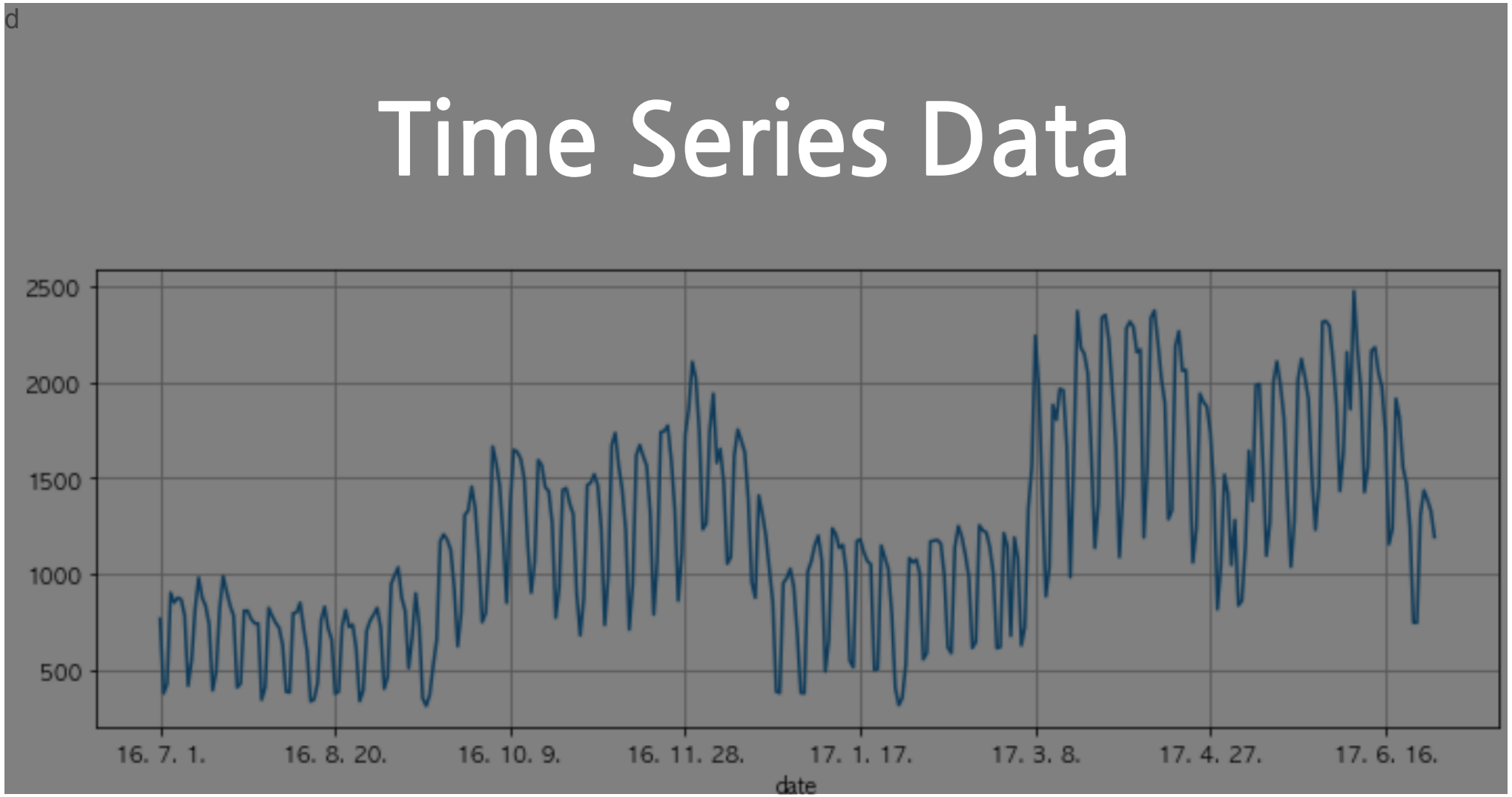
zero-base /

Life Changing Education

Project 05. forecast(시계열 분석)

배경 및 설치

Time Series Data



시계열 데이터란?

시간의 흐름에 대해 특정 패턴과 같은
정보를 가지고 있는 경우를
시계열 데이터라고 함

Time Series Data

통계학에서 시계열 데이터라는 것 만으로도 많은 학습량을 가지고 있다.

보통 데이터 분석에서

개요 정도의 레벨에서 접근하는 시계열은

대부분 Forecast이다…

그리고 시계열 데이터에서

주기성을 가지고 있는 데이터를 다루는 경우를

Seasonal Time Series라고 한다.

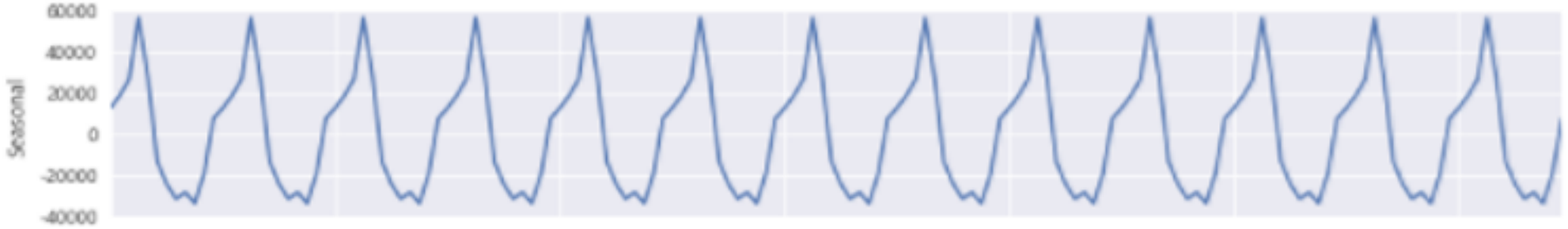
Time Series Data

- 만약 이런 데이터를 분석 한다면,

Time Series Data

- 먼저 트렌드를 찾고

Time Series Data



- 트렌트를 뺀 주기적 특성을 찾는다

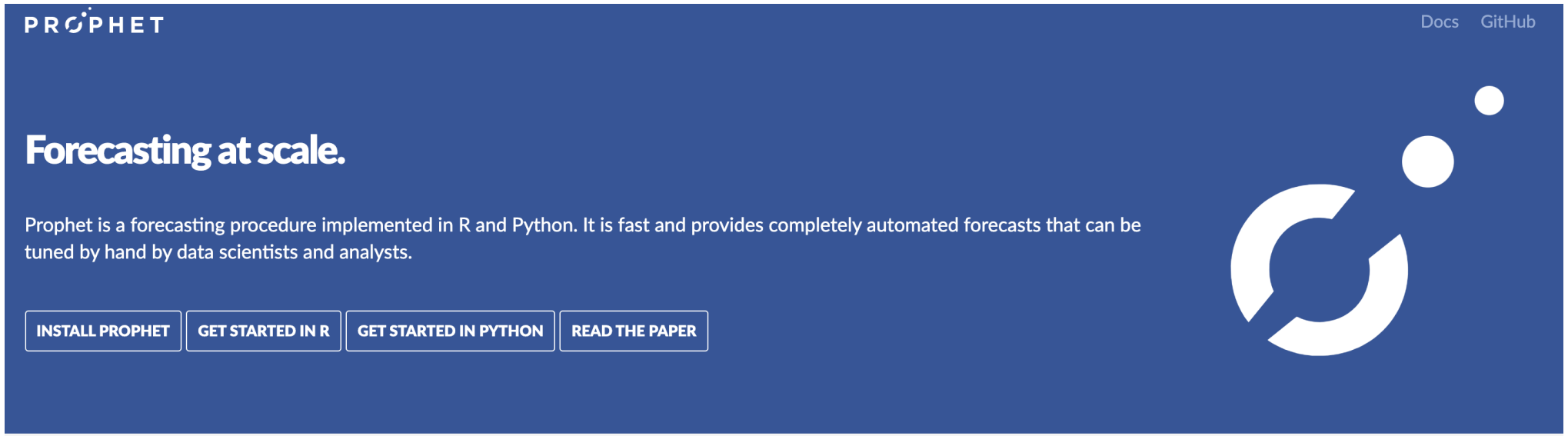
if

통계적 지식이나 이론에 대해 이야기할 것이 아니라
그저 목표로 하는 데이터가 하필 시계열 데이터였고…
하필 단일 함수로 쉽게 접근할 수 없어서…
Python이라는 도구로 Forecast를 하고 싶을 뿐이라면…

Seasonal Time Series에 대한 좋은 모듈이
최근 발표 되었다….
그것도… Facebook 애들이 만들었다…

fb-Prophet

fbprophet install



Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

Prophet is open source software released by Facebook's Core Data Science team. It is available for download on CRAN and PyPI.

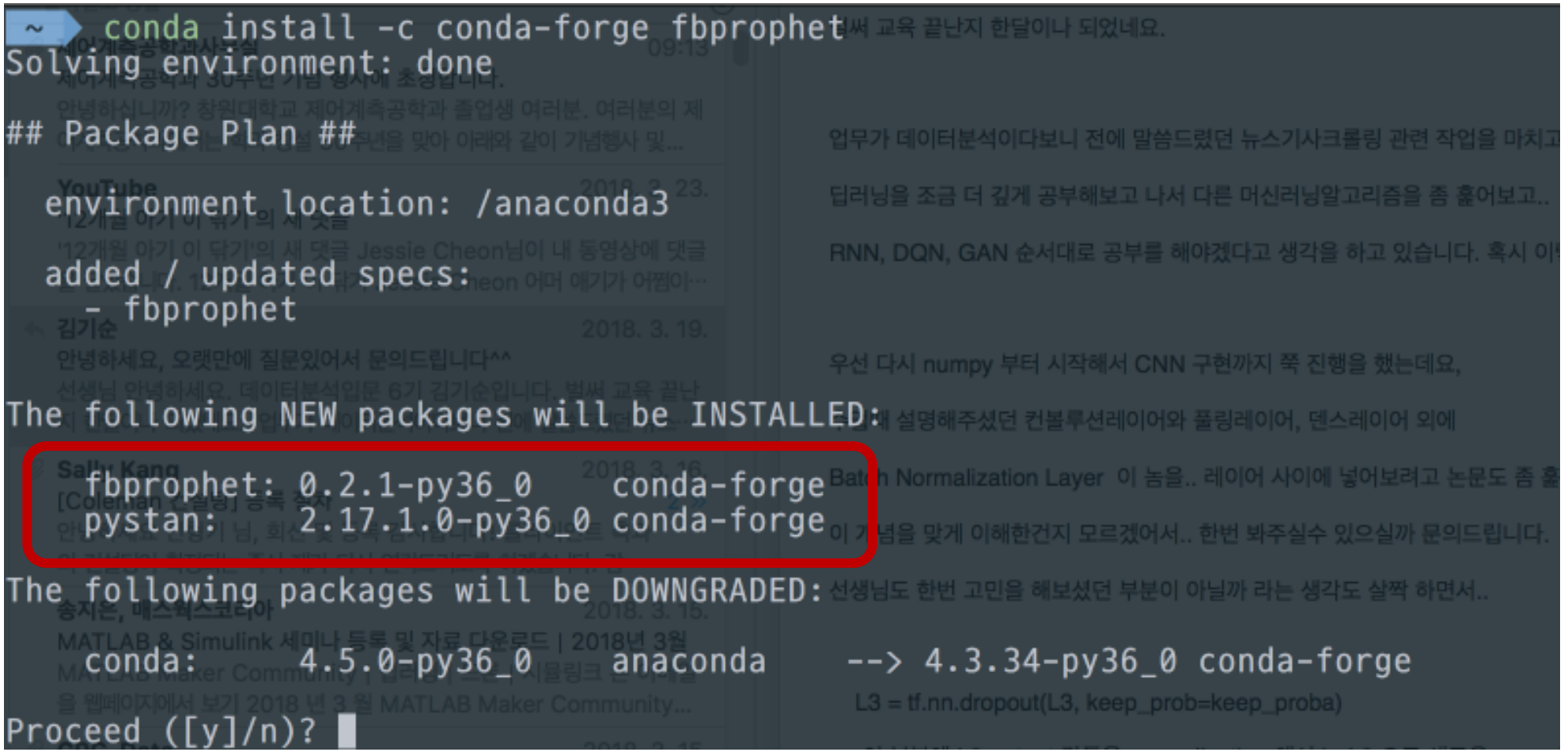
Accurate and fast. **Fully automatic.** **Tunable forecasts.** **Available in R or Python.**

출처: https://facebook.github.io/prophet/

fbprophet install

- 윈도우 유저는 Visual C++ Build Tool을 먼저 설치해야 한다.
- https://go.microsoft.com/fwlink/?LinkId=691126
- conda install pandas-datareader

fbprophet install



- conda install -c conda-forge fbprophet

def의 기초

본 주제에 들어가기 전에...

오늘은 사전에 함수(def)에 대해 학습하고 진행하겠습니다.

Python 함수(def)


```
In [1]: def test_def(a, b):
            return a + b

In [2]: test_def(2, 3)

Out[2]: 5
```

- 가장 기초적인 모양의 def 정의
- 이름(test_df)과 입력 인자(a, b)를 정해주고
- 출력(return)을 작성하는 모양

Python 함수(def)

In [3]:
```
a = 1

def edit_a(i):
    global a
    a = i
```

In [4]:
```
edit_a(2)
print(a)
```

```
2
```

- global 변수를 def 내에서 사용하고 싶다면 global로 선언할 것

Python 함수(def)


```
In [5]: def edit_a(i):
            a = i

In [6]: edit_a(3)
        print(a)

        2
```

- def 내에서의 변수와 밖에서의 변수는 같은 이름이어도 같은 것이 아니다

조금 더 고급지게...

$$y = a\sin\left(2\pi ft + t_0\right) + b$$

이 함수의 그래프를 그려주는 함수를 만들자…

Python 함수(def)

In [7]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

- 필요한 모듈 import 하고 ~

Python 함수(def)

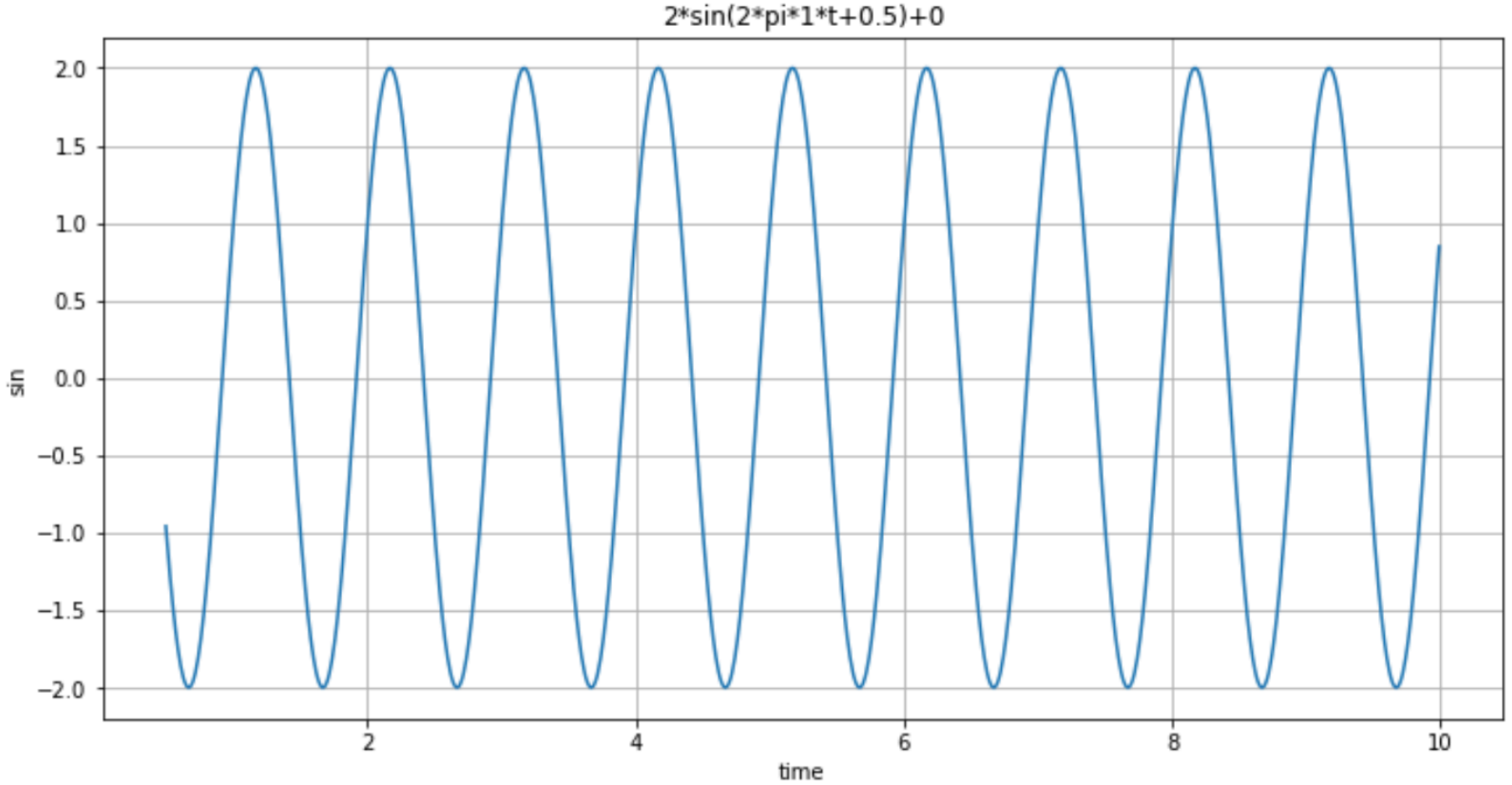
In [8]:
```
def plotSinWave(amp, freq, endTime, sampleTime, startTime, bias):
    """
    plot sine wave
    y = a sin(2 pi f t + t_0) + b
    """
    time = np.arange(startTime, endTime, sampleTime)
    result = amp * np.sin(2 * np.pi * freq * time + startTime) + bias

    plt.figure(figsize=(12, 6))
    plt.plot(time, result)
    plt.grid(True)
    plt.xlabel("time")
    plt.ylabel("sin")
    plt.title(
        str(amp) + "*sin(2*pi*" + str(freq) + "*t+" + str(startTime) + ")+" + str(bias)
    )
    plt.show()
```

- sin 함수를 구성하기 위해 필요한 인자들을 def의 입력변수로 받고…

Python 함수(def)


```
In [9]: plotSinWave(2, 1, 10, 0.01, 0.5, 0)
```



불편하다…

인자의 이름도… 순서도…

이건 python이 아니야~~~

Python 함수(def)

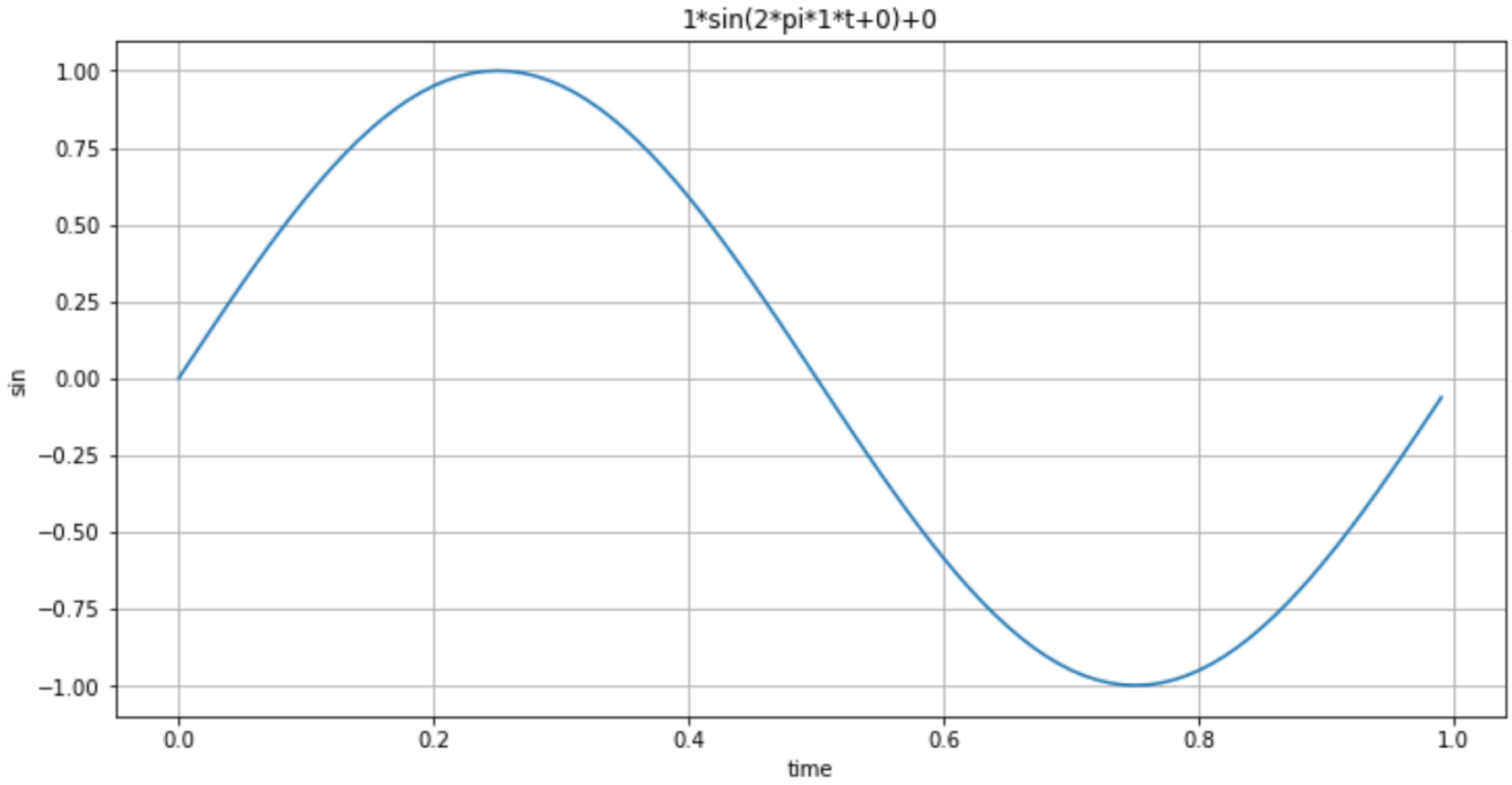
In [10]:
```
def plotSinWave(**kwargs):
    """
    plot sine wave
    y = a sin(2 pi f t + t_0) + b
    """
    endTime = kwargs.get("endTime", 1)
    sampleTime = kwargs.get("sampleTime", 0.01)
    amp = kwargs.get("amp", 1)
    freq = kwargs.get("freq", 1)
    startTime = kwargs.get("startTime", 0)
    bias = kwargs.get("bias", 0)
    figsize = kwargs.get("figsize", (12, 6))

    time = np.arange(startTime, endTime, sampleTime)
    result = amp * np.sin(2 * np.pi * freq * time + startTime) + bias

    plt.figure(figsize=figsize)
    plt.plot(time, result)
    plt.grid(True)
    plt.xlabel("time")
    plt.ylabel("sin")
    plt.title(
        str(amp) + "*sin(2*pi*" + str(freq) + "*t+" + str(startTime) + ")+" + str(bias)
    )
    plt.show()
```

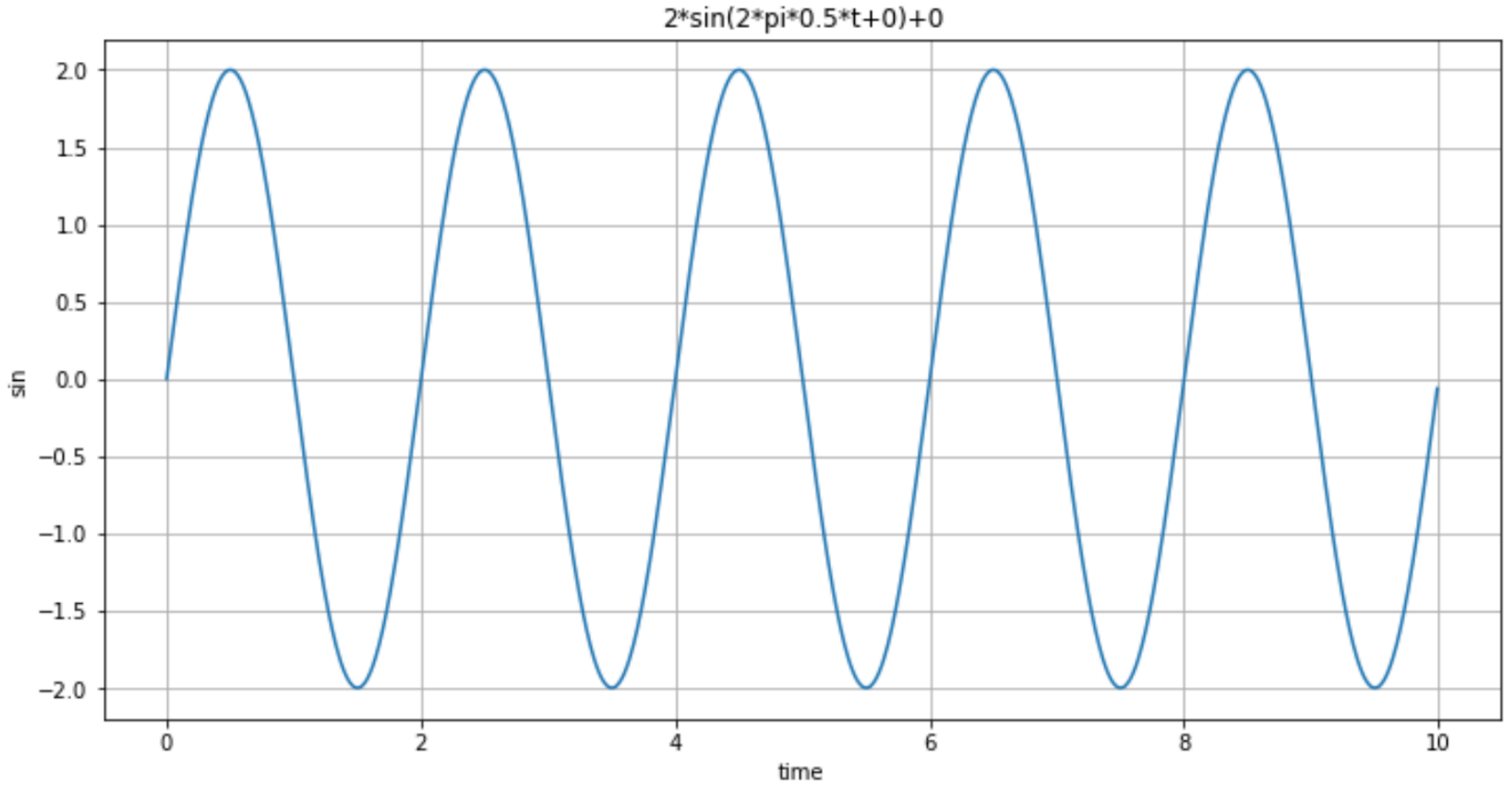
Python 함수(def)


```
In [11]: plotSinWave()
```



