



CRYPTOCURRENCY

2023 BITCOIN PREDICTION PRICE IN 30 DAYS USING RNN

CHECK THIS

@tainguyen30

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH
KHOA CƠ KHÍ CHẾ TẠO MÁY
BỘ MÔN CƠ ĐIỆN TỬ



HCMUTE

**BÁO CÁO CUỐI KÌ
MÔN: TRÍ TUỆ NHÂN TẠO**

DỰ ĐOÁN GIÁ BITCOIN TRONG 30 NGÀY BẰNG THUẬT TOÁN RNN

GVHD: PGS.TS Nguyễn Trường Thịnh

MHP: ARIN337629_22_2_09

SVTH: Nguyễn Hữu Duy Tài

MSSV: 20146143

Năm Học: HK2 2022-2023

Thành phố Thủ Đức, tháng 5 năm 2023

MỤC LỤC

1.	Giới thiệu	3
2.	Phương pháp.....	3
3.	Chi tiết.....	6
a)	<i>Cấu trúc hệ thống</i>	6
b)	<i>Mô hình hoá_Chuẩn hoá dữ liệu</i>	7
4.	Kết quả.....	8
5.	Kết luận.....	9
6.	Code.....	

1. Giới thiệu

Bitcoin (BTC) là một loại tiền mã hoá, có vốn hoá và giá trị đứng thứ nhất trong thị trường tiền mã hoá hiện nay. Bitcoin có cách hoạt động khác hẳn so với các loại tiền tệ điển hình: không có một ngân hàng trung ương nào quản lý nó và hệ thống hoạt động dựa trên một giao thức **mạng ngang hàng (peer to peer network)** trên Internet. Nguồn cung của BTC là hữu hạn (21 triệu đồng) nên giá của BTC sẽ có xu hướng tăng vượt bậc khi có dòng tiền đổ vào. Vì vậy, hàng loạt nhà đầu tư đã và đang có gambio tích luỹ càng nhiều BTC với giá thấp (mua thấp-bán cao), việc dự đoán chu trình tăng trưởng/suy thoái của BTC luôn là vấn đề được quan tâm hàng đầu đối với các nhà đầu tư.

2. Phương pháp

Với dữ liệu dạng chuỗi số, ta sử dụng mô hình mạng RNN (Recurrent neural network) hay còn được gọi là mạng nơron hồi quy.

a) Đôi nét về RNN:

Giả sử ta có bài toán: Cần phân loại hành động của người trong video, input là video 30s, output là phân loại hành động, ví dụ: đứng, ngồi, chạy, đánh nhau, bắn súng,... Khi xử lý video ta hay gặp khái niệm FPS (frame per second) tức là bao nhiêu frame (ảnh) mỗi giây. Ví dụ 1 FPS với video 30s tức là lấy ra từ video 30 ảnh, mỗi giây một ảnh để xử lý.

Hướng xử lý

Ta dùng 1 FPS cho video input ở bài toán trên, tức là lấy ra 30 ảnh từ video, ảnh 1 ở giây 1, ảnh 2 ở giây 2,... ảnh 30 ở giây 30. Bây giờ input là 30 ảnh: ảnh 1, ảnh 2,... ảnh 30 và output là phân loại hành động. Nhận xét:

- Các ảnh có thứ tự, ví dụ ảnh 1 xảy ra trước ảnh 2, ảnh 2 xảy ra trước ảnh 3,... Nếu ta đảo lộn các ảnh thì có thể thay đổi nội dung của video.
Ví dụ: nội dung video là cảnh bắn nhau, thứ tự đúng là A bắn trúng người B và B chết, nếu ta đảo thứ tự ảnh thành người B chết xong A mới bắn thì rõ ràng bây giờ A không phải là kẻ giết người.
=> Nội dung video bị thay đổi.
- Ta có thể dùng CNN để phân loại 1 ảnh trong 30 ảnh trên, nhưng rõ ràng là 1 ảnh không thể mô tả được nội dung của cả video.
Ví dụ: Cảnh người cướp điện thoại, nếu ta chỉ dùng 1 ảnh là người đẩy cầm điện thoại lúc cướp xong thì ta không thể biết được cả hành động cướp.
➔ Cần một mô hình mới có thể giải quyết được bài toán với input là sequence (chuỗi ảnh 1....30)
➔ RNN ra đời.

b) Dữ liệu dạng Sequence (chuỗi)

Dữ liệu có thứ tự như các ảnh tách từ video ở trên được gọi là sequence, time-series data.

Trong bài toán dự đoán đột quy tim cho bệnh nhân bằng các dữ liệu tim mạch khám trước đó.

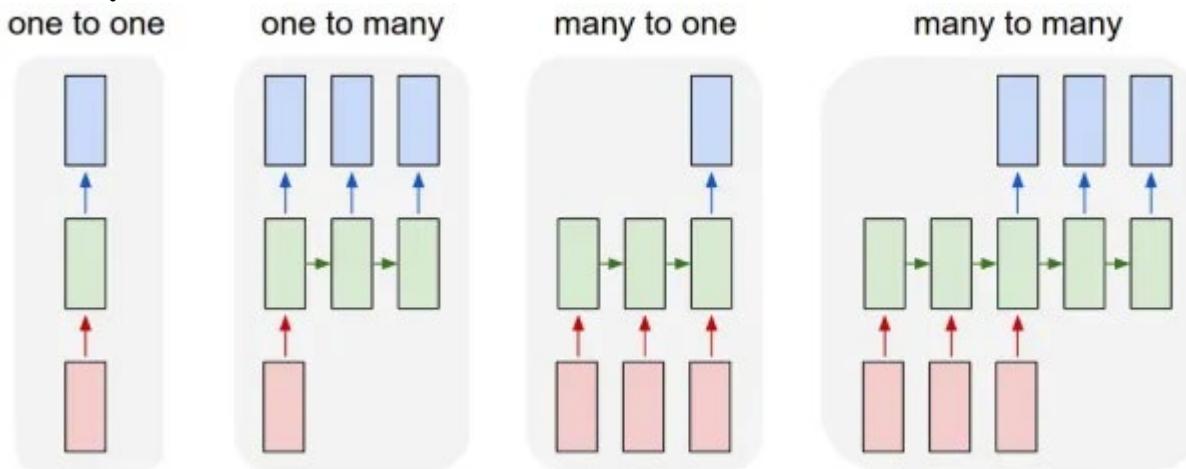
- + Input là dữ liệu của những lần khám trước đó, ví dụ i_1 là lần khám tháng 1, i_2 là lần khám tháng 2,... i_8 là lần khám tháng 8.
- + (i_1, i_2, \dots, i_8) được gọi là sequence data. RNN sẽ học từ input và dự đoán xem bệnh nhân có bị đột quy tim hay không.

