시계열 분해 추세, 계절성, 잔차로 분해하는 기법 Addictive, Multiplicative 뭐가 맞는지

In [33]:
```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
data=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/arima_data.csv",names=['day','price'])
```

In [34]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   day     60 non-null     object
 1   price   60 non-null     int64
dtypes: int64(1), object(1)
memory usage: 1.1+ KB
```

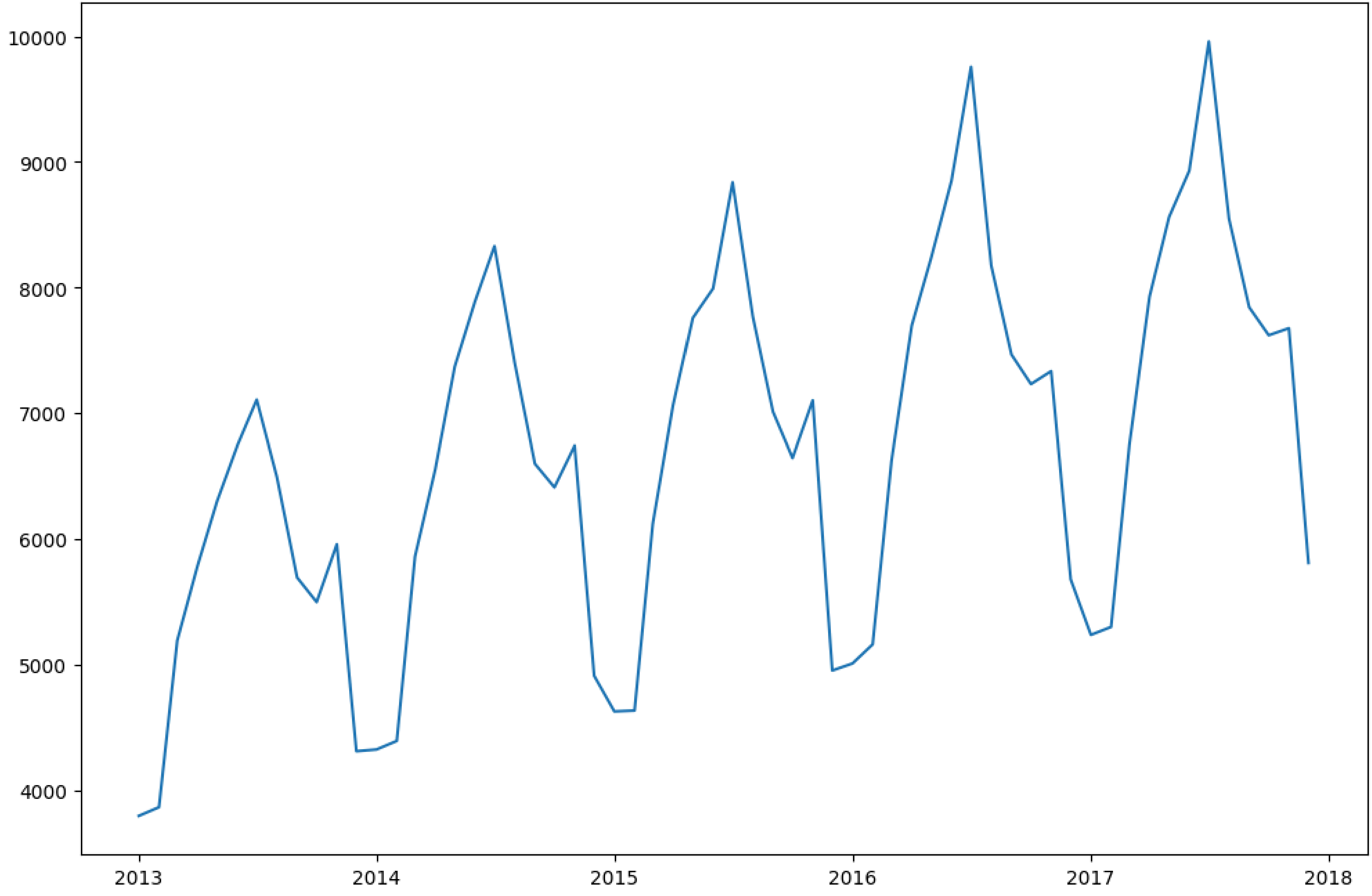
In [35]:
```
data['day']=pd.to_datetime(data['day'],format='%Y-%m-%d')
data.set_index('day',inplace=True)
data.head(3)
```

Out[35]:

day	price
2013-01-01	3794
2013-02-01	3863
2013-03-01	5190

In [36]:
```
import matplotlib.pyplot as plt
plt.plot(data.index,data['price'])
```

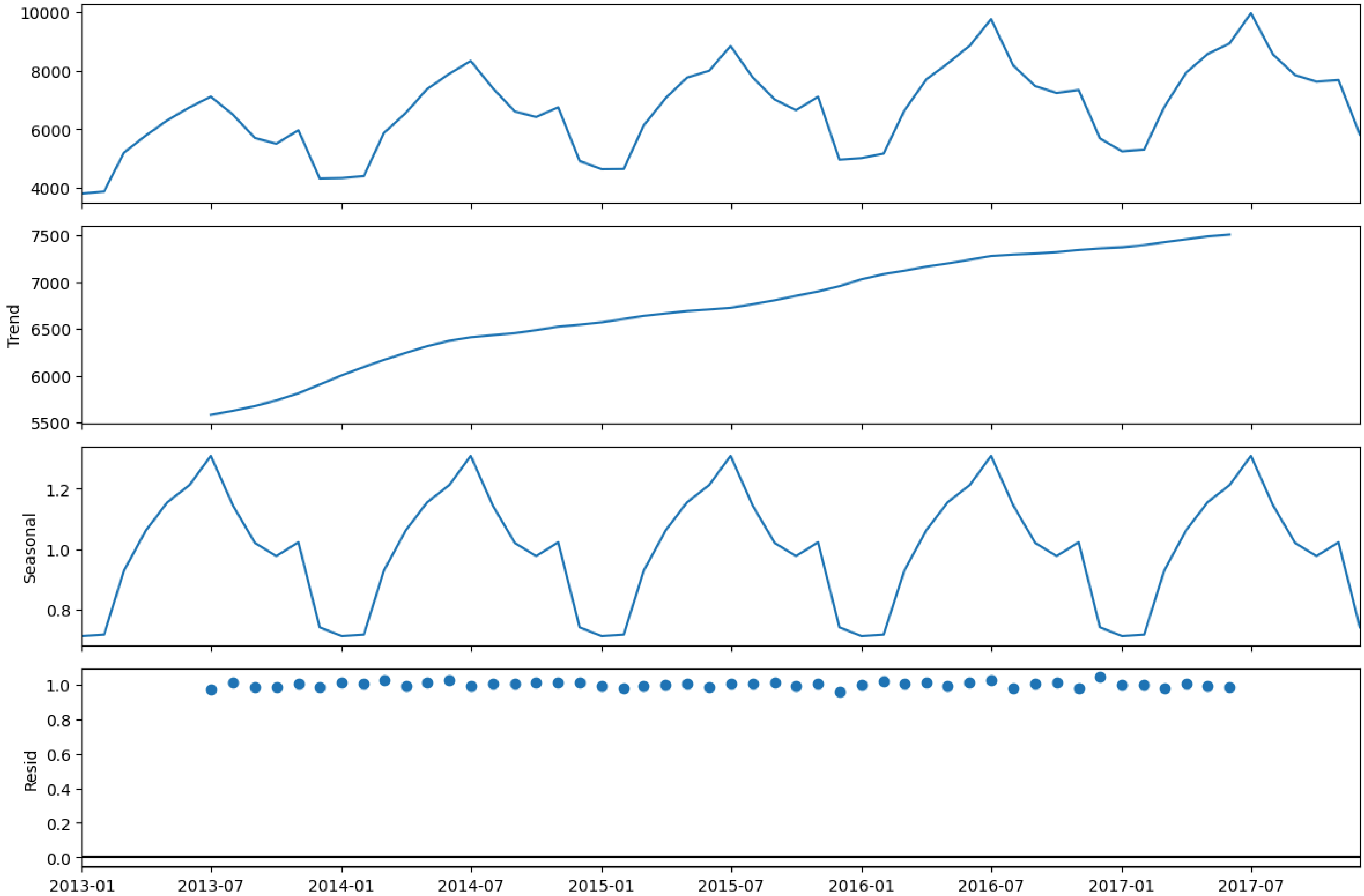
Out[36]:
```
[<matplotlib.lines.Line2D at 0x340421450>]
```


10000
9000
8000
7000
6000
5000
4000
2013
2014
2015
2016
2017
2018

📊 2️⃣ 직관적인 차이

항목	Additive	Multiplicative
수식	$Y_t = T_t + S_t + R_t$	$Y_t = T_t \times S_t \times R_t$
계절성의 크기	**고정** (진폭 일정)	**비례적** (추세가 커질수록 진폭 커짐)
데이터 형태	절댓값 기준으로 등락이 일정	추세가 커질수록 계절 진폭도 커짐
예시	하루 매출이 100 ± 10 반복	매출이 100 → 200 늘면 진폭도 10 → 20으로 커짐
모델링 예시	온도, 기압, 주식수익률 등	판매량, 방문자수 등 (비율적 변화)

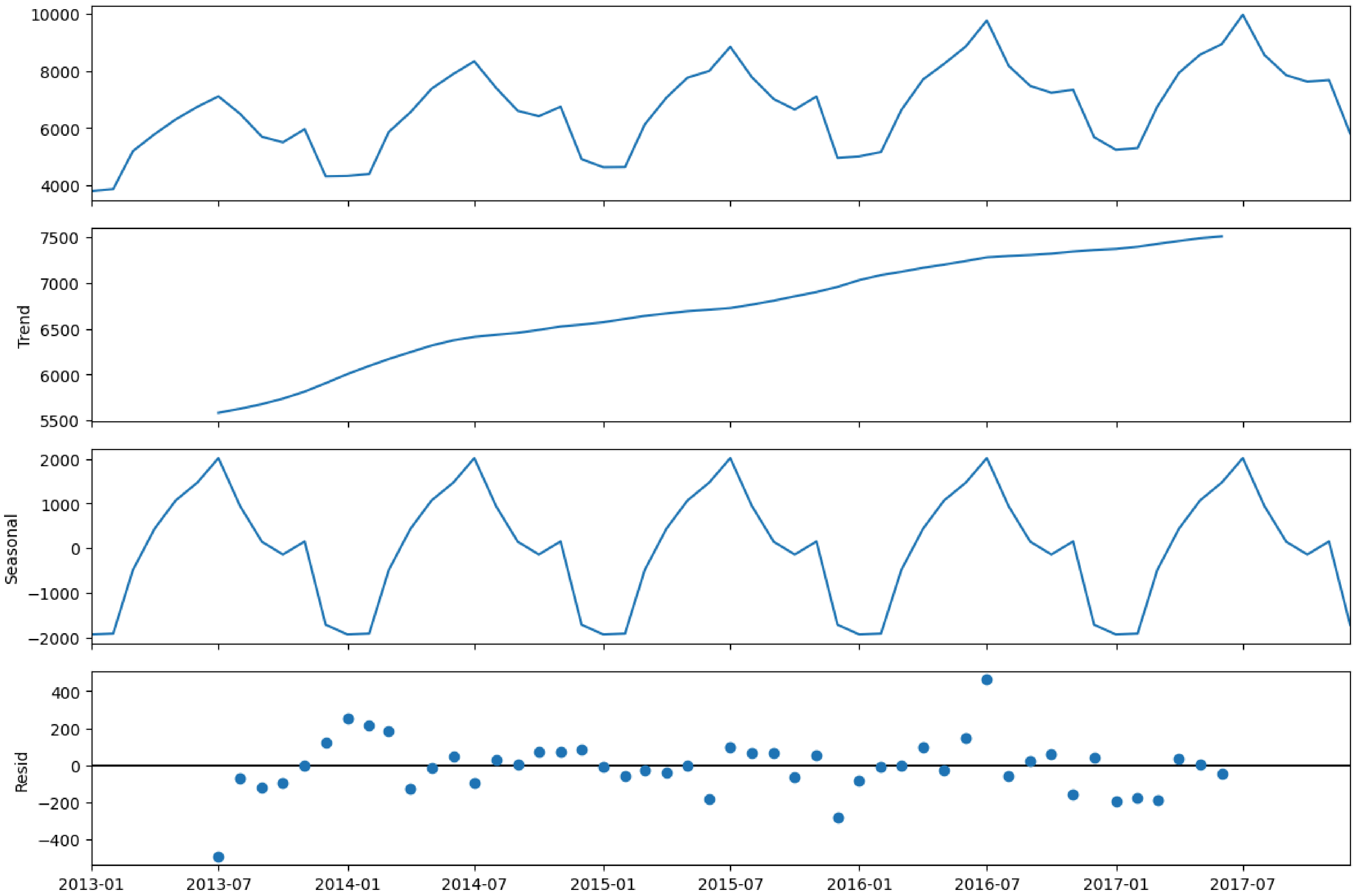
In [37]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

ts=data
result=seasonal_decompose(ts,model='multiplicative')
plt.rcParams['figure.figsize']=[12,8]
result.plot()
plt.show()
```



