VNUK Institute for Research & Executive Education

Artificial Intelligence

::Challenge 2 - Time Series Forecasting (10%)

Due: 24 Nov, 11:59 PM, Friday

Email to: vu.tran@vnuk.edu.vn

According to the VNUK *Academic integrity policy*, plagiarism is:

"Claiming and using the thoughts or writings or creative works of others without appropriate acknowledgement or attribution. It includes:
(a) copying part or all of another student's assignment;
(b) allowing another person to write some or all of an assignment;
(c) copying paragraphs, sentences or parts of sentences directly from texts or the internet without enclosing them in quotation marks or otherwise showing them to be copied - even if the source is acknowledged, this is still plagiarism;
(d) using concepts or developed ideas, even if paraphrased or summarised, from another person, from texts or the internet without acknowledging the source;
(e) copying graphics, architectural plans, multimedia works or other forms of intellectual property without appropriate acknowledgment."

The consequences of plagiarism (depending on the seriousness of the case) range from reducing your mark or failing the assignment up to a formal reference to a summary inquiry:

By signing below I certify that the attached assignment is my own work.

Student ID: 21020006 Student Name: Nguyen Tran Xuan Tri Signature:

Grade:

No.	Question	Grade
1	Question 1	
2	Question 2	
3	Question 3	
4	Question 4	
5	Question 5	
	Total gold coins	

This problem set will introduce you to using control flow in Python and formulating a computational solution to a problem.

Data

- You are free to choose or crawl data that could use the time series forecasting method. For example: finance, economics, sales

Requirements:

No.	Criteria	Weight (%)
1	Train the model	20%
2	Deploy the model	30%
3	Explain the math/model	15%
4	Complete app	15%
5	Git usage	10%

1. **Developing the model**
   Developing a time series forecasting model involves predicting future values based on historical time-ordered data. Time series forecasting is widely used in various fields, such as finance, economics, sales, and weather prediction.

   A Recurrent Neural Network (RNN) is a type of artificial neural network designed for sequence data and tasks. Unlike traditional feedforward neural networks, which process inputs in a single pass, RNNs have connections that form directed cycles, allowing them to maintain a hidden state that captures information about previous inputs in the sequence.

   RNNs and their variants have been widely used in various applications, including:

   - **Natural Language Processing (NLP):** RNNs are used for tasks such as language modeling, machine translation, and sentiment analysis.

- **Time Series Prediction:** RNNs can be applied to predict future values in time series data, such as stock prices or weather conditions.
- **Speech Recognition:** RNNs are used to recognize and transcribe spoken language.
- **Video Analysis:** RNNs can be applied to tasks like action recognition and video captioning.

In the context of time series prediction, several types of recurrent neural networks (RNNs) and their variants can be used. Here are some commonly used types:

1. **Vanilla RNNs (Simple RNNs):** The basic form of recurrent neural networks that maintain hidden states to capture information from previous time steps. However, they suffer from the vanishing gradient problem, limiting their ability to capture long-range dependencies.
2. **Long Short-Term Memory (LSTM):** LSTM networks address the vanishing gradient problem by introducing specialized memory cells and gating mechanisms. LSTMs can effectively capture and remember long-term dependencies in time series data.
3. **Gated Recurrent Unit (GRU):** Similar to LSTMs, GRUs are designed to address the vanishing gradient problem. They use a simpler architecture with fewer parameters compared to LSTMs, making them computationally more efficient in some cases.
4. **Bidirectional RNNs:** Bidirectional RNNs process the input sequence in both forward and backward directions, allowing the network to capture information from both past and future time steps. This can be beneficial in tasks where future context is important for predictions.
5. **Echo State Network (ESN):** ESN is a type of reservoir computing that simplifies the training of recurrent neural networks. It has fixed random connections between neurons, and only the readout layer is trained. ESNs have been used in time series prediction tasks.
6. **Clockwork RNN:** Clockwork RNN introduces different time scales for different neurons, allowing some neurons to update their states more frequently than others. This can be useful in capturing patterns with varying time scales in time series data.
7. **Attention Mechanisms:** While not a type of RNN per se, attention mechanisms have been integrated with RNNs to allow the model to focus on specific parts of the input sequence when making predictions. This is particularly useful for handling long sequences.
8. **Transformers:** Though initially designed for natural language processing tasks, Transformers have gained popularity in time series forecasting. They use a self-attention mechanism that enables capturing long-range dependencies efficiently.

Reference:

https://www.kaggle.com/code/meetnagadia/bitcoin-price-prediction-using-lstm
https://github.com/Ali619/Bitcoin-Price-Prediction-LSTM/blob/master/Bitcoin_Price_Prediction.ipynb

2. Developing a website for a model

Developing a website for a machine learning model involves several steps, including designing the user interface, creating the back-end to serve predictions

Choose at least 2 types of cryptocurrencies:

1. **Bitcoin (BTC):** The first and most well-known cryptocurrency, often referred to as digital gold.
2. **Ethereum (ETH):** Known for its smart contract functionality, allowing developers to build decentralized applications (DApps) on its blockchain.
3. **Binance Coin (BNB):** Originally created as a utility token for the Binance exchange, BNB has expanded its use cases and is used in various applications.
4. **Ripple (XRP):** Designed for facilitating fast and low-cost international money transfers.
5. **Litecoin (LTC):** Created as the "silver to Bitcoin's gold," Litecoin is known for its faster block generation time.
6. **Cardano (ADA):** A blockchain platform known for its focus on security and scalability.
7. **Polkadot (DOT):** A multi-chain network that enables different blockchains to transfer messages and value in a trust-free fashion.
8. **Chainlink (LINK):** A decentralized oracle network that enables smart contracts to interact with real-world data.
9. **Stellar (XLM):** A platform designed to facilitate fast, low-cost cross-border payments.
10. **Dogecoin (DOGE):** Originally created as a meme, Dogecoin gained popularity and is known for its active community.
11. **Uniswap (UNI):** A decentralized exchange (DEX) token on the Ethereum blockchain.
12. **Solana (SOL):** A high-performance blockchain known for its fast transaction speeds.
13. **Bitcoin Cash (BCH):** A fork of Bitcoin, designed to offer faster and cheaper transactions.
14. **VeChain (VET):** Focused on supply chain management and business processes.
15. **Polygon (MATIC):** A Layer 2 scaling solution for Ethereum to improve transaction speeds and reduce fees.
16. **EOS (EOS):** A blockchain platform designed for decentralized applications and smart contracts.
17. **Tezos (XTZ):** A blockchain that uses on-chain governance to evolve its protocol.
18. **Tron (TRX):** A platform for decentralized applications and entertainment content.

19. **Filecoin (FIL):** A decentralized storage network that allows users to rent out their excess storage space.
20. **Aave (AAVE):** A decentralized finance (DeFi) protocol for lending and borrowing.

