Project_M

스터디 보고서

2025 / 02 / 20

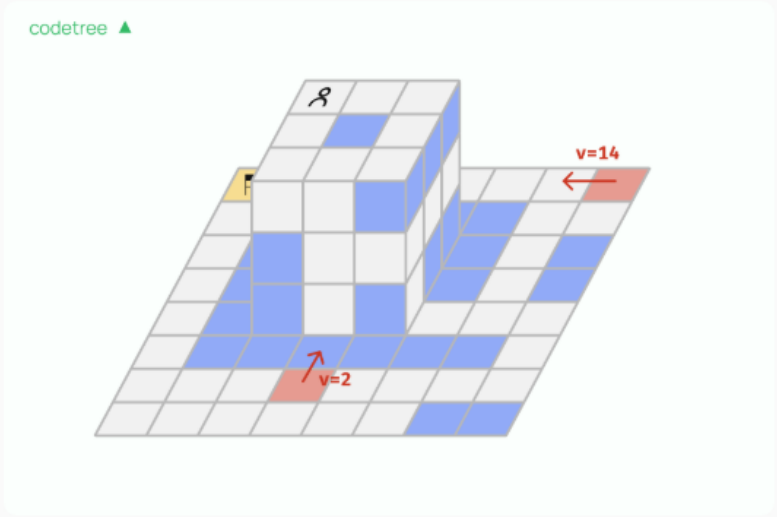
박정하

목차

1. 코딩테스트[미지의 공간 탈출]

미지의 공간 탈출

▪ 미지의 공간 탈출



이 공간은 한 변의 길이가 N인 2차원 평면이며, 그 사이 어딘가에는 한 변의 길이가 M인 정육면체 형태의 **시간의 벽**이 세워 있습니다.

이 평면도와 단면도는 빈 공간과 장애물로 구성되어 있으며, 각각 0과 1로 표현됩니다. 당신의 타임머신은 빈 공간만 이동할 수 있으며, 장애물 위치로는 이동할 수 없습니다.

당신의 타임머신은 시간의 벽의 **윗면** 어딘가에 위치하며, 시간의 벽의 윗면 단면도에서는 **타임머신의 위치 2가 추가로** 표시됩니다. 마찬가지로 미지의 공간의 평면도에서는 **시간의 벽의 위치 3**과 **탈출구 4**가 추가로 표시됩니다. 탈출구는 시간의 벽 외부에 있는 미지의 공간의 바닥에 위치합니다.

시간의 벽과 맞닿은 미지의 공간의 바닥은 기본적으로 장애물들로 둘러 쌓여있습니다. 그러나 단 한 칸만 빈 공간으로 뚫려 있기 때문에 시간의 벽에서 미지의 공간의 바닥으로 이어질 수 있는 출구는 하나입니다.

또한, 미지의 공간의 바닥에는 총 F개의 시간 이상 현상이 존재합니다. 각 시간 이상 현상은 바닥의 빈 공간 (r_i, c_i)에서 시작하여 매 v_i의 배수 턴마다 방향 d_i로 한 칸씩 확산되며, 확산된 이후에도 기존 위치의 시간 이상 현상은 사라지지 않고 남아 있습니다. 시간 이상 현상은 장애물과 탈출구가 없는 빈 공간으로만 확산되며, 더 이상 확산할 수 없을 경우 멈춥니다. 모든 시간 이상 현상은 서로 독립적이며 동시에 확산됩니다. 방향 d는 동(0), 서(1), 남(2), 북(3)으로 표시됩니다.

