LSTM을 이용한 주가 예측

발표자 오주석

2022.11.08

목차와 개요

Part 1 목적과 필요성

Part 1 목적과 필요성

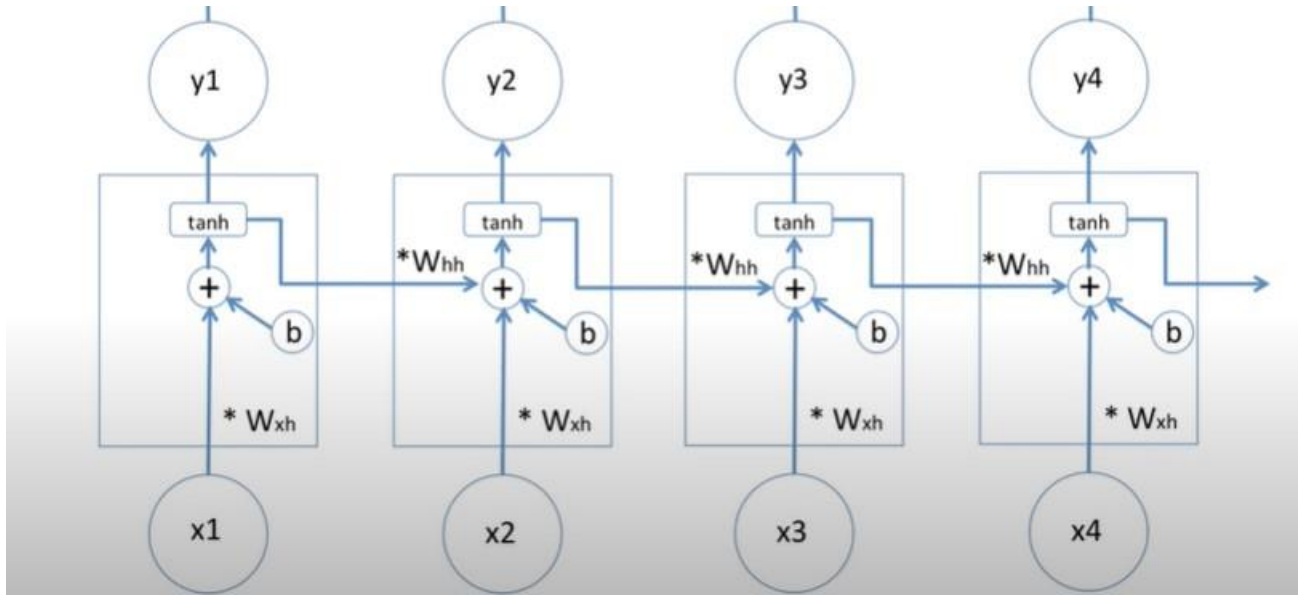
첫째	시계열 분석에 대해 이해한다
둘째	RNN에 대해 이해한다
셋째	자본주의의 꽃인 주가를 예측해본다

Part 2 관련내용

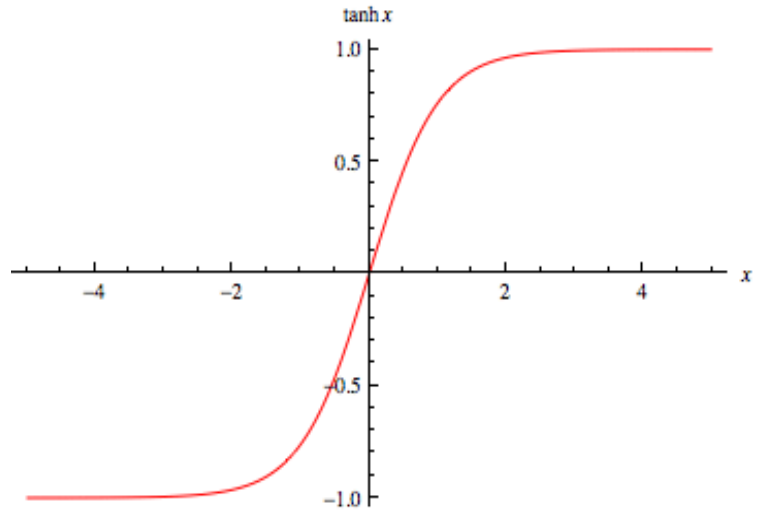
Part 2

관련내용 - RNN

RNN은 Recurrent Neural Network 의 줄임말로
추론능력을 수학적으로 구현한 알고리즘이다.