Python 함수(def)


```
In [12]: plotSinWave(amp=2, freq=0.5, endTime=10)
```



내가 만든 함수를 import 해보자

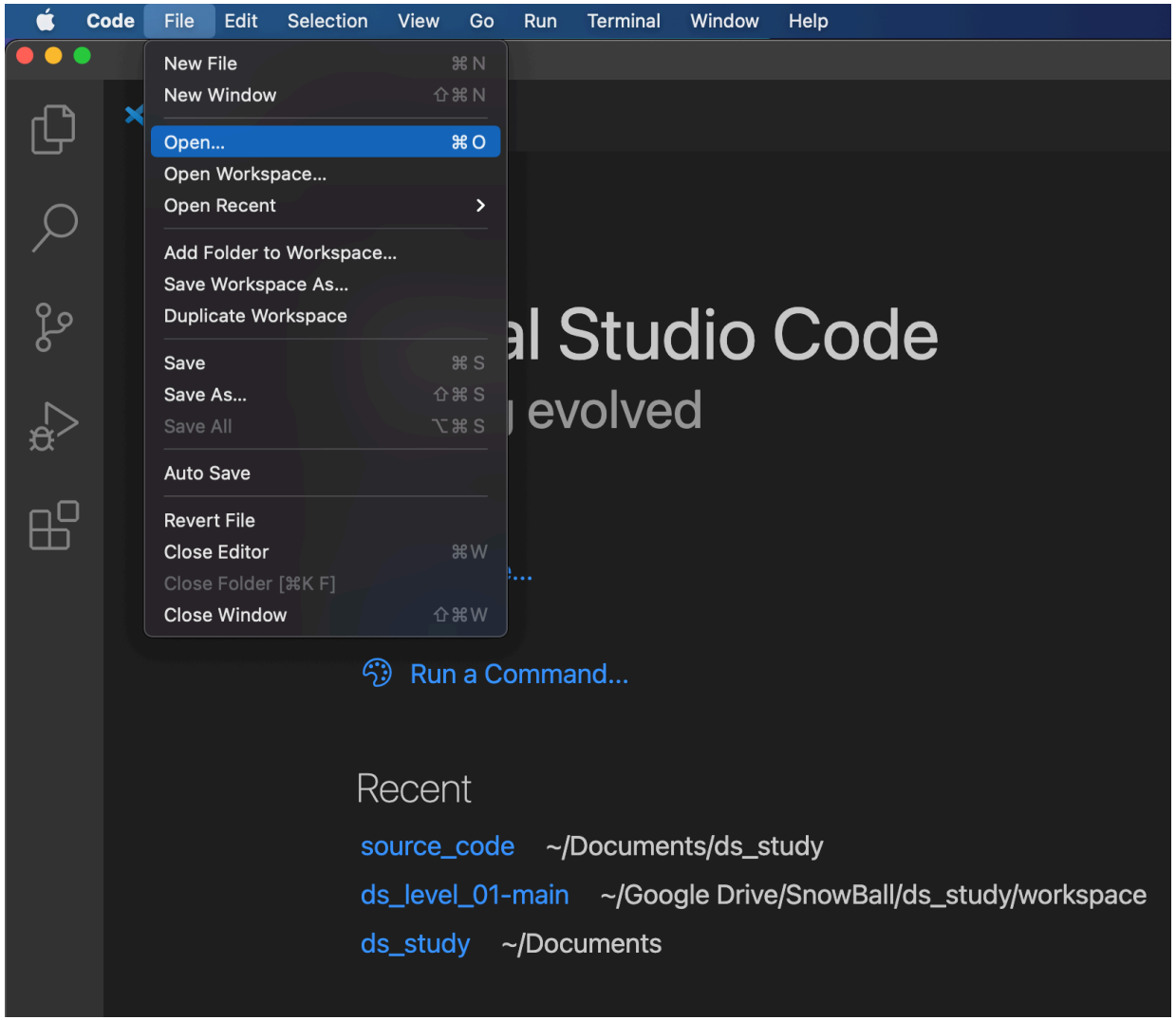
여기서 아예 module로 만들어서 import 할 수 있도록 할까요???

.py 파일을 만들어야 합니다.

음.. 어떤 에디터이든 사용해야 합니다.

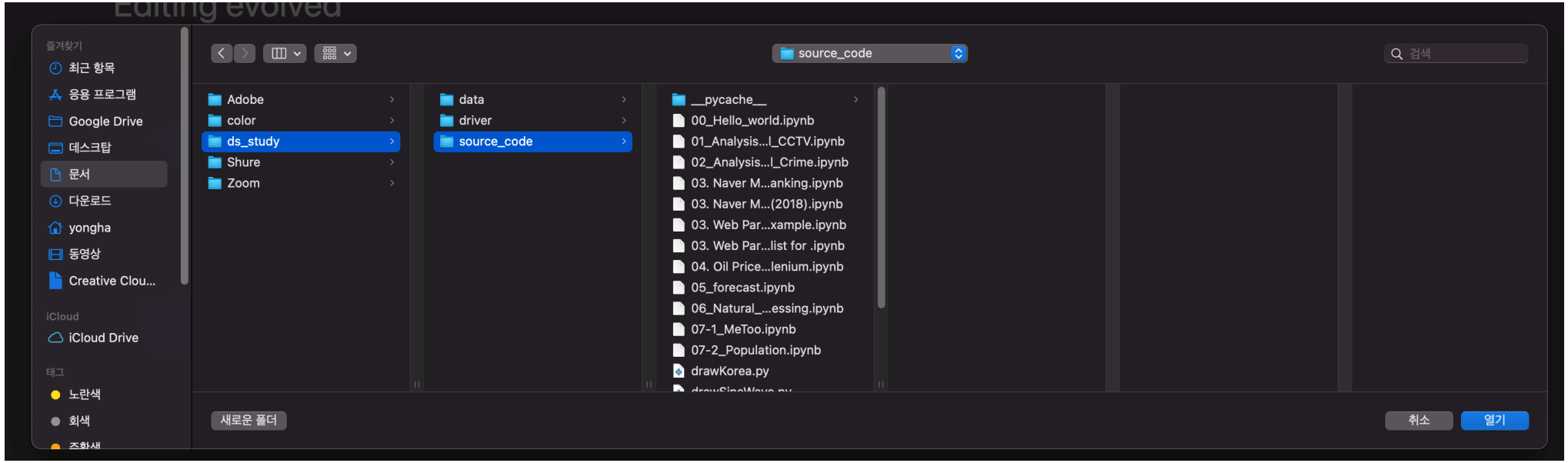
저희는 vscode 🤗

Python 모듈(module)로 만들어서 import 하기



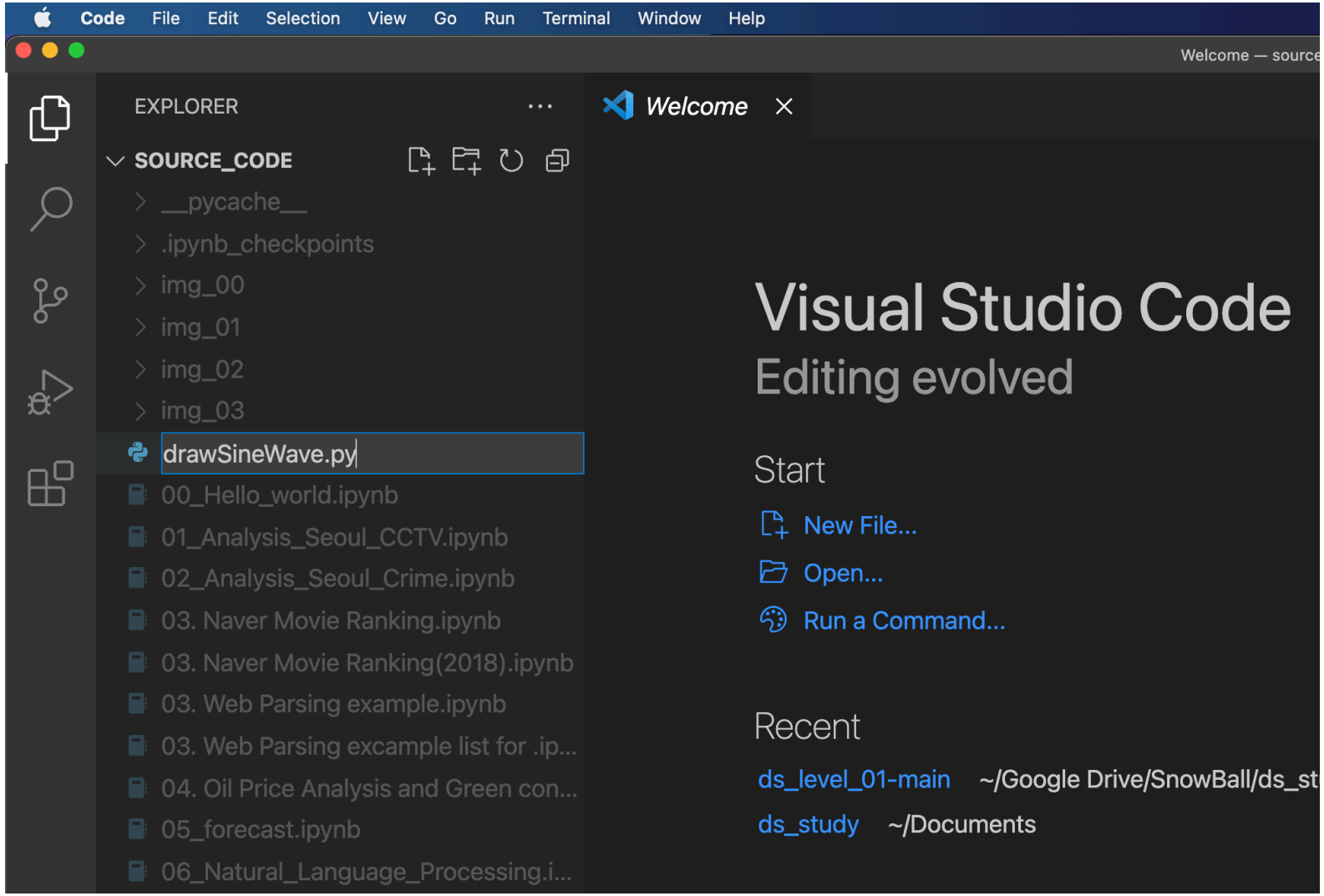
- vscode 실행 후 File - Open 선택

Python 모듈(module)로 만들어서 import 하기



- 문서 - ds_study - source_code
- 열기 선택

Python 모듈(module)로 만들어서 import 하기



- drawSineWave.py 파일 생성

Python 모듈(module)로 만들어서 import 하기

EXPLORER

SOURCE_CODE

- __pycache__
- .ipynb_checkpoints
- img_00
- img_01
- img_02
- img_03
- 00_Hello_world.ipynb
- 01_Analysis_Seoul_CCTV.ipynb
- 02_Analysis_Seoul_Crime.ipynb
- 03. Naver Movie Ranking.ipynb
- 03. Naver Movie Ranking(2018).ipynb
- 03. Web Parsing example.ipynb
- 03. Web Parsing excample list for .i...
- 04. Oil Price Analysis and Green co...
- 05_forecast.ipynb
- 06_Natural_Language_Processing.i...
- 07-1_MeToo.ipynb
- 07-2_Population.ipynb
- drawKorea.py
- drawSineWave.py 2
- set_matplotlib_hangul.py

drawSineWave.py 2

drawSineWave.py

```python
import numpy as np
import matplotlib.pyplot as plt


def plotSinWave(**kwargs):
    """
    plot sine wave
    y = a sin(2 pi f t + t_0) + b
    """
    endTime = kwargs.get("endTime", 1)
    sampleTime = kwargs.get("sampleTime", 0.01)
    amp = kwargs.get("amp", 1)
    freq = kwargs.get("freq", 1)
    startTime = kwargs.get("startTime", 0)
    bias = kwargs.get("bias", 0)
    figsize = kwargs.get("figsize", (12, 6))

    time = np.arange(startTime, endTime, sampleTime)
    result = amp * np.sin(2 * np.pi * freq * time + startTime) + bias

    plt.figure(figsize=figsize)
    plt.plot(time, result)
    plt.grid(True)
    plt.xlabel("time")
    plt.ylabel("sin")
    plt.title(
        str(amp) + "*sin(2*pi*" + str(freq) + "*t+" + str(startTime) + ")+" + str(bias)
    )
    plt.show()
```

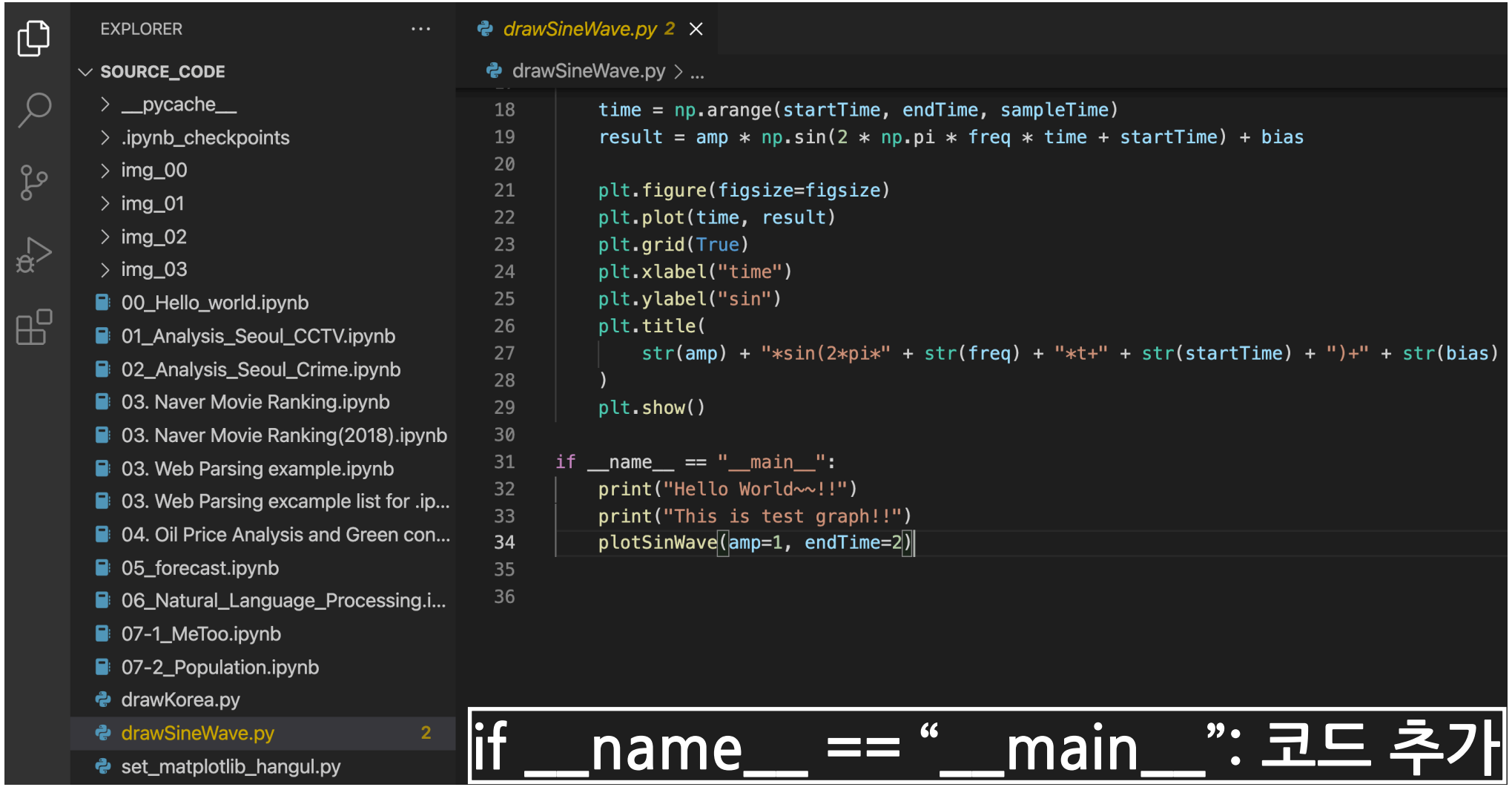
Python 모듈(module)로 만들어서 import 하기

zero-base /

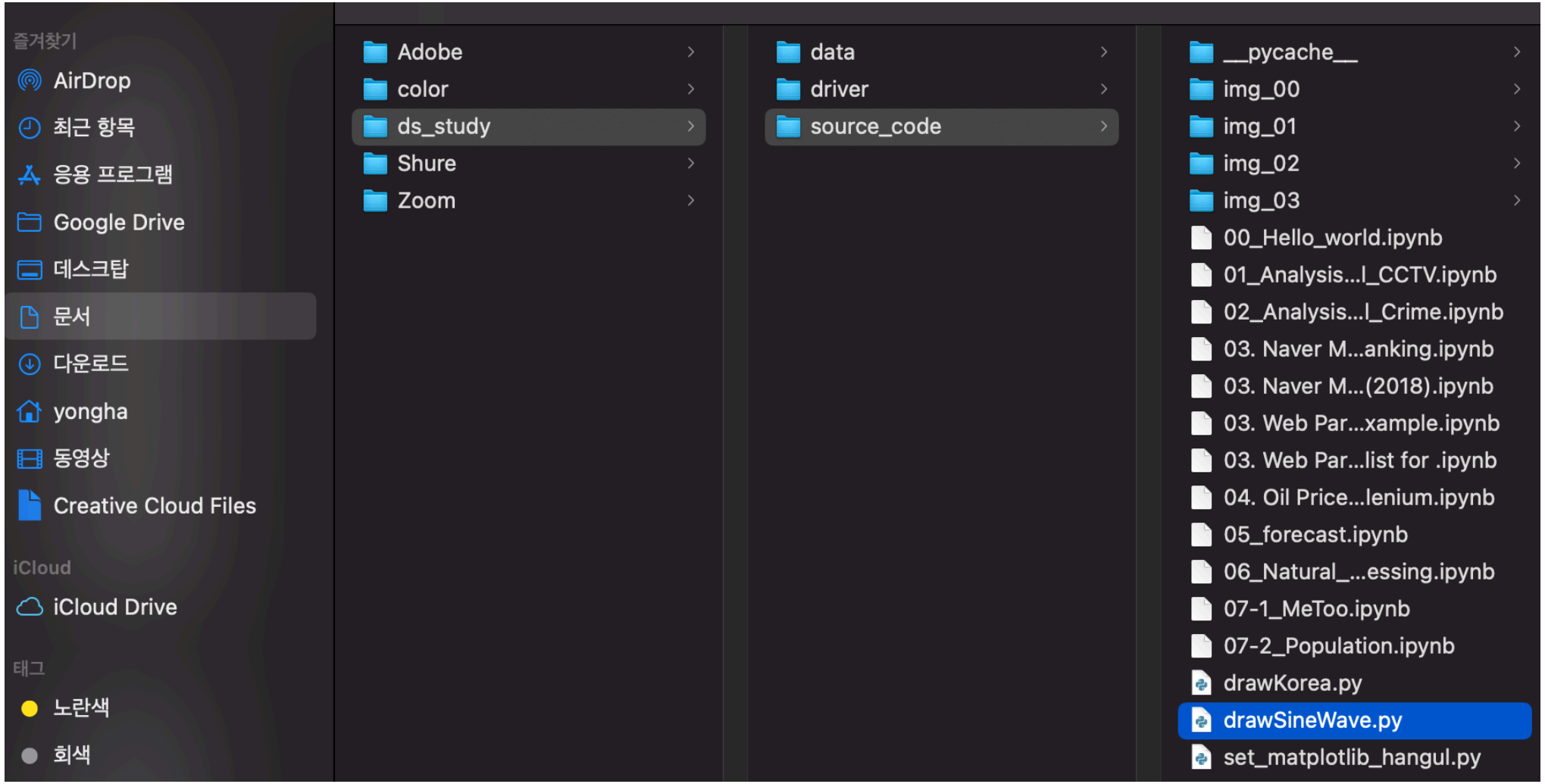
```
Last login: Tue Mar 27 12:34:28 on ttys000
 ~  ls
AndroidStudioProjects   Downloads               Pictures                seaborn-data
Applications            Library                 Public                  zsh-syntax-highlighting
Desktop                 Movies                  Samsung
Documents               Music                   nltk_data
 ~  cd Documents/FastCampus/source_code
 ~/Documents/FastCampus/source_code  ls *.py
drawSineWave.py         set_matplotlib_hangul.py slides_config.py
 ~/Documents/FastCampus/source_code  python drawSineWave.py
```

- **drawSineWave.py** 파일이 있는 폴더로 이동 (ds_study => source_code)
- python drawSineWave.py