Ví dụ khác là trong bài toán dịch tự động với input là 1 câu, ví dụ “tôi yêu Việt Nam”

- + Vị trí các từ và sự xếp xắp cực kì quan trọng đến nghĩa của câu

+ Dữ liệu input các từ [‘tôi’, ‘yêu’, ‘việt’, ‘nam’] được gọi là sequence data.

c) Phân loại bài toán RNN



- + One to one: mẫu bài toán cho Neural Network (NN) và Convolutional Neural Network (CNN), 1 input và 1 output,

VD: với CNN input là ảnh và output là ảnh được segment.

- + One to many: bài toán có 1 input nhưng nhiều output

VD: bài toán caption cho ảnh, input là 1 ảnh nhưng output là nhiều chữ mô tả cho ảnh đấy, dưới dạng một câu

- + Many to one: bài toán có nhiều input nhưng chỉ có 1 output

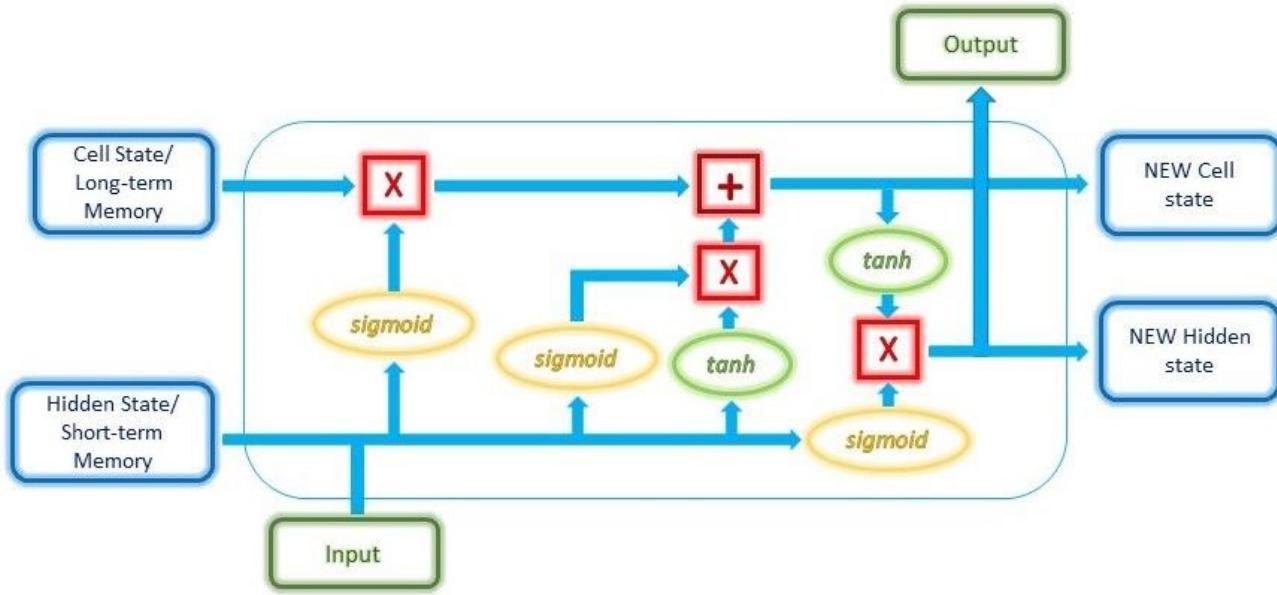
VD: bài toán phân loại hành động trong video, input là nhiều ảnh (frame) tách ra từ video, output là hành động trong video

- + Many to many: bài toán có nhiều input và nhiều output

VD: bài toán dịch từ tiếng anh sang tiếng việt, input là 1 câu gồm nhiều chữ: “I love Vietnam” và output cũng là 1 câu gồm nhiều chữ “Tôi yêu Việt Nam”.

d) Mô hình RNN-LSTM

LSTM Architecture



Cấu trúc bao gồm 4 tầng với các cổng Forget Gate (f_t), Input Gate (i_t), Output Gate (o_t). Các cổng này quyết định việc thông tin nào sẽ được lưu trữ, xoá, chỉnh sửa và truyền đi.

Trong các module này, có 2 output là Cell state (C_t) và Hidden state (h_t), với t là thời điểm đang xét. Cell state (C_t) đóng vai trò như một băng tải đưa thông tin đi xuyên suốt qua các module, nhờ vậy thông tin được lưu trữ cho các quá trình xử lý phía sau mà không mất đi. Đồng thời, các khôi xử lý khác có thể tác động vào nội dung của C_t để quyết định thông tin cần truyền tại các thời điểm. Các phép biến đổi trong modlue bao gồm sigma, tanh (chuyển về giá trị trong khoảng $[0;1]$), phép nhân và phép cộng.

+ Theo đó, giả sử hàm kích hoạt là hàm tanh, theo sơ đồ dễ thấy có 2 đầu vào qua hàm tanh:

$$\tilde{C}_t = \tanh(U_c * x_t + W_c * h_{t-1} + b_c)$$

+ Với các cổng f_t , i_t , o_t qua các sigma:

$$f_t = \sigma(U_f * x_t + W_f * h_{t-1} + b_f)$$

$$i_t = \sigma(U_i * x_t + W_i * h_{t-1} + b_i)$$

$$o_t = \sigma(U_o * x_t + W_o * h_{t-1} + b_o)$$

Khi đó, ct sẽ trở thành:

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

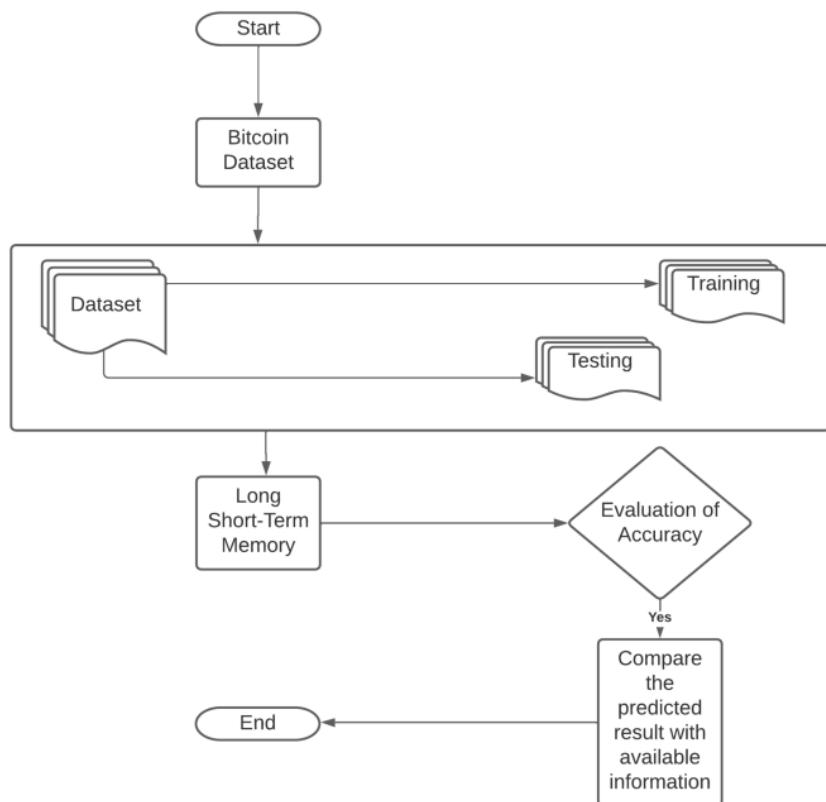
$$h_t = o_t * \tanh(c_t)$$

+ Nhận xét: ct chịu tác động bởi Forget gate nhằm loại bỏ các thông tin không cần thiết, đồng thời thêm các thông tin mới từ Input gate và từ Hidden layer của module trước, việc chọn lọc thông tin đưa vào hidden layer dựa vào kinh nghiệm qua các epochs.

Trong khi đó, ht thừa hưởng thông tin từ Output gate và ct. Do tính kế thừa này, mà LSTM phù hợp với các bài toán xử lý thông tin dạng chuỗi (Timeseries), ứng dụng trong các mô hình dự đoán giá trị liền kề (Forecast), phân loại thông tin, ...

3. Chi tiết

a) Cấu trúc hệ thống



+ B1: Lấy data từ file CSV từ ngày 17/9/2014 đến ngày 16/5/2023

+ B2: Để mô hình hoá dữ liệu, chúng ta sử dụng các cột sau: Date, Price, Open, High, Low, Close

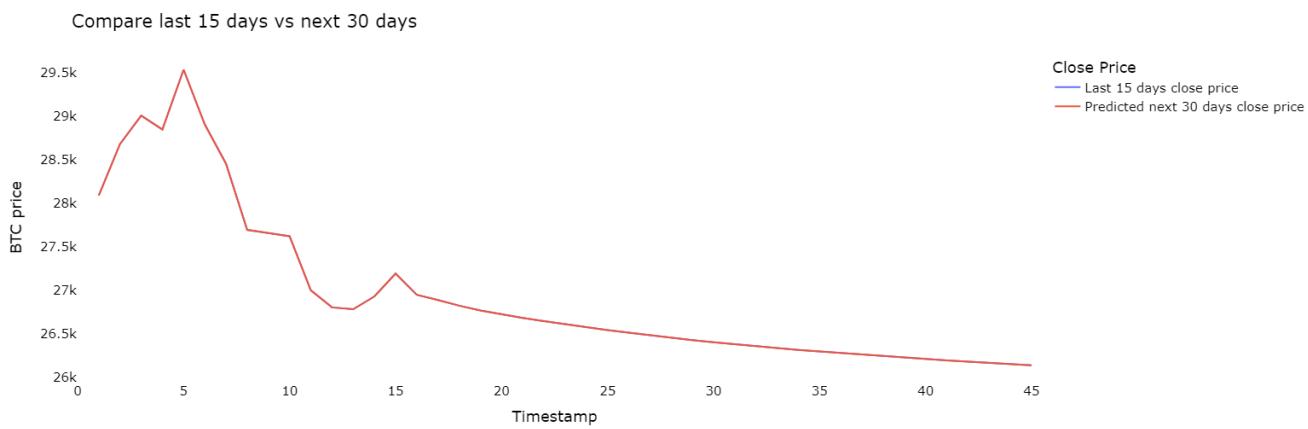
Features Used In The Dataset			
Sr. No.	Variable Name	Variable Description	Data Type
1	Time	Date and time of observation	Date
2	Volume	Sum total of trades taking place.	Number
3	Open	Opening price on the given day.	Number
4.	High	Highest price on the given day.	Number
5.	Low	Lowest price on the given day.	Number
6.	Close	Closing price on a given day.	Number

- + B3: Train model, sử dụng thuật toán và các tính năng để hỗ trợ mô hình nhằm dự đoán giá tương lai của BTC
- + B4: Kiểm nghiệm để đo lường tính chính xác của thuật toán
- + B5: Đánh giá độ chính xác mô hình bằng cách so sánh giá dự đoán và giá hiện tại tại cùng thời điểm cụ thể xem độ chính xác có cao hay không

b) **Mô hình hoá_ Chuẩn hoá dữ liệu**

- + Chia dữ liệu: 60% cho training và 40% cho test
- + Scale dữ liệu train và test (sau khi chia dữ liệu) sử dụng phương pháp MinMaxScaler
- + Gọi hàm tạo dữ liệu “create_dataset” cho mô hình LSTM
- + Cấu trúc model sử dụng một quãng thời gian “lookback” 15 ngày để dự đoán data những ngày tiếp theo
- + Đối với model LSTM: định hình lại (reshape) data đầu vào thành dạng 3 chiều: mẫu (sample), mốc thời gian (timestamp) và tính năng (feature)
 - Mẫu: bao gồm nhiều dấu thời gian xác định chiều rộng của cửa sổ trượt
 - Mốc thời gian: tương đương với số bước thời gian chúng ta sẽ chạy RNN của mình
 - Tính năng: bao gồm số lượng tính năng trong mỗi dấu thời gian
- + Do data đã được scale trước đó, mức scale của dự đoán sẽ nằm trong khoảng từ 0 đến 1, ta cần chuyển thang đo này về thang đo dữ liệu ban đầu.
- ➔ Dùng phép biến đổi nghịch đảo (inverse transform) để scale về dữ liệu nguyên bản
 - + Từ các dữ liệu đã được train, vẽ đồ thị cho biết mối tương quan giữa giá gốc và giá dự đoán

c) Kết quả



d) Kết luận

Theo mô hình dự đoán, giá BTC sẽ có chiều hướng giảm trong 30 ngày tới. Độ chính xác của mô hình khá cao, tuy nhiên thị trường vô cùng khắc nghiệt và khó đoán do nó còn mang các yếu tố khác như: tâm lý, dòng tiền,... nên việc dự đoán bằng các mô hình chỉ mang tính chất tham khảo vì nó tuân theo xu hướng các năm về trước.

