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:	Student Name:	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

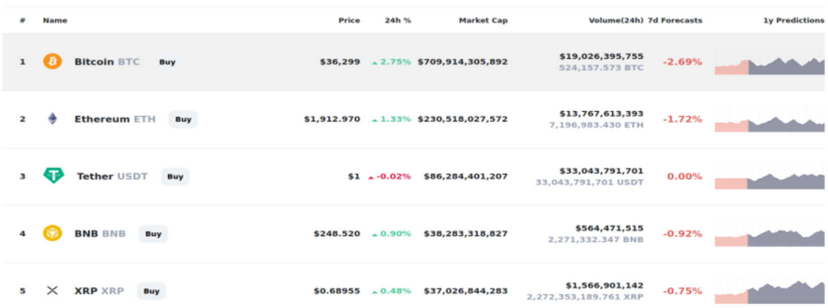
Developing a website for a machine learning model involves several steps, including designing the user interface, creating the back-end to serve predictions

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%	

Part 1. Report on the challenge.

1. Data: BTC-USD, BNB-USD

- Select the date to download datasets with d1 is end date and d2 is start date.

```
# set up date to download dataset (2 years)
d1 = today.strftime("%Y-%m-%d")
end_date = d1
d2 = date.today() - timedelta(days=365*2)
d2 = d2.strftime("%Y-%m-%d")
start_date = d2
```

- Write a function to download datasets from https://finance.yahoo.com/ with parameters being crypto currency name, start date and end date.

```
# function to download dataset from yahoo
def download_data(crypto_name, start_date, end_date):
    # Dowload data form Yahoo Finance
    data = yf.download(crypto_name,  start=start_date,  end=end_date,  progress=False)
    data["Date"] = data.index
    data = data[["Date", "Open", "High", "Low", "Close", "Adj Close","Volume"]]
    data.reset_index(drop=True, inplace=True)
    return data
```

- Then use the function you just created to download the datasets, discover information (Column, non-null count, data type), and mean, IQR, and standard deviation of the columns in the dataset.

```
 # dowload btc-usd dataset
btc_data = download_data("BTC-USD", start_date, end_date)
btc_data.head()
btc_data.info() # explore BTC information
# Explore BTC count-mean-min-IQR-max-std
btc_data.describe()# Explore BTC count-mean-min-IQR-max-std

# download bnb-usd dataset
bnb_data = download_data("BNB-USD", start_date, end_date)
bnb_data.head()
bnb_data.info()# explore BNB information
bnb_data.describe()# Explore BNB count-mean-min-IQR-max-std
```

- Finally, check null values in datasets. Because datasets are downloaded directly at finance yahoo and are tracked every second, there are no null values.

```
# check null values
print("BTC:")
print("Has Null values?:",btc_data.isnull().values.any())
print("Shape:", btc_data.shape)

print("\nBNB:")
print("Has Null values?:",bnb_data.isnull().values.any())
print("Shape:", bnb_data.shape)
```

```
-------------------------------------
BTC:
Has Null values?: False
Shape: (730, 7)

BNB:
Has Null values?: False
Shape: (730, 7)
```

2. EDA – Exploratory Data Analyst

- Write plot functions to draw “Monthly Comparison Between Stock Open and Close price”, “Month High and Low stock price”, “Stock analysis Chart”

```
# plot functions
def plot(y):
    names = cycle(['Stock Open Price','Stock Close Price','Stock High Price','Stock Low Price'])
    fig = px.line(y, x=y.Date, y=[y['Open'], y['Close'], y['High'], y['Low']],labels={'Date': 'Date','value':'Stock value'})
    fig.update_layout(title_text='Stock analysis chart', font_size=15, font_color='black',legend_title_text='Stock Parameters')
    fig.for_each_trace(lambda t:  t.update(name = next(names)))
    fig.update_xaxes(showgrid=False)
    fig.update_yaxes(showgrid=False)
    fig.show()
    return fig

def high_low_plot(df, y, new_order):
    monthvise_high = y.groupby(df['Date'].dt.strftime('%B'))['High'].max()
    monthvise_high = monthvise_high.reindex(new_order, axis=0)
    monthvise_low = y.groupby(df['Date'].dt.strftime('%B'))['Low'].min()
    monthvise_low = monthvise_low.reindex(new_order, axis=0)

    fig = go.Figure()
    fig.add_trace(go.Bar( x=monthvise_high.index, y=monthvise_high,
        name='Stock high Price', marker_color='rgb(0, 153, 204)'))
    fig.add_trace(go.Bar( x=monthvise_low.index, y=monthvise_low,
        name='Stock low Price', marker_color='rgb(255, 128, 0)'))
    fig.update_layout(barmode='group', title=' Monthly High and Low stock price')
    fig.show()
    return fig

def open_close_plot(data):
    fig = go.Figure()
    fig.add_trace(go.Bar( x=data.index, y=data['Open'],
        name='Stock Open Price', marker_color='crimson' ))
    fig.add_trace(go.Bar( x=data.index, y=data['Close'],
        name='Stock Close Price', marker_color='lightsalmon’ ))
```

```
    fig.update_layout(barmode='group', xaxis_tickangle=-45,
      title='Monthly comparision between Stock open and close price')
    fig.show()
    return fig
```

