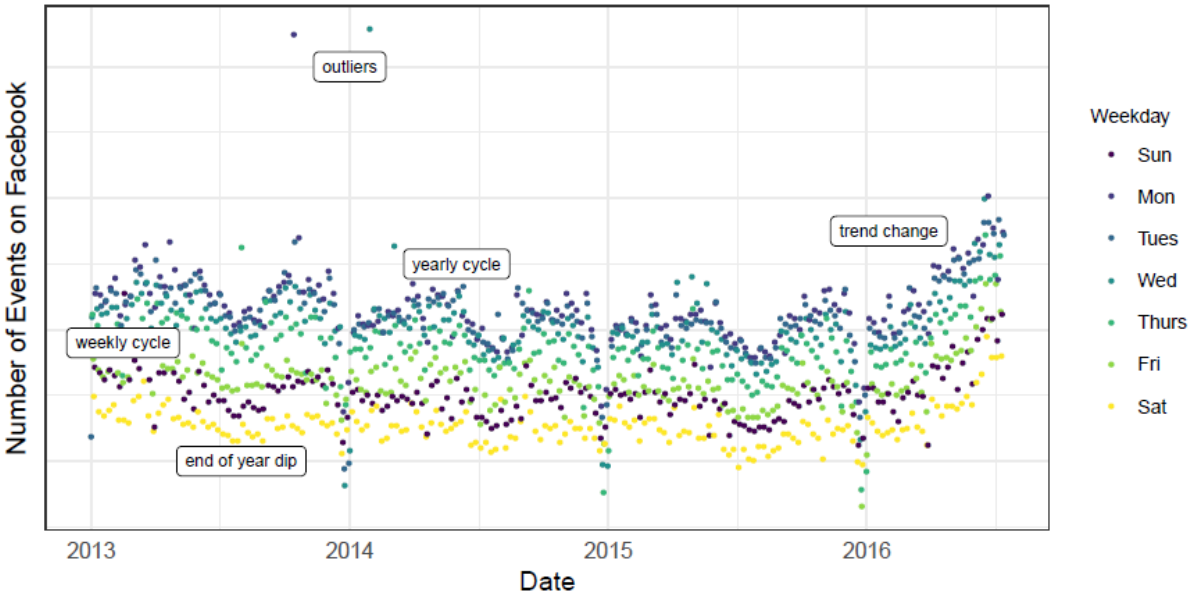
Anomaly Detection – Prophet

1. Prophet 도입 배경

- Facebook이 만든 시계열 예측 모델

- Facebook에서 개발한, 비선형 트렌드가 매년, 주별 및 일일 계절 성과 플러스 휴가 효과에 적합한 가산 모델을 기반으로 시계열 데이터를 예측이 가능한 모델

- 날짜 정보와 예측할 y만 있으면 되기 때문에 간편하면서 성능이 좋음



- business forecasting 문제는 다양하지만 공통적인 특징이 있음

- 계절적 요소(주간/연간 주기성이나 새해 전으로 하강하는 모습, 최근 6개월 동안 과도하게 변화) -> 시계열 데이터에서 많이 보이며 이러한 데이터 예측하는 것은 한계

2. Prophet Forecasting Model

- Prophet 모형은 트렌드(growth), 계절성(seasonality), 휴일(holidays) 3가지 main components로 구성

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t.$$

- g(t)는 비주기적 변화를 반영하는 추세함수, s(t)는 주기적인 변화, h(t)는 휴일(불규칙)의 영향력을 나타냄

- prophet 장점:

1) 유연성: 계절적 요소화 트렌드에 대한 가정을 쉽게 반영

2) ARIMA 모델과 달리 모델을 측정 간격을 regularly spaced할 필요가 없고, 결측이 있어도 상관 없음

3) 빠른 Fitting 속도 -> 여러 가지 시도 가능

4) 직관적인 파라미터 조정을 통한 모델 확장 용이

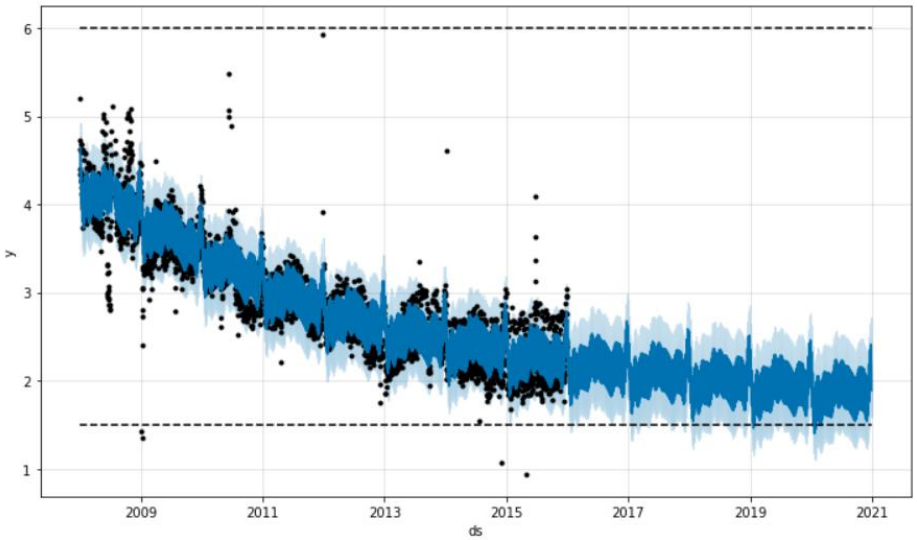
2.1 g(t), Growth

- 논문에서는 트렌드를 반영하기 위한 growth model로 두 가지 소개

- 첫 번째는 상한, 하한과 같은 한계점이 존재하는 경우: saturating growth model

- 두 번째는 없는 경우: piecewise logistic model, piecewise linear model을 기반

2.1.1 Nonlinear, Saturating Growth



- 상한/하한이 있는 growth는 기본적으로 아래와 같은 로지스틱 함수로써 나타내어짐

$$g(t) = \frac{C}{1 + \exp(-k(t - m))},$$

The logistic growth model(basic)

- C는 한계를 나타내는 carrying capacity, k는 성장률(growth rate) 의미

- but, 반영할 수 없는 2가지

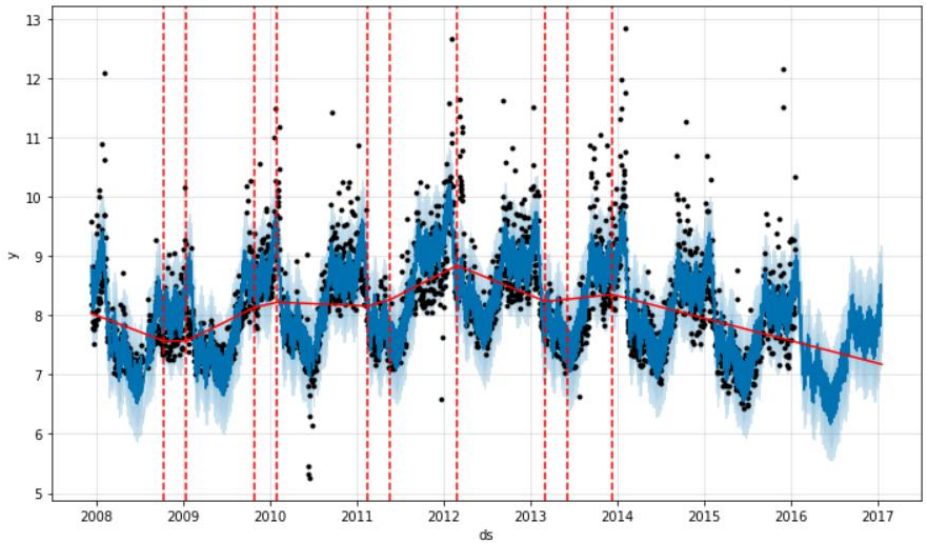
1) carrying capacity가 상수가 아닌 경우 -> 가입자 수 예측 시, 우리의 상한은 인터넷에 연결이 가능한 인구 수지만, 연결 가능 인구 수는 시간이 지남에 따라 증가하기 때문에 상수 X

2) 성장률이 상수가 아닌 경우 -> 신제품 출시로 기존의 성장률과 다른 양상 보임

- 이 문제 해결하기 위해 basic한 로지스틱 함수에서 C를 시간에 따라 변하는 C(t) 함수로 바꾸고, change points 전후로 달라지는 성장률을 대변하기 위해 대체한 piecewise logistic growth model을 사용

$$g(t) = \frac{C(t)}{1 + \exp(-(k + \mathbf{a}(t)^\intercal \boldsymbol{\delta})(t - (m + \mathbf{a}(t)^\intercal \boldsymbol{\gamma})))}.$$

2.1.2 Linear Trend with Changepoints



- 한계가 존재하지 않는 예측 문제에 있어서 piecewise linear 모델은 간결하고 유용

- piecewise하게 일정한 성장률을 가지는 trend model을 정의

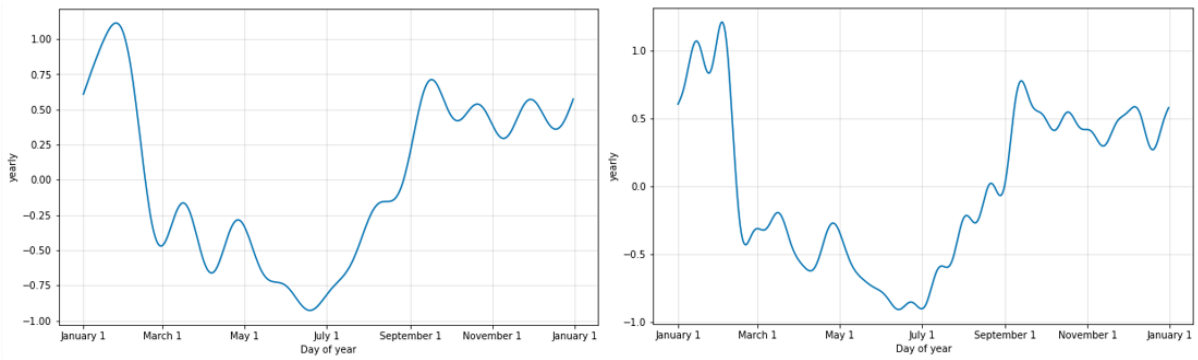
$$g(t) = (k + \mathbf{a}(t)^\intercal \boldsymbol{\delta})t + (m + \mathbf{a}(t)^\intercal \boldsymbol{\gamma}),$$

2.2 s(t), Seasonality

- 시계열 데이터는 인간의 행동에 의해서도 주기성 가질 수 있음

- 이러한 주기성을 시계열 모델에 반영하기 위해서는 시간에 주기적인 함수인 계절성 모델 필요

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right)$$



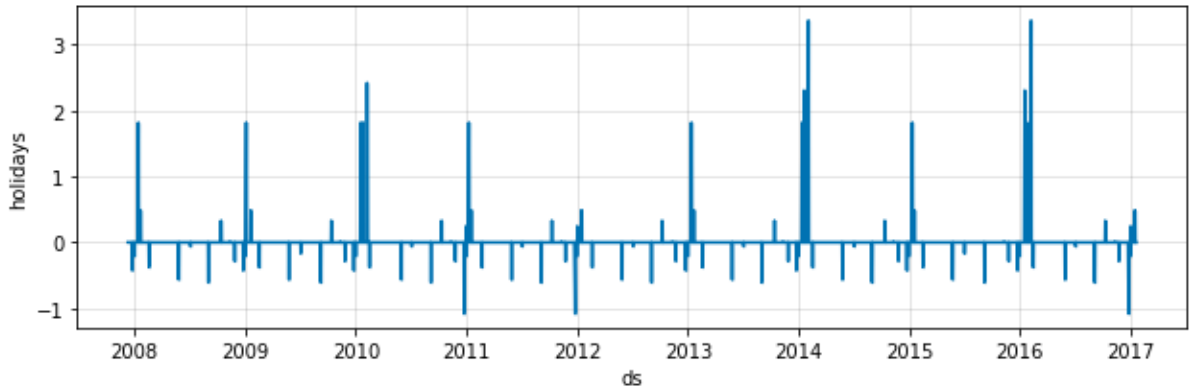
2.3 h(t), Holidays and Events

- 휴일과 이벤트들은 시계열 예측에 큰 영향을 미치지만 주기적인 패턴을 따르지 않아 모델링하기 어려움

- 이벤트의 효과는 독립으로 가정, 이벤트 앞뒤로 window 범위를 지정해 해당 이벤트의 영향력의 범위를 설정

- 각각의 휴일을 Di로 두었을 때, 아래와 같은 h(t)를 정의

$$Z(t) = [\mathbf{1}(t \in D_1), \ldots, \mathbf{1}(t \in D_L)] \qquad h(t) = Z(t)\boldsymbol{\kappa}.$$

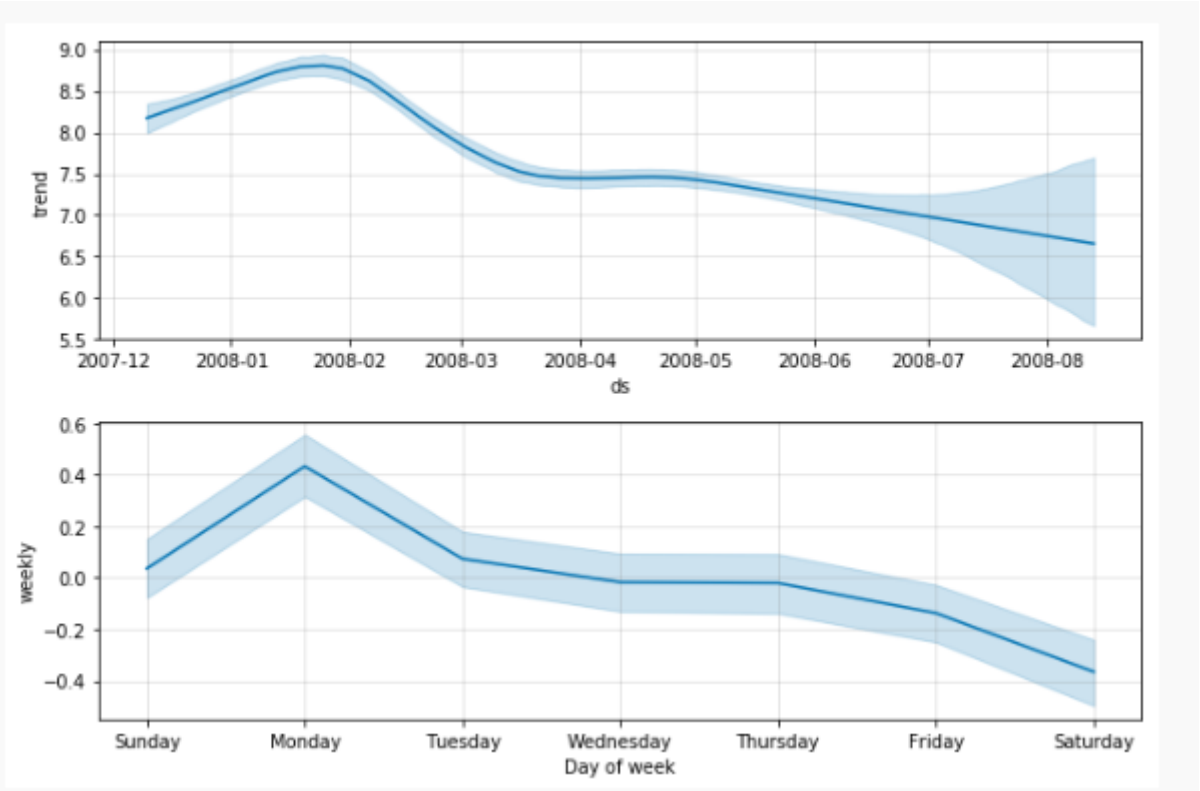


출처: https://m-insideout.tistory.com/13

3. Prophet Parameter

- Prophet 모델의 핵심인 interval에 대해 소개

- Upper/Lower limit이 존재



- yhat_lower: yhat의 가장 낮은 값

- yhat_upper: yhat의 가장 높은 값

- 즉, yhat은 예측 값의 범위

- 기본적으로 Prophet은 이 yhat의 값으로 예측 값의 불확실성 범위 표현

- 기본 값은 80%, 즉 예측 정확도가 80%로 보면 됨

- 정확도를 높이거나 낮추고 싶다면 조정 가능(95%로 조정)

forcast = Prophet(**interval_width=0.95**).fit(df).predict(future)

- 전체 trend에 대해서 정확도를 제어 가능

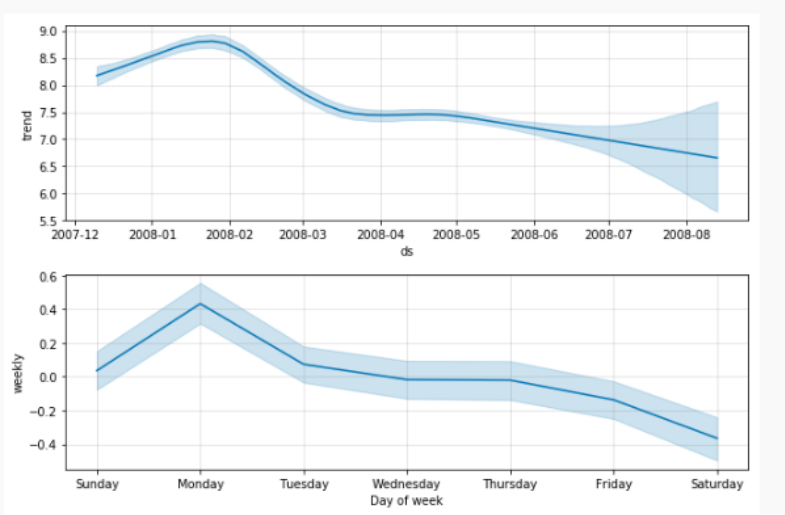
- Seasonality에도 추가적으로 제어 가능

- 이를 위해 Bayesian sampling을 사용

- mcmc_samples라는 매개변수 사용 (디폴트 = 0)

- 이 값을 300으로 주면 데이터의 최초 300일에 대해서 적용

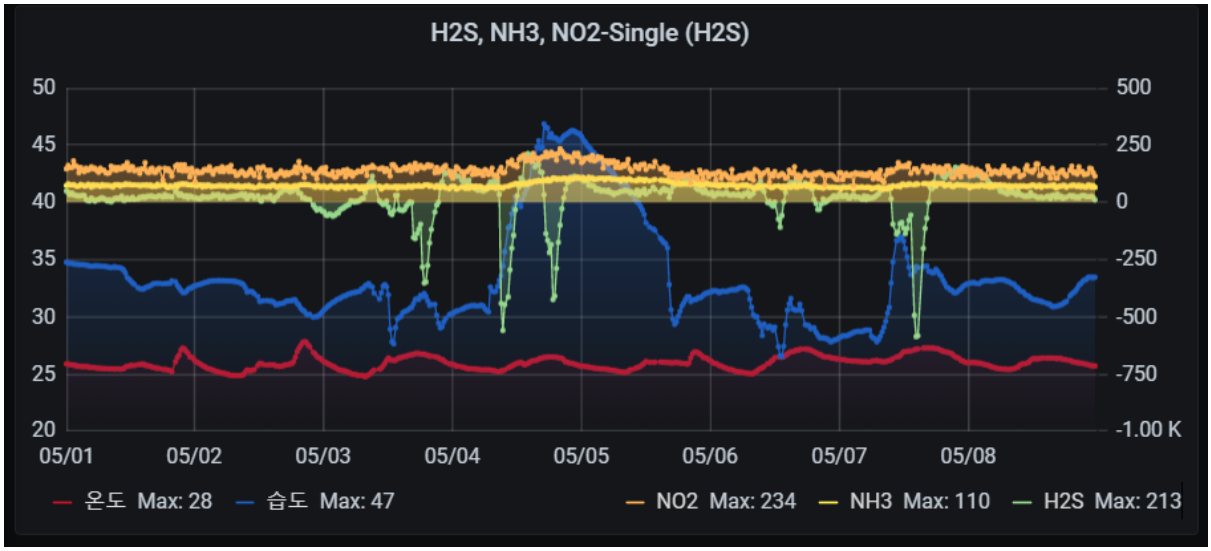
```
m = Prophet(mcmc_samples=300)
forecast = m.fit(df).predict(future)

fig = m.plot_components(forecast)
```



출처: https://zamezzz.tistory.com/285

4. 센서데이터 Prophet 적용

1) Grafana에서 온도, 습도, H2S, NH3, NO2 추출



2) csv import

```
dataset = pd.read_csv('/content/drive/MyDrive/Sensor/H2S, NH3, NO2-Single (H2S)-data-as-seriestocolumn
merged_data = pd.DataFrame()
merged_data=merged_data.append(dataset)
merged_data
```

	Time	온도	습도	H2S	NH3	NO2
0	2021-05-01 00:00:00	26	35	39	69	146
1	2021-05-01 00:10:00	26	35	49	73	143
2	2021-05-01 00:20:00	26	35	43	80	161
3	2021-05-01 00:30:00	26	35	27	71	149
4	2021-05-01 00:40:00	26	35	33	77	152
...	...	...	...	...	...	...
919	2021-05-08 23:00:00	26	33	21	65	146
920	2021-05-08 23:10:00	26	33	27	65	99
921	2021-05-08 23:20:00	26	33	16	71	120
922	2021-05-08 23:30:00	26	33	21	73	129
923	2021-05-08 23:50:00	26	33	10	65	114

924 rows × 6 columns

-> Time 행을 index로 바꾸는 작업 필요

-> 시계열 데이터로 만들기 위해

3) Time index로 변환

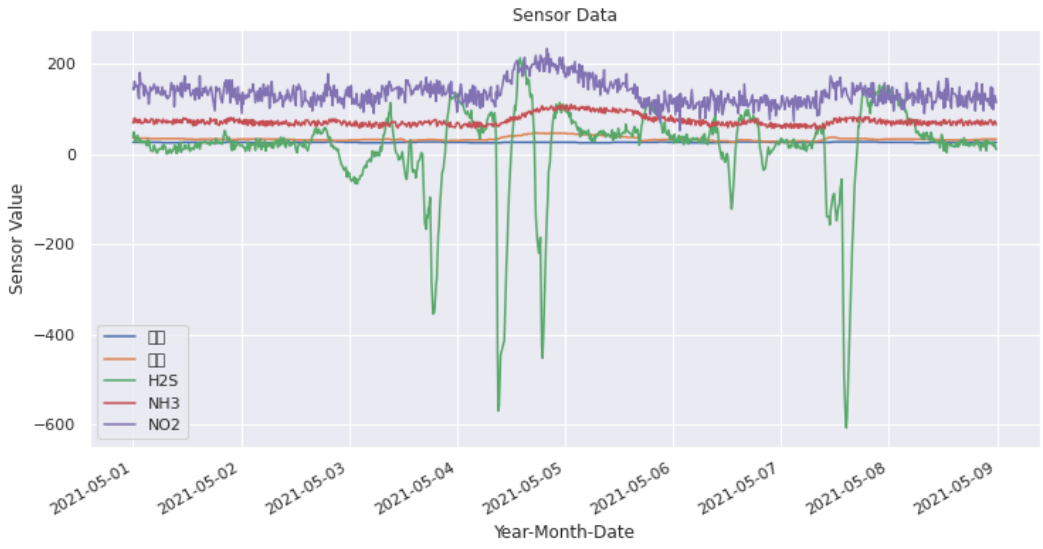
```
dataset = pd.read_csv('/content/drive/MyDrive/Sensor/H2S, NH3, NO2-Single (H2S)-data-as-seriestocolumns-2021-05-17 14_49_43.csv')
merged_data = pd.DataFrame()
merged_data=merged_data.append(dataset)
merged_data
merged_data.set_index('Time', inplace=True)
merged_data.index = pd.to_datetime(merged_data.index)
merged_data
```

Time	온도	습도	H2S	NH3	NO2
2021-05-01 00:00:00	26	35	39	69	146
2021-05-01 00:10:00	26	35	49	73	143
2021-05-01 00:20:00	26	35	43	80	161
2021-05-01 00:30:00	26	35	27	71	149
2021-05-01 00:40:00	26	35	33	77	152
...	...	...	...	...	...
2021-05-08 23:00:00	26	33	21	65	146
2021-05-08 23:10:00	26	33	27	65	99
2021-05-08 23:20:00	26	33	16	71	120
2021-05-08 23:30:00	26	33	21	73	129
2021-05-08 23:50:00	26	33	10	65	114

924 rows × 5 columns

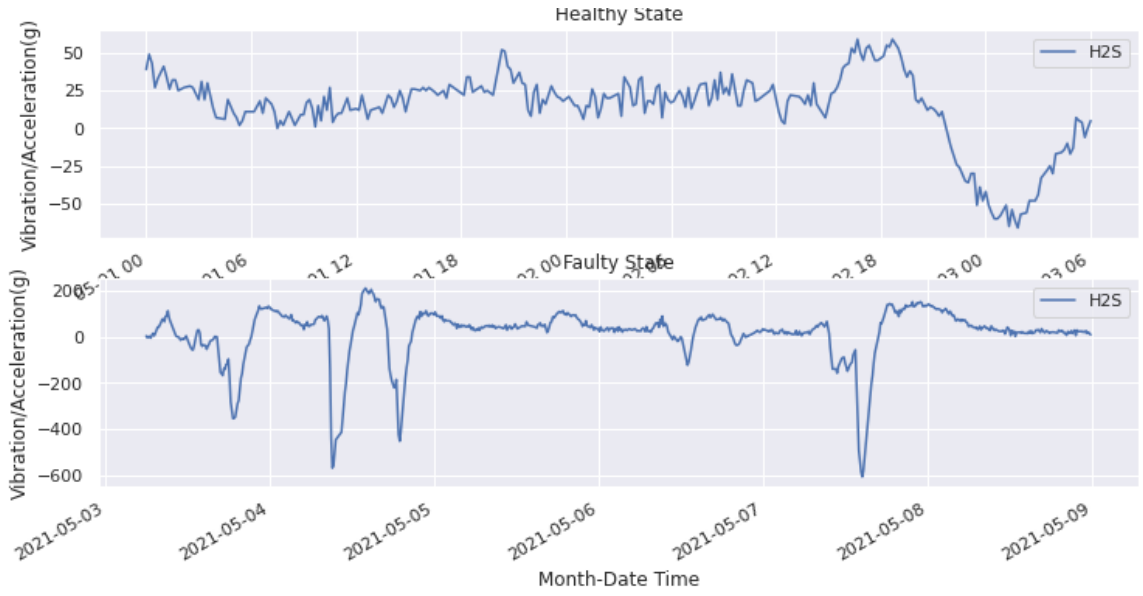
4) 센서 데이터 시각화

```
# Visualising Data
ax = merged_data.plot(figsize = (12,6), title="Sensor Data" , legend = True)
ax.set(xlabel="Year-Month-Date", ylabel="Sensor Value")
#plt.axvline(x='2021-05-08 00:00:00', linewidth=4, color='b', label ="Breakdown of Bearing 1")
#plt.text('2004-02-19 06:12:39',0.3,'Breakdown of Bearing 1',rotation=90, fontsize=14, color='b')
```



5) 5종 데이터 중 H2S 선택 후 Healthy/Faulty 데이터셋 구분

```
fig = plt.figure()

# Divide the figure into a 1x2 grid, and give me the first section
ax1 = fig.add_subplot(211)
# Divide the figure into a 1x2 grid, and give me the second section
ax2 = fig.add_subplot(212)

healthy = merged_data['2021-05-01 00:00:00':'2021-05-03 06:00:00']
healthy['H2S'].plot(figsize = (12,6), title="Healthy State" , legend = True, ax=ax1)
ax1.set(xlabel="Month-Date Time", ylabel="Vibration/Acceleration(g)")

faulty = merged_data['2021-05-03 06:00:00':'2021-05-08 23:50:00']
ax2 = faulty['H2S'].plot(figsize = (12,6), title="Faulty State" , legend = True, ax= ax2)
ax2.set(xlabel="Month-Date Time", ylabel="Vibration/Acceleration(g)")
```



-> 튀는 부분이 그나마 적은 1일부터 3일 6시까지를 Healthy State로 선정

-> 나머지를 Faulty State

6) Training dataset 시간 설정

```
H2S = merged_data['2021-05-01 00:00:00':'2021-05-03 06:00:00']['H2S']

# Creating training dataframe
prophet_healthy_train = pd.DataFrame()
prophet_healthy_train['ds'] = H2S.index
prophet_healthy_train['y'] = H2S.values

prophet_healthy_train.head()
H2S
#H2S.rename_axis(None, inplace=True)
H2S

#H2S.time = pd.to_datetime(H2S.time)
```

```
Time
2021-05-01 00:00:00    39
2021-05-01 00:10:00    49
2021-05-01 00:20:00    43
2021-05-01 00:30:00    27
2021-05-01 00:40:00    33
                       ..
2021-05-03 05:20:00     5
2021-05-03 05:30:00     4
2021-05-03 05:40:00    -6
2021-05-03 05:50:00     0
2021-05-03 06:00:00     5
Name: H2S, Length: 266, dtype: int64
```

-> 이때, Time index name('Time')으로 인해 추가적인 plotting에 방해

-> index name 없애는 작업

```
H2S = merged_data['2021-05-01 00:00:00':'2021-05-03 06:00:00']['H2S']

# Creating training dataframe
prophet_healthy_train = pd.DataFrame()
prophet_healthy_train['ds'] = H2S.index
prophet_healthy_train['y'] = H2S.values

prophet_healthy_train.head()
H2S
H2S.rename_axis(None, inplace=True)
H2S

#H2S.time = pd.to_datetime(H2S.time)
```

```
2021-05-01 00:00:00    39
2021-05-01 00:10:00    49
2021-05-01 00:20:00    43
2021-05-01 00:30:00    27
2021-05-01 00:40:00    33
                       ..
2021-05-03 05:20:00     5
2021-05-03 05:30:00     4
2021-05-03 05:40:00    -6
2021-05-03 05:50:00     0
2021-05-03 06:00:00     5
Name: H2S, Length: 266, dtype: int64
```

7) interval width 설정

```
m = Prophet(interval_width = 1)
# Using the training data from "healthy part"
m.fit(prophet_healthy_train)
```

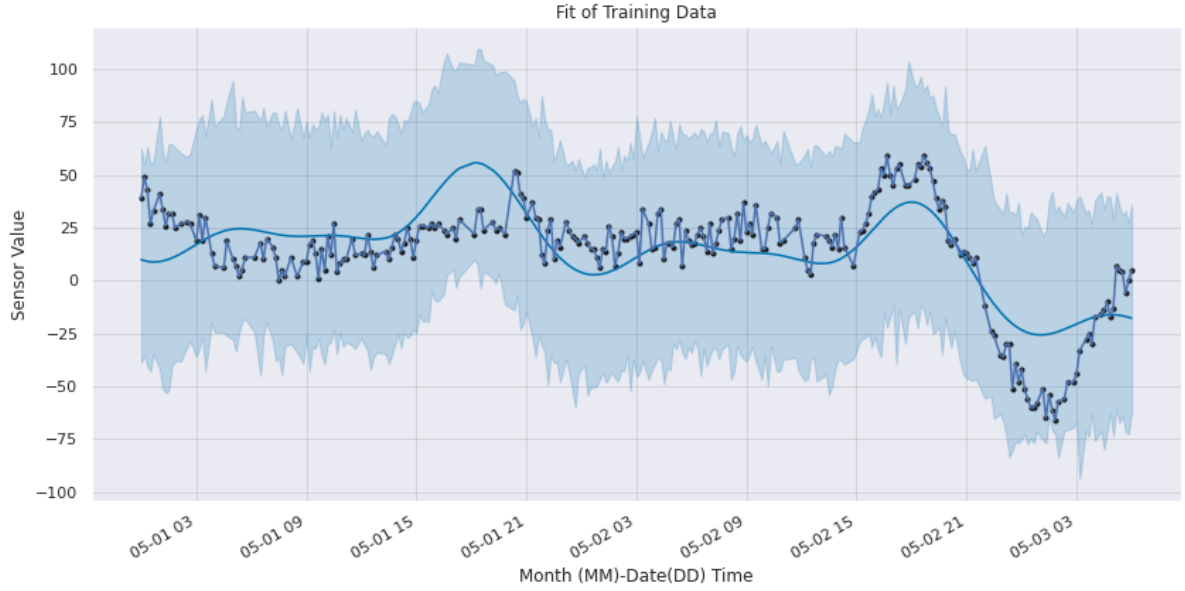
```
INFO:numexpr.utils:NumExpr defaulting to 2 threads.
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
<fbprophet.forecaster.Prophet at 0x7f5f89f04350>
```

-> default=0.8로 설정, 여기서는 1로 설정

8) Healthy data prophet plot

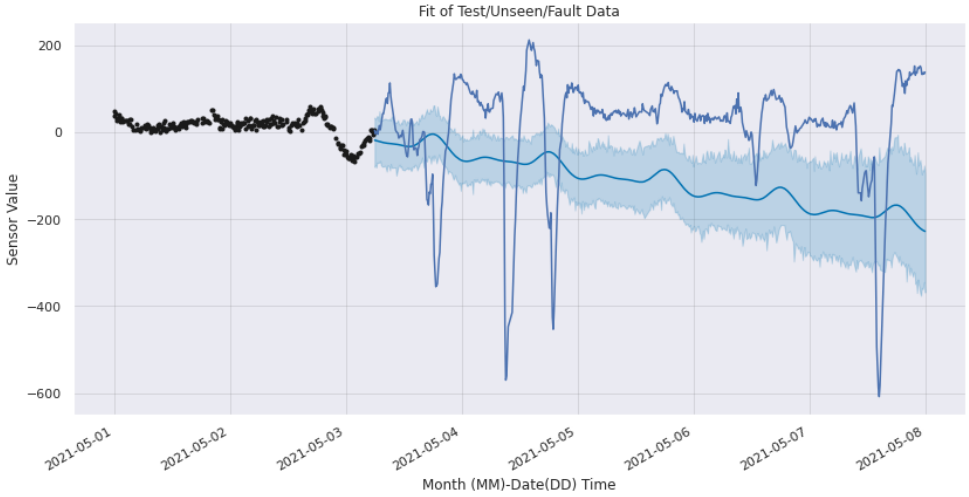
```
forecast = m.predict(prophet_healthy_train)
forecast['H2S'] = prophet_healthy_train['y'].reset_index(drop = True)
print('Displaying Prophet plot')
fig1 = m.plot(forecast)
fig1 = H2S.plot(figsize = (12,6), title="Fit of Training Data")
fig1.set(xlabel="Month (MM)-Date(DD) Time", ylabel="Sensor Value")
```

```
Displaying Prophet plot
[Text(46.25, 0.5, 'Sensor Value'), Text(0.5, 30.5, 'Month (MM)-Date(DD) Time')]
```



9) 전체 데이터셋 및 Faulty Data Prophet Plot

```
prophet_faultydata = merged_data['2021-05-03 06:00:00':'2021-05-08 00:00:00']['H2S']
prophet_faultydata.head()

prophet_faulty_test = pd.DataFrame()

prophet_faulty_test['ds'] = prophet_faultydata.index
#pd.to_datetime(prophet_healthy.index, format='%Y.%m.%d.%H.%M.%S')
prophet_faulty_test['y'] = prophet_faultydata.values

forecast = m.predict(prophet_faulty_test)
forecast['fact'] = prophet_faulty_test['y'].reset_index(drop = True)
print('Displaying Prophet plot')
fig1 = m.plot(forecast)
fig1 = prophet_faultydata.plot(figsize = (12,6),title="Fit of Test/Unseen/Fault Data")
fig1.set(xlabel="Month (MM)-Date(DD) Time", ylabel="Sensor Value")
#fig1.text(731626.875,0.057,'Expected/Predicted', fontsize=14, color='r')
#fig1.text(731626.875,0.075,'Actual/Faulty Data', fontsize=14, color='r')
#fig1.text(731624.875,0.057,'Actual/Healthy', fontsize=14, color='r')
```



10) yhat, yhat_upper, yhat_lower plot

```
# Healper functions to annotate the graph.
print(fig1.get_xticks())
print(fig1.get_yticks())
```

```
[737911. 737912. 737913. 737914. 737915. 737916. 737917. 737918.]
[-800. -600. -400. -200.    0.  200.  400.]
```

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

	ds	yhat	yhat_lower	yhat_upper
530	2021-05-07 23:20:00	-223.345005	-357.399172	-96.618370
531	2021-05-07 23:30:00	-224.582424	-360.756736	-86.266223
532	2021-05-07 23:40:00	-225.688767	-352.875536	-89.749222
533	2021-05-07 23:50:00	-226.662824	-343.974676	-75.942368
534	2021-05-08 00:00:00	-227.504247	-368.106741	-94.645342