ARIMA모델을 활용한 YG 주가 예측

souljoin0229@gmail.com

kimseojeong

1) AR(Auto Regressive)모형?

전제 : 과거의 값은 현재의 값에 영향을 미친다

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t \qquad X_t = c + \left(\sum_{i=1}^{n} X_{t-1} * w_{t-1}\right) + e_t$$

- $Z_t = \Phi_1 Z_{t-1} + \Phi_2 Z_{t-2} + \ldots + \Phi_p Z_{t-p} + \alpha_t$

· Z_t : 현재 시점의 시계열 자료

· Z_{t-p} : p 시점 전의 시계열 자료

· Φ_p : p 시점이 현재에 어느 정도 영향을 주는지를 나타내는 모수

· α_t : 시계열 분석에서 오차항 의미, 백색잡음과정

· AR(1) : $Z_t = \Phi_1 Z_{t-1} + \alpha_t$

P(시점)?

-특정 시점에서의 시계열 값
-상수항
-오차항(잔차)

$$AR(1) : X_t = a X_{t-1} + e_t \qquad (-1 < a < 1)$$

현재 시점의 잔차
이전 시점의 정상 시계열
상수
현재 시점의 정상 시계열

현재 시점의 정상 시계열
= a * 이전 시점의 정상 시계열 + 현재의 잔차

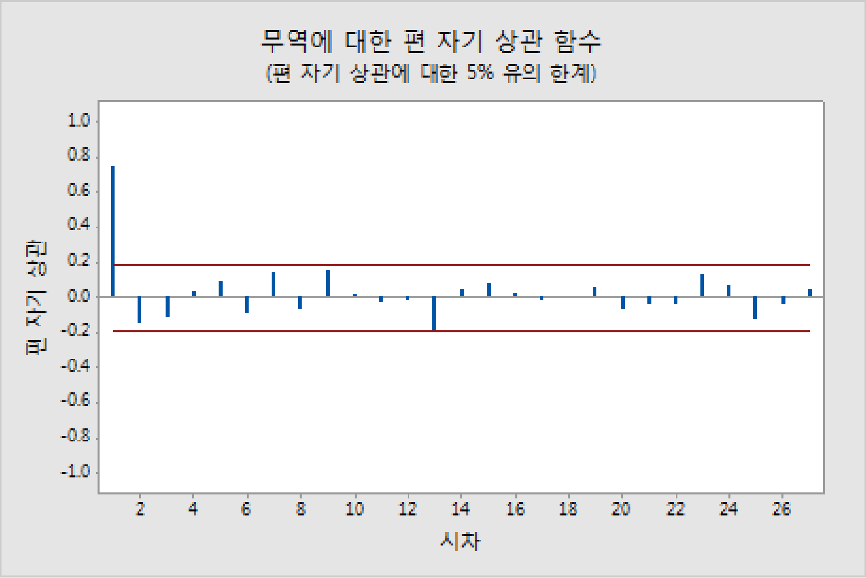
적절한 P(시점)는 어떻게 구하는가?

PACF그래프

PACF is a partial auto-correlation function.
The correlation between observations at two time spots given that we consider both observations are correlated to observations at other time spots

For example, today's stock price can be correlated to the day before yesterday, and yesterday can also be correlated to the day before yesterday. Then, PACF of yesterday is the real correlation between today and yesterday after taking out the influnce of the days before yesterday.

PACF에서 significant한 lag : 현재의 값에 유의미한 영향을 주는 과거 시점 도출가능



Lag=0 : 자기자신과의 상관성이므로 항상 1

Lag=1에서 자기상관도가 유의수준을 넘음,
이후값들은 유의수준 이하

P=1

2) MA(Moving Average)모형?

가설 : 데이터는 특정한 평균을 가지고 있으며, 이 상태에서 현재의 독립변수(y)는 이전 오차 항의 변동성에 의해 결정된다
-> 데이터의 평균도 오차항의 변동성에 의해 이동된다

이전 항에서의 오차(e(t-1)) 혹은 변동값을 이용하여 현재 항의 상태를 추론

뮤 : yt의 평균
q : 시점

$$y_t = \mu + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q}$$

이전 시점 t-1의 오차

$$y = \beta_0 + \beta_1 x + \epsilon$$

$$z_t = \phi_1 z_{t-1} + \phi_2 z_{t-2} + a_t$$

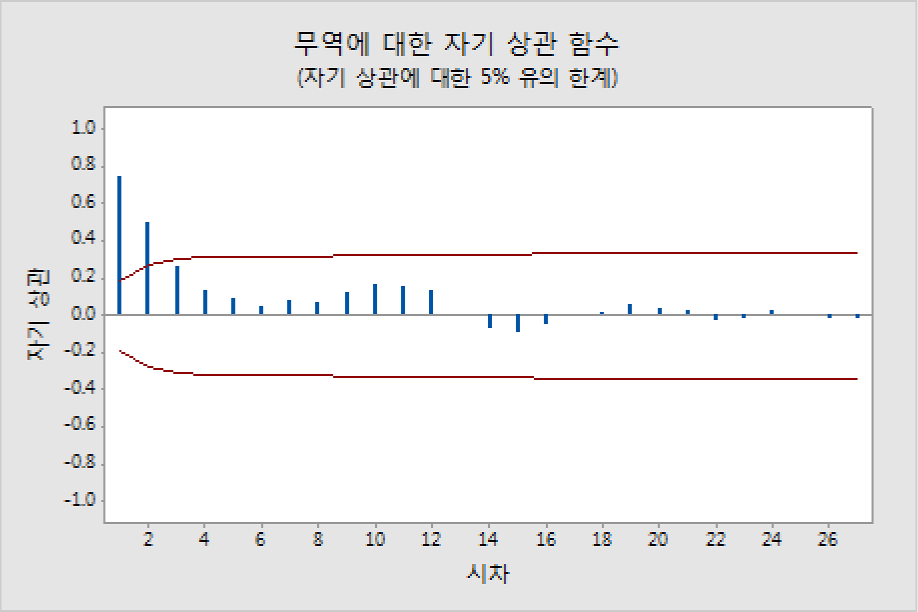
$$z_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2}$$

적절한 q(시점)는 어떻게 구하는가?

ACF그래프

ACF is an (complete) auto-correlation function which gives us values of auto-correlation of any series with its lagged values.
The correlation between the observation at the current time spot and the observations at previous time spots

현재 시점에 가장 영향을 주는 과거시점은 어디까지인지 도출



Lag=2에서 통계적으로 유의한 상관이 있음

q=2

3) ARIMA모형?

*ARMA모형

ARMA(1,1) : $X(t) = (X_{t-1} * w_{11}) + (e_{t-1} * w_{21}) + b + (e_t * u)$

(3) 자기회귀이동평균도

앞에서의 두 가지 도
될 수 있다는 것과 그
있는데, 경우에 따라
모두의 영향을 받는
합된 자기회귀이동평
$Y_t = \alpha_1 Y_{t-1} + \alpha_2 Y_{t-2}$
이다.

2) ARIMA (p, d, q) 모형의 형태

비정상 시계열 자료를 d차 차분하여, 정상성을 갖는 새로운 시계열 자료인 $\Delta^d y_t = \Delta y_t - \Delta y_{t-d} = (1-B)^d y_t$ 는 정상 확률과정 ARMA 모형으로 표현되는데, 이는 ARMA 모형과 차분 연산을 통합한 다음의 식으로 표현된다. 이를 ARIMA 모형(자기회귀 누적 이동평균 모형, 또는 자기회귀 결합 이동평균 모형 - Auto Regressive Integrated Moving Average Model)이라 한다.

$$\text{ARIMA } (p, d, q): \; a_p(B)(1-B)^d y_t = \theta_0 + \theta_q(B) f_t$$

수식에서 f_t 는 평균 0, 분산 σ_f^2인 White Noise이며, θ_0 는 추세모수(Trend Parameter)로서 zero이다. 추세모수가 제거된 1차 차분의 경우, 시계열 자료는 차분된 시계열 자료 z_t의 누적(Integrated) 형태로 나타난다.

$$y_t = (1-B)^{-1} \Delta y_t = \left(\frac{1}{1-B}\right) z_t = \sum_{k=-\infty}^{t} z_t$$

다음은 ARIMA 모형의 다양한 차수별 수식을 요약한 것으로서, 일반적으로 p, d, q 의 차수는 (2)를 넘지 않는다. 특히, ARIMA (0,1,0)의 경우 이는 Random Walk Model로 불리며, ARIMA 모형의 특수한 경우이다.

