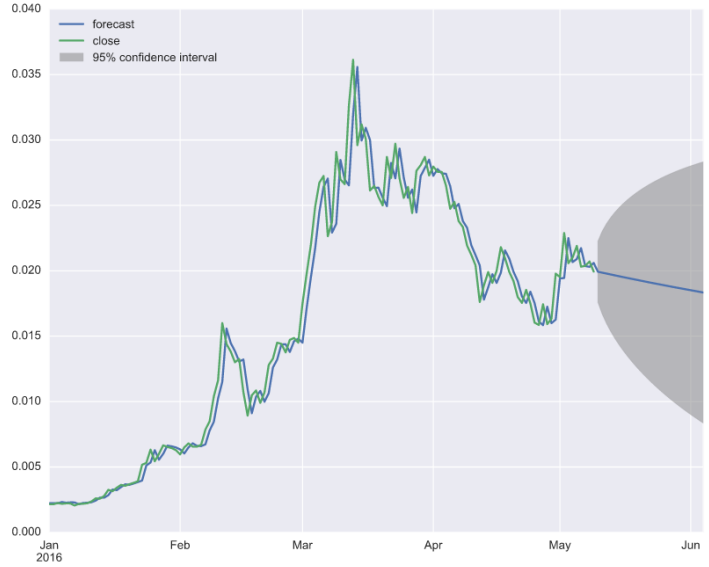

python

시계열 분석

- 시계열 분석은 ***회귀분석***을 기반
  - 회귀분석은 시간(시점)을 고려하지 않지만, 시계열은 시간을 고려
  - 1종류의 데이터라 하더라도, 획득 시간을 안다면 2개의 데이터를 갖는 결과
- ***규칙성***을 가지는 패턴과 ***불규칙***한 패턴의 결합
  - 규칙성 패턴
    - ***자기상관성***(Autocorrelativeness): 이전 결과와 이후 결과에 발생
  - 불규칙성 패턴
    - White Noise: 평균이 0이며 일정한 분산을 지닌 정규분포에서 추출된 임의의 수치



시계열 분석

- 모형
  - 일변량 정상 시계열 모형
    - ***이동평균법*** (Moving Average Model: MA)
    - ***자기회귀*** (Auto regression: AR), 스펙트럼 분석, 조건부 이분산성
    - AR + MR → ***자기회귀 이동평균모형*** (AutoRegressive Moving Average model; ARMA)
- ***추세***(Trend), ***순환***(Cycle), ***계절변동***(Seasonal Variation), ***불규칙변동***(Irregular fluctuation)
  - 분석을 시작하기 전에 자료의 계절성과 주기성, 순환성 파악을 위해 분산분석과 다중비교를 통해 계절효과가 있는지 확인해야 한다.

Prophet

- Facebook에서 만든 ***시계열 예측 라이브러리***
  - https://facebook.github.io/prophet/
  - Python, R로 사용 가능
- 통계적 지식이 없어도 직관적 파라미터를 통해 모형을 조정 가능
- 일반적인 경우 기본값만 사용해도 높은 성능을 보여줌
- 내부가 어떻게 동작하는지 고민할 필요가 없음

- 설치

  $ ***pip install fbprophet***

  $ ***pip install –upgrade plotly***

  (Jupyter Notebook) ***conda install –c conda-forge fbprophet***

Prophet

- 구성요소
  - ***Growth, Seasonality, Holidays***

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

- ***Growth***
  - Linear Growth(+Change Point)
    - Change Point는 자동으로 탐지
    - 예측할 때는 특정 지점이 change point인지 여부를 확률적으로 결정
  - Non-Linear Growth(Logistic Growth)
    - 자연적 상한이 존재하는 경우, Capacity가 있음
    - Capacity는 시간에 따라 변할 수 있음