Python 모듈(module)로 만들어서 import 하기



Python 모듈(module)로 만들어서 import 하기

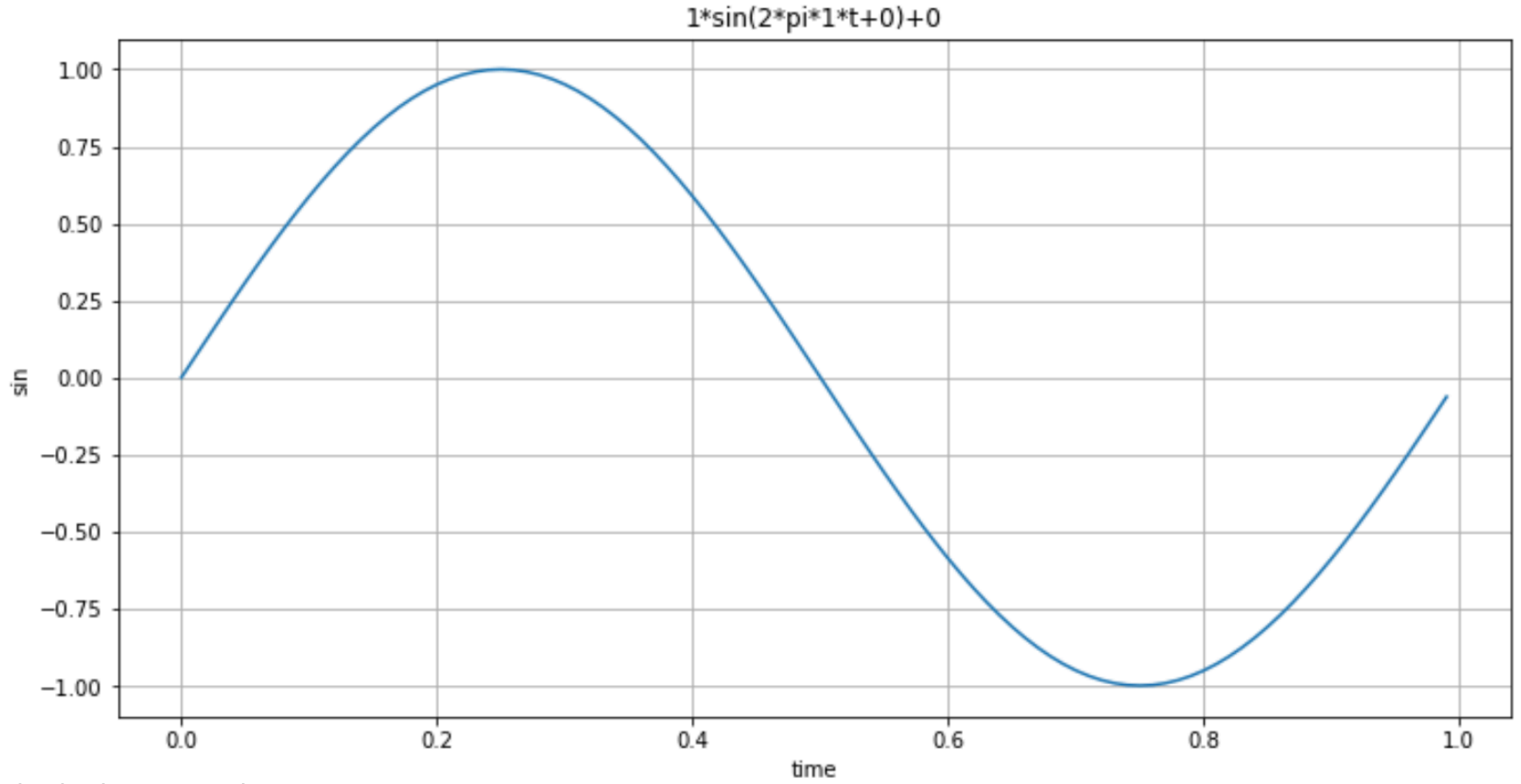


- **drawSineWave.py** 파일이 source_code 파일 안에 있는지 다시 확인^^

Python 모듈(module)로 만들어서 import 하기


```
In [13]: import drawSineWave as dS
```

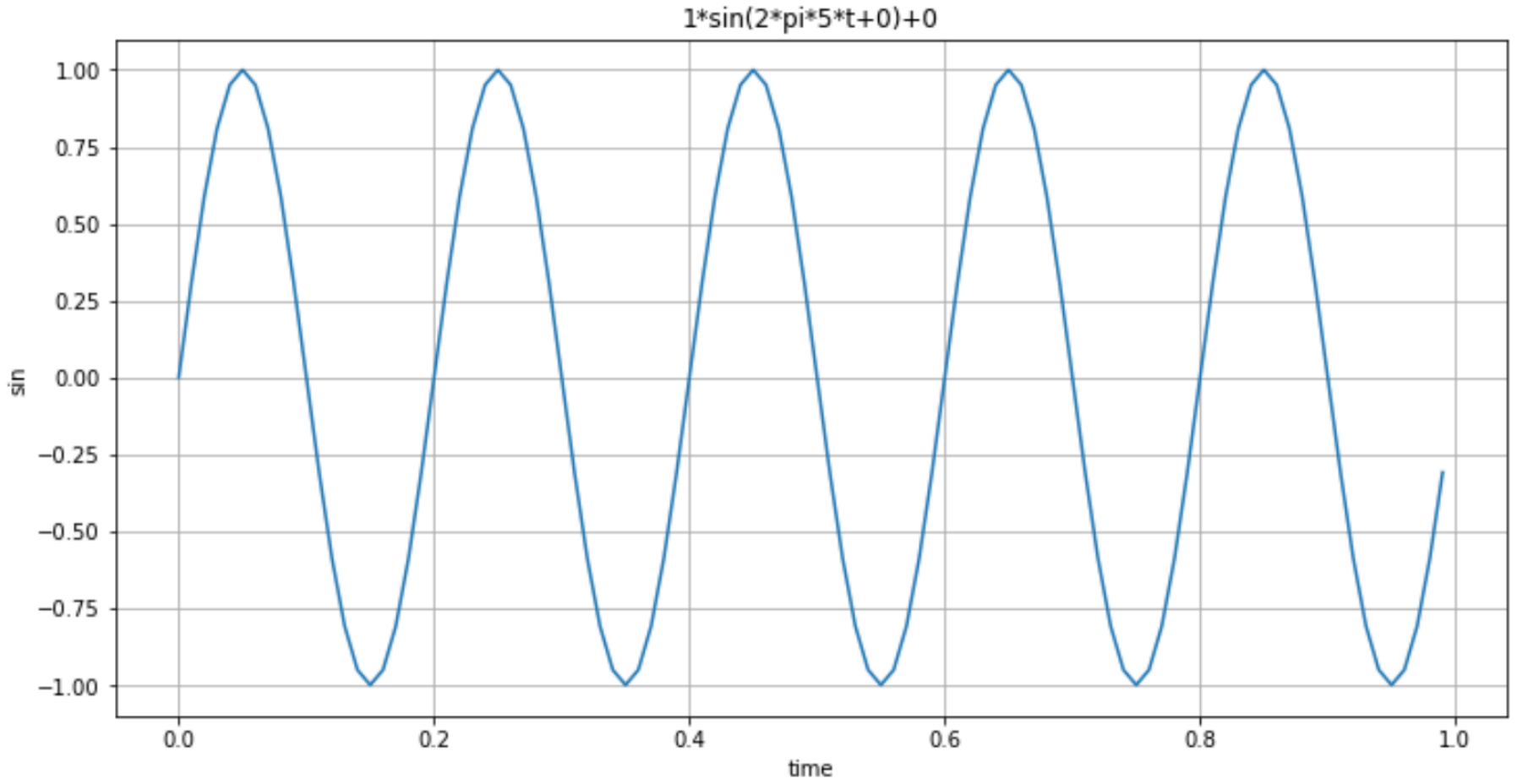
```
In [14]: dS.plotSinWave()
```



- 뭔가 멋지다… 우와 ~

Python 모듈(module)로 만들어서 import 하기


```
In [15]: dS.plotSinWave(freq=5)
```



이제 매번 하던 matplotlib의 한글 문제를 함수로 지정하자

Python 모듈(module)로 만들어서 import 하기

set_matplotlib_hangul.py 2 ●

set_matplotlib_hangul.py > ...

```python
import platform
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc

path = "c:/Windows/Fonts/malgun.ttf"

if platform.system() == "Darwin":
    print("Hangul OK in your MAC !!!")
    rc("font", family="AppleGothic")
elif platform.system() == "Windows":
    font_name = font_manager.FontProperties(fname=path).get_name()
    print("Hangul OK in your Windows !!!")
    rc("font", family=font_name)
else:
    print("Unknown system... sorry~~~~")

plt.rcParams["axes.unicode_minus"] = False
```

Fbprophet 기초

간단히 테스트 준비

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

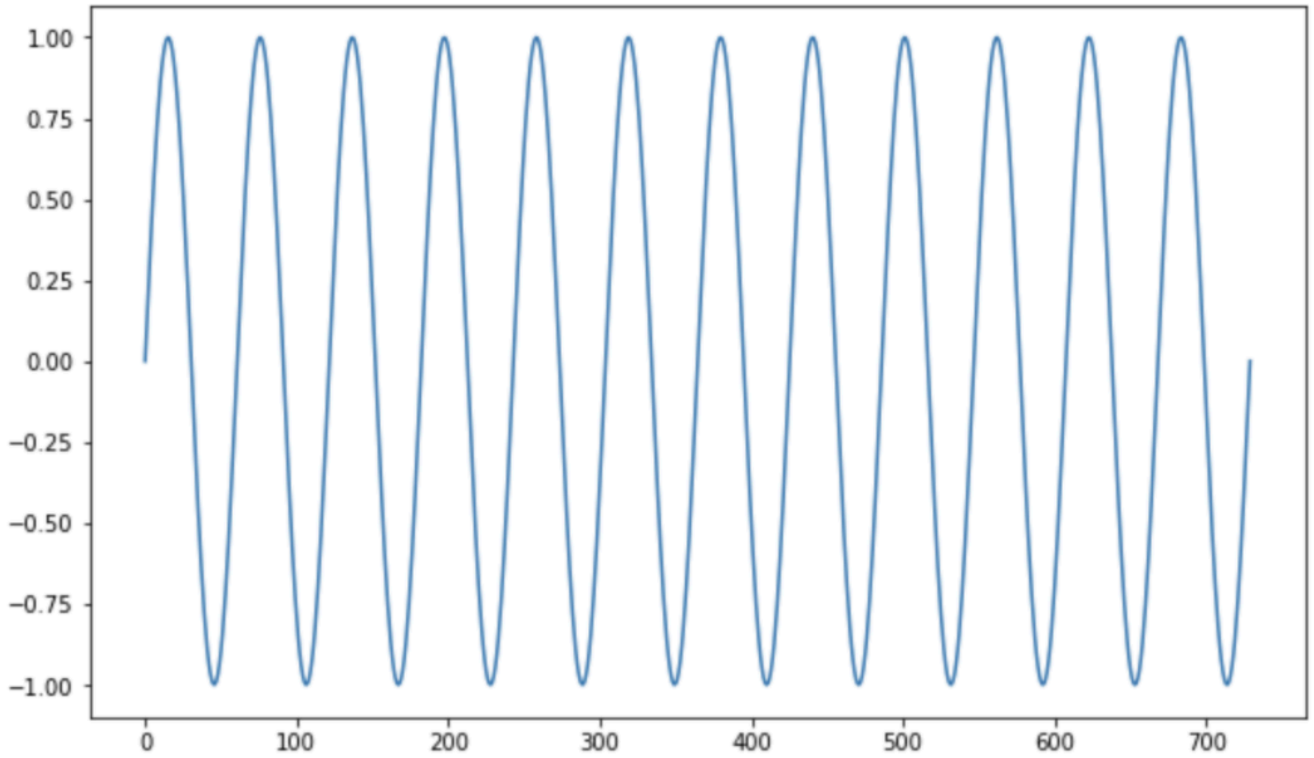
간단한 데이터 재료 준비

```
time = np.linspace(0, 1, 365*2)
result = np.sin(2*np.pi*12*time)
ds = pd.date_range('2017-01-01', periods=365*2, freq='D')
df = pd.DataFrame({'ds':ds, 'y':result})
df.head()
```

	ds	y
0	2017-01-01	0.000000
1	2017-01-02	0.103243
2	2017-01-03	0.205382
3	2017-01-04	0.305326
4	2017-01-05	0.402007

데이터는 이렇게 생겼다

```
df['y'].plot(figsize=(10,6));
```



그럼 예측을 시도해보자

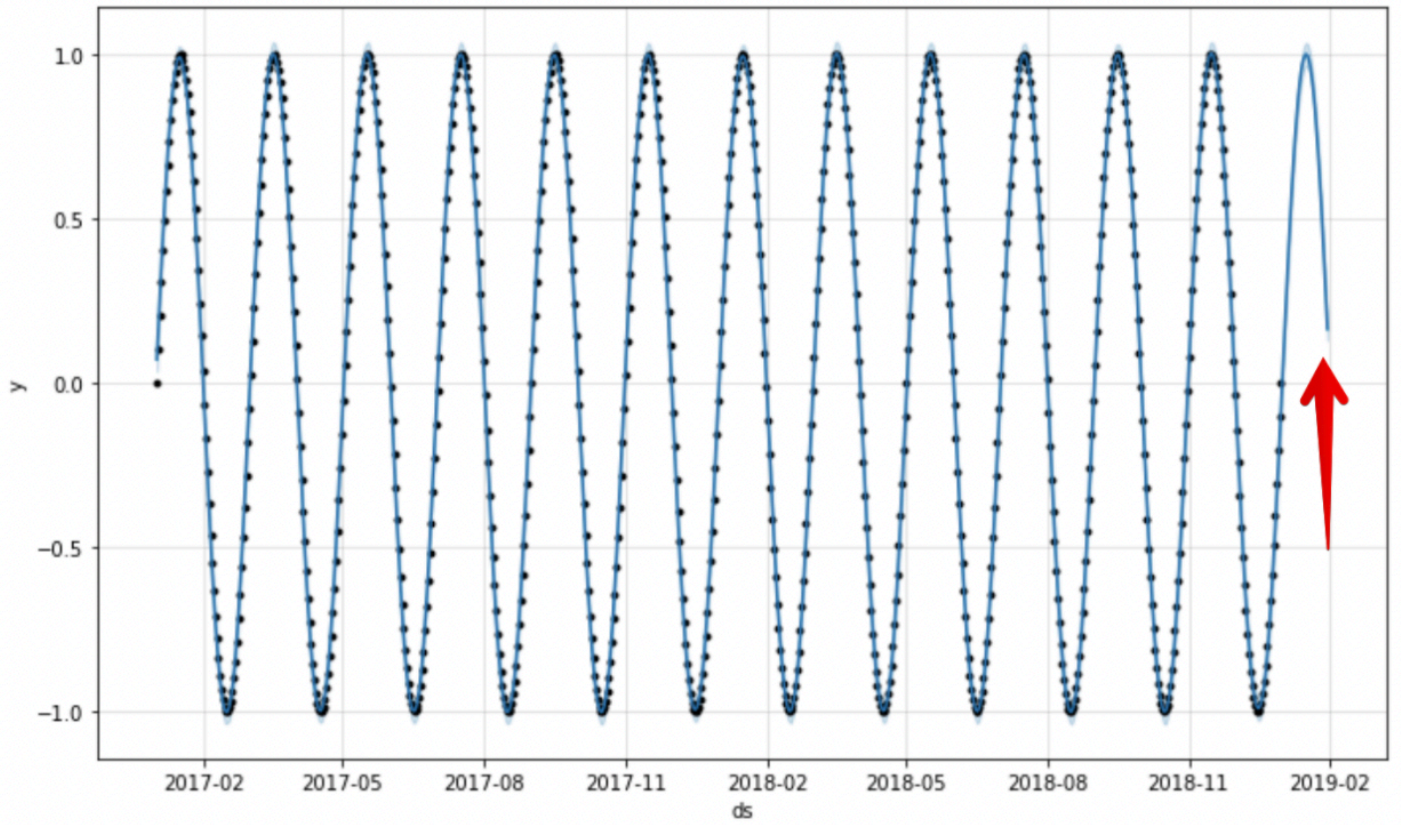
```
from fbprophet import Prophet

m = Prophet(yearly_seasonality=True, daily_seasonality=True)
m.fit(df);
```

```
future = m.make_future_dataframe(periods=30)
forecast = m.predict(future)
```

괜찮네!

```
m.plot(forecast);
```



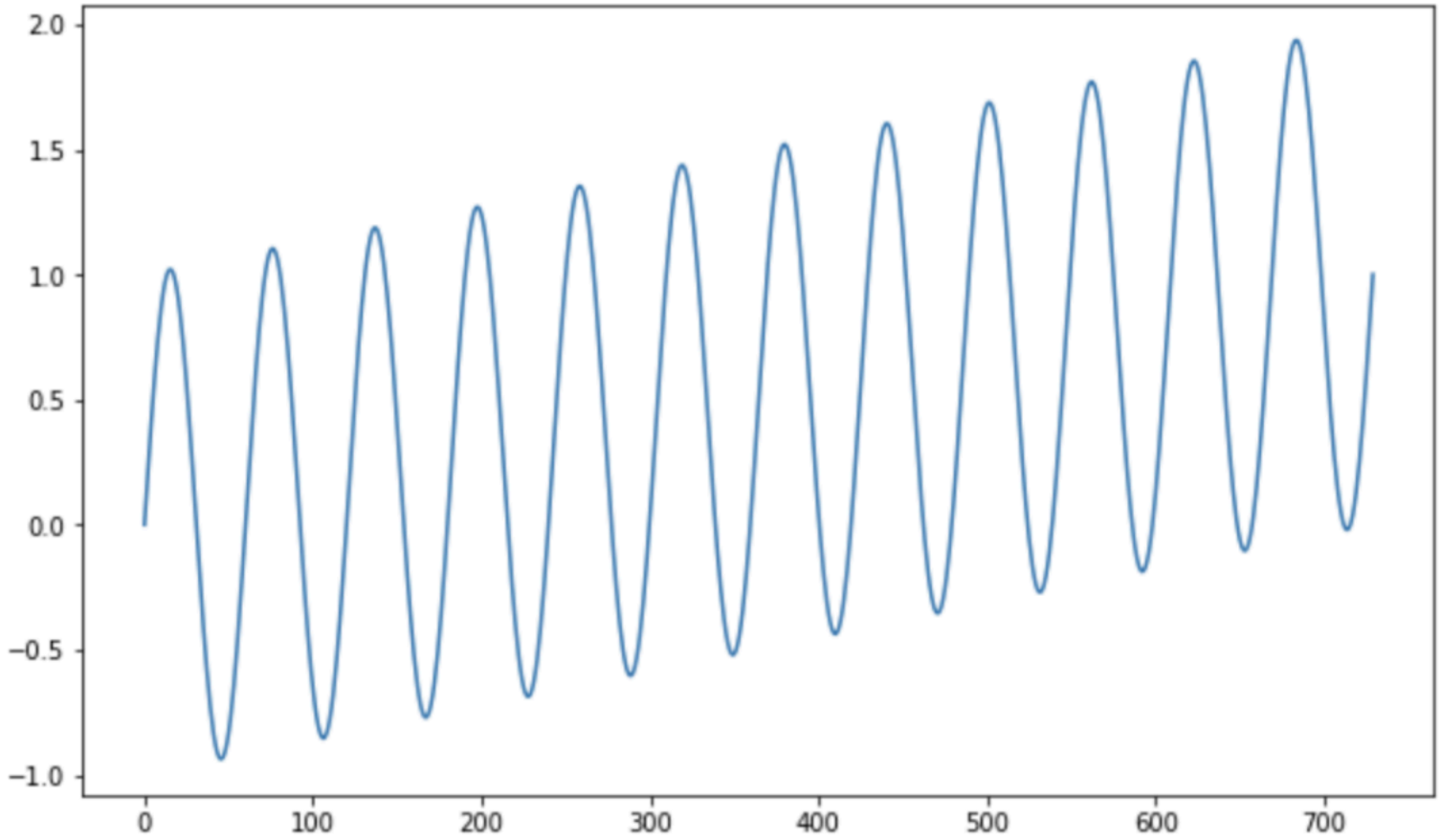
그럼 난이도를 조금 높여보자

```
time = np.linspace(0, 1, 365*2)
result = np.sin(2*np.pi*12*time) + time

ds = pd.date_range('2017-01-01', periods=365*2, freq='D')
df = pd.DataFrame({'ds':ds, 'y':result})

df['y'].plot(figsize=(10,6));
```

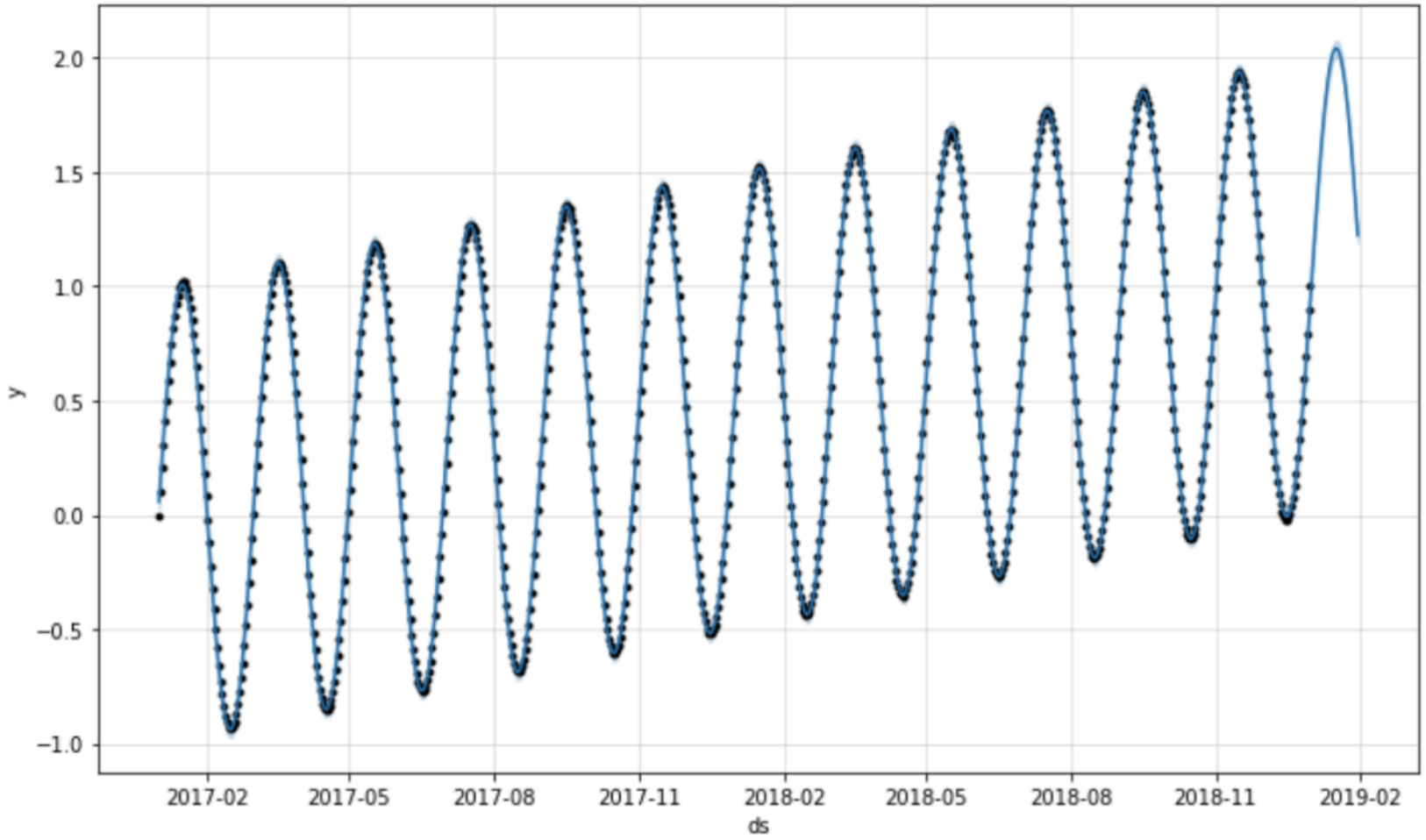
이렇게



이것도 예측해보자

```
m = Prophet(yearly_seasonality=True, daily_seasonality=True)
m.fit(df)
future = m.make_future_dataframe(periods=30)
forecast = m.predict(future)
m.plot(forecast);
```

이것도 괜찮다~



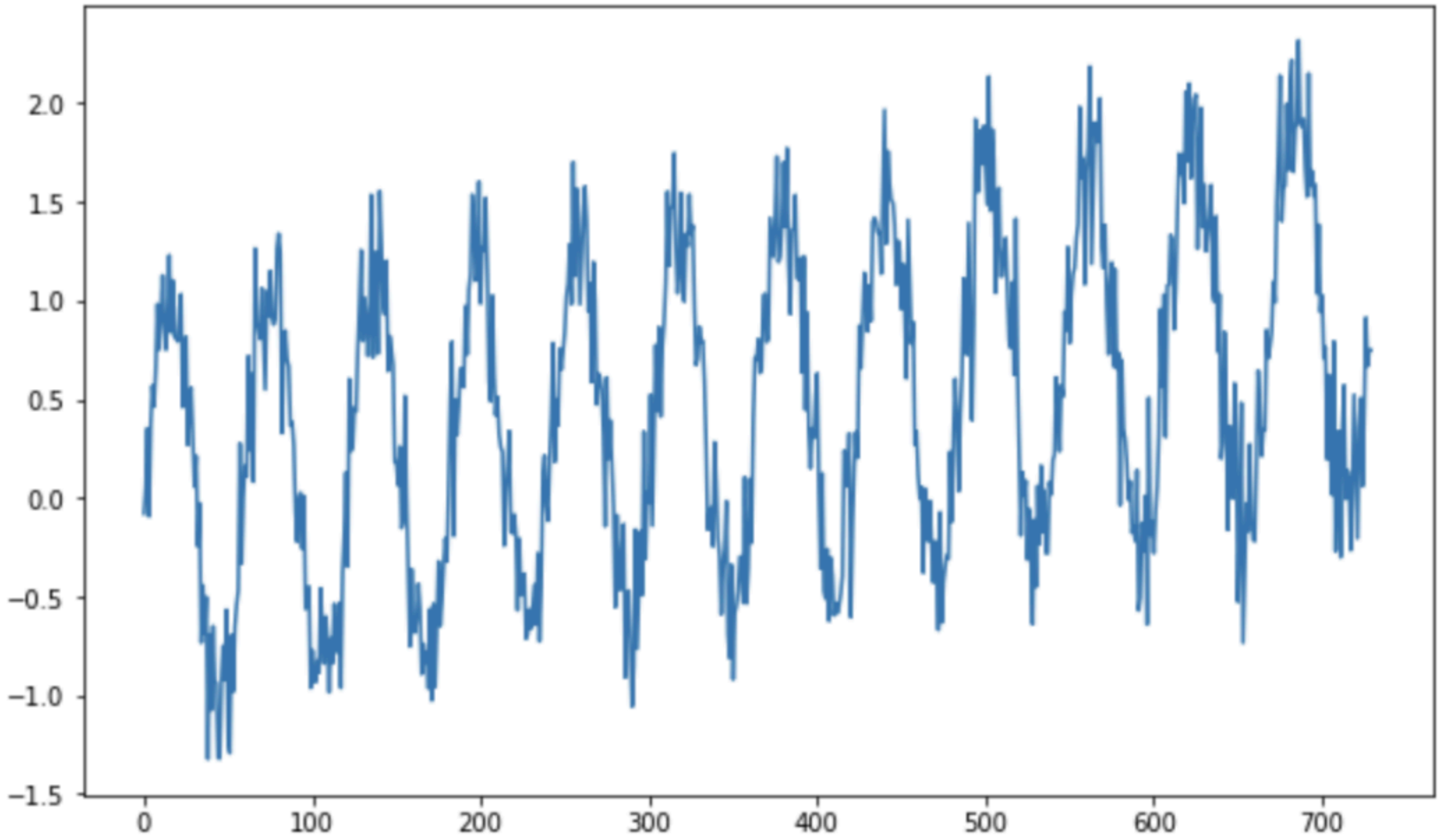
이번에는 노이즈도 살짝 실어보자~

```
time = np.linspace(0, 1, 365*2)
result = np.sin(2*np.pi*12*time) + time + np.random.randn(365*2)/4

ds = pd.date_range('2017-01-01', periods=365*2, freq='D')
df = pd.DataFrame({'ds':ds, 'y':result})

df['y'].plot(figsize=(10,6));
```

