การทำนายราคา Cryptocurrency Coin_Cardano

โดยใช้ Long Short-Term Memory (LSTM)

จัดทำโดย

นาย ณัฐกิตติ์ หงษ์ทอง 6204101315

นาย เดชาวัต ยุติธรรม 6204101317

เสนอ

อาจารย์ ทัศนีย์ ไชยา

รายงานเล่มนี้เป็นส่วนหนึ่งของวิชา คพ 345 ระบบเหมืองข้อมูล

เทอม 1 ชั้นปีที่ 3 สาขาวิชาวิทยาการคอมพิวเตอร์มหาวิทยาลัยแม่โจ้

สารบัญ

บทนำ

Cardano คือ แพลตฟอร์มสมาร์ทคอนแทรคท์ที่ถูกสร้างขึ้นในปี 2015 เพื่อที่จะเปลี่ยนแปลงและพัฒนา Cryptocurrency ให้มีรูปแบบที่แตกต่างไปจากเดิม ทางผู้พัฒนากล่าวว่า Cardano เป็นบล็อกเชนเจเนอเรชันที่ 3 ที่ระบุและแก้ไขปัญหาหลักๆ ที่เกิดขึ้นในบล็อกเชนยุคก่อนๆ ซึ่งประกอบไปด้วย ความสามารถในการเพิ่มขยาย (Scalability), ความสามารถในการทำงานร่วมกัน (Interoperability) และ ความยั่งยืน (Sustainability) ผ่านสถาปัตยกรรมแบบเป็นลำดับชั้น (layered architecture) โดยปกติแล้ว White paper ทั่วไปจะประกอบไปด้วยโค้ด (Code) เป็นหลัก ซึ่งจะแตกต่างจากของ Cardano ตรงที่มันจะประกอบไปด้วย แนวคิดของหลักการดีไซน์, การดำเนินงานให้เกิดประโยชน์สูงสุด และ เปิดโอกาสให้ผู้คนสามารถค้นคว้าเพิ่มเติมได้เพื่อสร้างสรรค์สิ่งใหม่ๆ ได้ ซึ่งก็ถือได้ว่าเป็น Cryptocurrency ตัวแรกๆเลยที่สร้างขึ้นมาจากแนวคิดปรัชญาที่อิงหลักการทางวิทยาศาสตร์และประกอบไปด้วยงานวิจัยทางวิชาการจากผู้เชี่ยวชาญทั้งหลายเข้าด้วยกัน

ราคาของเหรียญ Cardano มีความผันผวนตามความต้องการของมนุษย์ ขึ้น-ลง ตามความต้องการของช่วงเวลานั้นๆ ซึ่งเราไม่สามารถคาดการณ์ได้ด้วยความคิดส่วนตัว แต่เราคิดว่าการทำนายการ ขึ้น-ลง ของราคาเหรียญ Cardano นั้นสามารถทำได้ด้วยการวิเคราะห์จากสถิติย้อนหลัง เพื่อคาดการณ์ของราคาเหรียญ Cardano ที่จะเกิดขึ้นในอนาคตได้ เราจึงศึกษาการสร้างโมเดลโดยใช้ LSTM (Long Short-Term Memory) เพราะเป็นโครงข่ายประสาทเทียมแบบหนึ่งที่ถูกออกแบบมาสำหรับการประมวลผลลำดับ (sequence)

ดังนั้น ในการศึกษาการคาดการณ์ของราคาเหรียญ Cardano ในครั้งนี้จะทำการสร้างโมเดลโดยใช้ LSTM เพื่อทำการคาดการณ์ราคาของเหรียญ Cardano ที่จะเกิดขึ้นในอนาคต

วัตถุประสงค์

- เพื่อทำการศึกษาการสร้างโมเดลโดยใช้ LSTM (Long Short-Term Memory)