RNN

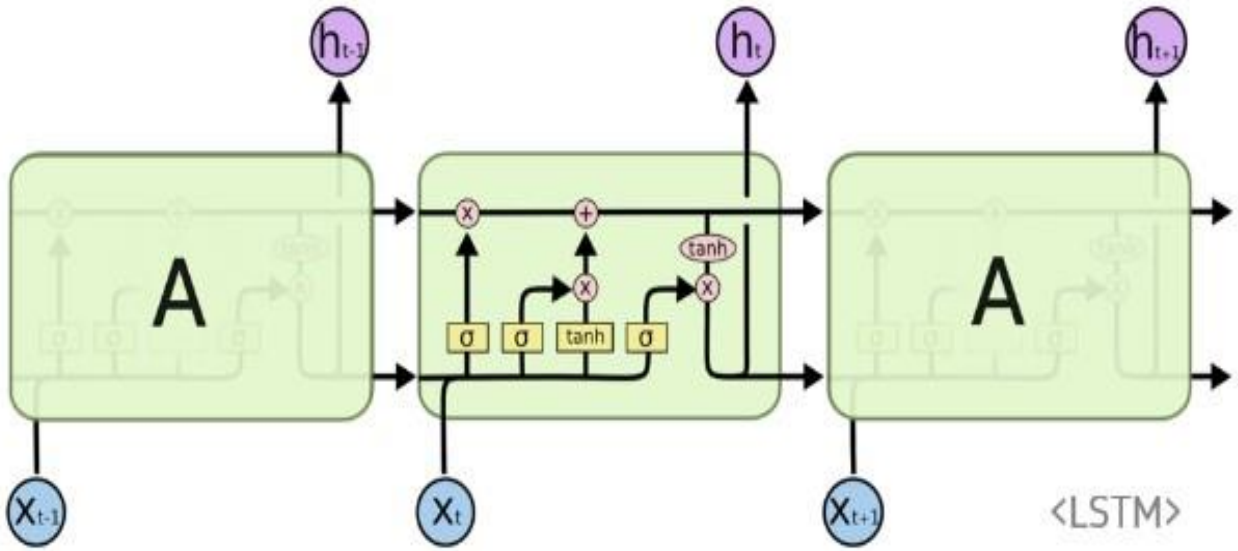


Tanh 함수

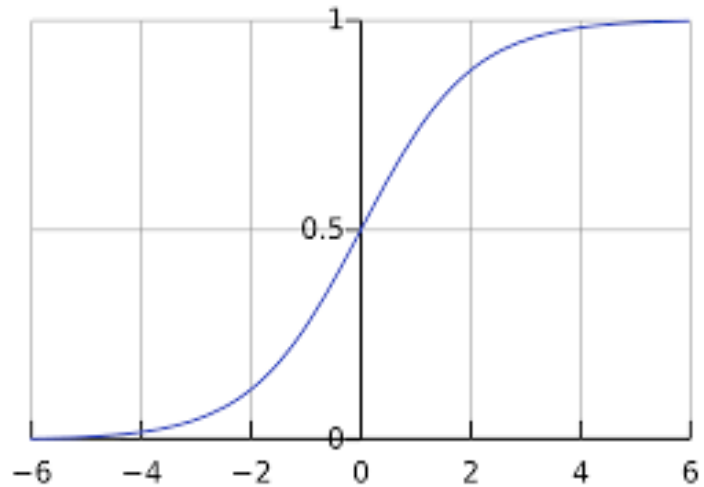
Part 2

관련내용 - LSTM

LSTM은 RNN모델에서 Vanishing Gradient를
해결하기 위해 고안된 모델이다



<LSTM>

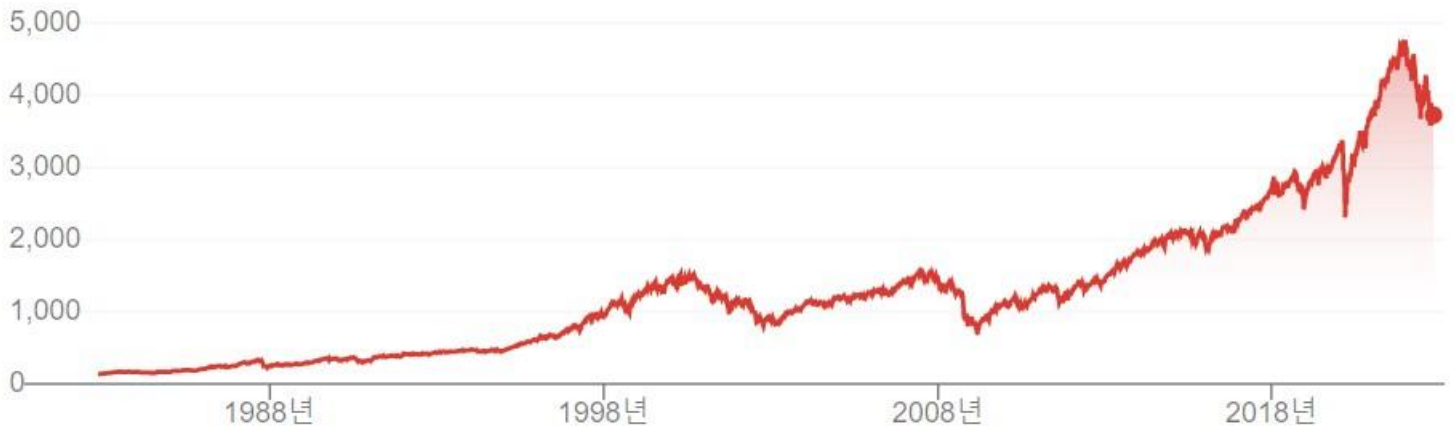


sigmoid 함수

Part 2

관련 내용 - S&P 500

미국 신용평가사 S&P Global 에서 미국에서 가장 비싼기업 상위 500개의 주식을 묶어 수치화 한것



Part 2

관련 내용 - 미시경제

자원, 분배, 기업의 행동에 초점을 두는 경제학이다.
개별 상품의 시장에서 이뤄지는 균형을 연구한다.

Part 2

관련 내용 - 거시경제

모든 개별주체들의 상호작용의 결과로 인해
나타나는 국가의 전체적인 경제 현상에 대한 분석

Part 3 요약

Part 3 요약

Step 1		Step 2		Step 3
S&P 500 지수는 거시경제에 영향을 받는 지수로 각종 거시경제 지표 데이터를 모은다.	>>	모은 데이터로 RNN 분석을 실행한다. 기본모델은 VanishingGradient 문제가 있으므로 LSTM 모델을 이용한다.	>>	LSTM으로 모델링 된 결과를 테스트 한다.

Part 4 예측 과정과 결과

Part 4

예측과정과 결과 - 데이터 로드

```python
import pandas as pd # 판다스 로드
# 데이터프레임으로 로드
data_rate = pd.read_csv('금리.csv')
data_stack = pd.read_csv('./S&P500 지수.csv')
data_m2 = pd.read_csv('./M2SL.csv')
#출력
print(data_rate)
print(data_stack)
print(data_m2)
```

Code

```
            Date         Open         High          Low        Close  \
0     1927-12-30    17.660000    17.660000    17.660000    17.660000
1     1928-01-03    17.760000    17.760000    17.760000    17.760000
2     1928-01-04    17.719999    17.719999    17.719999    17.719999
3     1928-01-05    17.549999    17.549999    17.549999    17.549999
4     1928-01-06    17.660000    17.660000    17.660000    17.660000
...          ...          ...          ...          ...          ...
23318 2020-10-29  3277.169922  3341.050049  3259.820068  3310.110107
23319 2020-10-30  3293.590088  3304.929932  3233.939941  3269.959961
23320 2020-11-02  3296.199951  3330.139893  3279.739990  3310.239990
23321 2020-11-03  3336.250000  3389.489990  3336.250000  3369.159912
```

```
     Year  Month  Day  Federal Funds Target Rate  Federal Funds Upper Target  \
0    1959      1    1                        NaN                         NaN
1    1959      2    1                        NaN                         NaN
2    1959      3    1                        NaN                         NaN
3    1959      4    1                        NaN                         NaN
4    1959      5    1                        NaN                         NaN
..    ...    ...  ...                        ...                         ...
889  2020      7    1                        NaN                         NaN
890  2020      8    1                        NaN                         NaN
891  2020      9    1                        NaN                         NaN
892  2020     10    1                        NaN                         NaN
```