CODE

- ❖ Github: [BTC Price 2023 Prediction in 30 days](#)
- ❖ Google Colab: [BTC Price Prediction in 30 days Colab](#)

```
# First we will import the necessary Library
import os
import pandas as pd
import numpy as np
import math
import datetime as dt

# For Evaluation we will use these library
from sklearn.metrics import mean_squared_error, mean_absolute_error,
explained_variance_score, r2_score
from sklearn.metrics import mean_poisson_deviance, mean_gamma_deviance,
accuracy_score
from sklearn.preprocessing import MinMaxScaler

# For model building we will use these library
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import LSTM

# For Plotting we will use these library
import matplotlib.pyplot as plt
from itertools import cycle
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

# Load our dataset
# Note it should be in same dir
maindf=pd.read_csv('/content/BTC-USD 2014_2023.csv')
print('Total number of days present in the dataset: ',maindf.shape[0])
print('Total number of fields present in the dataset:
',maindf.shape[1])
maindf.shape
```

```
# Viewing the first 5 lines  
maindf.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	9/17/2014	465.864014	468.174011	452.421997	457.334015	457.334015	21056800.0
1	9/18/2014	456.859985	456.859985	413.104004	424.440002	424.440002	34483200.0
2	9/19/2014	424.102997	427.834991	384.532013	394.795990	394.795990	37919700.0
3	9/20/2014	394.673004	423.295990	389.882996	408.903992	408.903992	36863600.0
4	9/21/2014	408.084991	412.425995	393.181000	398.821014	398.821014	26580100.0

```
# Viewing the last 5 lines  
maindf.tail()
```

	Date	Open	High	Low	Close	Adj Close	Volume
3159	5/12/2023	26987.66211	27055.64648	25878.42969	26804.99023	26804.99023	1.931360e+10
3160	5/13/2023	26807.76953	27030.48242	26710.87305	26784.07813	26784.07813	9.999172e+09
3161	5/14/2023	26788.97461	27150.97656	26661.35547	26930.63867	26930.63867	1.001486e+10
3162	5/15/2023	26931.38477	27646.34766	26766.09766	27192.69336	27192.69336	1.441323e+10
3163	5/16/2023	27171.22656	27297.75000	26878.94727	27023.26172	27023.26172	1.298952e+10

```
# Print info about Dataframe  
# Number of column, column labels, number of cells in each column (non-null values), column data types, memory usage  
maindf.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 3164 entries, 0 to 3163  
Data columns (total 7 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          --          --  
 0   Date        3164 non-null    object    
 1   Open         3164 non-null    float64  
 2   High         3164 non-null    float64  
 3   Low          3164 non-null    float64  
 4   Close        3164 non-null    float64  
 5   Adj Close    3164 non-null    float64  
 6   Volume       3164 non-null    float64  
dtypes: float64(6), object(1)  
memory usage: 173.2+ KB
```

```

# Return description of the data
# Numeric data
# count: the number of not-empty values, mean: the average value, std:
# standard deviation
# min: minimum value, 25%/50%/75%: 25%/50%/75% percentile
maindf.describe()

```

	Open	High	Low	Close	Adj Close	Volume
count	3164.000000	3164.000000	3164.000000	3164.000000	3164.000000	3.164000e+03
mean	13367.719887	13698.559985	13003.831583	13374.702320	13374.702320	1.660910e+10
std	16038.811751	16444.369898	15573.719870	16035.706420	16035.706420	1.967190e+10
min	176.897003	211.731003	171.509995	178.102997	178.102997	5.914570e+06
25%	736.317734	744.883255	730.288757	738.742493	738.742493	1.198100e+08
50%	7453.485840	7617.785889	7302.487060	7461.397949	7461.397949	9.615560e+09
75%	19343.587892	19656.905762	19025.690427	19354.905278	19354.905278	2.784107e+10
max	67549.734380	68789.625000	66382.062500	67566.828130	67566.828130	3.509680e+11

```

# Checking null values
print('Null Values:', maindf.isnull().values.sum())
print('NA values:', maindf.isnull().values.any())
# Final shape of the dataset after dealing with null values
maindf.shape

```

(3164, 7)

```

# EDA (Exploratory Data Analysis)
# Steps involved in EDA:
# Loading data -> Data Visualization -> Data Imputation -> Outlier
Analysis -> Gathering Insights

# Printing the start date and End date of the dataset
sd=maindf.iloc[0][0]
ed=maindf.iloc[-1][0]
print('Starting Date',sd)
print('Ending Date',ed)

Starting Date 9/17/2014
Ending Date 5/16/2023

```

```

# Analysis of year 2023
maindf['Date'] = pd.to_datetime(maindf['Date'], format='%m/%d/%Y')
y_2023 = maindf.loc[(maindf['Date'] >= '1/1/2023') & (maindf['Date'] <= '16/5/2023')]
y_2023.drop(['Adj Close', 'Volume'], axis=1)

```

```
<ipython-input-71-7fcf53a40962>:4: UserWarning:  
  Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was specified. This may lead to inconsistently parsed dates! Specify a format to ensure consistent parsing.
```

	Date	Open	High	Low	Close
3028	2023-01-01	16547.91406	16630.43945	16521.23438	16625.08008
3029	2023-01-02	16625.50977	16759.34375	16572.22852	16688.47070
3030	2023-01-03	16688.84766	16760.44727	16622.37100	16679.85742
3031	2023-01-04	16680.20508	16964.58594	16667.76367	16863.23828
3032	2023-01-05	16863.47266	16884.02148	16790.28320	16836.73633
...
3159	2023-05-12	26987.66211	27055.64648	25878.42969	26804.99023
3160	2023-05-13	26807.76953	27030.48242	26710.87305	26784.07813
3161	2023-05-14	26788.97461	27150.97656	26661.35547	26930.63867
3162	2023-05-15	26931.38477	27646.34766	26766.09766	27192.69336
3163	2023-05-16	27171.22656	27297.75000	26878.94727	27023.26172

136 rows × 5 columns

```
monthvise=  
y_2023.groupby(y_2023['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)  
monthvise
```

Date	Open	Close
January	20043.860132	20250.717490
February	23304.086008	23304.539203
March	24945.340412	25116.900895
April	28823.841731	28857.574544
May	28026.644166	27889.460450
June	NaN	NaN
July	NaN	NaN
August	NaN	NaN
September	NaN	NaN
October	NaN	NaN
November	NaN	NaN
December	NaN	NaN