- เพื่อกระตุ้นให้เกิดความสนใจ ในด้านการลงทุนในตลาด Cryptocurrency

- เพื่อช่วยเพิ่มการตัดสินใจและลดความเสี่ยงที่จะเกิดขึ้น

LSTM

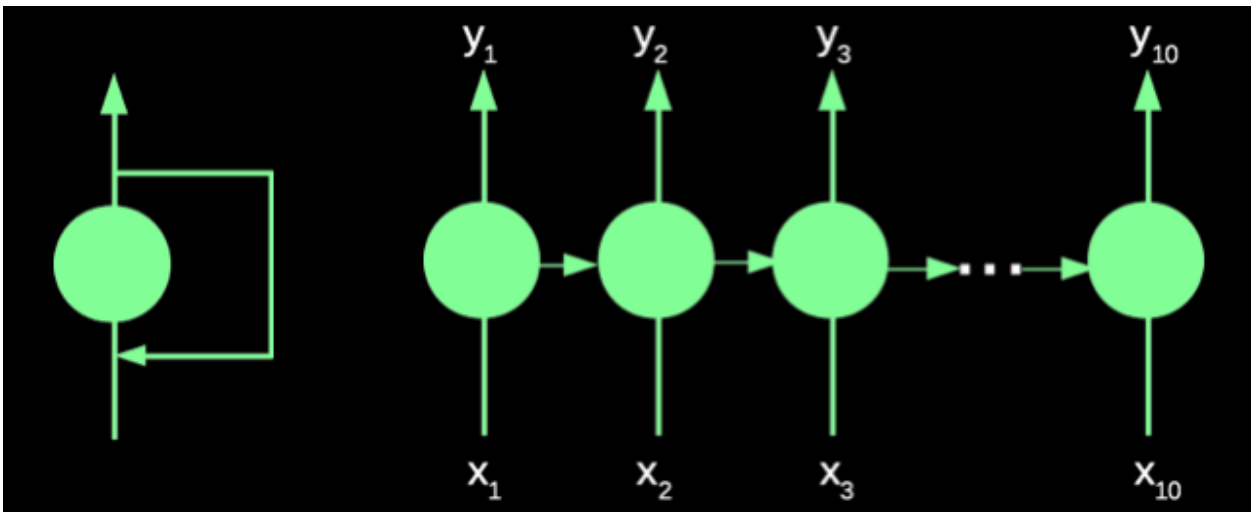
ชื่อเต็มคือ Long Short-Term Memory เป็นโครงข่ายประสาทเทียมแบบหนึ่งที่ถูกออกแบบมาสำหรับการประมวลผลลำดับ (sequence)

LSTM ถูกนำเสนอมาตั้งนานแล้วแต่พึ่งมาได้รับความนิยมไม่นานมานี้ หนึ่งในเหตุผล (อย่างน้อยสำหรับผม) คือมันดูวุ่นวาย เข้าใจยาก บทความแนะนำ LSTM มักจะเน้นไปที่ว่ามันทำงานอย่างไร แต่ในส่วนว่าทำไมมันถึงต้องเป็นเช่นนั้นนั้นมักถูกซ่อนไว้ในคำอธิบายที่ไม่ชัดเจนหรือรูปวาด ที่โดยส่วนตัวแล้วผมดูไม่ค่อยรู้เรื่อง

วันก่อนหยิบ LSTM มาอ่านใหม่แล้วรู้เรื่องมากขึ้น เลยมาจดไว้หน่อย

RNN พื้นฐานและ gradient vanishing

ก่อนอื่น LSTM นั้นจัดว่าเป็นโครงข่ายประเภท Recurrent Neural Network (RNN) นั่นคือ NN ที่มีการนำเอา output ของมันเองก่อนหน้านี้กลับมาใช้ใหม่เช่นรูปซ้ายข้างล่างนี้