In [38]:

```
from statsmodels.tsa.seasonal import seasonal_decompose

ts=data
result=seasonal_decompose(ts,model='addictive')
plt.rcParams['figure.figsize']=[12,8]
```

```python
result.plot()
plt.show()
```



시계열 형태	적절한 `regression` 값
평균만 있는 정상 시계열	`'c'`

시계열 형태	적절한 `regression` 값
추세를 포함한 비정상 시계열	`'ct'`
이차 추세 포함	`'ctt'`
평균도 없는 시계열 (ex: 무의미한 잔차)	`'n'`

🧠 1️⃣ ARIMA 모형 기본식

$$Y_t = \mu + (\text{AR terms}) + (\text{MA terms}) + \text{trend terms} + \varepsilon_t$$

여기서 `trend` 옵션은 이 식의 "μ" 또는 "추세항"이 포함되는지 결정합니다.

⚙️ 2️⃣ `trend` 옵션 종류 (statsmodels 기준)

옵션	의미	수식 형태	설명
`'n'`	no trend	없음	**상수항 없이 원점을 통과하는 모형**
`'c'`	constant only	μ	**상수항(평균 수준)** 포함
`'t'`	linear trend only	βt	**시간에 따라 선형 추세** 포함
`'ct'`	constant + linear trend	$\mu + \beta t$	**상수항 + 선형 추세** 포함
`'ctt'`	constant + quadratic trend	$\mu + \beta_1 t + \beta_2 t^2$	**상수 + 1차 + 2차 추세 포함**

> 💡 **'nc'는 옛 버전(`statsmodels < 0.12`)에서 사용된 표현으로,**
> `"no constant"` → 현재 버전에서는 `'n'` 로 대체됨.
> 즉, `trend='nc'` == `trend='n'`.