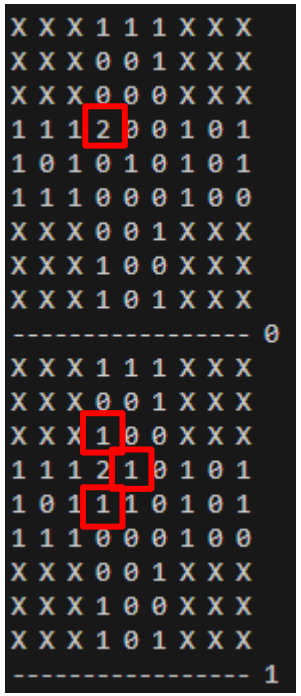
당신의 타임머신은 매 턴마다 상하좌우로 한 칸씩 이동할 수 있으며, **장애물**과 시간 이상 현상을 피해 **탈출구**까지 도달해야 합니다. 타임머신이 시작점에서 탈출구까지 이동하는 데 필요한 최소 시간(턴 수)을 출력하는 프로그램을 작성해야 합니다. 만약 탈출할 수 없다면 -1을 출력합니다. 시간 이상 현상이 확산된 직후 타임머신이 이동하기 때문에, 타임머신은 시간 이상 현상이 확산되는 곳으로 이동할 수 없음에 유의합니다.

미지공간 탈출 문제 풀이

- **코드 구상**
  1. 입력값을 각각 적절한 변수에 저장
  2. 이상현상 구현
  3. 시간의벽 단면도 → 전개도로 확장
  4. 시간의 벽에서 최단거리 탐색(BFS)
  5. 미지 공간에서 최단거리 탐색(BFS)
  6. While문 안에서 time을 1씩 증가시키며, 전체 코드 진행

미지공간 탈출 문제 풀이

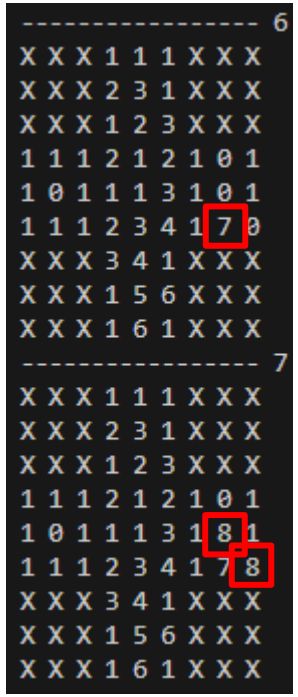
시간 벽 전개도에서 BFS진행



타임머신 위치 2에서 시작

BFS 시작

●●●

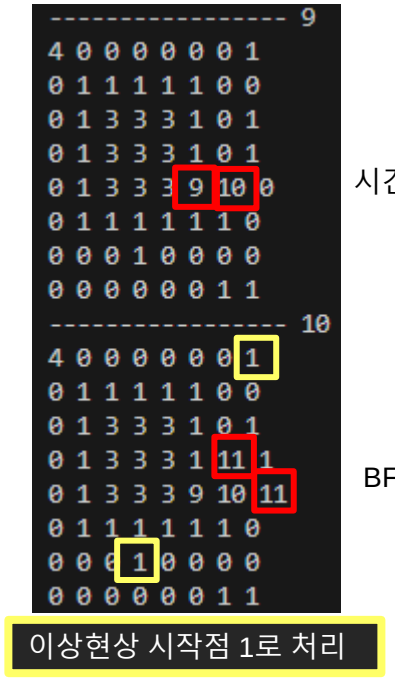


종료지점에 도착(시간의 벽 출구 도착)