รูปซ้ายนั้นคือตัวอย่าง RNN ที่มี 1 layer และ 1 node ใน layer นั้น เมื่อเราเอา RNN นี้ไปใช้ประมวลผลลำดับ (x(1),y(1)),...,(x(10),y(10)) x(t) คือข้อมูลนำเข้า ณ เวลา t และ y(t) คือค่าส่งออกที่เราต้องการ เช่น x(1),...,x(10) อาจจะเป็นลำดับคำในภาษาไทย ส่วน y(1),...,y(10) เป็น part of speech ของคำเหล่านี้เป็นต้น หรืออาจจะเป็นงาน sequence-to-sequence อื่นก็ได้

ในการปรับ RNN ข้างซ้ายจากข้อมูลลำดับนี้เราจะต้องทำการกางมันออกตามรูปขวา

จากนั้นขั้นตอนการ train ก็ใช้ gradient descent ตามปกติ โดย gradient ก็ได้จาก backprop มาตรฐาน

สิ่งที่ควรรู้คือการทำ backprop นั้นขึ้นกับ input และ output เช่นเราสามารถพิจารณา input x(1)และ backprop gradient กลับมาจาก y(1) หรือ y(2) หรือกระทั้ง y(10)

สิ่งที่ต่างกันคือหากเรา backprop จาก y(1) ไป x(1) ข้อมูล gradient นั้นจะผ่านเพียง layer เดียว หากเรา backprop จาก y(2) ไป x(1) ก็ต้องผ่าน 2 layers หรือจาก y(10) ไป x(1) ก็ผ่าน 10 layers

ปัญหาที่เกิดเมื่อเราส่ง gradient ผ่านหลายๆ layer คือ “ขนาด” หรือ amplitude ของมันจะลดลง ทำให้สุดท้ายแล้วค่าที่ได้มานั้นเล็กมาก เมื่อนำมาใช้กับ learning rate ที่ปกติก็เล็กอยู่แล้วจะทำให้ weights ของ layer ที่เราพิจารณานั้นแทบไม่ถูกปรับเลย

เราเรียกปัญหานี้ว่า gradient vanishing

ควรทราบว่า gradient vanishing นั้นเกิดได้เมื่อเราสร้างโครงข่ายปกติอย่าง multi-layer Perceptron (MLP) แต่ใช้หลายชั้น เช่นกัน ในกรณีนั้น weights ของ layer ล่างๆ ก็จะไม่ถูกปรับ ค่า weights ที่ได้จากการ train ก็คือค่าที่ random ไว้ตั้งแต่ต้น ดังนั้นผลที่ได้จึงให้ผลที่ไม่ดี

กระบวนการทำ pre-training ที่ค่อยๆ สร้างทีละ layer โดยใช้ unsupervised criteria นั้นถึงจะไม่ได้แก้ gradient vanishing โดยตรงแต่ช่วยให้ weights ที่ได้นั้นอย่างน้อยมีคุณภาพดีกว่าการ random ปกติ

สำหรับ RNN นั้นสถาณการณ์ต่างออกไปบ้างนั่นเพราะหากเรา backprop จาก y(t) ไป x(t) มันก็แค่ชั้นเดียว ดังนั้นเราสามารถ train weights ได้อยู่แล้ว แต่การที่ gradient ที่ backprop จากตำแหน่งไกลๆ มันหายไปนั้นทำให้ RNN ที่สร้างไม่สามารถ capture long-term dependency ได้

แล้ว gradient vanishing เกิดได้อย่างไร? คำตอบคือมาจากกฎ chain rule ที่ใช้ใน backprop นี่แหละ

chain rule ที่เรียน ม.ปลาย คือถ้า $h(x) = f(g(x))$ แล้ว $h'(x) = f'(g(x))g'(x)$

สำหรับ NN แล้วการทำงานภายในแต่ละ node จบที่ activation function ที่มักเป็น non-linear function ดังนั้นค่า derivative ของมันนั้นมักจะน้อยกว่า 1 ดังนั้นเมื่อนำมาคูณต่อไปเรื่อยๆ ค่า gradient จึงเล็กลงเรื่อยๆ