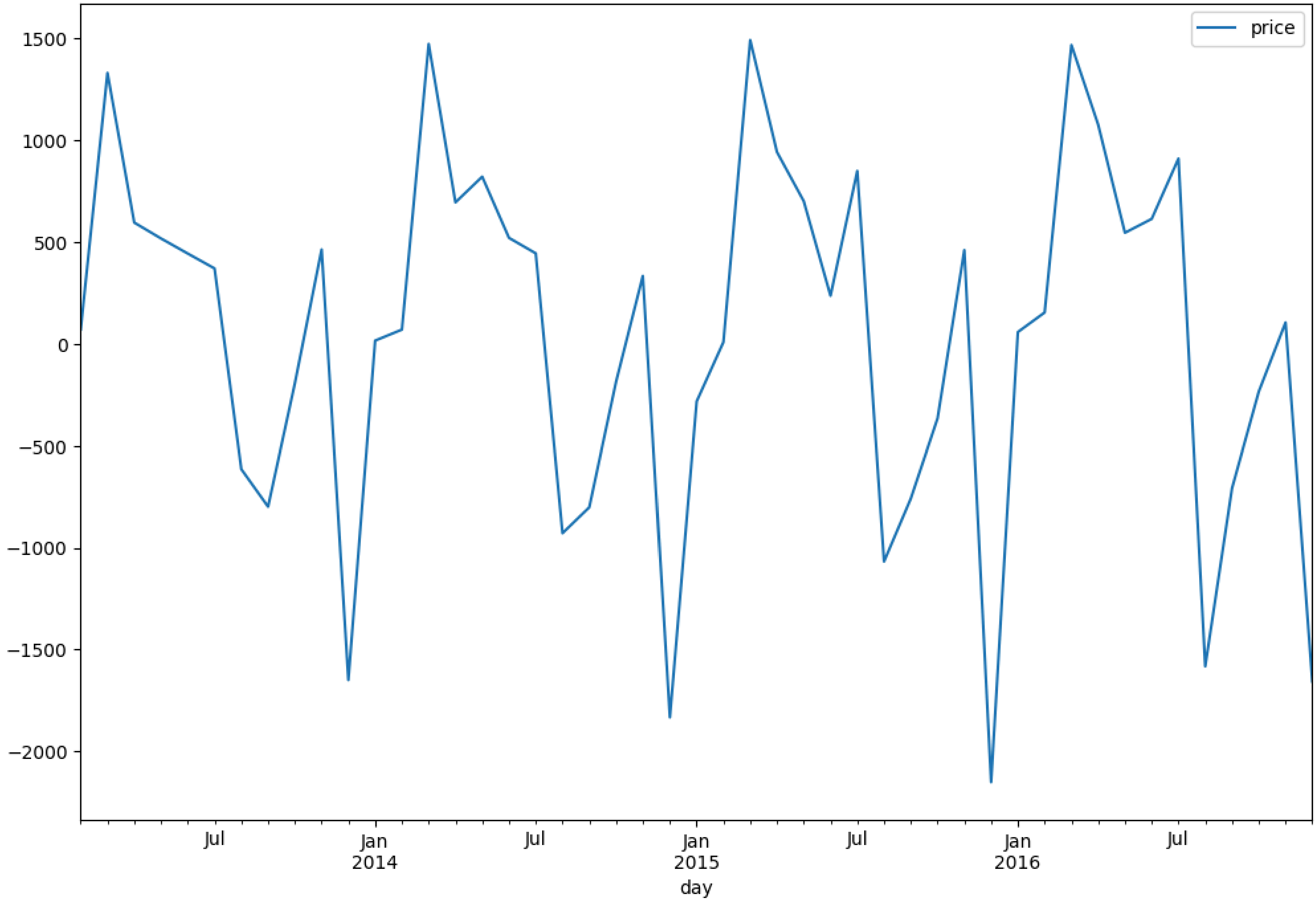
In [39]:
```
from statsmodels.tsa.stattools import adfuller
# ADF (Augmented Dickey–Fuller) Test,
# 즉 확장된 딕키-풀러 검정으로,
# 시계열이 정상성(stationarity) 을 만족하는지 판단하는 대표적인 통계적 방법입니다.
# 예시:
# 정상 시계열 → 노이즈, AR(1) 등
```

```python
# 비정상 시계열 → 추세(trend)나 계절성이 있는 데이터 (예: GDP, 기온, 매출 등)
# 🧩 2️⃣ ADF (Augmented Dickey–Fuller) 검정의 목적
# ADF 검정은 시계열에 단위근(unit root) 이 존재하는지 확인합니다.
# 단위근이 있다 → 비정상 (non-stationary)
# 단위근이 없다 → 정상 (stationary)
# 즉, “데이터가 자기자신의 과거 값에 너무 의존해서 랜덤워크 형태인가?” 를 확인하는 거예요.
training = data[:'2016-12-01']
test=data.drop(training.index)
adf=adfuller(training,regression='ct') #h0 : 정상성을 갖지 않는다. h1 : 정상성을 가진다.
print('ADF Statistic: {}'.format(adf[0]))
print('p-value: {}'.format(adf[1]))
```

```
ADF Statistic: -1.9997199341327319
p-value: 0.6015863303794438
```

In [40]:
```python
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

diff_data=training.diff(1)
diff_data=diff_data.dropna()
diff_data.plot()
```

Out[40]:
```
<Axes: xlabel='day'>
```



In [41]:
```
adf=adfuller(diff_data)
print('ADF Statistic: {}'.format(adf[0]))
print('p-value: {}'.format(adf[1]))
```

```
ADF Statistic: -12.094547576926377
p-value: 2.085160639961547e-22
```

확률과정	ACF	PACF
AR(p)	지수적으로 감소하거나 소멸하는 싸인함수 형태	시차 p이후에는 0으로의 절단형태
MA(q)	시차 q이후에는 0으로의 절단형태	지수적으로 감소하거나 소멸하는 싸인함수 형태
ARMA(p,q)	시차(q-p)이후에는 지수적으로 감소하거나 소멸하는 사인함수형태	시차(p-q)이후에는 지수적으로 감소하거나 소멸하는 싸인함수 형태

=> 시계열그림과 identity 문에 의한 결과로 나오는 ACF, PACF의 형태를 보고 차분의 필요여부 및 차수를 d를 결정하고 AR차수와 MA차수를 결정하면 된다.

PACF는 ACF가 모든 시계열 데이터의 특성을 분석하는 것에 한계가 있기에 추가적인 분석의 필요에 따라 사용된다. 예를 들어 Y_t와 Y_{t-1}이 관계가 있다면 Y_{t-1}와 Y_{t-2}도 상관이 있고 Y_t와 Y_{t-2}는 Y_t와 Y_{t-1}간의 유의미한 관계가 있음으로 인해 상관있다는 결과가 도출될 수 있다는 점이다. 이 추가적인 분석에선 부분 상관이란 확률 변수인 X와 Y에 의해 모든 변수들에 대한 상관 관계를 분석한 후에도 아직 남아있는 상관관계를 해석한다.

PACF는 y_t와 y_{t+k}간 상관 관계를 도출하는 것은 같지만 t와 $t+k$ 간 다른 y값의 영향력은 배제하고 측정하는 방식이다. 즉 시차 k에서의 k단계만큼 떨어져 있는 모든 데이터들 간의 상관 관계를 말한다.

PACF가 시차 n에서 떨어지는 경우 AR(n)을 사용하고 하락이 점진적이라면 MA를 사용한다.

아래는 PACF 수식과 python 코드로 구현한 내용이다.