이제 미지의 지역 BFS시작

미지공간 탈출 문제 풀이

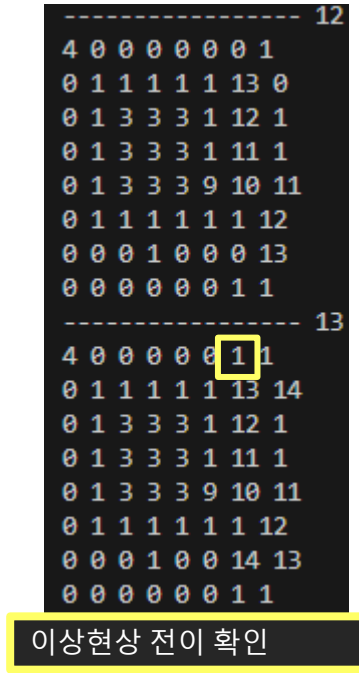
미지의 지역에서 BFS진행



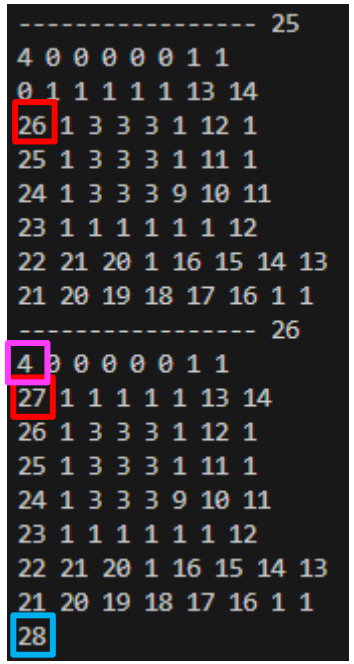
이상현상 시작점 1로 처리

시간의벽 탈출구 부터 시작

BFS탐색 진행



이상현상 전이 확인



BFS탐색 진행 중
미지의 지역 탈출 지역
도달 시 프로그램 종료 후
Time return

미지공간 탈출 문제 풀이

코드는 ipynb파일에 첨부

.py파일에서 실행해야 정상 작동

2. 주가 예측 프로그램

주가 예측 프로그램

- **종목 선정**
  - 종목 :Tesla

  - 선정 사유 : 큰 변동성을 가진 주식으로, LSTM 모델을 이용해 정확한 주가 예측이 가능한지 검증해보고자 함

주가 예측 프로그램

- **LSTM 모델**
  - Torch의 nn모듈을 이용해 LSTM 레이어를 생성

```python
class LSTMModel(nn.Module):
    def __init__(self, input_size=5, hidden_size=64, num_layers=2, dropout=0.1):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, dropout=dropout)
        self.fc = nn.Linear(hidden_size, 1)

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

  - Input size = 5
  - hidden state size = 64
  - LSTM 레이어 = 2
  - 과적합 방지 Dropout = 0.1
  - LSTM레이어를 거친 out값을 Fully connected 레이어 를 지나 1개의 output 반환

주가 예측 프로그램

- **데이터 전처리**
  - 5년치 시가, 종가, 고가, 저가, 거래량 그리고 날짜 정보를 불러옴

```
ticker = "TSLA"
start_date = "2020-02-25"
end_date = "2025-02-25"
data = yf.download(ticker, start=start_date, end=end_date)
```

```
date = data.index
data = data[['Open','High','Low','Volume','Close']].values
```

  - 각 항목별로 정규화를 진행 (Min-Max Scaling을 이용해 0~1값 사이로 정규화)
  - X 에는 60일치에 해당하는 시가, 종가, 고가, 저가, 거래량의 데이터를 모두
  - Y값에는 항목별로 60일치 데이터 다음날에 해당하는(EX 61일에 해당하는) 시가, 종가, 고가, 저가, 거래량 데이터 하나씩 넣음

  - Train_data 80%, Test_data 20%로 데이터 분류
  - Batch size 60, 무작위로 학습데이터 분할

주가 예측 프로그램

- **데이터 전처리**
  - 각 항목별로 정규화를 진행 (Min-Max Scaling을 이용해 0~1값 사이로 정규화)
  - X 에는 60일치에 해당하는 시가, 종가, 고가, 저가, 거래량의 데이터를 모두 포함
  - Y값에는 항목별로 60일치 데이터 다음날에 해당하는(EX 61일에 해당하는) 시가, 종가, 고가, 저가, 거래량 데이터 하나씩 넣음

```
X데이터 shape : (1197, 60, 5)
y_close.shape : (1197,)
y_open.shape : (1197,)
y_high.shape : (1197,)
y_low.shape : (1197,)
y_volume.shape : (1197,)
```

  - Train_data 80%, Test_data 20%로 데이터 분류
  - Batch size 60, 무작위로 학습데이터 분할

주가 예측 프로그램

- **LSTM 모델**
  - Torch의 nn모듈을 이용해 LSTM 레이어를 생성

