AI Portfolio Report

컴퓨터정보공학과 PE
20190715
신현정

INDEX

1
개발 환경

개발 환경

1. Google Colab

▲ Google Colab

본 포트폴리오에서 이용될 구글 코랩은 파이썬, 머신러닝, 딥러닝을
보다 쉽게 하기 위해 제공되는 클라우드 환경입니다.

구글 코랩은 따로 개발 환경을 구축하지 않아도
손쉽게 개발할 수 있도록 도와줍니다.

앞으로 사용될 파이썬 언어를 바탕으로
텐서플로우, 케라스 등의 딥러닝 라이브러리도 쉽게 사용할 수 있습니다.

또한, 구글 드라이브와 연동해 사용할 수 있기 때문에
파일 관리도 용이합니다.

(아주 굿)

2
기초

기초

1. 버전 변경 및 확인

```
#버전 변경 case1_기본
%tensorflow_version 1.x
```

▲ 1.X으로 버전 변경

```
import tensorflow as tf
tf.__version__
```

▲ 버전 확인

위 코드를 통해 현재 실행되는 tensorflow의 버전을 확인할 수 있습니다.

(기본은 2.x)

2. shape

```
a = tf.constant([1,2,3])
a.shape # TensorShape([3]) >>shape이 [n] 인 것을 보아 vector 타입이다!
#>>Vector / 1차원 Tensor
```

```
TensorShape([3])
```

▲ shape 사용

shape 명령어는 객체의 모양을 확인할 수 있는 명령어입니다.

위 코드는 1차원 배열인 Vector타입의 상수를 a에 생성한 뒤

a의 모양을 확인하는 코드이며,

결과는 [3]이므로 3행의 Vector 타입임을 알 수 있습니다.

3. tensorflow.Constnat()

```
aa = tf.constant(5)
aa #마지막줄에 적으면 numpy도 함께 뜸! >> print했을 때는 맨 앞에 인자가 numpy 값을 의미!
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=5>
```

▲ constant 사용

constant는 상수를 생성하는 tensorflow 명령어입니다.

인자로 5라는 숫자를 넣었더니

데이터 타입(dtype)이 int로 숫자형이 나오는 것을 알 수 있습니다.

```
c = tf.constant('Hello,world!')
```

▲ c string 객체 생성

c에 문자열 상수를 저장해 놓겠습니다.

4. numpy()

```
print(c.numpy()) #b'Hello,world!'
#>> c의 내용(값)만을 보고 싶다면, numpy()함수를 이용한다
```

```
tf.Tensor(b'Hello,world!', shape=(), dtype=string)
b'Hello,world!'
```

▲ numpy()

numpy 는 해당 객체의 값만을 확인할 수 있는 함수입니다.

코드를 확인하면 c라는 상수에 Hello,World!를 저장해 놓고,

numpy로 값을 확인하는 것을 알 수 있습니다.

5. print()

```
print(c) #tf.Tensor(b'Hello,world!', shape=(), dtype=string)
#>> 여기서 b는 byte타입을 의미한다
```

▲ print()

print는 여느 함수와 마찬가지로 print하는 함수입니다.

안에 객체를 넣으면 numpy, shape, dtype을 차례대로 반환합니다.

```
#+번외
print(a.shape) #형태만 출력
print(a.dtype) #내부데이터 타입만 출력
```

```
()
<dtype: 'int32'>
```

▲ print() 응용

shape와 dtype을 print를 하면

모양과 데이터 타입이 출력되는 것을 알 수 있습니다.

6. tensorflow.cond(pred,true_fn,false_fn,name)

```
x = tf.constant(1.)
bool = tf.constant(True)
res = tf.cond(bool,lambda: tf.add(x,1.), lambda: tf.add(x,10.))
#bool을 검사해 참일 경우 x(1.0)+1.0 = 2.0 / 거짓일 경우 x(1.0)+10.0 = 11.0
#lambda
print(res)
print(res.numpy())
```

```
tf.Tensor(2.0, shape=(), dtype=float32)
2.0
```

▲ 조건문 cond

x에 1.0 실수값을 저장하고, bool에 True 논리값을 저장합니다.

이후 해당 논리 값을 이용해 조건문을 실행합니다.