```

# Bar graph between Open price & Close price
fig = go.Figure()
fig.add_trace(go.Bar(
    x=monthvise.index,
    y=monthvise['Open'],
    name='Open Price',
    marker_color='medium blue'
))

fig.add_trace(go.Bar(
    x=monthvise.index,
    y=monthvise['Close'],
    name='Close Price',
    marker_color='red'
))

fig.update_layout(barmode='group', xaxis_tickangle=0,
                  title='2023 Monthwise comparision between Open price and Close price')

fig.show()

```



```

# Bar graph between High price & Low price
y_2023.groupby(y_2023['Date'].dt.strftime('%B'))['Low'].min()
monthvise_high =
y_2023.groupby(maindf['Date'].dt.strftime('%B'))['High'].max()
monthvise_high = monthvise_high.reindex(new_order, axis=0)

monthvise_low =
y_2023.groupby(y_2023['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='High Price',
    marker_color='crimson'
)

```

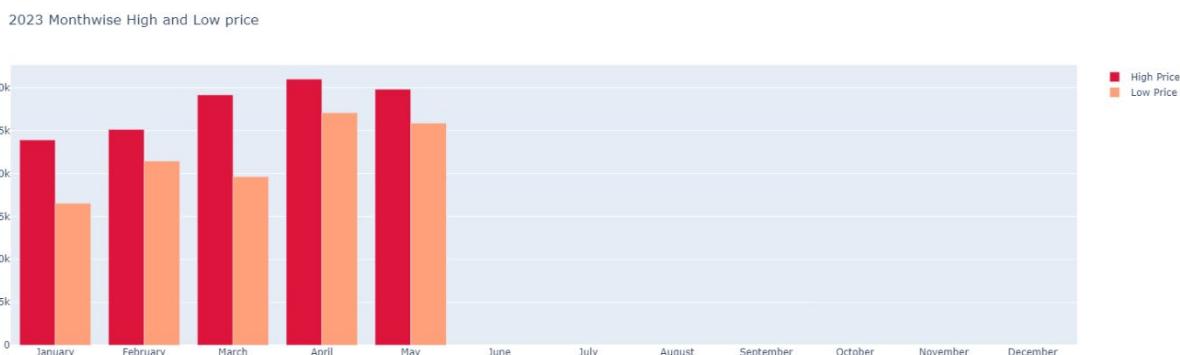
```

)) 

fig.add_trace(go.Bar(
    x=monthvise_low.index,
    y=monthvise_low,
    name='Low Price',
    marker_color='lightsalmon'
))

fig.update_layout(barmode='group',
                  title='2023 Monthwise High and Low price')
fig.show()

```



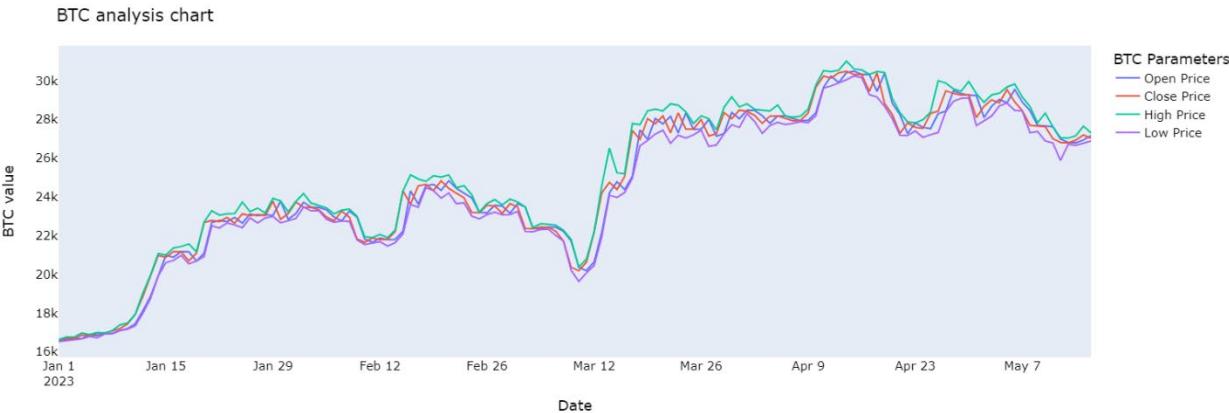
```

names = cycle(['Open Price', 'Close Price', 'High Price', 'Low Price'])

fig = px.line(y_2023, x=y_2023.Date, y=[y_2023['Open'],
y_2023['Close'],
y_2023['High'],
y_2023['Low']],
               labels={'Date': 'Date', 'value':'BTC value'})
fig.update_layout(title_text='BTC analysis chart', font_size=15,
font_color='black', legend_title_text='BTC Parameters')
fig.for_each_trace(lambda t: t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)

fig.show()

```



```
# Building LSTM Model
# Steps:
# 1st step: Preparing Data for training and testing
# 1 year Data (2022)
```

```
# Lets First Take all the Close Price
closedf = maindf[['Date','Close']]
print("Shape of close dataframe:", closedf.shape)
```

Shape of close dataframe: (3164, 2)

```
fig = px.line(closedf, x=closedf.Date,
y=closedf.Close,labels={'date':'Date','close':'Close BTC'})
fig.update_traces(marker_line_width=2, opacity=0.8,
marker_line_color='orange')
fig.update_layout(title_text='Whole period of timeframe of Bitcoin
close price 2014-2022', plot_bgcolor='white',
font_size=15, font_color='black')
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```



```
# Take data of 1 year
closedf = closedf[closedf['Date'] >= '5/16/2022']
close_BTC = closedf.copy()
print("Total data for prediction: ",closedf.shape[0])
```

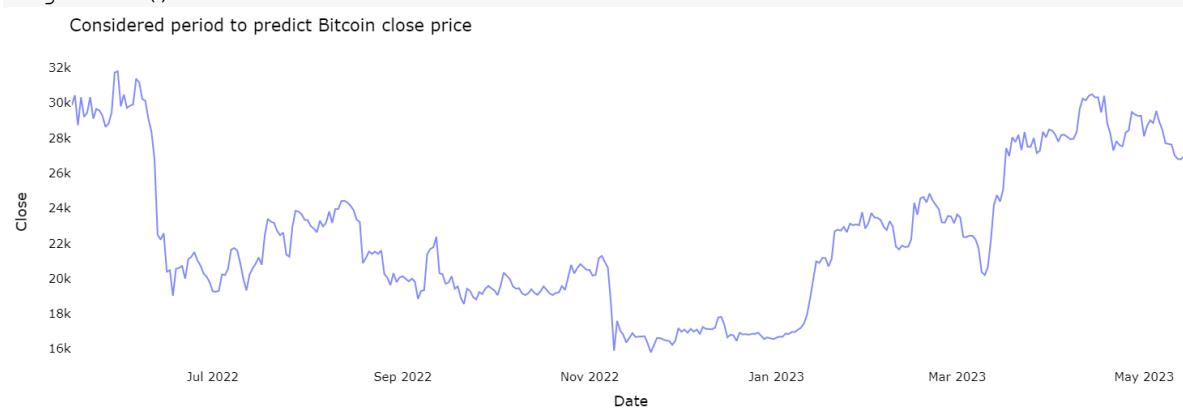
```
Total data for prediction: 366
```

```
closeddf
```