```python
class LSTMModel(nn.Module):
    def __init__(self, input_size=5, hidden_size=64, num_layers=2, dropout=0.1):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, dropout=dropout)
        self.fc = nn.Linear(hidden_size, 1)

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

  - Input size = 5
  - hidden state size = 64
  - LSTM 레이어 = 2
  - 과적합 방지 Dropout = 0.1
  - LSTM레이어를 거친 out값을 Fully connected 레이어 를 지나 1개의 output 반환

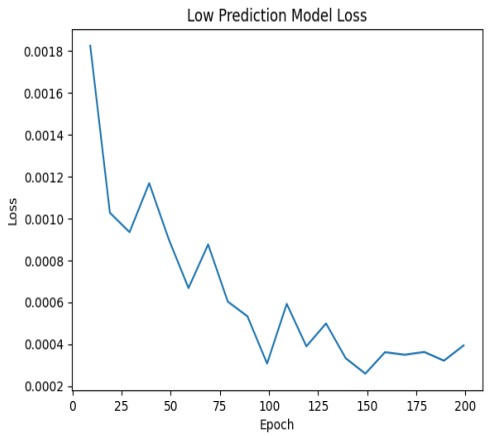
주가 예측 프로그램

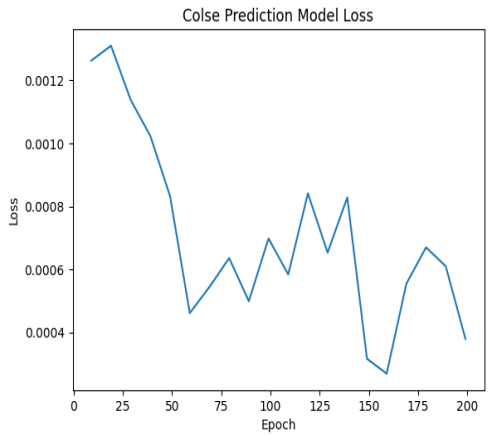
- **모델 객체 생성**
  - 모든 항목의 데이터를 학습하고 1가지 항목을 예측하는 모델 5개 생성
  - 각각 model_close, model_open, model_high, model_low, model_volume

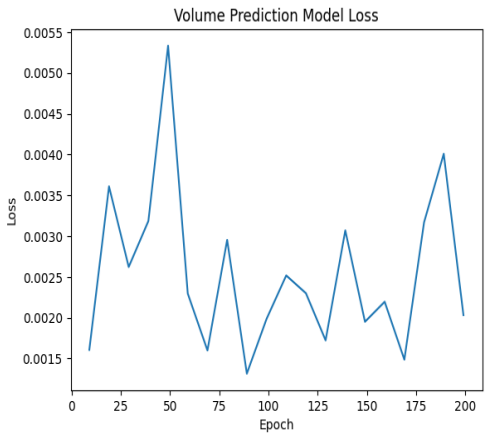
```
model_close = LSTMModel().to(device)
model_open = LSTMModel().to(device)
model_high  = LSTMModel().to(device)
```

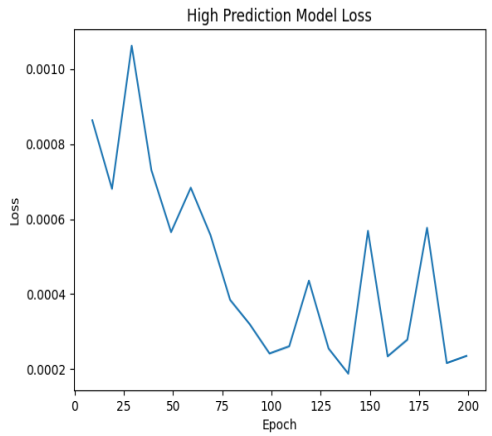
```
model_low  = LSTMModel().to(device)
model_volume  = LSTMModel().to(device)
```

