.reshape(test_x,(test_x.shape[0],test_x.shape[1],1))
```

Figure 64. Create Test data

10. Evaluate on test data

```
[ ]
Predictions= model.predict(test_x)
Predictions=sc.inverse_transform(Predictions)

9/9 [==============================] - 5s 385ms/step
```

Figure 65. Evaluate

11. Plot and visualize data

```
#plot data
train=df[:Train_len]
validation=df[Train_len:]
validation['Predictions']= Predictions
#visualize data
plt.figure(figsize=(16,8))
plt.title('Forcasting Gold Price According To The Bi-LSTM Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Vietnamese Dong',fontsize=18)
plt.plot(train['VND'])
plt.plot(validation[['VND','Predictions']])
plt.legend(['Training','Validation','Predictions'])
plt.show()
```

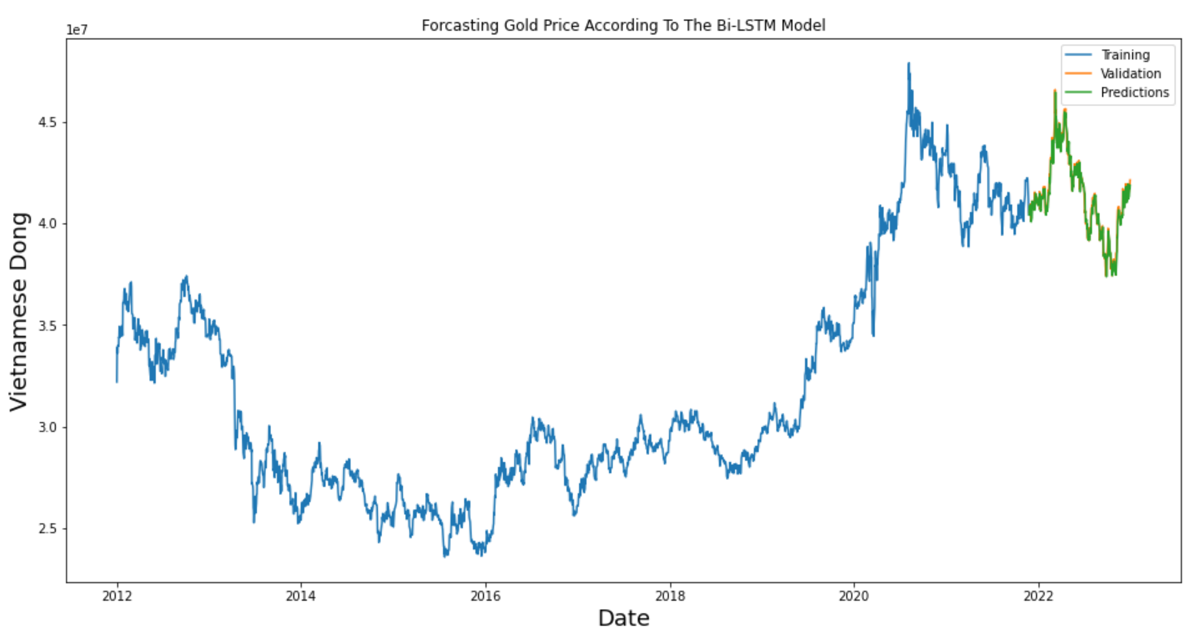

Figure 66. Visualze data

12. Compare the actual and prediction values

```
validation
```

Date	VND	Predictions
2021-11-25	40546299.04	40420472.0
2021-11-26	40840344.31	40598936.0
2021-11-29	40523204.39	40907112.0
2021-11-30	40948149.18	40590880.0
2021-12-01	40642813.75	40900380.0
...	...	...
2022-12-26	41586906.00	41331280.0
2022-12-27	41894838.00	41538016.0
2022-12-28	41727084.00	41863228.0
2022-12-29	41961480.00	41707044.0
2022-12-30	42138426.00	41880744.0

287 rows × 2 columns

Figure 67. Compare

13. Measure model by using MAPE and RMSE

```
mae = mean_absolute_error(data, Predictions)
mape = mean_absolute_percentage_error(data, Predictions)
mse = mean_squared_error(data, Predictions)
rmse = np.sqrt(mse)
r2 = r2_score(data, Predictions)
print(f"MAPE: {mape * 100:.2f}%")
print(f"RMSE: {rmse:.0f}")
```

```
MAPE: 0.73%
RMSE: 403701
```

Figure 68. Calculate MAPE,RMSE

14. Predict the next 30 days

14.1 Take records and create list

```
#Getting the last 100 days records
future=test_data[552:]
```

```
future=future.reshape(1,-1)
temp=list(future)
future.shape
```

```
(1, 100)
```

```
#Creating list of the last 600 data
temp=temp[0].tolist()
```

Figure 69. Create List

14.2 Predict next 30 days price using the current data

