ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเกษตรศาสตร์

เรื่อง

การวิเคราะห์และทำนายหุ้น Nvidia

จัดทำโดย

นางสาวเพ็ญพิชชา เพ็ชรสิงห์ 6814500175

ปีการศึกษา 2568

วิชา การเรียนรู้เชิงลึก (Deep Learning) (204466)

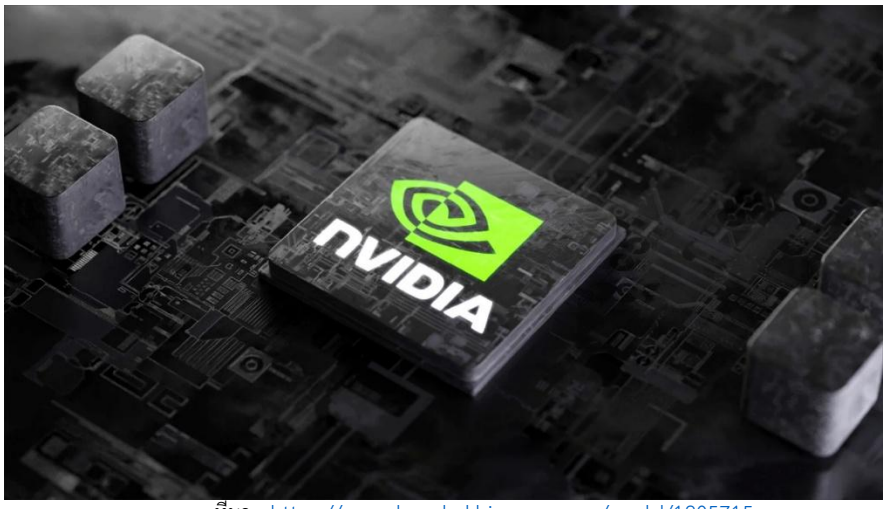
การวิเคราะห์และทำนายหุ้น Nvidia

ที่มา

NVIDIA Corporation (NVDA) ในปัจจุบันได้กลายเป็นบริษัทผู้ออกแบบชิปสำหรับการประมวลผลที่มีความซับซ้อน เป็นผู้ออกแบบ Graphics Processing Unit (GPU) ที่ถูกใช้งานในการพัฒนา AI Model มาอย่างต่อเนื่อง นอกจากนี้ ยังมีผลิตภัณฑ์ชิปการ์ดจอสำหรับอุตสาหกรรมเกม (Gaming) รวมถึงธุรกิจอื่น ๆ เช่น แพลตฟอร์ม Metaverse และยานยนต์ เป็นต้น

จากข่าวช่วงตลอด 2 ปีที่ผ่านมา (2023-2025) Nvidia คือหุ้นสหรัฐฯ ที่ปรับตัวขึ้นร้อนแรงที่สุดตัวหนึ่งจากกระแสการใช้งาน AI โดยเราได้เรียนวิชา Deep Learning ทำให้ทราบถึงจากการเติบโตของ AI ทำให้หุ้น Nvidia มีความน่าสนใจและน่าจับตามองของนักลงทุนในสายเทคโนโลยี AI

ในมุมมองระยะสั้นถึงระยะกลาง การลงทุนในหุ้น NVIDIA ยังคงต้องเผชิญกับความผันผวนที่อาจเกิดขึ้นจากปัจจัยภูมิรัฐศาสตร์ โดยเฉพาะผลกระทบจากมาตรการจำกัดการส่งออกไปยังประเทศจีน หุ้นที่มีความผันผวนสูงสามารถทำนายได้ยากเนื่องจากมีปัจจัยภายนอกต่างๆ เช่น การเมืองโลก ข่าวเศรษฐกิจต่างๆ



ที่มา : https://www.bangkokbiznews.com/world/1205715

การทำนายหุ้นมีรูปแบบที่ซับซ้อนและความสัมพันธ์ที่ไม่เป็นเชิงเส้น ความสัมพันธ์เชิงเวลา สามารถทำได้ยากในรูปแบบดั้งเดิม เช่น Linear Regression จึงเหมาะกับการใช้ Deep learning มาช่วยทำนาย โดยเราได้นำ LSTM มาใช้ซึ่งมากในการจัดการลำดับเวลาและความสัมพันธ์ระยะยาวได้ดี