Prophet

- ***Seasonality***
  - 사용자들의 행동 양식으로 주기적으로 나타나는 패턴
    - 방학, 휴가, 온도, 주말 등등
  - 퓨리에 급수(Fourier Series)를 이용해 패턴의 근사치를 찾음
    - 같은 형태를 반복하는 주기를 가진 파동은, 아무리 복잡한 것이라도 단순한 파동이 잔뜩 결합

- ***Holidays***
  - 주기성을 가지진 않지만 전체 추이에 큰 영향을 주는 이벤트가 존재
  - 이벤트의 효과는 독립적이라 가정
  - 이벤트 앞뒤로 window 범위를 지정해 해당 이벤트가 미치는 영향의 범위를 설정할 수 있음

Prophet

- ***Model Fitting***
  - Stan을 통해 모델을 학습
    - probabilistic programming language for statistical inference
  - 2가지 방식
    - MAP (Maximum A Posteriori) : Default, 속도가 빠름
    - MCMC (Markov Chain Monte Carlo) : 모형의 변동성을 더 자세히 살펴볼 수 있음
  - Analyst in the loop Modeling
    - 통계적 지식이 없어도 직관적 파라미터를 통해 모형을 조정할 수 있음
    - 일반적인 경우 기본값만 사용해도 높은 성능을 가능
    - 내부가 어떻게 동작하는지 고민할 필요가 없음
    - 요소
      - Capacities : 시계열 데이터 전체의 최대값
      - Change Points : 추세가 변화하는 시점
      - Holidays & Seasonality : 추세에 영향을 미치는 시기적 요인
      - Smoothing : 각각의 요소들이 전체 추이에 미치는 영향의 정도

Prophet

1) 데이터를 Prophet에 맞도록 가공
2) Prophet 객체 생성 → Fit
3) 미래 Dataframe 생성
4) 예측
5) Forecast 결과 확인
6) 시각화

Prophet

1) 데이터를 Prophet에 맞도록 가공

2) Prophet 객체 생성 → Fit

3) 미래 Dataframe 생성

4) 예측

5) Forecast 결과 확인

6) 시각화

```
import pandas as pd
from fbprophet import Prophet

df = pd.read_csv("./temp_2018_train.csv")
```

- 예측 값의 상한 하한 제어 → cap, floor

```
df['cap']=20
df['floor']=-5
```

Prophet

1) 데이터를 Prophet에 맞도록 가공
2) ***Prophet 객체 생성 → Fit***
3) 미래 Dataframe 생성
4) 예측
5) Forecast 결과 확인
6) 시각화

- Prophet 객체는 1번만 Fit 할 수 있음
  - 여러 번 Fit 하고 싶으면 새로운 객체를 생성

```
m = Prophet() # Default growth='linear'
m.fit(df)
```

- 상한과 하한을 설정할 경우
  - Prophet 객체를 생성할 때 growth='logistic' 추가

```
m = Prophet(growth='logistic')
m.fit(df)
```

Prophet

1) 데이터를 Prophet에 맞도록 가공
2) Prophet 객체 생성 → Fit
3) ***미래 Dataframe 생성***
4) 예측
5) Forecast 결과 확인
6) 시각화

```
future = m.make_future_dataframe(periods=365)
future.tail()
```

- 상한과 하한을 설정 했을 경우
  - Prophet 객체를 생성할 때와 동일하게 지정

```
future['cap'] = 6
future['floor'] = 1.5
```

Prophet

1) 데이터를 Prophet에 맞도록 가공
2) Prophet 객체 생성 → Fit
3) 미래 Dataframe 생성
4) ***예측***
5) ***Forecast 결과 확인***
6) 시각화

```
forecast = m.predict(future)
forecast.tail()
```

