3가지 모델을 활용한 ETF 종목 주가예측

싹쓰리
김제환, 정해빈, 최희녕

CONTENTS

1. 팀 목표 및 진행 과정

2. 시계열 3가지 모델 분석
   - Prophet
   - LSTM
   - Arima

3. 성능평가 & 결론

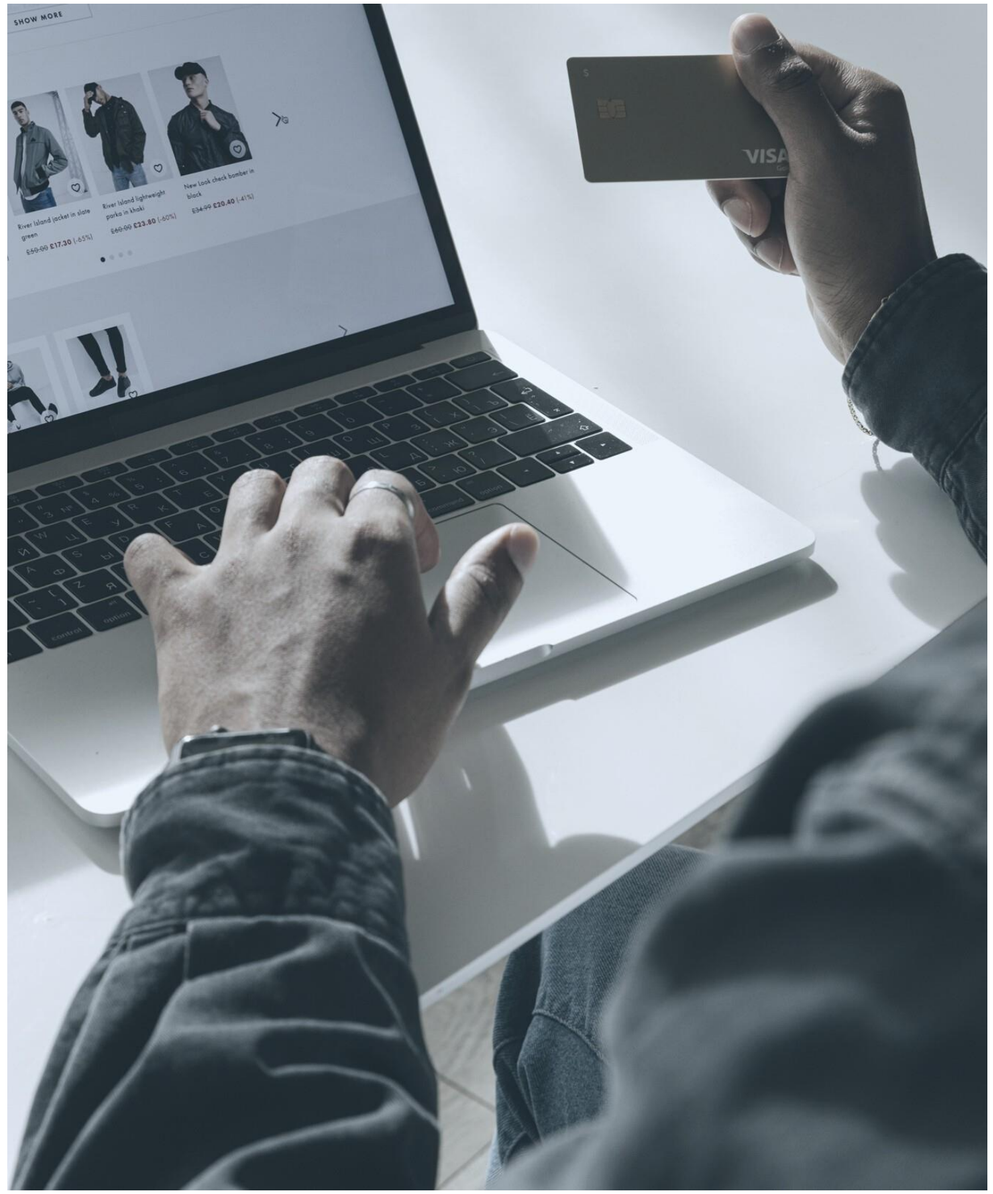
1.
팀 목표 와 진행 과정

제작동기

학습한 머신러닝을 이용하여 모델별 특징 및 예측율 확인

목표

지난 ETF주가 데이터를
3가지 모델을 활용하여 다음주
주가 예측 및 성능 비교



팀원 및 역할 담당

싹쓰리 팀원 소개

김제환

팀장 및 Arima모델링
전체적인 코드검정

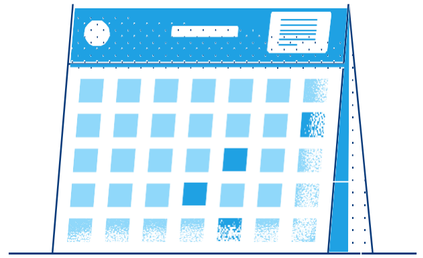
정해빈

프로젝트 계획수립
Prophet모델링

최희녕

LSTM 모형 모델링
보고자료 제작

프로젝트 일정



금요일 월요일 화요일 수요일 목요일 금요일 월요일

ETF 자료수집

데이터 전처리

모델별 코드작성

데이터 정규화

최종 발표

데이터 선정

데이터베이스

KODEX200(069500) 시세추이

기간

2022.06.24~ 07.29

사용모델

Prophet, LSTM , Arima

“

2022.07.25~ 2022.07.29
주가 종가 예측

”

2.
시계열 3가지 모델 분석

1

Prophet

트렌드를 포함된 데이터에 대한 정확한 예측이 가능함

$$y(t) = g(t) + s(t) + h(t) + et$$

페이스북에서 개발한 시계열 예측모형

g(t) 반복적인 요소를 가지지 않은 트렌드
s(t) 요일 혹은 연 계절성과 같은 반복적인 변화
h(t) Holiday와 같이 가끔씩 불규칙하게 영향을 미치는 요소
e는 정규분포를 따르는 잔차라고 가정

1

Prophet

트렌드를 포함된 데이터에 대한 정확한 예측이 가능함

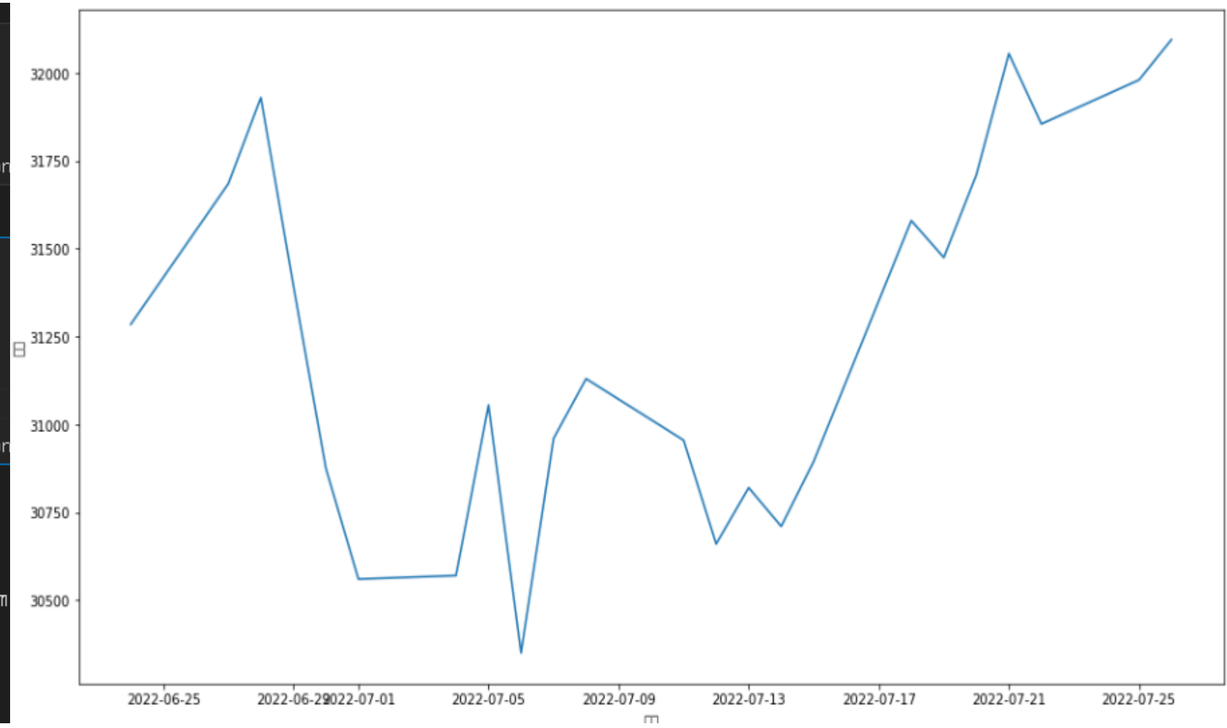
데이터 로드 및 분포 확인

```
    # 시각화를 위해 일자를 날짜형으로 변환
    df['일자'] = pd.to_datetime(df['일자'], format='%Y-%m-%d')
    df['월별'] = df['일자'].dt.month
[90]                                                                Python
```

```
    # 종가 시각화
    plt.figure(figsize=(16, 9))
    sns.lineplot(y=df['종가'], x=df['일자'])
    plt.xlabel('일자')
    plt.ylabel('종가')
[93]                                                                Python
```

```
…   Text(0, 0.5, '종가')

    c:\Users\HaeBin\AppData\Local\Programs\Python\Python310\lib\site-
    packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 51068 (\N{HANGUL SYLLABLE IL}) missing from
    current font.
      fig.canvas.print_figure(bytes_io, **kw)
    c:\Users\HaeBin\AppData\Local\Programs\Python\Python310\lib\site-
```



Kodex200 종가 자료 불러오기

1

Prophet

트렌드를 포함된 데이터에 대한 정확한 예측이 가능함

데이터 로드 및 분포 확인

```
import numpy as np
import pandas as pd

import FinanceDataReader as fdr
import matplotlib.pyplot as plt
import seaborn as sns

## FB - Prophet 가져오기
from prophet import Prophet
## interactive한 plot 만들기 위한 모듈입니다.
## plot_components_plotly는 Season별 시각화하기 위한 모듈입니다.
from prophet.plot import plot_plotly, plot_components_plotly
from prophet.plot import add_changepoints_to_plot

import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

```
# PUBlIC

probhet_df = mean_change.reset_index()
probhet_df.columns = ['ds','y']
probhet_df['ds'] = pd.to_datetime(probhet_df['ds'])
m = Prophet()
m.fit(probhet_df)
# public data 예측
public_probhet_df = pd.DataFrame()
public_probhet_df['ds'] = ['2022-07-18','2022-07-19','2022-07-20','2022-07-21','2022-07-22']
public_probhet_df['ds'] = pd.to_datetime(public_probhet_df['ds'])
public_result = m.predict(public_probhet_df)

# private data 예측
private_probhet_df = pd.DataFrame()
private_probhet_df['ds'] = ['2022-07-18','2022-07-19','2022-07-20','2022-07-21','2022-07-22']
private_probhet_df['ds'] = pd.to_datetime(private_probhet_df['ds'])
result = m.predict(private_probhet_df)