- Data_yearly function to explore dataset of each year

```
# Explore data yearly
def data_yearly(df, start_year, end_year):
    df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')
    y = df.loc[(df['Date'] >= f'{str(start_year)}-01-01') & (df['Date'] < f'{str(end_year)}-01-01')]
    y.drop(y[['Adj Close','Volume']],axis=1)
    monthvise= y.groupby(y['Date'].dt.strftime('%B'))[['Open','Close']].mean()
    new_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September',
'October', 'November', 'December']
    monthvise = monthvise.reindex(new_order, axis=0)
    print(monthvise)
    open_close_plot(monthvise)
    high_low_plot(df, y, new_order)
    plot(y)
```

2.1. BTC-USD

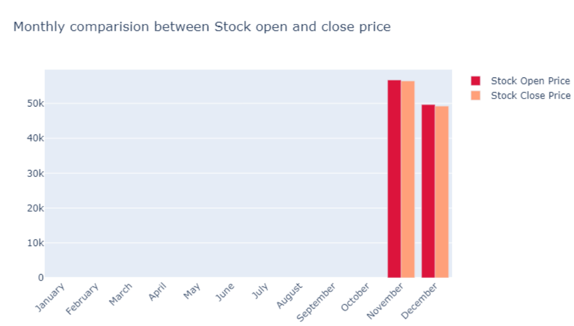
- Confirm the start and end dates of the dataset.

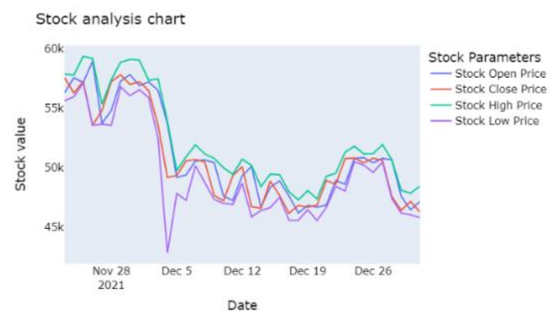
```
# start date and end date of the dataset
start_date=btc_data.iloc[0][0]
end_date=btc_data.iloc[-1][0]
print('Starting Date',start_date)
print('Ending Date',end_date)
-----------------------------------
Starting Date 2021-11-23 00:00:00
Ending Date 2023-11-22 00:00:00
```

2.1.1. In 2021

- Use data_yearly function to discover information about time and data trends in 2021.

```
# BTC
data_yearly(btc_data, 2021, 2022)
-----------------------------------
                Open        Close
Date
January          NaN          NaN
February         NaN          NaN
March            NaN          NaN
April            NaN          NaN
May              NaN          NaN
June             NaN          NaN
July             NaN          NaN
August           NaN          NaN
September        NaN          NaN
October          NaN          NaN
November  56708.447754  56446.184082
December  49670.411794  49263.209173
```



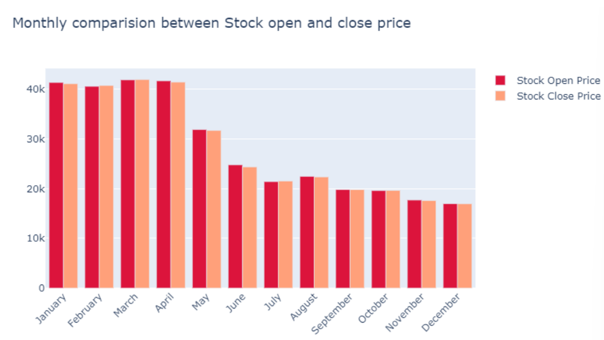


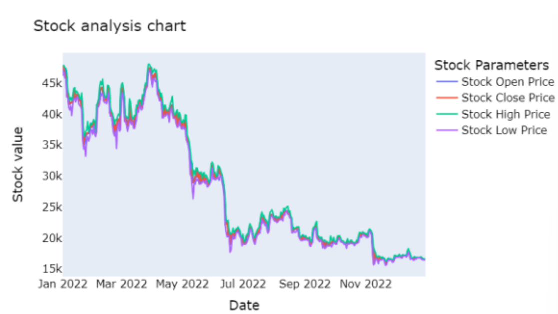
- In 2021, I only took BTC data for November-December. During this time, the value of BTC dropped sharply, and the close price reached its lowest on December 17 with a value of 46.2k.