ข้อดี

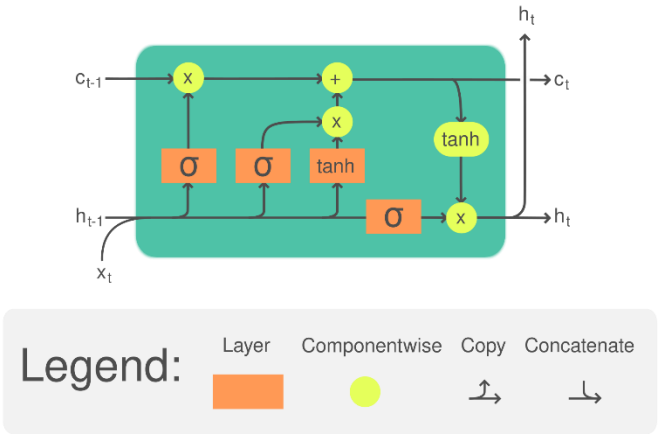
- Deep learning สามารถจับรูปแบบที่ซ้ำซ้อนได้ดี เช่น ลำดับเวลา
- LSTM สามารถจับลำดับและความสัมพันธ์เชิงเวลาได้ดี
- Deep learning สามารถรวบรวมข้อมูลหลายมิติได้

ข้อเสีย

- มีโอกาสเกิด overfitting สูง
- ต้องใช้ข้อมูลขนาดใหญ่
- ข้อมูลมี noise สูง

<u>สถาปัตยกรรมที่ใช้</u>

เราได้เลือก LSTM มาใช้ในการทำนาย LSTM ย่อจาก Long Short-Term Memory ถือเป็นประเภทหนึ่งของสถาปัตยกรรมแบบ Recurrent Neural Network (RNN) ถูกออกแบบให้จดจำ Patterns ในช่วงเวลานานๆ มีประสิทธิภาพสำหรับปัญหาการทำนายที่เป็น Sequential เนื่องจากสามารถเก็บข้อมูลก่อนหน้าและนำมาร่วมใช้ในการประมวลผลได้ สามารถแก้ปัญหา Long-term Dependency ได้ โดย RNN แบบดั้งเดิมจะเผชิญกับ ความท้าทายในเรื่อง Long-range Dependency และมีปัญหา Vanishing Gradient LSTM ถูกออกแบบมาให้จดจำ Long-term Information โดยใช้ Gating Mechanisms



ที่มา : https://www.nerd-data.com/deep_learning_lstm/

โดยโมเดลที่เราใช้ ประกอบด้วย 2 ชั้นซ้อนกัน (stacked LSTM) โดยแต่ละชั้นมี 64 โหนด ใช้ฟังก์ชันกระตุ้นแบบ tanh และ sigmoid ภายใน LSTM และใช้ ReLU ก่อนชั้นเชิงเส้นสุดท้ายเพื่อเพิ่มความไม่เชิงเส้นของโมเดล ผลลัพธ์สุดท้ายได้จากชั้น Linear ที่มี 1 โหนด (สำหรับพยากรณ์ราคาหุ้น 1 ค่า) รวมพารามิเตอร์ที่เรียนรู้ได้ทั้งหมดประมาณ 35,000 ค่า

ลำดับ	LAYER	INPUT SHAPE	OUTPUT SHAPE	PARAMETERS	ACTIVATION FUNCTION
1	LSTM layer 1	(batch, seq_len, 1)	(batch, seq_len, 64)	4×64×(1 + 64 + 2) ≈ 17,664	tanh และ sigmoid
2	LSTM layer 2	(batch, seq_len, 64)	(batch, seq_len, 64)	2 layer = 33,280	tanh และ sigmoid
3	Dropout	(batch, seq_len, 64)	(batch, seq_len, 64)	0	-
4	ReLU	(batch, 64)	(batch, 64)	0	ReLU
5	Linear	(batch, 64)	(batch, 1)	weight = 64 × 1 = 64 bias = 1 64+1 = 65	-

Code PyTorch

import libraries ที่ใช้

```
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

```
import torch.optim as optim
import torch.utils.data as data
```

import dataset จาก https://www.kaggle.com/ โดยเราใช้ dataset NVIDIA Stocks Data 2025 โดยข้อมูลประกอบด้วยข้อมูลหุ้น ตั้งแต่ปี 1999 - 2025