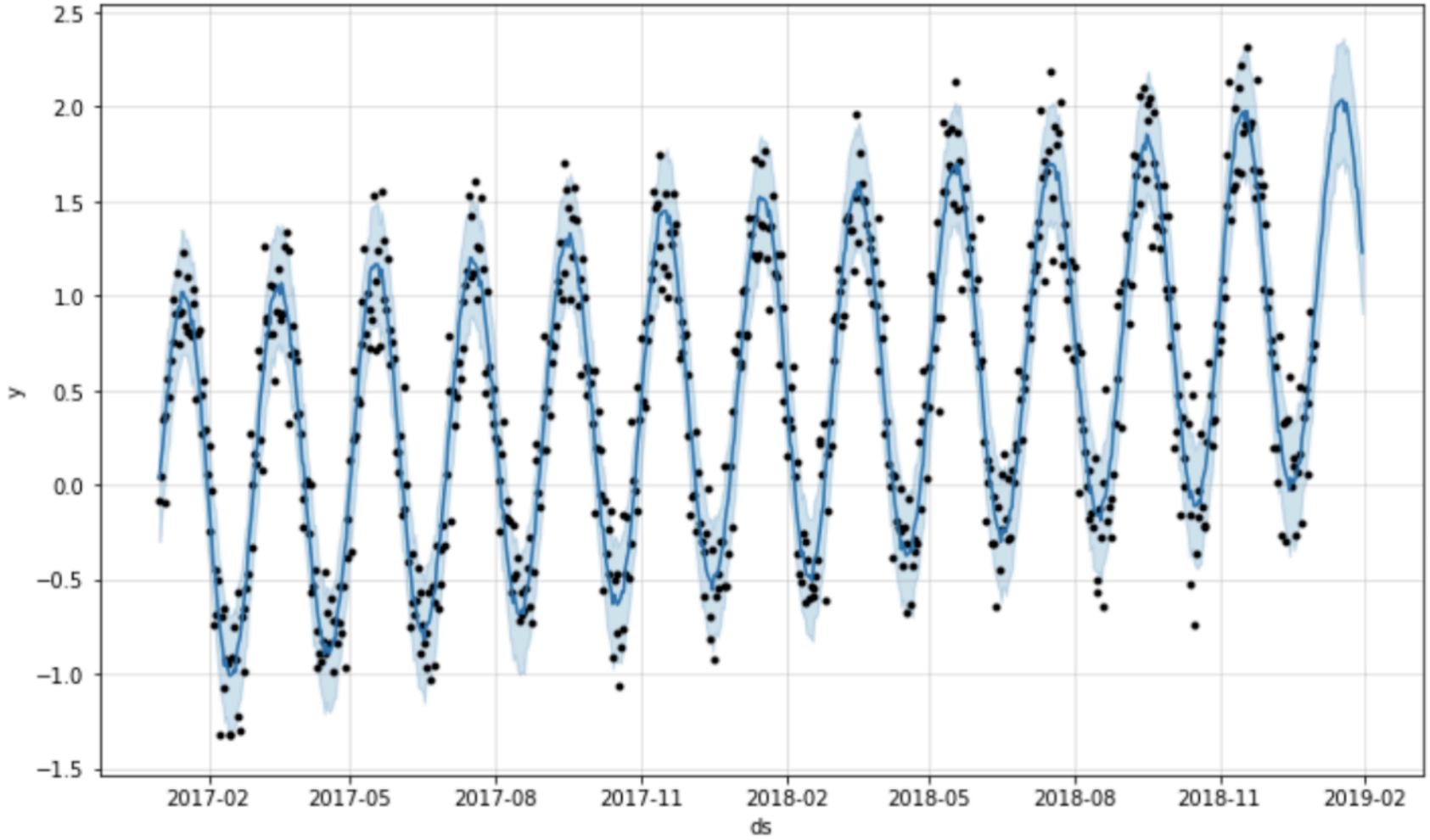
이렇게~



이것도 예측하라고 시켜보자~

```
m = Prophet(yearly_seasonality=True, daily_seasonality=True)
m.fit(df)
future = m.make_future_dataframe(periods=30)
forecast = m.predict(future)
m.plot(forecast);
```

와우~



시계열 데이터 실전 이용해보기

다시 본론으로 와서 ~

먼저 numpy를 이용해서

trend 파악은 해보자…

In [94]:
```
import pandas as pd
import pandas_datareader.data as web
import numpy as np
import matplotlib.pyplot as plt

from fbprophet import Prophet
from datetime import datetime

import set_matplotlib_hangul

#%matplotlib inline
get_ipython().run_line_magic("matplotlib", "inline")
```

In [95]:
```
pinkwink_web = pd.read_csv(
    "../data/05_PinkWink_Web_Traffic.csv",
    encoding="utf-8",
    thousands=",",
    names=["date", "hit"],
    index_col=0,
)
pinkwink_web = pinkwink_web[pinkwink_web["hit"].notnull()]
pinkwink_web.head()
```

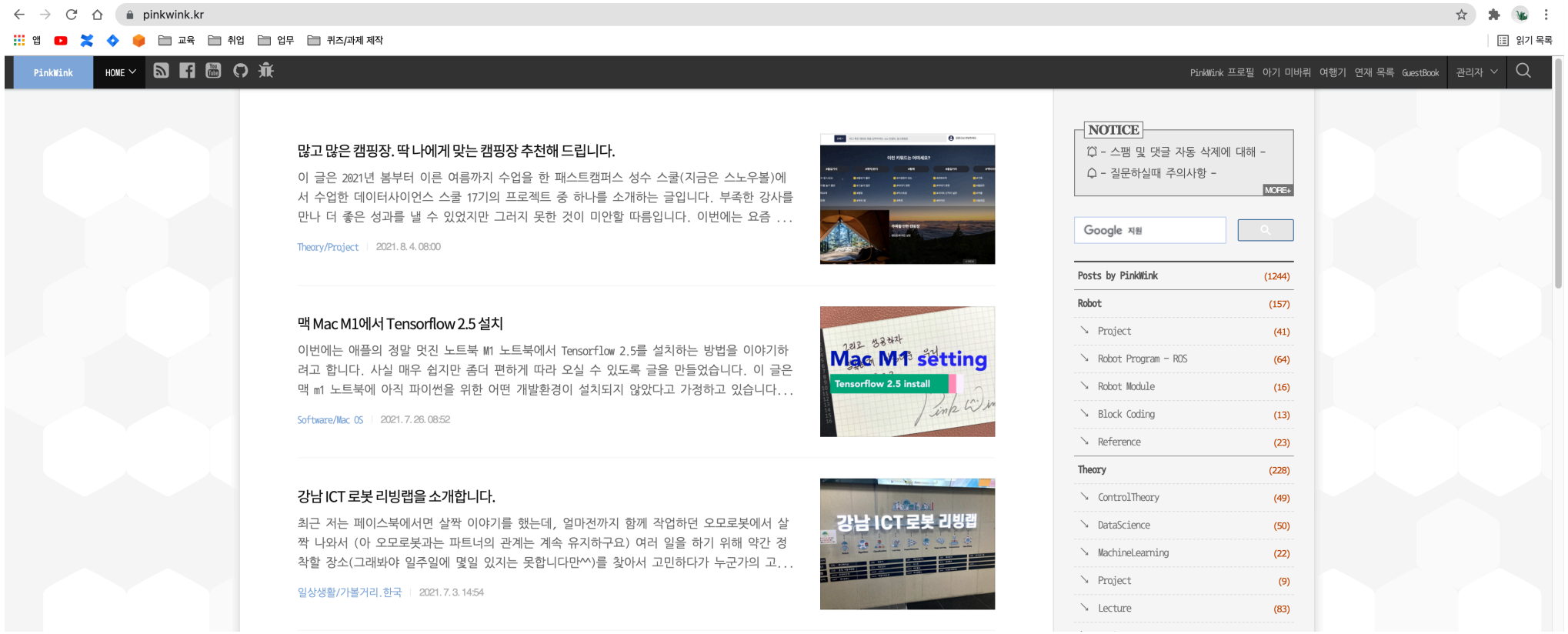
Out[95]:

date	hit
16. 7. 1.	766.0
16. 7. 2.	377.0
16. 7. 3.	427.0
16. 7. 4.	902.0
16. 7. 5.	850.0

음???

PinkWink???

블로그 데이터 **forecast**



출처: https://pinkwink.kr/

2009년 시작…

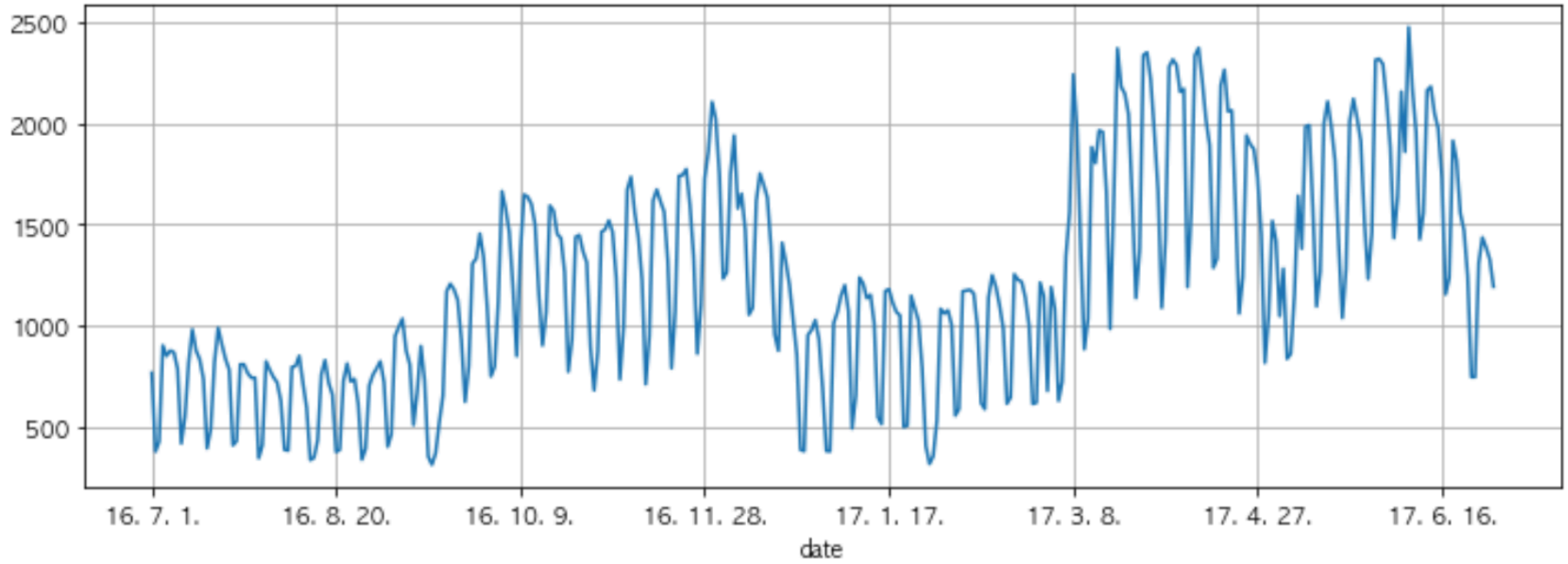
당시 시간강의 자료를 업로드하던 것을 계기로 운영…

다양한 분야를 아주 엷게 공부해서 리뷰형식으로…

지금은 그저 본인의 취미활동 자료~~~

블로그 데이터 **forecast**


```
In [96]: pinkwink_web["hit"].plot(figsize=(12, 4), grid=True);
```



- 한 번 전체 데이터를 그려보자…

블로그 데이터 **forecast**

In [97]:
```
time = np.arange(0, len(pinkwink_web))
traffic = pinkwink_web["hit"].values

fx = np.linspace(0, time[-1], 1000)
```

In [98]:
```
def error(f, x, y):
    return np.sqrt(np.mean((f(x) - y) ** 2))
```

- trend 분석을 시각화하기 위한 x 축 값들을 만들어 두고
- 에러를 계산할 함수도 만들어 두자

블로그 데이터 **forecast**

In [99]:
```
fp1 = np.polyfit(time, traffic, 1)
f1 = np.poly1d(fp1)

f2p = np.polyfit(time, traffic, 2)
f2 = np.poly1d(f2p)

f3p = np.polyfit(time, traffic, 3)
f3 = np.poly1d(f3p)

f15p = np.polyfit(time, traffic, 15)
f15 = np.poly1d(f15p)
```

- 예전에 했던 polyfit… 반갑다~~🙂

블로그 데이터 **forecast**

In [100]:
```
print(error(f1, time, traffic))
print(error(f2, time, traffic))
print(error(f3, time, traffic))
print(error(f15, time, traffic))
```

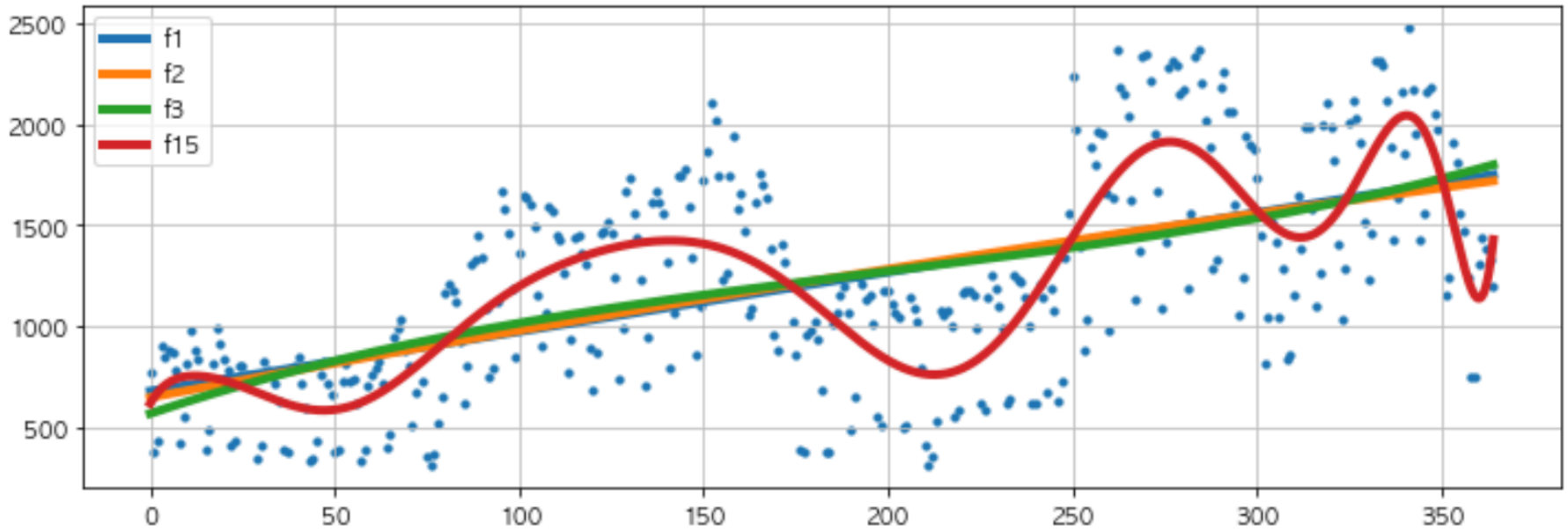
```
430.85973081109637
430.6284101894695
429.53280466762925
330.4777305307993
```

- 1차, 2차, 3차 함수까지는 에러의 큰 차이가 없다…
- 그럼 15차를 선택할까…
- 그냥 1차를 선택할까…

블로그 데이터 **forecast**


```
In [101]: plt.figure(figsize=(12, 4))
          plt.scatter(time, traffic, s=10)
          plt.plot(fx, f1(fx), lw=4, label="f1")
          plt.plot(fx, f2(fx), lw=4, label="f2")
          plt.plot(fx, f3(fx), lw=4, label="f3")
          plt.plot(fx, f15(fx), lw=4, label="f15")

          plt.grid(True, linestyle="-", color="0.75")
          plt.legend(loc=2)
          plt.show()
```



- 결정은 디자이너의 몫 ~~

블로그 데이터 **forecast**

In [102]:
```
df = pd.DataFrame({"ds": pinkwink_web.index, "y": pinkwink_web["hit"]})
df.reset_index(inplace=True)
df["ds"] = pd.to_datetime(df["ds"], format="%y. %m. %d.")
del df["date"]
df.head()
```

Out[102]:

	ds	y
0	2016-07-01	766.0
1	2016-07-02	377.0
2	2016-07-03	427.0
3	2016-07-04	902.0
4	2016-07-05	850.0

- 데이터를 prophet에서 사용하기 좋게 바꾸고…

블로그 데이터 **forecast**


```
In [103]: m = Prophet(yearly_seasonality=True, daily_seasonality=True)
          m.fit(df);
```

- prophet 에 적용 ~~~

블로그 데이터 **forecast**


```
In [104]: future = m.make_future_dataframe(periods=60)
          future.tail()
```

Out[104]:

	ds
420	2017-08-25
421	2017-08-26
422	2017-08-27
423	2017-08-28
424	2017-08-29

- 이 후 60일에 해당하는 데이터를 예측해보기~~~

블로그 데이터 **forecast**

In [105]:
```
forecast = m.predict(future)
forecast[["ds", "yhat", "yhat_lower", "yhat_upper"]].tail()
```

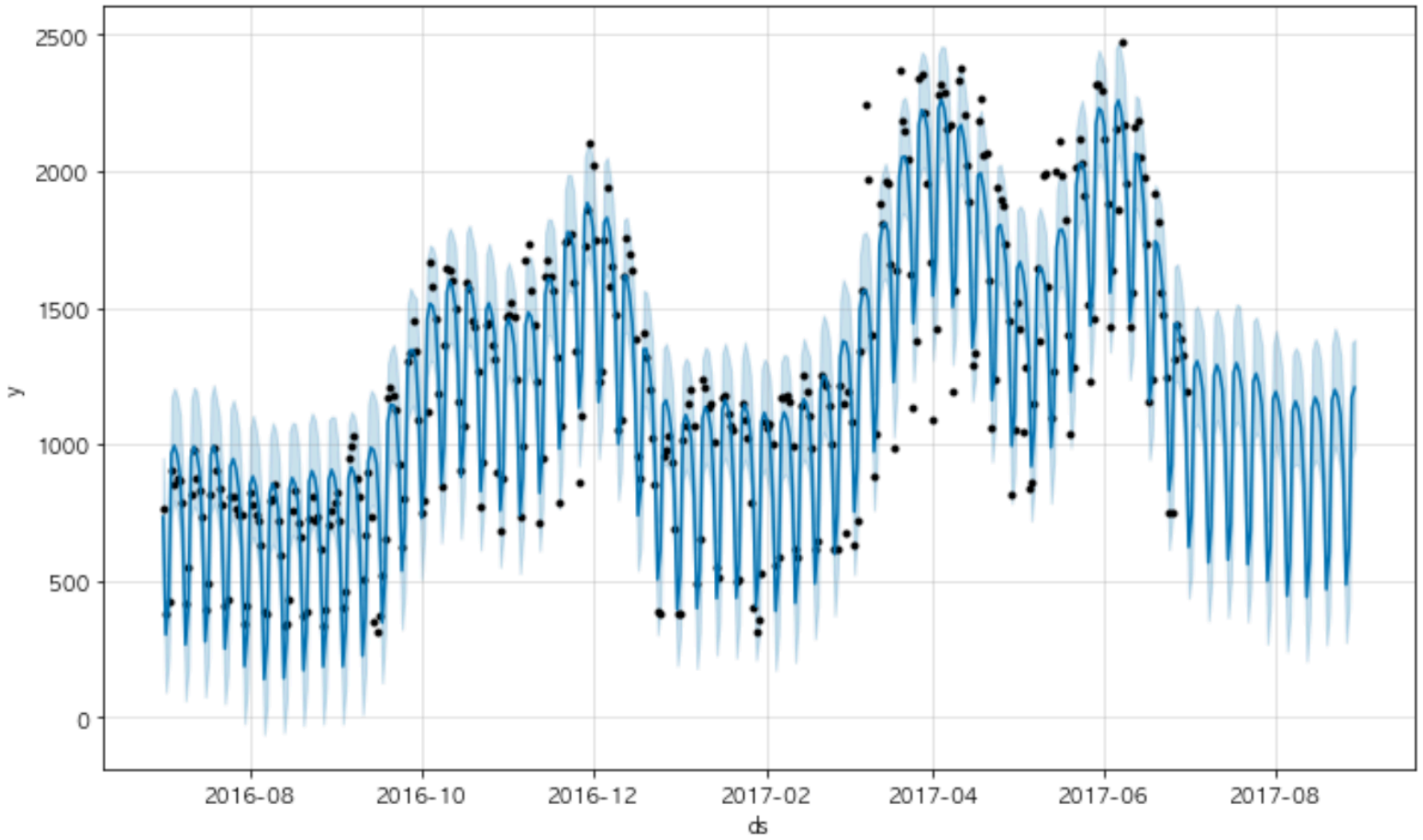
Out[105]:

	ds	yhat	yhat_lower	yhat_upper
420	2017-08-25	900.611910	677.623548	1104.637321
421	2017-08-26	485.482790	275.073449	687.862896
422	2017-08-27	618.155087	409.992065	833.947428
423	2017-08-28	1170.989096	949.011409	1371.448776
424	2017-08-29	1207.006811	986.597486	1383.323683

- 예측 결과는 상한/하한의 범위를 포함해서 얻어진다…

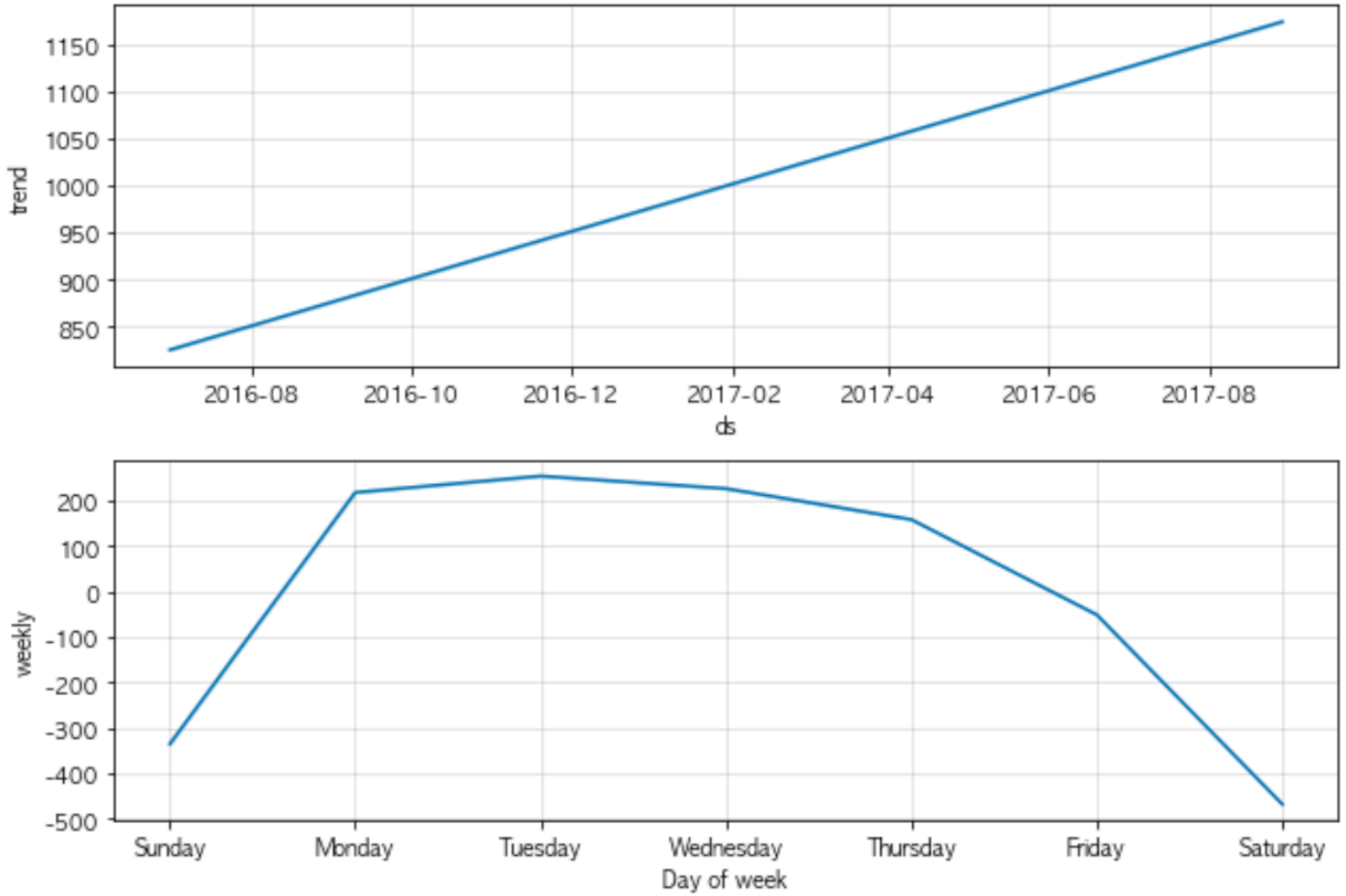
블로그 데이터 **forecast**


```
In [106]: m.plot(forecast);
```



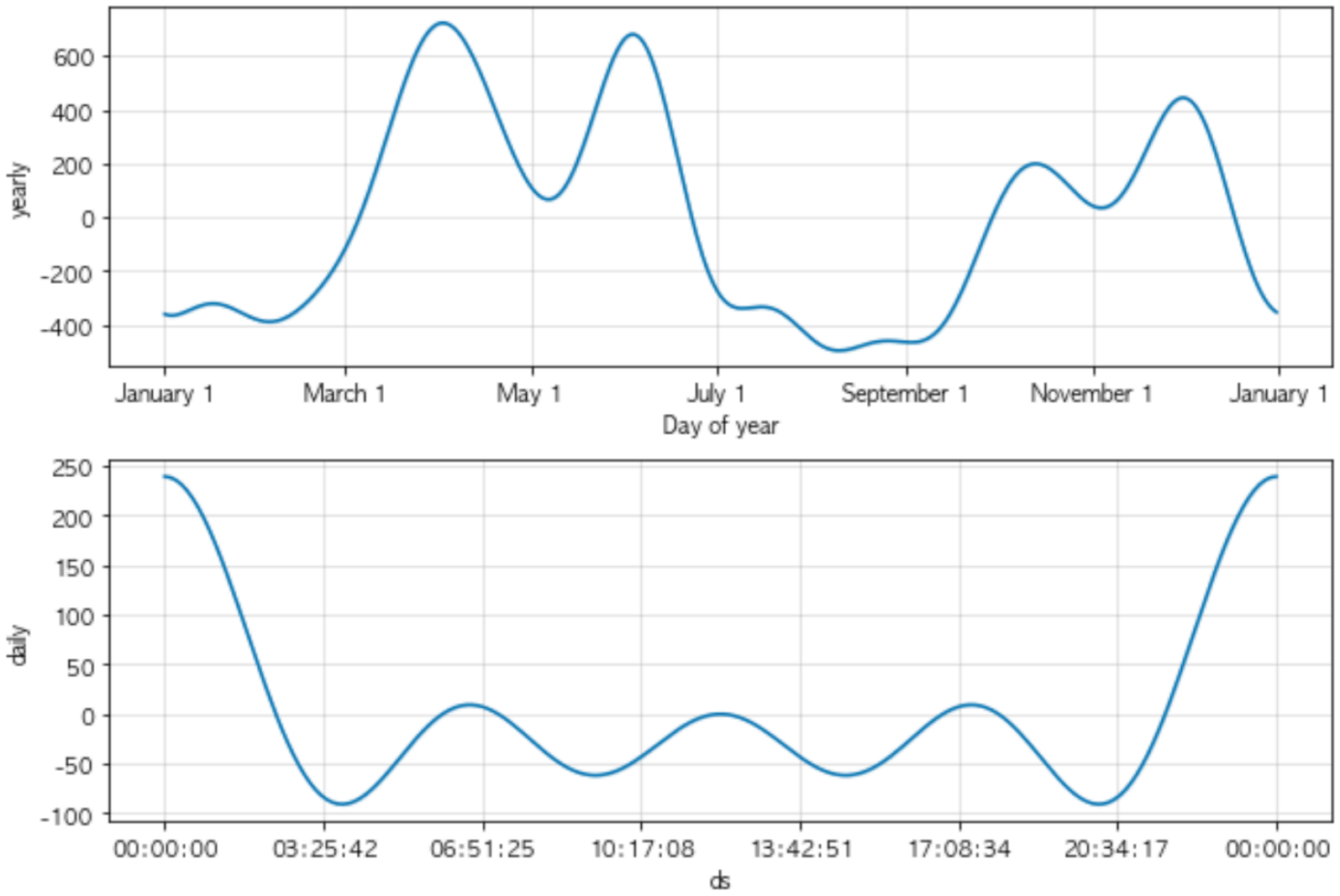
블로그 데이터 **forecast**


```
In [107]: m.plot_components(forecast);
```