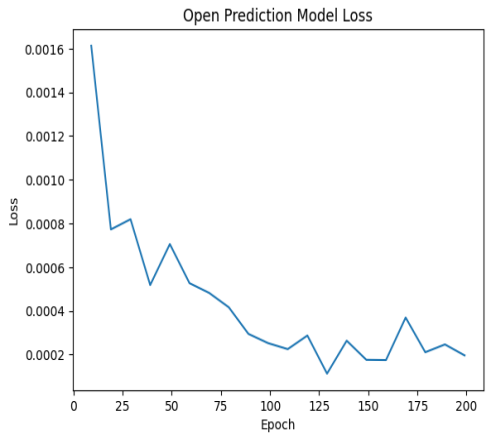
  - 하이퍼 파라미터 : epoch = 200 learning late : 0.001





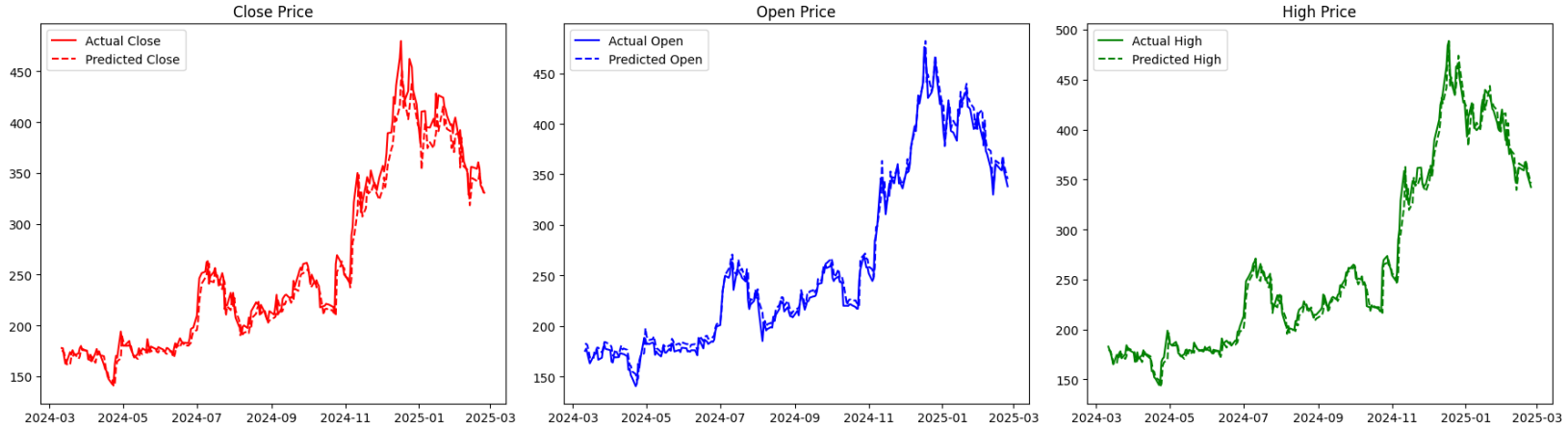