	Date	Close
2798	2022-05-16	29862.91797
2799	2022-05-17	30425.85742
2800	2022-05-18	28720.27148
2801	2022-05-19	30314.33398
2802	2022-05-20	29200.74023
...
3159	2023-05-12	26804.99023
3160	2023-05-13	26784.07813
3161	2023-05-14	26930.63867
3162	2023-05-15	27192.69336
3163	2023-05-16	27023.26172

366 rows × 2 columns

```
fig = px.line(closeddf, x=closeddf.Date,
y=closeddf.Close, labels={'date':'Date', 'close':'Close BTC'})
fig.update_traces(marker_line_width=2, opacity=0.8,
marker_line_color='orange')
fig.update_layout(title_text='Considered period to predict Bitcoin
close price',
plot_bgcolor='white', font_size=15,
font_color='black')
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```



```

# Normalizing Data (Chuẩn hóa dữ liệu)
# Goal: change the values of numeric columns in the dataset to use a
common scale, without distorting differences in the ranges of values or
losing information
# MinMaxScaler: For each value in a feature, (MinMaxScaler - min)/range
# range: the difference between the original maximum and original
minimum
# Preserve: the shape of the original distribution

# Deleting date column and normalizing using MinMax Scaler
del closedf['Date']
scaler=MinMaxScaler(feature_range=(0,1))
closedf=scaler.fit_transform(np.array(closedf).reshape(-1,1))
print(closedf.shape)

# We keep the training set as 60% and 40% testing set
training_size=int(len(closedf)*0.60)
test_size=len(closedf)-training_size
train_data,test_data=closedf[0:training_size,:],closedf[training_size:len(closedf),:1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)

train_data: (219, 1)
test_data: (147, 1)

# Transform the close price based on Time-series-analysis forecasting
requirement
# Take 15

# Convert an array of values into a dataset matrix
def create_dataset(dataset, time_step = 1):
    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)
time_step = 15
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)

X_train: (203, 15)
y_train: (203,)
X_test: (131, 15)
y_test (131,)

```

```

# 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: (203, 15, 1)
X_test: (131, 15, 1)

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

# Train
history =
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=200,batch_size=32,verbose=1)

```

▶ Epoch 176/200
 ⏵ Epoch 177/200
 ⏵ Epoch 178/200
 ⏵ Epoch 179/200
 ⏵ Epoch 180/200
 ⏵ Epoch 181/200
 ⏵ Epoch 182/200
 ⏵ Epoch 183/200
 ⏵ Epoch 184/200
 ⏵ Epoch 185/200
 ⏵ Epoch 186/200
 ⏵ Epoch 187/200
 ⏵ Epoch 188/200
 ⏵ Epoch 189/200
 ⏵ Epoch 190/200
 ⏵ Epoch 191/200
 ⏵ Epoch 192/200
 ⏵ Epoch 193/200
 ⏵ Epoch 194/200
 ⏵ Epoch 195/200
 ⏵ Epoch 196/200

```

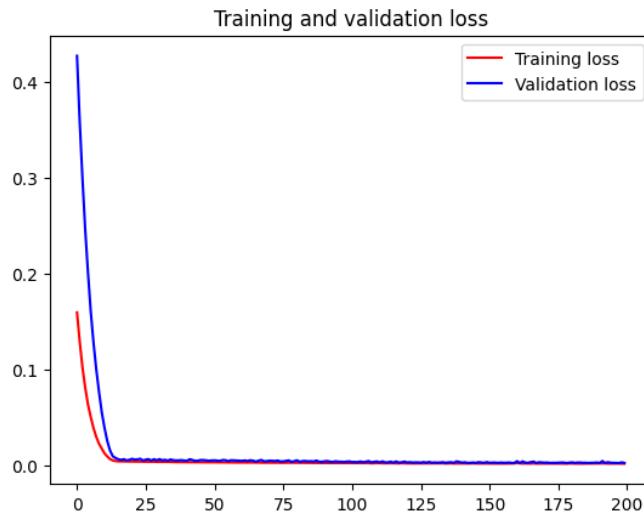
# Plotting Loss and Validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']

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()

```



```

# Make the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
train_predict.shape, test_predict.shape

7/7 [=====] - 0s 4ms/step
5/5 [=====] - 0s 4ms/step
((203, 1), (131, 1))

```

```

# Model Evaluation
# 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))

```

```

# Evaluation metrics RMSE, MSE and MAE
print("Train data RMSE: ",
math.sqrt(mean_squared_error(original_ytrain,train_predict)))
print("Train data MSE: ",
mean_squared_error(original_ytrain,train_predict))
print("Train data MAE: ",
mean_absolute_error(original_ytrain,train_predict))
print("-----")
print("Test data RMSE: ",
math.sqrt(mean_squared_error(original_ytest,test_predict)))
print("Test data MSE: ",
mean_squared_error(original_ytest,test_predict))
print("Test data MAE: ",
mean_absolute_error(original_ytest,test_predict))

Train data RMSE: 750.801105350237
Train data MSE: 563702.2997951376
Train data MAE: 496.977774627463
-----
Test data RMSE: 870.062806351166
Test data MSE: 757009.2869956667
Test data MAE: 613.0081854198472

```

```

# Variance Regression Score
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: 0.9456718596730026
Test data explained variance regression score: 0.9463606508117169

```

```

# R square score for regression
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: 0.945609373525394
Test data R2 score: 0.9379818229140873

```

```

# Regression Loss Mean Gamma deviance regression loss (MGD) and Mean
Poisson deviance regression loss (MPD)
print("Train data MGD: ", mean_gamma_deviance(original_ytrain,
train_predict))
print("Test data MGD: ", mean_gamma_deviance(original_ytest,
test_predict))
print("-----")
print("Train data MPD: ", mean_poisson_deviance(original_ytrain,
train_predict))
print("Test data MPD: ", mean_poisson_deviance(original_ytest,
test_predict))

```

```

# Comparision of original Bitcoin close price and predicted close price
# shift train predictions for plotting
look_back=time_step
trainPredictPlot = np.empty_like(closedf)
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(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] =
test_predict
print("Test predicted data: ", testPredictPlot.shape)