Key หลักเพื่อให้ capture long-term dependency ได้ก็คือหาทางส่ง gradient กลับมาให้ได้มากที่สุด

Formulation พื้นฐานและการส่ง gradient

เพื่อประมวลผลลำดับ node ของ RNN ต้องมี หน่วยความจำภายใน ที่จำสิ่งที่เกิดขึ้นแล้วและใช้ในการตัดสินใจในเวลาถัดไป โดยหน่วยความจำภายในนี้ก็ต้องถูกปรับไปเรื่อยๆ ตามค่าของลำดับที่เราได้ประมวลผลมาเช่นกัน

ให้ h(t) เป็นหน่วยความจำภายใน ณ เวลา t

h(t) เองก็ต้องถูกปรับโดยใช้ 1) h(t-1), 2) ค่าส่งออกก่อนนี้ y(t-1) และ 3) ข้อมูลนำเข้า x(t)

สังเกตว่ามันก็เหมือนกับที่เราใช้ในการคำนวณค่าส่งออก y(t) ด้วย

นั่นคือ โดย concept แล้วแต่ละ node ต้องคำนวณ

$h(t) = f1(x(t), y(t-1), h(t-1))$

$y(t) = f2(x(t), y(t-1), h(t-1))$

โดย f1 และ f2 เป็นฟังก์ชันใดๆ หนึ่งในตัวอย่างที่เป็นไปได้คือ

$h(t) = h(t-1) + \tanh(U\, x(t) + W\, y(t-1))$

$y(t) = \tanh(h(t))$

(tanh นี้ apply กับทุก element ของ vector แยกกัน)

ตัวแปรที่เราต้องปรับของ node นี้คือเมตริกซ์ U และ W

Formulation นี้น่าสนใจเพราะการปรับหน่วยความจำภายใน h(t) นั้นทำแบบ linear หากดูตาม concept แล้วก็เหมือนกับเรามี 2 เส้นทางในการ backprop gradient ทางแรกคือผ่าน h(t-1) ที่ผ่านง่าย อีกทางหนึ่งคือผ่าน tanh ที่ต้องโดน derivative ตบลง

เส้นทางแรกนี้เองที่ทำให้เราสามารถส่ง gradient กลับมาได้ง่ายขึ้น ทำให้เราสามารถจับ long-term dependency ได้ดีขึ้น

Gates

นอกจากการเพิ่มเส้นทางนำ gradient แล้ว LSTM ยังเสนอแนวความคิดเพิ่มอีกคือ

ข้อมูลบางอย่างก็ควรจะลืมไปบ้าง

ข้อมูลบางอย่างก็เป็น noise ที่ไม่ควรนำมาพิจารณา

ข้อมูลบางอย่างอาจต้อง scale หรือ filter ก่อนส่งออก

แนวความคิดทั้ง 3 นี้นำไปสู่การเพิ่ม gates ต่างๆ คือ forget gate f, input gate i, และ output gate oให้กับ formulation ก่อนหน้านี้ นั่นคือ

$h(t) = f \otimes h(t-1) + i \otimes \tanh(U\, x(t) + W\, y(t-1))$

$y(t) = o \otimes \tanh(h(t))$

โดยสัญลักษณ์ $\otimes$ แทนการคูณกันของแต่ละ coordinate แยกกัน

และ gates ทั้ง 3 นี้ให้ค่าอยู่ในช่วง [0,1] ค่า 0 แปลว่าจะ ทำการลืม/ลบ input นั้นทิ้ง/ไม่ส่งค่านั้นออก หากค่าของ gates ทั้ง 3 เป็น 1 เราก็จะกลับไปที่ formulation ตั้งต้นข้างบน

Gates ทั้งสามนี้ยังสามารถตั้งให้เป็น function ที่สามารถถูก train ไปพร้อมๆ กับ node นี้ได้เช่น