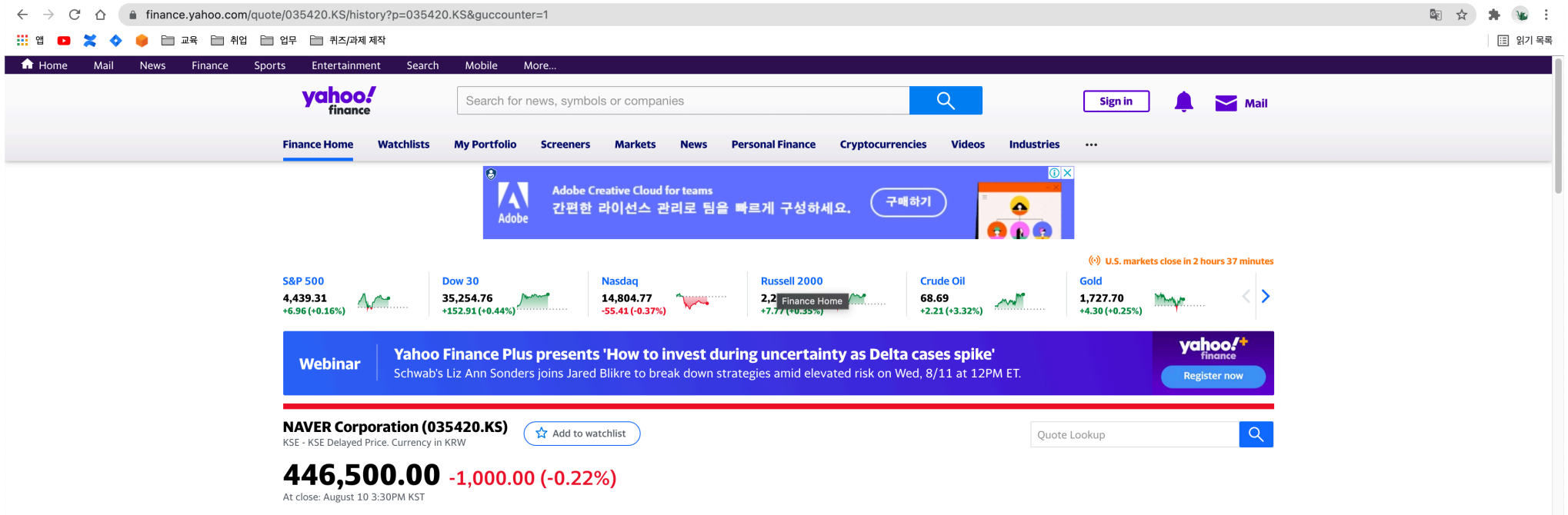
- forecast 결과에 대해 component를 확인할 수 있다.

블로그 데이터 **forecast**



주식 데이터 fbprophet으로 분석하기

야후 **finance** 주식 데이터 **forecast**



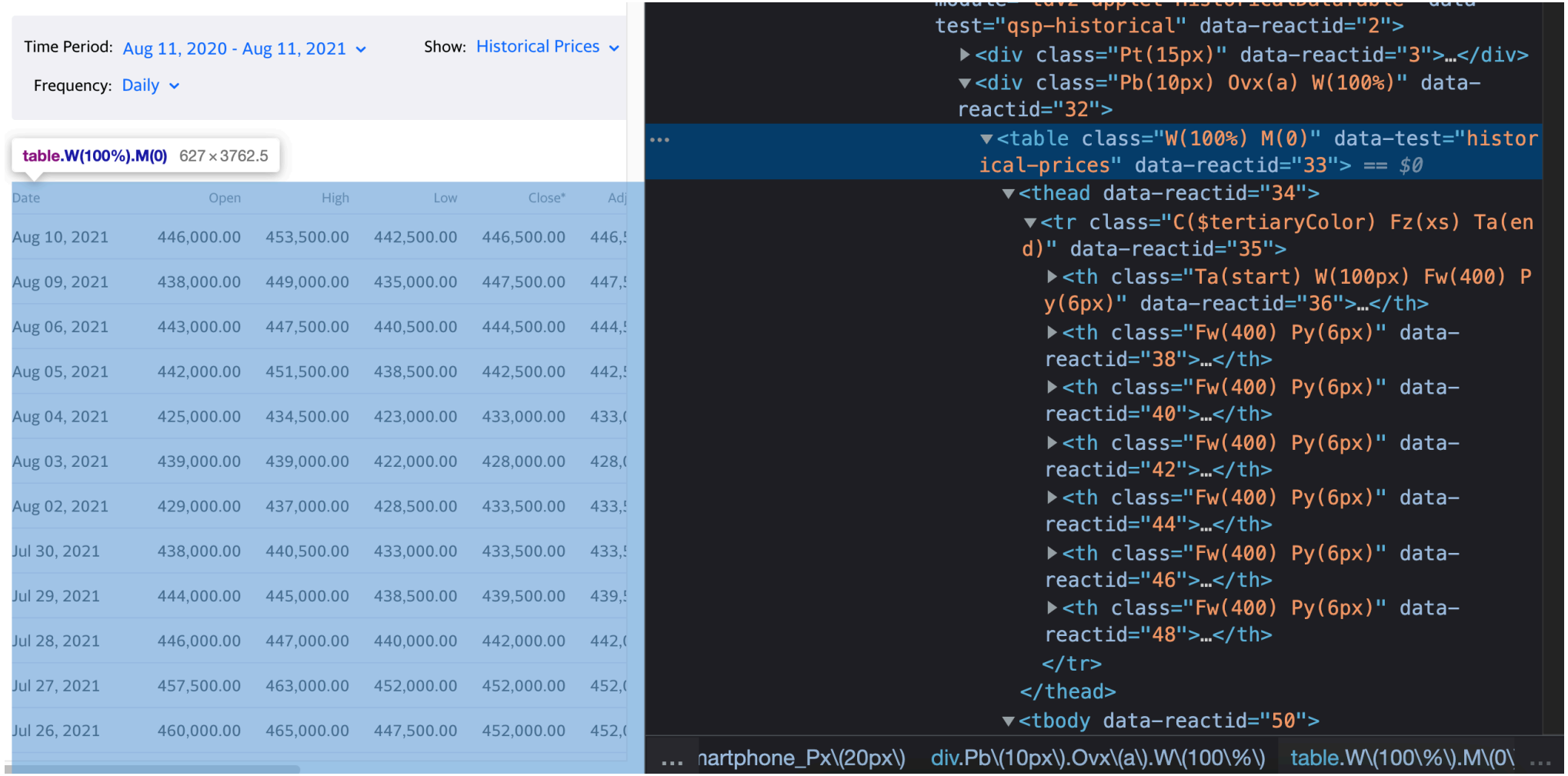
- https://finance.yahoo.com/quote/035420.KS/history?p=035420.KS&guccounter=1

야후 **finance** 주식 데이터 **forecast**

zero-base /

Date	Open	High	Low	Close*	Adj Close**	Volume
Aug 10, 2021	446,000.00	453,500.00	442,500.00	446,500.00	446,500.00	460,450
Aug 09, 2021	438,000.00	449,000.00	435,000.00	447,500.00	447,500.00	528,032
Aug 06, 2021	443,000.00	447,500.00	440,500.00	444,500.00	444,500.00	416,829
Aug 05, 2021	442,000.00	451,500.00	438,500.00	442,500.00	442,500.00	744,924
Aug 04, 2021	425,000.00	434,500.00	423,000.00	433,000.00	433,000.00	545,873
Aug 03, 2021	439,000.00	439,000.00	422,000.00	428,000.00	428,000.00	830,841
Aug 02, 2021	429,000.00	437,000.00	428,500.00	433,500.00	433,500.00	551,937
Jul 30, 2021	438,000.00	440,500.00	433,000.00	433,500.00	433,500.00	628,812

야후 **finance** 주식 데이터 **forecast**



야후 finance 주식 데이터 forecast

In [116]:
```
from bs4 import BeautifulSoup
from urllib.request import Request, urlopen

req = Request(
    "https://finance.yahoo.com/quote/035420.KS/history?p=035420.KS",
    headers={"User-Agent": "Chrome"},
)
page = urlopen(req).read()

soup = BeautifulSoup(page, "html.parser")
table = soup.find("table")
df_raw = pd.read_html(str(table))[0]
df_raw.head()
```

Out[116]:

	Date	Open	High	Low	Close*	Adj Close**	Volume
0	Jul 22, 2021	433000.00	445000.00	429000.00	440000.00	440000.00	1033174
1	Jul 21, 2021	443000.00	446000.00	428000.00	428000.00	428000.00	885519
2	Jul 20, 2021	438500.00	441500.00	431000.00	439000.00	439000.00	789090

- 원하는 정보가 table 태그안에 있을 경우 아주 쉽게 정보를 가져올 수 있다.
- 단 약간의 사용법상 주의점만 감안하면 된다.

야후 **finance** 주식 데이터 **forecast**

In [117]:
```
df_tmp = pd.DataFrame({"ds": df_raw["Date"], "y": df_raw["Close*"]})
df_target = df_tmp[:-1]
df_target.head()
```

Out[117]:

	ds	y
0	Jul 22, 2021	440000.00
1	Jul 21, 2021	428000.00
2	Jul 20, 2021	439000.00
3	Jul 19, 2021	443000.00
4	Jul 16, 2021	447000.00

- fbprophet을 사용하는 형식에 맞춰주고…
- 맨 마지막에 NaN이 있어서 제외

야후 **finance** 주식 데이터 **forecast**

In [118]:
```
df = df_target.copy()
df["ds"] = pd.to_datetime(df_target["ds"], format="%b %d, %Y")
df.head()
```

Out[118]:

	ds	y
0	2021-07-22	440000.00
1	2021-07-21	428000.00
2	2021-07-20	439000.00
3	2021-07-19	443000.00
4	2021-07-16	447000.00

- Hard Copy를 시키고
- 날짜를 fbprophet이 요구하는 형태로 바꿔준다

야후 **finance** 주식 데이터 **forecast**

In [11]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   ds      100 non-null    datetime64[ns]
 1   y       100 non-null    object        
dtypes: datetime64[ns](1), object(1)
memory usage: 1.7+ KB
```

In [12]:
```
1 # 데이터형 변환 object => float
2
3 df["y"] = df["y"].astype("float")
```

- y 컬럼의 데이터 타입이 문자열(object)로 되어 있다
- 실수형(float)으로 변환

야후 finance 주식 데이터 forecast

In [119]:
```
m = Prophet(yearly_seasonality=True, daily_seasonality=True)
m.fit(df);
```

In [120]:
```
future = m.make_future_dataframe(periods=30)
forecast = m.predict(future)
forecast[["ds", "yhat", "yhat_lower", "yhat_upper"]].tail()
```

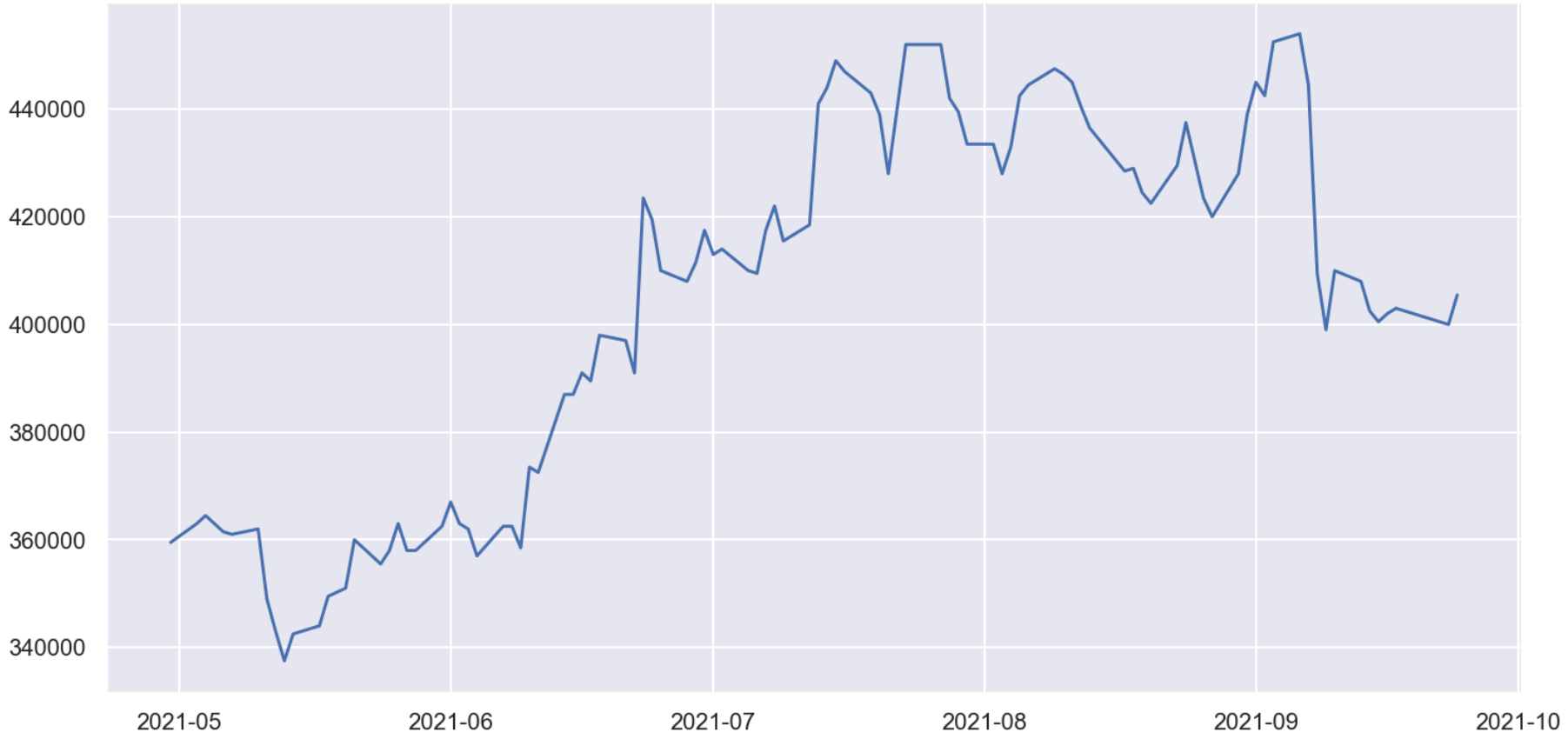
Out[120]:

	ds	yhat	yhat_lower	yhat_upper
125	2021-08-17	158876.621229	151729.320817	166550.338362
126	2021-08-18	175954.147258	168002.354820	183509.518822
127	2021-08-19	196065.510574	188036.800733	203681.647316
128	2021-08-20	217076.225565	208600.456398	224352.799461
129	2021-08-21	242931.856011	234918.296635	251086.568424

야후 **finance** 주식 데이터 **forecast**

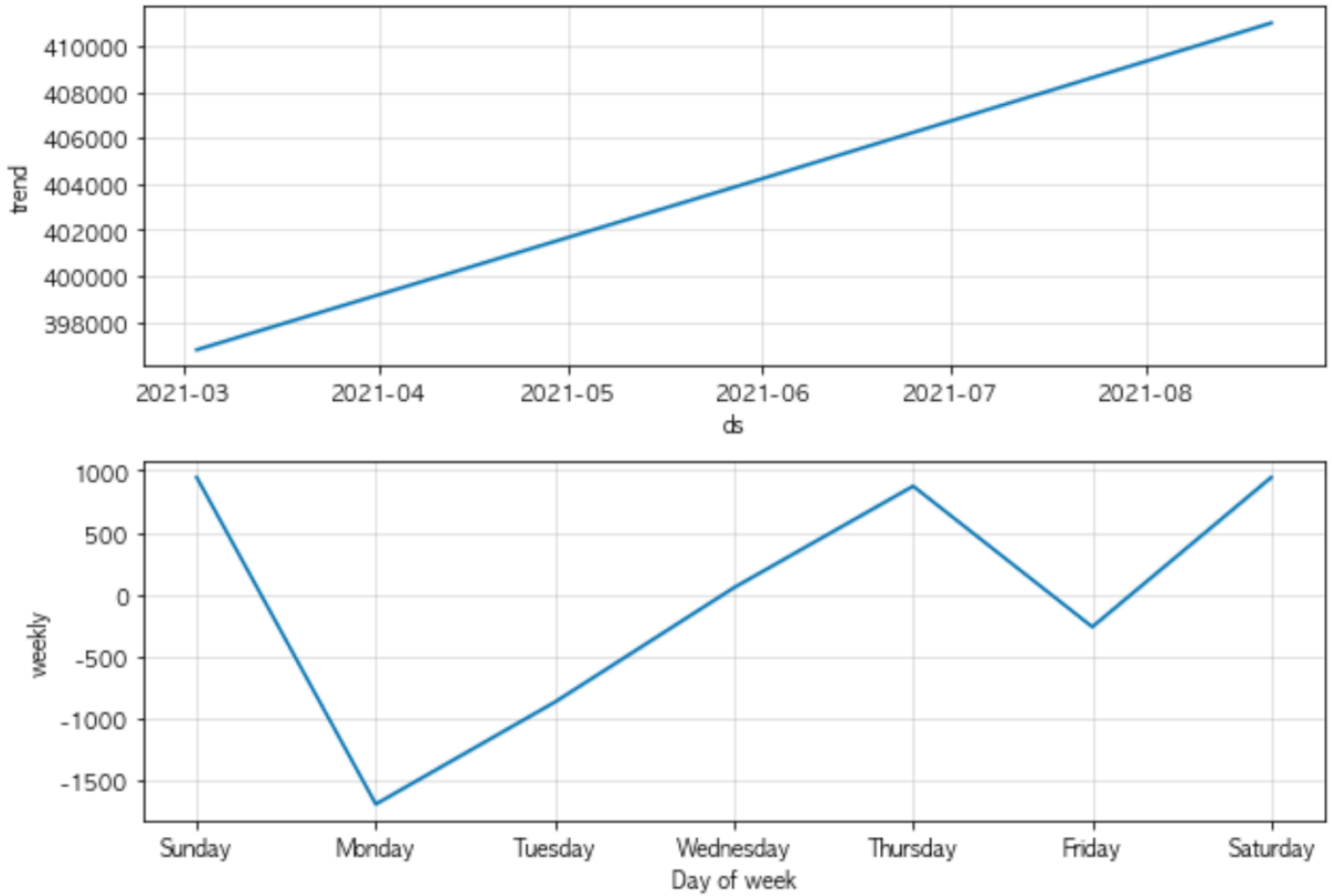
zero-base /

```
In [121]: plt.figure(figsize=(12, 6))
          plt.plot(df["ds"], df["y"], label="real")
          plt.grid()
          plt.show()
```

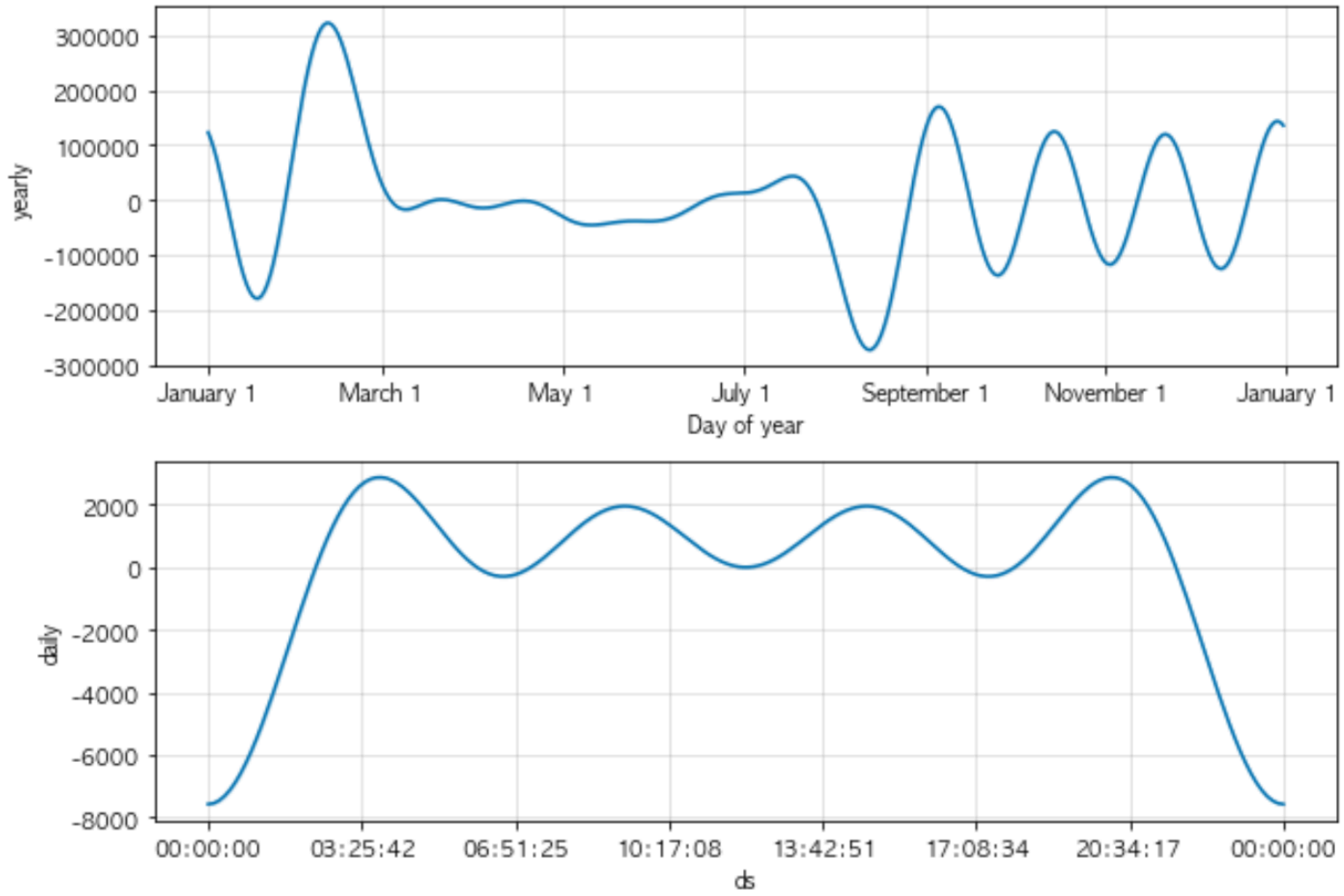


야후 **finance** 주식 데이터 **forecast**


```
In [122]: m.plot_components(forecast);
```



야후 **finance** 주식 데이터 **forecast**



야후 finance 주식 데이터 forecast


```
(ds_study) ✘ yongha ~/Documents/ds_study/source_code pip install yfinance
Collecting yfinance
  Downloading yfinance-0.1.63.tar.gz (26 kB)
Requirement already satisfied: pandas>=0.24 in /opt/homebrew/Caskroom/miniforge/base/envs/ds_study/lib/python3.8/site-pac
kages (from yfinance) (1.3.1)
Requirement already satisfied: numpy>=1.15 in /opt/homebrew/Caskroom/miniforge/base/envs/ds_study/lib/python3.8/site-pack
ages (from yfinance) (1.21.1)
Requirement already satisfied: requests>=2.20 in /opt/homebrew/Caskroom/miniforge/base/envs/ds_study/lib/python3.8/site-p
```

- **pip install yfinance**
- 원래 Pandas는 data_reader를 통해 주가 정보를 얻어오는 기능이 있었다
- 그런데 최근 구글과 야후의 안정성 문제로 이 기능이 동작하지 않는다
- 그래서 우회적으로 야후의 기능을 복구시켜주는 모듈이 임시로 만들어 졌다…
- 설치하자 ~~~

야후 **finance** 주식 데이터 **forecast**

In [123]:
```
from pandas_datareader import data
import yfinance as yf

yf.pdr_override()

start_date = "2010-03-01"
end_date = "2018-02-28"
KIA = data.get_data_yahoo("000270.KS", start_date, end_date)
```

```
[**********100%**********]  1 of 1 completed
```

- 사용법은 간단하다.. 기아자동차의 종목코드를 가지고 기간을 입력하면 끝~

야후 finance 주식 데이터 forecast