ARIMA (0,1,0) : $(1-B)y_t = f_t$

$y_t = y_{t-1} + f_t$

ARIMA (0,1,1) : $(1-B)y_t = (1-\theta B)f_t$

$y_t = y_{t-1} + f_t - \theta f_{t-1}$

ARIMA (1,1,0) : $(1-aB)(1-B)y_t = f_t$

$y_t = y_{t-1} + a y_{t-1} - a y_{t-2} + f_t$

ARIMA (1,1,1) : $(1-aB)(1-B)y_t = (1-\theta B)f_t$

$y_t = y_{t-1} + a y_{t-1} - a y_{t-2} + f_t - \theta f_{t-1}$

ARIMA (1,1,2) : $(1-aB)(1-B)y_t = (1-\theta_1 B - \theta_2 B^2)f_t$

$y_t = y_{t-1} + a y_{t-1} - a y_{t-2} + f_t - \theta_1 f_{t-1} - \theta_2 f_{t-2}$

ARIMA (2,1,1) : $(1-a_1 B - a_2 B^2)y_t(1-B)y_t = (1-\theta B)f_t$

$y_t = y_{t-1} + a_1 y_{t-1} - a_1 y_{t-2} + a_2 y_{t-2} - a_2 y_{t-3} + f_t - \theta f_{t-1}$

ARIMA (2,1,2) : $(1-a_1 B - a_2 B^2)y_t(1-B)y_t = (1-\theta_1 B - \theta_2 B^2)f_t$

$y_t = y_{t-1} + a_1 y_{t-1} - a_1 y_{t-2} + a_2 y_{t-2} - a_2 y_{t-3} + f_t - \theta_1 f_{t-1} - \theta_2 f_{t-2}$

3) ARIMA모형?

2) ARIMA (p,d,q) 모형의 형태

비정상 시계열 자료를 d차 차분하여, 정상성을 갖는 새로운 시계열 자료인 $\Delta^d y_t = \Delta y_t - \Delta y_{t-d} = (1-B)^d y_t$ 는 정상 확률과정 ARMA 모형으로 표현되는데, 이는 ARMA 모형과 차분 연산을 통합한 다음의 식으로 표현된다. 이를 ARIMA 모형(자기회귀 누적 이동평균 모형, 또는 자기회귀 결합 이동평균 모형 – Auto Regressive Integrated Moving Average Model)이라 한다.

$$\text{ARIMA } (p,d,q): \ a_p(B)(1-B)^d y_t = \theta_0 + \theta_q(B) f_t$$

수식에서 f_t 는 평균 0, 분산 σ_f^2인 White Noise이며, θ_0 는 추세모수(Trend Parameter)로서 zero이다. 추세모수가 제거된 1차 차분의 경우, 시계열 자료는 차분된 시계열 자료 z_t의 누적(Integrated) 형태로 나타난다.

$$y_t = (1-B)^{-1}\Delta y_t = (\frac{1}{1-B}) z_t = \sum_{k=-\infty}^{t} z_t$$

다음은 ARIMA 모형의 다양한 차수별 수식을 요약한 것으로서, 일반적으로 p, d, q의 차수는 (2)를 넘지 않는다. 특히, ARIMA (0,1,0)의 경우 이는 Random Walk Model로 불리며, ARIMA 모형의 특수한 경우이다.

$$\text{ARIMA } (0,1,0): \ (1-B)y_t = f_t$$
$$y_t = y_{t-1} + f_t$$

$$\text{ARIMA } (0,1,1): \ (1-B)y_t = (1-\theta B)f_t$$
$$y_t = y_{t-1} + f_t - \theta f_{t-1}$$

$$\text{ARIMA } (1,1,0): \ (1-aB)(1-B)y_t = f_t$$
$$y_t = y_{t-1} + a y_{t-1} - a y_{t-2} + f_t$$

$$\text{ARIMA } (1,1,1): \ (1-aB)(1-B)y_t = (1-\theta B)f_t$$
$$y_t = y_{t-1} + a y_{t-1} - a y_{t-2} + f_t - \theta f_{t-1}$$

$$\text{ARIMA } (1,1,2): \ (1-aB)(1-B)y_t = (1-\theta_1 B - \theta_2 B^2)f_t$$
$$y_t = y_{t-1} + a y_{t-1} - a y_{t-2} + f_t - \theta_1 f_{t-1} - \theta_2 f_{t-2}$$

$$\text{ARIMA } (2,1,1): \ (1-a_1 B - a_2 B^2)y_t(1-B)y_t = (1-\theta B)f_t$$
$$y_t = y_{t-1} + a_1 y_{t-1} - a_1 y_{t-2} + a_2 y_{t-2} - a_2 y_{t-3} + f_t - \theta f_{t-1}$$

$$\text{ARIMA } (2,1,2): \ (1-a_1 B - a_2 B^2)y_t(1-B)y_t = (1-\theta_1 B - \theta_2 B^2)f_t$$
$$y_t = y_{t-1} + a_1 y_{t-1} - a_1 y_{t-2} + a_2 y_{t-2} - a_2 y_{t-3} + f_t - \theta_1 f_{t-1} - \theta_2 f_{t-2}$$

정상성 확인방법 :

- Original 그래프 형태 확인
- Augmented Dickey-fuller Test
- Box-Cox() (only can be used for positive data values) OR CUBE-Root (세제곱근)
- ACF, PACF그래프

But **box-cox** transformation can be used only for strictly positive target **values**. If you have **negative values** in your target (dependent) variable, the **box-cox** and log transformation cannot be used.
Cube root can be used to transform **negative**, zero and positive data **values**

Augmented Dickey-fuller Test

The Augmented Dickey-Fuller test is a type of statistical test called a unit root test.

The intuition behind a unit root test is that it determines how strongly a time series is defined by a trend.
There are a number of unit root tests and the Augmented Dickey-Fuller may be one of the more widely used. It uses an autoregressive model and optimizes an information criterion across multiple different lag values.

- The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary (has some time-dependent structure).
- The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary.

- **Null Hypothesis (H0)**: If failed to be rejected, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure.
- **Alternate Hypothesis (H1)**: The null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. It does not have time-dependent structure.

```
ADF Statistic: -4.808291
p-value: 0.000052
Critical Values:
5%: -2.870
1%: -3.449
10%: -2.571
```

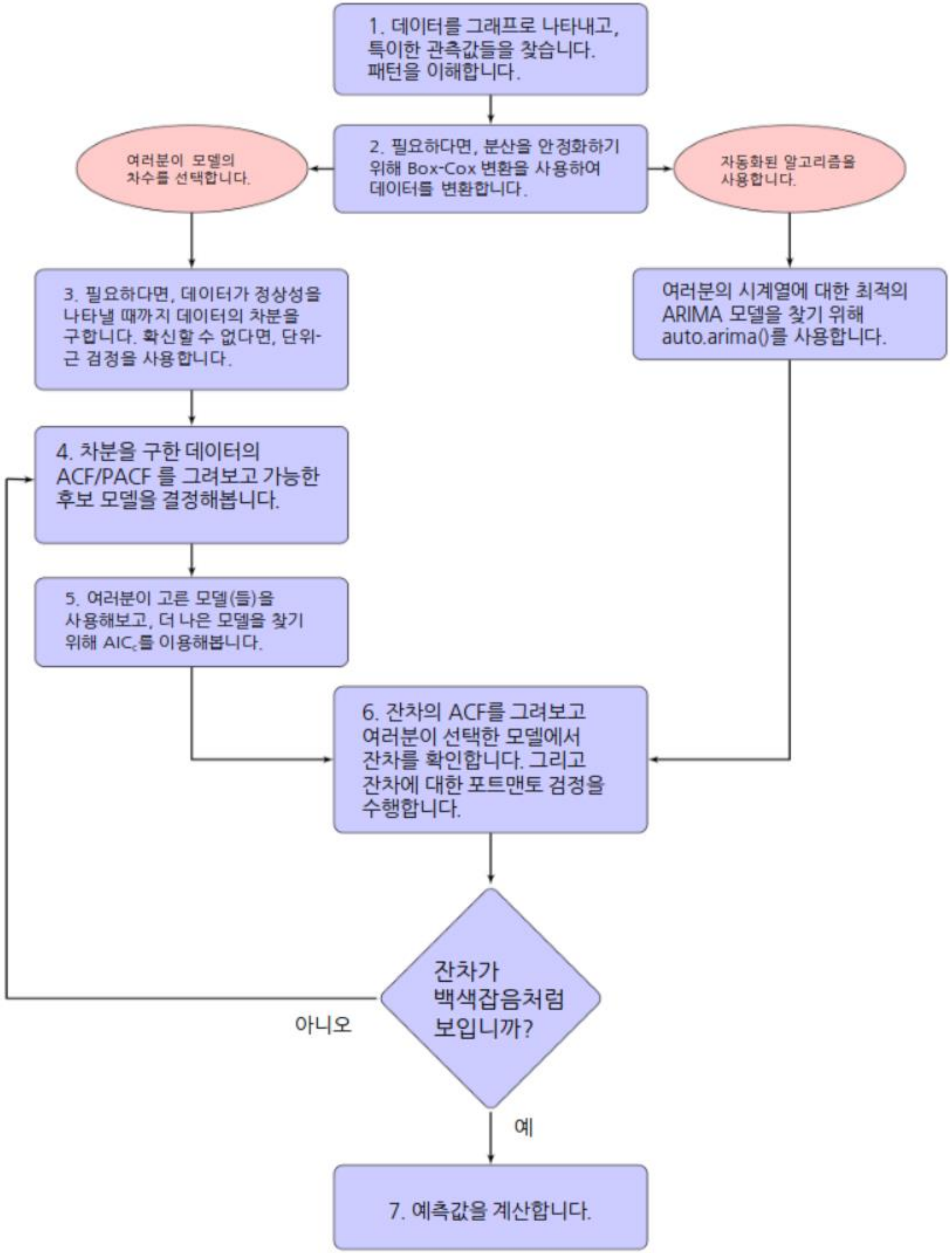
- **p-value > 0.05**: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.
- **p-value <= 0.05**: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

Running the example prints the test statistic value of -4.
The more negative this statistic, the more likely we are to reject the null hypothesis (we have a stationary dataset).