```
#Predicting next 30 days price uisng the current data
lst_output=[]
n_steps=100
i=0
while(i<30):

    if(len(temp)>100):
        future = np.array(temp[1:])
        future=future.reshape(1,-1)
        future = future.reshape((1, n_steps, 1))
        yhat = model.predict(future, verbose=0)
        temp.extend(yhat[0].tolist())
        temp = temp[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        future = future.reshape((1, n_steps,1))
        yhat = model.predict(future, verbose=0)
        temp.extend(yhat[0].tolist())
        lst_output.extend(yhat.tolist())
        i=i+1

print(lst_output)
```

```
[[0.7600336670875549], [0.757377564907074], [0.7539637088775635],
```

Figure 70. Predict 30 days

14.3 Inverse tranformations and print out

```
tam=sc.inverse_transform(lst_output)
print(tam)
```

```
[[42069188.21399622]
 [42004603.29368713]
 [41921593.07622153]
 [41845726.5927042 ]
 [41786211.4190681 ]
 [41742551.88539837]
 [41710226.09071739]
 [41683483.10455108]
 [41658151.76311304]
 [41631269.64153203]
 [41601988.88337476]
 [41570474.71194617]
 [41537577.88233405]
 [41503895.51735979]
 [41470411.71021692]
 [41437603.28956621]
 [41406083.32082868]
 [41375686.58069938]
 [41346788.44493251]
 [41319094.70009912]
 [41292405.33904056]
 [41266794.27744588]
 [41241754.25078236]
 [41217317.14424919]
 [41192826.41260826]
 [41168279.15720512]
 [41143402.90451931]
 [41117757.05907097]
 [41091043.05944938]
 [41063372.50385176]]
```

Figure 71. Result predict data

14.4 visualization the next 30 days price of gold

```python
#Creating a dummy plane to plot graph one after another
plot_new=np.arange(1,101)
plot_pred=np.arange(101,131)
plt.plot(plot_new, sc.inverse_transform(scaled_data[2770:]))
plt.plot(plot_pred, sc.inverse_transform(lst_output),c='pink')
```

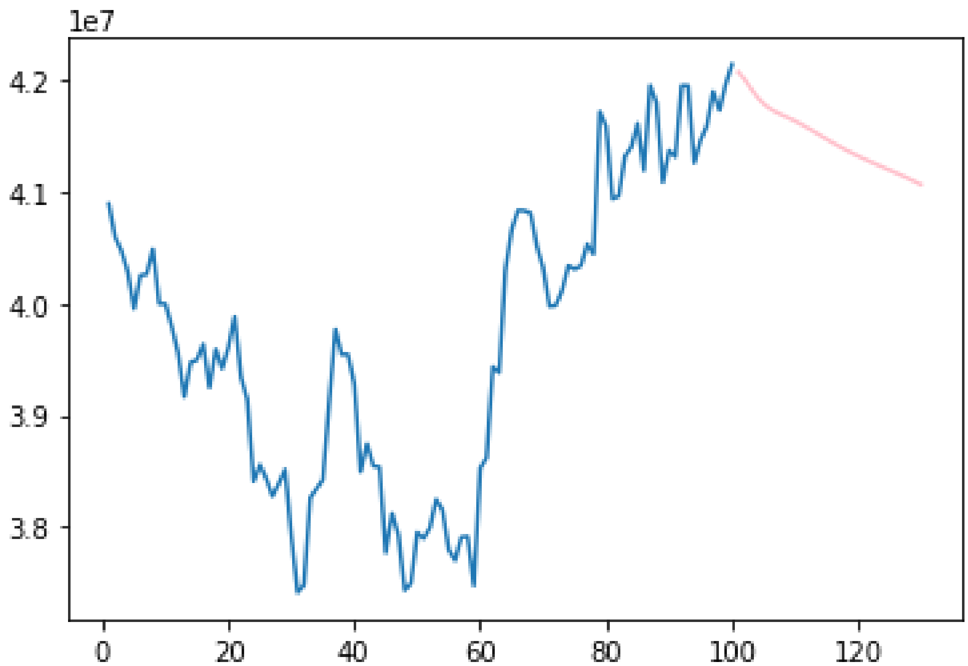

Figure 72. Visualize the next 30 days

```python
#Entends helps us to fill the missing value with approx value
dataset_new.extend(lst_output)
```

```python
final=sc.inverse_transform(dataset_new).tolist()