# 여러번의 실험 결과 월요일의 예측율이 가장 맞추기 힘들었습니다..
start_idx = 1
```

Prophet 모델 학습

1

Prophet

트렌드를 포함된 데이터에 대한 정확한 예측이 가능함

데이터 로드 및 분포 확인

```
# train set 최근 한달 생성
last_1month = list()
for i in range(18, 23):
    last_1month.append(['2022-07-%02d' % i])
last_1month = pd.DataFrame(last_1month, columns = ['ds'])
last_1month['ds']= pd.to_datetime(last_1month['ds'])
```

```
# 예측
forecast = model.predict(last_1month)
```

```
# 예측
forecast = model.predict(last_1month)
```

```
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']])
```

	ds	yhat	yhat_lower	yhat_upper
0	2022-07-18	31455.618579	31099.532121	31772.005745
1	2022-07-19	31617.631175	31265.934967	31969.242725
2	2022-07-20	31522.866284	31201.377123	31855.586667
3	2022-07-21	31672.331084	31326.059587	31978.409742
4	2022-07-22	31642.853501	31316.163418	31976.616799

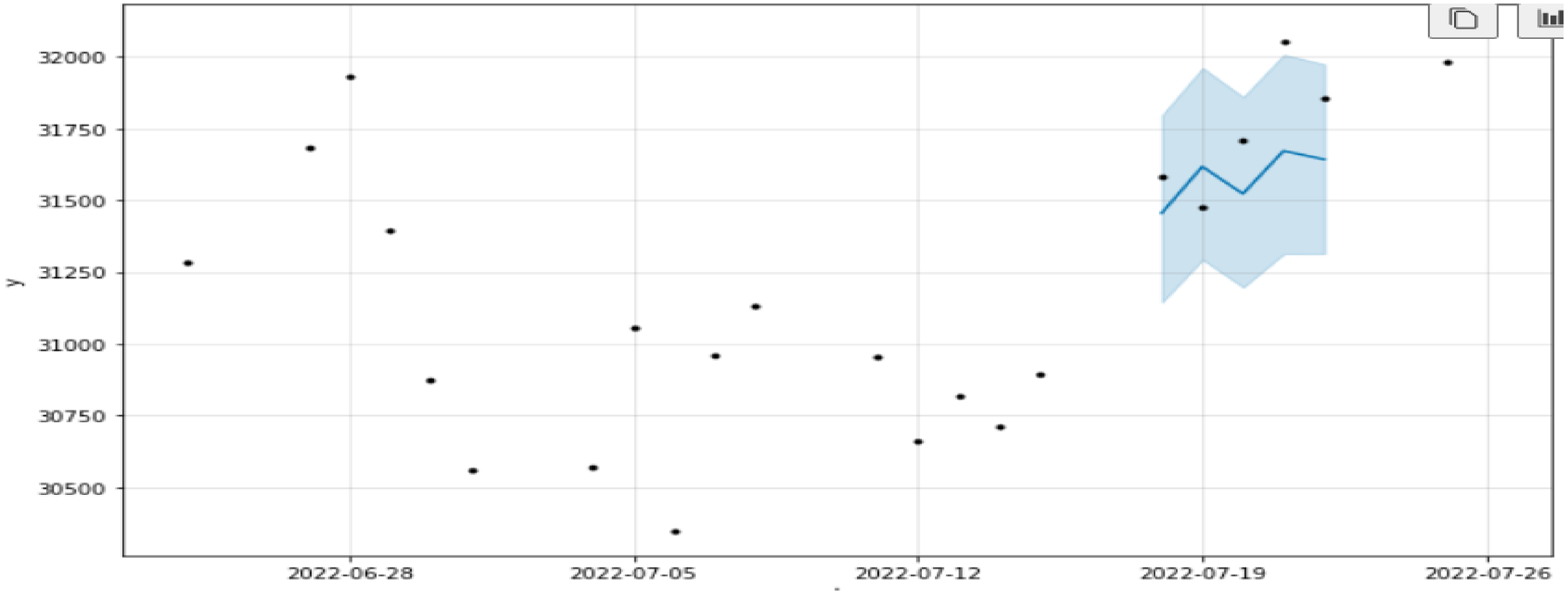
Simple test 로 18일에서 22일 주가 비교

1

Prophet

트렌드를 포함된 데이터에 대한 정확한 예측이 가능함

데이터 로드 및 분포 확인



Simple test 로 18일에서 22일 예측 결과

1

Prophet

트렌드를 포함된 데이터에 대한 정확한 예측이 가능함

데이터 로드 및 분포 확인

```
# 정규화
from matplotlib.pyplot import sca
from sklearn.preprocessing import MinMaxScaler
import numpy as np



df.sort_index(ascending=False).reset_index(drop=True)
scaler = MinMaxScaler()
scale_cols = ['시가', '고가', '저가', '종가', '거래량']
df_scaled = scaler.fit_transform(df[scale_cols])
df_scaled = pd.DataFrame(df_scaled)
df_scaled.columns = scale_cols