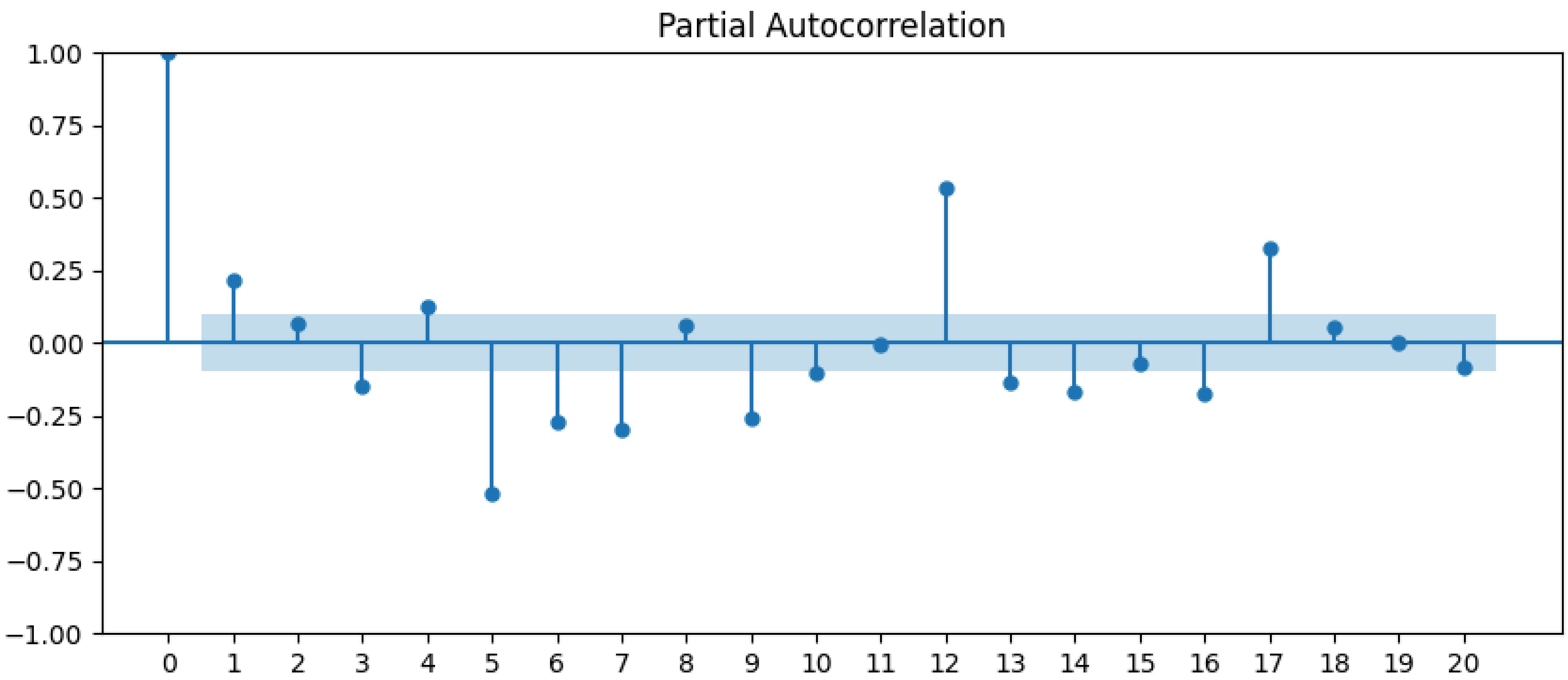
$$PACF(k) = Corr(e_t, e_{t-k})$$

In [ ]:
```python
def pacf(data, k):
    if k == 0:
        pacf_val = 1
    else:
        gamma_array = np.array([acf(data, k) for k in range(1, k + 1)])

        gamma_matrix = []
        for i in range(k):
```

```
            temp = [0] * k
            temp[i:] = [acf(data, j) for j in range(k - i)]     # making diagonal
            gamma_matrix.append(temp)

        gamma_matrix = np.array(gamma_matrix)
        gamma_matrix = gamma_matrix + gamma_matrix.T - np.diag(gamma_matrix.diagonal())     # making symmetric matrix
        pacf_val = np.linalg.inv(gamma_matrix).dot(gamma_array)[-1]
    return pacf_val
```

In [42]:

```
import numpy as np
fig, ax = plt.subplots(figsize=(10, 4))
plot_pacf(diff_data,lags=20, alpha=0.5, ax=ax)#절단점 직전에 값이 의미가 있음
ax.set_xticks(np.arange(0, 21, 1))  # 1 lag 단위로 tick

plt.show()
```



ACF(Autocorrelation function)

ACF는 k시간 단위로 구분된 시계열 관측치 간의 y_t와 y_{t+k}간 상관 관계를 측정하는 것이다. ACF의 반환값의 절대값이 커질수록 시차 시계열 데이터의 상관성이 크다고 할 수 있다. 이 크다는 것의 기준은 p-value와 같이 95% 근사 신뢰구간으로 정할 수 있다. 또는 $\pm 1.96 * \frac{SE_k}{\sqrt{n}}$를 신뢰구간으로 적용할 수 있는데 이때,

$SE_k = \sqrt{1 + 2\Sigma_{j=1}^{k} |\hat{\gamma_j^2}|}$이다.

ACF는 정상성 데이터에 대해서는 0으로 빠르게 떨어지고 비정상성 데이터에는 천천히 수치가 떨어진다.

시차(lag)에 대한 ACF수식과 python 코드로 구현한 내용은 아래와 같다.

$$\mathrm{ACF}(k) = \frac{\mathrm{cov}(y_t, y_{t+k})}{\mathrm{var}(y_t)} = \frac{\sum_{t=1}^{N-k}(y_t - \bar{y})(y_{t+k} - \bar{y})}{\sum_{t=1}^{N}(y_t - \bar{y})^2} \frac{N}{N-k}, \quad k = 0, 1, \ldots, N-1$$

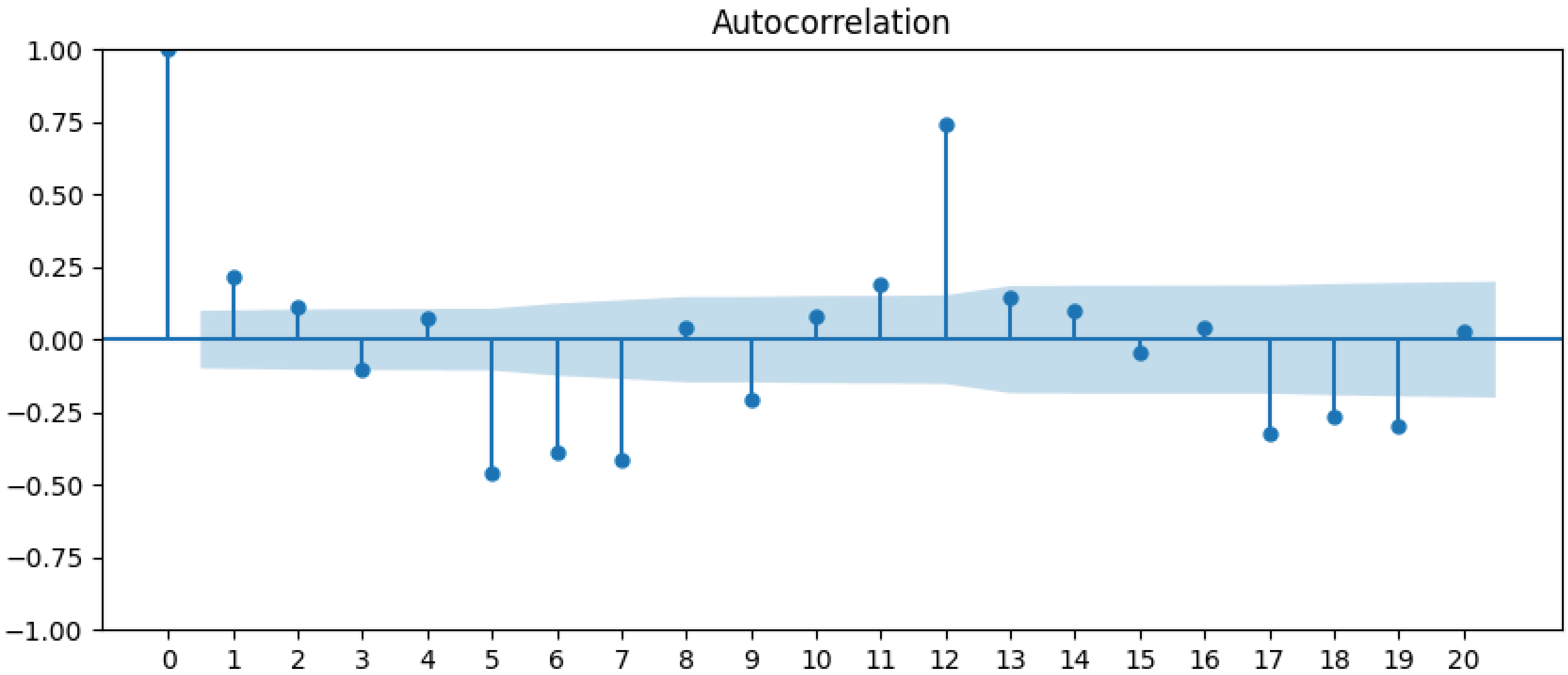
In [ ]:

```
def acf(data, k):
    data = np.array(data).reshape(-1)
    mean = data.mean()

    numerator = np.sum((data[:len(data)-k] - mean) * (data[k:] - mean))
    denominator = np.sum(np.square(data - mean))

    acf_val = numerator / denominator

    return acf_val

# [1.0, 0.1048904805897309, -0.0021663171476797435, 0.025054864781747933, -0.016516709724606914,
# -0.038303324302299956, -0.025356924583006032, 0.005668275317742138, -0.03779181516554617,
# -0.03389716988734305, -0.04605595433803343, 0.09043427594152544, -0.04105851291659383,
# -0.03529945852644305, 0.03141655496394635, 0.03049954206267939 4]
```