plt.ylabel("Gold Price in Future")
plt.xlabel("Time")
plt.title("Prediction of next 30 days for Gold Price")
plt.plot(final,)
plt.show()
```

Figure 73. Visualize the next 30 days

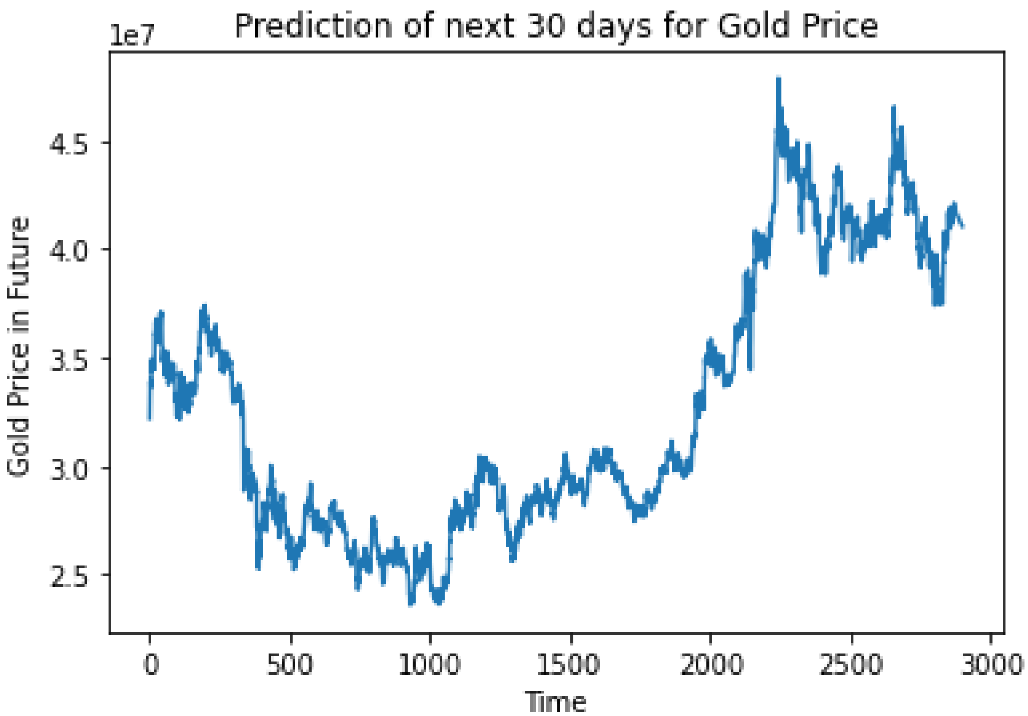

Figure 74. Visualize the next 30 days

- **Conclusion:** After train and test split for three cases such as 90% train with 10% test, 80% train with 20% test, and 70% train with 30% test. I had the table of results below:

Table 5. Measuring the Bi-LSTM model according to split data

Model	Train-Test	RMSE	MAPE
Bi-LSTM	7-3	493253	0.96%
	8-2	511867	0.97%
	9-1	397070	0.73%

After measuring the Bi-LSTM model, the model with 90% data for training and 10% data for testing is the most optimal in all three cases. We see that the results are very good, with very low RMSE (397070) and MAPE (0.73%), because there has been no strong fluctuation or sudden change over the years due to the characteristics of the gold data set.

E. Linear Regression

Definition

In the most simple words, Linear Regression is the supervised Machine Learning model in which the model finds the best fit linear line between the independent and dependent variable i.e it finds the linear relationship between the dependent and independent variable. [12]

Linear Regression is of two types: Simple and Multiple. Simple Linear Regression is where only one independent variable is present and the model has to find the linear relationship of it with the dependent variable. Whereas, In Multiple Linear Regression there are more than one independent variables for the model to find the relationship.

Equation of Simple Linear Regression, where b_0 is the intercept, b_1 is coefficient or slope, x is the independent variable and y is the dependent variable.

$$y = b_0 + b_1 x$$

Equation of Multiple Linear Regression, where b_0 is the intercept, b_1, b_2, b_3, b_4…,b_n are coefficients or slopes of the independent variables x_1, x_2, x_3, x_4…, x_n and y is the dependent variable.

$$y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + \cdots + b_n x_n$$

A Linear Regression model’s main aim is to find the best fit linear line and the optimal values of intercept and coefficients such that the error is minimized.

Error is the difference between the actual value and Predicted value and the goal is to reduce this difference.

Mathematical Approach:

Residual/Error = Actual values – Predicted Values

Sum of Residuals/Errors = Sum(Actual- Predicted Values)

Square of Sum of Residuals/Errors = (Sum(Actual- Predicted Values))^2

Implementation in Python language

1. First step we need to add the necessary libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

from sklearn.metrics import mean_squared_error
from sklearn.utils import column_or_1d
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error
```

Figure 75. Import libraries

2. Next, we need to get the data for analysis

```
df = pd.read_csv("C:\\Users\\DNV\\OneDrive\\Desktop\\Data-Gold.csv",parse_dates=True,index_col=0)
```

Figure 76. Import data

3. To be able to predict by time series we need to create an additional column "Timestamp"

```
df['Timestamp'] = pd.to_datetime(df.index).astype(np.int64) / 10**9
df_index = df.index
df_open = df['VND'].values.reshape(-1, 1)
df.head()

df
```

	VND	Timestamp
Date		
2012-01-02	32202288.50	1.325462e+09
2012-01-03	33607538.00	1.325549e+09
2012-01-04	33923003.00	1.325635e+09
2012-01-05	33628569.00	1.325722e+09
2012-01-06	34000652.75	1.325808e+09
...	...	...
2022-12-26	41586906.00	1.672013e+09
2022-12-27	41894838.00	1.672099e+09
2022-12-28	41727084.00	1.672186e+09
2022-12-29	41961480.00	1.672272e+09
2022-12-30	42138426.00	1.672358e+09