```
#import data API key from kaggle
import opendatasets as od
od.download("https://www.kaggle.com/datasets/meharshanali/nvidia-stocks-data-2025?select=NVDA.csv")
data = pd.read_csv("/content/nvidia-stocks-data-2025/NVDA.csv")
```

data.head()

	Date	Adj Close	Close	High	Low	Open	Volume
0	1999-01-22	0.037615	0.041016	0.048828	0.038802	0.043750	2714688000
1	1999-01-25	0.041556	0.045313	0.045833	0.041016	0.044271	510480000
2	1999-01-26	0.038331	0.041797	0.046745	0.041146	0.045833	343200000
3	1999-01-27	0.038212	0.041667	0.042969	0.039583	0.041927	244368000
4	1999-01-28	0.038092	0.041536	0.041927	0.041276	0.041667	227520000

เลือกข้อมูลที่จะใช้ โดยเลือกตั้งแต่ 2021 – 2025 ที่มีข้อมูลล่าสุด และตรวจสอบว่าข้อมูลครบถ้วน ไม่มีข้อมูลที่ขาดหายไป

```
# เลือกเฉพาะข้อมูลตั้งแต่ 2022 ถึง 2025
data_filtered = data[(data['Date'] >= '2021-01-01') & (data['Date'] <= '2025-02-14')]

# ตรวจสอบขนาดข้อมูล
print(data_filtered.shape)
print(data_filtered.head())
#check ว่าข้อมูลครบไหม
data_filtered.info()
data_filtered.describe()
```

ตรวจสอบการเรียงวันที่ให้ถูกต้อง เพราะสำคัญมากในการใช้โมเดล LSTM จากนั้นทำการเลือก features ที่เราสนใจ โดยเลือกเราเลือก 'Open', 'High', 'Low', 'Close', 'Volume' มาใช้เป็น features

```
# Set the 'Date' column as the index
df_filtered = df_filtered.set_index('Date')

#features selected
features = ['Open', 'High', 'Low', 'Close', 'Volume']
data = df_filtered[features]
```

MinMaxScaler เพื่อปรับขนาดข้อมูลราคาหุ้นให้อยู่ในช่วง 0–1 ก่อนนำเข้า LSTM ช่วยให้โมเดลเรียนรู้ได้เร็วขึ้น เสถียรกว่า และลดปัญหา gradient หาย โดยเฉพาะเมื่อข้อมูลราคาหุ้นมีสเกลแตกต่างกันมาก

```
#apply MinMax scaler
price_scaler = MinMaxScaler(feature_range=(0,1))
data_scaled = price_scaler.fit_transform(np.array(data)).reshape(-1,1)
```

แบ่งข้อมูลเป็น train และ test โดยแบ่งข้อมูล train เป็น 80%

```
#แบ่งข้อมูล train/test
train = int(int(len(data))*0.8)
test = len(data) - train
train_data , test_data = data[0:train,:],data[train:len(data),:1]
```

แปลงข้อมูลราคาหุ้นให้เป็นชุดข้อมูลแบบลำดับเวลา (time-series sequences) โดยใช้ time_step วันย้อนหลัง เพื่อพยากรณ์ราคาวันถัดไป ข้อมูลที่ได้จะถูกจัดให้มีมิติ (batch_size, time_step, features) สำหรับนำเข้าโมเดล LSTM ได้โดยตรง

```
#convert array of values into a dataset matrix
def data_set(dataset, time_step=1):
  data_x, data_y = [], []
  for i in range(len(dataset)-time_step-1):
    x_seq = dataset[i:(i+time_step),0]
    y_seq = dataset[i+time_step,0]
    data_x.append(x_seq)
    data_y.append(y_seq)

  x = np.array(data_x)
  y = np.array(data_y)
```

```
  x_tensor = torch.tensor(x[:, :, None], dtype=torch.float32)
  y_tensor = torch.tensor(y[:, None], dtype=torch.float32)

  return x_tensor,y_tensor
```

reshape ให้อยู่ในรูป x=t , t+1 , t+2, ... โดยตั้ง time step = 100