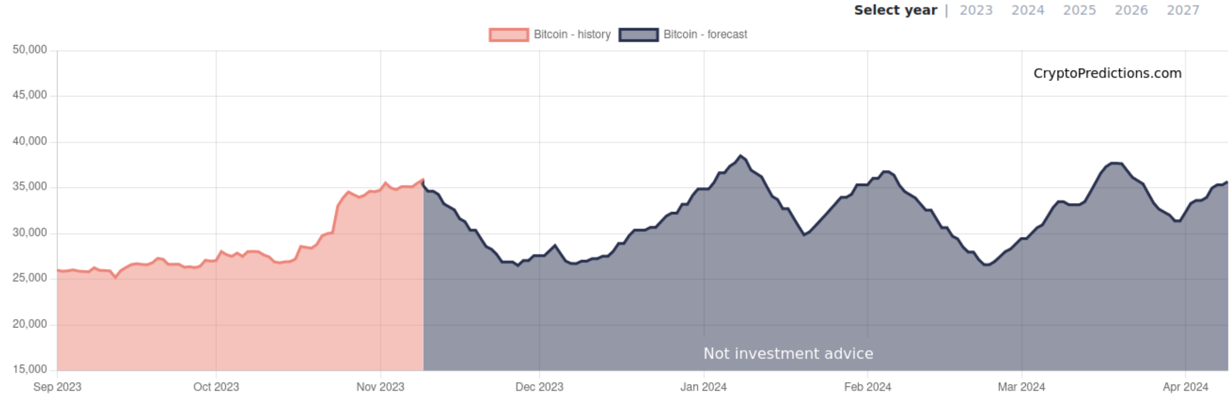
Below a example of Bitcoinn Prediction

#	Name		Price	24h %	Market Cap	Volume(24h)	7d Forecasts	1y Predictions
1	Bitcoin BTC	Buy	$36,299	2.75%	$709,914,305,892	$19,026,395,755 524,157.573 BTC	-2.69%	
2	Ethereum ETH	Buy	$1,912.970	1.33%	$230,518,027,572	$13,767,613,393 7,196,983.430 ETH	-1.72%	
3	Tether USDT	Buy	$1	-0.02%	$86,284,401,207	$33,043,791,701 33,043,791,701 USDT	0.00%	
4	BNB BNB	Buy	$248.520	0.90%	$38,283,318,827	$564,471,515 2,271,332.347 BNB	-0.92%	
5	XRP XRP	Buy	$0.68955	0.48%	$37,026,844,283	$1,566,901,142 2,272,353,189.761 XRP	-0.75%	

Bitcoin BTC $36,299 Buy BTC Trade now

24H | **$972.2487** (2.75%) | **Market Cap:** $709,914,305,892 | **24hr Volume:** $19,026,395,755

Select year | 2023 2024 2025 2026 2027



Debriefing Report :: Part 1

Part 1. Report on the challenge.

Load the data:

In this challenge, I chose BNB-USD (BNB) and Emeren Group Ltd (SOL) for the challenge.

Using finance to download data frames of the cryptocurrencies.

For BNB, I chose the range from 01/01/2021 to 25/11/2023, and a duration of 15 years for SOL.

```
import yfinance as yf

start = '2021-01-01'
end = '2023-11-25'
period= '5y'
bnb = yf.download("BNB-USD", start, end)
bnb.head()
```

```
[*********************100%%**********************]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2021-01-01	37.374573	38.928177	37.046307	37.905010	37.905010	459165743
2021-01-02	37.917107	38.836254	36.925602	38.241592	38.241592	521965394
2021-01-03	38.253727	41.606323	37.818104	41.148979	41.148979	758008613
2021-01-04	41.198280	43.132122	38.143982	40.926353	40.926353	807877171
2021-01-05	40.937279	41.734600	38.978954	41.734600	41.734600	644270927

```
sol = yf.download(tickers=['SOL'], period='15y')
sol.head()
```

```
[*********************100%%**********************]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2008-11-25	14.90	15.600000	12.75	14.75	14.75	413100
2008-11-26	13.65	17.700001	13.60	15.95	15.95	753680
2008-11-28	16.10	18.250000	16.10	17.35	17.35	284100
2008-12-01	16.10	16.100000	13.55	13.60	13.60	349420
2008-12-02	14.50	16.000000	13.70	15.65	15.65	297100

Explore the data:

Using .info() and .describe()

```
bnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1058 entries, 2021-01-01 to 2023-11-24
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       1058 non-null   float64
 1   High       1058 non-null   float64
 2   Low        1058 non-null   float64
 3   Close      1058 non-null   float64
 4   Adj Close  1058 non-null   float64
 5   Volume     1058 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 57.9 KB
```

```
len(bnb)
```

```
1058
```

```
bnb.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	1058.000000	1058.000000	1058.000000	1058.000000	1058.000000	1.058000e+03
mean	325.344776	334.721835	315.226615	325.476360	325.476360	1.656161e+09

```
sol.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3775 entries, 2008-11-25 to 2023-11-24
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       3775 non-null   float64
 1   High       3775 non-null   float64
 2   Low        3775 non-null   float64
 3   Close      3775 non-null   float64
 4   Adj Close  3775 non-null   float64
 5   Volume     3775 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 206.4 KB
```