2870 rows × 2 columns

Figure 77. Create 'Timestamp'

4. Draw graphs to visualize input data

```python
formatter = ticker.StrMethodFormatter('VND{x:,.0f}')

title = 'Gold Price'
ylabel = 'VND'
xlabel = 'Date'

ax = df['VND'].plot(figsize=(16, 9), title=title)
ax.autoscale(axis='x', tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter)
ax.grid(True)
```

Figure 78. Visualize data

5. **Then we start normalizing / scaling the input data with the StandardScaler() method.**

```
LR_sc = StandardScaler()

df_scaled = df.copy()
df_scaled['VND'] = LR_sc.fit_transform(df_open)
df_scaled
```

```
X_sc = StandardScaler()
y_sc = StandardScaler()

X = df.iloc[:, 1].values.reshape(-1, 1)
y = df.iloc[:, 0].values.reshape(-1, 1)


X_scaled = X_sc.fit_transform(X)
y_scaled = y_sc.fit_transform(y)


df_scaled = pd.DataFrame(index=df_index)
df_scaled['Timestamp'] = X_scaled
df_scaled['VND'] = y_scaled
df_scaled.head()
```

Figure 79. Scale data

6. **To have data for train and test, we need to divide the normalized data set into 2 fractions with a ratio of 9/1 corresponding to train and test.**

```
interrupt = int(len(df_scaled) * .9)

train_data, test_data = df_scaled[:interrupt], df_scaled[interrupt:]
index_test = df_scaled.index[interrupt:]
print(train_data.shape)
print(test_data.shape)
```

```
(2583, 2)
(287, 2)
```

Figure 80. Split data

```
plt.figure(figsize=(16, 9))
plt.grid(True)
plt.ylabel('Gold Prices')
plt.plot(train_data['VND'], 'blue', label='Train data')
plt.plot(test_data['VND'], 'green', label='Test data')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1850350fcd0>
```

Figure 81. Visualize data

7. Train and run the prediction results

```
from sklearn.linear_model import LinearRegression
LR_model = LinearRegression()
LR_model.fit(X_train,y_train)
pred = LR_model.predict(X_test)
pred
```

```
array([[0.69174327],
       [0.6922137 ],
       [0.69362501],
       [0.69409545],
       [0.69456588],
       [0.69503632],
       [0.69550675],
       [0.69691806],
       [0.69738849],
       [0.69785893],
       [0.69832936],
       [0.6987998 ],
       [0.70021111],
```

Figure 82. Build model

8. After training and predicting the value, we return the data to its original form.

```
inv_pred = y_sc.inverse_transform(pred.reshape(-1, 1))
inv_test = y_sc.inverse_transform(y_test.reshape(-1, 1))
```

Figure 83. Inverse data

9. Plot graphs to see predicted results against test data.

```
plt.figure(figsize=(16, 9))
plt.grid(True)
plt.ylabel('Gold Prices')
plt.plot(column_or_1d(inv_test), 'blue', label='Actual data')
plt.plot(column_or_1d(inv_pred), 'red', label='Predicted data')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x18505f309d0>
```

Figure 84. Visualize predictive data

10. Finally, we evaluate the model through MAE, MAPE, MSE, RMSE and R2 indexes.

```
from sklearn.metrics import r2_score
mae = mean_absolute_error(inv_test, inv_pred)
mape = mean_absolute_percentage_error(inv_test, inv_pred)
mse = mean_squared_error(inv_test, inv_pred)
rmse = np.sqrt(mse)
r2 = r2_score(inv_test, inv_pred)
print(f"MAE: {mae:.2f}")
print(f"MAPE: {mape*100:.2f}%")
print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R2: {r2:.2f}")
```

```
MAE: 3941190.10
MAPE: 9.33%
MSE: 19858429883611.97
RMSE: 4456279.83
R2: -4.39
```

Figure 85. Calculate MAPE,RMSE

11.Predicting next 30 days price

```
df1 = pd.read_csv("C:\\Users\\DNV\\OneDrive\\Desktop\\next30days.csv",parse_dates=True,index_col=0)
```

```
df1['Timestamp'] = pd.to_datetime(df1.index).astype(np.int64) / 10**9
df_index1 = df1.index
df1
```

```
X_sc = StandardScaler()
X = df1.iloc[:, 1].values.reshape(-1, 1)
X_scaled = X_sc.fit_transform(X)
df_scaled_future = pd.DataFrame(index=df_index1)
df_scaled_future['Timestamp'] = X_scaled
df_scaled_future.head()
```

```
X_future = df_scaled_future['Timestamp'].values.reshape(-1, 1)
```

```
pred_future = LR_model.predict(X_future)
```

```
inv_pred_future = y_sc.inverse_transform(pred_future.reshape(-1, 1))
```

```
df_pred1 = pd.DataFrame(columns=['Pred'], index=df_index1)
df_pred1['Pred'] = column_or_1d(inv_pred_future)
df_pred1.head()
```

```
df_pred1.to_csv(r"C:\\Users\\DNV\\OneDrive\\Desktop\\LR30d.csv")
```