- yhat_lower, yhat_upper 같이 범위로 제공

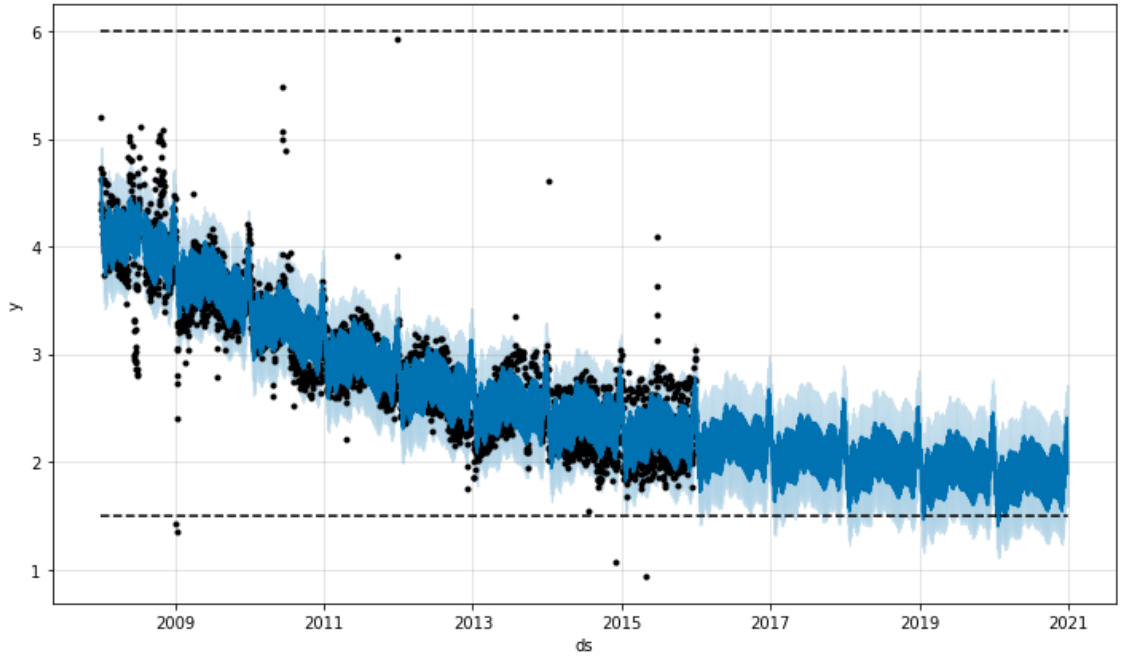
```
forecast.tail()
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail(60)
```

Prophet

1) 데이터를 Prophet에 맞도록 가공
2) Prophet 객체 생성 → Fit
3) 미래 Dataframe 생성
4) 예측
5) Forecast 결과 확인
6) ***시각화***