```
len(sol)
```

```
3775
```

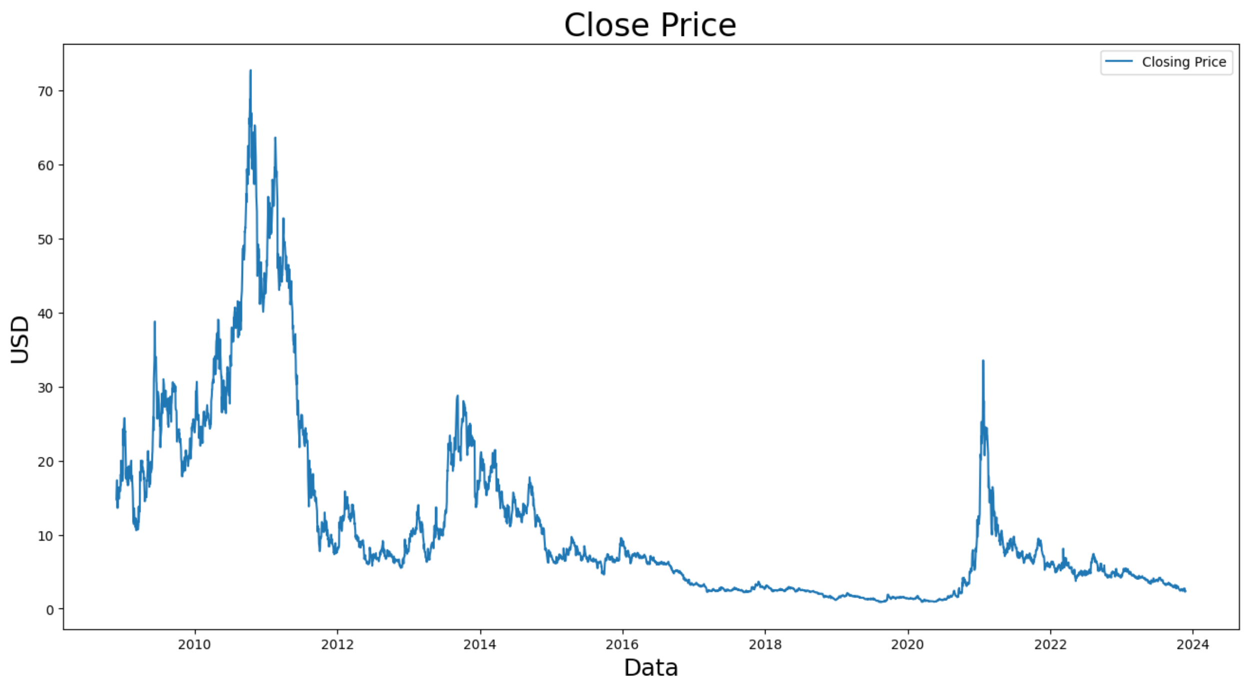
```
sol.describe()
```

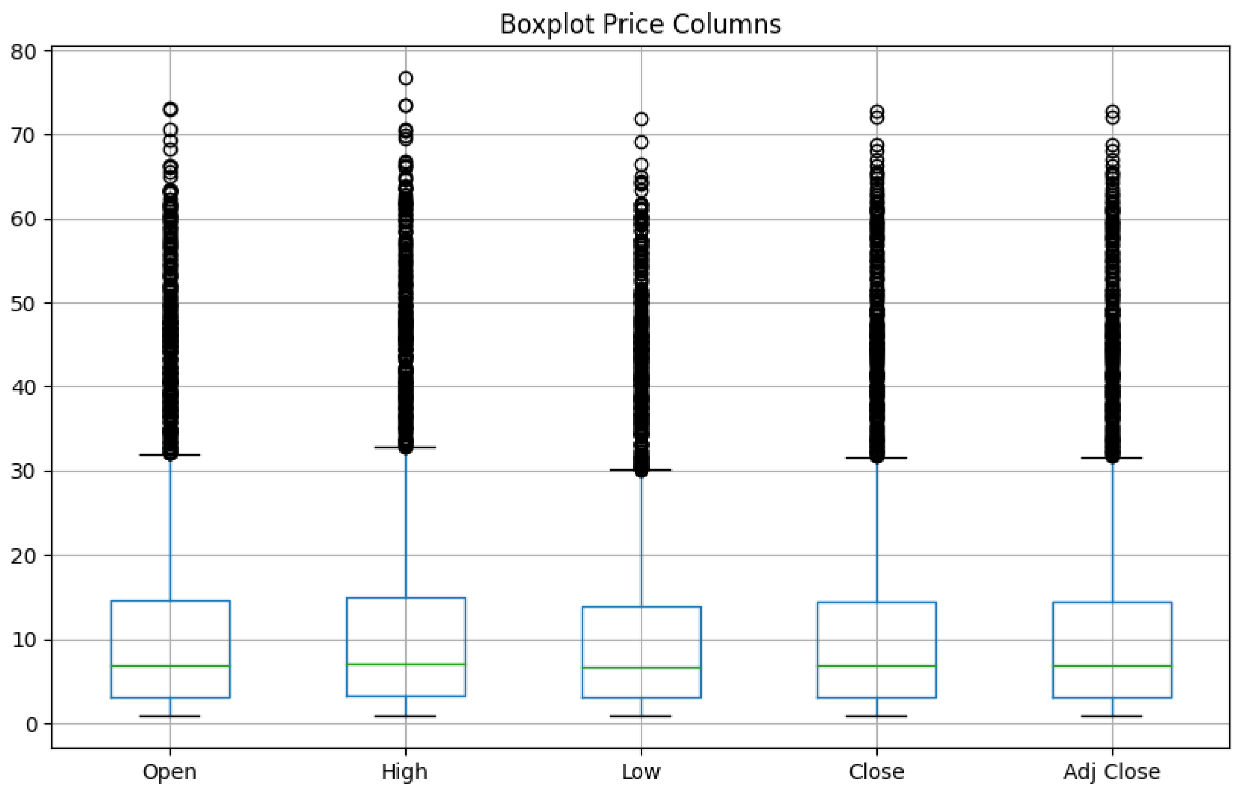
	Open	High	Low	Close	Adj Close	Volume
count	3775.000000	3775.000000	3775.000000	3775.000000	3775.000000	3.775000e+03
mean	11.394922	11.766005	11.000188	11.365764	11.365764	5.217399e+05
std	12.338297	12.674638	11.939748	12.293157	12.293157	1.039419e+06
min	0.850000	0.920000	0.850000	0.860000	0.860000	5.000000e+02

We can see that the data of BNB includes 1058 rows with six columns: open price, high price, low price, close price, adjusted close, and volume. While that of SOL has 3775 rows.

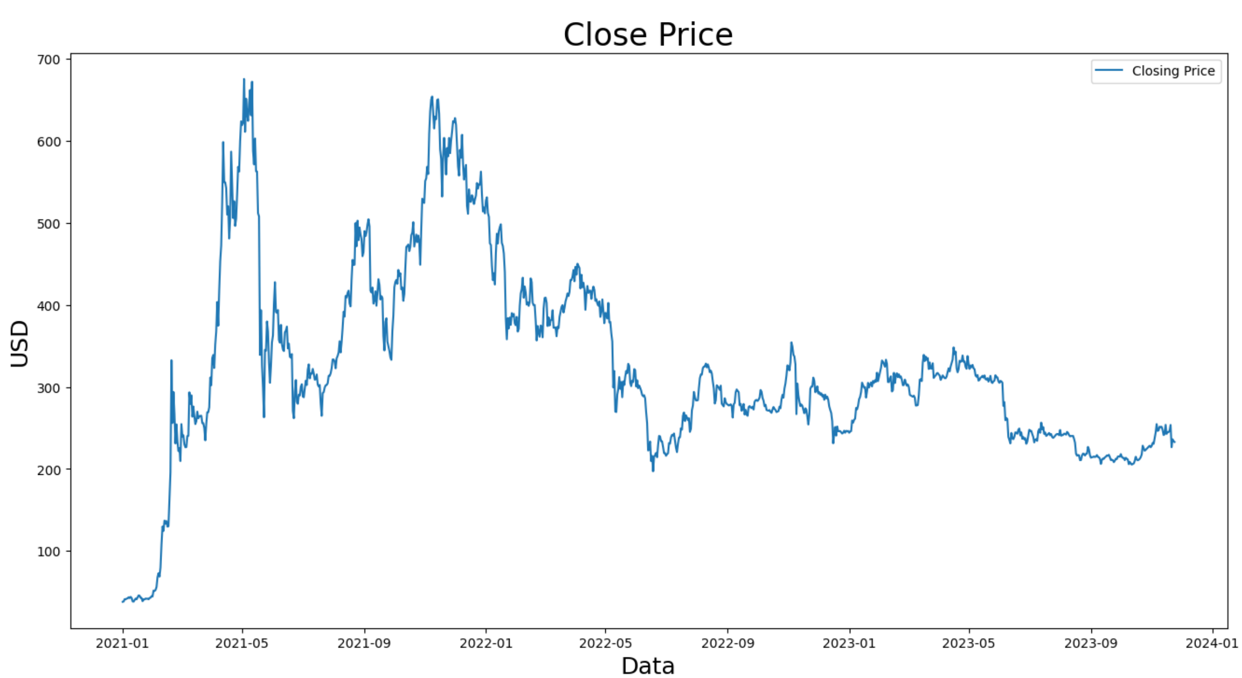
Visualize the data:

BNB's Historical Data:

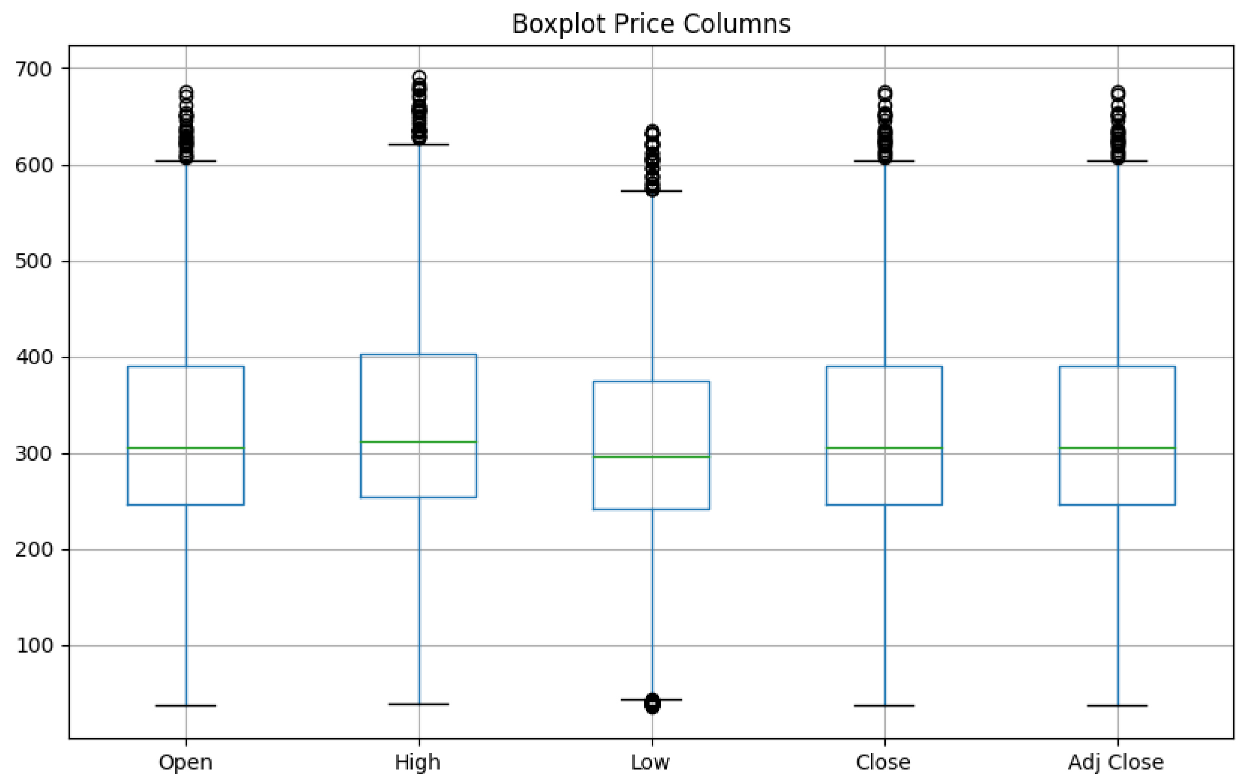




SOL’s Historical Data:



158A Le Loi, Danang, Vietnam
www.vnuk.edu.vn | contact@vnuk.edu.vn
Phone: +84 (236) 3 64 65 77


Boxplot Price Columns
700
600
500
400
300
200
100
Open
High
Low
Close
Adj Close

Checking the missing values:

Checking a DataFrame for missing values is critical for data quality and analysis integrity. Missing values in statistical analyses, machine learning models, or visualizations can result in biased or inaccurate results. Identifying and dealing with missing data correctly ensures that data-driven decisions and insights are based on a complete and reliable dataset, contributing to the overall robustness and reliability of data-driven processes.

Using .isnull().sum() to check how many null values the data has.

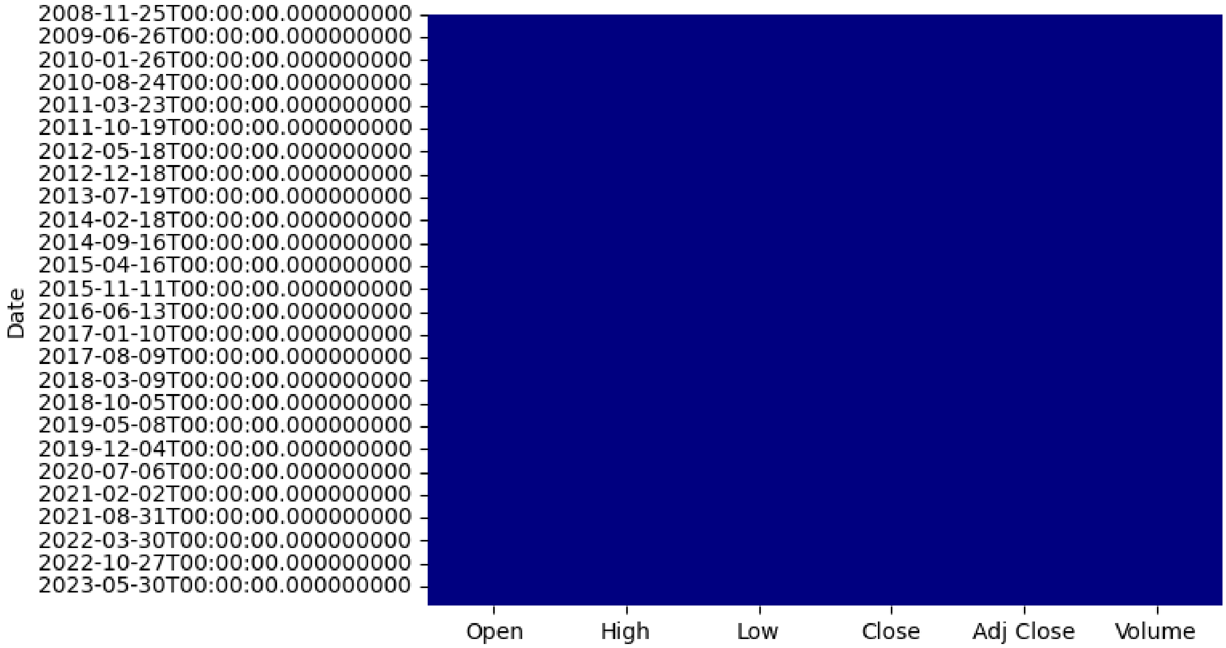
```
print("Missing values: ")
print(bnb.isnull().sum())
```

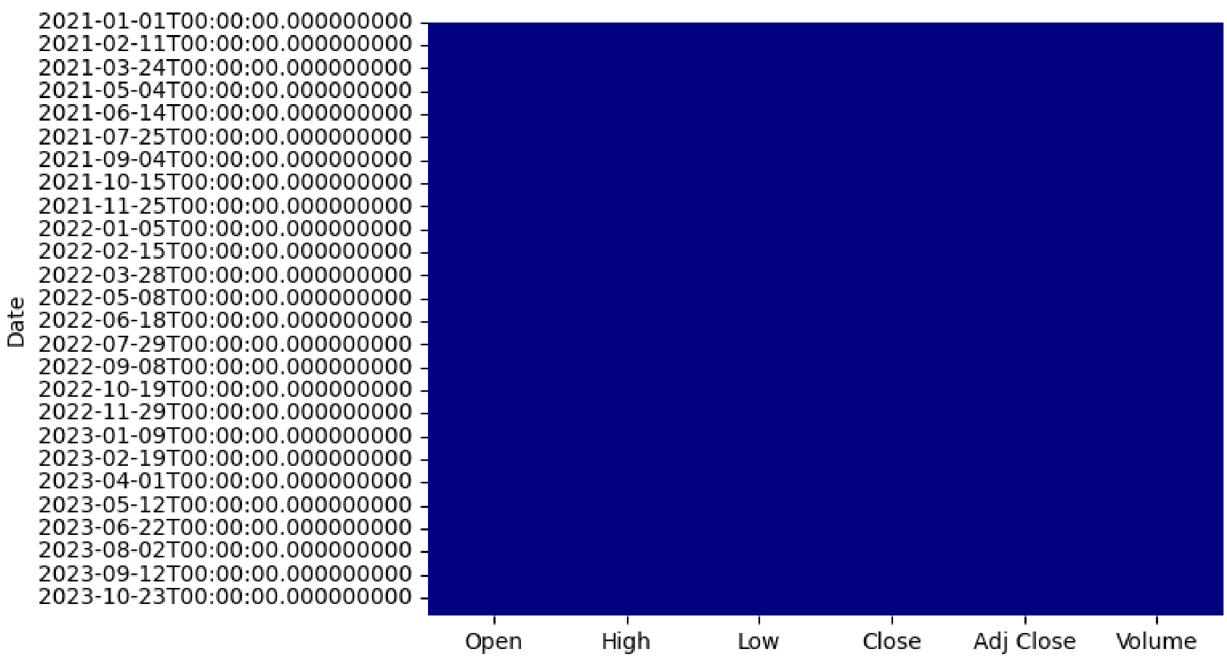
```
Missing values:
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

```
print("Missing values: ")
print(sol.isnull().sum())
```

```
Missing values:
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Also, using sns.heatmap() to check.





As we can see, both data don't have any missing values.

Preprocess the data:

For forecasting cryptocurrencies, we only need the close price column

and reshape the data.