Figure 86. Predict next 30 days

F. Support Vector Regression

Definition

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. The main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated. In SVR, the best fit line is the hyperplane that has the maximum number of points. **[13]**

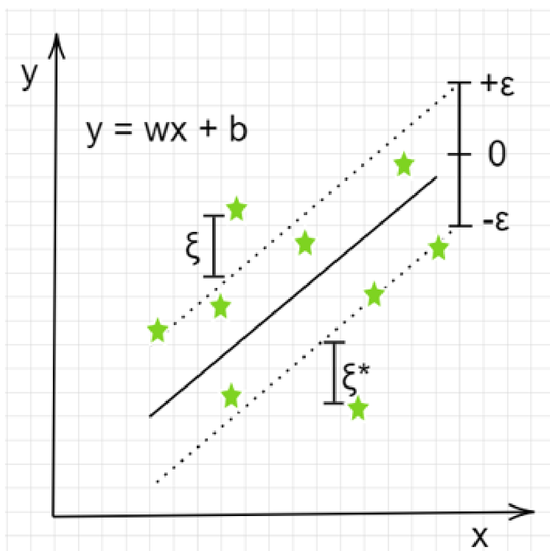

Figure 87.

Solution: $min \frac{1}{2} \|w\|^2$

Constraints: $y_i - wx_i - b \leq \varepsilon$

$wx_i + b - y_i \leq \varepsilon$

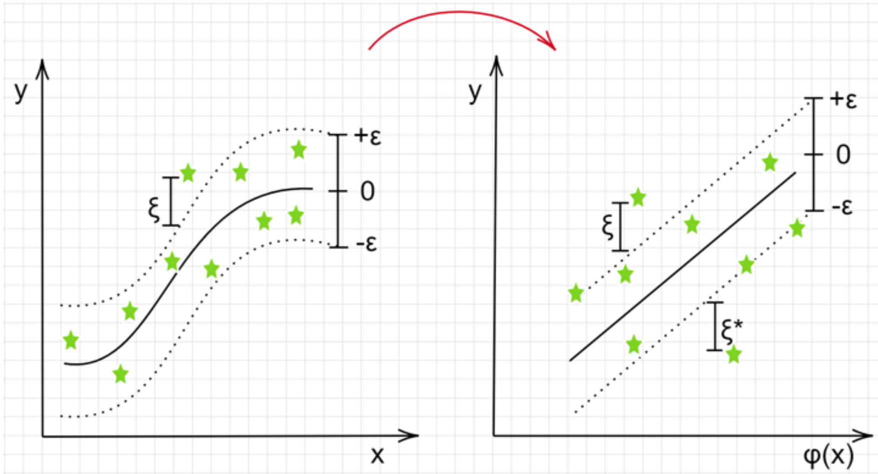

Figure 88

Minimize: $\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$

Constraints: $y_i - wx_i - b \leq \varepsilon + \xi_i$

$$wx_i + b - y_i \leq \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

Linear SVR: $y = \sum_{i=1}^{N}(a_i - a_i^*).\langle x_i, x\rangle + b$

Non-linear SVR:

The kernel functions transform the data into a higher dimensional feature space to make it possible to perform the linear separation.

$$y = \sum_{i=1}^{N}(a_i - a_i^*).\langle \varphi(x_i), \varphi(x)\rangle + b$$

$$y = \sum_{i=1}^{N}(a_i - a_i^*).K(x_i, x) + b$$

Types of Kernel functions: **[14]**

Gaussian RBF Kernel: The RBF kernel is also called the Gaussian kernel. There is an infinite number of dimensions in the feature space because it can be expanded by the Taylor Series. In the format below, The γ parameter defines how much influence a single training example has. The larger it is, the closer other examples must be to be affected. It is a general-purpose kernel; used when there is no prior knowledge about the data.

$$k(x, y) = \exp\left(-\gamma\left\|x_i - x_j\right\|^2\right)$$

Polynomial kernel: "Intuitively, the polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these". With n original features and d degrees of polynomial, the polynomial kernel yields n^d expanded features where d is the degree of polynomial. It is popular in image processing.

$$k(x, y) = (x_i . x_j + 1)^d$$

Sigmoid Kernel: The Hyperbolic Tangent Kernel is also known as the Sigmoid Kernel and as the Multilayer Perceptron (MLP) kernel. The Sigmoid Kernel comes from the Neural Networks field, where the bipolar sigmoid function is often used as an activation function for artificial neurons. We can use it as the proxy for neural networks.

$$k(x, y) = \tanh(\alpha x^T y + c)$$

Implementation in Python language

1. First step we need to add the necessary libraries

```
import matplotlib.pyplot as plt
import warnings
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error, r2_score
from sklearn.svm import SVR
from sklearn.utils import column_or_1d

warnings.filterwarnings('ignore')
```

Figure 89. Import libraries

2. Next, we need to get the data for analysis

```
df = pd.read_csv("C:\\Users\\DNV\\OneDrive\\Desktop\\Data-Gold.csv",parse_dates=True,index_col=0)
```