```
In [124]: KIA.head()
```

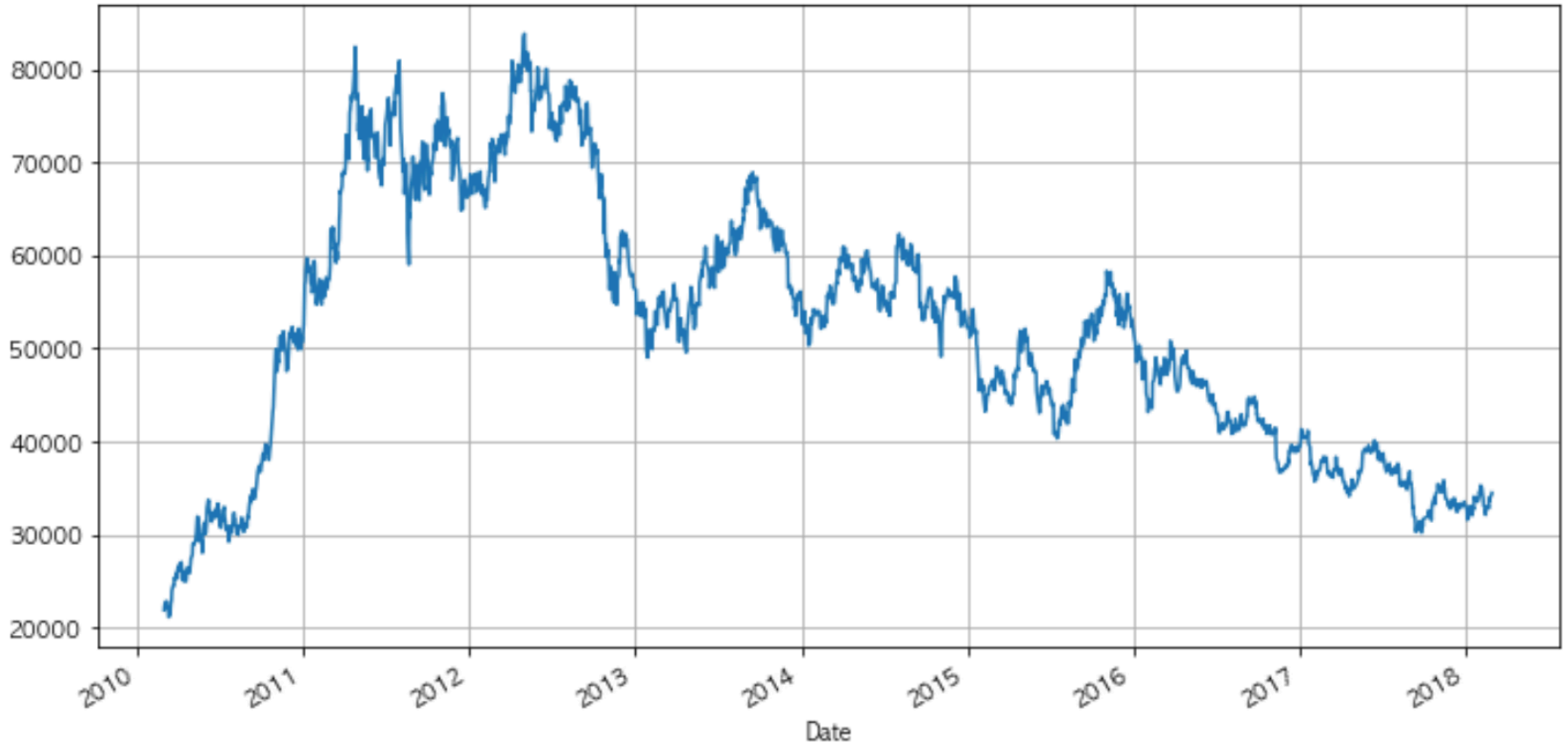
Out[124]:

Date	Open	High	Low	Close	Adj Close	Volume
2010-03-02	22050.0	22300.0	21800.0	21850.0	17800.662109	3935516
2010-03-03	22100.0	22450.0	21850.0	22400.0	18248.732422	4380617
2010-03-04	22400.0	22600.0	22300.0	22500.0	18330.199219	2490087
2010-03-05	22500.0	22750.0	22350.0	22750.0	18533.869141	2379282
2010-03-08	23050.0	23100.0	22500.0	22800.0	18574.603516	4326618

야후 **finance** 주식 데이터 **forecast**

zero-base /

```
In [125]: KIA["Close"].plot(figsize=(12, 6), grid=True);
```



야후 **finance** 주식 데이터 **forecast**

In [126]:
```
KIA_trunc = KIA[:"2017-11-30"]
KIA_trunc.head();
```

- 나중에 비교를 위해 조금 잘라두자~~ 뭐 개념상 accuracy 확인??

야후 **finance** 주식 데이터 **forecast**

In [127]:
```
df = pd.DataFrame({"ds": KIA_trunc.index, "y": KIA_trunc["Close"]})
df.reset_index(inplace=True)
del df["Date"]
df.head()
```

Out[127]:

	ds	y
0	2010-03-02	21850.0
1	2010-03-03	22400.0
2	2010-03-04	22500.0
3	2010-03-05	22750.0
4	2010-03-08	22800.0

- Forecast 를 위한 준비

야후 finance 주식 데이터 forecast

In [128]:
```
m = Prophet(yearly_seasonality=True, daily_seasonality=True)
m.fit(df);
```

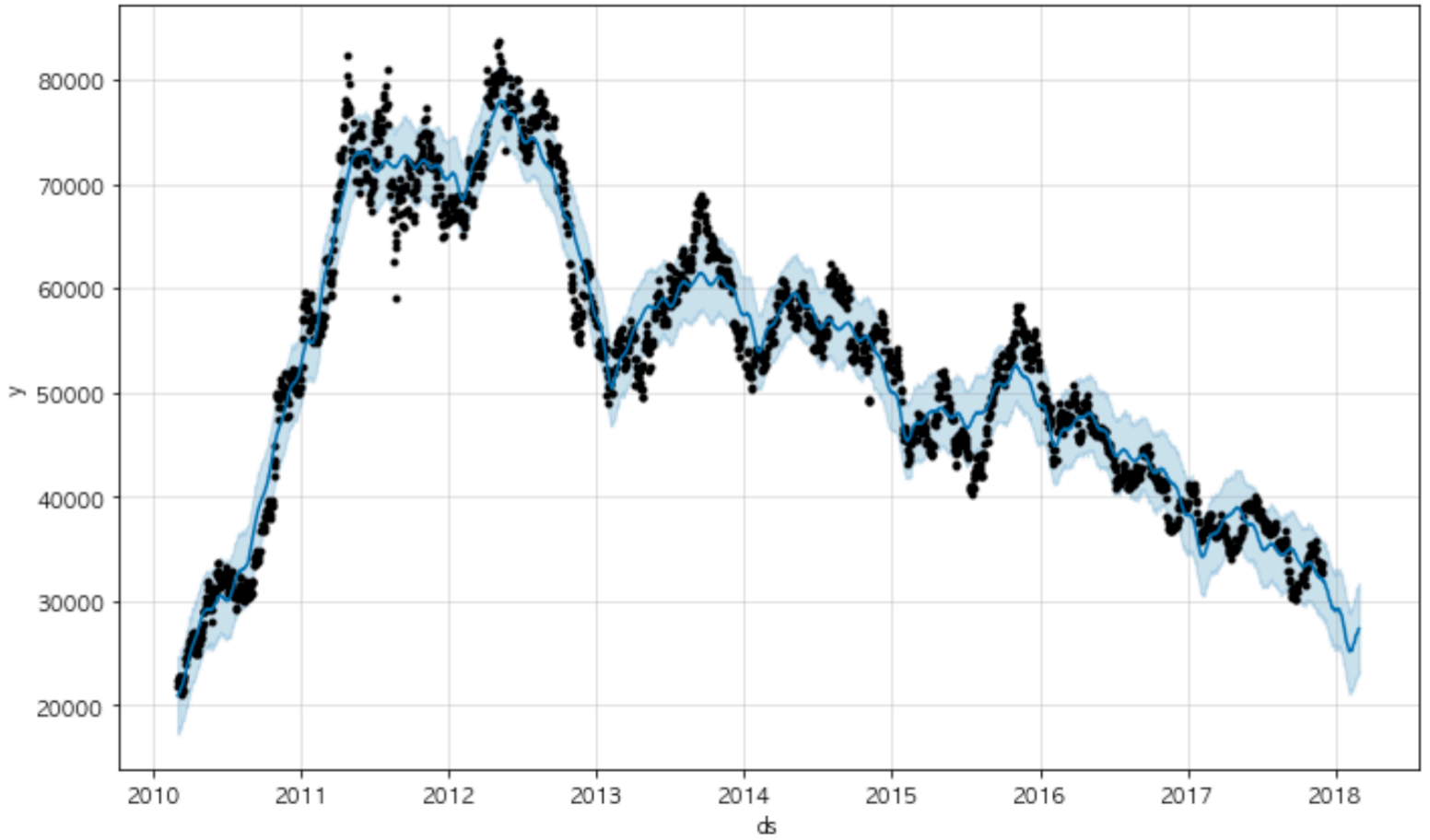
In [129]:
```
future = m.make_future_dataframe(periods=90)
forecast = m.predict(future)
forecast[["ds", "yhat", "yhat_lower", "yhat_upper"]].tail()
```

Out[129]:

	ds	yhat	yhat_lower	yhat_upper
2001	2018-02-24	26832.616679	22753.979451	31123.507621
2002	2018-02-25	26919.956402	22951.377356	31325.183463
2003	2018-02-26	27108.360356	22832.995913	31463.171359
2004	2018-02-27	27278.531780	23259.640683	31656.860081
2005	2018-02-28	27326.424186	23147.115541	31708.866097

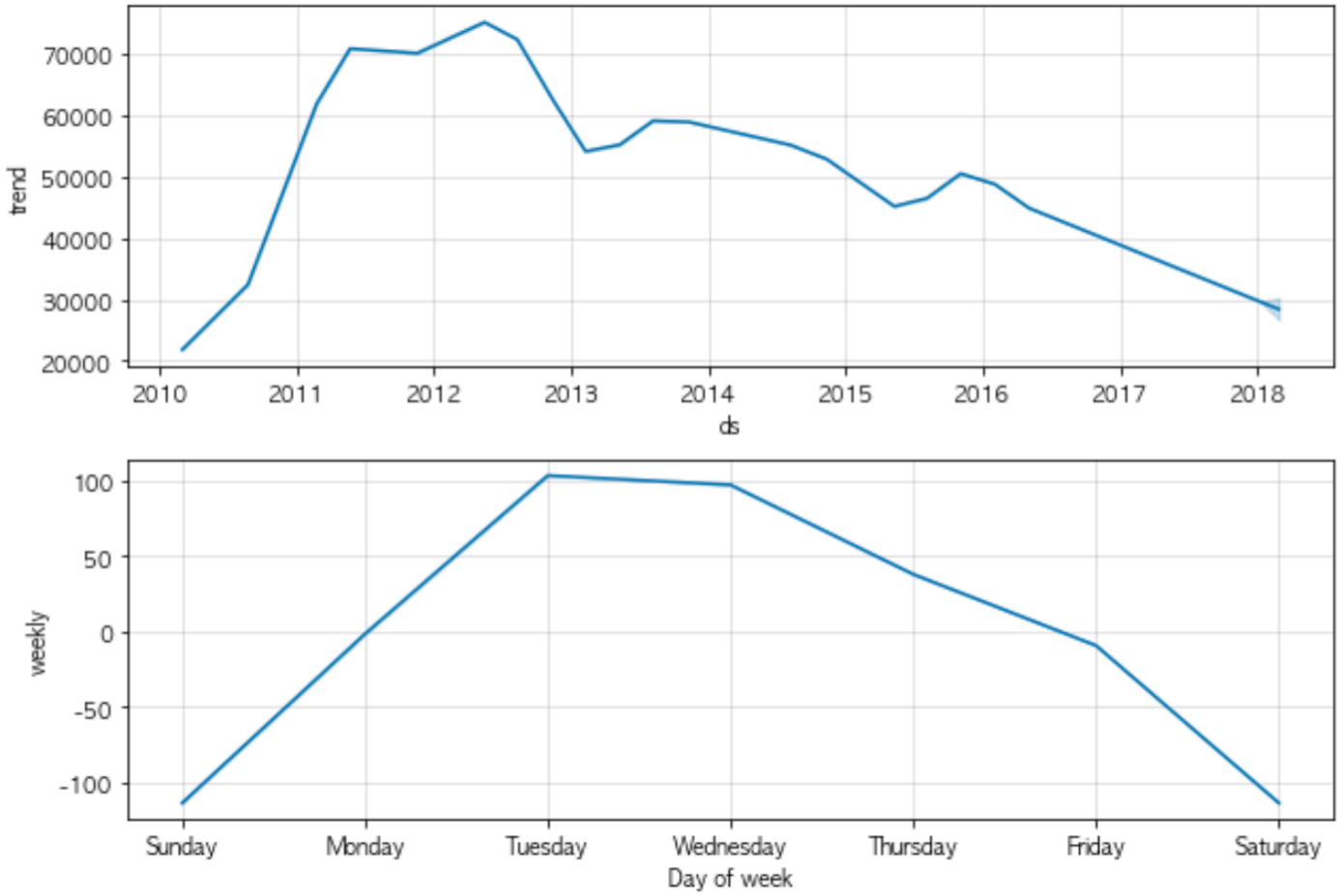
야후 **finance** 주식 데이터 **forecast**


```
In [130]: m.plot(forecast);
```

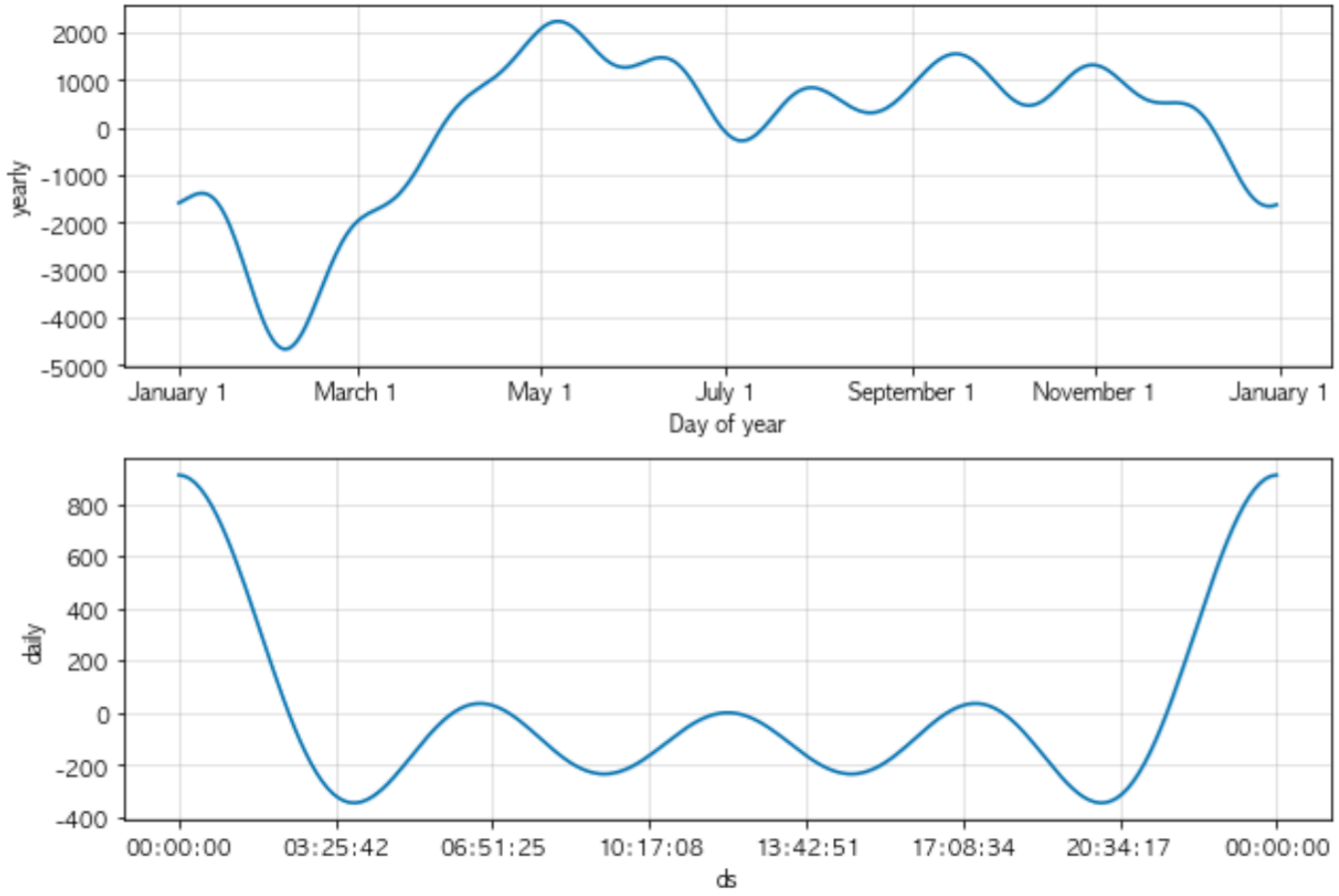


야후 **finance** 주식 데이터 **forecast**


```
In [131]: m.plot_components(forecast);
```



야후 **finance** 주식 데이터 **forecast**



야후 **finance** 주식 데이터 **forecast**

야후 **finance** 주식 데이터 **forecast**

In [132]:
```
plt.figure(figsize=(12, 6))
plt.plot(KIA.index, KIA["Close"], label="real")
plt.plot(forecast["ds"], forecast["yhat"], label="forecast")
plt.grid()
plt.legend()
plt.show()
```

하나 더 해보죠 ~ 대한항공

야후 finance 주식 데이터 forecast

In [133]:
```
# 003490 대한항공
start_date = "2010-03-01"
end_date = "2018-02-28"
KoreaAir = data.get_data_yahoo("003490.KS", start_date, end_date)
KoreaAir.head()
```

```
[**********100%**********]  1 of 1 completed
```

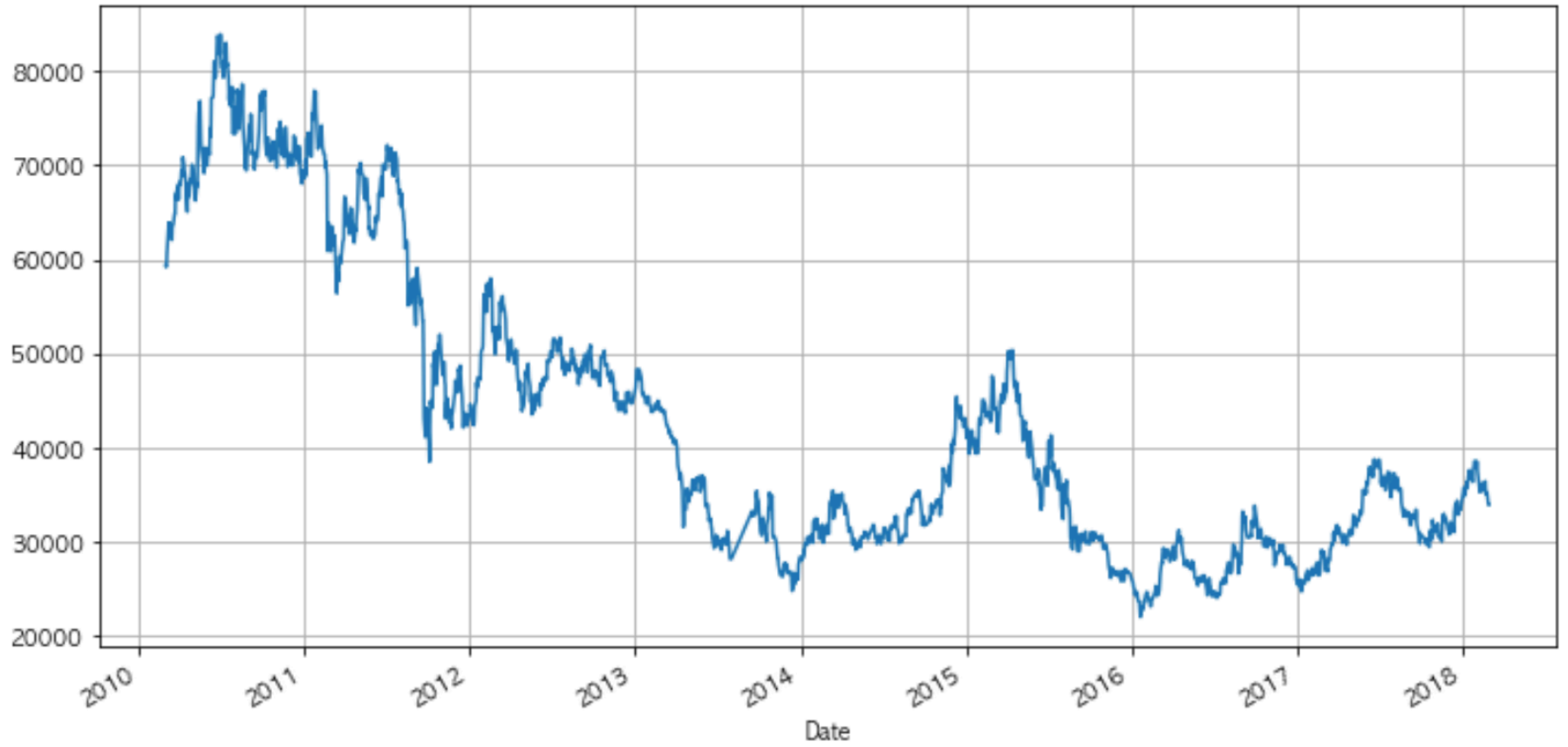
Out[133]:

Date	Open	High	Low	Close	Adj Close	Volume
2010-03-02	58192.625000	59390.417969	58192.625000	59390.417969	58454.984375	303805
2010-03-03	59090.968750	59490.230469	58691.707031	59190.785156	58258.496094	189214
2010-03-04	58991.152344	60188.941406	58891.335938	59290.601562	58356.742188	357038
2010-03-05	59290.601562	61087.285156	59290.601562	60588.207031	59633.906250	782451
2010-03-08	61286.917969	63083.601562	61087.285156	62784.152344	61795.265625	945708

야후 **finance** 주식 데이터 **forecast**

zero-base /