Part 4

예측과정과 결과 - 데이터 전처리

```
#54년 7월 이전데이터 제거
for i in range(0, 7754):
    data_stack = data_stack.drop([i], axis=0)
data_stack.reset_index(inplace=True)
data_stack
```

Code

	Date	Open	High	Low	Close	\
0	1927-12-30	17.660000	17.660000	17.660000	17.660000	
1	1928-01-03	17.760000	17.760000	17.760000	17.760000	
2	1928-01-04	17.719999	17.719999	17.719999	17.719999	
3	1928-01-05	17.549999	17.549999	17.549999	17.549999	
4	1928-01-06	17.660000	17.660000	17.660000	17.660000	
...	...	...	...	...	...	
23318	2020-10-29	3277.169922	3341.050049	3259.820068	3310.110107	
23319	2020-10-30	3293.590088	3304.929932	3233.939941	3269.959961	
23320	2020-11-02	3296.199951	3330.139893	3279.739990	3310.239990	
23321	2020-11-03	3336.250000	3389.489990	3336.250000	3369.159912	

Date	Open	High	Low	Close	Adj Close	Volume
1959-01-02	55.439999	55.439999	55.439999	55.439999	55.439999	3380000
1959-01-05	55.660000	55.660000	55.660000	55.660000	55.660000	4210000
1959-01-06	55.590000	55.590000	55.590000	55.590000	55.590000	3690000
1959-01-07	54.889999	54.889999	54.889999	54.889999	54.889999	4140000
1959-01-08	55.400002	55.400002	55.400002	55.400002	55.400002	4030000
...	...	...	...	...	...	...
2020-10-29	3277.169922	3341.050049	3259.820068	3310.110107	3310.110107	4903070000
2020-10-30	3293.590088	3304.929932	3233.939941	3269.959961	3269.959961	4840450000

Part 4

예측과정과 결과 - 데이터 전처리(결측치 제거)

```
ch = data_rate['Real GDP (Percent Change)'].isnull()
ch_rate = data_rate['Effective Federal Funds Rate'].isnull()
ch_rate1 = data_rate['Unemployment Rate'].isnull()
ch_rate2 = data_rate['Inflation Rate'].isnull()
for i in range(0, len(data_rate)):
        if ch[i] == True:
            data_rate['Real GDP (Percent Change)'][i] = data_rate['Real GDP (Percent Change)'][i-1]
        if ch_rate[i] == True:
            data_rate['Effective Federal Funds Rate'][i] = data_rate['Effective Federal Funds Rate'][i-1]
        if ch_rate1[i] == True:
            data_rate['Unemployment Rate'][i] = data_rate['Unemployment Rate'][i-1]
        if ch_rate2[i] == True:
            data_rate['Inflation Rate'][i] = data_rate['Inflation Rate'][i-1]
data_rate
```

Code

```
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Year                          894 non-null    int64
 1   Month                         894 non-null    int64
 2   Day                           894 non-null    int64
 3   Federal Funds Target Rate     462 non-null    float64
 4   Federal Funds Upper Target    103 non-null    float64
 5   Federal Funds Lower Target    103 non-null    float64
 6   Effective Federal Funds Rate  742 non-null    float64
 7   Real GDP (Percent Change)     247 non-null    float64
 8   Unemployment Rate             742 non-null    float64
 9   Inflation Rate                742 non-null    float64
dtypes: float64(7), int64(3)
```

```
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Year                          894 non-null    int64
 1   Month                         894 non-null    int64
 2   Day                           894 non-null    int64
 3   Federal Funds Target Rate     462 non-null    float64
 4   Federal Funds Upper Target    103 non-null    float64
 5   Federal Funds Lower Target    103 non-null    float64
 6   Effective Federal Funds Rate  894 non-null    float64
 7   Real GDP (Percent Change)     894 non-null    float64
 8   Unemployment Rate             894 non-null    float64
 9   Inflation Rate                894 non-null    float64
```