Figure 90. Import data

3. To be able to predict by time series we need to create an additional column "Timestamp"

```
df['Timestamp'] = pd.to_datetime(df.index).astype(np.int64) / 10**9
df_index = df.index
df_open = df['VND'].values.reshape(-1, 1)
df.head()

df
```

	VND	Timestamp
Date		
2012-01-02	32202288.50	1.325462e+09
2012-01-03	33607538.00	1.325549e+09
2012-01-04	33923003.00	1.325635e+09
2012-01-05	33628569.00	1.325722e+09
2012-01-06	34000652.75	1.325808e+09
...	...	...
2022-12-26	41586906.00	1.672013e+09
2022-12-27	41894838.00	1.672099e+09
2022-12-28	41727084.00	1.672186e+09
2022-12-29	41961480.00	1.672272e+09
2022-12-30	42138426.00	1.672358e+09

2870 rows × 2 columns

Figure 91. Add 'Timestamp'

4. Draw graphs to visualize input data

```
formatter = ticker.StrMethodFormatter('VND{x:,.0f}')

title = 'Gold Price'
ylabel = 'VND'
xlabel = 'Date'

ax = df['VND'].plot(figsize=(16, 9), title=title)
ax.autoscale(axis='x', tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter)
ax.grid(True)
```

Figure 92. Visualize data

5. **Then we start normalizing / scaling the input data with the StandardScaler() method.**

```
LR_sc = StandardScaler()

df_scaled = df.copy()
df_scaled['VND'] = LR_sc.fit_transform(df_open)
df_scaled
```

```
X_sc = StandardScaler()
y_sc = StandardScaler()

X = df.iloc[:, 1].values.reshape(-1, 1)
y = df.iloc[:, 0].values.reshape(-1, 1)


X_scaled = X_sc.fit_transform(X)
y_scaled = y_sc.fit_transform(y)


df_scaled = pd.DataFrame(index=df_index)
df_scaled['Timestamp'] = X_scaled
df_scaled['VND'] = y_scaled
df_scaled.head()
```

Figure 93. Scale data

6. **To have data for train and test, we need to divide the normalized data set into 2 fractions with a ratio of 9/1 corresponding to train and test.**

```
interrupt = int(len(df_scaled) * .9)

train_data, test_data = df_scaled[:interrupt], df_scaled[interrupt:]
index_test = df_scaled.index[interrupt:]
print(train_data.shape)
print(test_data.shape)
```

```
(2583, 2)
(287, 2)
```

Figure 94. Split data

```
plt.figure(figsize=(16, 9))
plt.grid(True)
plt.ylabel('Gold Prices')
plt.plot(train_data['VND'], 'blue', label='Train data')
plt.plot(test_data['VND'], 'green', label='Test data')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1a92df22fe0>
```

Figure 95. Visualize data

7. Next, we create auxiliary functions to help find suitable parameters for the model

The function that creates Hyperparameters

```
def para_range(minimum, maximum, step):
    para_list = np.arange(minimum, maximum, step)
    return para_list
```

The function that outputs the results of each case in GridSearch

```
def print_stats(hyperparams, stats):
    if(hyperparams[0] == 'poly'):
        print(
            f'SVR(kernel={hyperparams[0]}, C={hyperparams[1]}, gamma={hyperparams[2]}, degree={hyperparams[3]}')
    else:
        print(
            f'SVR(kernel={hyperparams[0]}, C={hyperparams[1]}, gamma={hyperparams[2]}')

    print(
        f"MAE={stats[0]:.2f} | MAPE={stats[1]:.2f} | MSE={stats[2] :.2f} | RMSE={stats[3]:.2f}")
```

Figure 96. Find parameters

```
warnings.filterwarnings('ignore')

stats_df = pd.DataFrame(
    columns=['kernel', 'C', 'gamma', 'degree', 'MAE', 'MAPE', 'MSE', 'RMSE'])

# GridSearch to find suitable hyperparameters
for ker in kernels:
    # If that is = = poly, make a choice degree.
    if(ker != 'poly'):
        for C in Cs:
            for gamma in gammas:
                rgs = SVR(kernel=ker, C=C, gamma=gamma, verbose=False)
                rgs.fit(X_train, y_train)
                pred = rgs.predict(X_test)

                # Transform back to original form
                inv_pred = y_sc.inverse_transform(
                    column_or_1d(pred).reshape(-1, 1))
                inv_test = y_sc.inverse_transform(
                    column_or_1d(y_test).reshape(-1, 1))

                # Model Evaluation
                mae = mean_absolute_error(inv_test, inv_pred)
                mape = mean_absolute_percentage_error(inv_test, inv_pred)
                mse = mean_squared_error(inv_test, inv_pred)
                rmse = np.sqrt(mse)

                result = {'kernel': ker, 'C': C, 'gamma': gamma, 'degree': 0,
                          'MAE': mae, 'MAPE': mape, 'MSE': mse, 'RMSE': rmse}

                hyperparam = [ker, C, gamma, 0]
                stats = [mae, mape, mse, rmse]
                stats_df = stats_df.append(result, ignore_index=True)
                print_stats(hyperparam, stats)
```