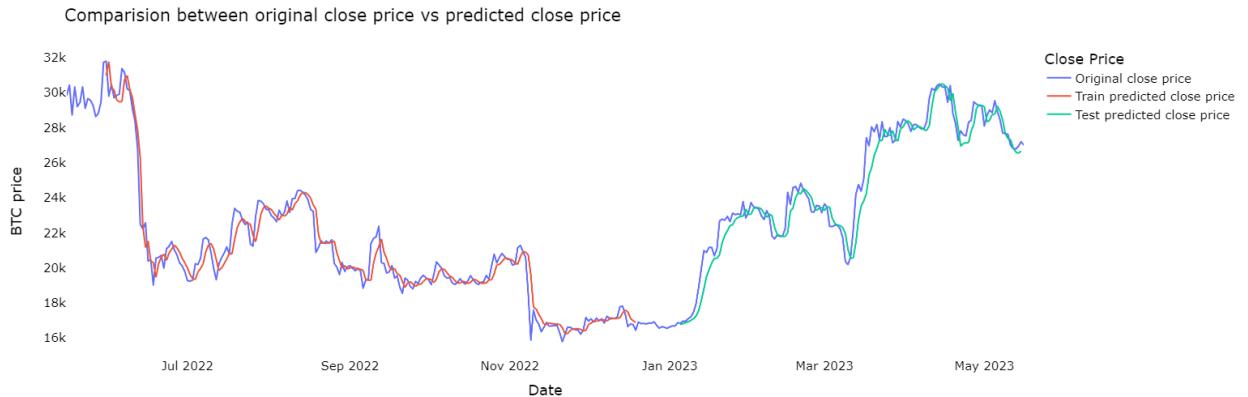
names = cycle(['Original close price','Train predicted close
price','Test predicted close price'])

plotdf = pd.DataFrame({'date': close_BTC['Date'],
                       'original_close': close_BTC['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':'BTC 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()

```



```
# Predicting next 30 days
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:])
        #print("{} day input {}".format(i,x_input))
        x_input = x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))

        yhat = model.predict(x_input, verbose=0)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)

        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

```

# Plotting last 15 days of dataset and next predicted 30 days
last_days=np.arange(1,time_step+1)
day_pred=np.arange(time_step+1,time_step+pred_days+1)
print(last_days)
print(day_pred)

[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
[16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
 40 41 42 43 44 45]

predicted = np.empty((len(last_days)+pred_days+1,1))
predicted[:] = np.nan
predicted = predicted.reshape(1,-1).tolist()[0]

last_original_days_value = predicted
next_predicted_days_value = predicted

last_original_days_value[1:time_step+1] =
scaler.inverse_transform(closedf[len(closedf)-time_step -
1:]).reshape(1,-1).tolist()[0]
next_predicted_days_value[time_step+1:] =
scaler.inverse_transform(np.array(lst_output).reshape(
1,1)).reshape(1,-1).tolist()[0]

# print(last_original_days_value)
# print(next_predicted_days_value)

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})

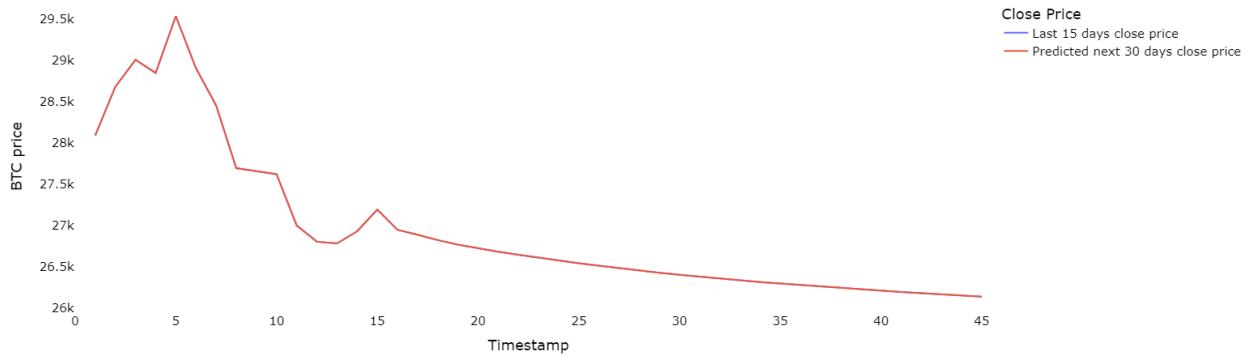
names = cycle(['Last 15 days close price','Predicted next 30 days close
price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index,
y=[new_pred_plot['last_original_days_value'],
new_pred_plot['ne
xt_predicted_days_value']],
labels={'value': 'BTC price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 30 days',
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()

```

Compare last 15 days vs next 30 days



```
# Plotting entire closing BTC price with next 30 days period of prediction
lstmdf=closedf.tolist()
lstmdf.extend((np.array(lst_output).reshape(-1,1)).tolist())
lstmdf=scaler.inverse_transform(lstmdf).reshape(1,-1).tolist()[0]

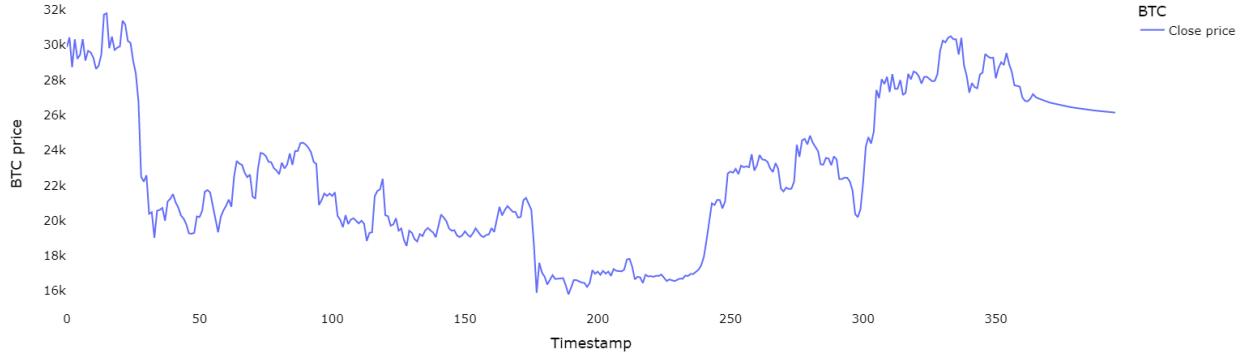
names = cycle(['Close price'])

fig = px.line(lstmdf,labels={'value': 'BTC price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting 2023 whole closing BTC price in 30 days with prediction',
                  plot_bgcolor='white', font_size=15,
font_color='black',legend_title_text='BTC')

fig.for_each_trace(lambda t: t.update(name = next(names)) )

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

Plotting 2023 whole closing BTC price in 30 days with prediction



GUI

```
# First we will import the necessary Library
import os
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np
import math
import datetime as dt

# For Evaluation we will use these library
from sklearn.metrics import mean_squared_error, mean_absolute_error,
explained_variance_score, r2_score
from sklearn.metrics import mean_poisson_deviance, mean_gamma_deviance,
accuracy_score
from sklearn.preprocessing import MinMaxScaler

# For model building we will use these library
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense, Dropout, LSTM

# For Plotting we will use these library
import matplotlib.pyplot as plt
from itertools import cycle
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

import yfinance as yfin
import streamlit as st
from datetime import datetime
from sklearn.preprocessing import MinMaxScaler
from PIL import Image