As part of the output, we get a look-up table to help determine the ADF statistic. We can see that our statistic value of -4 is less than the value of -3.449 at 1%.
This suggests that we can reject the null hypothesis with a significance level of less than 1% (i.e. a low probability that the result is a statistical fluke).

Rejecting the null hypothesis means that the process has no unit root, and in turn that the time series is stationary or does not have time-dependent structure.

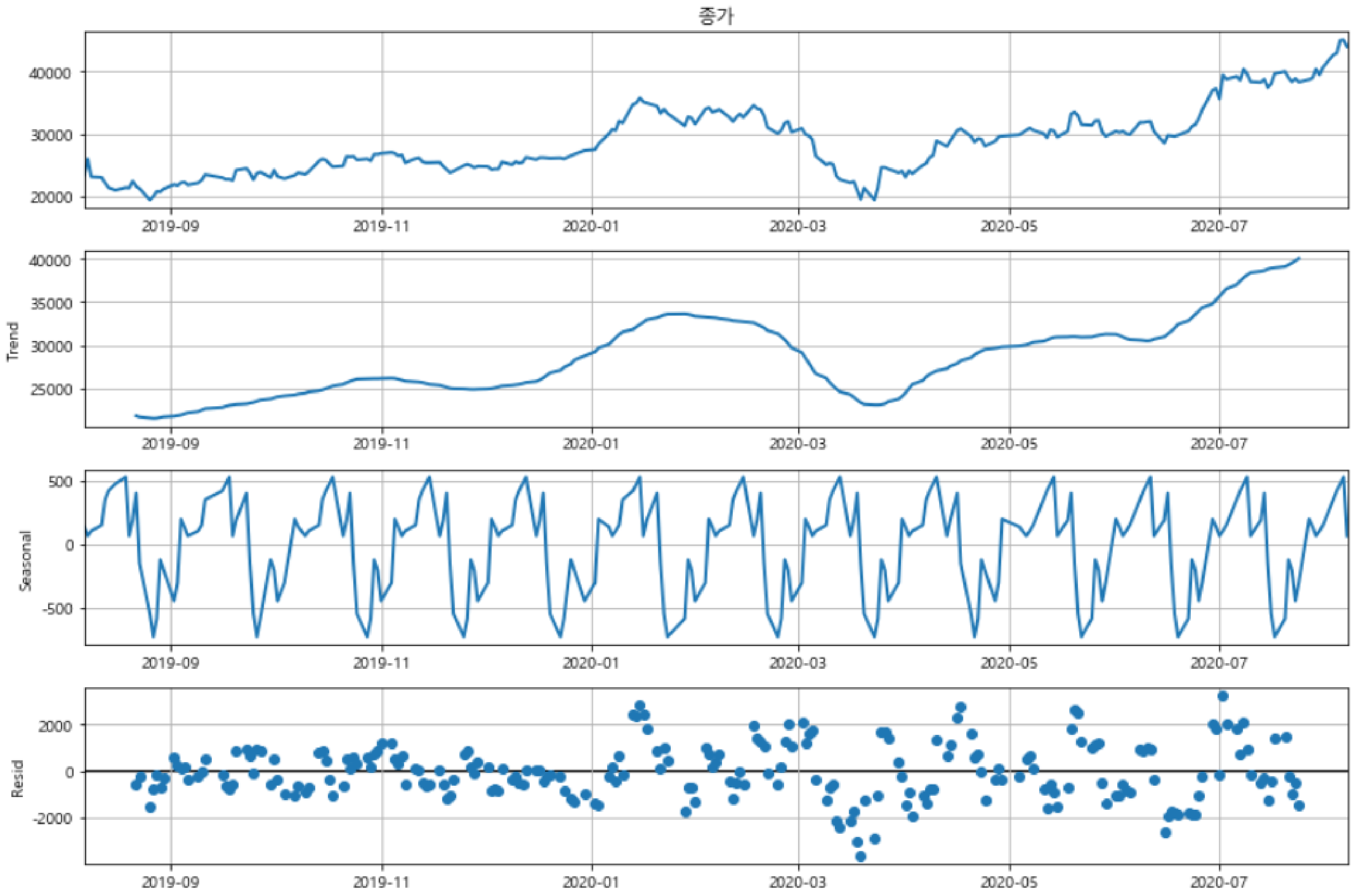
데이터로 ARIMA model 만들기


1. 데이터를 그래프로 나타내고,
특이한 관측값들을 찾습니다.
패턴을 이해합니다.
2. 필요하다면, 분산을 안정화하기
위해 Box-Cox 변환을 사용하여
데이터를 변환합니다.
여러분이 모델의
차수를 선택합니다.
자동화된 알고리즘을
사용합니다.
3. 필요하다면, 데이터가 정상성을
나타낼 때까지 데이터의 차분을
구합니다. 확신할 수 없다면, 단위-
근 검정을 사용합니다.
여러분의 시계열에 대한 최적의
ARIMA 모델을 찾기 위해
auto.arima()를 사용합니다.
4. 차분을 구한 데이터의
ACF/PACF 를 그려보고 가능한
후보 모델을 결정해봅니다.
5. 여러분이 고른 모델(들)을
사용해보고, 더 나은 모델을 찾기
위해 AIC_c를 이용해봅니다.
6. 잔차의 ACF를 그려보고
여러분이 선택한 모델에서
잔차를 확인합니다. 그리고
잔차에 대한 포트맨토 검정을
수행합니다.
잔차가
백색잡음처럼
보입니까?
아니오
예
7. 예측값을 계산합니다.

1. 패턴이해

- 시계열 분해 (Additive model)

```
result2_2 = seasonal_decompose(df_close, model='additive',period=20) # data에 0이 존재하지 않기때문에, multiplicative사용가능
result2_2.plot()
plt.show()
```

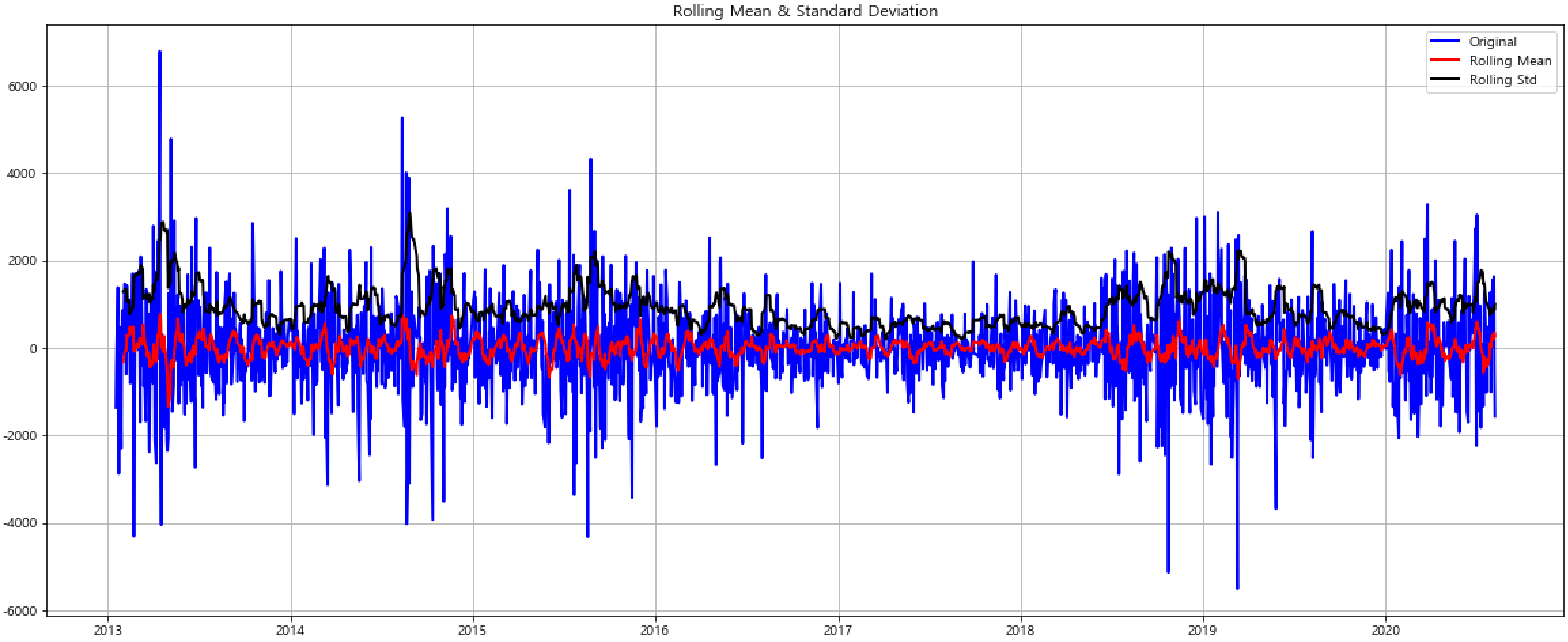


Data :

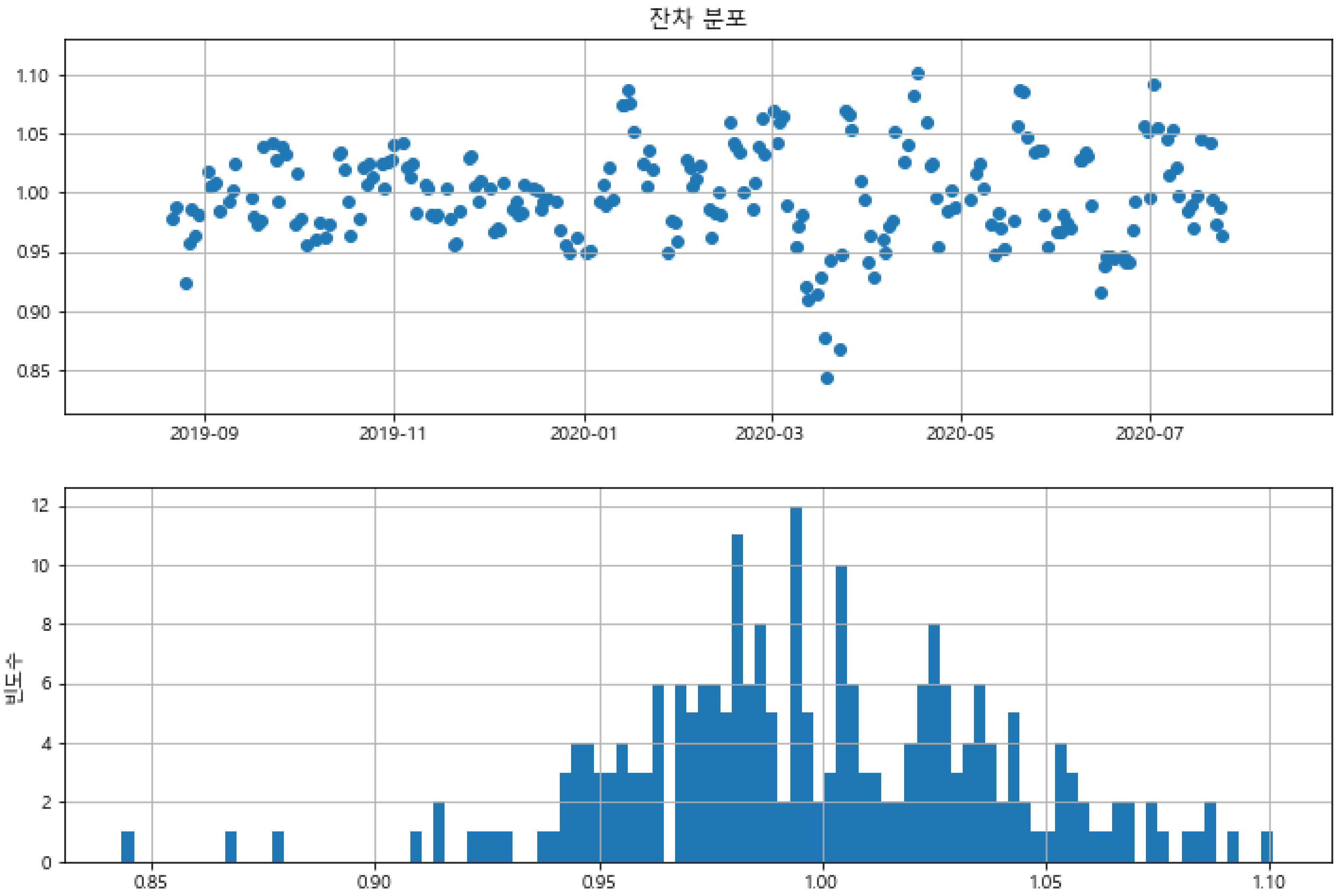
Train(2019-07-01 ~ 2020-07-01)

Test(2020-07-02 ~ 2020-08-31)

- 1차 차분값의 이동평균과 이동 표준편차



- Residual 분포



잔차 분포
1.10
1.05
1.00
0.95
0.90
0.85
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
빈도수
12
10
8
6
4
2
0
0.85
0.90
0.95
1.00
1.05
1.10

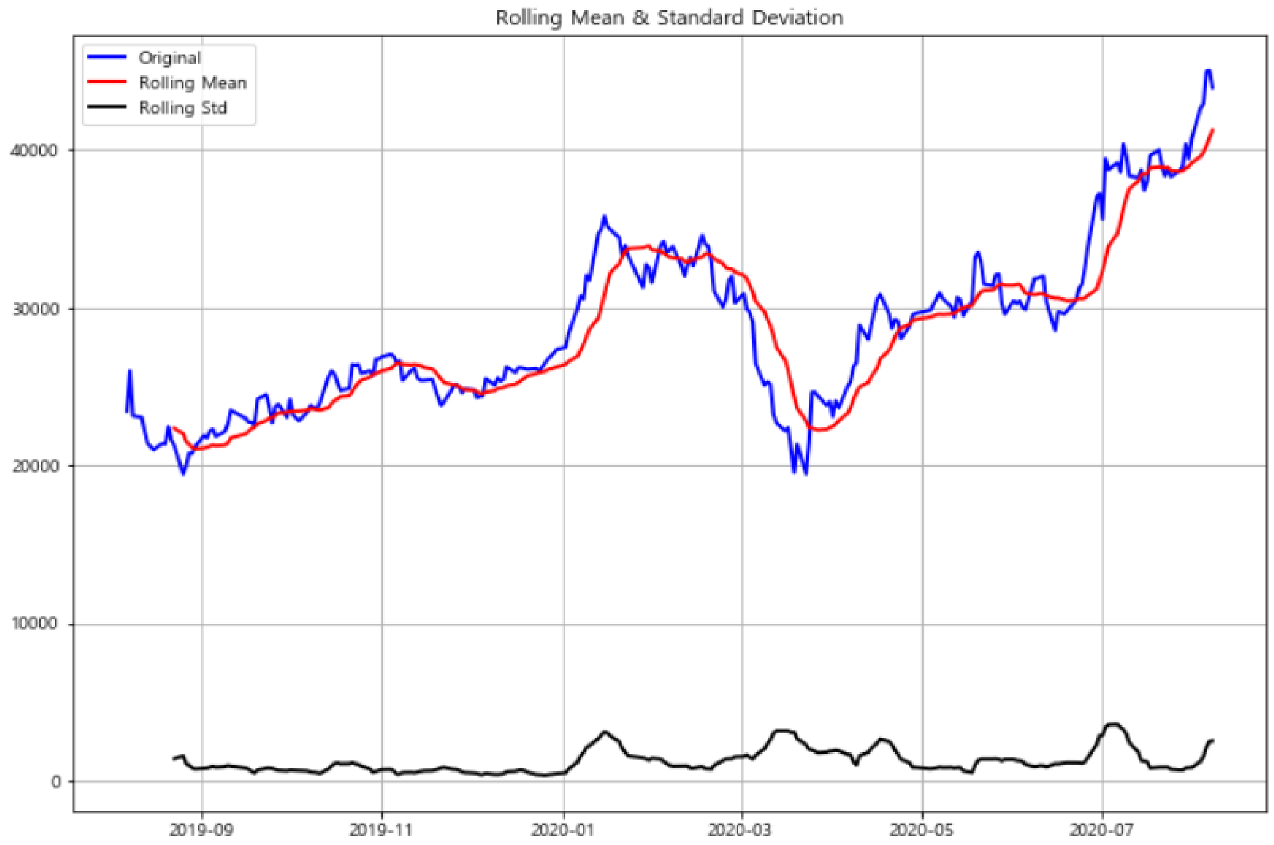
정상성 확인 방법 : Augmendted Dickey-Fuller, ACF, PACF

<Augmented Dickey-Fuller>

```python
# 종가의 10일 이동평균과 이동 표준편차 및 Dickey-Fuller Test 보기

get_stationarity(df_close) # p-value : 0.90713.. > 0.05
```

종가




```
ADF Statistic: -0.26956514335786763
p-value: 0.9297055794931203
Critical Values:
	1%: -3.4569962781990573
	5%: -2.8732659015936024
	10%: -2.573018897632674
```

> 0.05 (유의수준 5%)

정상성 확인 방법 : Augmendted Dickey-Fuller, ACF, PACF

<Augmented Dickey-Fuller>

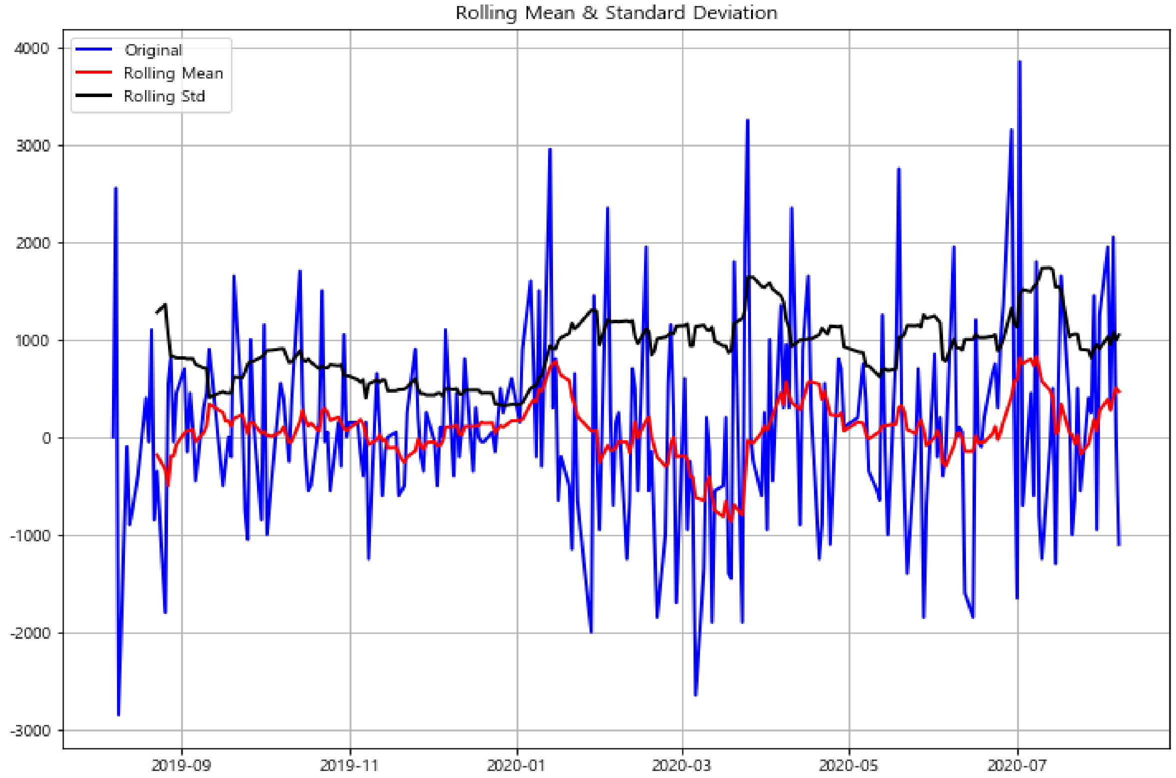
종가_1차차분

차분으로인한 음수발생,
boxcox를 사용x

```
# 종가-1차차분의 이동평균과 이동 표준편차 및 Dickey-Fuller Test 보기

get_stationarity(df_close_diff) # p-value : 0.000000008.. < 0.05

```

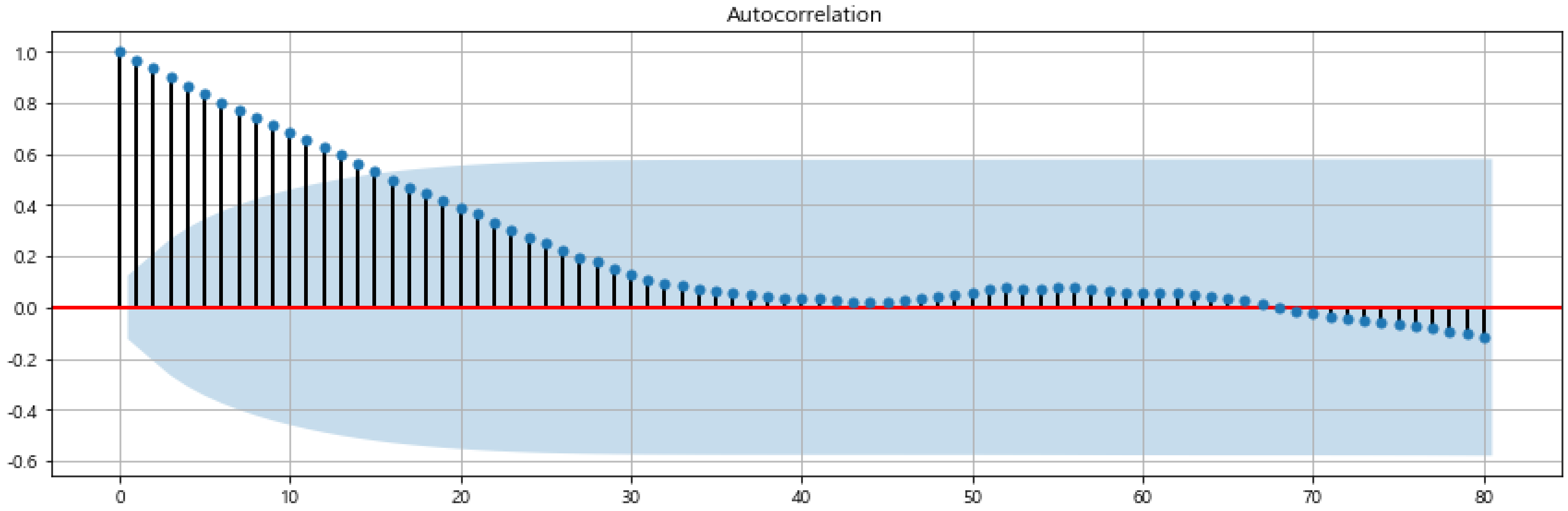



```
ADF Statistic: -16.125383257295088
p-value: 4.848263906617066e-29
Critical Values:
	1%: -3.4569962781990573
	5%: -2.8732659015936024
	10%: -2.573018897632674
```

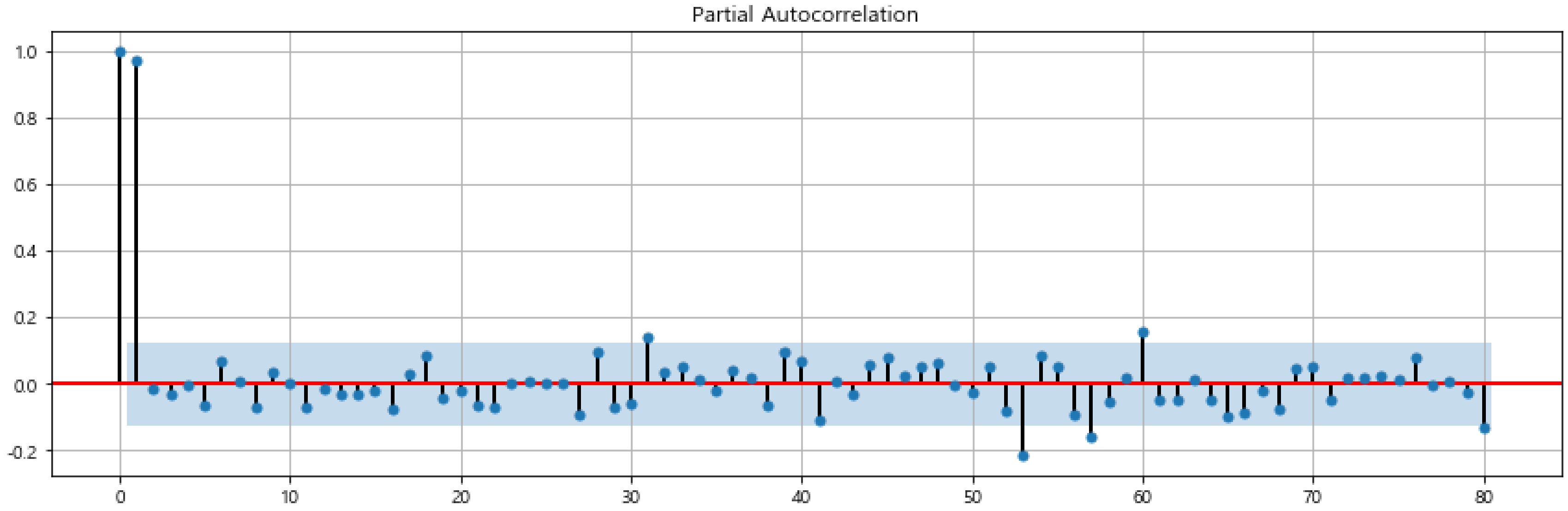
< 0.05 (유의수준 5%)

<종가>

ACF



PACF

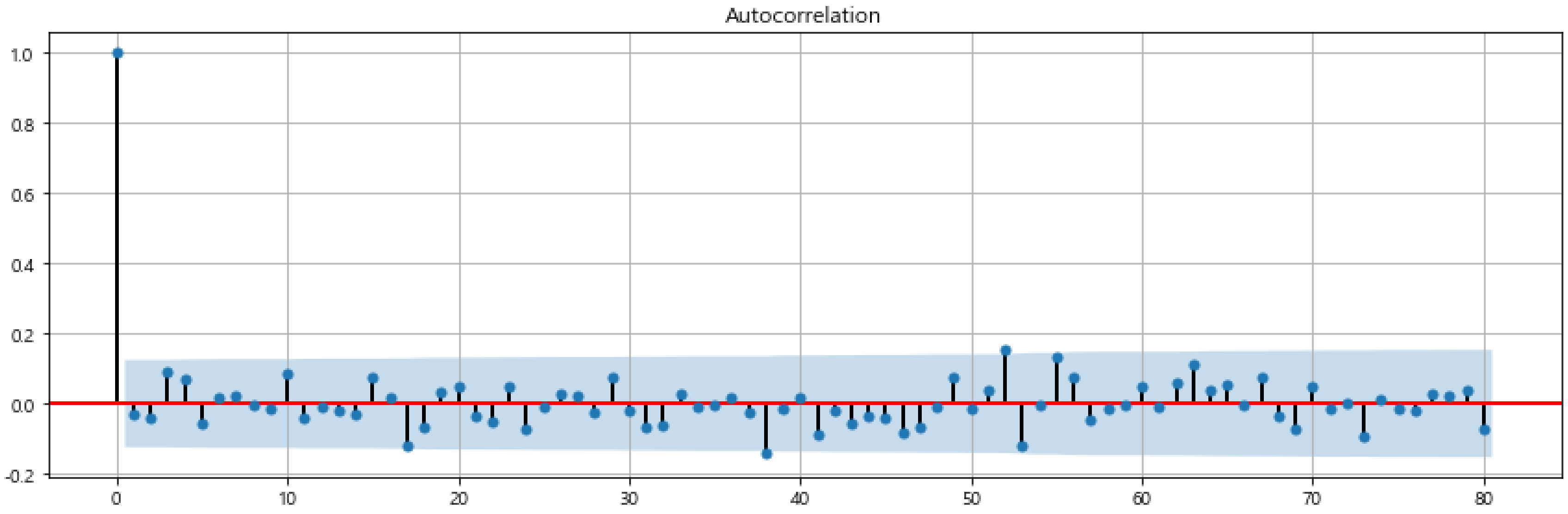


ACF그래프 : 점점 감소하는 모양
PACF그래프 : 1에서 상관성 ↑, lab가 커질 수록, 감쇠모양이아니라 불규칙하게 변동

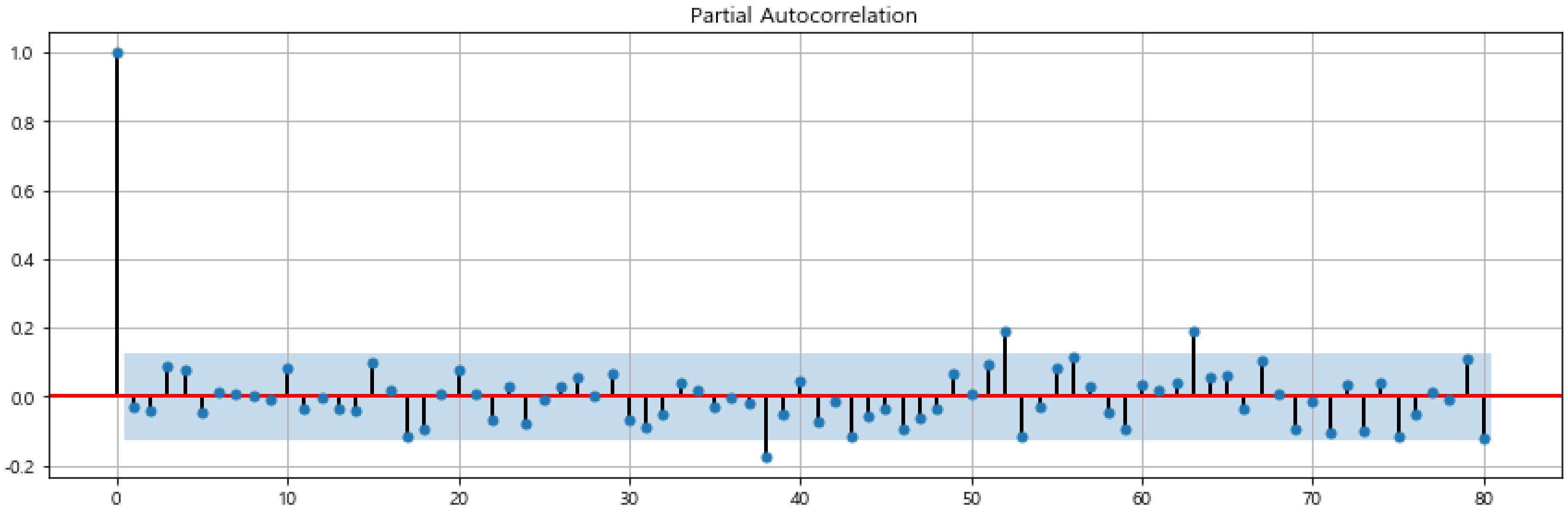
=> Non-Stationary

<종가_1차차분>

ACF :



PACF :



=> Stationary **ARIMA(0,1,0)**

ARIMA모델

- SARIMA((P,D,Q), (P,D,Q)m)

(P,D,Q) : Non Seasonal part of the Model

(P,D,Q)m : Seasonal part of the Model   m= number of observations per year.

The seasonal part of the model consists of terms that are similar to the non-seasonal components of the model, but involve backshifts of the seasonal period

Ex) ARIMA(1,1,1)(1,1,1)4 : quarterly data (m=4)

(P,D,Q)의 **P,Q**와 (P,D,Q)m 의 **P,Q** : Auto_arima 반복을 통해 AIC,BIC값이 가장 적게 나온 parameter로 결정

(P,D,Q)의 **D**와 (P,D,Q)m 의 **D** : 차분값의 Dickey-fullertest를 통해 유의값(0.05)이하 였을때의 parameter로 결정

(P,D,Q)m의 m : 분기별 변동성 고려 (주가특성) -> 60(5*4*3)으로 결정

예측 및 MSE

Auto-Arima

```
from pmdarima import auto_arima
stepwise_model = auto_arima(df_close, start_p=1, start_q=1,
                            max_p=3, max_q=3, m=12,
                            start_P=0, seasonal=True,
                            d=1, D=1, trace=True,
                            error_action='ignore',
                            suppress_warnings=True,
                            stepwise=True)
```

```
Example of parameter combinations for Seasonal ARIMA ...
SARIMAX:  (0, 0, 1) x (0, 0, 1, 12)
SARIMAX:  (0, 0, 1) x (0, 0, 2, 12)
SARIMAX:  (0, 0, 2) x (0, 0, 3, 12)
SARIMAX:  (0, 0, 2) x (0, 1, 0, 12)
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=inf, Time=0.98 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=4093.328, Time=0.02 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=4045.160, Time=0.34 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=3994.311, Time=0.35 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=4094.546, Time=0.04 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=3993.772, Time=0.91 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=4045.090, Time=0.33 sec
 ARIMA(0,1,1)(2,1,1)[12]             : AIC=3992.668, Time=1.20 sec
 ARIMA(0,1,1)(2,1,0)[12]             : AIC=4012.065, Time=0.76 sec
 ARIMA(0,1,1)(2,1,2)[12]             : AIC=3994.551, Time=2.14 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=inf, Time=2.06 sec
 ARIMA(0,1,0)(2,1,1)[12]             : AIC=3991.754, Time=0.88 sec
 ARIMA(0,1,0)(1,1,1)[12]             : AIC=3992.565, Time=0.35 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=4011.225, Time=0.25 sec
 ARIMA(0,1,0)(2,1,2)[12]             : AIC=3993.709, Time=1.83 sec
 ARIMA(0,1,0)(1,1,0)[12]             : AIC=4043.395, Time=0.09 sec
 ARIMA(0,1,0)(1,1,2)[12]             : AIC=3992.070, Time=2.04 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=3992.915, Time=1.31 sec
 ARIMA(1,1,1)(2,1,1)[12]             : AIC=4016.296, Time=2.76 sec
 ARIMA(0,1,0)(2,1,1)[12] intercept   : AIC=3993.733, Time=1.85 sec

Best model:  ARIMA(0,1,0)(2,1,1)[12]
Total fit time: 20.499 seconds
```

AIC설명 :

The AIC value will allow us to compare how well a model fits the data and takes into account the complexity of a model, so models that have a better fit while using fewer features will receive a better (lower) AIC score than similar models that utilize more features.

Data_Fitting

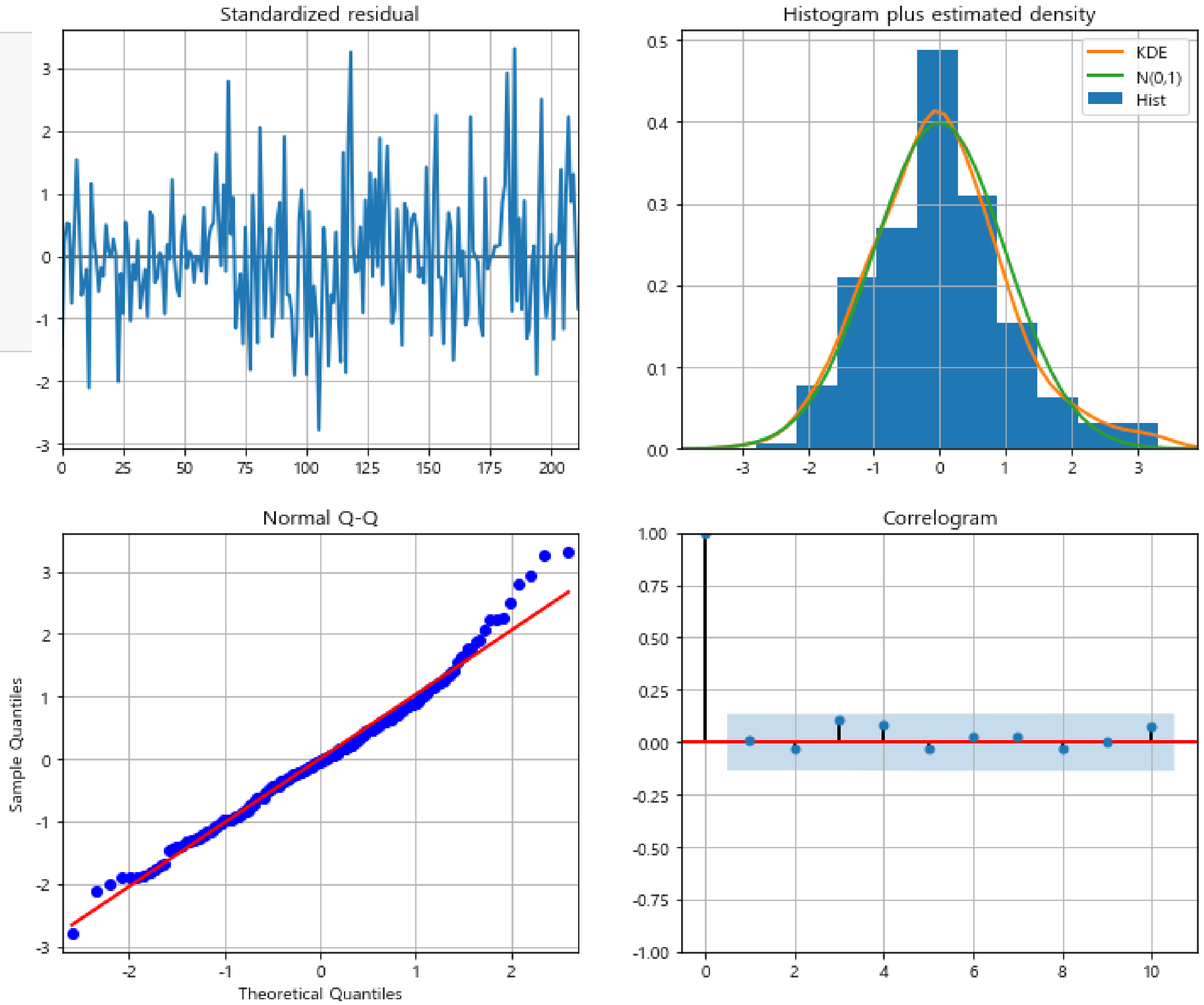
```python
mod = sm.tsa.statespace.SARIMAX(
    df_close['종가'],
    order=(0, 1, 0),
    seasonal_order=(2, 1, 1, 12),
    enforce_stationarity=False,
    enforce_invertibility=False
)

results = mod.fit()

print( results.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                  종가   No. Observations:               249
Model:             SARIMAX(0, 1, 0)x(2, 1, [1], 12)   Log Likelihood              -1784.340
Date:                              Tue, 08 Sep 2020   AIC                          3576.680
Time:                                      16:53:27   BIC                          3590.106
Sample:                                           0   HQIC                         3582.106
                                              - 249
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.S.L12      -0.1062      0.096     -1.111      0.266      -0.294       0.081
ar.S.L24      -0.1418      0.086     -1.653      0.098      -0.310       0.026
ma.S.L12      -0.7119      0.084     -8.511      0.000      -0.876      -0.548
sigma2      1.126e+06   9.43e+04     11.946      0.000    9.41e+05    1.31e+06
===================================================================================
Ljung-Box (Q):                       23.73   Jarque-Bera (JB):                10.84
Prob(Q):                              0.98   Prob(JB):                         0.00
Heteroskedasticity (H):               2.05   Skew:                             0.44
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.66
===================================================================================
```

Residual_EDA



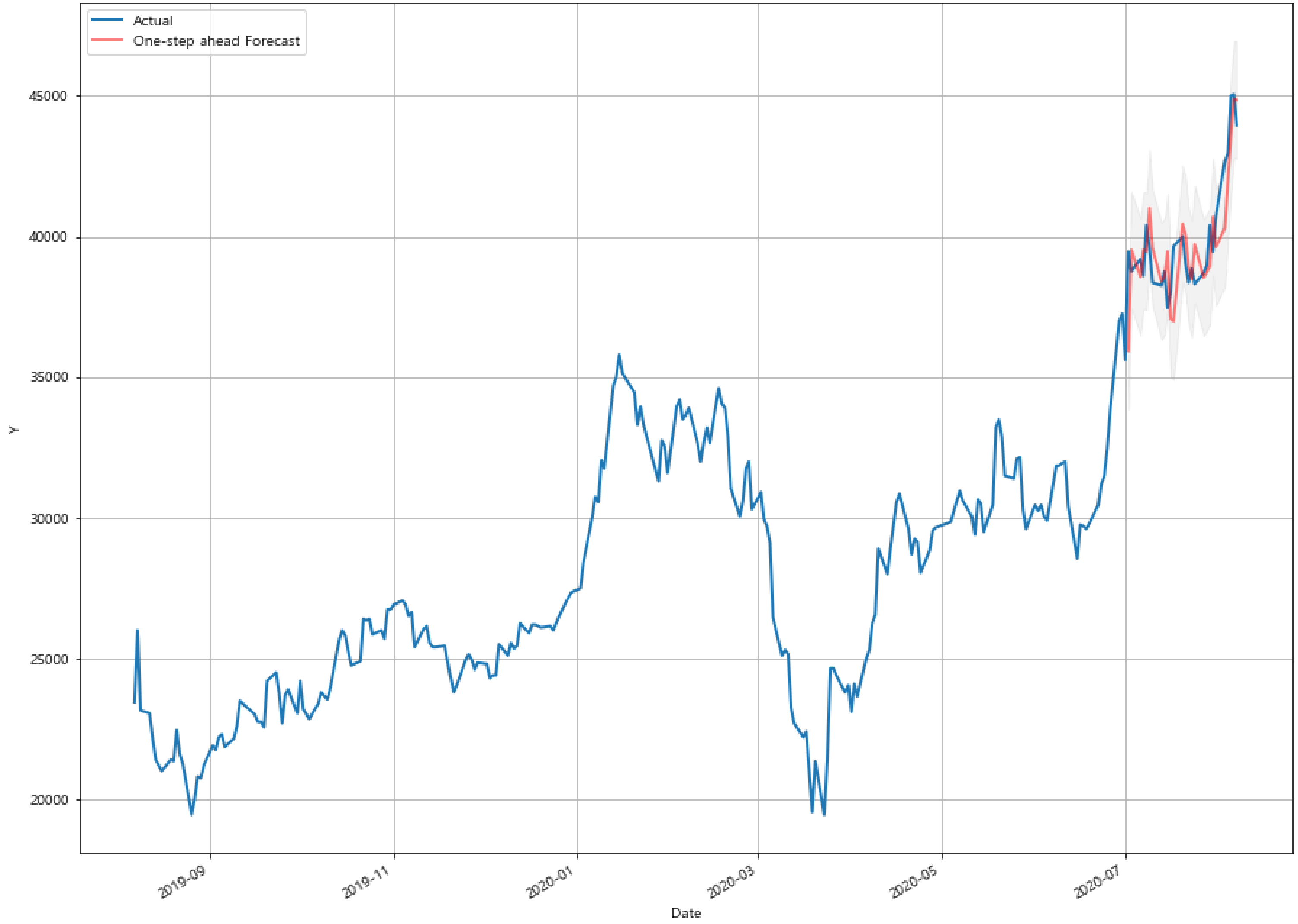
- Histogram plus estimated density에서 KDE, N(0,1)이 정규분포를 따르는 것으로 보임
- Normal Q-Q 에서 선형성
- Standardized residual에서 어떤 경향성도 보이지 않음 -> White Noise

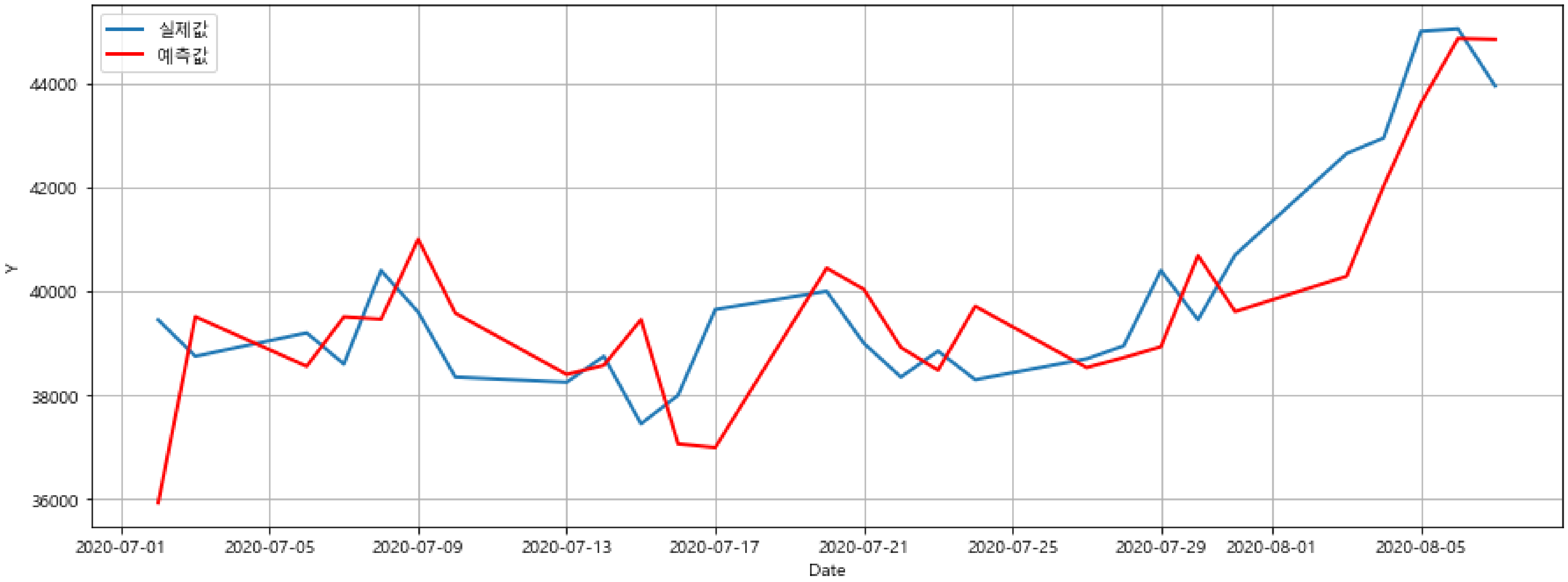
Forecast

1 month 예측

```
y_forecasted = pred.predicted_mean
y_truth = df['종가']['2020-07-02':]
print(y_forecasted.shape,y_truth.shape)
print(y_forecasted.index)
print(y_truth.index)
```

```
(27,) (27,)
DatetimeIndex(['2020-07-02', '2020-07-03', '2020-07-06', '2020-07-07',
               '2020-07-08', '2020-07-09', '2020-07-10', '2020-07-13',
               '2020-07-14', '2020-07-15', '2020-07-16', '2020-07-17',
               '2020-07-20', '2020-07-21', '2020-07-22', '2020-07-23',
               '2020-07-24', '2020-07-27', '2020-07-28', '2020-07-29',
               '2020-07-30', '2020-07-31', '2020-08-03', '2020-08-04',
               '2020-08-05', '2020-08-06', '2020-08-07'],
              dtype='datetime64[ns]', name='날짜', freq=None)
DatetimeIndex(['2020-07-02', '2020-07-03', '2020-07-06', '2020-07-07',
               '2020-07-08', '2020-07-09', '2020-07-10', '2020-07-13',
               '2020-07-14', '2020-07-15', '2020-07-16', '2020-07-17',
               '2020-07-20', '2020-07-21', '2020-07-22', '2020-07-23',
               '2020-07-24', '2020-07-27', '2020-07-28', '2020-07-29',
               '2020-07-30', '2020-07-31', '2020-08-03', '2020-08-04',
               '2020-08-05', '2020-08-06', '2020-08-07'],
              dtype='datetime64[ns]', name='날짜', freq=None)
```





$$\mathrm{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

MAPE	Interpretation
<10	Highly accurate forecasting
10-20	Good forecasting
20-50	Reasonable forecasting
>50	Inaccurate forecasting

Source: Lewis (1982, p. 40)

```python
# Compute the Mean of Absolute Percentage Errors

mape = ((abs( y_forecasted_d - y_truth_d ) / y_truth_d)*100).mean()
print('The MAPE of our forecasts is {}%'.format( round(mape, 2) ) )
```

```
The MAPE of our forecasts is 5.29%
```

Facebook – Prophet모델

https://zzsza.github.io/data/2019/02/06/prophet/

정상성 확인 방법 : Augmendted Dickey-Fuller, ACF, PACF

<Augmented Dickey-Fuller>

종가_변화율

$= \sum \ln(y_t) - \ln(y_{t-i})$

정상성 확인 방법 : Augmendted Dickey-Fuller, ACF, PACF

<Augmented Dickey-Fuller>

종가_로그

In [89]:

```
result = adfuller(df_close_log)
print('ADF Statistic: {}'.format(result[0]))
print('p-value: {}'.format(result[1]))
print('Critical Values:')
for key, value in result[4].items():
    print('₩t{}: {}'.format(key, value))
```

```
ADF Statistic: -2.470474285938524
p-value: 0.12282450264219175
Critical Values:
	1%: -3.4338536404563853
	5%: -2.863087660163165
	10%: -2.5675939181074106
```

> 0.05 (유의수준 5%)

정상성 확인 방법 : Augmendted Dickey-Fuller, ACF, PACF

<Cube-root Test>

종가_1차차분

```python
def cube_root(series):

    lst=[]
    for i in series:
        if i>0:
            i=round(i**(1./3),3)
        elif i<0:
            i=-i
            i=round(i**(1/3),3)
        else: # i==0
            i=0

        lst.append(i)
```

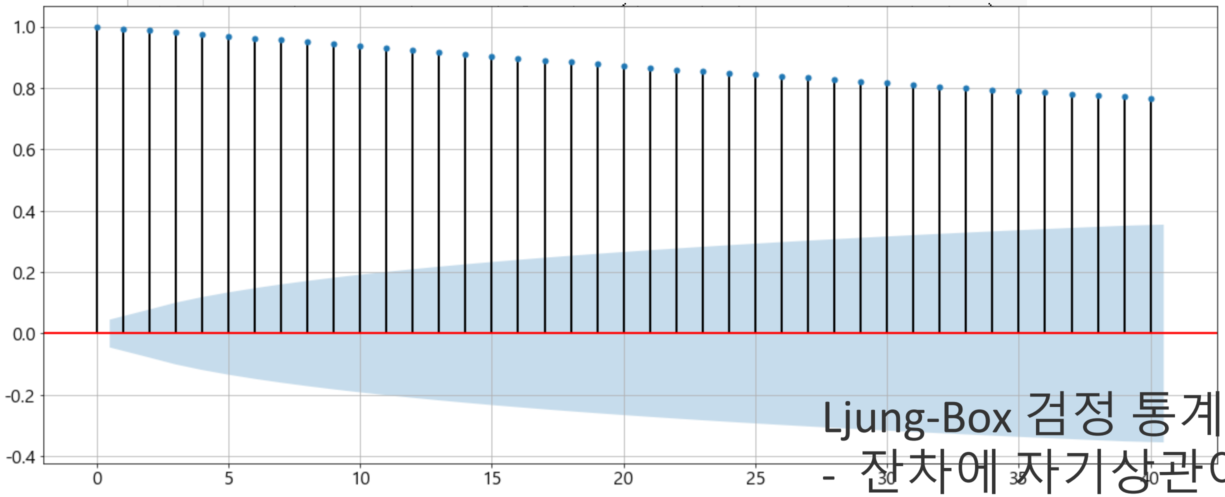
```python
result = adfuller(y_cls_cbrt)
print('ADF Statistic: {}'.format(result[0]))
print('p-value: {}'.format(result[1]))
print('Critical Values:')
for key, value in result[4].items():
    print('\t{}: {}'.format(key, value))
```

```
ADF Statistic: -46.01973139467104
p-value: 0.0
Critical Values:
    1%: -3.4338536404563853
```

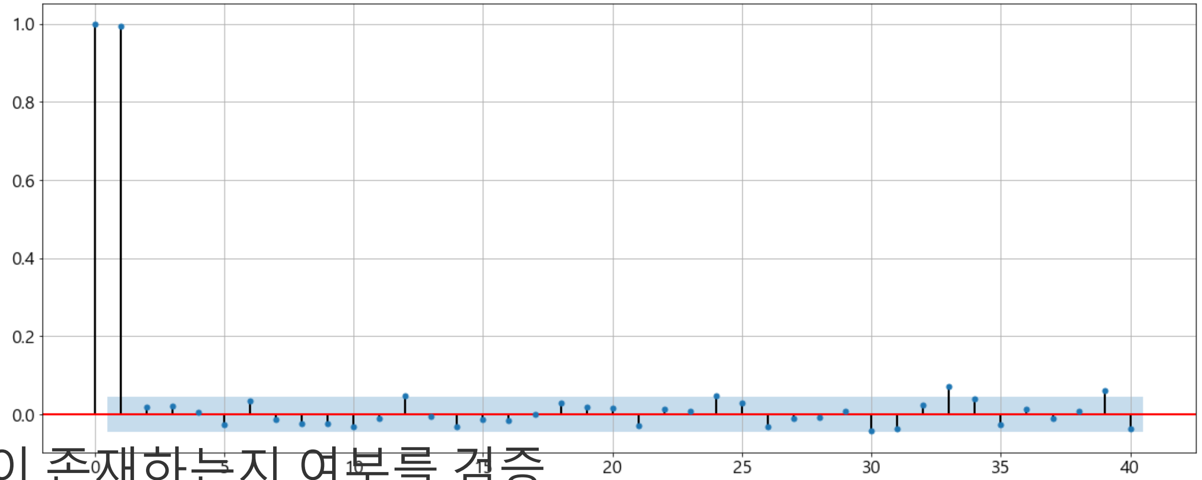
< 0.05 (유의수준 5%)

<종가_log>

ACF



PACF



Ljung-Box 검정 통계

- 잔차에 자기상관이 존재하는지 여부를 검증