Figure 97. Code find parameters

```
else:
    for C in Cs:
        for gamma in gammas:
            for deg in degrees:
                rgs = SVR(kernel=ker, C=C, gamma=gamma, degree=deg, verbose=False)
                rgs.fit(X_train, y_train.reshape(-1, 1))
                pred = rgs.predict(X_test)

                # Transform back to original form
                inv_pred = y_sc.inverse_transform(
                    column_or_1d(pred).reshape(-1, 1))
                inv_test = y_sc.inverse_transform(
                    column_or_1d(y_test).reshape(-1, 1))

                # Model Evaluation
                mae = mean_absolute_error(inv_test, inv_pred)
                mape = mean_absolute_percentage_error(inv_test, inv_pred)
                mse = mean_squared_error(inv_test, inv_pred)
                rmse = np.sqrt(mse)

                result = {'kernel': ker, 'C': C, 'gamma': gamma, 'degree': deg,
                        'MAE': mae, 'MAPE': mape, 'MSE': mse, 'RMSE': rmse}

                hyperparam = [ker, C, gamma, deg]
                stats = [mae, mape, mse, rmse]

                stats_df = stats_df.append(result, ignore_index=True)

                print_stats(hyperparam, stats)
```

Figure 98. Code find parameters

8. Initialize values for the parameter finding process

Generate values for the hyperparameters

```
kernels = ['sigmoid', 'rbf', 'poly']
Cs = para_range(0.01, 0.2, 0.1)
gammas = para_range(0.1, 5, 0.1)
degrees = para_range(1, 6, 1)
```

Figure 99. Generate values

9. From the above results, we use to write a function to get the most optimal parameters for the model

```
best_fit_model = stats_df[stats_df['MAPE'] == stats_df['MAPE'].min()].head(1)
best_fit_model
```

	kernel	C	gamma	degree	MAE	MAPE	MSE	RMSE
155	rbf	0.11	0.9	0	1.428046e+06	0.034203	2.871548e+12	1.694564e+06

Figure 100. Find best fit model

10.Train and run the prediction results

```
kernel = str(best_fit_model['kernel'].values[0])
C = float(best_fit_model['C'])
gamma = float(best_fit_model['gamma'])
degree = int(best_fit_model['degree'])

if(kernel == 'poly'):
    rgs = SVR(kernel=kernel, C=C, gamma=gamma, degree=degree)
else:
    rgs = SVR(kernel=kernel, C=C, gamma=gamma)

rgs.fit(X_train, y_train.reshape(-1, 1))
pred = rgs.predict(X_test)
pred
```

Figure 101. Build model

11.After training and predicting the value, we return the data to its original form.

```
inv_pred = y_sc.inverse_transform(
    column_or_1d(pred).reshape(-1, 1))
inv_test = y_sc.inverse_transform(
    column_or_1d(y_test).reshape(-1, 1))
```

Figure 102. Inverse model

12.Plot graphs to see predicted results against test data.

```
plt.figure(figsize=(16, 9))
plt.grid(True)
plt.ylabel('Gold Prices')
plt.plot(column_or_1d(inv_test), 'blue', label='Actual data')
plt.plot(column_or_1d(inv_pred), 'red', label='Predicted data')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1a92dfeab90>
```

Figure 103. Visualize predictive data

13. Finally, we evaluate the model through MAE, MAPE, MSE, RMSE and R2 indexes.

```
from sklearn.metrics import r2_score
mae = mean_absolute_error(inv_test, inv_pred)
mape = mean_absolute_percentage_error(inv_test, inv_pred)
mse = mean_squared_error(inv_test, inv_pred)
rmse = np.sqrt(mse)
r2 = r2_score(inv_test, inv_pred)
print(f"MAE: {mae:.2f}")
print(f"MAPE: {mape*100:.2f}%")
print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R2: {r2:.2f}")
```

```
MAE: 1428046.38
MAPE: 3.42%
MSE: 2871548085287.00
RMSE: 1694564.28
R2: 0.22
```

Figure 104. Calculate MAPE,RMSE

14. Predicting next 30 days price

```
df1 = pd.read_csv("C:\\Users\\DNV\\OneDrive\\Desktop\\next30days.csv",parse_dates=True,index_col=0)
```

```
df1['Timestamp'] = pd.to_datetime(df1.index).astype(np.int64) / 10**9
df_index1 = df1.index
df1
```

...

```
X_sc = StandardScaler()
X = df1.iloc[:, 1].values.reshape(-1, 1)
X_scaled = X_sc.fit_transform(X)
df_scaled_future = pd.DataFrame(index=df_index1)
df_scaled_future['Timestamp'] = X_scaled
df_scaled_future.head()
```

...

```
X_future = df_scaled_future['Timestamp'].values.reshape(-1, 1)
```

```
pred_future = rgs.predict(X_future)
```

```
inv_pred_future = y_sc.inverse_transform(pred_future.reshape(-1, 1))
```

```
df_pred1 = pd.DataFrame(columns=['Pred'], index=df_index1)
df_pred1['Pred'] = column_or_1d(inv_pred_future)
df_pred1.head()
```