# Load our dataset
# Note it should be in same dir
df = pd.read_csv(r'C:\Users\nguye\PycharmProjects\BTC Prediction
Price_Final\BTC-USD 2014-2023.csv')
df['Date'] = pd.to_datetime(df['Date'], format='%m/%d/%Y')
print('Total number of days present in the dataset: ', df.shape[0])
print('Total number of fields present in the dataset: ', df.shape[1])

# Visualizations
st.subheader("Closing price vs time chart")
fig = plt.figure(figsize=(12, 6))
plt.plot(df.Close)
st.pyplot(fig)

st.subheader("Closing price vs time chart with MA100")
ma100 = df.Close.rolling(100).mean()
fig = plt.figure(figsize=(12, 6))
plt.plot(ma100)
```

```

plt.plot(df.Close)
st.pyplot(fig)

st.subheader("Closing price vs time chart with MA100 & MA200")
ma100 = df.Close.rolling(100).mean()
ma200 = df.Close.rolling(200).mean()
fig = plt.figure(figsize=(12, 6))
plt.plot(ma100)
plt.plot(ma200)
plt.plot(df.Close)
st.pyplot(fig)

#



# Building LSTM Model
# Steps:
# 1st step: Preparing Data for training and testing
# 1year Data (2022)

# First Take all the Close Price
closedf = df[['Date', 'Close']]
print("Shape of close dataframe:", closedf.shape)

# Take data of 1 year
closedf = closedf[closedf['Date'] >= '5/16/2022']
close_BTC = closedf.copy()
print("Total data for prediction: ", closedf.shape[0])

# Normalizing Data (Chuẩn hóa dữ liệu)
# Goal: change the values of numeric columns in the dataset to use a common
# scale, without distorting differences in the ranges of values or losing
# information MinMaxScaler: For each value in
# a feature, (MinMaxScaler - min)/range range: the difference between the
# original maximum and original minimum
# Preserve: the shape of the original distribution

# Deleting date column and normalizing using MinMax Scaler
del closedf['Date']
scaler = MinMaxScaler(feature_range=(0, 1))
closedf = scaler.fit_transform(np.array(closedf).reshape(-1, 1))
print(closedf.shape)

# We keep the training set as 60% and 40% testing set
training_size = int(len(closedf) * 0.60)
test_size = len(closedf) - training_size
train_data, test_data = closedf[0:training_size, :],
closedf[training_size:len(closedf), :1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)

# Transform the close price based on Time-series-analysis forecasting
requirement
# Take 15
# Convert an array of values into a dataset matrix

```

```

def create_dataset(dataset, time_step=1):
    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)

time_step = 15
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)

# Load model
model = load_model(r"C:\Users\nguye\PycharmProjects\BTC Prediction
Price_Final\BTC Prediction Price.h5")

# Make the prediction and check performance metrics
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
print(train_predict.shape, test_predict.shape)

# Model Evaluation
# 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))

# Evaluation metrices RMSE, MSE and MAE
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,
train_predict)))
print("Train data MSE: ", mean_squared_error(original_ytrain, train_predict))
print("Train data MAE: ", mean_absolute_error(original_ytrain,
train_predict))
print("-----")
print("Test data RMSE: ", math.sqrt(mean_squared_error(original_ytest,
test_predict)))
print("Test data MSE: ", mean_squared_error(original_ytest, test_predict))
print("Test data MAE: ", mean_absolute_error(original_ytest, test_predict))

# Variance Regression Score
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))

# R square score for regression
print("Train data R2 score:", r2_score(original_ytrain, train_predict))
print("Test data R2 score:", r2_score(original_ytest, test_predict))

# Regression Loss Mean Gamma deviance regression loss (MGD) and Mean Poisson
deviance regression loss (MPD)
print("Train data MGD: ", mean_gamma_deviance(original_ytrain,
train_predict))
print("Test data MGD: ", mean_gamma_deviance(original_ytest, test_predict))
print("-----")
print("Train data MPD: ", mean_poisson_deviance(original_ytrain,
train_predict))
print("Test data MPD: ", mean_poisson_deviance(original_ytest, test_predict))

#





---




---


# Comparision of original Bitcoin close price and predicted close price
# shift train predictions for plotting
look_back = time_step
trainPredictPlot = np.empty_like(closedf)
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(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict) + (look_back * 2) + 1:len(closedf) - 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_BTC['Date'],
                      'original_close': close_BTC['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': 'BTC 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()

# Predicting next 30 days
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:])
        # print("{} day input {}".format(i,x_input))
        x_input = x_input.reshape(1, -1)
        x_input = x_input.reshape((1, n_steps, 1))

        yhat = model.predict(x_input, verbose=0)
        # print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input = temp_input[1:]
        # print(temp_input)

        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))

# Plotting last 15 days of dataset and next predicted 30 days
last_days = np.arange(1, time_step + 1)
day_pred = np.arange(time_step + 1, time_step + pred_days + 1)
print(last_days)
print(day_pred)

predicted = np.empty((len(last_days) + pred_days + 1, 1))
predicted[:] = np.nan
predicted = predicted.reshape(1, -1).tolist()[0]

last_original_days_value = predicted
next_predicted_days_value = predicted

last_original_days_value[1:time_step + 1] = \
    scaler.inverse_transform(closedf[len(closedf) - time_step - 1:]).reshape(1, -1).tolist()[0]
next_predicted_days_value[time_step + 1:] = \
    scaler.inverse_transform(np.array(lst_output).reshape(-1, 1)).reshape(1, -1).tolist()[0]

```

```

# print(last_original_days_value)
# print(next_predicted_days_value)

new_pred_plot = pd.DataFrame({
    'last_original_days_value': last_original_days_value,
    'next_predicted_days_value': next_predicted_days_value
})

names = cycle(['Last 15 days close price', 'Predicted next 30 days close
price'])

fig = px.line(new_pred_plot, x=new_pred_plot.index,
y=[new_pred_plot['last_original_days_value'],

new_pred_plot['next_predicted_days_value']],
    labels={'value': 'BTC price', 'index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 30 days',
                    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()

# Plotting entire closing BTC price with next 30 days period of prediction
lstmdf = closedf.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': 'BTC price', 'index': 'Timestamp'})
fig.update_layout(title_text='Plotting 2023 whole closing BTC price in 30
days with prediction',
                    plot_bgcolor='white', font_size=15, font_color='black',
legend_title_text='BTC')

fig.for_each_trace(lambda t: t.update(name=next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()

# Final Graph
st.subheader("Predictions vs Original")
fig2 = plt.figure(figsize=(12, 6))
plt.plot(original_ytest, "b", label="Original price")
plt.plot(test_predict, "r", label="Predicted price")
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
st.pyplot(fig2)
plt.show()

```