데이터 준비

In [1]:
```
import pandas as pd
import numpy as np
df = pd.read_csv("C:/Users/Gilseung/Downloads/market-price.csv", header = None)
df.columns = ["date", "price"]
```

In [2]:
```
df.head()
```

Out[2]:

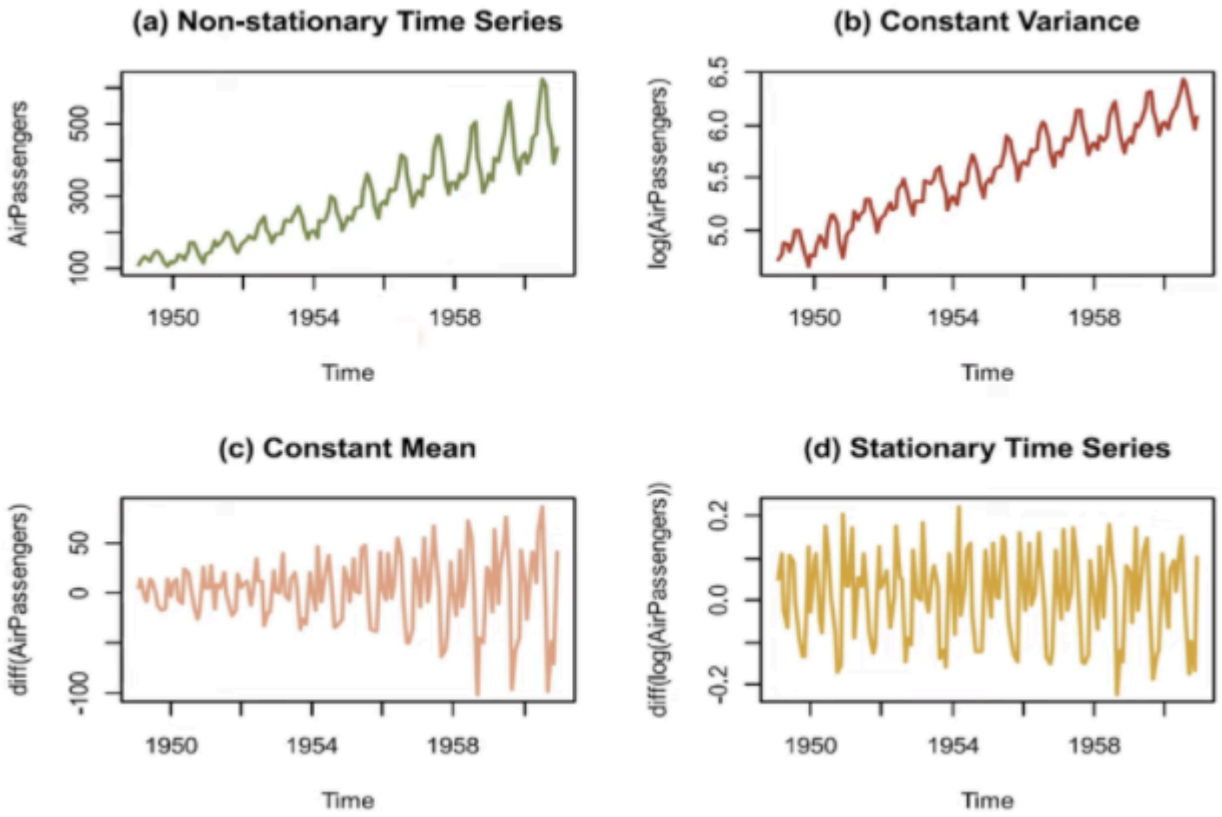
	date	price
0	2017-08-27 00:00:00	4354.308333
1	2017-08-28 00:00:00	4391.673517
2	2017-08-29 00:00:00	4607.985450
3	2017-08-30 00:00:00	4594.987850
4	2017-08-31 00:00:00	4748.255000

In [3]:
```
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace = True)
```

정상성

정상성 확인

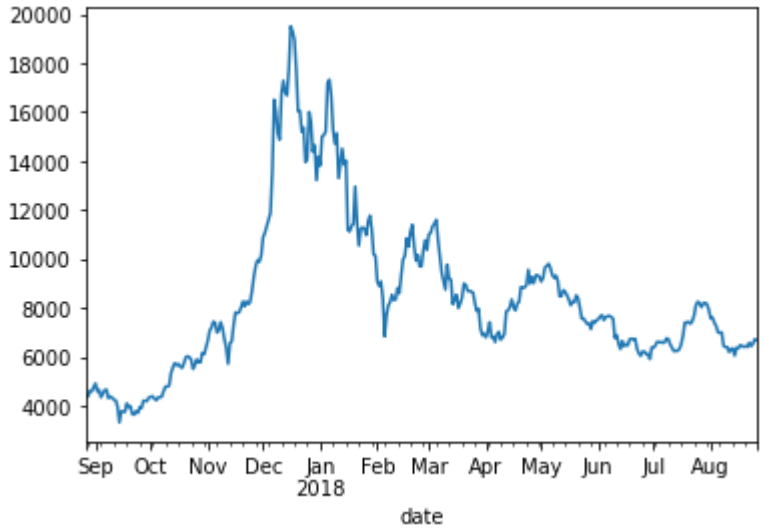
- 평균이 일정하다.
- 분산이 시점에 의존하지 않는다.
- 공분산은 시차에만 의존하며, 시점 자체에는 의존하지 않는다.



시각화

In [4]:
```
# 딱봐도 정상성 없어보임
df['price'].plot()
```

Out[4]:
```
<AxesSubplot:xlabel='date'>
```



Augmented Dickey-Fuller (ADF) 검정

- p-value가 유의 수준보다 작으면 정상 시계열로 간주 가능
- adfuller 함수의 출력값의 1번째 요소가 p-value임

In [5]:

```
from statsmodels.tsa.stattools import adfuller
Y = df['price']

print(round(adfuller(Y)[1], 4)) # 0.3972 --> 정상 시계열 X
```

```
0.3972
```

정상 시계열로 변환

- 평균이 일정하지 않으면 차분(difference) --> 차분해도 정상 시계열이 안되면 차분을 반복
- 분산이 일정하지 않으면 변환(transformation)

boxcox 수식

y = (x**lmbda - 1) / lmbda, for lmbda != 0

log(x), for lmbda = 0

- y = stats.boxcox(X)[0]
- lambda = stats.boxcox(X)[1]

In [6]:

```
from scipy import stats
Y_1diff = df['price'].diff().dropna() # 1차 차분
Y_2diff = df['price'].diff(2).dropna() # 2차 차분 (그러나 차분 반복하므로 실제 사용은 안함)
Y_log = np.log(df['price'])
boxcox_Y, l = stats.boxcox(Y)

print(round(adfuller(Y_1diff)[1], 4)) # 0.0 --> 정상 시계열 O
print(round(adfuller(Y_2diff)[1], 4)) # 0.0 --> 정상 시계열 O
print(round(adfuller(Y_log)[1], 4)) # 0.3972 --> 정상 시계열 X
print(round(adfuller(boxcox_Y)[1], 4)) # 0.3772 --> 정상 시계열 X
```

```
0.0
0.0005
0.3989
0.3772
```

MA, AR, ARIMA

기본 개념

AR 모형이란?

자귀 회귀 모형으로, Auto Correlation의 약자이다.
자기상관성을 시계열 모형으로 구성하였으며, 예측하고자 하는 특정 변수의 과거 관측값의 선형결합으로 해당 변수의 미래값을 예측하는 모형이다.
이전 자신의 관측값이 이후 자신의 관측값에 영향을 준다는 아이디어에 기반하였다.

AR(p) 모형의 식은 다음과 같다.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

y_t는 t시점의 관측값, c는 상수, ϕ는 가중치, ε_t는 오차항을 의미한다.

MA 모형이란?

Moving Average 모형으로, 예측 오차를 이용하여 미래를 예측하는 모형이다.

MA(q) 모형의 식은 다음과 같다.

$$y_t = c + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

ARIMA 모형이란?

ARIMA(p,d,q) 모형은 d차 차분한 데이터에 위 AR(p) 모형과 MA(q) 모형을 합친 모형으로, 식은 다음과 같다.

$$y_t' = c + \phi_1 y_{t-1}' + \phi_2 y_{t-2}' + \ldots + \phi_p y_{t-p}' + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

단, y'은 d차 차분을 구한 시계열, p는 자기회귀 부분의 차수, d는 차분 회수, q는 이동평균 부분의 차수이다.

AR(p)모형과 ARIMA(p,0,0)모형은 같은 모형이며, MA(q)모형과 ARIMA(0,0,q) 모형은 같은 모형이다.

차수 결정

ACF / PACF 그래프

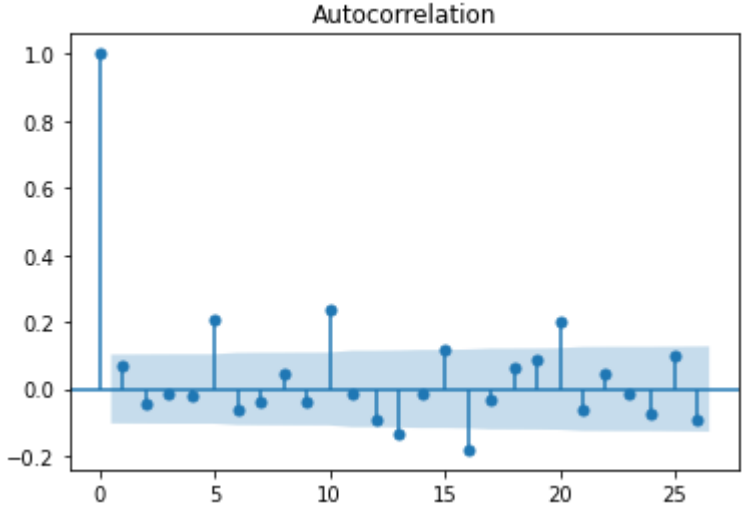
즉, ACF와 PACF를 동시에 고려하여 ARIMA 모델의 p와 q를 결정하는데 그 방법은 다음과 같다.

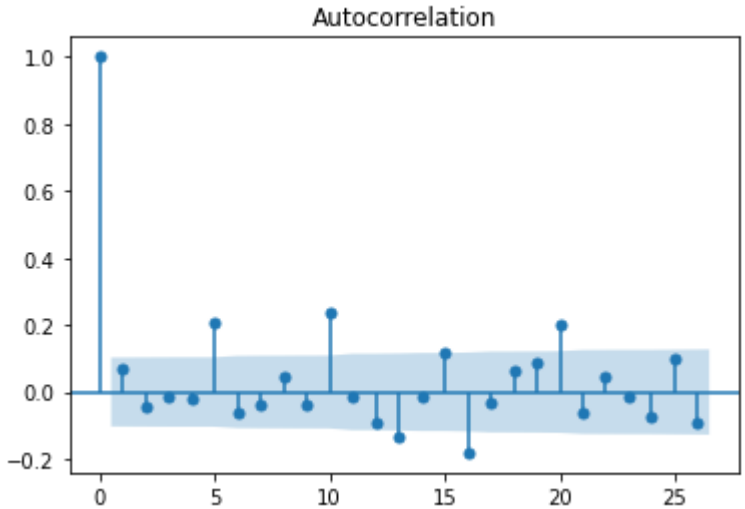
Model	ACF	PACF
AR(p)	점차 감소하여 0에 접근	시차 p 이후에 0
MA(q)	시차 q 이후에 0	점차 감소하여 0에 접근
ARMA(p,q)	점차 감소하여 0에 접근 (시차 q 이후에 0)	점차 감소하여 0에 접근 (시차 p 이후에 0)

In [7]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(Y_1diff)
# acf는 1 이후에 0에 가까움 --> MA 차수는 1
```

Out[7]:

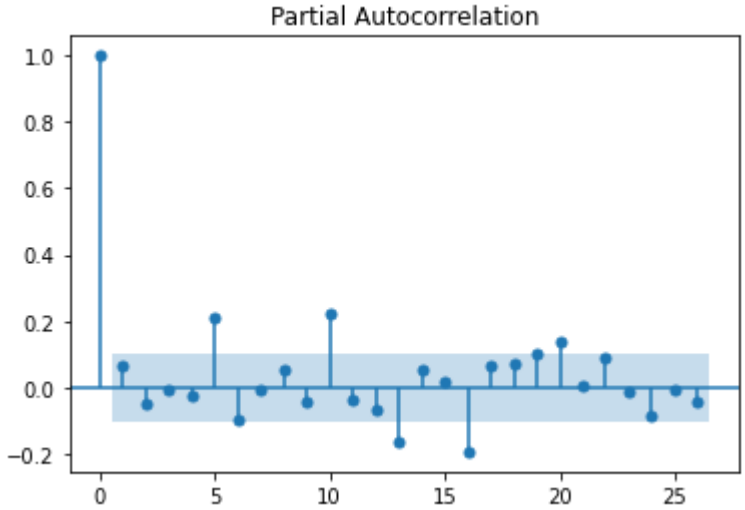


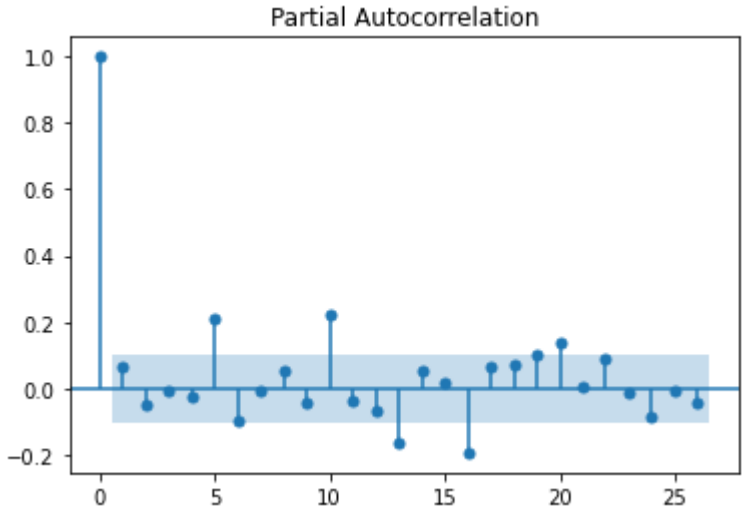


In [8]:

```
# pacf는 1 이후에 0에 가까움 --> AR 차수는 1
plot_pacf(Y_1diff)
```

Out[8]:





AIC를 이용한 그리드 서치: AIC가 작을수록 좋음

In [9]:

```
from statsmodels.tsa.arima_model import ARIMA
import itertools
import warnings
warnings.filterwarnings("ignore")

p = range(0,7)
d = range(0,1)
q = range(0,7)
pdq = list(itertools.product(p,d,q))

dict_model ={}
```

```
for i in pdq:

    try:
        model = ARIMA(Y_1diff, order=(i))
        model_fit = model.fit(maxiter = 100)
        dict_model[i]=[model_fit.llf, model_fit.aic]

    except:
        print("error lag:",i)

result=pd.DataFrame.from_dict(dict_model, orient ="index", columns =["llf", "AIC"])
```

```
error lag: (3, 0, 4)
```

In [65]:

```
result.sort_values(by = "AIC", ascending = True) # (6, 0, 6)이 best
```

Out[65]:

	llf	AIC
(6, 0, 6)	-2762.193000	5552.386001
(5, 0, 4)	-2767.544127	5557.088254
(5, 0, 5)	-2767.291775	5558.583550
(4, 0, 6)	-2767.500816	5559.001633
(6, 0, 4)	-2767.620487	5559.240973
(5, 0, 6)	-2767.131829	5560.263658
(6, 0, 5)	-2767.169022	5560.338044
(6, 0, 2)	-2771.399033	5562.798065
(6, 0, 3)	-2771.369657	5564.739315
(2, 0, 6)	-2772.730756	5565.461512
(3, 0, 6)	-2772.093056	5566.186112
(4, 0, 2)	-2775.305185	5566.610370
(5, 0, 3)	-2774.476934	5568.953867
(5, 0, 2)	-2777.772772	5573.545544
(6, 0, 0)	-2778.801941	5573.603882
(3, 0, 3)	-2779.201033	5574.402066
(5, 0, 1)	-2779.306093	5574.612187
(5, 0, 0)	-2780.529747	5575.059495

	llf	AIC
(6, 0, 1)	-2778.796259	5575.592518
(2, 0, 5)	-2779.055919	5576.111838
(2, 0, 4)	-2780.153814	5576.307628
(3, 0, 5)	-2778.486034	5576.972067
(4, 0, 3)	-2779.688228	5577.376455
(0, 0, 5)	-2782.190036	5578.380072
(0, 0, 6)	-2781.415440	5578.830880
(4, 0, 5)	-2778.484238	5578.968475
(4, 0, 4)	-2779.576866	5579.153733
(1, 0, 5)	-2781.803356	5579.606712
(1, 0, 6)	-2781.321379	5580.642757
(1, 0, 1)	-2787.580718	5583.161436
(0, 0, 0)	-2789.858881	5583.717762
(0, 0, 1)	-2788.944096	5583.888191
(1, 0, 0)	-2789.031459	5584.062917
(0, 0, 2)	-2788.518739	5585.037478
(1, 0, 2)	-2787.550783	5585.101567
(2, 0, 0)	-2788.560105	5585.120211
(4, 0, 1)	-2785.576420	5585.152841
(1, 0, 4)	-2786.031593	5586.063186
(0, 0, 3)	-2788.478981	5586.957963
(1, 0, 3)	-2787.529348	5587.058695
(3, 0, 0)	-2788.546089	5587.092179
(2, 0, 1)	-2788.547685	5587.095370
(2, 0, 2)	-2787.553310	5587.106619
(0, 0, 4)	-2787.935212	5587.870425
(2, 0, 3)	-2787.305335	5588.610670
(3, 0, 2)	-2787.308958	5588.617917

	llf	AIC
(4, 0, 0)	-2788.453270	5588.906540
(3, 0, 1)	-2788.544042	5589.088085

모델 학습 및 예측

In [11]:
```
model = ARIMA(Y_1diff, order=(3, 0, 1))
model_fit = model.fit(maxiter = 1000) # 모델 학습

pd.DataFrame({"실제":Y_1diff, "예측":model_fit.predict()})
```

Out[11]:

	실제	예측
2017-08-28	37.365183	6.337275
2017-08-29	216.311933	8.424635
2017-08-30	-12.997600	19.605493
2017-08-31	153.267150	-5.824706
2017-09-01	163.485017	15.650900
...	...	...
2018-08-22	173.983013	2.726573
2018-08-23	-140.347500	19.890193
2018-08-24	108.764048	-12.002535
2018-08-25	175.783516	19.336548
2018-08-26	-46.155064	14.382185

364 rows × 2 columns

model_fit.forecast() = (forecast, stderr, conf_int)

- forecast: ndarray, Array of out of sample forecasts
- stderr: ndarray, Array of the standard error of the forecasts.
- conf_int: ndarray, 2d array of the confidence interval for the forecast

In [12]:
```
# forecast를 이용한 예측
model_fit.forecast(steps=7)[0]
```

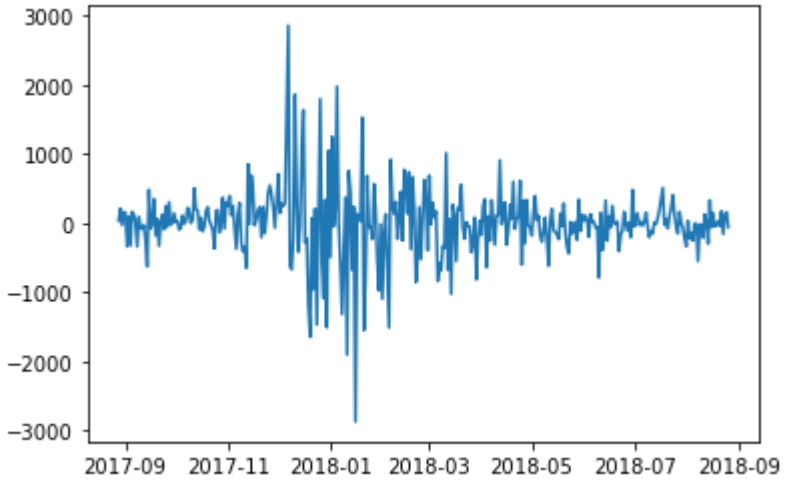
Out[12]:
```
array([-6.72013339,  6.42391462,  7.39600944,  6.55094248,  6.3063976 ,
        6.31494228,  6.33473943])
```

잔차 그래프 출력 및 해석

In [15]:
```
from matplotlib import pyplot as plt
E = Y_1diff -  model_fit.predict()
plt.plot(E)

# 그래프 상에서 편차가 심한 부분 (2017년 12월 ~ 2018년 2월)이 존재 --> 등분산성 만족 못할듯?
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x2d55cae1a00>]
```



In [17]:
```
# 잔차의 평균이 0에 가까움: 편향되지 않았음
E.mean()
```

Out[17]:
```
0.001226024198111137
```

In [18]:
```
# 정상성 검증 --> 정상
print(round(adfuller(E)[1], 4)) # 0.0 --> 정상 시계열 O
```

```
0.0
```

In [19]:
```
# 정규분포 검증: p value가 0.05 미만이므로 정규분포를 따른다고 보기 힘듦
from scipy import stats
k2, p = stats.normaltest(E)
print(p)
```

```
2.2146371257194384e-14
```

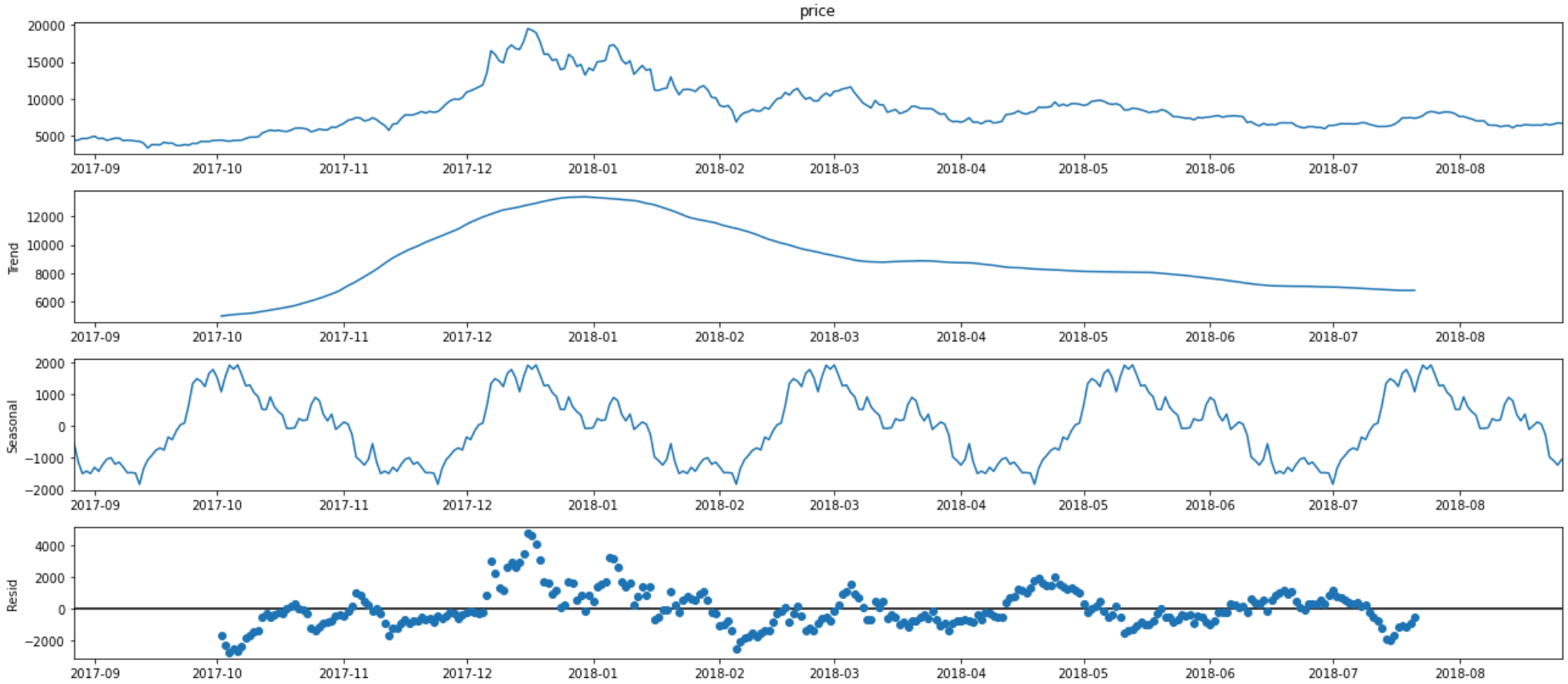
SARIMA: Seasonal ARIMA

statsmodels.tsa.statespace.sarimax.SARIMAX(endog, exog=None, order=(1, 0, 0), seasonal_order=(0, 0, 0, 0), trend=None, measurement_error=False, time_varying_regression=False, mle_regression=True, simple_differencing=False, enforce_stationarity=True, enforce_invertibility=True, hamilton_representation=False, concentrate_scale=False, trend_offset=1, use_exact_diffuse=False, dates=None, freq=None, missing='none', validate_specification=True, **kwargs)

seasonal_decompose

In [61]:

```
import statsmodels.api as sm
decomposition = sm.tsa.seasonal_decompose(Y, model='additive', period = int(len(Y) / 5)) #
# period: 반복주기 (즉, int(len(Y)/5)는 Y에 주기가 다섯 번 포함되었다는 이야기)
fig = decomposition.plot()
plt.show()
```



하이퍼 파라미터 튜닝

In [62]:

```
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
print('Examples of parameter for SARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter for SARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

In [64]:

```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(Y, order=param,seasonal_order=param_seasonal,enforce_stationarity=False,enforce_inver
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param,param_seasonal, results.aic))
        except:
            continue
```

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:7663.9900657544895
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:7045.375881101838
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:6356.5695456408885
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:6151.480055746566
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:6372.673237286446
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:6357.420106705768
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:6168.5347668521
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:6153.97851920833
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:7380.771233776079
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:7080.314675970305
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:5982.450328596621
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:5788.943439839841
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:6014.2643680588935
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:5982.271753002396
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:5821.368122721085
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:5790.766758744484
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:5567.438118284642
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:5391.693775067902
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:5668.7005409398525
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:5247.829054863669
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:5406.824715885541
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:5392.384422095154
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:5400.470803331551
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:5246.139455805438
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:5553.106050609133
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:5378.2471870208665
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:5646.862388106902
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:5233.385729528412
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:5407.633259453982
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:5378.810647116834
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:5401.04663589194
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:5232.348676377371
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:5583.611583055569
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:5414.802189991573
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:5672.627388774592
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:5262.496933593559
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:5408.782388491135
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:5408.76591977426
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:5395.282323324129
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:5261.507137119598
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:5569.407546297522
```

```
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:5402.39683775181
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:5645.7613242865045
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:5247.280378105937
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:5409.498528795242
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:5395.165941666838
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:5394.502654659082
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:5246.953385127753
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:5567.775768351542
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:5392.611262140451
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:5664.688269143238
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:5248.077990030424
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:5393.3945036935165
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:5393.232971091336
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:5386.1172696564845
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:5246.945480008049
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:5552.881604626817
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:5378.398053229088
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:5634.8202395021235
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:5232.741307005787
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:5393.278878856841
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:5379.035313522883
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:5382.513691214475
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:5234.945225173917
```

MARIMA: Multivariate ARIMA

class statsmodels.tsa.statespace.varmax.VARMAX(endog, exog=None, order=(1, 0), trend='c', error_cov_type='unstructured', measurement_error=False, enforce_stationarity=True, enforce_invertibility=True, trend_offset=1, **kwargs)

This is a brief introduction notebook to VARMAX models in statsmodels. The VARMAX model is generically specified as:

$$y_t = \nu + A_1 y_{t-1} + \cdots + A_p y_{t-p} + B x_t + \epsilon_t + M_1 \epsilon_{t-1} + \ldots M_q \epsilon_{t-q}$$

where y_t is a $\mathrm{k\_endog} \times 1$ vector.

In [66]:

```
dta = sm.datasets.webuse('lutkepohl2', 'https://www.stata-press.com/data/r12/')
dta.index = dta.qtr
dta.index.freq = dta.index.inferred_freq
endog = dta.loc['1960-04-01':'1978-10-01', ['dln_inv', 'dln_inc', 'dln_consump']]

exog = endog['dln_consump']
# input: endog[['dln_inv', 'dln_inc']], output: exog
mod = sm.tsa.VARMAX(endog[['dln_inv', 'dln_inc']], order=(2,0), trend='n', exog=exog)
```

```
res = mod.fit(maxiter=1000, disp=False)
print(res.summary())
```

```
                            Statespace Model Results
==============================================================================
Dep. Variable:     ['dln_inv', 'dln_inc']   No. Observations:                   75
Model:                            VARX(2)   Log Likelihood                 361.039
Date:                    Sat, 04 Sep 2021   AIC                           -696.078
Time:                            13:17:03   BIC                           -665.950
Sample:                        04-01-1960   HQIC                          -684.048
                             - 10-01-1978
Covariance Type:                      opg
===================================================================================
Ljung-Box (L1) (Q):            0.05, 10.15   Jarque-Bera (JB):          11.14, 2.40
Prob(Q):                        0.83, 0.00   Prob(JB):                   0.00, 0.30
Heteroskedasticity (H):         0.45, 0.40   Skew:                      0.16, -0.38
Prob(H) (two-sided):            0.05, 0.03   Kurtosis:                   4.86, 3.44
                           Results for equation dln_inv
==============================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
L1.dln_inv          -0.2404      0.093     -2.587      0.010      -0.423      -0.058
L1.dln_inc           0.2913      0.449      0.648      0.517      -0.590       1.172
L2.dln_inv          -0.1658      0.155     -1.069      0.285      -0.470       0.138
L2.dln_inc           0.0718      0.421      0.171      0.865      -0.754       0.897
beta.dln_consump     0.9615      0.639      1.505      0.132      -0.290       2.213
                           Results for equation dln_inc
==============================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
L1.dln_inv           0.0632      0.036      1.765      0.078      -0.007       0.133
L1.dln_inc           0.0816      0.107      0.761      0.447      -0.129       0.292
L2.dln_inv           0.0099      0.033      0.300      0.764      -0.055       0.074
L2.dln_inc           0.0330      0.134      0.246      0.805      -0.230       0.296
beta.dln_consump     0.7755      0.112      6.910      0.000       0.556       0.996
                            Error covariance matrix
==============================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sqrt.var.dln_inv           0.0434      0.004     12.301      0.000       0.036       0.050
sqrt.cov.dln_inv.dln_inc  4.79e-05      0.002      0.024      0.981      -0.004       0.004
sqrt.var.dln_inc           0.0109      0.001     11.216      0.000       0.009       0.013
==============================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

In [69]:

```
res.predict()
```

Out[69]:

qtr	dln_inv	dln_inc
1960-04-01	0.028275	0.028974
1960-07-01	0.035506	0.027036
1960-10-01	0.037984	0.031099
1961-01-01	0.016717	0.024212
1961-04-01	-0.020466	0.007786
...	...	...
1977-10-01	0.024182	0.015739
1978-01-01	0.011379	0.013705
1978-04-01	0.011913	0.017746
1978-07-01	0.004931	0.014283
1978-10-01	0.001865	0.009078

75 rows × 2 columns