위 조건문은 첫번째 인자인 bool이 True일 경우

1.0+1.0인 2.0을 반환하고, 그렇지 않을 경우 1.0+10.0인 11.0을 반환합니다.

이 반환 값은 res라는 변수에 저장하였습니다.

7. 브로드 캐스팅

```
x = tf.constant([[0],[10],[20],[30]]) #4행 1열
y = tf.constant([0,1,2]) #1행 3열

# x = 4행 1열을 4행 3열로 늘린다
# y = 1행 3열을 4행 3열로 늘린다
# x+y = 4행 3열이 된 x와 y의 각 자리의 내부 데이터를 더한다.
print((x+y).numpy())
```

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

▲ 브로드 캐스팅

각자 다른 행과 열을 가진 배열이 더해졌음에도

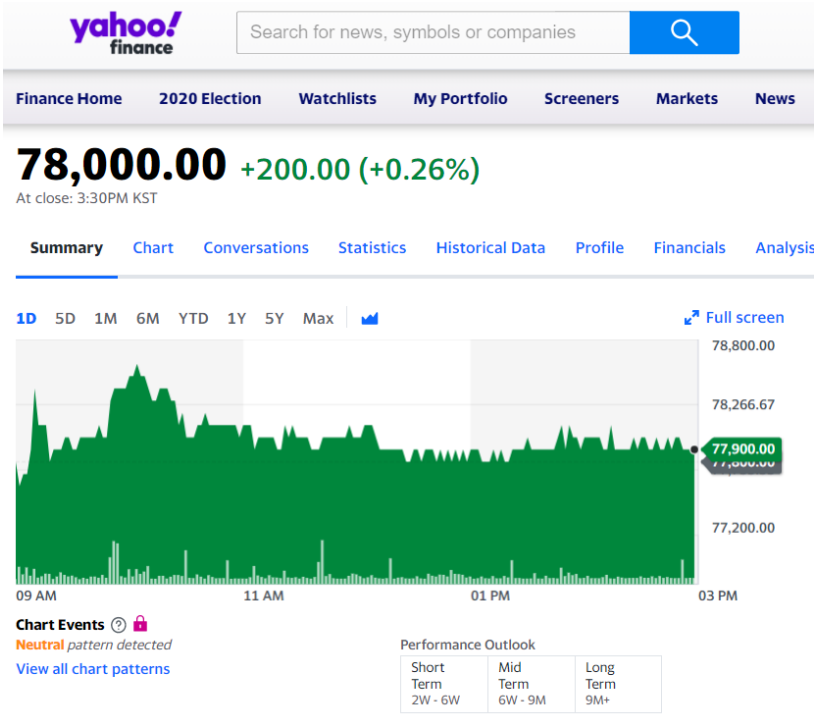
오류가 나지않고 저절로 배열이 늘려져 연산이 되었음을 확인할 수 있습니다.