```
#Creat a new dataframe with only Close Price
sol_data = sol.filter(['Close'])
#Convert the dataframe to numpy array
sol_dataset = sol_data.values.reshape(-1, 1)
print(sol_dataset)
# Get the number of rows to train the model on. we need this number to create our train and test sets
print("Data's length: ", len(sol_dataset))
```

After that, normalize the data:

1. Scaling: make sure all feature contribute equally

```
# Scale the data
scaler = MinMaxScaler(feature_range=(0,1))
sol_dataset = scaler.fit_transform(sol_dataset)
sol_dataset
```

```
array([[0.19321185],
       [0.20990402],
       [0.22937822],
       ...,
       [0.02448185],
       [0.0197524 ],
       [0.02086521]])
```

```
# Scale the data
scaler = MinMaxScaler(feature_range=(0,1))
bnb_dataset = scaler.fit_transform(bnb_dataset)
bnb_dataset
```

```
array([[0.        ],
       [0.00052774],
       [0.00508635],
       ...,
       [0.31081545],
       [0.30709819],
       [0.30574215]])
```

2. Making training dataset:

```
# Create the training dataset
train_data = sol_dataset[0:sol_training_data_len, :]

n_lookback = 120 # Input sequences
n_forecast = 60 # Prediction

# Split the data into X_train and y_train data sets
sol_X = []
sol_y = []

for i in range(n_lookback, len(train_data) - n_forecast + 1):
  sol_X.append(train_data[i - n_lookback: i])
  sol_y.append(train_data[i: i + n_forecast])

print(len(sol_X))
print(len(sol_y))
```

```
# math.ceil will round up the number
sol_training_data_len = math.ceil(len(sol_dataset) * .8) # We are using %80 of the data for training
sol_training_data_len
```

```
sol_training_size = int(bnb_X.shape[0] * 0.8)
sol_training_size
```

```
534
```

```
sol_X_train, sol_y_train = sol_X[:sol_training_size], sol_y[:sol_training_size]
sol_X_test, sol_y_test = sol_X[sol_training_size:], sol_y[sol_training_size: ]
```

After appending 80% of the dataset to training datasets, turn them into numpy arrays.

```
# Convert the X_train and y_train to numpy array
sol_X, sol_y = np.array(sol_X), np.array(sol_y)
```

```
print(sol_X.shape)
print(sol_y.shape)
```

```
(2841, 120, 1)
(2841, 60, 1)
```

```
# Convert the X_train and y_train to numpy array
bnb_X, bnb_y = np.array(bnb_X), np.array(bnb_y)
```

```
print(bnb_X.shape)
print(bnb_y.shape)
```

```
(668, 120, 1)
(668, 60, 1)
```

Make model:

Sequential data handling is critical when developing a time series prediction model for tasks such as stock market forecasting. Recurrent Neural Networks (RNNs) are an obvious choice for such scenarios because they take advantage of the data's sequential nature by feeding the output from the previous step into the current step. Long Short-Term Memory networks (LSTMs), a type of RNN, are preferred because of their ability to capture and remember long-term dependencies, making them ideal for modeling stock market data. LSTMs excel at handling sequential information, which is critical for forecasting stock prices, which are inherently dependent on past values. However, it is critical to recognize the stock market's inherent unpredictability, emphasizing the need for caution and acknowledging the risks and uncertainties associated with using any model in this domain.

I use Kera and TensorFlow to make a LTMS model:

Initializing a Sequential, allowing all layers stacked. Then add LTMS layers with ‘return_sequences’ = True, and subsequent LTMS layers. Then add a dense layer with n_forecast.

```
# Create the testing dataset
# Create a new array containing scaled values from index 2083
bnb_model = Sequential()
bnb_model.add(LSTM(units=50, return_sequences=True, input_shape=(n_lookback, 1)))
bnb_model.add(LSTM(units=50))
bnb_model.add(Dense(n_forecast))