a = df_scaled
a
```

```
result = pd.concat([b,a] , axis = 1)
result
```

	일자	시가	고가	저가	종가	거래량
0	2022-07-26	0.918728	0.947917	0.996774	1.000000	0.000000
1	2022-07-25	0.833922	0.961806	0.941935	0.934097	0.326621
2	2022-07-22	1.000000	1.000000	1.000000	0.862464	0.484868
3	2022-07-21	0.837456	0.920139	0.938710	0.977077	0.400147
4	2022-07-20	1.000000	0.913194	0.893548	0.779370	0.554284
5	2022-07-19	0.625442	0.520833	0.661290	0.644699	0.294282
6	2022-07-18	0.448763	0.600694	0.577419	0.704871	0.601195
7	2022-07-15	0.233216	0.128472	0.045161	0.312321	0.608723
8	2022-07-14	0.014134	0.128472	0.151613	0.206304	0.442463
9	2022-07-13	0.116608	0.194444	0.254839	0.269341	0.557988
10	2022-07-12	0.183746	0.104167	0.148387	0.177650	0.400239
11	2022-07-11	0.445230	0.399306	0.425806	0.346705	0.524948
12	2022-07-08	0.434629	0.475694	0.558065	0.446991	0.802324

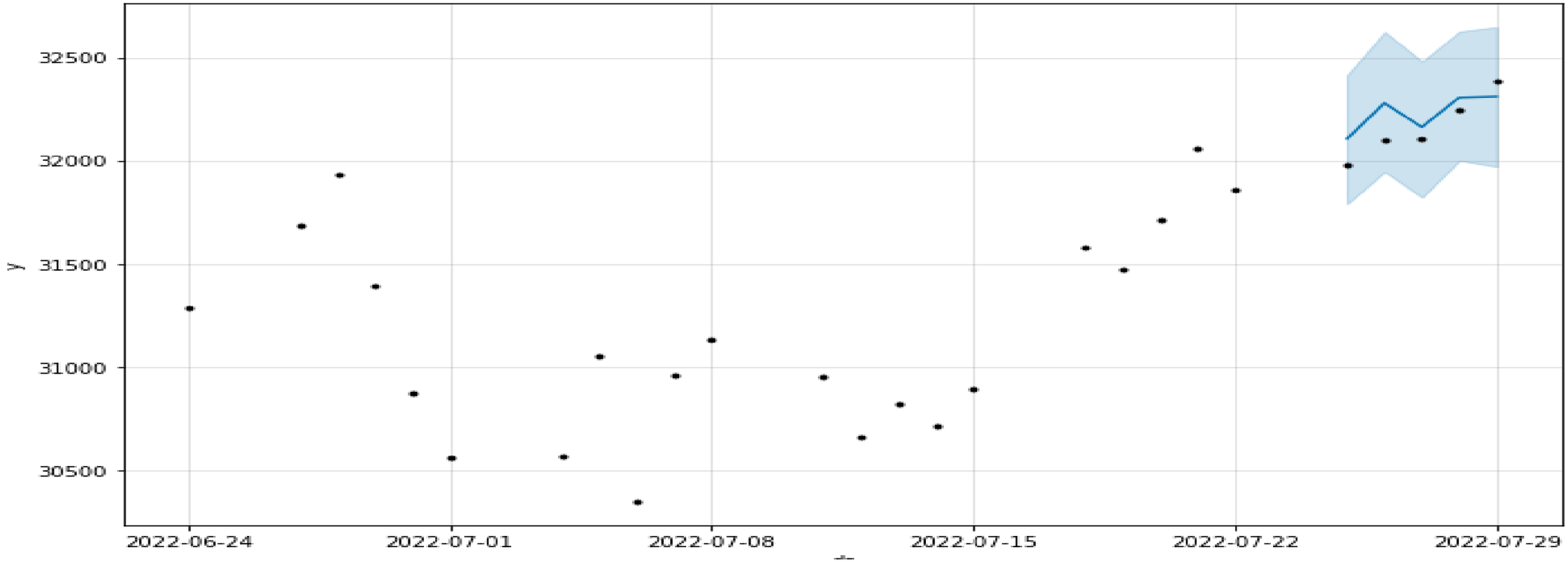
25일에서 29일 주가 예측 정규화

1

Prophet

트렌드를 포함된 데이터에 대한 정확한 예측이 가능함

데이터 로드 및 분포 확인



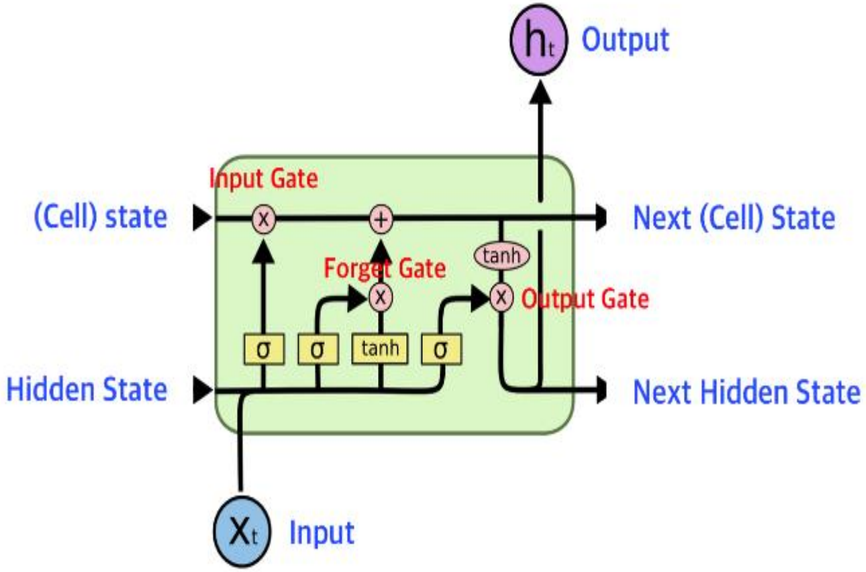
Prophet 최종 25일~ 29일 주가 예측

2

LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습

LSTM :
Long Short Term Memory의 줄임말로 주로 시계열 처리나 자연어 처리



LSTM 구조

중요) 입력값과 은닉 상태에 곱해지는 가중치가 다름

Foret Gate(삭제 게이트): 셀 상태에 있는 정보를 제거

Input Gate(입력 게이트): 새로운 정보를 셀 상태에 추가

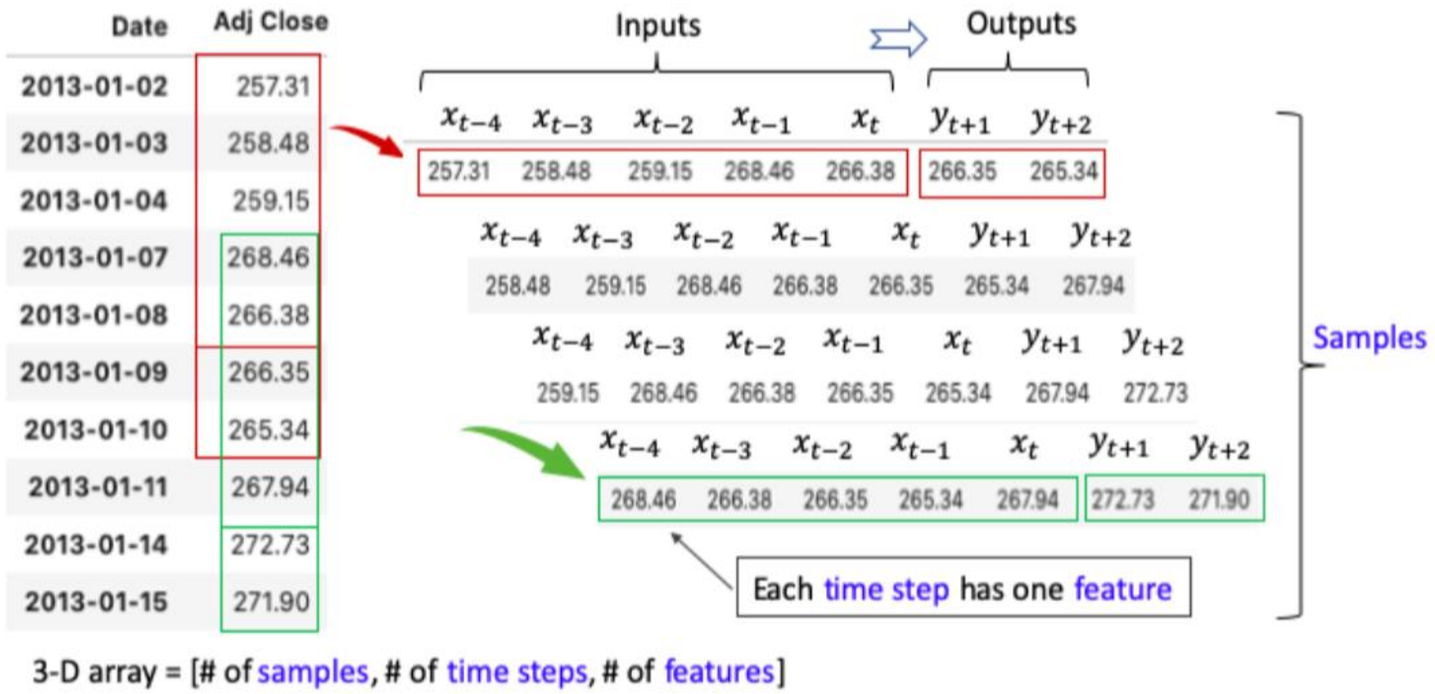
Output Gate(출력 게이트): 셀 상태가 다음 은닉 상태로 출력

2

LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습

LSTM/GRU 에서 활용하는 입력-출력 구조 : 3차원 데이터



①samples

데이터의 크기이며 원본 데이터를 window size에 따라 슬라이싱 할 경우 생기는 데이터의 개수

②time steps

과거 몇개의 데이터를 볼 것인가를 나타내며 네트워크에 사용할 시간 단위

③features

X의 차원을 의미. 쉽게 말해 X의 변수 갯수

2

LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습

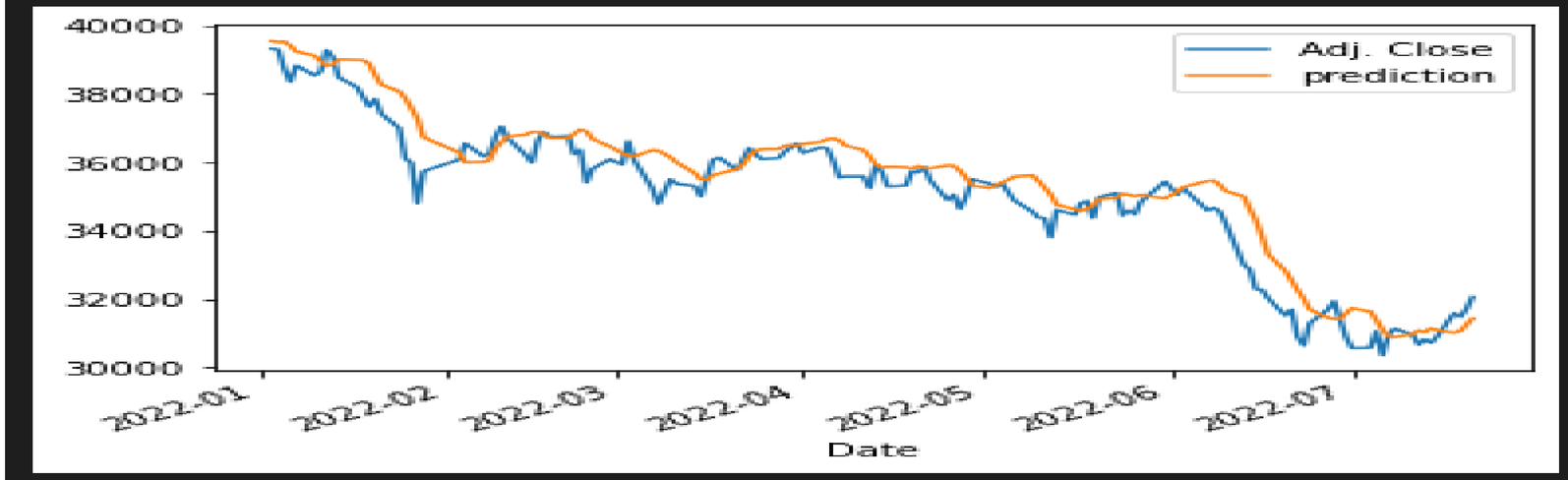
데이터 로드 및 분포 확인

```
c:\Users\82104\AppData\Local\Programs\Python\Python39\lib\site-pac
  super(SGD, self).__init__(name, **kwargs)

5/5 [==============================] - 1s 2ms/step

(577361.25, <AxesSubplot:xlabel='Date'>)

<Figure size 1152x648 with 0 Axes>
```



2

LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습

1）Minmax Scaler

```
Epoch 6: val_loss improved from 0.00075 to 0.00069, saving model to model\tmp_checkpoint.h5
49/49 [==============================] - 0s 7ms/step - loss: 9.0197e-04 - val_loss: 6.9293e-04
Epoch 7/200
...
Epoch 86/200
49/49 [==============================] - ETA: 0s - loss: 3.3830e-04
Epoch 86: val_loss did not improve from 0.00022
49/49 [==============================] - 0s 6ms/step - loss: 3.3830e-04 - val_loss: 2.2171e-04
```

▶MinMax Scaler를 처리한 loss 값

```
 64/598 [==>...........................] - ETA: 0s - loss: 0.0019
192/598 [========>.....................] - ETA: 0s - loss: 0.0020
320/598 [==============>...............] - ETA: 0s - loss: 0.0030
448/598 [====================>.........] - ETA: 0s - loss: 0.0039
576/598 [==========================>..] - ETA: 0s - loss: 0.0042
598/598 [==============================] - 0s 710us/step - loss: 0.0041
```

▶MinMax Scaler를 처리하지 않은 loss 값

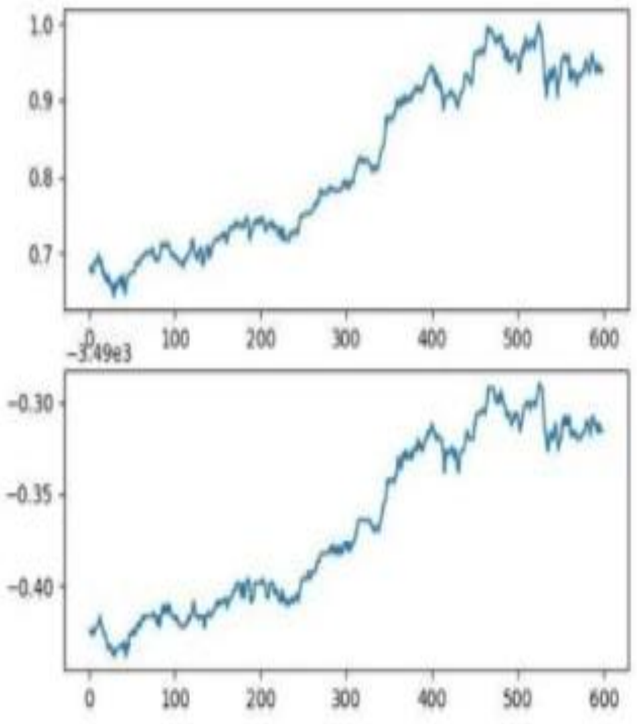
출처:PARK SUNG MIN
머신러닝 프로젝트 - LSTM을 이용한 주가 (KODEX200) 예측 (slideshare.net)

2

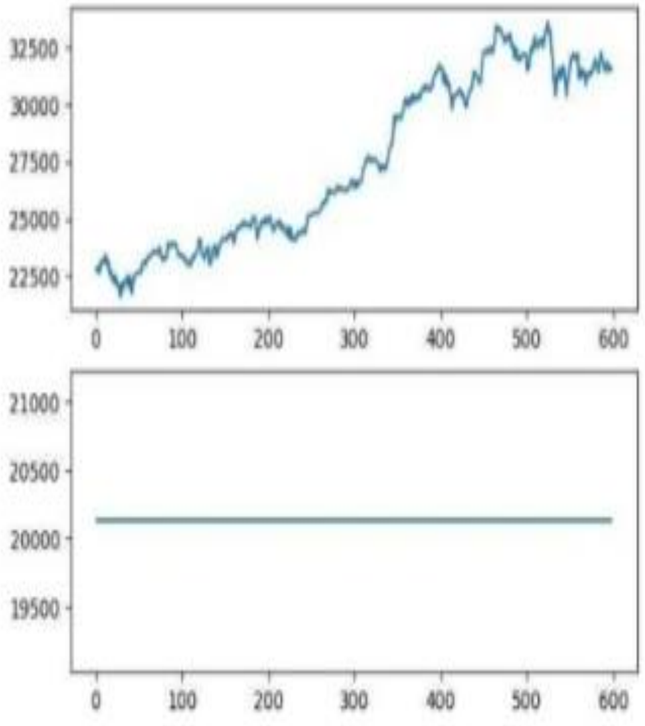
LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습

1）Minmax Scaler



▶MinMax Scaler를 처리한 loss 값



▶MinMax Scaler를 처리하지 않은 loss 값

결론 1)Minmax scaler처리를 해야 데이터의 크기가 조절되어 loss값을 줄여 줄 수 있다.

출처:PARK SUNG MIN
머신러닝 프로젝트 - LSTM을 이용한 주가(KODEX200) 예측 (slideshare.net)

2

LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습

2)LSTM신경망 크기