3
주가 예측 모델 만들기

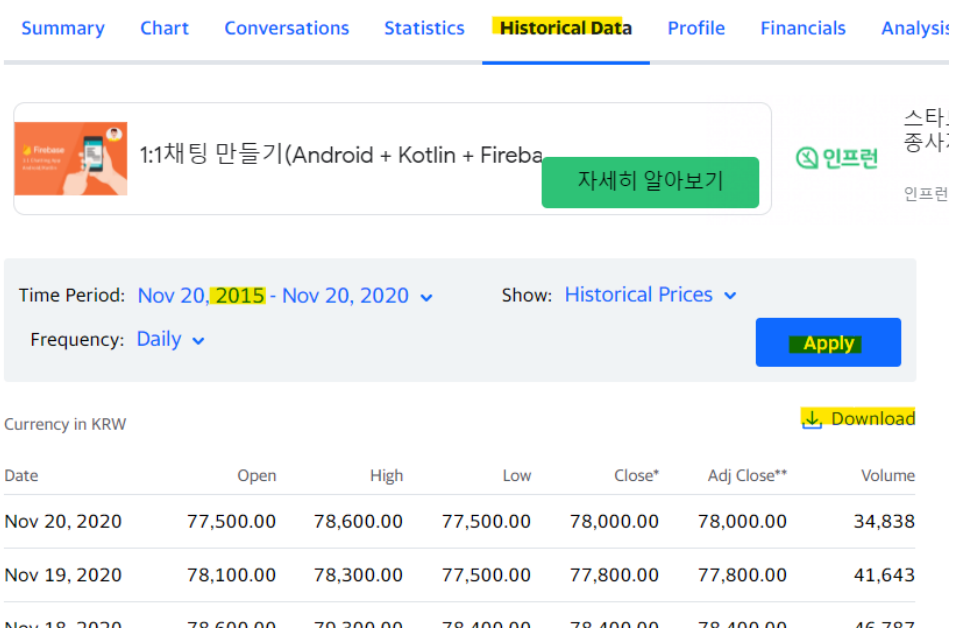
주식 예측

1. 데이터 구하기

동양미래대학교 설립 기업인 효성 기업의 5년치 데이터를 구해봅시다.



▲ 야후 판다스에서 효성 기업 검색



▲ 현재 일로 부터 5년치 데이터를 조회하고 csv파일을 download

주식 예측

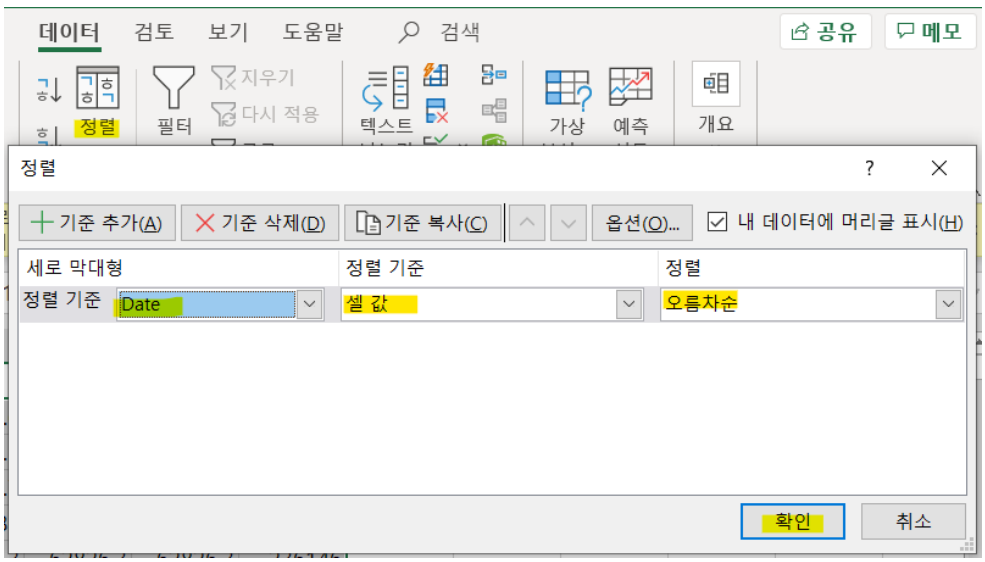
2. import

```
[5]  import io
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from keras.models import Sequential
     from keras.layers import LSTM, Dropout, Dense, Activation
     # from keras.callbacks import TensorBoard, ModelCheckpoint, ReduceLROnPlateau
     import datetime
```

본격적으로 이전 과정의 csv 파일을 업로드 하기 위해
io와 pandas를 import하고
예측을 하기 위한 numpy, matplotlib, keras 등을
import 해줍니다.

3. 데이터 로드하기

▲ 앞 순서에서 다운받은 csv파일



▲ 파일을 열어 데이터 정렬

주식 예측

csv 데이터를 날짜 순으로 정렬해주고 저장을 합니다.

```
[4]  from google.colab import files
     myfile = files.upload()
```

파일 선택 004800.KS.csv

- **004800.KS.csv**(application/vnd.ms-excel) - 75619 bytes, last modified: 2020. 11. 20. - 100% done

Saving 004800.KS.csv to 004800.KS.csv

▲ 파일을 Goolgle Colab에 Load

Google Colab으로 들어가 위와 같은 코드를 써줍니다.
Run을 하면 파일을 선택할 수 있는 버튼이 생성되며
위 csv파일을 Colab에 업로드할 수 있게 됩니다.

4. 데이터 로드하기

```
[6]  data = pd.read_csv(io.BytesIO(myfile['004800.KS.csv']))
     data.head()
```

▲ 코드

pandase의 read_csv() 함수를 이용해
Colab에 업로드 시킨 csv파일을 로드해줍니다.

* (선택) 이후 head() 함수를 이용해 상위 5개 행만 나타나게 해 데이터가 잘 로드됐는지 확인합니다.

	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-11-20	58637.89844	59423.19922	57852.60156	58376.10156	58376.10156	199374.0
1	2015-11-23	58899.69922	59685.00000	58376.10156	59423.19922	59423.19922	201677.0
2	2015-11-24	59685.00000	61779.19922	58899.69922	61517.50000	61517.50000	306708.0
3	2015-11-25	61255.69922	61517.50000	59685.00000	60732.10156	60732.10156	241578.0
4	2015-11-26	61517.50000	62826.30078	60470.30078	62826.30078	62826.30078	236146.0

▲ 실행 결과

주식 예측

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	2015-11-20	58637.9	59423.2	57852.6	58376.1	58376.1	199374
3	2015-11-23	58899.7	59685	58376.1	59423.2	59423.2	201677
4	2015-11-24	59685	61779.2	58899.7	61517.5	61517.5	306708
5	2015-11-25	61255.7	61517.5	59685	60732.1	60732.1	241578
6	2015-11-26	61517.5	62826.3	60470.3	62826.3	62826.3	236146

▲ 다운받은 csv 파일 (소수점 무시)

잘 로드된 것을 확인하면

5. 결측치 제거

```
data = data.dropna()
```

▲ 결측치 제거

결측치를 제거해 나중에 훈련에 무리가 없게 합니다.

(필수 작업) 불 이행 시, 이후 훈련 시 loss값이 nan으로 뜬다.)

6. 최저가/ 최고가 / 중간가 추출

```
[7]  high_prices = data['High'].values
     low_prices = data['Low'].values
     mid_prices = (high_prices + low_prices) / 2
```

▲ 최고가 / 최저가 / 중간가

csv파일이 저장된 data의 high / low 값을 추출해

중간 (mid) 가격을 구합니다.

주식 예측

7. window 생성

```
[8]  seq_len = 50
     sequence_length = seq_len + 1

     result = []
     for index in range(len(mid_prices) - sequence_length):
         result.append(mid_prices[index: index + sequence_length])
```

▲ 최고가 / 최저가 / 중간가

최근 50일 간의 데이터를 보고 예측을 할 것이기 때문에
시퀀스 길이를 50으로 지정해줍니다.
즉, 50개의 데이터를 보고 예측 데이터 1개를 만드므로
최종적으로 적용되는 시퀀스 길이는 50+1로 지정해줍니다.

* seq_len : 윈도우 사이즈

sequence_length : 예측값이 저장된 윈도우 사이즈

이러한 윈도우 사이즈를 기반으로 for문을 돌며
result에 51개의 데이터가 들어간 윈도우가 제작 됩니다.

8. 데이터 정규화 (z-score)

```
[9]
     normalized_data = []
     for window in result:
         normalized_window = [((float(p) / float(window[0])) - 1) for p in window]
         normalized_data.append(normalized_window)

     result = np.array(normalized_data)
```

모델의 좀 더 정확한 예측을 위해 데이터 정규화를 진행합니다.
이 역시 for문을 돌며,
각 윈도우 값들을 각 윈도우의 첫번재 값으로 나누어 준 다음 1을 빼줍니다.

즉, 첫번째 윈도우의 첫번째 값이 00이 되게끔 기준을 잡고
다른 값들에 대한 비율이 자연스럽게 정해지게 합니다.
(이때, 0은 자기자신을 나누면 1인데 여기서 1을 빼니 00이 되는 것)

해당 값은 result에 저장합니다.

9. 훈련 / 테스트 데이터 생성

이렇게 만들어진 result에서 학습과 학습 검증을 위해
훈련 데이터와 테스트 데이터를 분리합니다.

```
row = int(round(result.shape[0] * 0.9))
train = result[:row, :]
np.random.shuffle(train)

x_train = train[:, :-1]
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
y_train = train[:, -1]
```

▲ 훈련 데이터 생성

본 데이터의 90%는 훈련 (train) 데이터로
10%는 테스트 (test) 데이터로 이용됩니다.