# adam = Adam(learning_rate = 5e-3)
bnb_model.compile(loss='mean_squared_error', optimizer='adam')
```

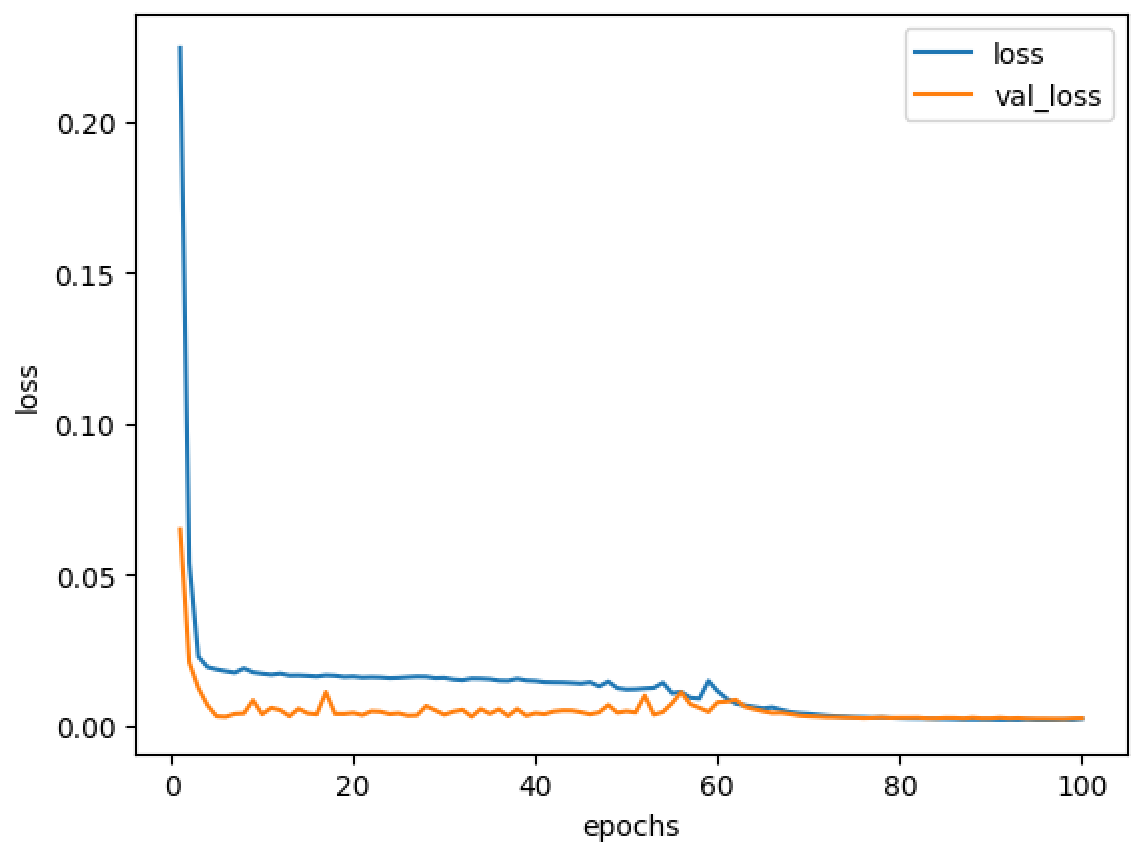
Train model:

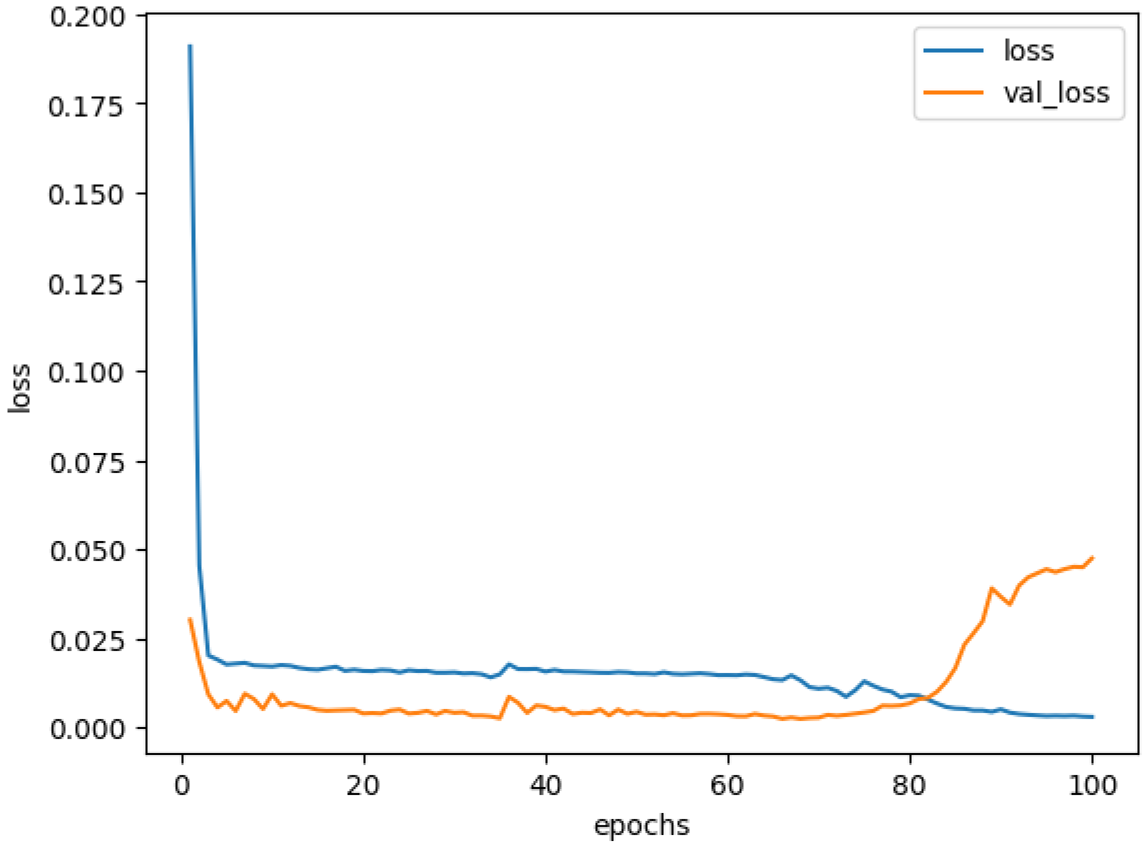
Train with epochs = 100 and batch size = 32.

```
history = sol_model.fit(sol_X_train, sol_y_train,
                        epochs = 100,
                        batch_size = 32,
                        validation_data = (sol_X_test, sol_y_test))
```

Then, plot the loss graph of the model.

```
#Plot the training
historyForPlot = pd.DataFrame(history.history)
historyForPlot.index += 1 # we plus 1 to the number of indexing so our epochs Plot picture will be counting from 1 not 0.
historyForPlot.plot()
plt.ylabel("loss")
plt.xlabel("epochs")
```



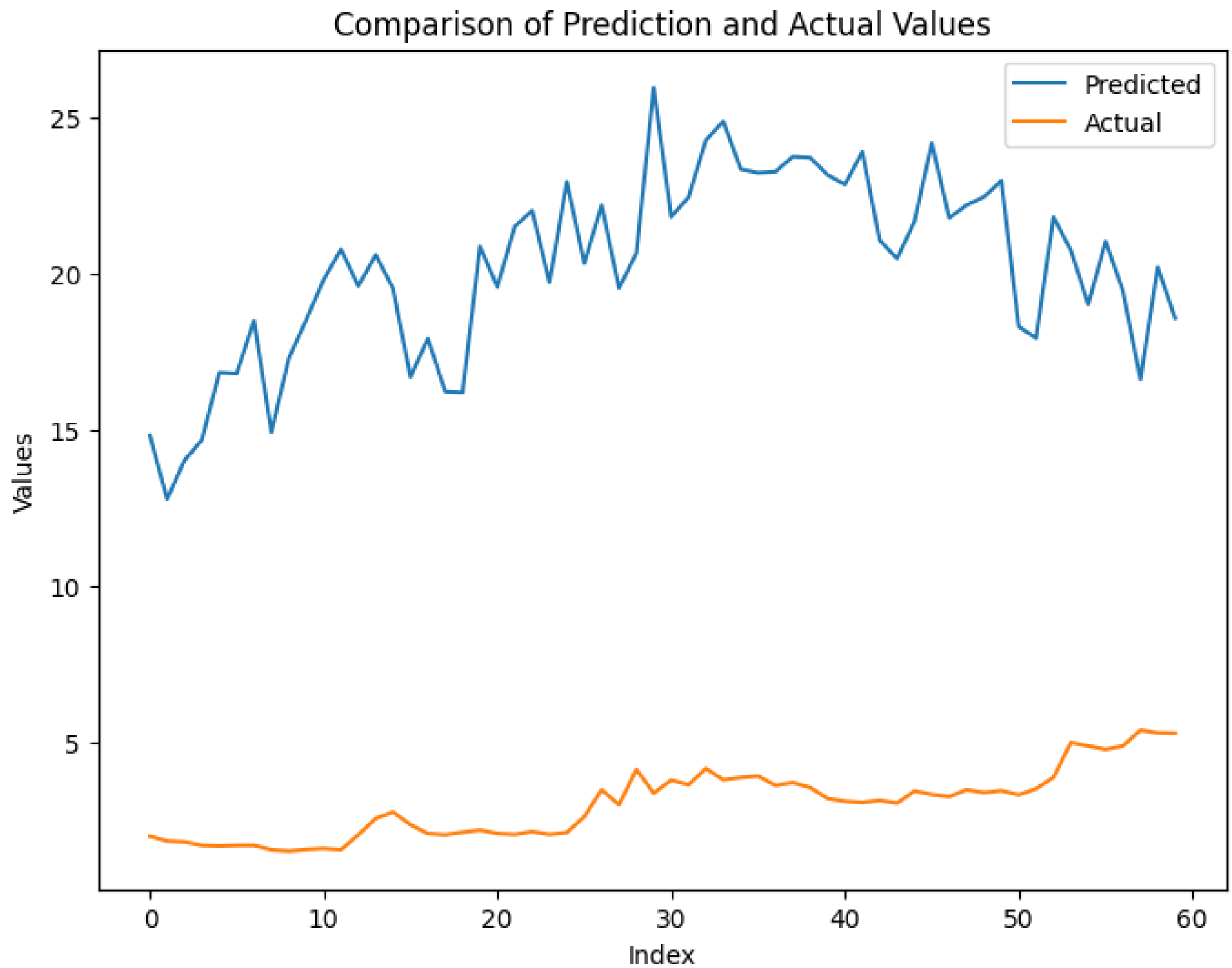

loss
val_loss
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
20
40
60
80
100
epochs

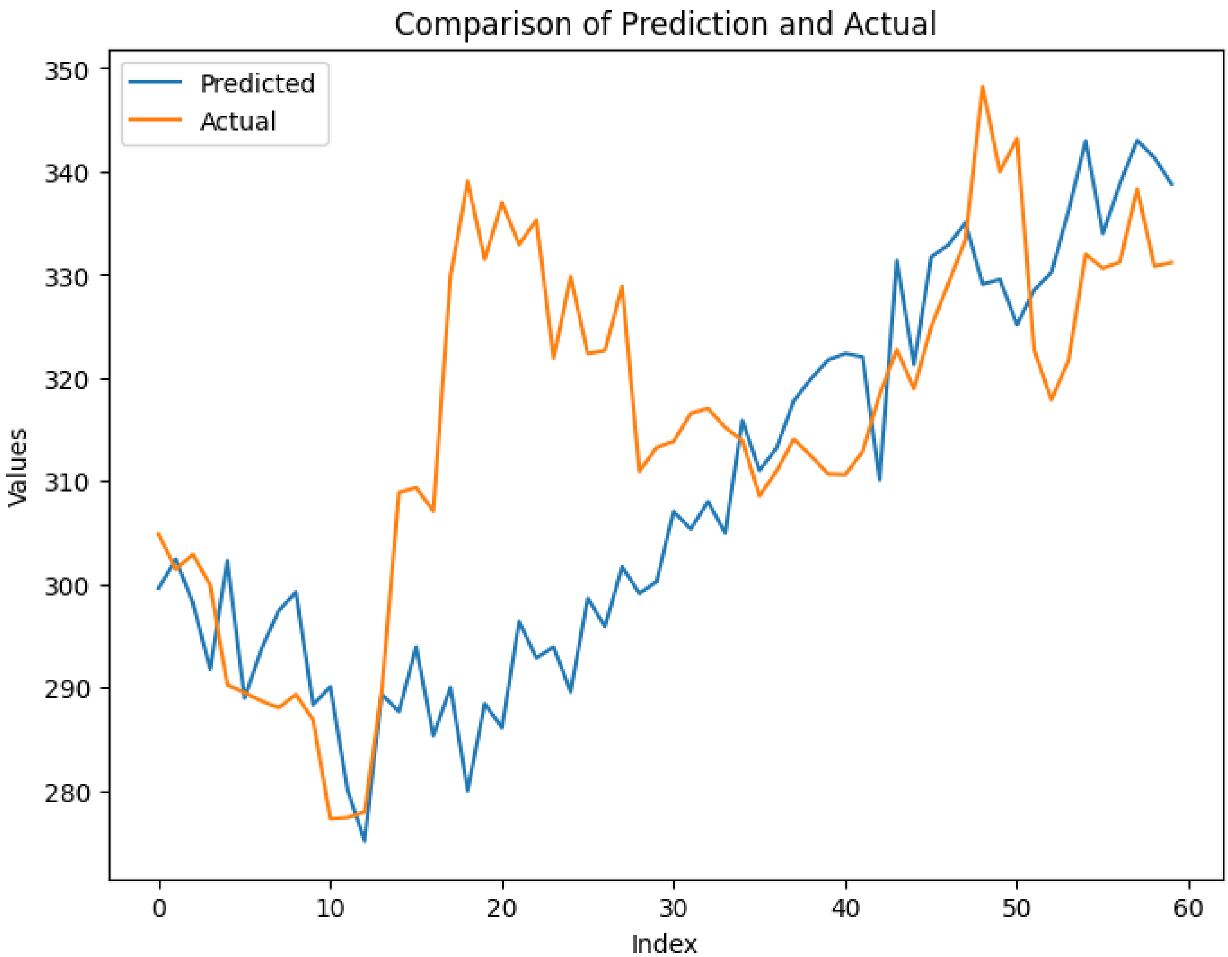
Test model:

```
predict = bnb_model.predict(bnb_X_test[-1 : : ])
predict = scaler.inverse_transform(predict)
predict
```

```
check_pred = sol_y_test[-1 : : ]
check_pred = check_pred.reshape(-1 , 1)
check_pred = scaler.inverse_transform(check_pred)
check_pred
```

```
import matplotlib.pyplot as plt
import numpy as np

predict = np.array(predict)  # Convert predict to numpy array if it's not already
check_pred = np.array(check_pred)  # Convert check_pred to numpy array if it's not already

plt.figure(figsize=(8, 6))
plt.plot(predict, label='Predicted')
plt.plot(check_pred, label='Actual')
plt.xlabel('Index')
plt.ylabel('Values')
plt.title('Comparison of Prediction and Actual Values')
plt.legend()
plt.show()
```


Comparison of Prediction and Actual Values
Predicted
Actual
Values
Index
0
10
20
30
40
50
60
5
10
15
20
25


Comparison of Prediction and Actual
Predicted
Actual
Values
Index
350
340
330
320
310
300
290
280
0
10
20
30
40
50
60

```
sol_lookback = sol_dataset[-n_lookback:]
sol_lookback = sol_lookback.reshape(1, n_lookback, 1)
```

```
sol_forecast = sol_model.predict(sol_lookback)
sol_forecast = scaler.inverse_transform(sol_forecast)
```

```
1/1 [==============================] - 0s 33ms/step
```

```
sol_forecast
```

```
array([[15.002086, 12.952719, 14.190477, 14.789086, 16.974154, 17.010696,
        18.598455, 15.133918, 17.372429, 18.639963, 19.817951, 20.817474,
        19.755726, 20.653421, 19.612455, 16.816292, 18.046354, 16.427767,
        16.38512 , 20.924765, 19.69618 , 21.51828 , 22.026415, 19.846916,
        23.036465, 20.43736 , 22.22751 , 19.648228, 20.699106, 25.912663,
        21.9023  , 22.452477, 24.292894, 24.887457, 23.400692, 23.241318,
        23.341732, 23.752913, 23.724508, 23.204384, 22.962889, 23.96491 ,
        21.191334, 20.620455, 21.728243, 24.240461, 21.8276  , 22.327091,
        22.478724, 23.038008, 18.473421, 18.11603 , 21.833527, 20.901827,
        19.099264, 21.066519, 19.546474, 16.81202 , 20.292513, 18.666483]],
      dtype=float32)
```

```
bnb_lookback = bnb_dataset[-n_lookback:]

bnb_lookback = bnb_lookback.reshape(1, n_lookback, 1)
```

```
bnb_forecast = bnb_model.predict(bnb_lookback)
bnb_forecast = scaler.inverse_transform(bnb_forecast)
```

```
1/1 [==============================] - 0s 35ms/step
```

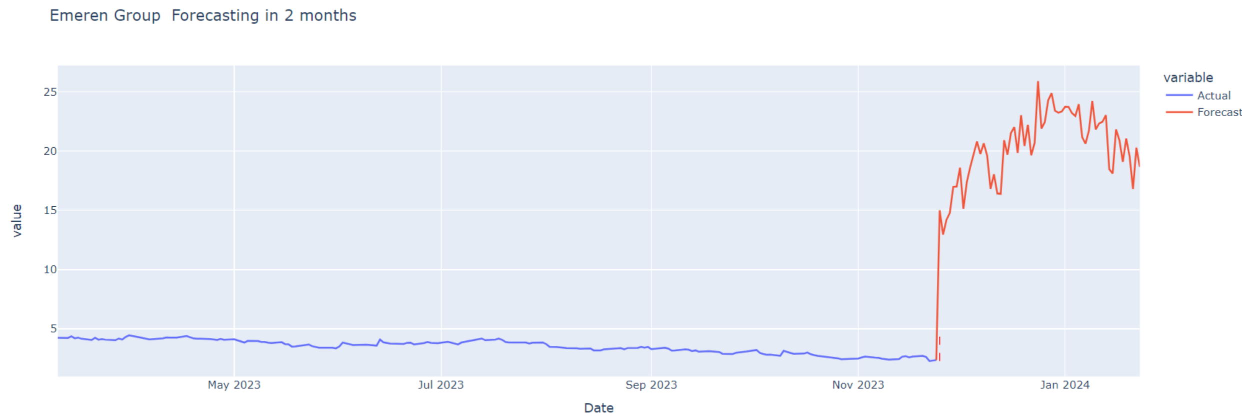
```
bnb_forecast
```

```
array([[240.08662, 238.69693, 236.50417, 239.8516 , 229.64127, 233.12392,
        234.41017, 235.93614, 241.42558, 237.96858, 235.93011, 228.71606,
        218.30359, 236.89218, 234.5356 , 236.63474, 232.32861, 236.63635,
        229.00868, 238.34532, 234.23245, 244.7664 , 232.3045 , 230.13853,
        230.31883, 239.77493, 228.30736, 235.22853, 230.23386, 232.2053 ,
        235.576  , 234.15662, 235.34723, 229.45982, 237.12527, 234.63853,
        228.001  , 228.9367 , 226.95013, 227.9263 , 230.98349, 226.14075,
        218.55931, 224.27531, 220.92628, 221.8687 , 218.02823, 222.4459 ,
        217.61823, 218.80713, 217.58792, 217.6105 , 217.04839, 219.91557,
        224.728  , 215.20862, 217.19765, 218.2891 , 221.69212, 211.7042 ]],
      dtype=float32)
```

Visualize the forecast:

```
sol_past = sol[['Close']][-180:].reset_index()
sol_past.rename(columns={'index': 'Date', 'Close': 'Actual'}, inplace=True)
sol_past['Date'] = pd.to_datetime(sol_past['Date'])
sol_past['Forecast'] = np.nan
sol_past['Forecast'].iloc[-1] = sol_past['Actual'].iloc[-1]


sol_future = pd.DataFrame(columns=['Date', 'Actual', 'Forecast'])
sol_future['Date'] = pd.date_range(start=sol_past['Date'].iloc[-1] + pd.Timedelta(days=1), periods=n_forecast)
sol_future['Forecast'] = sol_forecast.flatten()
sol_future['Actual'] = np.nan


results = pd.concat([sol_past, sol_future]).set_index('Date')


fig = px.line(results, x=results.index, y=['Actual', 'Forecast'], title='Emeren Group  Forecasting in 2 months')
fig.add_shape(
    go.layout.Shape(
        type="line",
        x0=results.index[-n_forecast], y0=results['Actual'].min(),
        x1=results.index[-n_forecast], y1=results['Actual'].max(),
        line=dict(color="red", width=1, dash="dash")
    )
)
fig.show()
```

```
bnb_past = bnb[['Close']][-180:].reset_index()
bnb_past.rename(columns={'index': 'Date', 'Close': 'Actual'}, inplace=True)
bnb_past['Date'] = pd.to_datetime(bnb_past['Date'])
bnb_past['Forecast'] = np.nan
bnb_past['Forecast'].iloc[-1] = bnb_past['Actual'].iloc[-1]


bnb_future = pd.DataFrame(columns=['Date', 'Actual', 'Forecast'])
bnb_future['Date'] = pd.date_range(start=bnb_past['Date'].iloc[-1] + pd.Timedelta(days=1), periods=n_forecast)
bnb_future['Forecast'] = bnb_forecast.flatten()
bnb_future['Actual'] = np.nan


results = pd.concat([bnb_past, bnb_future]).set_index('Date')


fig = px.line(results, x=results.index, y=['Actual', 'Forecast'], title='Binance Coin Forecasting in 2 months')
fig.add_shape(
    go.layout.Shape(
        type="line",
        x0=results.index[-n_forecast], y0=results['Actual'].min(),
        x1=results.index[-n_forecast], y1=results['Actual'].max(),
        line=dict(color="red", width=1, dash="dash")
    )
)
fig.show()
```




```
results.shape
print("Price of Emeren Group on", results.index[-n_forecast], "should be ",results.Forecast[-n_forecast])
```

```
Price of Emeren Group on 2023-11-25 00:00:00 should be  15.00208568572998
```

Emeren Group Ltd (SOL) Follow

NYSE - NYSE Delayed Price. Currency in USD

2.3600 +0.0800 (+3.5088%) **2.3510** -0.01 (-0.38%)

At close: November 24 01:00PM EST After hours: Nov 24, 04:47PM EST

```
results.shape
print("Price of Binance Coin on", results.index[-n_forecast], "should be ",results.Forecast[-n_forecast])
```

```
Price of Binance Coin on 2023-11-25 00:00:00 should be  240.0866241455078
```

BNB USD (BNB-USD)

CCC - CoinMarketCap. Currency in USD

Follow

233.80 +0.44 (+0.19%)

As of 09:29PM UTC. Market open.

Save model

```
sol_model.save('sol.h5')
```

```
bnb_model.save('bnb.h5')
```

158A Le Loi, Danang, Vietnam
www.vnuk.edu.vn | contact@vnuk.edu.vn
Phone: +84 (236) 3 64 65 77