```
def build_model(self):
    model = Sequential()
    model.add(LSTM(2048, input_shape=[1,4], activation='tanh'))
model.add(Dense(1,activation = 'linear'))

model.summary()
model.compile(rmsprop(lr=self.learning_rate), loss='mse')

return model
```

```
def build_model(self):
    model = Sequential()
    model.add(LSTM(1024, input_shape=[1,4], activation='tanh'))
model.add(Dense(1,activation = 'linear'))

model.summary()
model.compile(rmsprop(lr=self.learning_rate), loss='mse')

return model
```

2

LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습

2)LSTM신경망 크기

```
 64/598 [==>............................] - ETA: 0s - loss: 0.0019
192/598 [========>.....................] - ETA: 0s - loss: 0.0014
320/598 [==============>...............] - ETA: 0s - loss: 0.0014
448/598 [====================>.........] - ETA: 0s - loss: 0.0017
576/598 [==========================>...] - ETA: 0s - loss: 0.0031
598/598 [==============================] - 0s 710us/step - loss: 0.0030
```

```
예측 값 =  31789.6
실제 값 =  31645
```

```
 64/598 [==>............................] - ETA: 0s - loss: 0.0013
320/598 [==============>...............] - ETA: 0s - loss: 9.9835e-04
576/598 [==========================>...] - ETA: 0s - loss: 0.0039
598/598 [==============================] - 0s 230us/step - loss: 0.0040
```

```
예측 값 =  32107.0
실제 값 =  31645
```

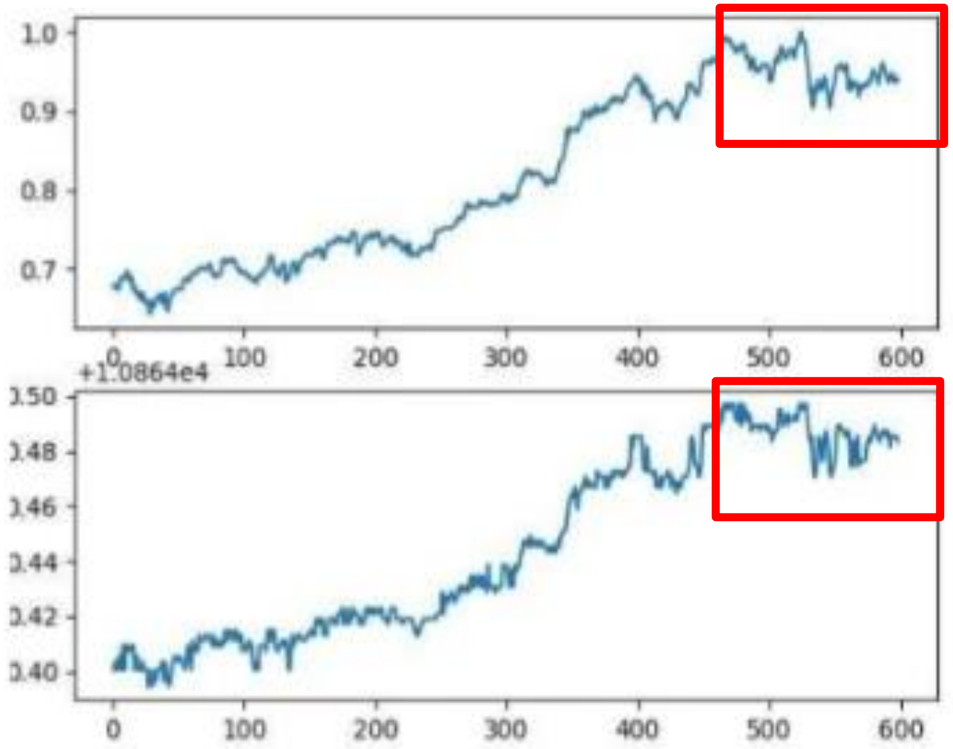
결론 2)신경망의 크기가 큰 경우가 평균적으로 더 잘 학습된 결과를 보여준다.

출처:PARK SUNG MIN
머신러닝 프로젝트 - LSTM을 이용한 주가(KODEX200) 예측 (slideshare.net)

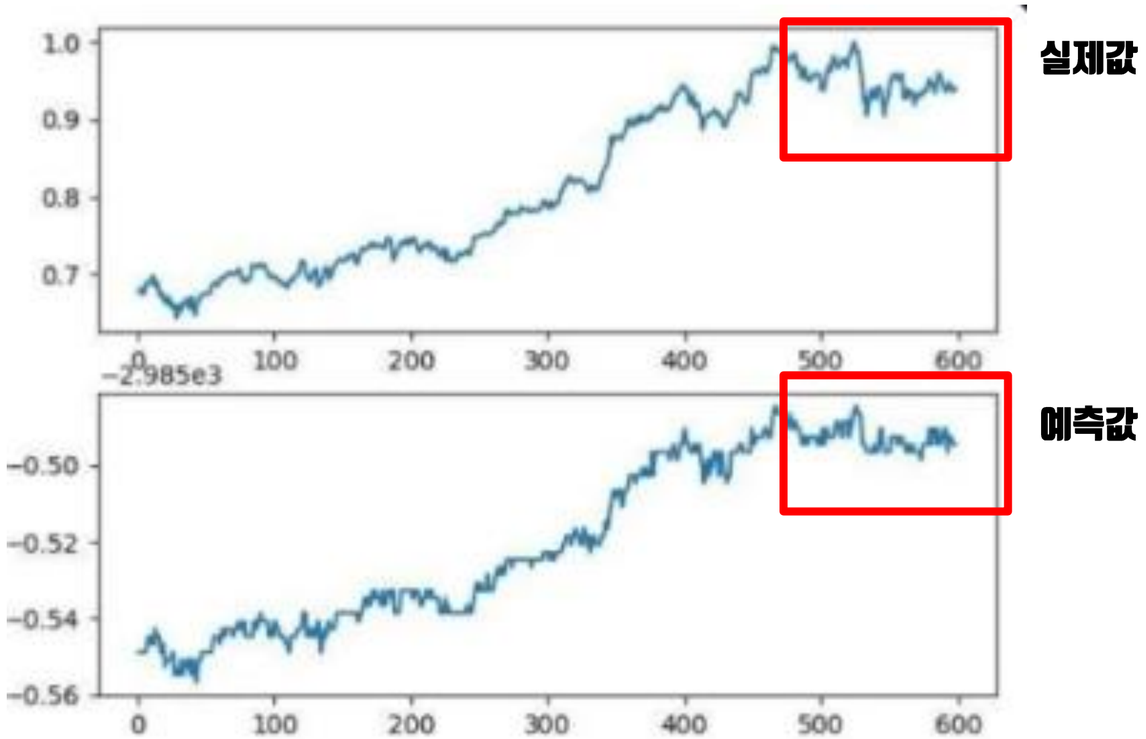
2

LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습



▶신경망의 크기:2048



▶신경망의 크기:1024

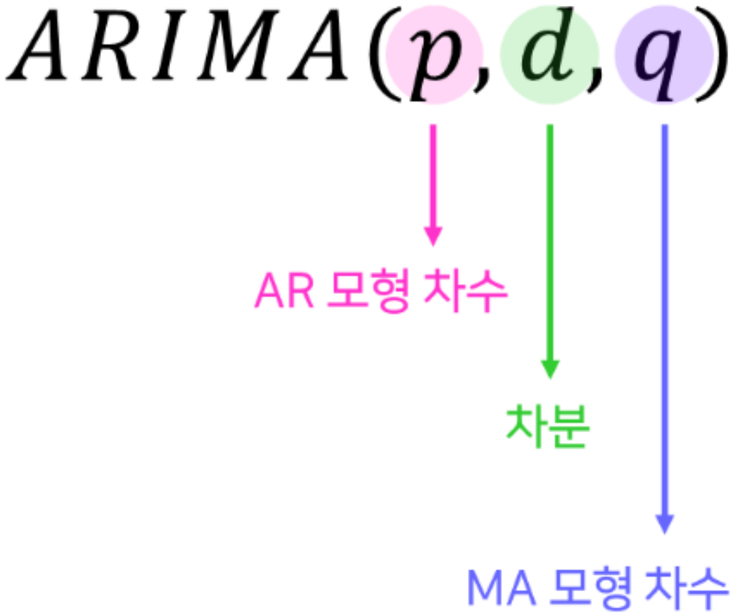
3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

Arima모형 : 자동회귀 누적이동평균

현재와 추세간의 관계로 시계열 데이터를 잘 예측하지 못해 한계를 보완하기 위해 등장



ARIMA는 차분, 변환을 통해

AR, MA, ARMA로 정상화

- p=0이면 IMA(d,q) -> d번 차분하면 MA(q)
- d=0이면 ARMA(p,q) -> 정상성 만족
- q=0이면 ARI(p,d) -> d번 차분하면 AR(p)

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import FinanceDataReader as fdr
from statsmodels.tsa.stattools import adfuller
```

```
kodex_200 = pd.read_csv('../헤커톤/KOdex_200.csv', encoding='euc-kr')
kodex_200.head()
```

✓ 0.1s

	Date	Open	High	Low	Close	Volume	Change
0	2022-06-24	30732	31341	30633	31217	10474792	0.021064
1	2022-06-27	31451	31860	31166	31616	8591446	0.012781
2	2022-06-28	31655	31860	31491	31860	6761245	0.007718
3	2022-06-29	31486	31585	31296	31327	6191026	-0.016729
4	2022-06-30	31211	31211	30802	30808	7957762	-0.016567

```
kodex_200 = kodex_200.sort_values(by='Date')
kodex_200['Date'] = pd.to_datetime(kodex_200['Date'])
kodex_200.index = kodex_200['Date']
kodex_200.set_index('Date', inplace=True)
kodex_200.head()
```

✓ 0.7s

Date	Open	High	Low	Close	Volume	Change
2022-06-24	30732	31341	30633	31217	10474792	0.021064
2022-06-27	31451	31860	31166	31616	8591446	0.012781
2022-06-28	31655	31860	31491	31860	6761245	0.007718
2022-06-29	31486	31585	31296	31327	6191026	-0.016729
2022-06-30	31211	31211	30802	30808	7957762	-0.016567

모듈생성 및 주가 불러오기

3

ARIMA

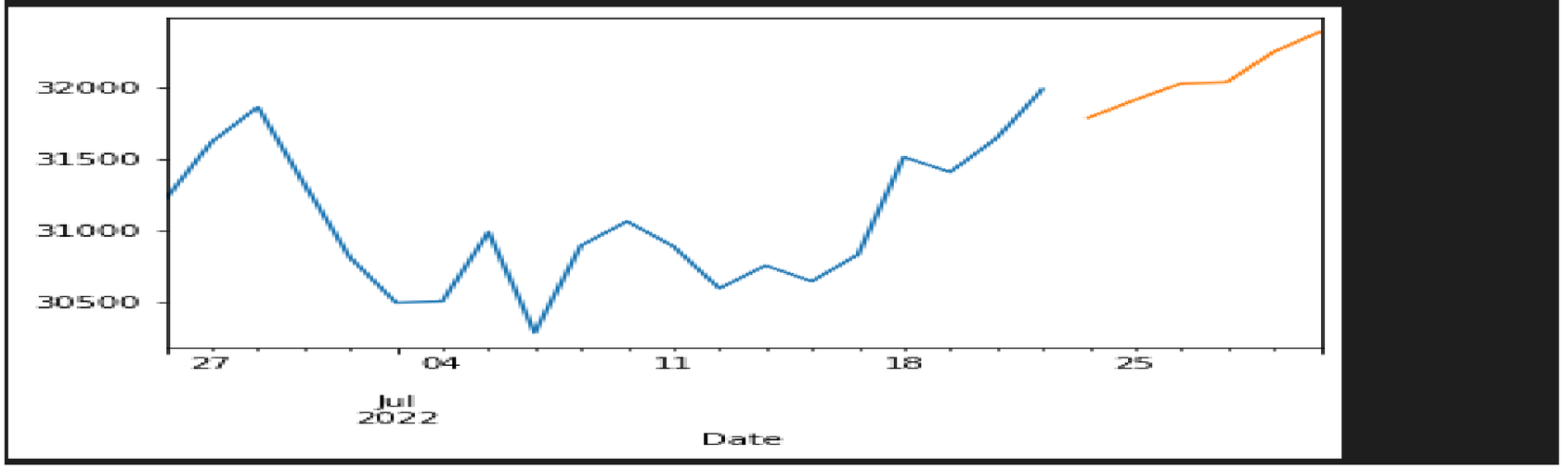
시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
import matplotlib.pyplot as plt
y_train = kodex_200['Close'][:int(0.8*len(kodex_200))]
y_test = kodex_200['Close'][int(0.8*len(kodex_200)):]
y_train.plot()
y_test.plot()
```

0.3s

```
<AxesSubplot:xlabel='Date'>
```



종가 데이터 시각화

3

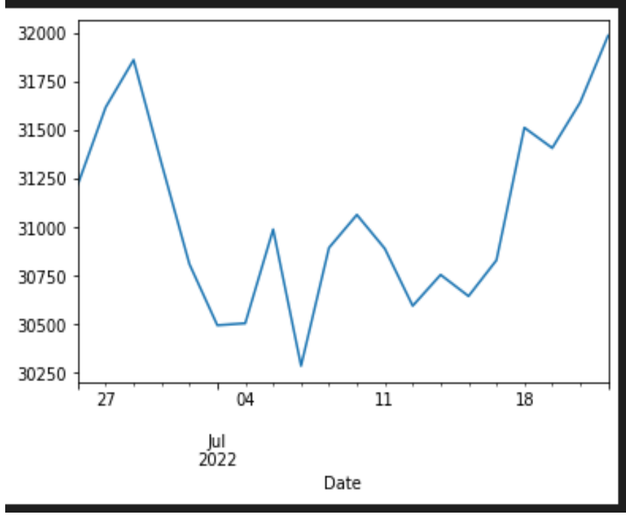
ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

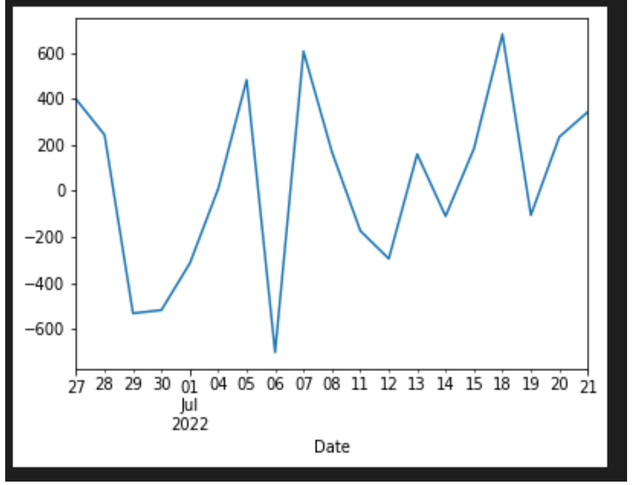
```
y_train.plot()
```
✓ 0.1s

```
<AxesSubplot:xlabel='Date'>
```



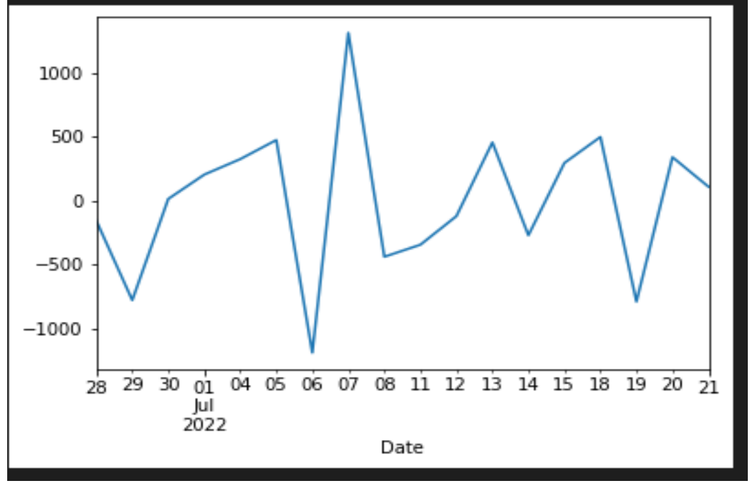

```
diff_1 = y_train.diff().dropna()
diff_1.plot()
```
✓ 0.1s

```
<AxesSubplot:xlabel='Date'>
```




```
diff_2 = diff_1.diff().dropna()
diff_2.plot()
```
✓ 0.1s

```
<AxesSubplot:xlabel='Date'>
```



차분을 통해 비정상시계열데이터 정상성으로 변환

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
from statsmodels.tsa.stattools import adfuller
dataX = y_train.values
fuller = adfuller(dataX)
print('ADF Statistic: %f' % fuller[0])
print('p-value: %f' % fuller[1])
print('Critical Values:')
for key, value in fuller[4].items():
    print('\t%s: %.3f' % (key, value))
```

✓ 0.5s

```
ADF Statistic: 1.460105
p-value: 0.997371
Critical Values:
        1%: -4.223
        5%: -3.189
        10%: -2.730
```

```
from statsmodels.tsa.stattools import adfuller
dataX = diff_1.values
fuller = adfuller(dataX)
print('ADF Statistic: %f' % fuller[0])
print('p-value: %f' % fuller[1])
print('Critical Values:')
for key, value in fuller[4].items():
    print('\t%s: %.3f' % (key, value))
```

✓ 0.4s

```
ADF Statistic: -4.364598
p-value: 0.000343
Critical Values:
        1%: -3.859
        5%: -3.042
        10%: -2.661
```

```
from statsmodels.tsa.stattools import adfuller
dataX = diff_2.values
fuller = adfuller(dataX)
print('ADF Statistic: %f' % fuller[0])
print('p-value: %f' % fuller[1])
print('Critical Values:')
for key, value in fuller[4].items():
    print('\t%s: %.3f' % (key, value))
```

✓ 0.7s

```
ADF Statistic: -9.081100
p-value: 0.000000
Critical Values:
        1%: -4.332
        5%: -3.233
        10%: -2.749
```

차분을 통해 비정상시계열데이터 정상성으로 변환

3

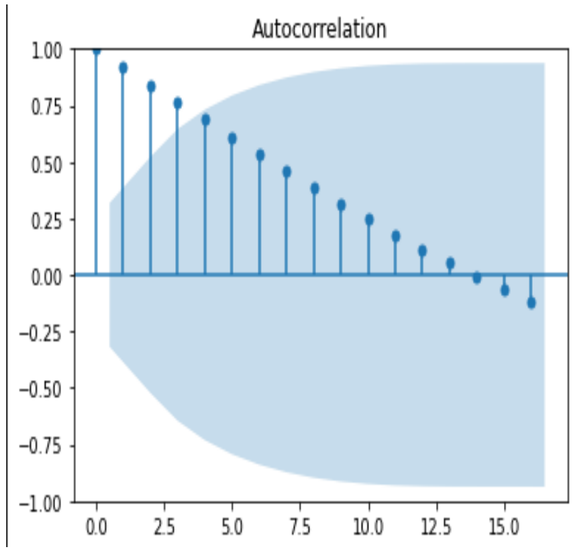
ARIMA

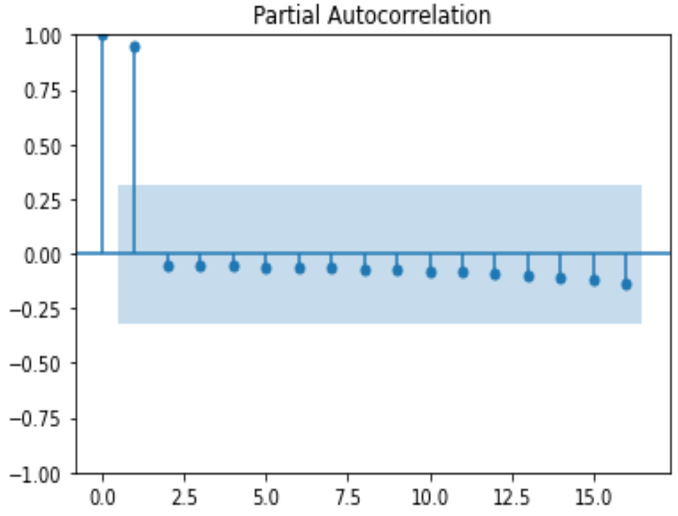
시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
import numpy as np
lags=np.arange(len(diff_2)*2)
plot_acf(lags)
plot_pacf(lags)
plt.show()
```

✓ 0.2s





ACF, PACF로 AR, MA 구하기

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm

model = sm.tsa.arima.ARIMA(y_train, order=(1,2,0))
model_fit = model.fit()
print(model_fit.summary())
```

✓ 0.1s

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                   20
Model:                 ARIMA(1, 2, 0)   Log Likelihood                -139.583
Date:                Mon, 01 Aug 2022   AIC                            283.167
Time:                        09:33:18   BIC                            284.948
Sample:                    06-24-2022   HQIC                           283.412
                         - 07-21-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0429      0.115     -0.373      0.709      -0.268       0.182
sigma2      3.098e+05   1.19e+05      2.593      0.010    7.56e+04    5.44e+05
===================================================================================
Ljung-Box (L1) (Q):                   3.88   Jarque-Bera (JB):                 0.00
Prob(Q):                              0.05   Prob(JB):                         1.00
Heteroskedasticity (H):               0.93   Skew:                             0.01
Prob(H) (two-sided):                  0.93   Kurtosis:                         2.94
===================================================================================
```

ARIMA모델 학습

3

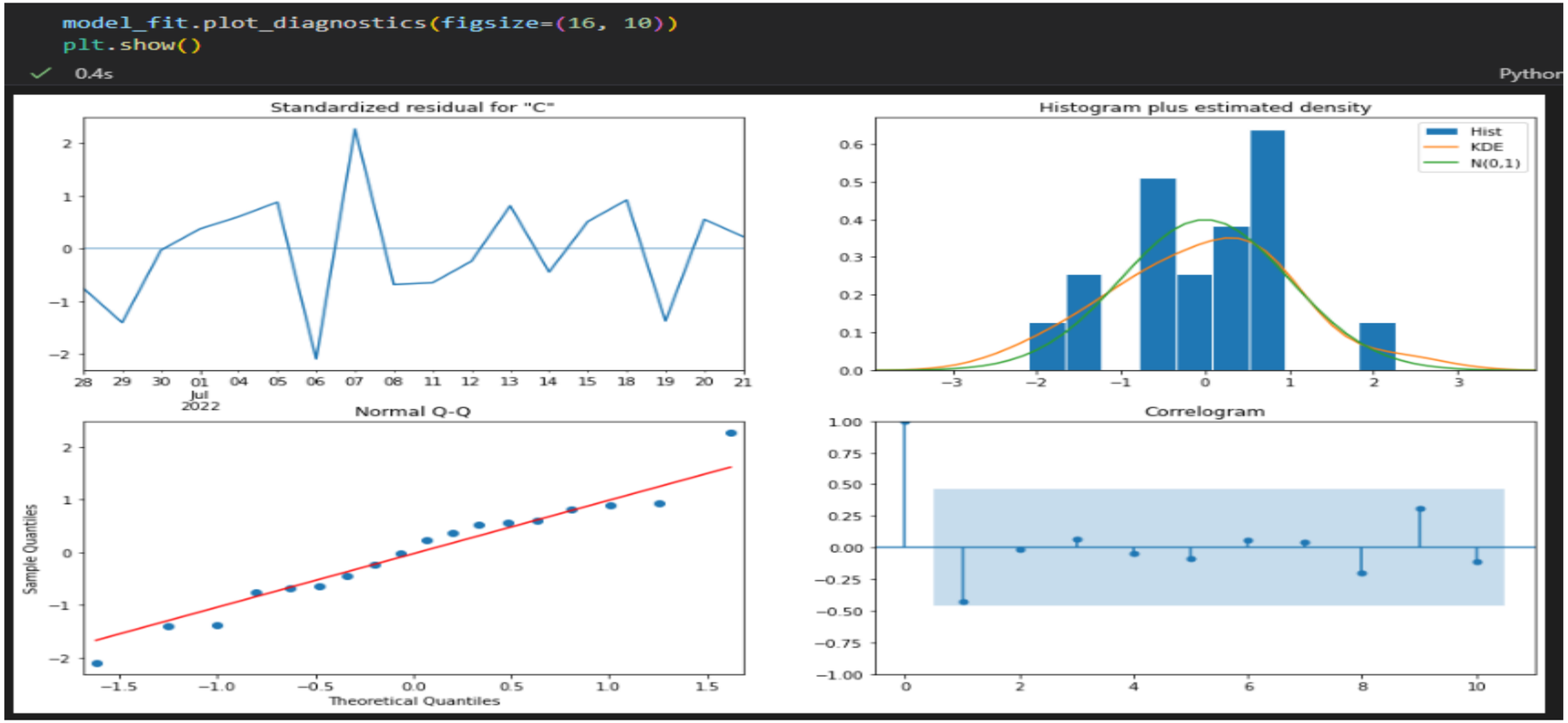
ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
model_fit.plot_diagnostics(figsize=(16, 10))
plt.show()
```

0.4s



학습한 모델 시각화

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
pred = model_fit.forecast(steps=6).astype(int)
print(pred)
```

✓ 0.6s

```
2022-07-22    32324
2022-07-25    32663
2022-07-26    33003
2022-07-27    33342
2022-07-28    33682
2022-07-29    34021
Freq: B, Name: predicted_mean, dtype: int32
```

```
# MAPE지표로 모형 평가
def MAPE(y_test, y_pred):
    return np.mean(np.abs((y_test - y_pred) / y_test)) * 100

print(f"MAPE: {MAPE(y_test, pred):.3f}")
```

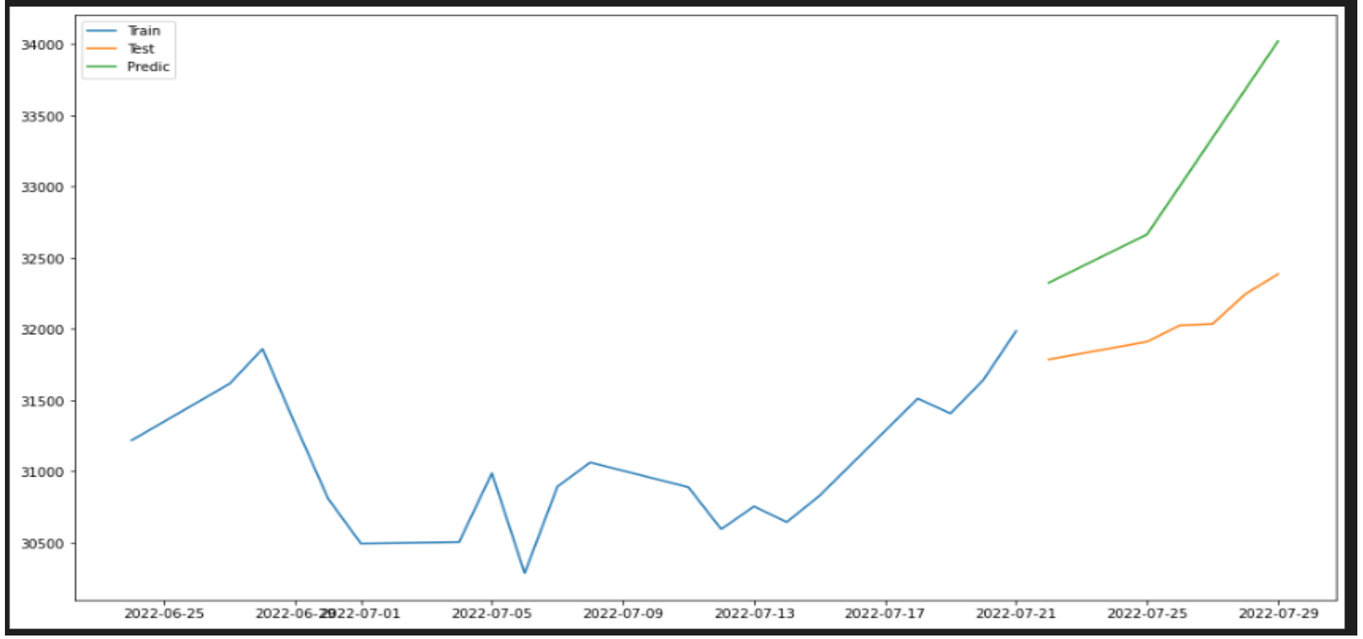
✓ 0.4s

```
MAPE: 3.451
```

```
plt.figure(figsize=(16,9))
plt.plot(y_train, label='Train')
plt.plot(y_test, label='Test')
plt.plot(pred, label='Predic')
plt.legend()
```

✓ 0.2s

```
<matplotlib.legend.Legend at 0x1901bddf220>
```



ARIMA 최종 25일~ 29일 주가 예측

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scale_cols = ['Open','High','Low','Close','Volume']
kd_scaled = scaler.fit_transform(kodex_200[scale_cols])
kd_scaled = pd.DataFrame(kd_scaled)
kd_scaled.columns = scale_cols

a = kd_scaled
a.head()
```

✓ 0.6s

	Open	High	Low	Close	Volume
0	0.162181	0.458594	0.280724	0.548501	1.000000
1	0.671388	0.819763	0.625485	0.783069	0.674084
2	0.815864	0.819763	0.835705	0.926514	0.357366
3	0.696176	0.628392	0.709573	0.613169	0.258688
4	0.501416	0.368128	0.390039	0.308054	0.564425

```
b = kodex_200.reset_index(level=0)
c = b['Date']
d = pd.concat([c,a], axis=1,).reindex(a.index)
d.head()
```

✓ 0.4s

	Date	Open	High	Low	Close	Volume
0	2022-06-24	0.162181	0.458594	0.280724	0.548501	1.000000
1	2022-06-27	0.671388	0.819763	0.625485	0.783069	0.674084
2	2022-06-28	0.815864	0.819763	0.835705	0.926514	0.357366
3	2022-06-29	0.696176	0.628392	0.709573	0.613169	0.258688
4	2022-06-30	0.501416	0.368128	0.390039	0.308054	0.564425

데이터 정규화

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
ko_1 = d[['Date','Close']]
ko_1 = ko_1.sort_values(by='Date')
ko_1['Date'] = pd.to_datetime(ko_1['Date'])
ko_1.index = ko_1['Date']
ko_1.set_index('Date', inplace=True)
ko_1.head()
```

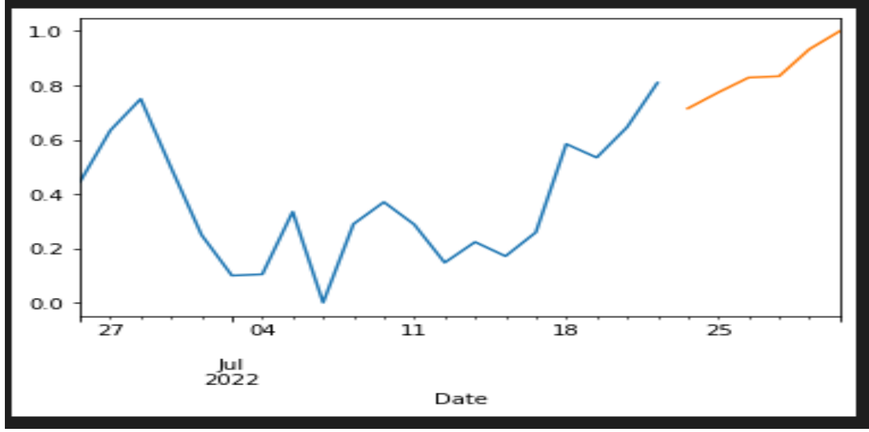
✓ 0.1s

Date	Close
2022-06-24	0.548501
2022-06-27	0.783069
2022-06-28	0.926514
2022-06-29	0.613169
2022-06-30	0.308054

```
y_train = ko_1['Close'][:int(0.8*len(kodex_200))]
y_test = ko_1['Close'][int(0.8*len(kodex_200)):]
y_train.plot()
y_test.plot()
```

✓ 0.2s

```
<AxesSubplot:xlabel='Date'>
```



종가 데이터 분리 후 시각화

3

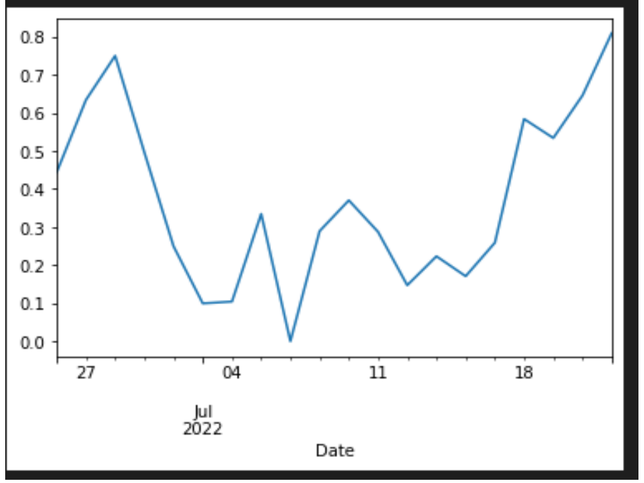
ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

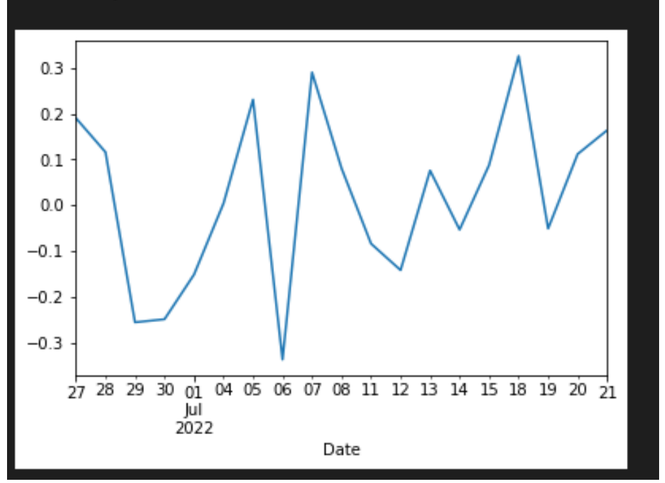
```
y_train.plot()
```
✓ 0.1s

```
<AxesSubplot:xlabel='Date'>
```



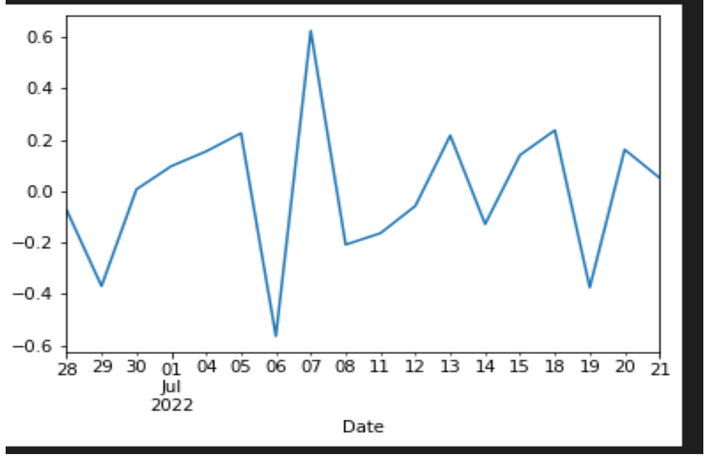

```
diff_1 = y_train.diff().dropna()
diff_1.plot()
```
✓ 0.1s

```
<AxesSubplot:xlabel='Date'>
```




```
diff_2 = diff_1.diff().dropna()
diff_2.plot()
```
✓ 0.1s

```
<AxesSubplot:xlabel='Date'>
```



차분을 통해 비정상시계열데이터 정상성으로 변환

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
from statsmodels.tsa.stattools import adfuller
dataX = y_train.values
fuller = adfuller(dataX)
print('ADF Statistic: %f' % fuller[0])
print('p-value: %f' % fuller[1])
print('Critical Values:')
for key, value in fuller[4].items():
    print('\t%s: %.3f' % (key, value))
```

✓ 0.5s

```
ADF Statistic: 1.460105
p-value: 0.997371
Critical Values:
        1%: -4.223
        5%: -3.189
        10%: -2.730
```

```
from statsmodels.tsa.stattools import adfuller
dataX = diff_1.values
fuller = adfuller(dataX)
print('ADF Statistic: %f' % fuller[0])
print('p-value: %f' % fuller[1])
print('Critical Values:')
for key, value in fuller[4].items():
    print('\t%s: %.3f' % (key, value))
```

✓ 0.4s

```
ADF Statistic: -4.364598
p-value: 0.000343
Critical Values:
        1%: -3.859
        5%: -3.042
        10%: -2.661
```

```
from statsmodels.tsa.stattools import adfuller
dataX = diff_2.values
fuller = adfuller(dataX)
print('ADF Statistic: %f' % fuller[0])
print('p-value: %f' % fuller[1])
print('Critical Values:')
for key, value in fuller[4].items():
    print('\t%s: %.3f' % (key, value))
```

✓ 0.7s

```
ADF Statistic: -9.081100
p-value: 0.000000
Critical Values:
        1%: -4.332
        5%: -3.233
        10%: -2.749
```

차분을 통해 비정상시계열데이터 정상성으로 변환

3

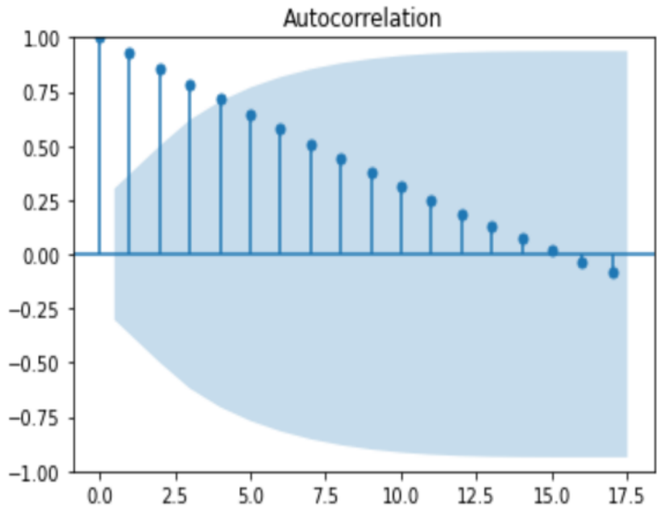
ARIMA

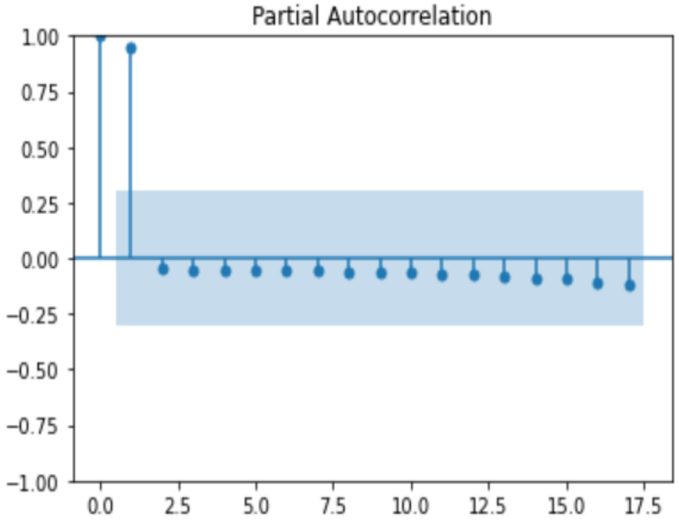
시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
import numpy as np
lags=np.arange(len(diff_2)*2)
plot_acf(lags)
plot_pacf(lags)
plt.show()
```

✓ 0.2s





ACF, PACF로 AR, MA 구하기

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm

model = sm.tsa.arima.ARIMA(y_train, order=(1,2,0))
model_fit = model.fit()
print(model_fit.summary())
```

SARIMAX Results

Dep. Variable:	Close	No. Observations:	20
Model:	ARIMA(1, 2, 0)	Log Likelihood	0.450
Date:	Mon, 01 Aug 2022	AIC	3.101
Time:	09:37:02	BIC	4.882
Sample:	06-24-2022	HQIC	3.346
	- 07-21-2022		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4762	0.230	-2.071	0.038	-0.927	-0.025
sigma2	0.0549	0.027	2.030	0.042	0.002	0.108

Ljung-Box (L1) (Q):	0.73	Jarque-Bera (JB):	0.89
Prob(Q):	0.39	Prob(JB):	0.64
Heteroskedasticity (H):	0.55	Skew:	-0.34
Prob(H) (two-sided):	0.48	Kurtosis:	2.14

ARIMA모델 학습

3

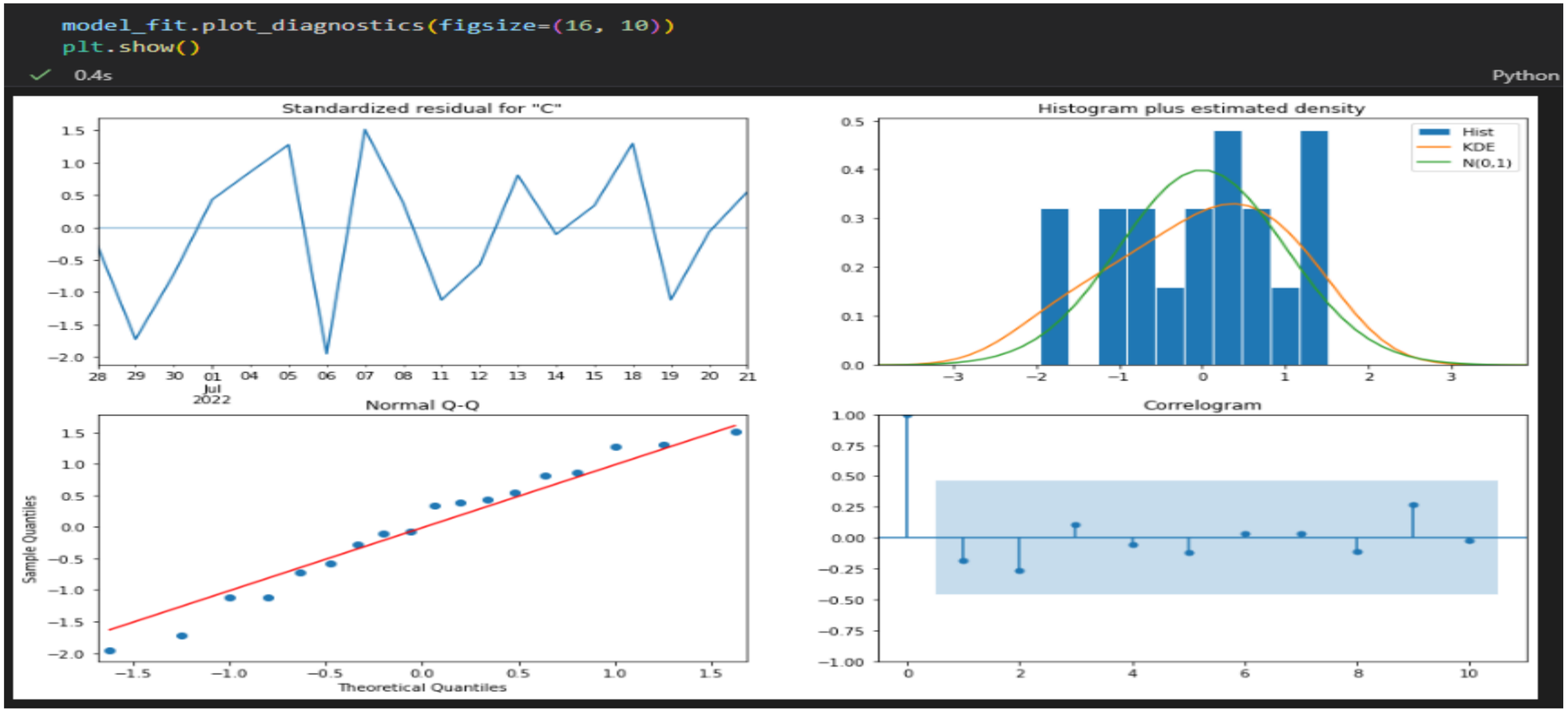
ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
model_fit.plot_diagnostics(figsize=(16, 10))
plt.show()
```

0.4s Python



학습한 모델 시각화

3

ARIMA

시계열의 변동형태를 파악하고 이를 통해 예측이 가능

데이터 로드 및 분포 확인