$f = \text{logistic}(U_f\, x(t) + W_f\, y(t-1))$

$i = \text{logistic}(U_i\, x(t) + W_i\, y(t-1))$

$o = \text{logistic}(U_o\, x(t) + W_o\, y(t-1))$

เรายังสามารถอนุญาตให้ gates ต่างๆ เข้าถึงหน่วยความจำภายใน h(t) ได้อีก เช่นให้

$f = \text{logistic}(U_f\, x(t) + W_f\, y(t-1) + V_f\, h(t-1))$

$i = \text{logistic}(U_i\, x(t) + W_i\, y(t-1) + V_i\, h(t-1))$

$o = \text{logistic}(U_o\, x(t) + W_o\, y(t-1) + V_o\, h(t-1))$

รู้สึกว่า formulation นี้จะเรียกว่ามีการเพิ่ม peephole gate หรือช่องแอบดู ให้กับ LSTM มาตรฐาน

Code

```
import math
import pandas_datareader as web
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM

import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

ทำการ import library ที่จำเป็น

```
df= pd.read_csv("/content/drive/MyDrive/coin_Cardano1.csv")
df
```

	Date	High	Low	Open	Close	Volume	Marketcap
0	10/2/2017	0.030088	0.019969	0.024607	0.025932	5.764130e+07	6.288991e+08
1	10/3/2017	0.027425	0.020690	0.025757	0.020816	1.699780e+07	5.396927e+08
2	10/4/2017	0.022806	0.020864	0.020864	0.021931	9.000050e+06	5.686195e+08
3	10/5/2017	0.022154	0.020859	0.021951	0.021489	5.562510e+06	5.571390e+08
4	10/6/2017	0.021542	0.018360	0.021359	0.018539	7.780710e+06	4.806646e+08
...	...	...	...	...	...	...	...
1369	7/2/2021	1.394397	1.286607	1.332942	1.394397	2.159410e+09	4.454587e+10
1370	7/3/2021	1.441714	1.359664	1.394152	1.406836	2.028094e+09	4.494324e+10
1371	7/4/2021	1.493717	1.382153	1.404008	1.458184	1.806362e+09	4.658364e+10
1372	7/5/2021	1.461221	1.379284	1.461221	1.404898	1.759461e+09	4.488134e+10
1373	7/6/2021	1.456887	1.393282	1.404712	1.418053	1.477700e+09	4.530158e+10

1374 rows × 7 columns

ทำการสร้าง Data farm โดยการนำข้อมูลมาจาก coin_Cadano.csv และ แสดงข้อมูลของ coin_Cordano โดยใช้ pandas

```
df.shape
```

```
(1374, 7)
```

ทำการดูข้อมูลใน Data farm ของเราว่ามีกี่ แถว กี่ คอลัม

```
plt.figure(figsize=(16,8))
plt.title('Closing Price')
plt.plot(df['High'])
plt.xticks(range(0,df.shape[0],60),df['Date'].loc[::60],rotation=45)
plt.xlabel('Date', fontsize=18)
plt.ylabel('High Price', fontsize=18)
```

```
Text(0, 0.5, 'High Price')
```

ทำการ plot graph โดยขนาดของกราฟเท่ากับ 16 x 8 แสดง ราคาสูงสุดของ coin_Cadano ในแต่ละเดือน ตั้งแต่ วันที่ 2 เดือน กุมภาพันธ์ 2017 ถึง วันที่ 6 เดือน กรกฎาคม 2021 ดังรูป 1. จะเห็นได้ว่า ราคาที่สูงที่สุด จะอยู่ในช่วงประมาณ เดือน มีนาคม 2020 ถึง เดือน มิถุนายน 2021

รูป 1.

```
# ดึงเฉพาะราคาสูงสุดมา
data = df.filter(['High'])

#แปลงข้อมูลใน data ให้เป็น Numpy array
dataset = data.values

#แบ่งข้อมูลแบบ train 80, test 20
training_data_len = math.ceil(len(dataset) * 0.8)
#ดูว่า train 80% มีขนาดเท่าไร
training_data_len
```

```
1100
```

เราจะใช้ ราคาสูงสุด ดังนั้นเราจะทำการ สร้างตัวแปลขึ้นมาใหม่โดยดึงเฉพาะราคาสูงสุด ของแต่ละเดือน จาก Data farm นั้นคือ คอลัม High จากนั้นทำการแปลงข้อมูลให้เป็น Numpy array จากนั้น ทำการแบ่งข้อมูลเป็น train 80% teat 20% และดูว่าสิ่งที่เรา train 80% มีข้อมูลอยู่เท่าไร่ จากโค็ดเราจะได้ข้อมูลจำนวน 1100 แถว

```
# ปรับ Scaling
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
```

ทำ Normalization โดยการปรับ Scale ทุก Feature ให้เป็น [0, 1] ให้เก็บไว้ในตัวแปลที่ชื่อว่า scaled_data

```
# เลือก 80 % จาก scaled_data
train_data = scaled_data[:training_data_len, :]

x_train = []
y_train = []

# เลือกข้อมูลมา 60วัน(x_train) เพื่อทำนายวันถัดไป(y_train)
for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])

#จากนั้นแปลงกับให้เป็น Numpy array
x_train, y_train = np.array(x_train), np.array(y_train)
```

```
# reshape
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1))
```

เลือกข้อมูลจาก scaled_data มา 80% เพื่อ train จากนั้น เลือกข้อมูลมา 60 แถว (ให้เป็น x_train) เพื่อที่จะทำนายวันต่อไป(ให้เป็น y_train) ทำการแปลง x_train และ y_train ให้เป็น Numpy array และ ทำการเปลี่ยนเป็นอาเรย์ 3 มิติ

```
# init model ขึ้นมาโดยใช้ LSTM
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(1))

# Compile
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])
```

สร้าง LSTM ด้วย library ของ Keras โดยกำหนดขนาด hidden layer ให้เท่ากับ 50 ตั้งว่าเมื่อ return_sequences =True ให้กำหนดอาเรย์ของข้อมูลใน input shape ใหม่ (x_train.shape) แต่เมื่อ return_sequences = False ไม่ต้องเปลี่ยงแปลงอะไร จากนั้นสร้าง hidden layer ตามขนาด output หรือจำนวนวันที่ต้องการ ในที่นี้จะทำการทำนาย เราจะแค่หนึ่งวันเท่านั้น และทำการ compile ค่า loss และ accuracy

```
# ทำการ Train model
history = model.fit(x_train, y_train,  epochs=1,batch_size=1)
model.save("lstmCanano3")
```

```
1040/1040 [==============================] - 27s 26ms/step - loss: 0.1527 - accuracy: 0.0000e+00
WARNING:absl:Found untraced functions such as lstm_cell_8_layer_call_fn, lstm_cell_8_layer_call_and_
INFO:tensorflow:Assets written to: lstmCanano3/assets
INFO:tensorflow:Assets written to: lstmCanano3/assets
```

ทำการ train model 1 ครั้ง และทำการ Save model ไว้ใช้ในภาคหลัง

```
history = model.load_weights('/content/drive/MyDrive/lstmCanano3')
```

โหลด model ที่ทำการ Save ไว้มาใช้งาน

```
test_data = scaled_data[training_data_len - 60: , :]

x_test = []
y_test = dataset[training_data_len: :]

for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

ทำการ test model ที่สร้างมา เลือกข้อมูลมา 60 แถว (ให้เป็น x_test) เพื่อที่จะทำนายวันต่อไป(ให้เป็น y_trest) ทำการแปลง x_ trest)และ y_ trest) ให้เป็น Numpy array และ ทำการเปลี่ยนเป็นอาเรย์ 3 มิติ

```
# ทำนาย model
predictions = model.predict(x_test)

# transform scaler กลับเป็นค่าเดิม
predictions = scaler.inverse_transform(predictions)
```

```
# ใช้ metrics แบบ root mean squared error
rmse = np.sqrt(np.mean(predictions - y_test)**2)
rmse
```

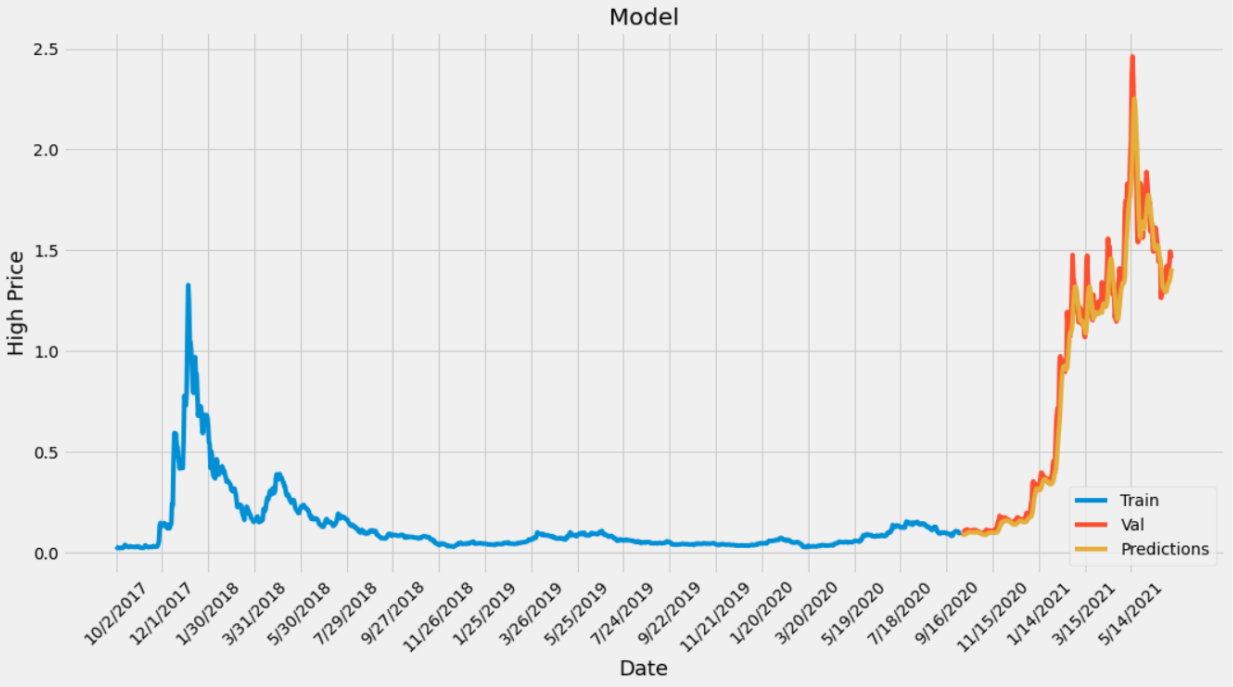
```
0.039807317339224775
```

ทำการ predict model จากนั้นทำการ transform scaler กลับเป็นค่าเดิม ในที่นี้เราจะใช้ metrics แบบ root mean squared error เพื่อหาค่า loss

```
# plot ค่าจริงกับค่าที่ทำนายไว้มาดู
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
valid['Date'] = df['Date']


plt.figure(figsize=(16,8))
plt.title('Model')
plt.xticks(range(0,df.shape[0],60),df['Date'].loc[::60],rotation=45)
plt.xlabel('Date', fontsize=18)
plt.ylabel('High Price', fontsize=18)
plt.plot(train['High'])
plt.plot(valid[['High', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'] , loc='lower right')
plt.show()
```

ทำการ plot กราฟ ที่เปรียบเทียบข้อมูลจริงกับข้อมูลที่เราทำนาย train คือข้อมูลจริงที่ไม่ได้นำมาเปรียบเทียบ Val คือ ข้อมูลจริงที่เราเปรียบเทียบ Prediction คือ ข้อมูลที่เราทำการทำนายไว้ ดังรูป 2. และทำการ plot ค่าจริงกับค่าทำนาย ดังรูป 3.



รูป 2.

```
valid
```

	High	Predictions	Date
1100	0.098498	0.090680	10/6/2020
1101	0.094325	0.090169	10/7/2020
1102	0.096698	0.089151	10/8/2020
1103	0.102475	0.088467	10/9/2020
1104	0.110540	0.088977	10/10/2020
...	...	...	...
1369	1.394397	1.340004	7/2/2021
1370	1.441714	1.347537	7/3/2021
1371	1.493717	1.364109	7/4/2021
1372	1.461221	1.392232	7/5/2021
1373	1.456887	1.409676	7/6/2021

274 rows × 3 columns

รูป 3.

```
# ดึง data มาอีกรอบเพื่ออ้างอิง
quote = pd.read_csv("/content/drive/MyDrive/coin_Cardano1.csv")

#สร้าง DataFrame มาใหม่
new_df = quote.filter(['High'])

# ดึงราคาสูงสุดจาก 60 วันล่าสุด
last_60_days = new_df[-60:].values

# Scale
last_60_days_scaled = scaler.transform(last_60_days)

X_test = []
X_test.append(last_60_days_scaled)
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

#predicted
pred_price = model.predict(X_test)

#Undo the scaling
pred_price = scaler.inverse_transform(pred_price)
print(pred_price)
```

```
[[1.4190297]]
```

ผลการทำนายวันถัดไป นั้นก็คือข้อมูลราคาของวันถัดไปหนึ่งวัน หรือ ในที่นี้ข้อมูลลำดับที่ก็ 1374 สามารถดูเปรียบเทียบได้จาก รูป 4.

```
quote2 = pd.read_csv("/content/drive/MyDrive/coin_Cardano1.csv")
quote2['High']
```

```
0       0.030088
1       0.027425
2       0.022806
3       0.022154
4       0.021542
          ...
1369    1.394397
1370    1.441714
1371    1.493717
1372    1.461221
1373    1.456887
Name: High, Length: 1374, dtype: float64
```

รูป 4.

อ่างอิง

ชื่อเรื่อง Cryptocurrency Historical Prices

ผู้เขียน SRK

https://www.kaggle.com/sudalairajkumar

ข้อมูล coin_Aave.csv // coin_Cadano.csv

https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory

ผู้เขียน Sanparith Marukatat

https://sanparithmarukatat.medium.com/

ชื่อเรื่อง LSTM

https://sanparithmarukatat.medium.com/lstm-%E0%B9%80%E0%B8%97%E0%B9%88%E0%B8%B2%E0%B8%97%E0%B8%B5%E0%B9%88%E0%B9%80%E0%B8%82%E0%B9%89%E0%B8%B2%E0%B9%83%E0%B8%88-75027db3167f

เว็บที่ใช้ศึกษา code

1. https://data-flair.training/blogs/stock-price-prediction-machine-learning-project-in-python/

2. https://medium.com/datawiz-th/%E0%B8%A1%E0%B8%B2%E0%B8%A5%E0%B8%AD%E0%B8%87-forecast-%E0%B8%A3%E0%B8%B2%E0%B8%84%E0%B8%B2%E0%B8%AB%E0%B8%B8%E0%B9%89%E0%B8%99%E0%B9%81%E0%B8%9A%E0%B8%9A%E0%B8%87%E0%B9%88%E0%B8%B2%E0%B8%A2%E0%B9%86-%E0%B8%94%E0%B9%89%E0%B8%A7%E0%B8%A2-deep-learning-lstm-python-305c480db223

3. https://ichi.pro/th/withi-kheiyn-khod-lstm-recurrent-neural-network-taw-raek-khxng-khun-ni-keras-251064361096373

เว็บที่ใช้ในการเขียนโปรแกรม

https://colab.research.google.com