```
#reshape into x=t, t+1 , t+2 , t+3 , y=t+4
time_step = 100
x_train , y_train = data_set(train_data, time_step)
x_test , y_test = data_set(test_data, time_step)

print(x_train.shape, y_train.shape)
```

โมเดลนี้ประกอบด้วย 2 ส่วนหลัก:

1. LSTM layer สำหรับเรียนรู้ลำดับเวลา (Temporal dependencies)

2. Fully Connected (Linear) layer สำหรับแปลงผลลัพธ์ของ LSTM เป็นค่าพยากรณ์สุดท้าย

- input_size=1 คือ จำนวน feature ต่อ timestep
- hidden_size=64 คือ จำนวนหน่วยใน hidden layer ของ LSTM
- num_layers=2 คือ จำนวนชั้นของ LSTM (stacked LSTM)
- dropout=0.3 ลด overfitting โดยสุ่มปิด neuron บางส่วนระหว่าง training
- self.fc = nn.Sequential(nn.ReLU(), nn.Linear(hidden_size, 1)) สร้าง fully connected layer และ activation function เพื่อแปลง hidden output เป็นค่าทำนายสุดท้าย

ผลลัพธ์สุดท้ายผ่าน ReLU activation และ linear layer เพื่อทำนายค่าตัวเลขของวันถัดไป

```
#define model
class LSTMmodel(nn.Module):
  def __init__(self, input_size=1, hidden_size=64, num_layers=2,
batch_first=True,dropout=0.3):
    super(LSTMmodel,self).__init__()
    self.lstm = nn.LSTM(input_size, hidden_size,
num_layers,batch_first=batch_first, dropout=dropout)
    self.fc = nn.Sequential(nn.ReLU(),nn.Linear(hidden_size, 1))
```

```
  def forward(self,x):
    out, _ = self.lstm(x)
    out = self.fc(out[:,-1,:])
    return out
```

เลือก Optimizer โดยเลือกใช้ GPU cuda มาใช้ประมวลผลทำให้ประมวลผลได้เร็วมากขึ้น และตั้ง learning rate = 0.001 ใช้ dataloader เพื่อแบ่ง batch

```
#Initialize Model, Loss Function, and Optimizer
from torch.utils.data import DataLoader, TensorDataset
import torch

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

model = LSTMmodel().to(device) # ใช้ GPU ถ้ามี
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

#ใช้ DataLoader เพื่อแบ่ง batch
batch_size = 64
train_dataset = TensorDataset(x_train, y_train)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

loop การเทรนโมเดล LSTM ที่ทำให้โมเดลเรียนรู้ความสัมพันธ์ของข้อมูล โดยใช้ Mixed Precision เพื่อลดการใช้หน่วยความจำของ GPU และเพิ่มความเร็วในการคำนวณ โดยใช้ torch.cuda.amp.autocast() และ GradScaler() เพื่อเพิ่มประสิทธิภาพการประมวลผลบน GPU และลดการใช้หน่วยความจำ โดยกระบวนการฝึกดำเนินการทั้งหมด 100 epochs พร้อมเก็บค่า loss เพื่อวิเคราะห์แนวโน้มการเรียนรู้ของโมเดล