- forecast 시각화

```
fig1 = m.plot(forecast)
```

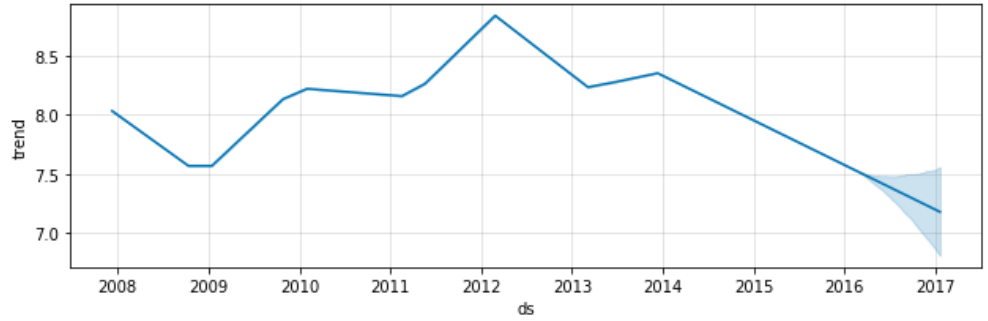


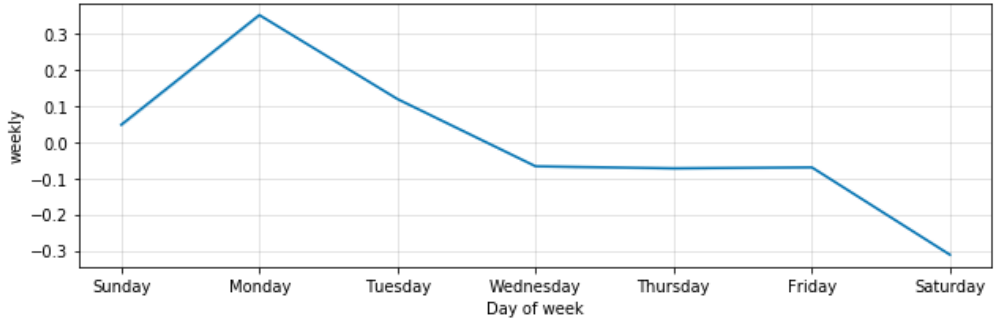
Prophet

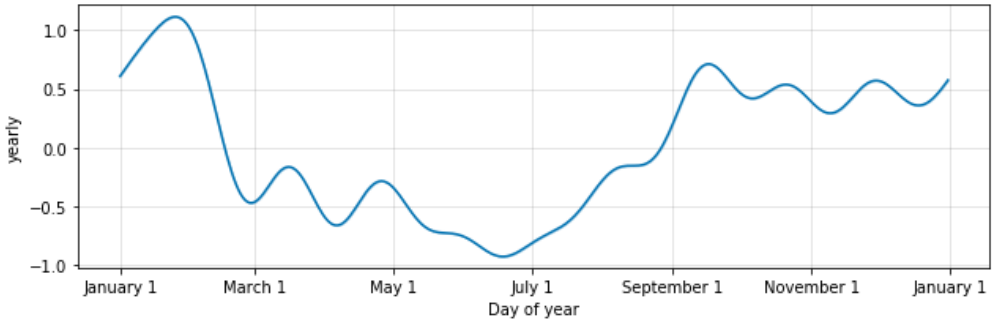
1) 데이터를 Prophet에 맞도록 가공
2) Prophet 객체 생성 → Fit
3) 미래 Dataframe 생성
4) 예측
5) Forecast 결과 확인
6) ***시각화***

- forecast component 시각화(Trend, Weakly, Yearly)

```
fig2 = m.plot_components(forecast)
```







Prophet

- ***Trend Change Points***
  - 트렌드가 변경되는 지점을 자동으로 감지해 트렌드를 예측함
    - 감지하는 것을 사용자가 조절 가능
  - Prophet 객체를 생성할 때 ***changepoint_range, changepoint_prior_scale, changepoints***을 조절

    1) changepoint_range

    - 기본적으로 Prophet은 시계열 데이터의 80% 크기에서 잠재적으로 ChangePoint를 지정
    - 90%만큼 ChangePoint로 지정하고 싶다면 아래와 같이 설정

```
m = Prophet(changepoint_range=0.9)
```

Prophet

- ***Trend Change Points***

  2) changepoint_prior_scale

  - Change Point의 유연성을 조정하는 방법
  - 오버피팅이 심하면 너무 유연한 그래프가 나와서 모든 값에 근접하고, 언더피팅일 경우 유연성이 부족
  - 기본 값은 0.05
  - 이 값을 늘리면 그래프가 유연해지고(=언더피팅 해결), 이 값을 줄이면 유연성이 감소(=오버피팅 해결)

```
m = Prophet(changepoint_prior_scale=0.05)
```