먼저 데이터의 90%만큼 뽑아내 train에 배열로 저장하고,
더 좋은 훈련을 위해 numpy의 random.shuffle() 함수를 이용해
배열의 값을 랜덤으로 섞습니다.

주식 예측

윈도우 사이즈만큼의 데이터를 가지고 이후 한 개의 데이터를 예측할 예정이므로
train 데이터의 input(x_train)에 50(=윈도우 사이즈)개의 데이터를 넣고
output(y_train)에 나머지 1개의 데이터를 넣어 훈련을 시키도록 합니다.

```
x_test = result[row:, :-1]
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
y_test = result[row:, -1]
```

▲ 테스트 데이터 생성

이후 test 데이터도 마찬가지로 input과 out 데이터를 넣어줍니다.

```
x_train.shape, x_test.shape
```

```
((1058, 50, 1), (117, 50, 1))
```

▲ 훈련 / 테스트 데이터 확인

확인을 하면,
1058개의 데이터로 학습을 하고
117개의 데이터로 예측을 할 예정임을 알 수 있습니다.

10. Model 생성

```
model = Sequential()
```

▲ 모델 생성

먼저, keras의 modals.sequential() 클래스를 이용해
모델을 순차적으로 정의할 준비를 합니다.

주식 예측

```
model.add(LSTM(50, return_sequences=True, input_shape=(50, 1)))

model.add(LSTM(64, return_sequences=False))
```

▲ 층 정의

먼저, keras의 modals.sequential() 클래스를 이용해
모델을 순차적으로 정의할 준비를 합니다.

이후 모델에 레이어를 추가하기 위해 add로 레이어를 추가하는데
이때, keras에서 제공되는 모듈인 layers.LSTM을 이용해
레이어를 생성해 추가합니다.
input을 50개로 지정한 뒤,
첫번재 레이어는 유닛이 50개, 두번째는 64개로 지정해 놓았습니다.
(성능에 따라 조정 가능)

```
model.add(Dense(1, activation='linear'))
```

▲ 층 정의

keras의 layers.Dense를 이용해 output를 지정합니다.

```
model.compile(loss='mse', optimizer='rmsprop')
```

▲ 컴파일

손실 함수로 mean squared error를 사용하고,
옵티마이저는 rmsprop을 사용해 컴파일을 동작합니다.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50, 50)            10400
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                29440
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 39,905
Trainable params: 39,905
Non-trainable params: 0
_________________________________________________________________
```

▲ 결과 (요약)

summery를 이용해 만들어진 modal의 정보를 출력합니다.
결과적으로 50개가 들어가 1개의 결과가 나오게 될 것이란 것을 알 수 있습니다.

11. Model 훈련

```
model.fit(x_train, y_train,
    validation_data=(x_test, y_test),
    batch_size=10,
    epochs=20)
```

▲ 데이터 훈련

fit을 이용해 모델을 훈련 시킵니다.
이때 10개씩 묶어 에폭를 20으로 두어 20번 반복 훈련 시킵니다.
(에폭 조정 가능)

주식 예측

```
model.fit(x_train, y_train,
    validation_data=(x_test, y_test),
    batch_size=10,
    epochs=20)
```

```
Epoch 1/20
106/106 [==============================] - 5s 47ms/step - loss: 0.0030 - val_loss: 0.0022
Epoch 2/20
106/106 [==============================] - 4s 42ms/step - loss: 0.0014 - val_loss: 0.0017
Epoch 3/20
106/106 [==============================] - 4s 40ms/step - loss: 9.8488e-04 - val_loss: 0.0014
Epoch 4/20
106/106 [==============================] - 4s 40ms/step - loss: 8.3160e-04 - val_loss: 8.6832e-04
Epoch 5/20
106/106 [==============================] - 4s 42ms/step - loss: 6.9534e-04 - val_loss: 0.0010
Epoch 6/20
106/106 [==============================] - 4s 42ms/step - loss: 5.6881e-04 - val_loss: 0.0014
Epoch 7/20
106/106 [==============================] - 4s 41ms/step - loss: 4.7420e-04 - val_loss: 0.0010
```

▲ 훈련 중

손실도가 낮아지는 것을 확인할 수 있습니다.

12. Model 검증 (테스트)