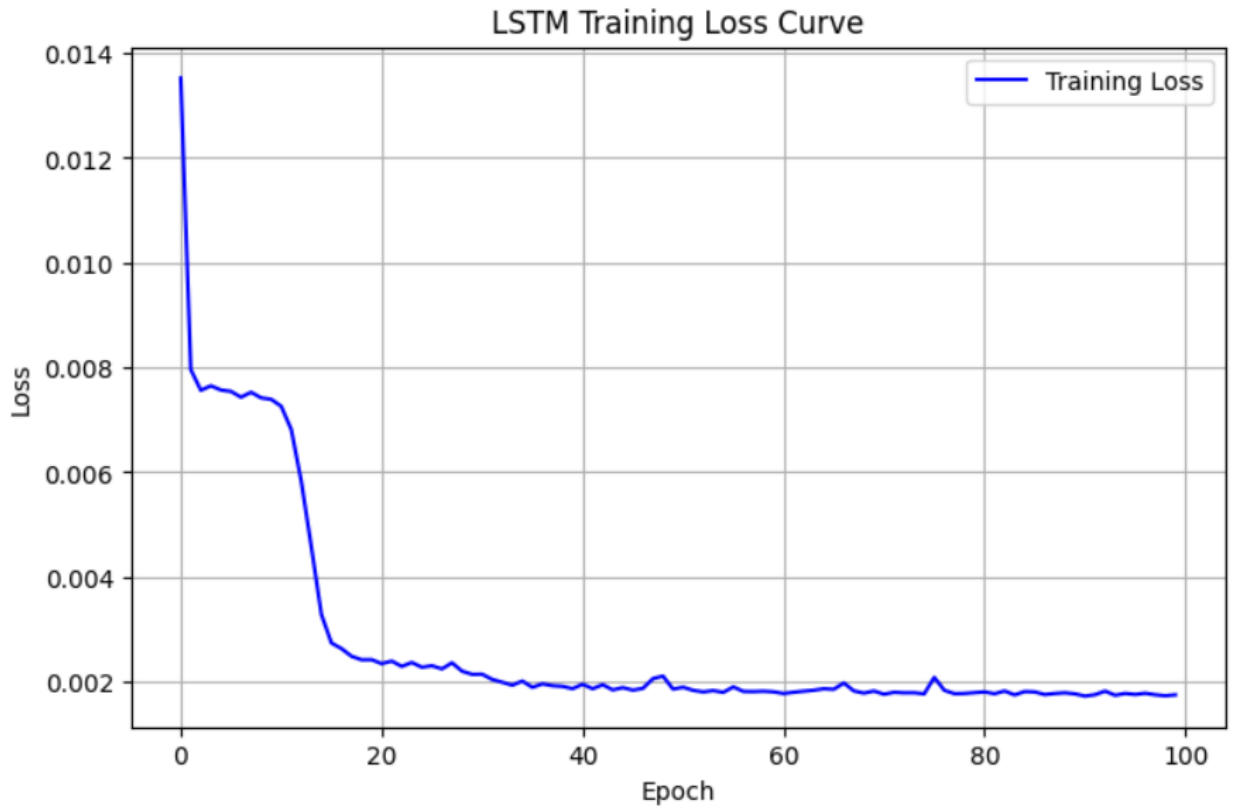
```
# ใช้ Mixed Precision Training
scaler = torch.cuda.amp.GradScaler()

loss_list = []
epoch_loss_list = [] # List to store average loss per epoch
num_epochs = 100
for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0
```

```
    for batch_x, batch_y in train_loader:
        batch_x, batch_y = batch_x.to(device), batch_y.to(device)

        optimizer.zero_grad(set_to_none=True)
        with torch.cuda.amp.autocast(): #ใช้ mixed precision เร่งความเร็ว
            outputs = model(batch_x)
            loss = criterion(outputs, batch_y)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        # เก็บค่าลง list
        loss_list.append(loss.item())

        total_loss += loss.item() * batch_x.size(0)

    avg_loss = total_loss / len(train_loader.dataset)
    epoch_loss_list.append(avg_loss) # Append average loss for the epoch

    # print ทุก 10 epochs
    if (epoch + 1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}] Loss: {avg_loss:.6f}")
```

ค่า Epoch และ Loss ที่ได้จากการ train

```
Epoch [10/100]  Loss: 0.007392
Epoch [20/100]  Loss: 0.002421
Epoch [30/100]  Loss: 0.002141
Epoch [40/100]  Loss: 0.001867
Epoch [50/100]  Loss: 0.001859
Epoch [60/100]  Loss: 0.001807
Epoch [70/100]  Loss: 0.001822
Epoch [80/100]  Loss: 0.001791
Epoch [90/100]  Loss: 0.001768
Epoch [100/100]  Loss: 0.001749
```

กราฟ Loss ในแต่ละรอบ Epoch



Evaluate

```python
#evaluate
model.eval()
with torch.no_grad():
    x_test = x_test.to(device)
    y_test = y_test.to(device)
    pred = model(x_test)
    test_loss = criterion(pred, y_test)
    print("Test Loss:", test_loss.item())

    #RMSE
    rmse = torch.sqrt(test_loss)
    print("RMSE:", rmse.item())

    # MAE (Mean Absolute Error)
    mae = torch.mean(torch.abs(pred - y_test))
    print("MAE:", mae.item())
```

```
#MAPE
epsilon = 1e-3 # ค่าป้องกันศูนย์
mask = torch.abs(y_test) > epsilon  # ใช้เฉพาะค่าที่ไม่ใกล้ศูนย์
mape = torch.mean(torch.abs((y_test[mask] - pred[mask])/(y_test[mask] + 1e-8))) * 100
print("MAPE:", mape.item(), "%")
```