Part 4

예측과정과 결과 - 데이터 전처리(데이터셋 합치기)

```
#열추가
data_stack['Gdp'] = 0
data_stack['Funds Rate'] = 0
data_stack['Inflation Rate'] = 0
data_stack['Unemployment Rate'] = 0
data_stack['M2'] = 0
```

```
#값입력
c = 0
for j in range(0, len(data_m2)):
    #data_m2 년도 추출
    r_year_search = data_m2['DATE'][j][0]+data_m2['DATE'][j][1]+data_m2['DATE'][j][2]+data_m2['DATE'][j][3]
    r_month_search = data_m2['DATE'][j][5]+data_m2['DATE'][j][6]
    for i in range(0, len(data_stack)-2):
        try:
            #스탁 년도추출
            s_year_search = data_stack['Date'][c][0]+data_stack['Date'][c][1]+data_stack['Date'][c][2]+data_stack
             ['Date'][c][3]
            s_month_search = data_stack['Date'][c][5]+data_stack['Date'][c][6]
            #배치
            if str(s_year_search) == str(r_year_search):
                if str(s_month_search) == str(r_month_search):
                    data_stack['M2'][c] = data_m2['M2per'][j]
                    # print(f'{s_month_search} 에 ', data_rate['Real GDP (Percent Change)'][j])
                    c += 1
                else:
                    break
            else:
                break
        except:
            pass
df = data_stack.drop(['index', 'Open', 'High', 'Low', 'Close', 'Volume'], axis=1, inplace=True)
df#값
```

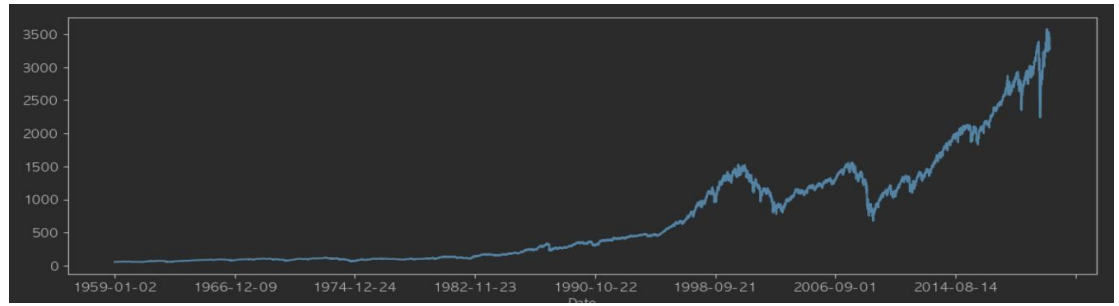
	Date	Adj Close	Gdp	Funds Rate	Inflation Rate	Unemployment Rate	M2
0	1959-01-02	55.439999	7.7	2.48	1.7	6.0	0.3
1	1959-01-05	55.660000	7.7	2.48	1.7	6.0	0.3
2	1959-01-06	55.590000	7.7	2.48	1.7	6.0	0.3
3	1959-01-07	54.889999	7.7	2.48	1.7	6.0	0.3
4	1959-01-08	55.400002	7.7	2.48	1.7	6.0	0.3
5	1959-01-09	55.770000	7.7	2.48	1.7	6.0	0.3
6	1959-01-12	55.779999	7.7	2.48	1.7	6.0	0.3
7	1959-01-13	55.470001	7.7	2.48	1.7	6.0	0.3
8	1959-01-14	55.619999	7.7	2.48	1.7	6.0	0.3