  3) changepoints

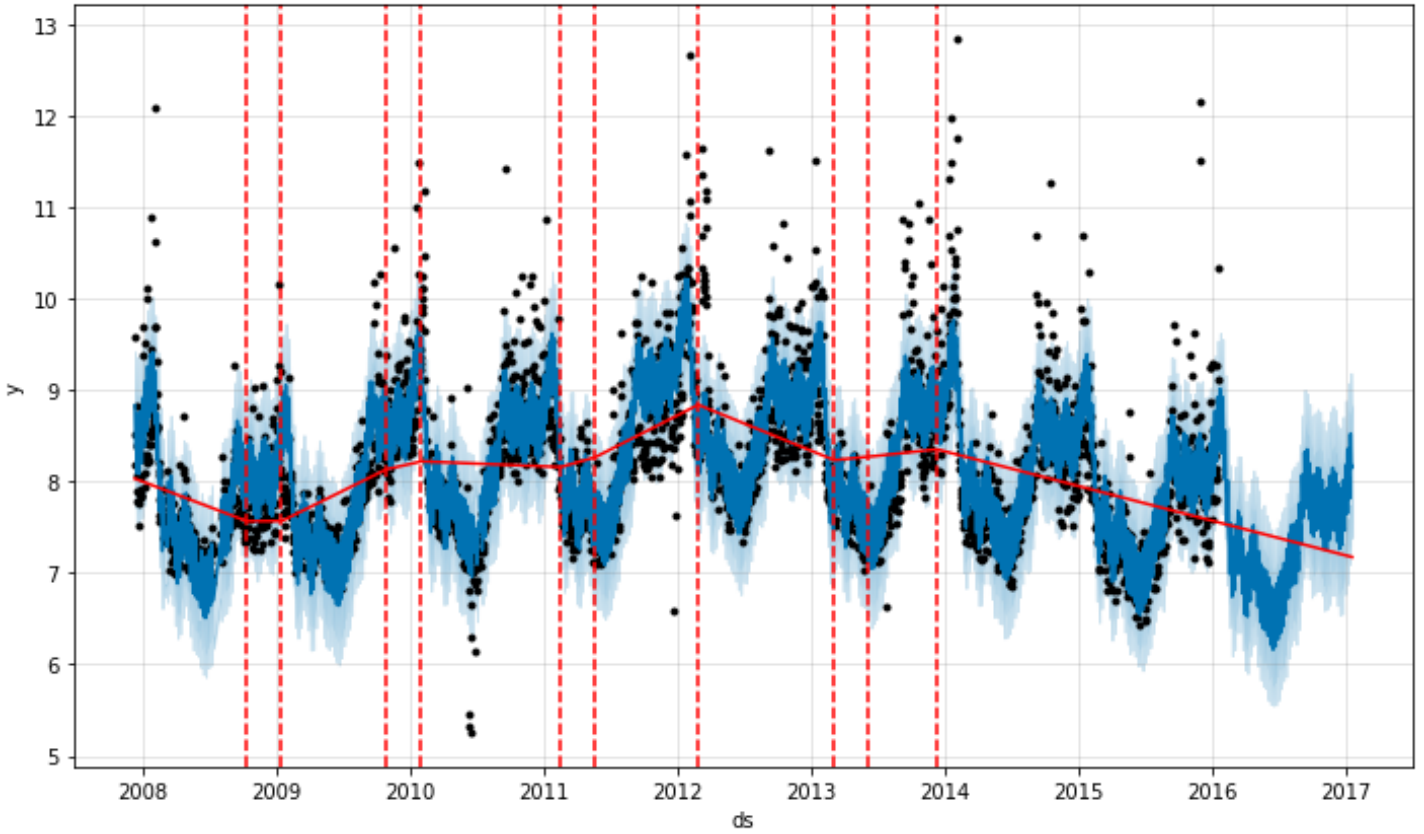
  - 잠재적으로 change point일 수 있는 날짜들
  - 명시하지 않으면 잠재적인 changepoint가 자동으로 설정됨

```
m = Prophet(changepoints=['2019-02-04'. '2019-02-05'])
```

Prophet

- *시각화*

```
from fbprophet.plot import add_changepoints_to_plot

fig = m.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), m, forecast)
```