การวัดค่าเราได้อ้างอิงจากงานวิจัย A Hybrid LSTM-GRU Model for Stock Price Prediction (https://ieeexplore.ieee.org/document/11072109) โดยเลือกใช้ค่า RMSE, MAE และ MAPE โดยได้ค่าดังนี้

```
Test Loss: 0.049062665551900864
RMSE: 0.2215009331703186
MAE: 0.20092272758483887
MAPE: 75.0616683959961 %
```

จากงานวิจัยที่อ้างอิงและในการทำนายหุ้นที่ค่ามีความแปรปวนและผันผวนสูง ถือว่าค่าออกมาปกติดี test loss ค่อยข้างมี overfitting แต่โมเดลยังทำงานได้ดี

กราฟเปรียบเทียบค่าที่ทำนายได้เทียบกับราคาหุ้นจริง โดยวัดค่าความถูกต้องได้ ดังนี้

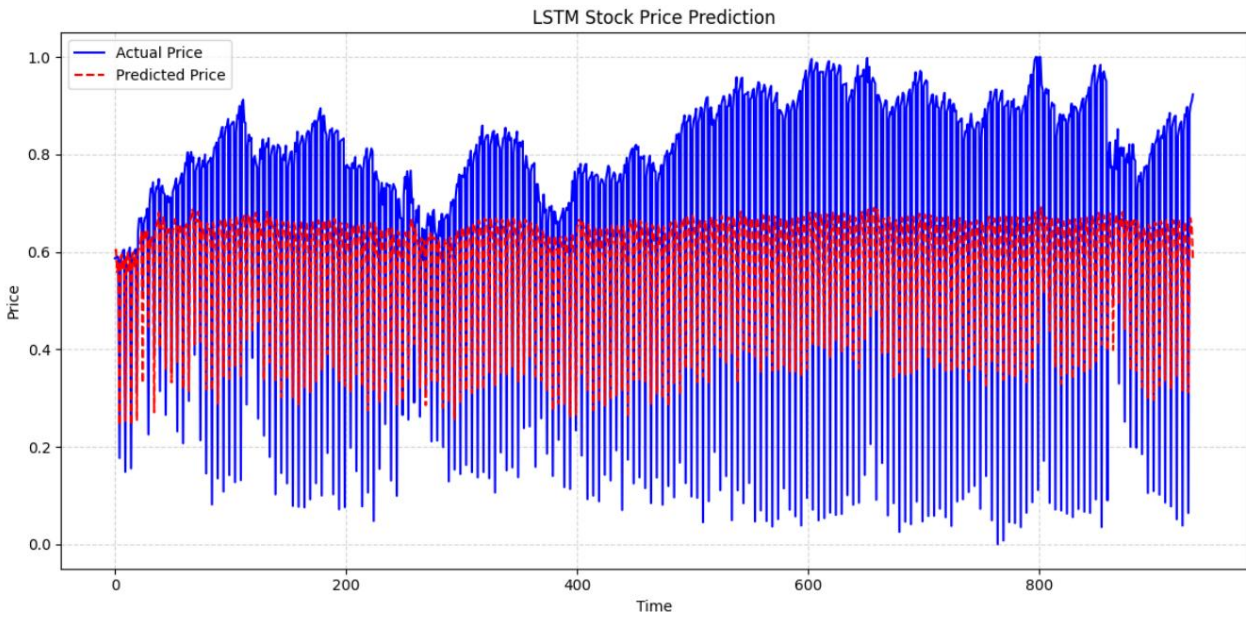
MSE : 0.049063

RMSE : 0.221501

MAE : 0.200923

MAPE : 75.06%

R^2 : 0.4264



แหล่งข้อมูลอ้างอิง

แหล่งที่มาข้อมูล

https://www.finnomena.com/definit/nvidia-dr/

https://www.bangkokbiznews.com/world/1205715

https://www.efinancethai.com/recommended-for-you/nvidiastock

https://www.nerd-data.com/deep_learning_lstm/

แหล่งที่มางานวิจัย

https://ieeexplore.ieee.org/document/11072109

แหล่งที่มา code

https://machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/

https://www.geeksforgeeks.org/deep-learning/long-short-term-memory-networks-using-pytorch/

https://github.com/krishnaik06/Stock-MArket-Forecasting