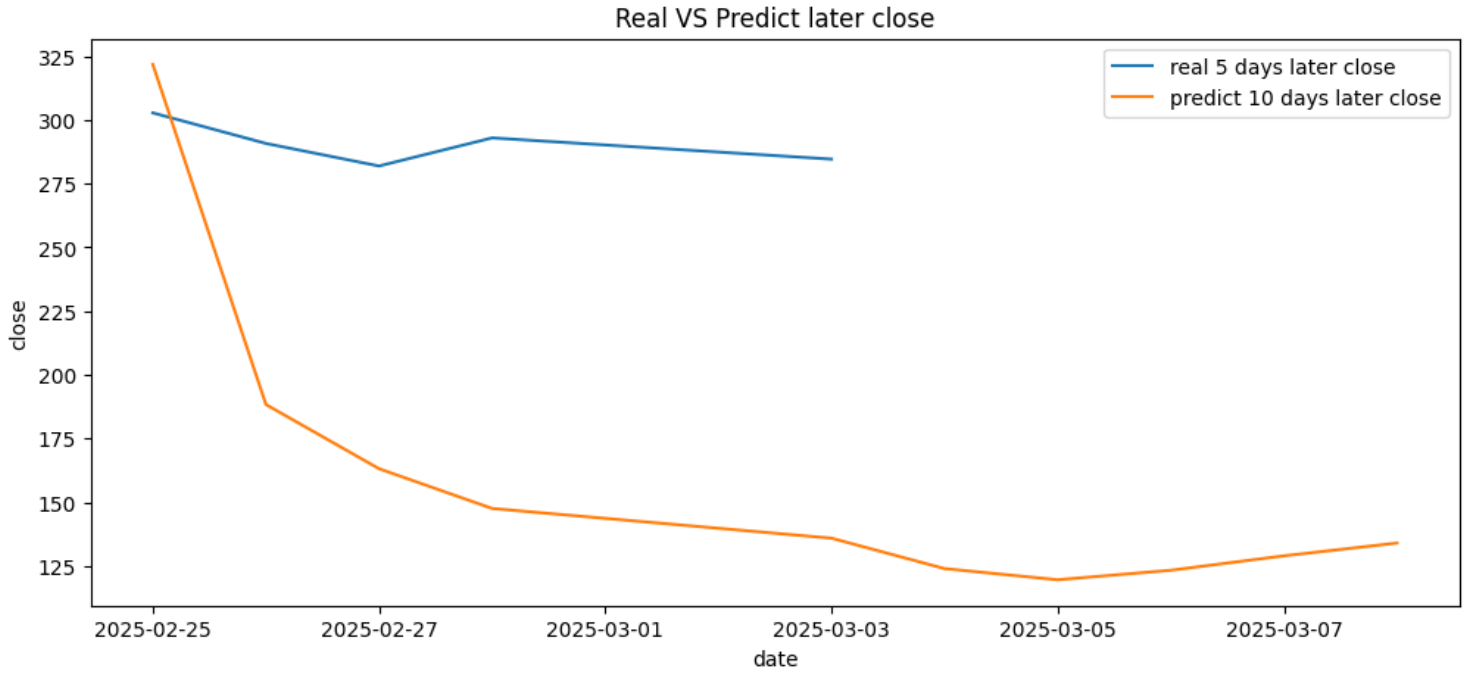
주가 예측 프로그램

- **Test data를 이용한 예측**
  - 학습이 완료된 모델에 테스트 데이터를 넣어 각 항목을 예측
  - 예측 결과와 실제 값을 그래프로 나타냄



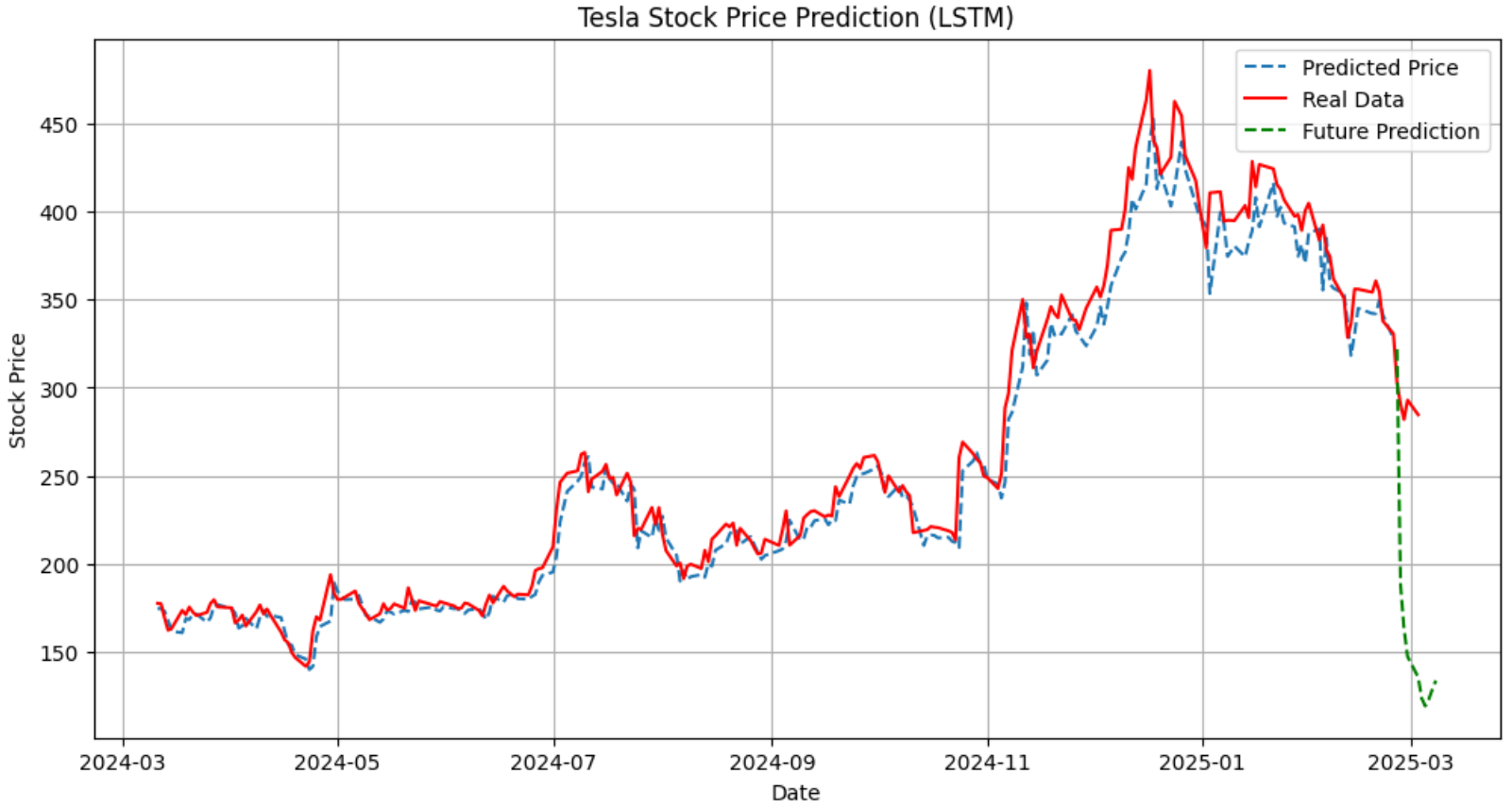
주가 예측 프로그램

- **학습된 모델을 이용한 실제 주가 예측**
  - 학습이 완료된 모델에 실제 데이터를 넣어 10일 후의 종가(Close)예측
  - 2025-02-24일부터 60일 전의 데이터를 이용해 02-25에서의 종가 예측, 그 후 02-25에서부터 60일전 데이터로 다음 값 예측



주가 예측 프로그램

- **학습된 모델을 이용한 실제 주가 예측**
  - Test데이터와 test데이터를 이용해 예측한 값
  - 그리고 실제 데이터를 이용해 예측한 값을 더해 그래프로 표현



주가 예측 프로그램

- **결론**
  - Test데이터를 이용해 모델을 테스트한 결과는 실제 주가와 매우 유사했음.
  - 하지만 학습된 모델을 이용해 test셋의 범위를 넘어가는 주가 하루하루 예측하고, 그 예측값을 이용해 다시 예측을 실행할 경우 실제 주가와 다른 경향을 보이는 것을 확인
  - 이는 실제 이번 프로젝트를 통해 만든 LSTM 주가 예측 프로그램이 실제 예측에는 좋은 성능을 보이지 않는다고 말할 수 있음.
  - 이는 실제 변동성이 매우 큰 Tesla주식에만 낮은 정확도를 보이는 것인지, 추세가 비슷한 시가, 종가, 고가, 저가, 거래량 만을 이용해 학습한 모델 자체의 실제 성능이 떨어지는 것인지 추가적인 연구가 필요해 보임.

감사합니다