...

```
df_pred1.to_csv(r"C:\\Users\\DNV\\OneDrive\\Desktop\\SVR30d.csv")
```

Figure 105. Predict the next 30 days

V. CONCLUSION AND DISCUSSION

The following table are the experimental results of six models for training and testing we got it:

Table 6. Measuring the six models according to the past values

Model	RMSE	MAPE
LR	4456279.83	9.33%
SVR	1694564.28	3.42%
ARIMA	2074521.9	3.94%
PROPHET	1573796.94	2.99%
LSTM	404883	0.74%
BI-LSTM	397070	0.73%

After analysis of six models of time series forecasting and its corresponding RMSE and MAPE. This can be easily verified that the Recurrent Neural Network (RNN) based Bidirectional - Long Short-Term Memory (Bi - LSTM) Model-design gives the best forecast on test data with RMSE error of 397070 and MAPE Error of 0.73%.

Table 7. Compare actual price and predicted price check the accuracy of 6 models

Date	Actual	Bi-LSTM	LSTM	ARIMA	PROPHET	SVR	Linear
12/31/2022	**41927010**	42069188.21	42054234.06	42129308	37603478.8	35288112	26603497
1/1/2023	**41927010**	42004603.29	41944918.55	42121773	37967170.6	34008069	26963968
1/2/2023	**41908626**	41921593.08	41826965.05	42115622	39736599.1	32536728	27324439

Table 8. Calculate percentage error of 6 models

Date	Percentage Error (%)					
	Bi-LSTM	LSTM	ARIMA	PROPHET	SVR	Linear
12/31/2022	-0.339	-0.303	-0.483	10.312	15.834	36.548
1/1/2023	-0.185	-0.043	-0.465	9.445	18.887	35.688
1/2/2023	-0.031	0.195	-0.494	5.183	22.363	34.8

After collecting the actual price and getting the forecast price, we calculate the percentage error, we see that the lowest percentage error belongs to the Bi-LSTM model and the highest percentage error belongs to the Linear Regression model. So the most optimal model is the Bi-LSTM model and the least optimal model is the Linear Regression model.

VI. WORK ASSIGNMENT

Members / Work	Phạm Thành Đạt (Leader)	Thiều Huy Hoàng	Nguyễn Quang Vy
Linear Regression			x
SVR			x
ARIMA		x	
FBPROPHET		x	
LSTM	x		
Bi-LSTM	x		

Reference:

[1]: **Global Gold price - Historical Data (1979-Present)**

[2], [3]: Chicco, D., Warrens, M.J. and Jurman, G., 2021. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Computer Science*, *7*, p.e623.

[4]: **Autoregressive Integrated Moving Average - ARIMA**

[5]: Nguyễn Minh Nhựt – LAB4_PhanTichDuLieuChuoiThoiGian.

[6]: Data Science - Autoregressive Integrated Moving Average - ARIMA

[7]: Facebook Prophet Model - Time-series Analysis

[9]: Advanced Recurrent Neural Networks

[10]: Univariate Time-series with BiLSTM

[11]: Cai, Changchun, Yuan Tao, Tianqi Zhu, and Zhixiang Deng. 2021. "Short-Term Load Forecasting Based on Deep Learning Bidirectional LSTM Neural Network" *Applied Sciences* 11, no. 17: 8129. https://doi.org/10.3390/app11178129

[12]: https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-your-first-machine-learning-model-linear-regression/

[13]: https://www.saedsayad.com/support_vector_machine_reg.htm

[14]: Support Vector Regression and it’s Mathematical Implementation | by Priyanka Parashar | The Startup | Medium

❖ Reference Of Related Work:

[1]. Risse, M., (2019) Combining wavelet decomposition with machine learning to forecast gold returns. International Journal of Forecasting, [online] 352, pp.601–615. Available at: https://doi.org/10.1016/j.ijforecast.2018.11.008.

[2].Vidya, G.S. and Hari, V.S., (2020) Gold Price Prediction and Modelling using Deep Learning Techniques. 2020 IEEE Recent Advances in Intelligent Computational Systems, RAICS 2020, pp.28–31.

[3]. Tripathy, N., (2017) Forecasting Gold Price with Auto-Regressive Integrated Moving Average Model. International Journal of Economics and Financial Issues, 74, pp.324–329.

[4]. Guha, B., & Bandyopadhyay, G. (2016). Gold price forecasting using ARIMA model. *Journal of Advanced Management Science*, *4*(2).

[5] .Yang, X. (2019, January). The prediction of gold price using ARIMA model. In *2nd Conference on Social Science, Public Health and Education (SSPHE 2018)* (pp. 273-276). Atlantis Press.

[6]. He, Z., Zhou, J., Dai, H. N., & Wang, H. (2019, August). Gold price forecast based on LSTM-CNN model. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)* (pp. 1046-1053). IEEE.