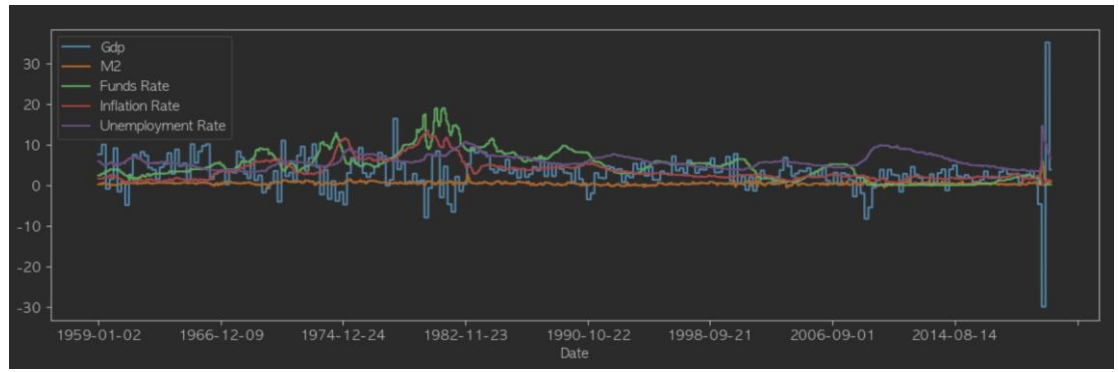
Code

Part 4

예측과정과 결과 - 데이터 시각화

```
import matplotlib.pyplot as plt # 데이터 시각화 모듈
#폰트 깨짐
plt.rcParams['font.family'] = 'AppleGothic'
#특수문자 깨짐
plt.rcParams['axes.unicode_minus'] = False
#크기 조절
plt.rcParams["figure.figsize"] = (14,4)
df.index = df['Date']
df['Adj Close'].plot()
df[['Gdp', 'M2','Funds Rate', 'Inflation Rate', 'Unemployment Rate']].plot()
```



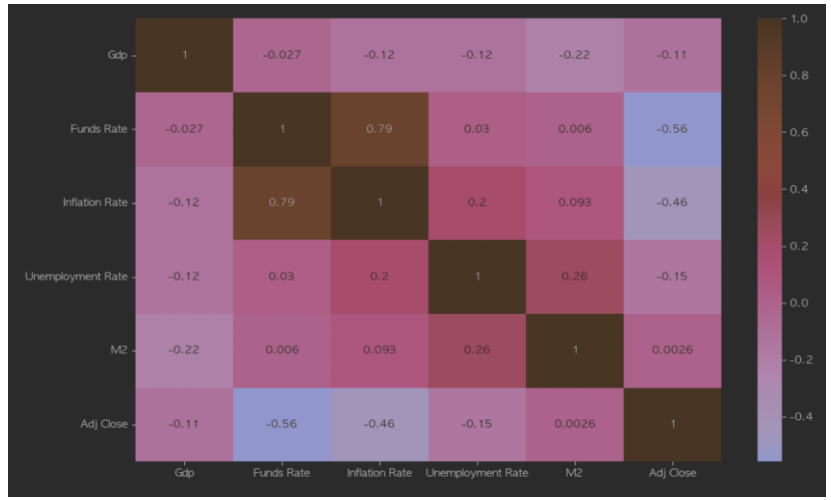


Code

Part 4

예측과정과 결과 - 상관관계 분석

```
import seaborn as sns # 데이터 시각화 모듈
#상관관계
plt.figure(figsize=(12,8))
sns.heatmap(df[['Gdp', 'Funds Rate', 'Inflation Rate',
                'Unemployment Rate', 'M2', 'Adj Close']].corr(),annot=True)
plt.show()
```



Code

Part 4

예측과정과 결과 - 정규화

```python
#정규화
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

# scale_cols = ['Adj Close', 'Funds Rate', 'Inflation Rate', 'Unemployment Rate', 'Gdp']
scale_cols = ['Funds Rate', 'Inflation Rate', 'Unemployment Rate', 'Gdp']
scaled_df = scaler.fit_transform(df[scale_cols])
scaled_df = pd.DataFrame(scaled_df, columns=scale_cols)
print(scaled_df)

scaler1 = RobustScaler()

scale_cols1 = ['Adj Close']
scaled1_df = scaler1.fit_transform(df[scale_cols1])
scaled1_df = pd.DataFrame(scaled1_df, columns=scale_cols1)
print(scaled1_df)
```

	Funds Rate	Inflation Rate	Unemployment Rate	Gdp
0	0.126642	0.118519	0.230088	0.576687
1	0.126642	0.118519	0.230088	0.576687
2	0.126642	0.118519	0.230088	0.576687
3	0.126642	0.118519	0.230088	0.576687
4	0.126642	0.118519	0.230088	0.576687
...	...	...	...	...
15564	0.009459	0.081481	0.309735	0.518405
15565	0.009459	0.081481	0.309735	0.518405
15566	0.009459	0.081481	0.309735	0.518405
15567	0.009459	0.081481	0.309735	0.518405