```
In [134]: KoreaAir["Close"].plot(figsize=(12, 6), grid=True);
```



야후 finance 주식 데이터 forecast

In [135]:
```
KoreaAir_trunc = KoreaAir[:"2017-11-30"]
KoreaAir_trunc.head()
```

Out[135]:

Date	Open	High	Low	Close	Adj Close	Volume
2010-03-02	58192.625000	59390.417969	58192.625000	59390.417969	58454.984375	303805
2010-03-03	59090.968750	59490.230469	58691.707031	59190.785156	58258.496094	189214
2010-03-04	58991.152344	60188.941406	58891.335938	59290.601562	58356.742188	357038
2010-03-05	59290.601562	61087.285156	59290.601562	60588.207031	59633.906250	782451
2010-03-08	61286.917969	63083.601562	61087.285156	62784.152344	61795.265625	945708

야후 **finance** 주식 데이터 **forecast**

In [136]:
```
df = pd.DataFrame({"ds": KoreaAir_trunc.index, "y": KoreaAir_trunc["Close"]})
df.reset_index(inplace=True)
del df["Date"]
df.head()
```

Out[136]:

	ds	y
0	2010-03-02	59390.417969
1	2010-03-03	59190.785156
2	2010-03-04	59290.601562
3	2010-03-05	60588.207031
4	2010-03-08	62784.152344

야후 finance 주식 데이터 forecast

In [137]:
```
m = Prophet(yearly_seasonality=True, daily_seasonality=True)
m.fit(df);
```

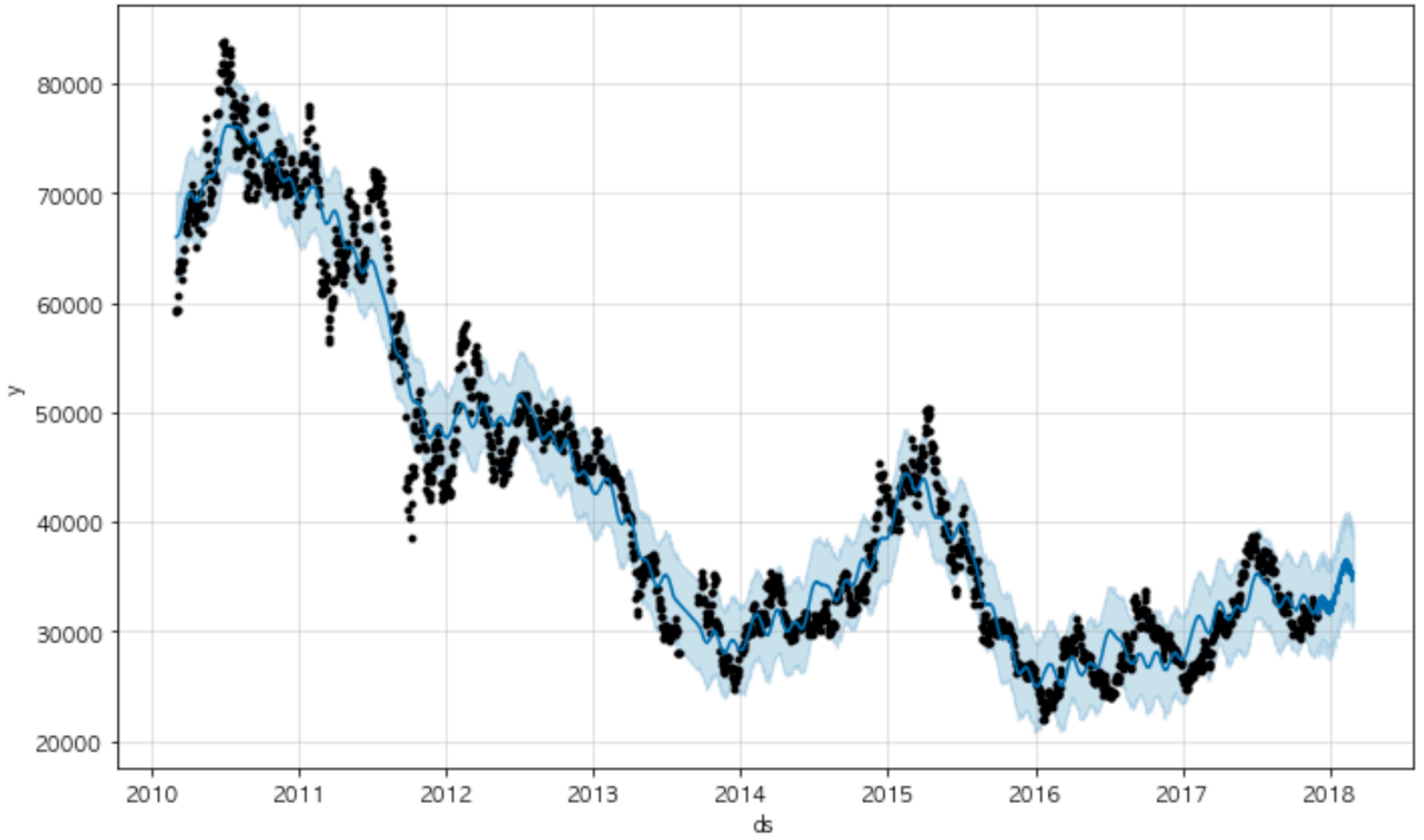
In [138]:
```
future = m.make_future_dataframe(periods=90)
forecast = m.predict(future)
forecast[["ds", "yhat", "yhat_lower", "yhat_upper"]].tail()
```

Out[138]:

	ds	yhat	yhat_lower	yhat_upper
1970	2018-02-24	34674.234793	30423.640236	38749.043687
1971	2018-02-25	34573.220496	30533.187982	39062.403088
1972	2018-02-26	35541.290267	31074.178590	39880.715713
1973	2018-02-27	35377.823818	31153.878030	39513.168489
1974	2018-02-28	35309.225433	31049.022779	39474.995897

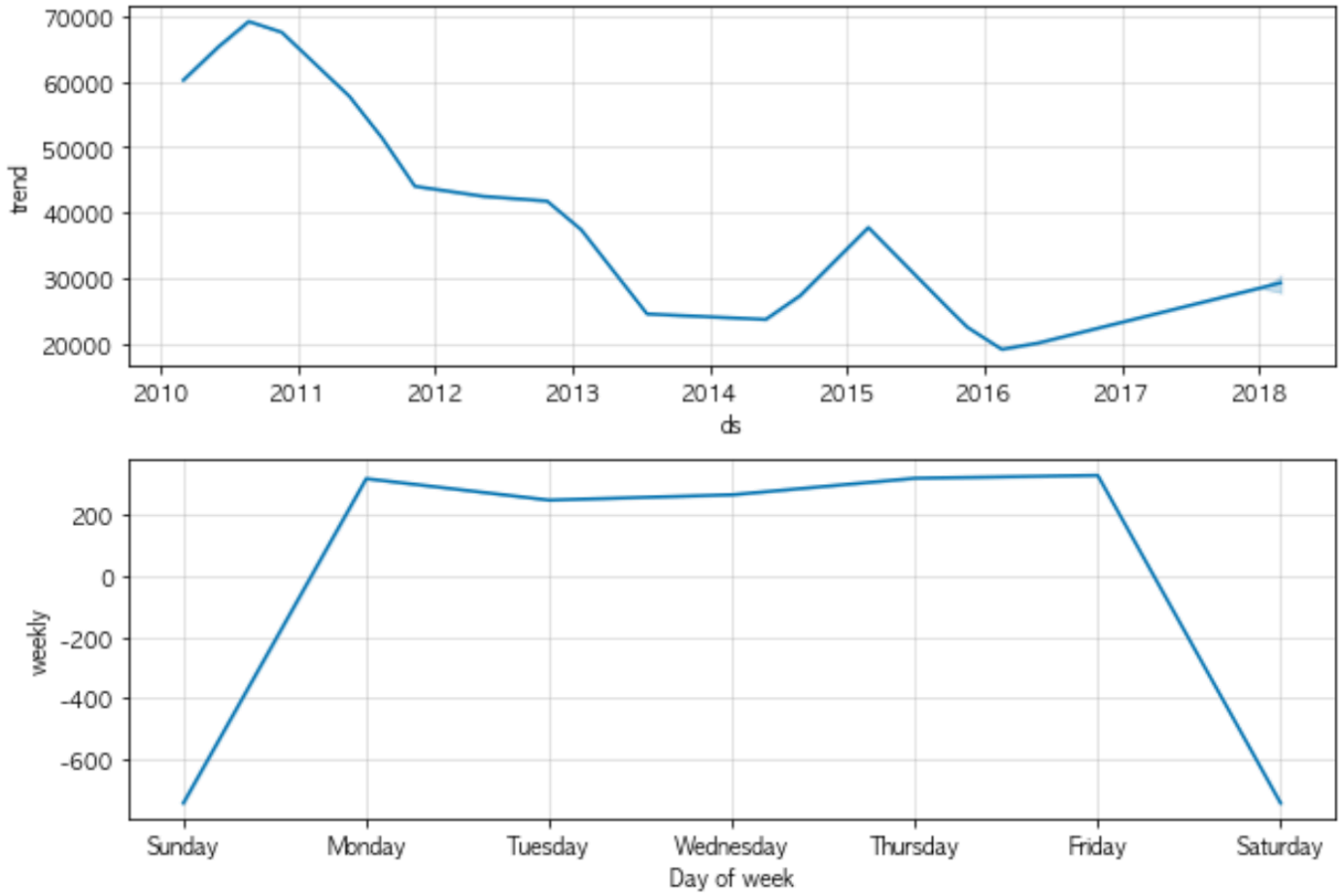
야후 **finance** 주식 데이터 **forecast**


```
In [139]: m.plot(forecast);
```

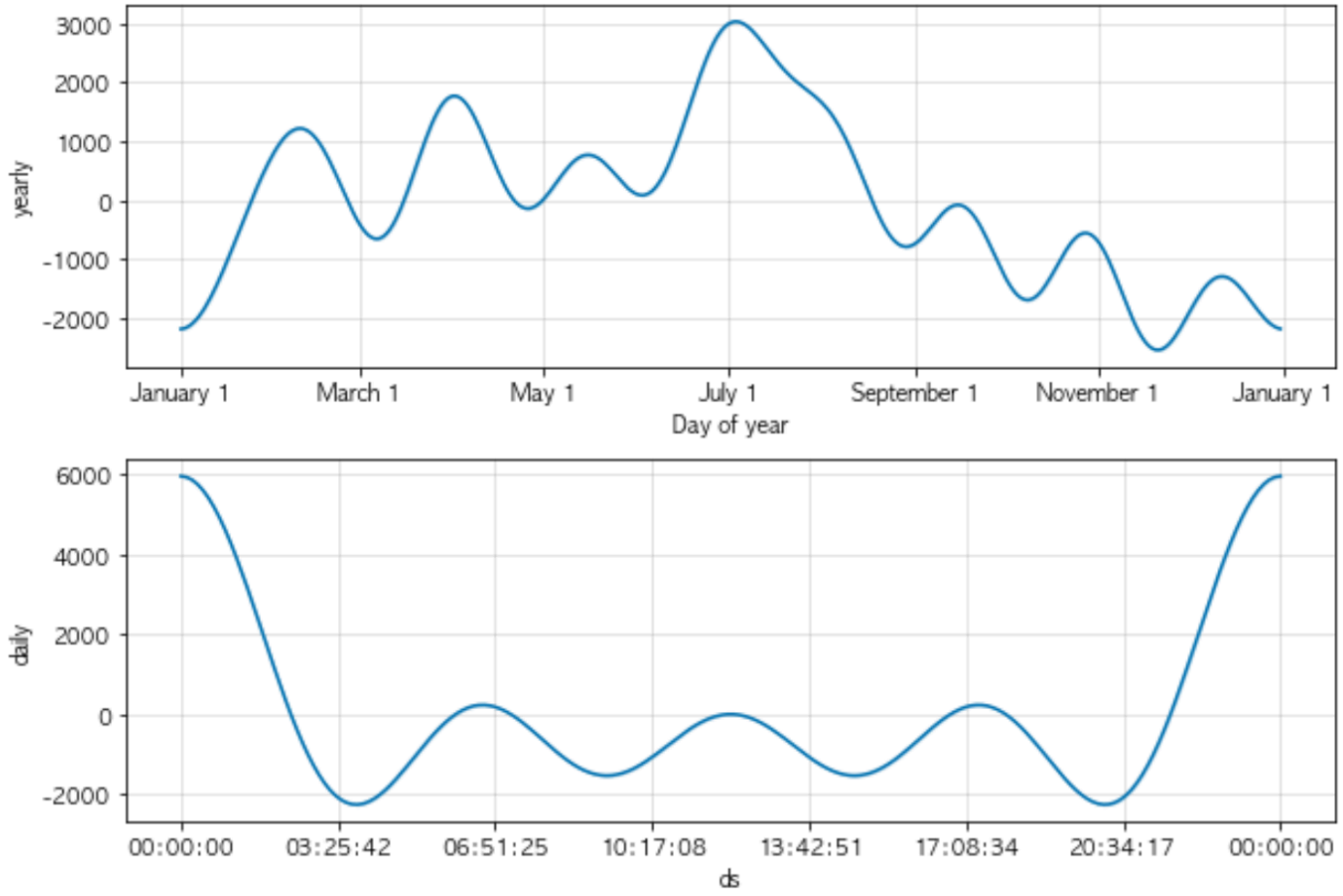


야후 **finance** 주식 데이터 **forecast**


```
In [140]: m.plot_components(forecast);
```



야후 **finance** 주식 데이터 **forecast**



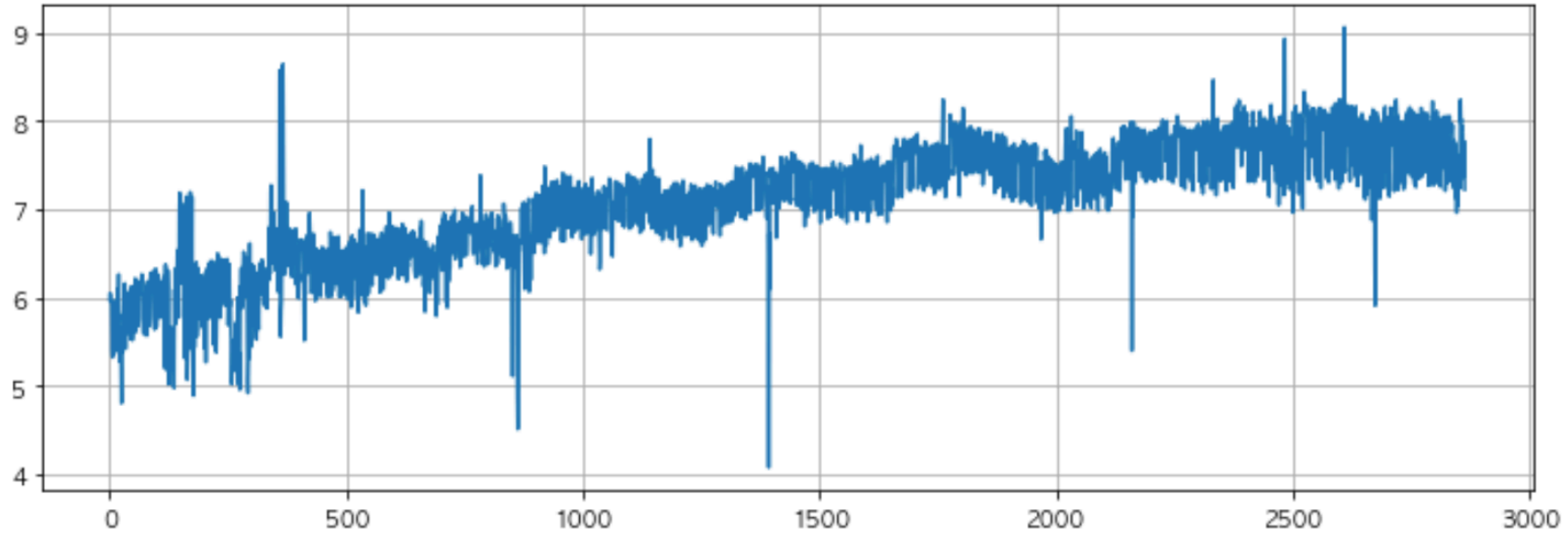
야후 **finance** 주식 데이터 **forecast**

In [141]:
```
plt.figure(figsize=(12, 6))
plt.plot(KoreaAir.index, KoreaAir["Close"], label="real")
plt.plot(forecast["ds"], forecast["yhat"], label="forecast")
plt.grid()
plt.legend()
plt.show()
```

조금 특이한 형태의 데이터에 대한 forecast

조금 특이한 형태의 데이터 **forecast**


```
In [142]: df = pd.read_csv("../data/05_example_wp_R2.csv", index_col=0)
          df["y"].plot(figsize=(12, 4), grid=True);
```



- Logistic 성장형 그래프를 가진 데이터에 대한 Forecast

조금 특이한 형태의 데이터 **forecast**

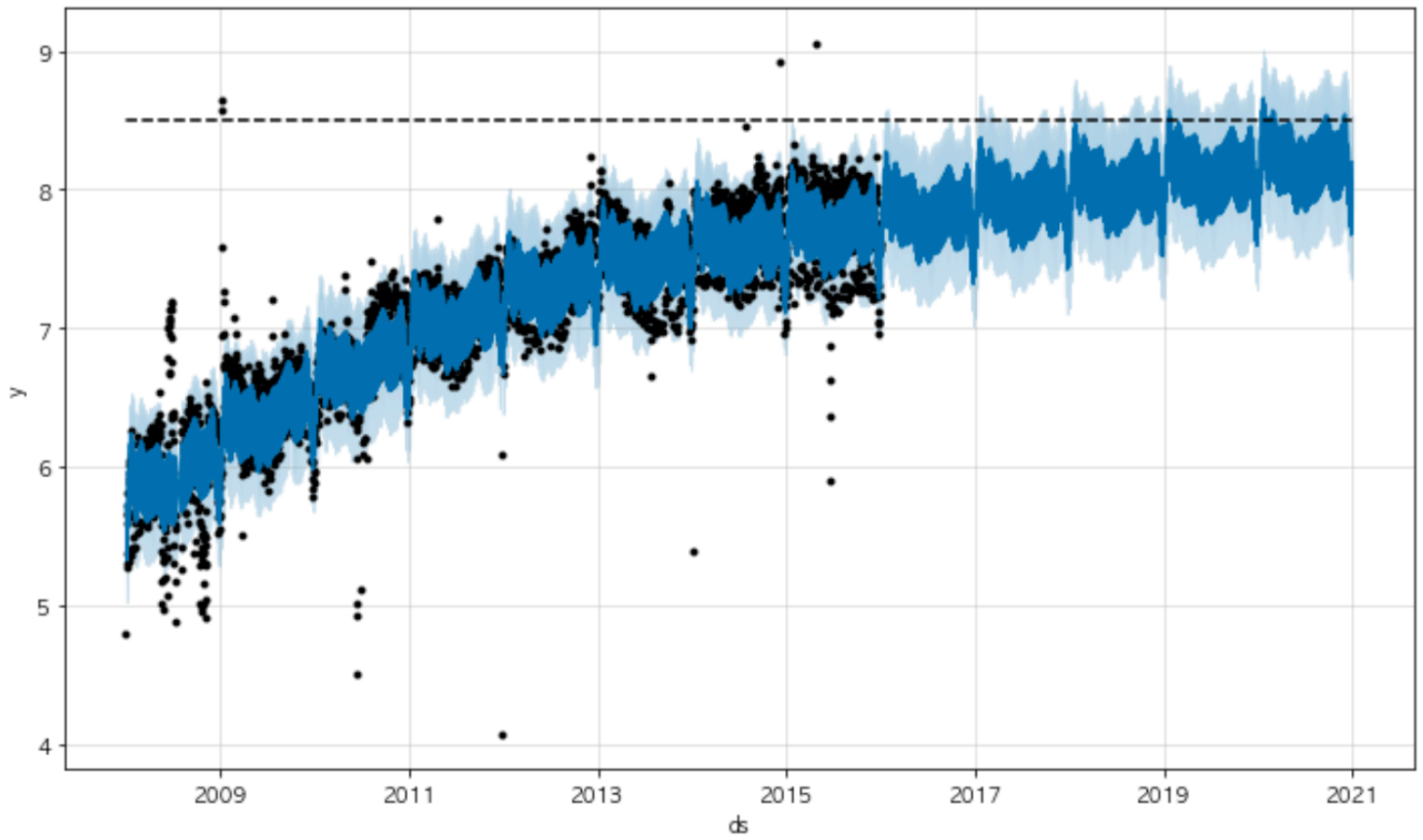

```
In [143]: df["cap"] = 8.5
```

```
In [144]: m = Prophet(growth="logistic", daily_seasonality=True)
          m.fit(df)
```

```
Out[144]: <fbprophet.forecaster.Prophet at 0x7f8c397bd6a0>
```

조금 특이한 형태의 데이터 **forecast**

In [145]:
```
future = m.make_future_dataframe(periods=1826)
future["cap"] = 8.5
fcst = m.predict(future)
m.plot(fcst);
```



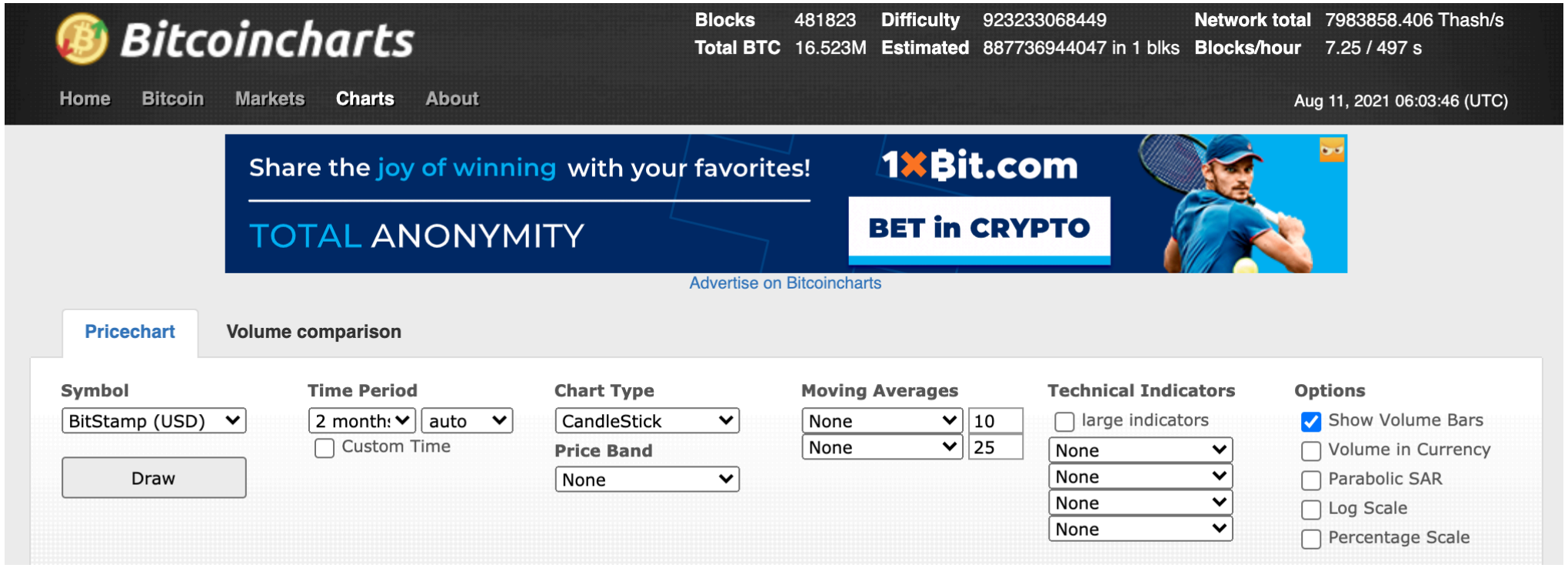
비트코인 데이터 fbprophet으로 분석하기

하나 더 해보자 ~~

요즘 그렇게도 유행한다는

비트 코인 ~~~~~

비트코인 데이터 forecast



출처: https://bitcoincharts.com/charts/bitstampUSD#rg60ztgSzm1g10zm2g25zv

- 비트 코인 정보를 제공하는 사이트이다...

비트코인 데이터 **forecast**

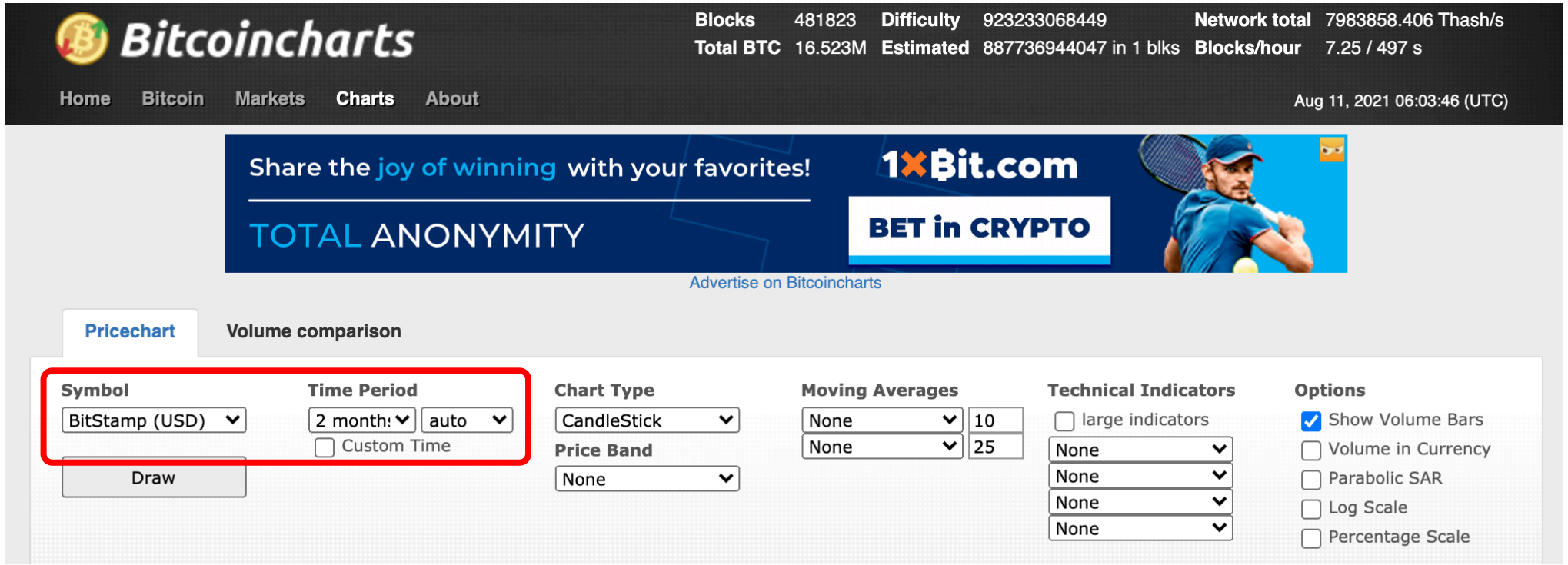
- 다양한 정보를 제공하고 있으며
- raw data도 얻을 수 있다

비트코인 데이터 **forecast**

Load raw data

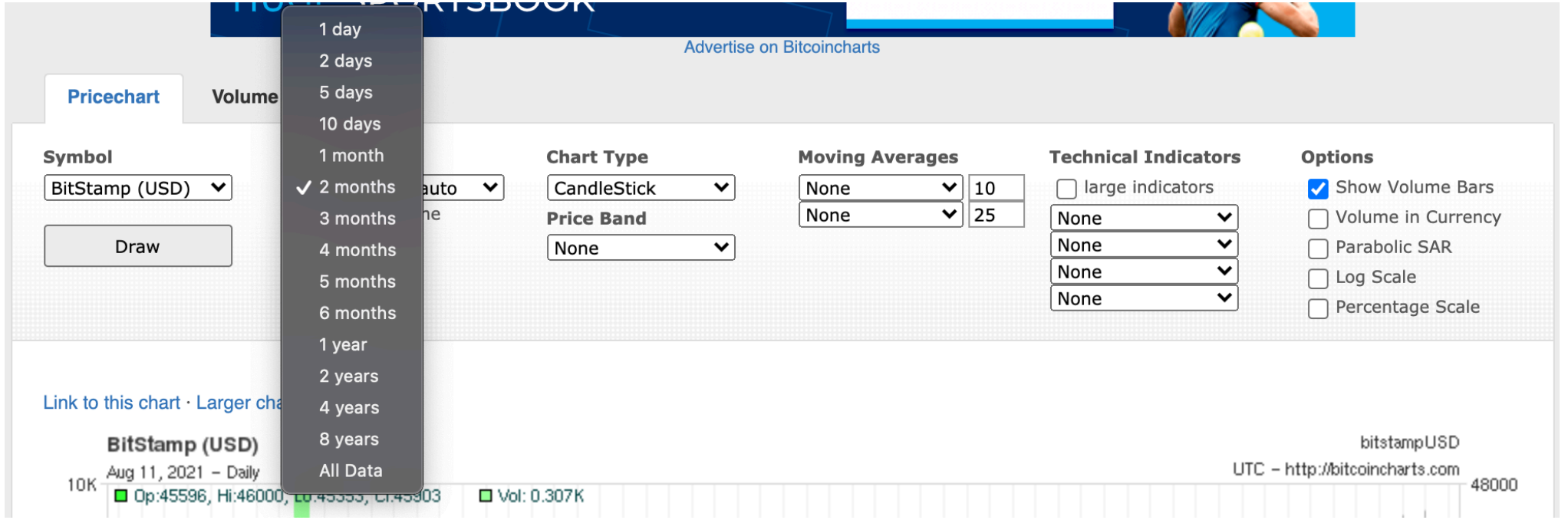
- 여기를 선택하면
- raw data 를 table 형태로 열람할 수 있다

비트코인 데이터 forecast



- 여기의 내용을 바꾸고 Draw 버튼을 누르면 접근 주소가 바뀐다.
- 그 주소를 저장해 두자

비트코인 데이터 **forecast**



- Time Period 를 2달, 3달, 4달… 2년, 4년, 8년…변경하면
- 그에 따라 URL 정보가 바뀐다

비트코인 데이터 **forecast**

https://bitcoincharts.com/charts/bitstampUSD#rg30ztgSzm1g10zm2g25zv

https://bitcoincharts.com/charts/bitstampUSD#rg60ztgSzm1g10zm2g25zv

https://bitcoincharts.com/charts/bitstampUSD#rg360ztgSzm1g10zm2g25zv

https://bitcoincharts.com/charts/bitstampUSD#rg730ztgSzm1g10zm2g25zv

- 이렇게!
- 조회 기간에 따라 주소값이 변경된다

비트코인 데이터 **forecast**

In [146]:
```
from selenium import webdriver
from bs4 import BeautifulSoup