```
pred = model_fit.forecast(steps=6)
print(pred)
```

✓ 0.3s

```
2022-07-22    0.948639
2022-07-25    1.099430
2022-07-26    1.244617
2022-07-27    1.392472
2022-07-28    1.539057
2022-07-29    1.686247
Freq: B, Name: predicted_mean, dtype: float64
```

```
# MAPE지표로 모형 평가
def MAPE(y_test, y_pred):
    return np.mean(np.abs((y_test - y_pred) / y_test)) * 100

print(f"MAPE: {MAPE(y_test, pred):.3f}")
```

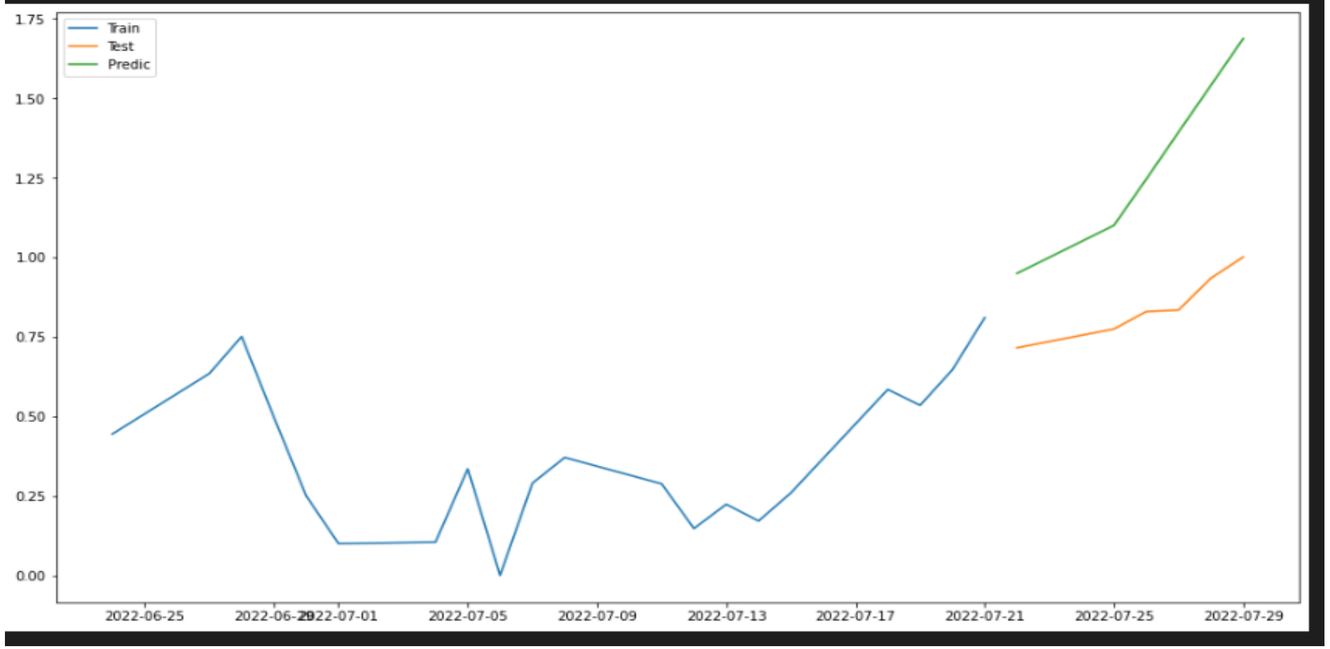
✓ 0.4s

```
MAPE: 54.259
```

```
plt.figure(figsize=(16,9))
plt.plot(y_train, label='Train')
plt.plot(y_test, label='Test')
plt.plot(pred, label='Predic')
plt.legend()
```

✓ 0.1s Python

```
<matplotlib.legend.Legend at 0x2151b12ba30>
```



ARIMA 최종 25일~ 29일 주가 예측

3.
성능평가 & 결론

성능평가

```
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head())
```

[60] ✓ 0.1s

```
          ds         yhat   yhat_lower   yhat_upper
0 2022-07-25  32105.243891  31789.780826  32414.907107
1 2022-07-26  32278.239942  31945.054712  32623.185543
2 2022-07-27  32161.854834  31820.835689  32479.925028
3 2022-07-28  32305.285009  31999.370476  32625.007598
4 2022-07-29  32310.638335  31969.977737  32646.036381
```

1. Prophet

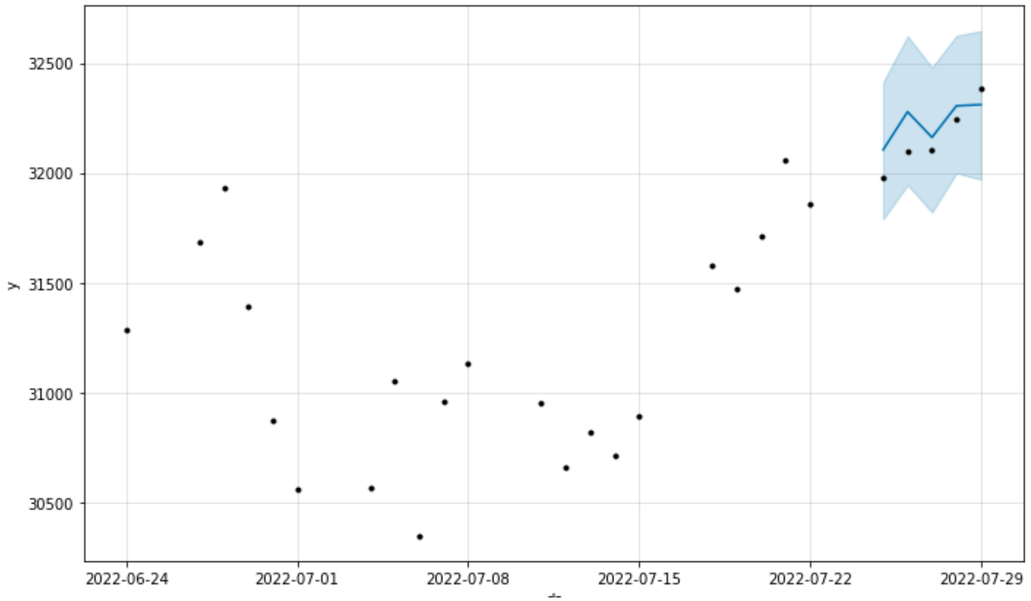
```
pred = model_fit.forecast(steps=6).astype(int)
print(pred)
```

✓ 0.6s

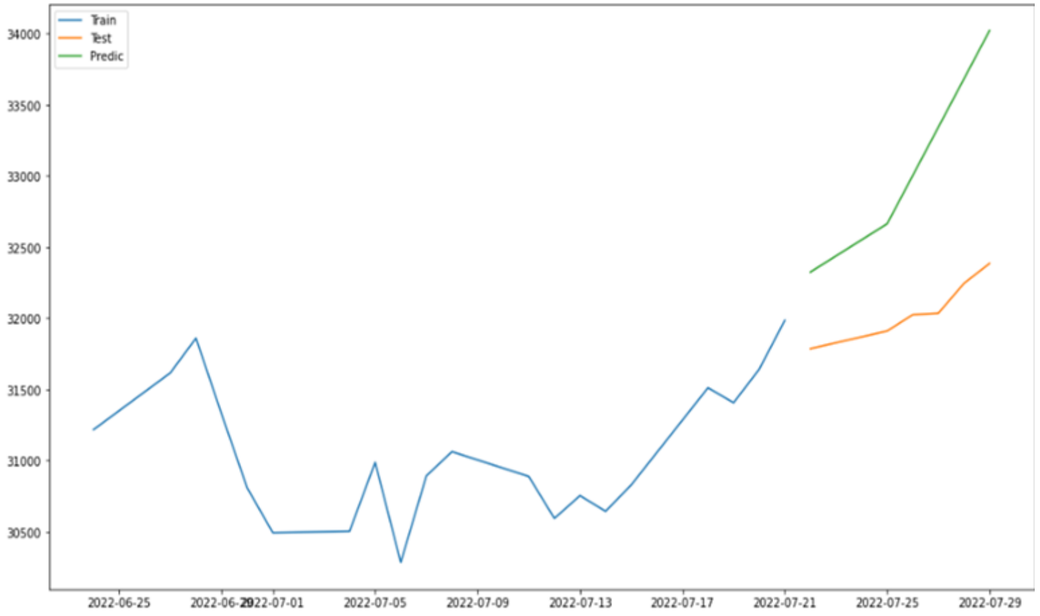
```
2022-07-22    32324
2022-07-25    32663
2022-07-26    33003
2022-07-27    33342
2022-07-28    33682
2022-07-29    34021
Freq: B, Name: predicted_mean, dtype: int32
```

3. Arima

성능평가



1. Prophet



3. ARIMA

성능평가

정확도 비교

Prophet

예측 추이는 비슷하게 맞춤

예상 최대폭과 최소폭이 너무 큼

일정 폭 간격으로 증가 예측성공

LSTM

예측값 추출은 실패

그러나 데이터의 정확도를 높이기 위한 2가지 방법 발견

1) Normalization

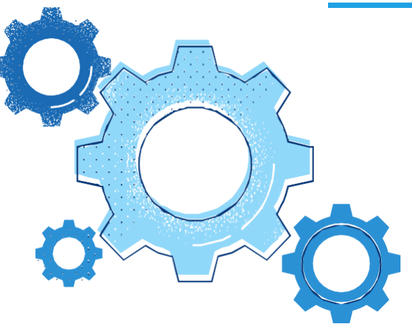
2) LSTM 신경망 크기

Arima

점점 우상향 할 것으로 예상성공

정규화과정 만들기 힘듬.

28~29일 상승폭이 실제 값보다 차이가 심함



결론

01

예측값 정확성 떨어짐

다양한 변수로 인한
정확도 부족

02

정규화 작업 실패

ARIMA모형 정규화
작업 예측값 생성실패

03

LSTM 모델이해 부족

RNN구조와 차이점
이해힘듬

04

데이터수 부족

한달 데이터로 5일
예측하기 힘듬

2

LSTM

순차열이 길어져도 과거의 데이터를 잃지 않고 학습

생각해 보고 싶은 부분

1)추적오차(순자산 가치(NAC)수익률 – 벤치마크 수익률

2)괴리율(ETF의 종가 – 순자산 가치)

두 변수와의 상관관계를 찾은 후 LSTM모형을 구축하면 더 정교해 지지 않을까?

참고논문 : 구성자산 특성이 ETF가격 효율성에 미치는 효과

감사합니다
Q&A