Prophet

- ***Seasonality, Holiday Effects, And Regressors***
  - Modeling Holidays and Special Events
    - 휴일이나 모델에 반영하고 싶은 이벤트가 있으면 Dataframe을 생성해 반영할 수 있음
    - 이벤트는 과거 데이터와 미래 데이터가 모두 포함되어 있어야 함
    - 주변 날짜를 포함시키기 위해 lower_window, upper_window를 사용해 업데이트의 영향을 조절 가능

    ex) Play Off 경기일과 SUperbowl 경기날을 Holiday로 설정

```
playoffs = pd.DataFrame({
            'holiday': 'playoff',
            'ds': pd.to_datetime(['2008-01-13', '2009-01-03', '2010-01-16',
                                  '2010-01-24', '2010-02-07', '2011-01-08', '2013-01-12',
                                  '2014-01-12', '2014-01-19', '2014-02-02', '2015-01-11',
                                  '2016-01-17', '2016-01-24', '2016-02-07']),
            'lower_window': 0, 'upper_window': 1, })

superbowls = pd.DataFrame({ 'holiday': 'superbowl',
            'ds': pd.to_datetime(['2010-02-07', '2014-02-02', '2016-02-07']),
            'lower_window': 0, 'upper_window': 1, })

holidays = pd.concat((playoffs, superbowls))
```

Prophet

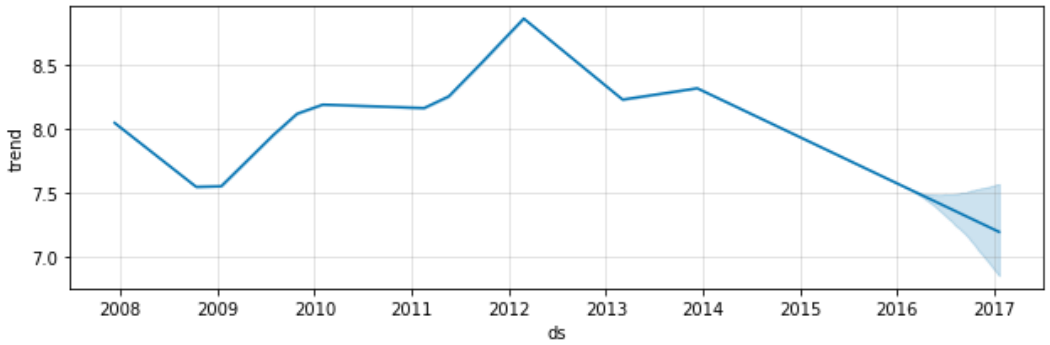
- ***Seasonality, Holiday Effects, And Regressors***
  - Modeling Holidays and Special Events
    - 휴일이나 모델에 반영하고 싶은 이벤트가 있으면 Dataframe을 생성해 반영할 수 있음
    - 이벤트는 과거 데이터와 미래 데이터가 모두 포함되어 있어야 함
    - 주변 날짜를 포함시키기 위해 lower_window, upper_window를 사용해 업데이트의 영향을 조절 가능
    - 예제는 Play Off 경기일과 Superbowl 경기날을 Holiday로 설정

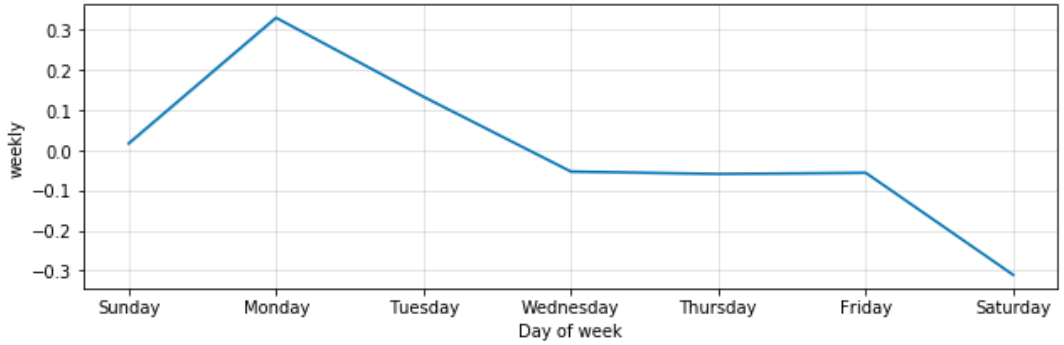
```
m = Prophet(holidays=holidays)
forecast = m.fit(df).predict(future)

forecast[(forecast['playoff'] + forecast['superbowl']).abs() > 0][ ['ds', 'playoff', 'superbowl']][-10:]
```

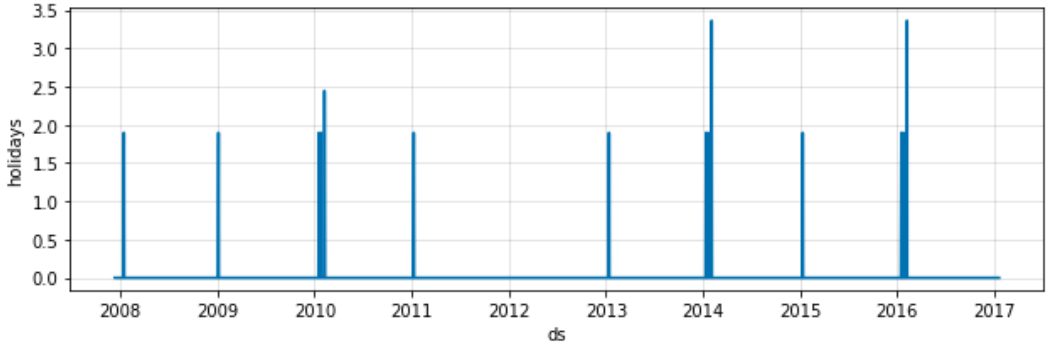
Prophet

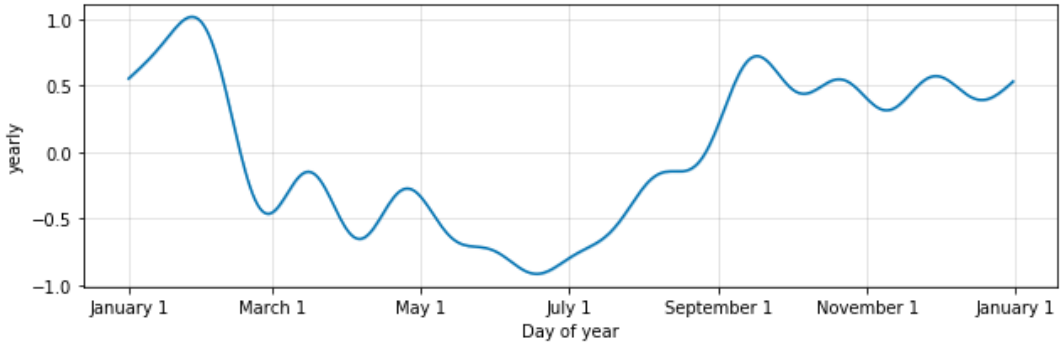
- ***Seasonality, Holiday Effects, And Regressors***
  - plot_component로 시각화할 경우 holidays의 영향도 볼 수 있음

```
fig = m.plot_components(forecast)
```









ts_demo2.ipynb

Prophet - demo

- ***주식 데이터 예측***
  - 시작가, 고가, 최저가, 종가
  - https://finance.yahoo.com 에서 주식 데이터 가져오기
    - $ ***pip install pandas_datareader***
    - $ ***pip install yfinance***
    - $ ***pip install fix_yahoo_finance***

```
from pandas_datareader import data
import datetime
import yfinance as yf
yf.pdr_override()
```

```
start_date = '2008-01-01'
name = 000880.KS'
hw = data.get_data_yahoo(name, start_date)
hw.head(3)
```

Date	Open	High	Low	Close	Adj Close	Volume
2008-01-02	62805.5	64952.7	62000.3	62000.3	52696.57	369645
2008-01-03	61731.9	63431.7	60032.0	63163.3	53685.05	376728
2008-01-04	62626.5	63431.7	61016.1	63252.8	53761.13	395231