2.1.2 In 2022

- Use data_yearly function to discover information about time and data trends in 2022.

```
# BTC
data_yearly(btc_data, 2022, 2023)
--------------------------------------------------
                Open         Close
Date
January    41368.073463  41114.422379
February   40591.103934  40763.474051
March      41889.148438  41966.237525
April      41694.653646  41435.319661
May        31900.711127  31706.105217
June       24783.338477  24383.685482
July       21424.733052  21539.253843
August     22471.866557  22366.266318
September  19821.353711  19804.779232
October    19616.090285  19650.525643
November   17711.480599  17600.814323
December   16969.578818  16949.608808
```





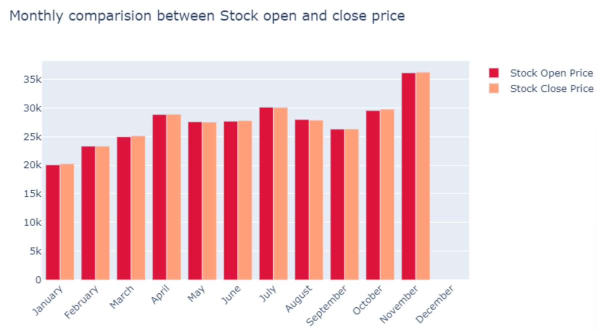
- In 2022, BTC maintained its year-long downward trend. Although there was an increase in fluctuation around March with the close price reaching a peak value of 47.44k. But after that, the currency continued to fall sharply, reaching 15.88k in early November and fluctuating slightly until the end of the year.

2.1.3 In 2023

- Use data_yearly function to discover information about time and data trends in 2023.

```
# BTC
```

```
data_yearly(btc_data, 2023, 2024)
------------------------------------
                 Open          Close
Date
January    20043.860131  20250.717490
February   23304.086007  23304.539202
March      24945.340411  25116.900895
April      28823.841732  28857.574544
May        27562.723160  27499.307145
June       27651.926758  27763.198437
July       30098.767578  30057.469947
August     27959.756615  27852.792843
September  26271.268229  26306.136393
October    29511.811114  29755.895161
November   36096.618253  36224.665305
December            NaN           NaN
```



- From the beginning of 2023 until now, the increasing trend is clearly shown compared to last year. BTC has increased sharply to date with a peak of 37.31k in mid-November and tends to continue to increase until the end of the year.

2.2. BNB-USD

- Confirm the start and end dates of the dataset.

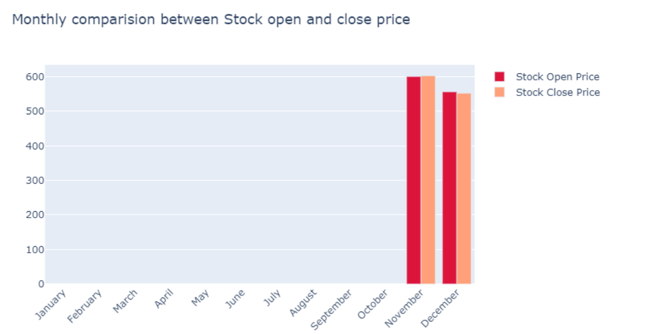
```
# start date and end date of the dataset
start_date=bnb_data.iloc[0][0]
end_date=bnb_data.iloc[-1][0]
print('Starting Date',start_date)
print('Ending Date',end_date)
----------------------------------
Starting Date 2021-11-23 00:00:00
Ending Date 2023-11-22 00:00:00
```

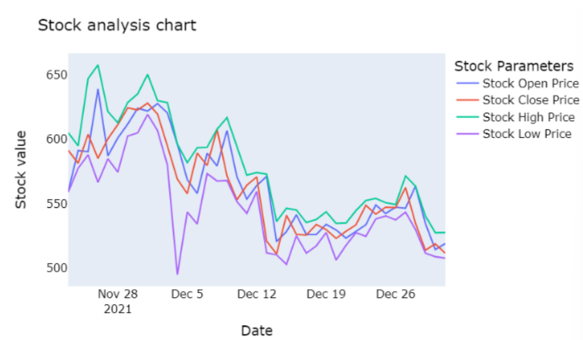
2.2.1. In 2021

- Use data_yearly function to discover information about time and data trends in 2021.

```
# BTC
data_yearly(bnb_data, 2021, 2022)
-----------------------------------
             Open      Close
```

Date		
January	NaN	NaN
February	NaN	NaN
March	NaN	NaN
April	NaN	NaN
May	NaN	NaN
June	NaN	NaN
July	NaN	NaN
August	NaN	NaN
September	NaN	NaN
October	NaN	NaN
November	600.494461	602.54950
December	556.038056	551.83804