Code

Part 4

예측과정과 결과 - window 설정과 Test 데이터 분할

```
feature_cols = ['Funds Rate', 'Unemployment Rate', 'Inflation Rate', 'Gdp']
label_cols = ['Adj Close']

feature_df = pd.DataFrame(scaled_df, columns=feature_cols)
label_df = pd.DataFrame(scaled1_df, columns=label_cols)

feature_np = feature_df.to_numpy()
label_np = label_df.to_numpy()
```

```
import numpy as np
window_size = 90
def make_sequene_dataset(feature, label, window_size):

    feature_list = []
    label_list = []

    for i in range(len(feature)-window_size):

        feature_list.append(feature[i:i+window_size])
        label_list.append(label[i+window_size])

    return np.array(feature_list), np.array(label_list)

X, Y = make_sequene_dataset(feature_np, label_np, window_size)

print(X.shape, Y.shape)
```

Code

```
split = -3500

x_train = X[0:split]
y_train = Y[0:split]

x_test = X[split:]
y_test = Y[split:]

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

Code

Part 4

예측과정과 결과 - 모델 설계와 학습

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
model = Sequential()

model.add(LSTM(128, input_shape=x_train[0].shape,  return_sequences = True))
model.add(LSTM(64))
model.add(Dense(1, activation='tanh'))
model.compile(loss='mse', optimizer='adam', metrics='mae')

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 90, 128)           68096

 lstm_1 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 1)                 65

=================================================================
```

```python
hit = model.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          epochs=25, batch_size=16,)
```

```
val_mae: 1.0077
Epoch 22/25
749/749 [==============================] - 49s 65ms/step - loss: 5.3638e-04 - mae: 0.0144 - val_loss: 1.4887
- val_mae: 1.0341
Epoch 23/25
749/749 [==============================] - 49s 65ms/step - loss: 4.3087e-04 - mae: 0.0131 - val_loss: 1.5854
- val_mae: 1.0667
Epoch 24/25
749/749 [==============================] - 49s 65ms/step - loss: 4.1692e-04 - mae: 0.0124 - val_loss: 1.5614
- val_mae: 1.0546
Epoch 25/25
749/749 [==============================] - 49s 65ms/step - loss: 0.0134 - mae: 0.0445 - val_loss: 1.4193 -
```

Code

Part 4

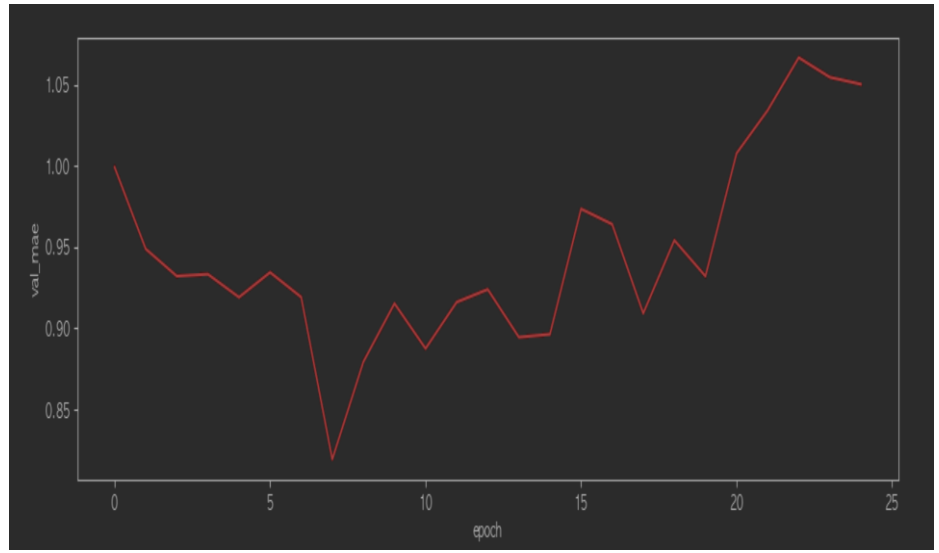
예측과정과 결과 - 학습

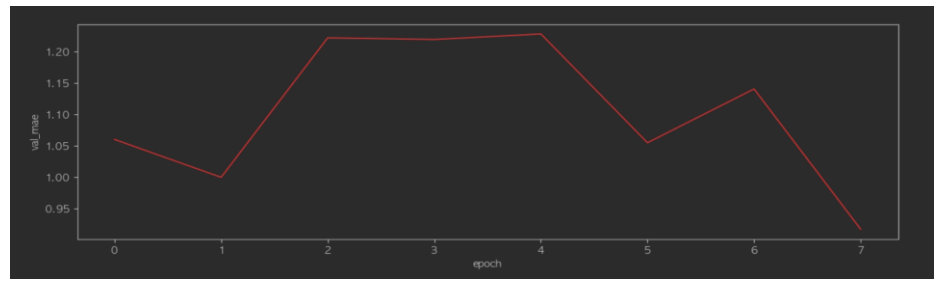
```
fig, loss_ax = plt.subplots()

loss_ax.plot(hit.history['val_mae'], 'r', label='mae')
loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('val_mae')

plt.show()
```

Code





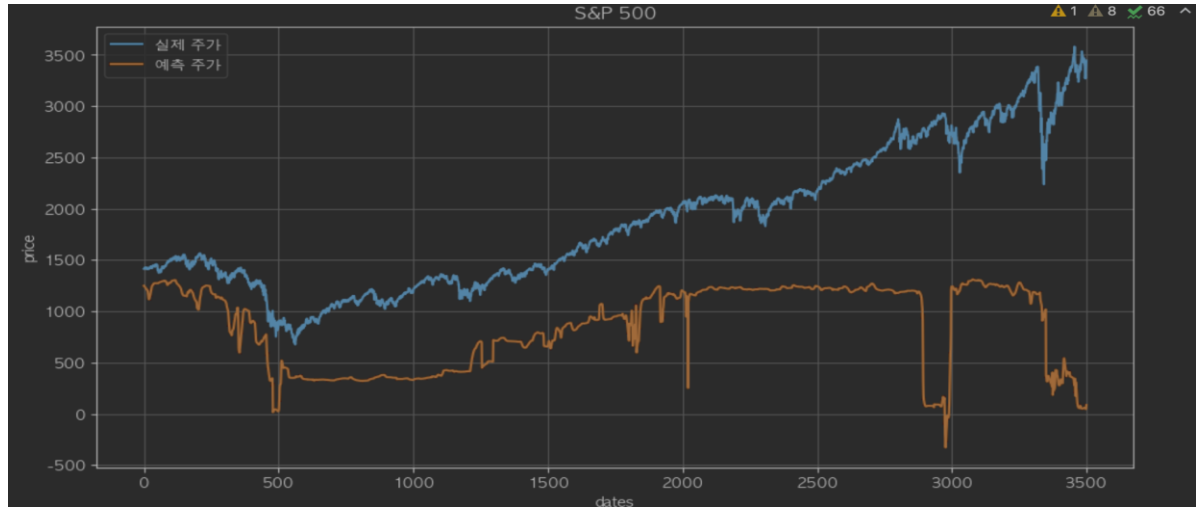
Part 4

예측과정과 결과 - 결과 및 테스트

```python
#정규화 역변환 함수
y_test = scaler1.inverse_transform(y_test)
#모델 출력
pred = model.predict(x_test)
#모델 정규화 역변환
pred = scaler1.inverse_transform(pred)

plt.figure(figsize=(12, 6))
plt.title('S&P 500')
plt.ylabel('price')
plt.xlabel('dates')
plt.plot(y_test, label='실제 주가')
plt.plot(pred, label='예측 주가')
plt.grid()
plt.legend(loc='best')

plt.show()
```



Code

결론

예측값과 실제값의 괴리가 상당히 벌어져 있었는데 원인으로는 데이터셋 자체가 주식가격에 후행하는 즉 데이터 즉 주식가격보다 늦게 반영되는 데이터 이고 예측하는 기간이 길어져 맞추지 못하는 기간이 늘어나 이로인해 격차가 점점 벌어지기 때문으로 추측된다.

감사합니다