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from fbprophet import Prophet
from selenium.webdriver.common.action_chains import ActionChains

import time

get_ipython().run_line_magic("matplotlib", "inline")
```

- 필요 모듈 import 하고 ~~

비트코인 데이터 **forecast**

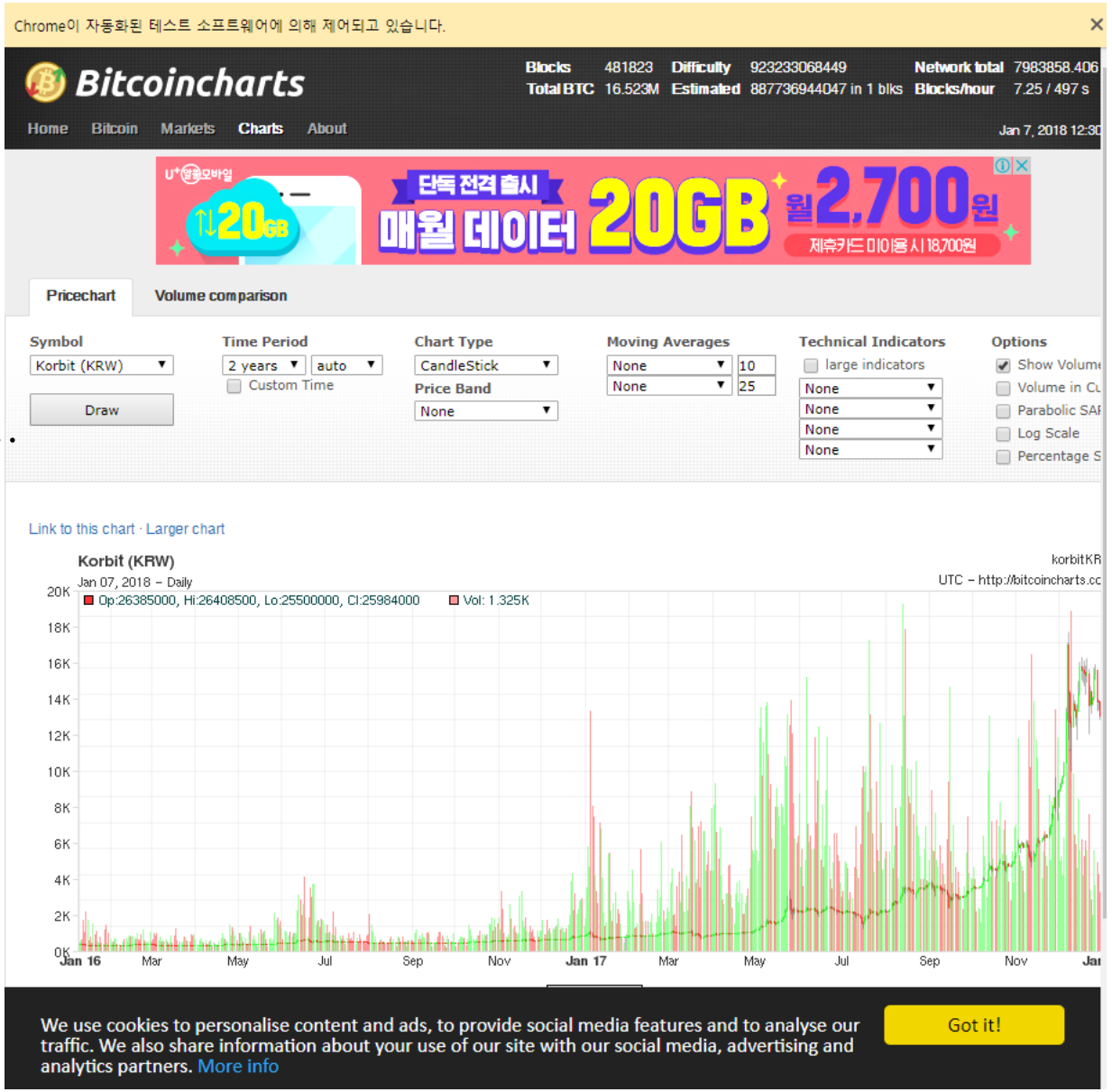

```
In [147]: driver = webdriver.Chrome("../driver/chromedriver")

In [148]: driver.get("https://bitcoincharts.com/charts/korbitKRW#rg730ztgSzm1g10zm2g25zv")
```

- 크롬 드라이버 불러주고 ~~~
- 2년 짜리 URL 주소값으로 접근하자
- https://bitcoincharts.com/charts/bitstampUSD#rg730ztgSzm1g10zm2g25zv

비트코인 데이터 **forecast**

- 그런데 raw_data 메뉴가 가려짐...
- 사람이라면 스크롤 하겠지...
- Selenium은???
- Selenium도 스크롤 할줄 안다^^



비트코인 데이터 **forecast**


```
In [23]: xpath = """//*[@id="content_chart"]/div/div[2]/a"""
         variable = driver.find_element_by_xpath(xpath)
         driver.execute_script("return arguments[0].scrollIntoView();", variable)
         variable.click()
```

- 해당 Xpath가 나타날때까지 화면을 스크롤하라는 명령

비트코인 데이터 **forecast**

Load raw data

Timestamp	Open	High	Low	Close	Volume (BTC)	Volume (Currency)	Weighted Price
2019-08-13 00:00:00	12200000	12249500	11121000	11607500	43.3	512954843.93	11847632.56
2019-08-14 00:00:00	11600000	11800000	10961000	11360000	44.68	503512451.94	11269292.95
2019-08-15 00:00:00	11529500	11530000	10450000	11176500	54.85	604270626.36	11016978.43
2019-08-16 00:00:00	11250000	12842500	10825500	12537500	151.43	1867206817.02	12330150.32
2019-08-17 00:00:00	12537500	12700000	12220000	12408500	186.1	2325145466.69	12494339.16
2019-08-18 00:00:00	12444500	12680000	12235500	12502500	164.01	2042755221.14	12455440.42
2019-08-19 00:00:00	12490500	13122500	12414000	13122500	228.82	2936759784.42	12834332.13
2019-08-20 00:00:00	13122500	13191000	12742500	12936500	185.56	2403044839.86	12950145.61

- 저 표를 읽어오면 되겠다 ~~~^^
- 이번엔 웹 페이지에 table이 여러 개 이다.
- 다행히 table에 class가 data로 되어 있다.

비트코인 데이터 **forecast**

In [150]:
```
html = driver.page_source

soup = BeautifulSoup(html, "html.parser")
table = soup.find("table", "data")
table
```

```
</tr>
</thead>
<tbody>
<tr><td>2019-07-24 00:00:00</td><td>10113000</td><td>10185000</td><td>8
d>1374203764.81</td><td>9723447.59</td></tr><tr><td>2019-07-25 00:00:00
9000</td><td>9299000</td><td>72.1</td><td>670772289.74</td><td>9303746.
<td>9270500</td><td>9333000</td><td>9200500</td><td>9217500</td><td>29.
td></tr><tr><td>2019-07-27 00:00:00</td><td>9220500</td><td>10500000</t
td><td>977499836.35</td><td>9863935.95</td></tr><tr><td>2019-07-28 00:0
>9255000</td><td>9611500</td><td>86.5</td><td>830955419.76</td><td>9606
</td><td>9576000</td><td>9781000</td><td>9518000</td><td>9530500</td><t
```

비트코인 데이터 **forecast**

In [219]:
```
df = pd.read_html(str(table))
bitcoin = df[0]
bitcoin.head()
```

Out[219]:

	Timestamp	Open	High	Low	Close	Volume (BTC)	Volume (Currency)	Weighted Price
0	2016-03-28 00:00:00	500600	502100	499000	500000	507.10	2.539829e+08	500855.33
1	2016-03-29 00:00:00	500000	500500	485500	489000	943.14	4.673012e+08	495473.89
2	2016-03-30 00:00:00	488900	491000	480000	482500	873.16	4.236390e+08	485178.81
3	2016-03-31 00:00:00	482500	487900	482000	484000	945.63	4.586144e+08	484982.89
4	2016-04-01 00:00:00	484000	485700	483100	485600	665.88	3.223434e+08	484086.51

- 매우 잘 ~~ 읽었다 ^^

비트코인 데이터 forecast

In [152]:
```
bitcoin.to_csv("../data/05_bitcoin_history.csv", sep=",")
```

In [153]:
```
bitcoin = pd.read_csv("../data/05_bitcoin_history.csv", index_col=0)
bitcoin.head()
```

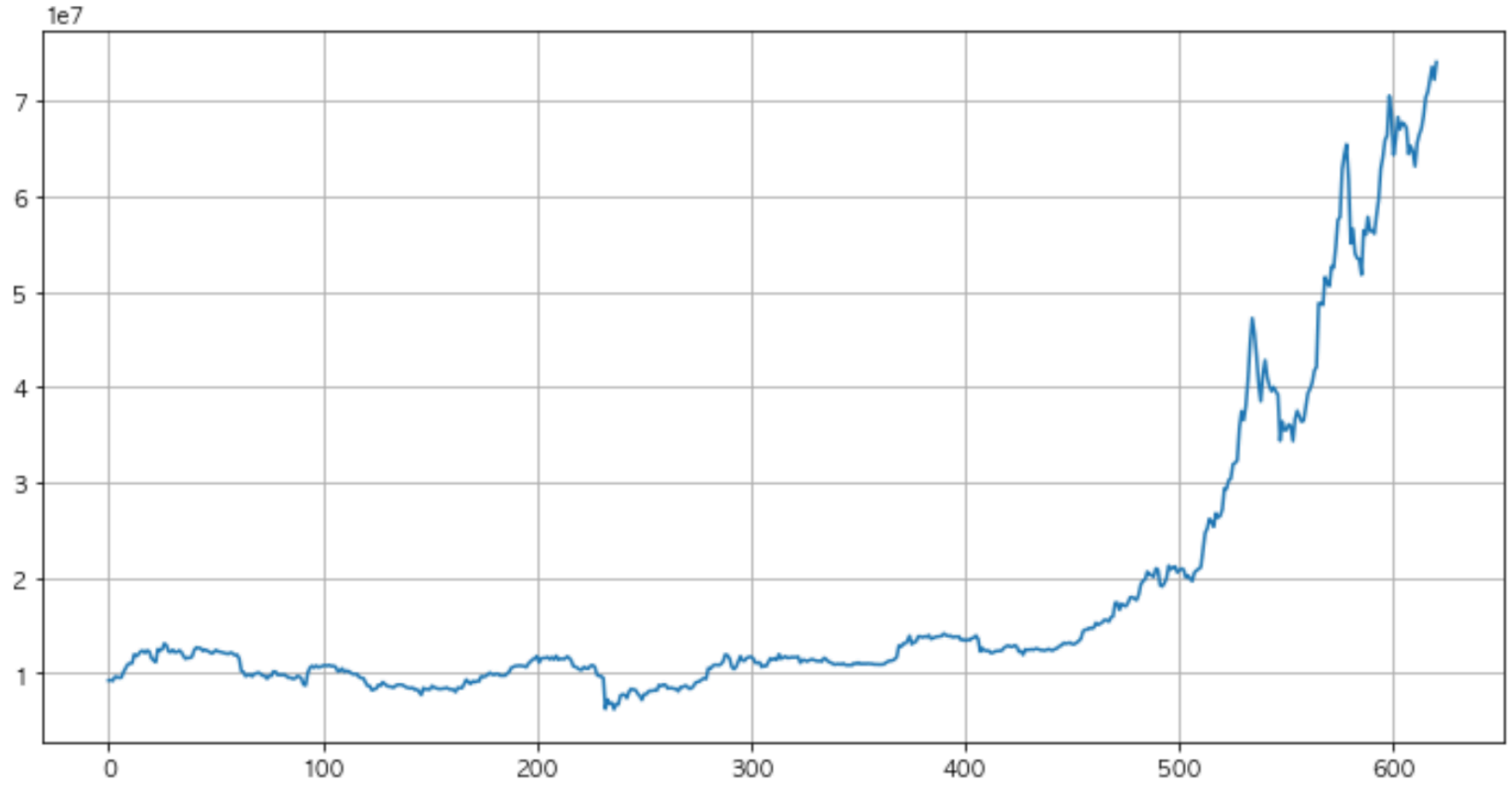
Out[153]:

	Timestamp	Open	High	Low	Close	Volume (BTC)	Volume (Currency)	Weighted Price
0	2019-07-24 00:00:00	10113000	10185000	8999000	9255000	141.33	1.374204e+09	9723447.59
1	2019-07-25 00:00:00	9343500	9500000	9159000	9299000	72.10	6.707723e+08	9303746.14
2	2019-07-26 00:00:00	9270500	9333000	9200500	9217500	29.04	2.689692e+08	9261046.33
3	2019-07-27 00:00:00	9220500	10500000	9220500	9650000	99.10	9.774998e+08	9863935.95
4	2019-07-28 00:00:00	9697500	9892500	9255000	9611500	86.50	8.309554e+08	9606011.76

- 정신건강을 위한 저장을 한 다음에…
- 다시 읽어와보면…
- 매우 잘 ~~ 읽었다 ^^

비트코인 데이터 **forecast**


```
In [154]: bitcoin["Close"].plot(figsize=(12, 6), grid=True);
```



비트코인 데이터 **forecast**


```
In [155]: df = pd.DataFrame({"ds": bitcoin["Timestamp"], "y": bitcoin["Close"]})
          m = Prophet(yearly_seasonality=True, daily_seasonality=True)
          m.fit(df);
```

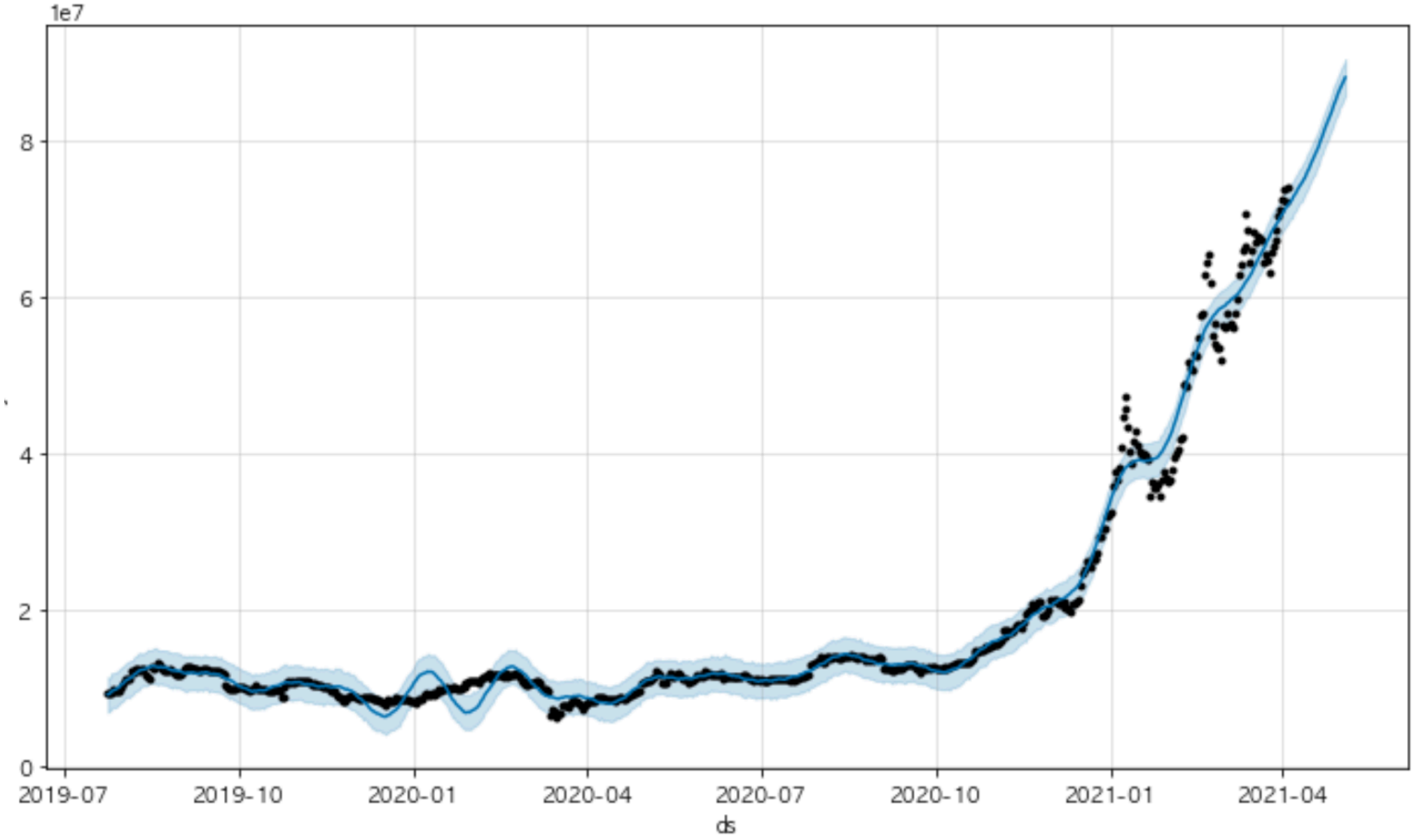
- 분석하고 싶은 항목(Close)만 가지고, Prophet을 적용하자 ~~

비트코인 데이터 **forecast**


```
In [156]: future = m.make_future_dataframe(periods=30)
          forecast = m.predict(future)
          m.plot(forecast);
```

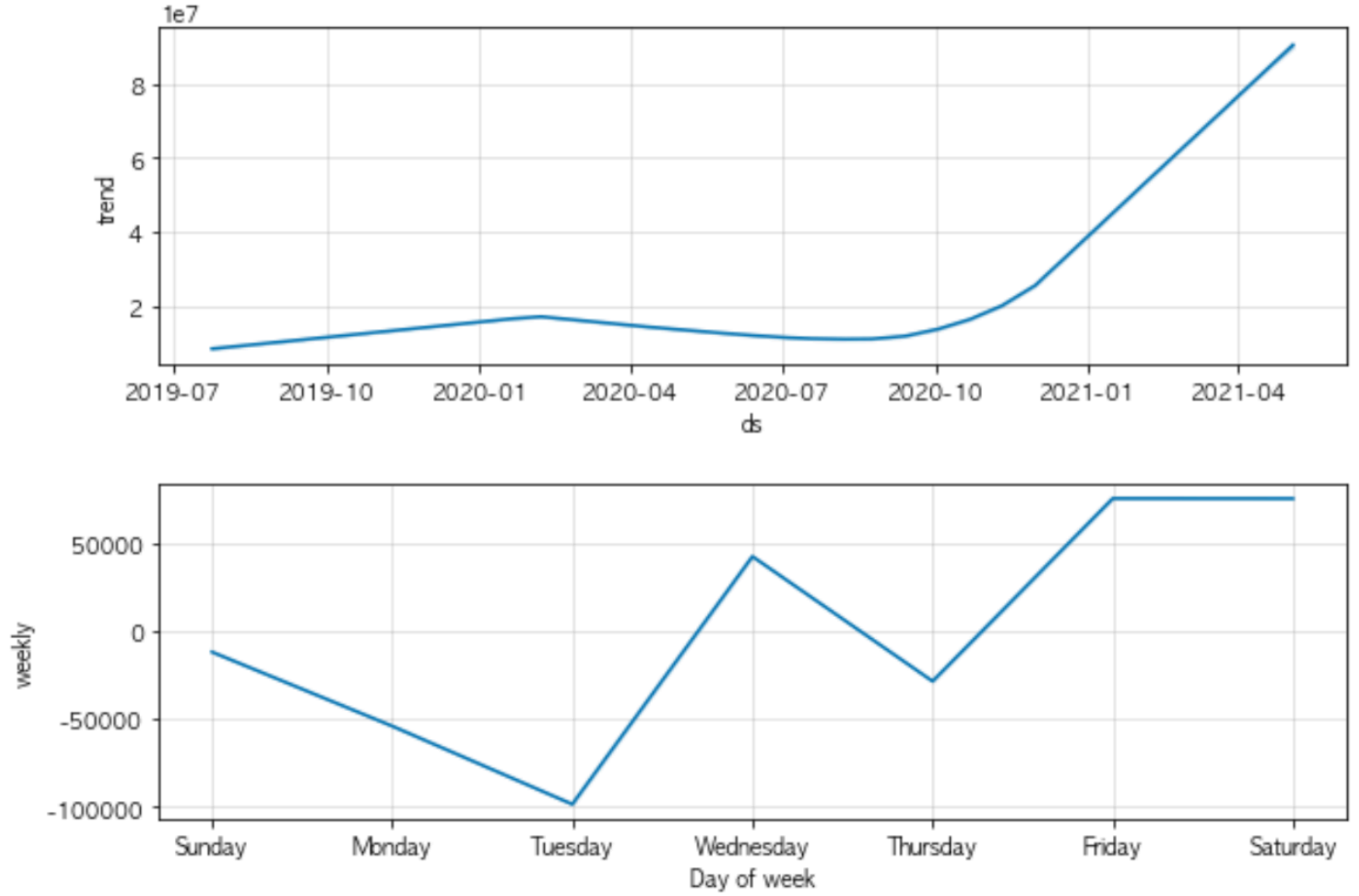
- 향후 30일간의 Forecast를 조사해보자 !

비트코인 데이터 **forecast**



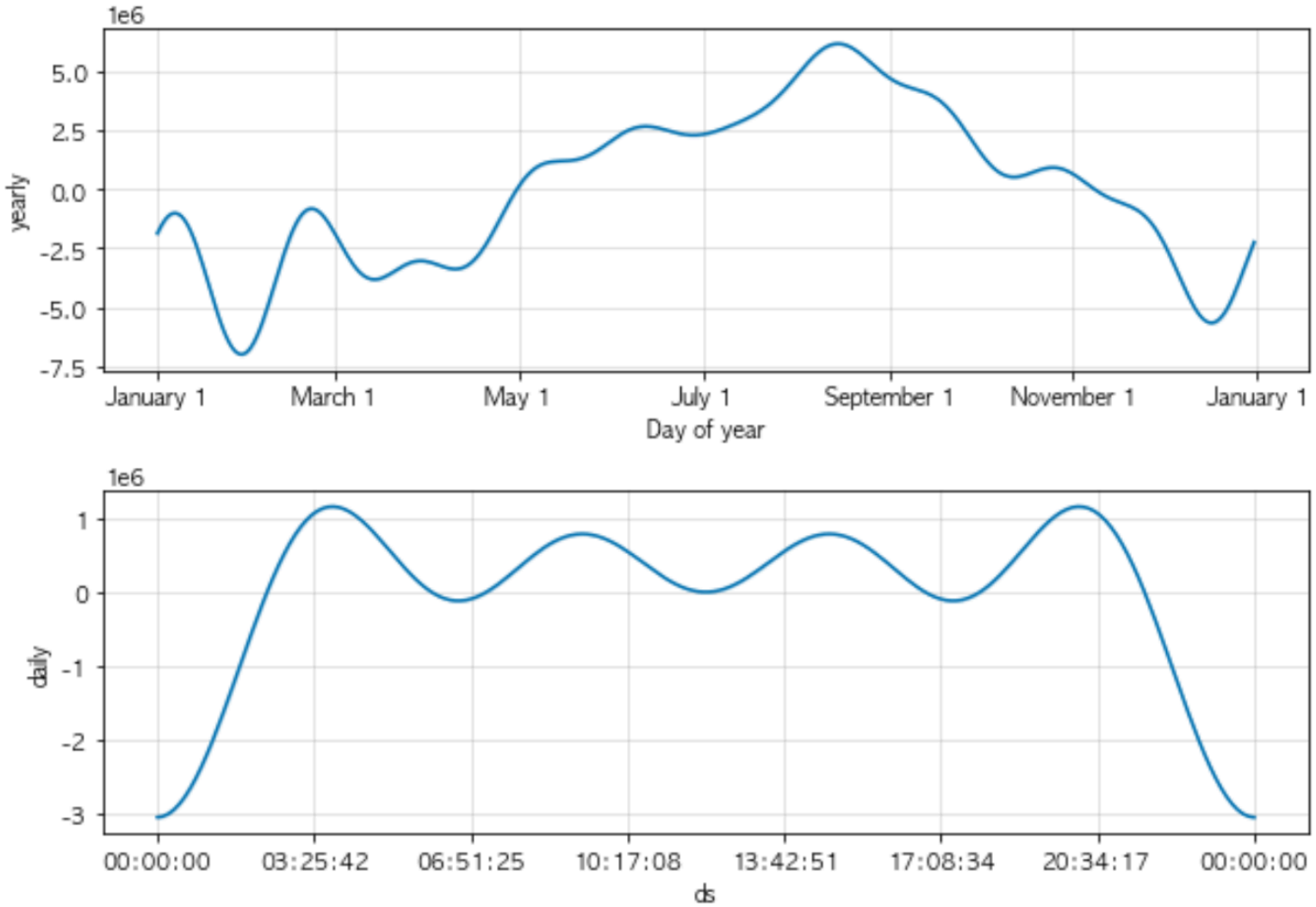
비트코인 데이터 **forecast**


```
In [157]: m.plot_components(forecast);
```



- 트렌드

비트코인 데이터 **forecast**



비트코인 데이터 **forecast**


```
In [158]: driver.close()
```

- 잊지 말자 driver.close() 🙂

화요일이 유난히 낮은 편이다.

단...

재미삼아 진행한 것으로 절대 의미를 두지 말자...

비트 코인은 주식보다 더 어렵고 위험하니까....!!!!