In [43]:

```
import numpy as np
fig, ax = plt.subplots(figsize=(10, 4))
plot_acf(diff_data,lags=20, alpha=0.5, ax=ax)#절단점 직전에 값이 의미가 있음
ax.set_xticks(np.arange(0, 21, 1))  # 1 lag 단위로 tick
```

```
plt.show()
```



In [44]:
```
training
```

Out[44]:

day	price
2013-01-01	3794
2013-02-01	3863
2013-03-01	5190
2013-04-01	5783
2013-05-01	6298
2013-06-01	6739
2013-07-01	7107
2013-08-01	6492
2013-09-01	5693
2013-10-01	5497
2013-11-01	5958
2013-12-01	4309
2014-01-01	4323
2014-02-01	4391
2014-03-01	5860
2014-04-01	6552
2014-05-01	7370
2014-06-01	7888
2014-07-01	8330
2014-08-01	7401
2014-09-01	6599
2014-10-01	6411
2014-11-01	6742
2014-12-01	4910
2015-01-01	4626

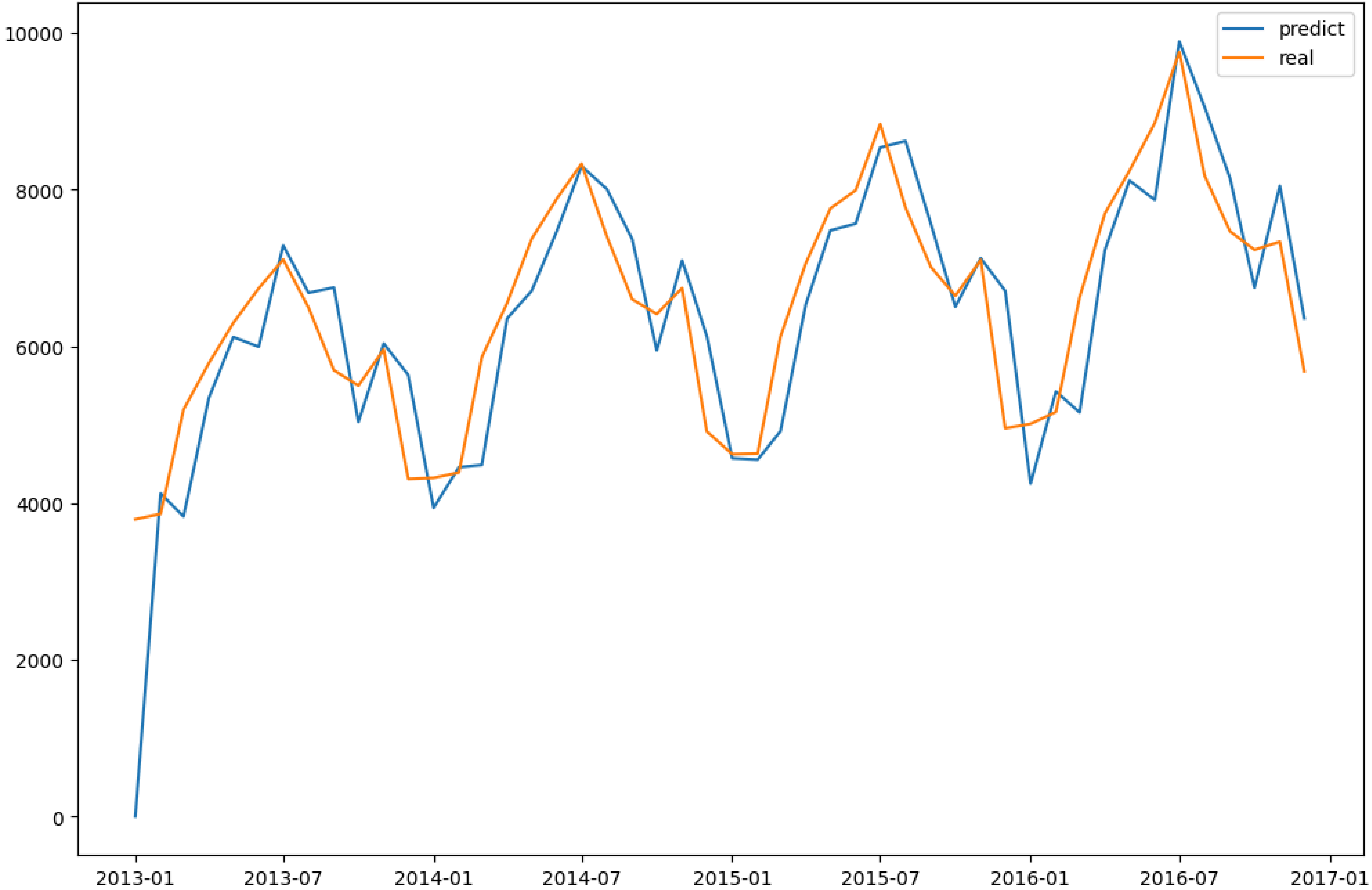
	price
day	
2015-02-01	4633
2015-03-01	6121
2015-04-01	7060
2015-05-01	7758
2015-06-01	7992
2015-07-01	8838
2015-08-01	7770
2015-09-01	7011
2015-10-01	6644
2015-11-01	7102
2015-12-01	4952
2016-01-01	5008
2016-02-01	5160
2016-03-01	6624
2016-04-01	7697
2016-05-01	8240
2016-06-01	8851
2016-07-01	9758
2016-08-01	8176
2016-09-01	7467
2016-10-01	7231
2016-11-01	7334
2016-12-01	5678

In [ ]:

```
from statsmodels.tsa.arima.model import ARIMA
model=ARIMA(training,order=(2,1,2))
```

```
res=model.fit()
print(res.summary())
plt.plot(res.predict())
# forecast = model_fit.forecast(steps=24*7)
#arma_forecast = model_fit.predict(start='2023-01-01', end='2023-12-31')

plt.plot(training)
plt.legend(['predict','real'])
plt.show()
```