```
pred = model.predict(x_test)
```

▲ 데이터 예측

predict를 사용와 테스트 데이터를 이용해

이전 과정에서 훈련된 데이터의 예측 값을 살펴 봅니다.

예측된 데이터는 pred라는 변수에 저장하겠습니다.

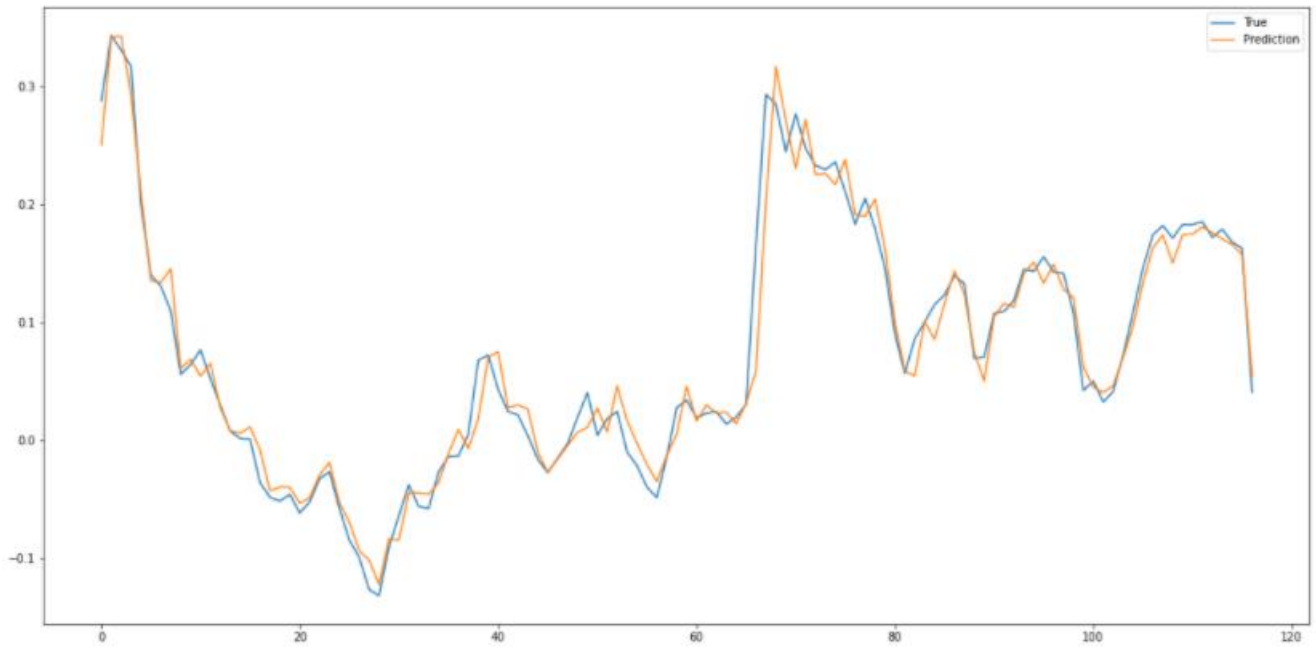
```
fig = plt.figure(facecolor='white', figsize=(20, 10))
ax = fig.add_subplot(111)
ax.plot(y_test, label='True')
ax.plot(pred, label='Prediction')
ax.legend()
plt.show()
```

▲ 그래프 도출

주식 예측

이후 실제 주가를 파란색으로,
예측 값을 주황색으로 표현할 수 있도록 하고
그래프를 보여주는 코드를 실행합니다..

13. 결과



▲ 그래프

파란색은 실제 값, 주황색은 인공지능이 예측한 값입니다.
비교해 보았을 때, 얼추 맞는 것을 확인할 수 있습니다.

이처럼 50일 간의 데이터를 가지고 예측하고
예측한 값이 꽤 정확한 결과를 도출할 수 있게 되었습니다.

* 주의) 사회적 환경이 반영되지 않았으므로 완전히 정확한 예측은 아닙니다 !

The End