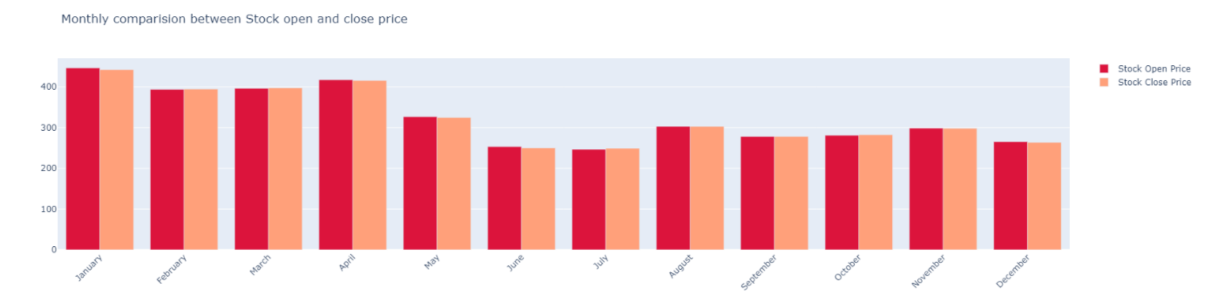


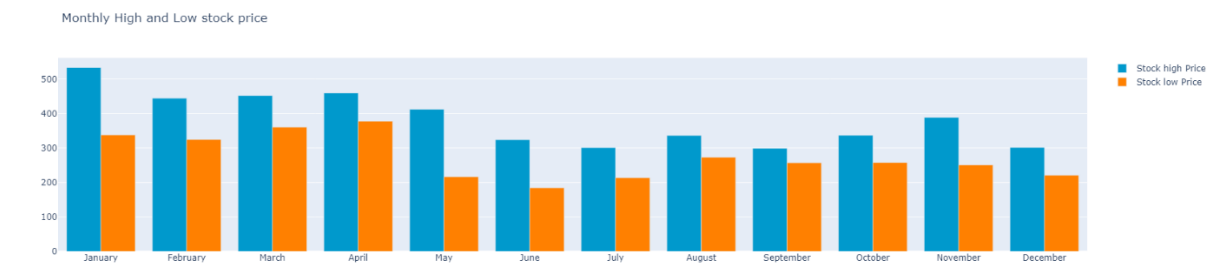
- In 2021, I have data for the last 40 days of the year. During these 40 days, BNB fluctuated sharply until the end of the year and the close price reached its lowest point in mid-December with a value of 511.22k.

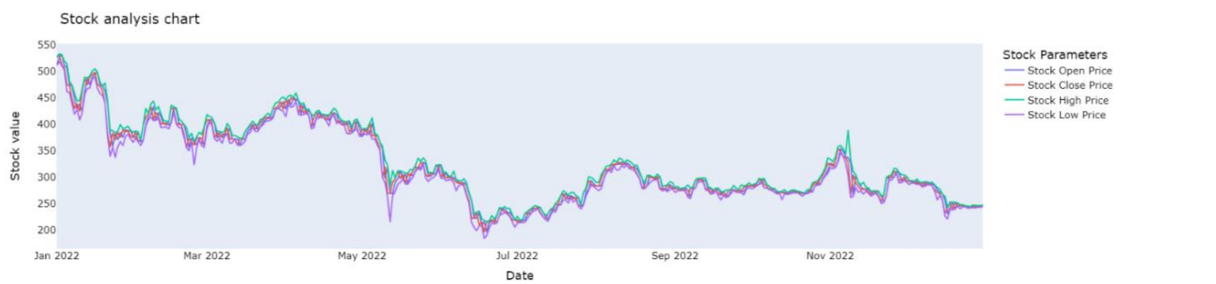
2.2.2 In 2022

- Use data_yearly function to discover information about time and data trends in 2022.

```
# BTC
data_yearly(bnb_data, 2022, 2023)
--------------------------------------------------
                Open        Close
Date
January    446.455420  442.029668
February   393.852540  394.614115
March      396.261171  397.352503
April      417.316946  415.605955
May        326.631463  324.777569
June       253.123441  249.748236
July       246.791995  248.866326
August     302.907766  302.763520
September  277.855970  278.032468
October    280.898194  282.266879
November   298.695680  297.861898
December   265.371770  263.616611
```





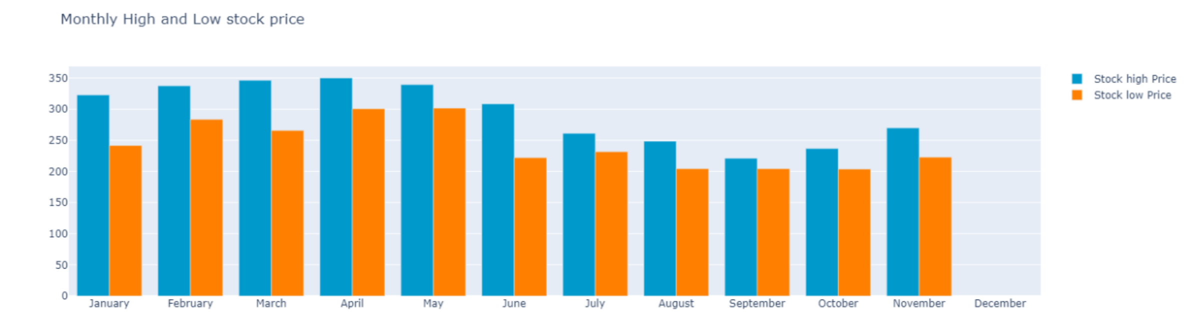


- In 2022, BNB decreased sharply and the close price reached the smallest value of 197.04k. After that, fluctuations increased slightly until the end of the year.

2.2.3 In 2023

- Use data_yearly function to discover information about time and data trends in 2023.

```
# BTC
data_yearly(bnb_data, 2023, 2024)
------------------------------------
                 Open       Close
Date
January    285.929391  288.004860
February   313.786586  313.408470
March      308.748325  309.237018
April      323.645232  324.307049
May        314.534315  313.544577
June       255.371225  253.156196
July       242.476818  242.500122
August     230.508589  229.714023
September  213.420660  213.360542
October    215.122406  215.475766
November   242.951512  243.389517
December          NaN         NaN
```



- In the first half of 2023, BNB increased sharply in the first half of the year and the close price peaked at 343.19k. However, in the second half of the year, BNB dropped sharply with the smallest value reaching 206.03k.

3. Building Model

- Model: Long Short-Term Memory (LSTM)
- Lenght training data: 730
- Output: Predicting Close Price
- Tạo create_dataset function để chuyển dataset sang array X và array Y dựa vào biến time_steps để train model.

```
 # convert an array of values into a dataset matrix
def create_dataset(dataset, time_step):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
```

```
        a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

- Create pre_processing function to explore the chart about Close price in 2021-2023, size of training data with output close_stock (data frame includes Date and Close columns to training, close_df (data frame includes scaled Close values), train_data (data to train), test_data (data to test) and scaler (to scaler or inverse transform).

3.1. BTC-USD

3.1.1 Pre-processing

```
close_stock, close_df, train_data, test_data, scaler = pre_processing(btc_data)
close_stock.to_csv("btc_training_data.csv")
----------------------------------------
Shape of close dataframe: (730, 2)
```


```
Total data for prediction:  730
Dataset: (730, 1)
train_data:  (584, 1)
test_data:  (146, 1)
```

- Now, I set the time_step parameter representing the number of previous time steps that will be used to predict the current time step. Then use create_dataset function above to divide train_data and test_data into X_train, y_train, X_test, y_test based on time_step. Finally, I need to reshape X_train and X_test to ensure they are in the correct format [samples, time steps, features] while using the LSTM model.

```
time_step = 1
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)

# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)
---------------------------------------
X_train:  (582, 1)
```

```
y_train:  (582,)
X_test:  (144, 1)
y_test (144,)
X_train:  (582, 1, 1)
X_test:  (144, 1, 1)
```

3.1.2. Train Model

- To start training the model, first we need to initialize the model and add an LSTM layer with 10 units, the input has dimension (None, 1) and use the ReLU activation function. Then add a Dense layer with 1 unit, this is the output layer to predict the closing price, compile the model, choose the mean squared error loss function and use the Adam optimizer. Finally, train the model with the set X_train and y_train with the validation data X_test, y_test that I just created in section 3.1.1 with epochs = 80 and batch_size = 1.

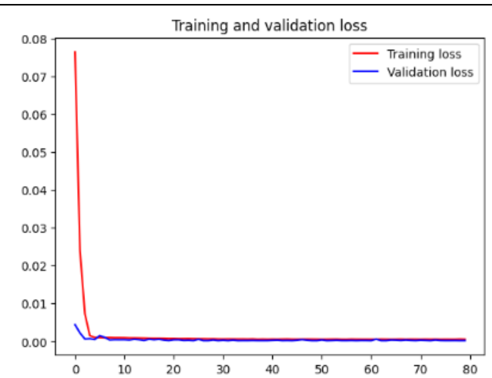
```
model=Sequential()
model.add(LSTM(10,input_shape=(None,1),activation="relu"))
model.add(Dense(1))
model.compile(loss="mean_squared_error",optimizer="adam")

history = model.fit(X_train, y_train, validation_data=(X_test,y_test),                    epochs=80,
batch_size=1, verbose=1)
```

3.1.3 Evaluate Model.

- To start training the model, first we need to initialize the model and add an LSTM layer with 10 units, the input has dimension (None, 1) and use the ReLU activation function. Then add a Dense layer with 1 unit, this is the output layer to predict the closing price, compile the model, choose the mean squared error loss function and use the Adam optimizer. Finally, train the model with the set X_train and y_train with the validation data X_test, y_test that I just created in section 3.1.1 with epochs = 80 and batch_size = 1.

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(loss))
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.figure()
plt.show()
```



- The plot shows that the model is good with training loss and validation loss decreasing as the number of epochs increases and the model is not overfitting because the number of epochs is just enough so that the model does not learn more.
- Now, let's explore whether the evaluation metrics are really as good as the model shows. Before calculating the metrics, I need to transform back the scaled parameters to the original form.

```
# the prediction
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
train_predict.shape, test_predict.shape
```

```
# Transform back to original form
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))

print("Train data explained variance regression score:",
      explained_variance_score(original_ytrain, train_predict))
print("Test data explained variance regression score:",
      explained_variance_score(original_ytest, test_predict))
--------------------------------------
Train data explained variance regression score: 0.9910205833995471
Test data explained variance regression score: 0.9704530610197392
```

- Train data explained variance regression score: 0.9910: A score close to 1 indicates that your model is very good at explaining the variance in the training data set. This is a positive result and shows that the model learned important variations in the training data.
- Test data explained variance regression score: 0.9704: Similar to the above, a score close to 1 for the test set is also a positive result. It represents the model's ability to explain the variance in the test data

```
print("Train data R2 score:", r2_score(original_ytrain, train_predict))
print("Test data R2 score:", r2_score(original_ytest, test_predict))
------------------------------------
Train data R2 score: 0.9909388821326138
Test data R2 score: 0.9690370316861815
```

- Train data R2 score: 0.9909: R2 score close to 1 for the training data set is a positive result. It indicates that your model explains a large portion of the variation in the training data.
- Test data R2 score: 0.9690: For the test set, the R2 score is also quite high, implying that the model explains a large portion of the variation in the test data.

```
 # predictions plot
look_back=time_step
trainPredictPlot = np.empty_like(close_df)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(close_df)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(close_df)-1, :] = test_predict
print("Test predicted data: ", testPredictPlot.shape)
names = cycle(['Original close price','Train predicted close price','Test predicted close price'])
plotdf = pd.DataFrame({'date': close_stock['Date'],
    'original_close': close_stock['Close'],
    'train_predicted_close': trainPredictPlot.reshape(1,-1)[0].tolist(),
    'test_predicted_close': testPredictPlot.reshape(1,-1)[0].tolist()})
```

```
fig = px.line(plotdf,x=plotdf['date'], y=[plotdf['original_close'],plotdf['train_predicted_close'],
  plotdf['test_predicted_close']],
  labels={'value':'Stock price','date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predicted close price',
plot_bgcolor='white', font_size=15, font_color='black', legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
-----------------------------------
Train predicted data:  (730, 1)
Test predicted data:  (730, 1)
```

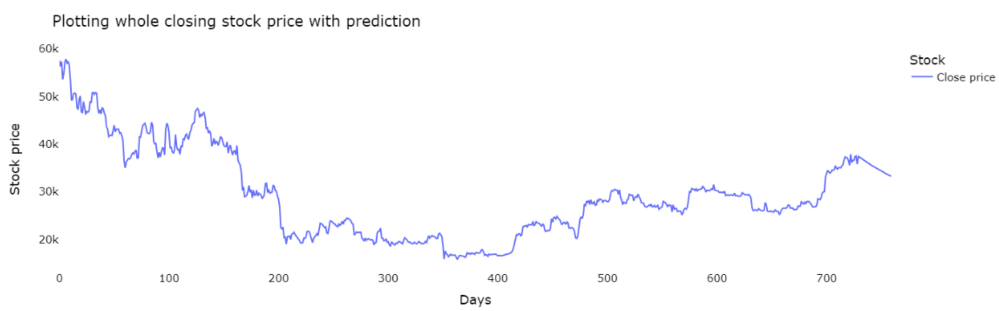
- Looking at the graph above, the results of the train predicted and test predicted are quite close to the original value. This ensures again a well-functioning model.
- Next, I will predict the close price within the next 30 days. First, I need to prepare the input data (x_input) then convert it to a list so that it can be easily expanded and updated. Then I use a loop to predict the next 30 days, if temp_input has enough data then the model predicts the next value on temp_input and expands lst_out. In contrast, the model uses the current temp_input to predict the outcome.

```
 # Prepare input dataset
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
lst_output=[]
n_steps=time_step
i=0
pred_days = 30
while(i<pred_days):
   if(len(temp_input)>time_step):
       x_input=np.array(temp_input[1:])
       x_input = x_input.reshape(1,-1)
       x_input = x_input.reshape((1, n_steps, 1))
       yhat = model.predict(x_input, verbose=0)
```

```
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())
        lst_output.extend(yhat.tolist())
        i=i+1
print("Output of predicted next days: ", len(lst_output))
---------------------------------------
Output of predicted next days:  30
```

- Next, I will convert the lst_output prediction result into a list and combine it with the normalized close Price (close_df). Use scaler.inverse_transform to convert the normalized value to the original value. Finally, plot the predicted and actual values of the Close price.

```
lstmdf=close_df.tolist()
lstmdf.extend((np.array(lst_output).reshape(-1,1)).tolist())
lstmdf=scaler.inverse_transform(lstmdf).reshape(1,-1).tolist()[0]
names = cycle(['Close price'])
fig = px.line(lstmdf,labels={'value': 'Stock price','index': 'Days'})
fig.update_layout(title_text='Plotting whole closing stock price with prediction',plot_bgcolor='white',
font_size=15, font_color='black',legend_title_text='Stock')
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
---------------------------------------
```



- Finally, save model.

```
model.save("btc_lstm.h5")
```

3.2. BNB-USD

3.2.1 Pre-processing

```
close_stock, close_df, train_data, test_data, scaler = pre_processing(btc_data)
close_stock.to_csv("btc_training_data.csv")
---------------------------------------
```

```
Shape of close dataframe: (730, 2)
```


```
Total data for prediction:  730
Dataset: (730, 1)
train_data:  (584, 1)
test_data:  (146, 1)
```

- Now, I set the time_step parameter representing the number of previous time steps that will be used to predict the current time step. Then use create_dataset function above to divide train_data and test_data into X_train, y_train, X_test, y_test based on time_step. Finally, I need to reshape X_train and X_test to ensure they are in the correct format [samples, time steps, features] while using the LSTM model.

```
time_step = 1
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)

# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)
--------------------------------------
X_train:  (582, 1)
y_train:  (582,)
X_test:  (144, 1)
y_test (144,)
X_train:  (582, 1, 1)
X_test:  (144, 1, 1)
```

3.2.2. Train Model

- To start training the model, first we need to initialize the model and add an LSTM layer with 10 units, the input has dimension (None, 1) and use the ReLU activation function. Then add a Dense layer with 1 unit, this is the output layer to predict the closing price, compile the model, choose the mean squared error loss function and use the Adam optimizer. Finally, train the model with the set X_train and y_train with the validation data X_test, y_test that I just created in section 3.1.1 with epochs = 80 and batch_size = 1.

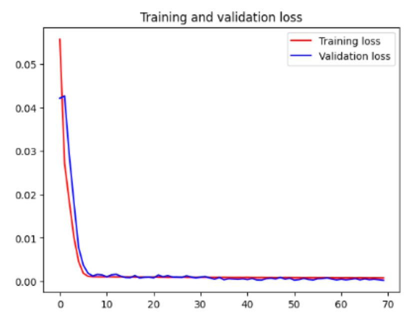
```
model=Sequential()
```

```
model.add(LSTM(10,input_shape=(None,1),activation="relu"))
model.add(Dense(1))
model.compile(loss="mean_squared_error",optimizer="adam")
history = model.fit(X_train, y_train, validation_data=(X_test,y_test), epochs=80, batch_size=1,
verbose=1)
```

3.2.3 Evaluate Model.

- To start training the model, first we need to initialize the model and add an LSTM layer with 10 units, the input has dimension (None, 1) and use the ReLU activation function. Then add a Dense layer with 1 unit, this is the output layer to predict the closing price, compile the model, choose the mean squared error loss function and use the Adam optimizer. Finally, train the model with the set X_train and y_train with the validation data X_test, y_test that I just created in section 3.1.1 with epochs = 80 and batch_size = 1.

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(loss))
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.figure()
plt.show()
```



- The plot shows that the model is good with training loss and validation loss decreasing as the number of epochs increases and the model is not overfitting because the number of epochs is just enough so that the model does not learn more.
- Now, let's explore whether the evaluation metrics are really as good as the model shows. Before calculating the metrics, I need to transform back the scaled parameters to the original form.

```
# the prediction
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
train_predict.shape, test_predict.shape

# Transform back to original form
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))

print("Train data explained variance regression score:",
    explained_variance_score(original_ytrain, train_predict))
print("Test data explained variance regression score:",
    explained_variance_score(original_ytest, test_predict))
--------------------------------------
Train data explained variance regression score: 0.9805741027397359
Test data explained variance regression score: 0.9027978363108912
```

- Train data explained variance regression score: 0.9806: A score close to 1 indicates that your model is very good at explaining the variance in the training data set. This is a positive result and shows that the model learned important variations in the training data.
- Test data explained variance regression score: 0.9028: Similar to the above, a score close to 1 for the test set is also a positive result. It represents the model's ability to explain the variance in the test data

```
print("Train data R2 score:", r2_score(original_ytrain, train_predict))
print("Test data R2 score:", r2_score(original_ytest, test_predict))
------------------------------------
Train data R2 score: 0.9805681145794243
Test data R2 score: 0.7935875833409349
```

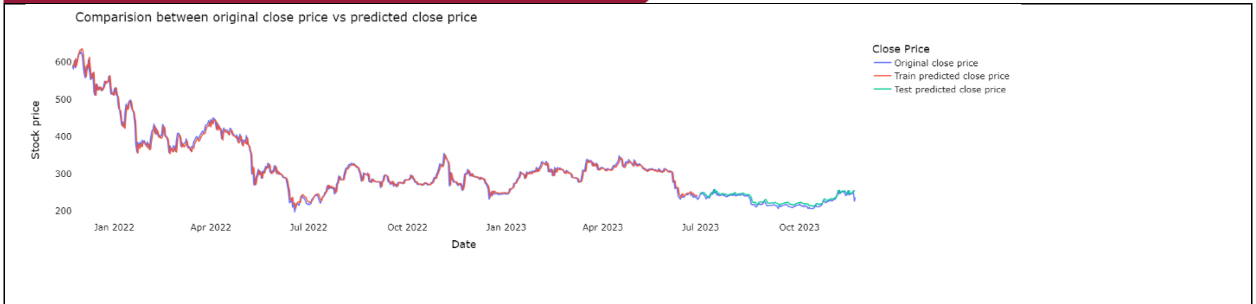
- Train data R2 score: 0.9806: R2 score close to 1 for the training data set is a positive result. It indicates that your model explains a large portion of the variation in the training data.
- Test data R2 score: 0.7936: For the test set, the R2 score is also not high.

```
 # predictions plot
look_back=time_step
trainPredictPlot = np.empty_like(close_df)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(close_df)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(close_df)-1, :] = test_predict
print("Test predicted data: ", testPredictPlot.shape)
names = cycle(['Original close price','Train predicted close price','Test predicted close price'])
plotdf = pd.DataFrame({'date': close_stock['Date'], 'original_close': close_stock['Close'],
    'train_predicted_close': trainPredictPlot.reshape(1,-1)[0].tolist(),
    'test_predicted_close': testPredictPlot.reshape(1,-1)[0].tolist()})

fig = px.line(plotdf,x=plotdf['date'], y=[plotdf['original_close'],plotdf['train_predicted_close'],
  plotdf['test_predicted_close']], labels={'value':'Stock price','date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predicted close price',
plot_bgcolor='white', font_size=15, font_color='black', legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
-----------------------------------
Train predicted data:  (730, 1)
Test predicted data:  (730, 1)
```

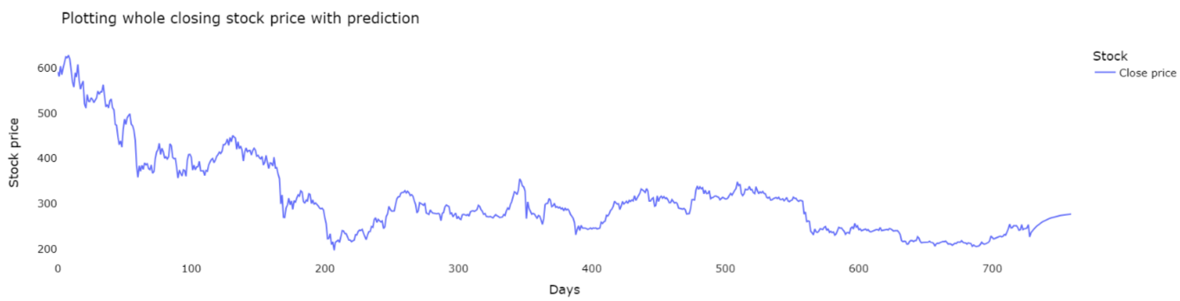


- Looking at the graph above, the results of the train predicted and test predicted are quite close to the original value. This ensures again a well-functioning model.
- Next, I will predict the close price within the next 30 days. First, I need to prepare the input data (x_input) then convert it to a list so that it can be easily expanded and updated. Then I use a loop to predict the next 30 days, if temp_input has enough data then the model predicts the next value on temp_input and expands lst_out. In contrast, the model uses the current temp_input to predict the outcome.

```
# Prepare input dataset
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
lst_output=[]
n_steps=time_step
i=0
pred_days = 30
while(i<pred_days):
    if(len(temp_input)>time_step):
        x_input=np.array(temp_input[1:])
        x_input = x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())
        lst_output.extend(yhat.tolist())
        i=i+1
print("Output of predicted next days: ", len(lst_output))
----------------------------------------
Output of predicted next days:  30
```

- Next, I will convert the lst_output prediction result into a list and combine it with the normalized close Price (close_df). Use scaler.inverse_transform to convert the normalized value to the original value. Finally, plot the predicted and actual values of the Close price.

```
lstmdf=close_df.tolist()
lstmdf.extend((np.array(lst_output).reshape(-1,1)).tolist())
lstmdf=scaler.inverse_transform(lstmdf).reshape(1,-1).tolist()[0]
names = cycle(['Close price'])
fig = px.line(lstmdf,labels={'value': 'Stock price','index': 'Days'})
fig.update_layout(title_text='Plotting whole closing stock price with prediction',plot_bgcolor='white',
font_size=15, font_color='black',legend_title_text='Stock')
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
---------------------------------------
```



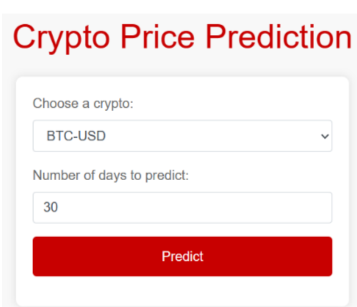
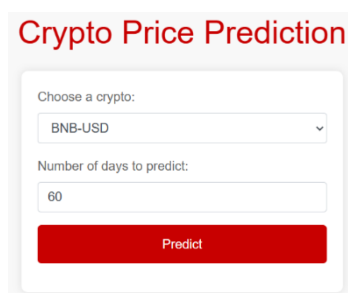
- Finally, save model.

```
model.save("bnb_lstm.h5")
```

4. Visualization Website:

- The Website is used to visualize prediction results from the trained model and enter input (crypto name and number of days to predict).
- Websites architecture:

  --------- app.py
  --------- templates
  ---------------- index.html
  ---------------- result.html
  ---------------- error.html

BTC-USD Crypto with 30 days to predict:	BNB-USD Crypto with 30 days to predict:
 	 

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