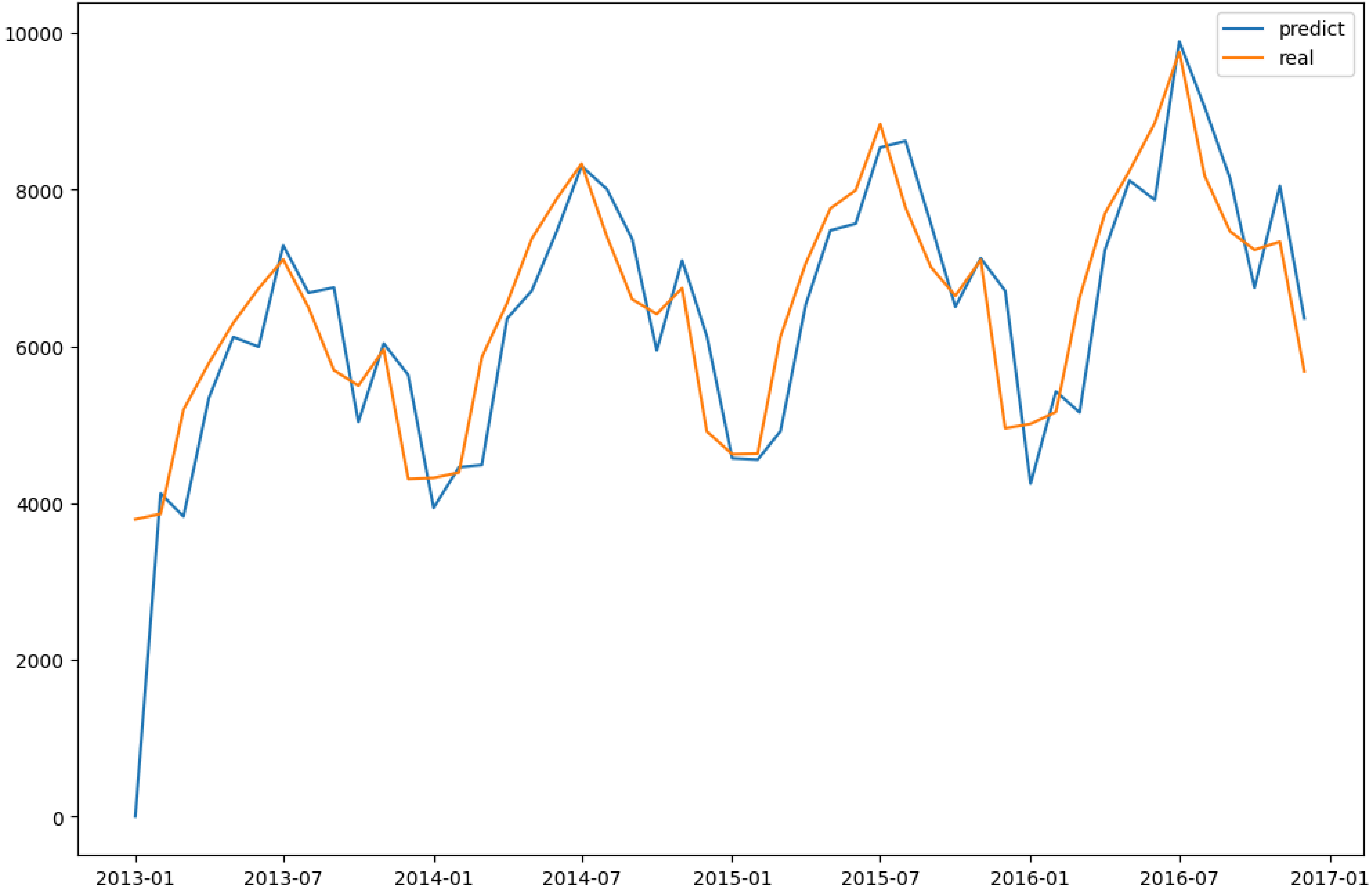
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  price   No. Observations:                   48
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -375.875
Date:                Wed, 08 Oct 2025   AIC                            761.750
Time:                        23:37:03   BIC                            771.001
Sample:                    01-01-2013   HQIC                           765.231
                         - 12-01-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.3167      0.190     -6.920      0.000      -1.690      -0.944
ar.L2         -0.3191      0.191     -1.670      0.095      -0.694       0.055
ma.L1          1.9692      0.244      8.074      0.000       1.491       2.447
ma.L2          0.9941      0.243      4.098      0.000       0.519       1.470
sigma2      4.455e+05   1.15e-06   3.88e+11      0.000    4.45e+05    4.45e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.11   Jarque-Bera (JB):                 0.38
Prob(Q):                              0.74   Prob(JB):                         0.83
Heteroskedasticity (H):               1.49   Skew:                            -0.21
Prob(H) (two-sided):                  0.44   Kurtosis:                         2.88
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 3.61e+27. Standard errors may be unstable.
```



In [46]:
```
from statsmodels.tsa.arima.model import ARIMA
model=ARIMA(training,order=(2,1,2))
res=model.fit()
print(res.summary())
plt.plot(res.predict())
plt.plot(training)
```

```
plt.legend(['predict','real'])
plt.show()
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  price   No. Observations:                   48
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -375.875
Date:                Wed, 08 Oct 2025   AIC                            761.750
Time:                        23:37:03   BIC                            771.001
Sample:                    01-01-2013   HQIC                           765.231
                         - 12-01-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.3167      0.190     -6.920      0.000      -1.690      -0.944
ar.L2         -0.3191      0.191     -1.670      0.095      -0.694       0.055
ma.L1          1.9692      0.244      8.074      0.000       1.491       2.447
ma.L2          0.9941      0.243      4.098      0.000       0.519       1.470
sigma2      4.455e+05   1.15e-06   3.88e+11      0.000    4.45e+05    4.45e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.11   Jarque-Bera (JB):                 0.38
Prob(Q):                              0.74   Prob(JB):                         0.83
Heteroskedasticity (H):               1.49   Skew:                            -0.21
Prob(H) (two-sided):                  0.44   Kurtosis:                         2.88
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 3.61e+27. Standard errors may be unstable.
```

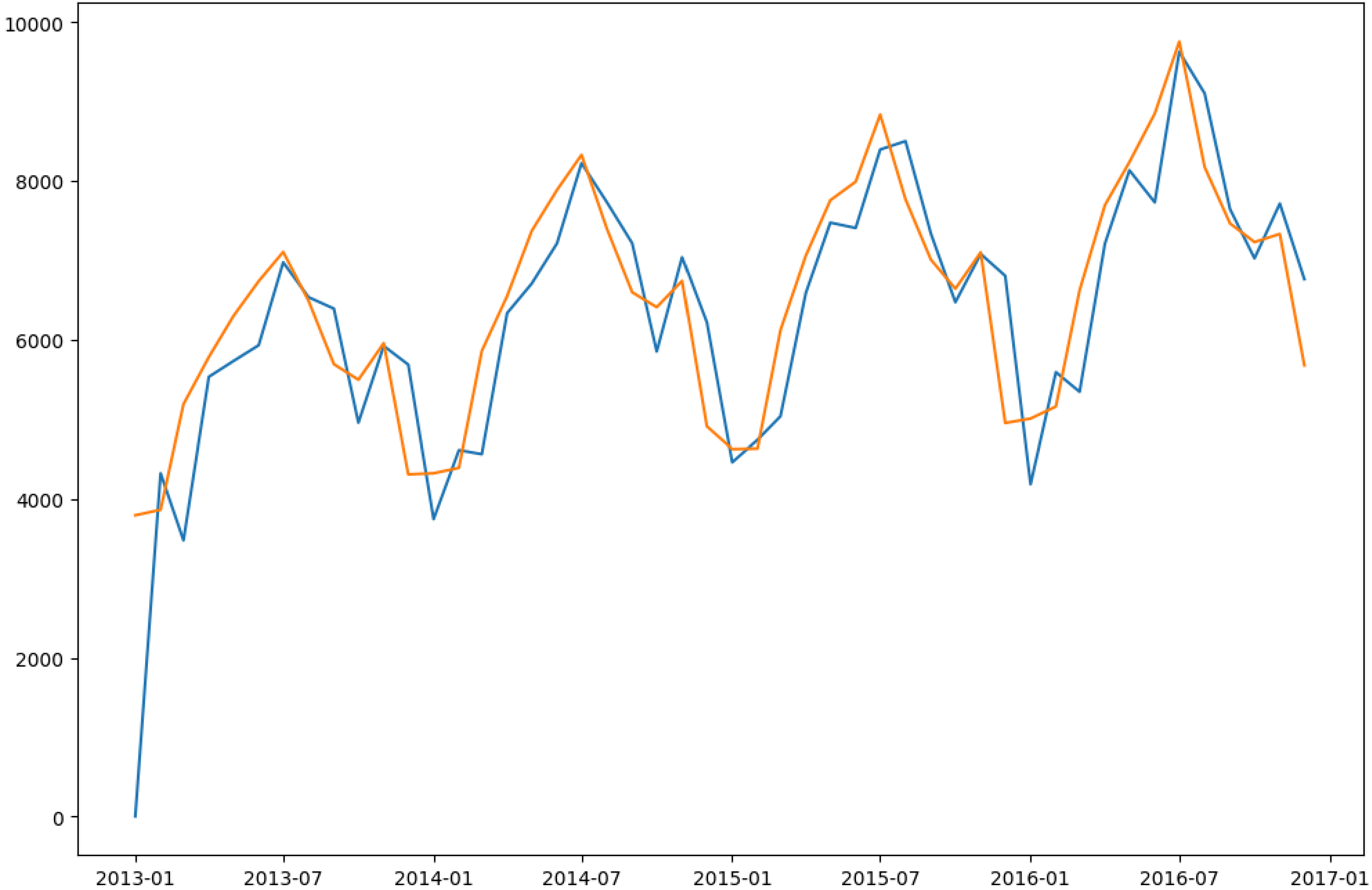


In [47]:

```
from statsmodels.tsa.arima.model import ARIMA
model=ARIMA(training,order=(2,1,3))
res=model.fit()
print(res.summary())
plt.plot(res.predict())
```

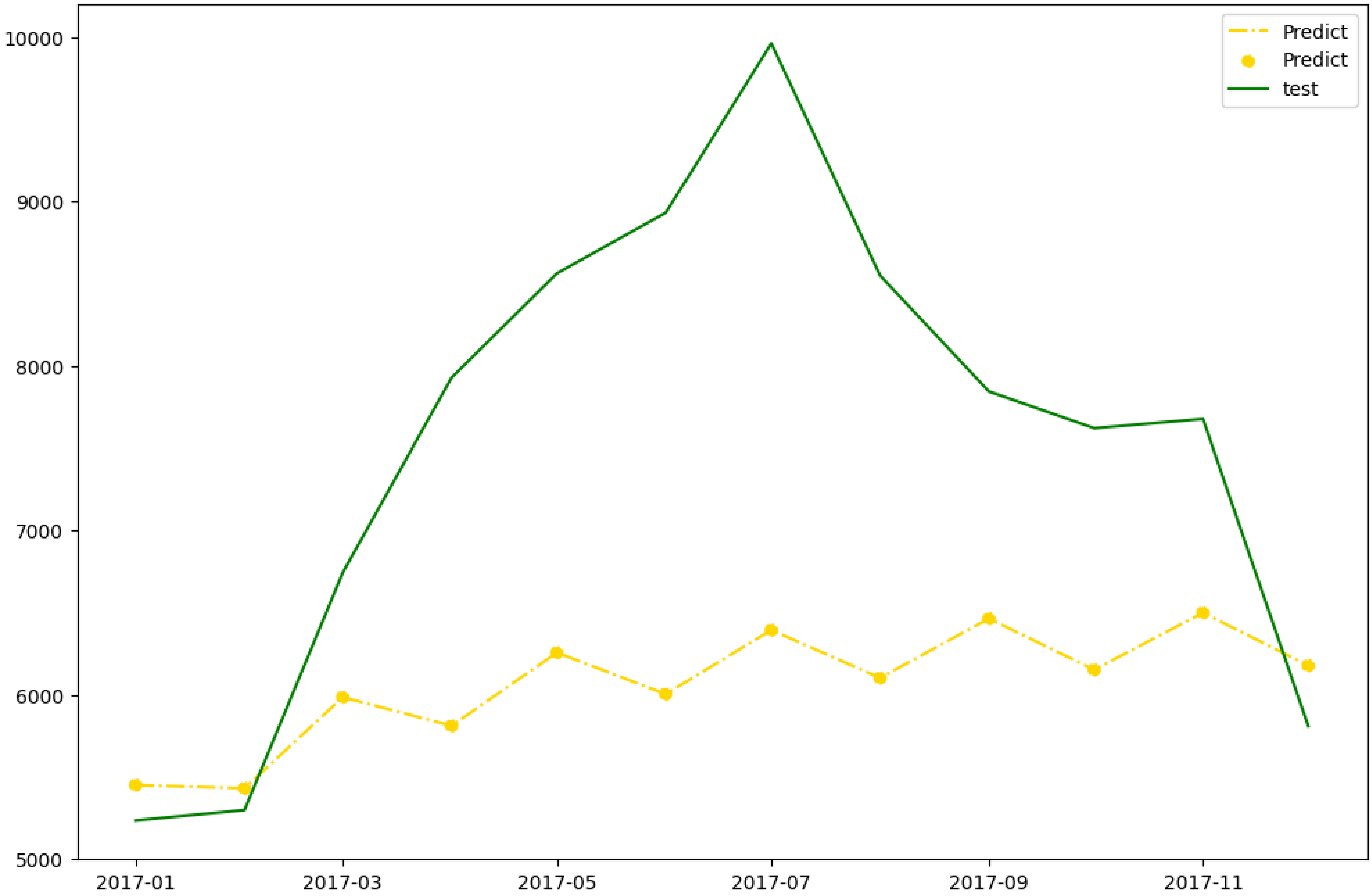
```python
plt.plot(training)
plt.show()
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:                  price   No. Observations:                   48
Model:                 ARIMA(2, 1, 3)   Log Likelihood                -376.749
Date:                Wed, 08 Oct 2025   AIC                            765.498
Time:                        23:37:04   BIC                            776.599
Sample:                    01-01-2013   HQIC                           769.676
                         - 12-01-2016                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.2876      0.452     -0.636      0.524      -1.173       0.598
ar.L2          0.7124      0.452      1.578      0.115      -0.173       1.598
ma.L1          0.8864      0.588      1.507      0.132      -0.267       2.040
ma.L2         -0.7799      0.582     -1.340      0.180      -1.920       0.360
ma.L3         -0.6676      0.233     -2.870      0.004      -1.124      -0.212
sigma2      5.727e+05   7.63e-07   7.51e+11      0.000    5.73e+05    5.73e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.14   Jarque-Bera (JB):                 1.56
Prob(Q):                              0.71   Prob(JB):                         0.46
Heteroskedasticity (H):               1.25   Skew:                            -0.43
Prob(H) (two-sided):                  0.66   Kurtosis:                         3.21
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 5.46e+28. Standard errors may be unstable.
```



In [48]:
```
forecast_data=res.forecast(steps=len(test),alpha=0.05)
pred_y=forecast_data
test_y=test

plt.plot(pred_y,color="gold",linestyle='dashdot',label="Predict")
plt.scatter(pred_y.index,pred_y,color="gold",linestyle='dashdot',label="Predict")
```

```python
plt.plot(test_y,color="green",label="test")
plt.legend()
plt.show()
```



In [49]:
```python
from pmdarima import auto_arima
```

```python
auto_model= auto_arima(training, step_p=0, d=1,start_q=0,max_p=3,max_q=3,start_P=0,start_Q=0,max_P=3,max_Q=3,m=12,seasonal=True
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,0)(0,1,0)[12]             : AIC=482.616, Time=0.03 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=481.846, Time=0.01 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=482.652, Time=0.32 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=482.466, Time=0.08 sec
 ARIMA(0,1,0)(1,1,0)[12]             : AIC=483.637, Time=0.05 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=483.669, Time=0.02 sec
 ARIMA(0,1,0)(1,1,1)[12]             : AIC=inf, Time=0.26 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=481.031, Time=0.02 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=482.740, Time=0.08 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=inf, Time=0.35 sec
 ARIMA(1,1,1)(0,1,0)[12]             : AIC=482.682, Time=0.08 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=480.687, Time=0.02 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=482.403, Time=0.06 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=inf, Time=0.29 sec
 ARIMA(0,1,2)(0,1,0)[12]             : AIC=482.683, Time=0.04 sec
 ARIMA(1,1,2)(0,1,0)[12]             : AIC=inf, Time=0.05 sec
 ARIMA(0,1,1)(0,1,0)[12] intercept   : AIC=482.687, Time=0.02 sec

Best model:  ARIMA(0,1,1)(0,1,0)[12]
Total fit time: 1.800 seconds
```

In [50]:

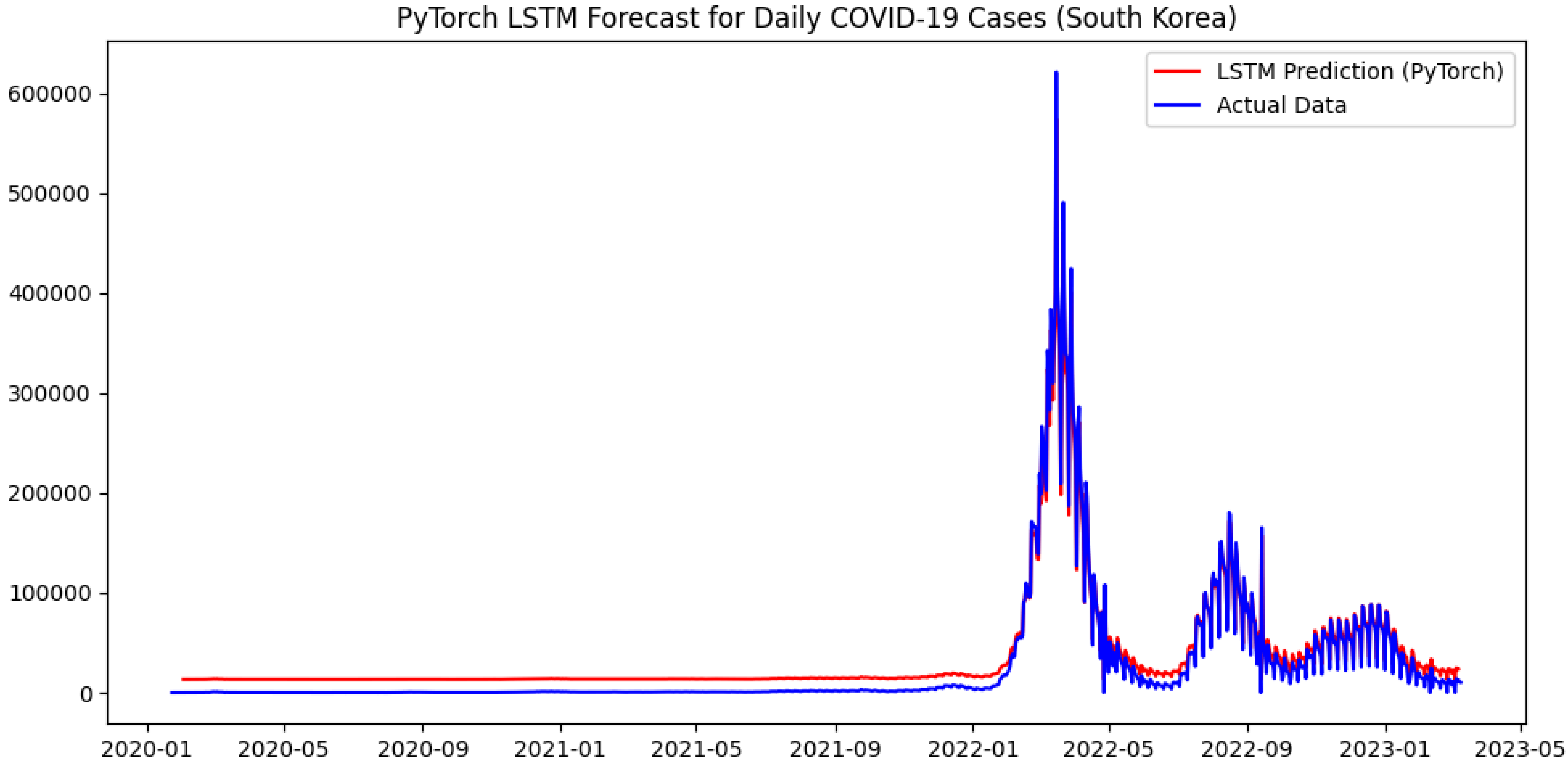
```python
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# ========================================
# 1️⃣ 데이터 로드 및 전처리
# ========================================
url = "https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/jhu/new_cases.csv"
df = pd.read_csv(url)

korea_data = df[['date', 'South Korea']].dropna()
korea_data['date'] = pd.to_datetime(korea_data['date'])
korea_data.set_index('date', inplace=True)
offset=10
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(korea_data['South Korea'].values[offset:].reshape(-1, 1))

# ========================================
# 2️⃣ LSTM 입력 데이터 구성 함수
```

```python
# ========================================
def create_dataset(data, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        a = data[i:(i + time_step), 0]
        X.append(a)
        y.append(data[i + time_step, 0])
    return np.array(X), np.array(y)

time_step = 1
X, y = create_dataset(scaled_data, time_step)

# reshape to (samples, time_step, features)
X = X.reshape(X.shape[0], X.shape[1], 1)

# numpy → torch tensor
X_tensor = torch.FloatTensor(X)
y_tensor = torch.FloatTensor(y).view(-1, 1)

# ========================================
# 3️⃣ PyTorch LSTM 모델 정의
# ========================================
class LSTMModel(nn.Module):
    def __init__(self, input_size=1, hidden_size=150, num_layers=2, output_size=1):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        # LSTM
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)

        # fully connected layers
        self.fc1 = nn.Linear(hidden_size, 125)
        self.fc2 = nn.Linear(125, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
        out, _ = self.lstm(x, (h0, c0))
        out = out[:, -1, :]   # 마지막 시점(hidden state)
        out = self.fc1(out)
        out = self.fc2(out)
        return out

model = LSTMModel(input_size=1, hidden_size=50, num_layers=2, output_size=1)
```

```python
# ========================================
# 4️⃣ 손실함수 & 옵티마이저
# ========================================
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# ========================================
# 5️⃣ 학습 루프
# ========================================
epochs = 100
batch_size = 50
n = X_tensor.shape[0]

for epoch in range(epochs):
    model.train()
    epoch_loss = 0.0

    for i in range(0, n, batch_size):
        X_batch = X_tensor[i:i+batch_size]
        y_batch = y_tensor[i:i+batch_size]

        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    if epoch%10==0:
        print(f"Epoch [{epoch+1}/{epochs}] | Loss: {epoch_loss/n:.6f}")

# ========================================
# 6️⃣ 예측
# ========================================
model.eval()
with torch.no_grad():
    predictions = model(X_tensor).detach().numpy()

# 역정규화
predictions = scaler.inverse_transform(predictions)
actual = korea_data['South Korea'].values

# ========================================
# 7️⃣ 시각화
# ========================================
plt.figure(figsize=(10, 5))
plt.plot(korea_data.index[offset:offset+len(predictions)], predictions, label='LSTM Prediction (PyTorch)', color='red')
```

```python
plt.plot(korea_data.index, actual, label='Actual Data', color='blue')
plt.legend()
plt.title('PyTorch LSTM Forecast for Daily COVID-19 Cases (South Korea)')
plt.tight_layout()
plt.show()
```

```
Epoch [1/100] | Loss: 0.000254
Epoch [11/100] | Loss: 0.000209
Epoch [21/100] | Loss: 0.000029
Epoch [31/100] | Loss: 0.000029
Epoch [41/100] | Loss: 0.000029
Epoch [51/100] | Loss: 0.000029
Epoch [61/100] | Loss: 0.000029
Epoch [71/100] | Loss: 0.000029
Epoch [81/100] | Loss: 0.000029
Epoch [91/100] | Loss: 0.000029
```



In [ ]: