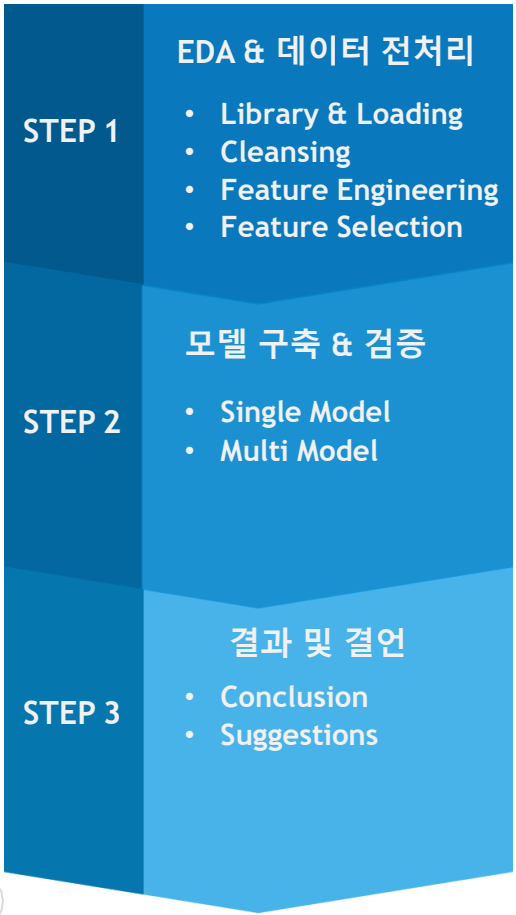
DACON

AI프렌즈 시즌1
공공데이터를 활용한
온도추정 경진대회

최상혁

목차



1. EDA & 데이터 전처리 : Library & Loading

In [1]:
```
# Fundamentals
import os
import sys
import numpy as np
import pandas as pd
np.random.seed(55)

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

# Machine Learning Algorithms
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LassoCV, Lasso
from sklearn.metrics import mean_squared_error

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Multiprocessing
from multiprocessing.dummy import Pool as ThreadPool
```

In [2]:
```
# Versions
print('python', sys.version.split(' ')[0])
print('numpy', np.__version__)
print('pandas', pd.__version__)
print('sklearn', sklearn.__version__)
```

```
python 3.7.4
numpy 1.17.2
pandas 0.25.1
sklearn 0.21.3
```

1. EDA & 데이터 전처리 : Library & Loading

In [5]:
```
# Load datasets
sample_submission = pd.read_csv("../da
train = pd.read_csv("../data/train.csv
test = pd.read_csv("../data/test.csv", index_col='id'

# 편리하고 직관적인 Visionaliztion을 위한 Indicator D
indicator_dict = {
    'temperature'   : ["X00","X07","X28","X31","X32"]
    'localpressure' : ["X01","X06","X22","X27","X29"]
    'windspeed'     : ["X02","X03","X18","X24","X26"]
    'precipitation' : ["X04","X10","X21","X36","X39"]
    'atmpressure'   : ["X05","X08","X09","X23","X33"]
    'insolation'    : ["X11","X34","X14","X16","X19"]
    'humidity'      : ["X12","X20","X30","X37","X38"]
    'winddirection' : ["X13","X15","X17","X25","X35"]
}

# Training Data를 Y18 값이 null인 30일(Train1)과 Y18 값이 존재하는 3일(Train2)로 나눔.
# 그리고 각각을 기상청 데이터(X)와 센서 데이터(Y01-Y17 or Y18)로 나눔.
idx1 = train[train['Y18'].isnull()].index
idx2 = train[~train['Y18'].isnull()].index

train1_X = train[train.columns[:40]].loc[idx1].copy()
train1_Ys = train[train.columns[40:-1]].loc[idx1].copy()

train2_X = train[train.columns[:40]].loc[idx2].copy()
train2_Y18 = train[train.columns[-1]].loc[idx2].copy()

test_X = test.copy()

combined_X = pd.concat([train1_X, train2_X, test_X], axis=0)
X_list = [train1_X, train2_X, test_X]
```

45 기상캐스터 잔나의 데이터를 만지는 5가지 꿀팁 1,008 view 2020-03-12 댓글 27 by 26

1) 시각화 쉽게하기

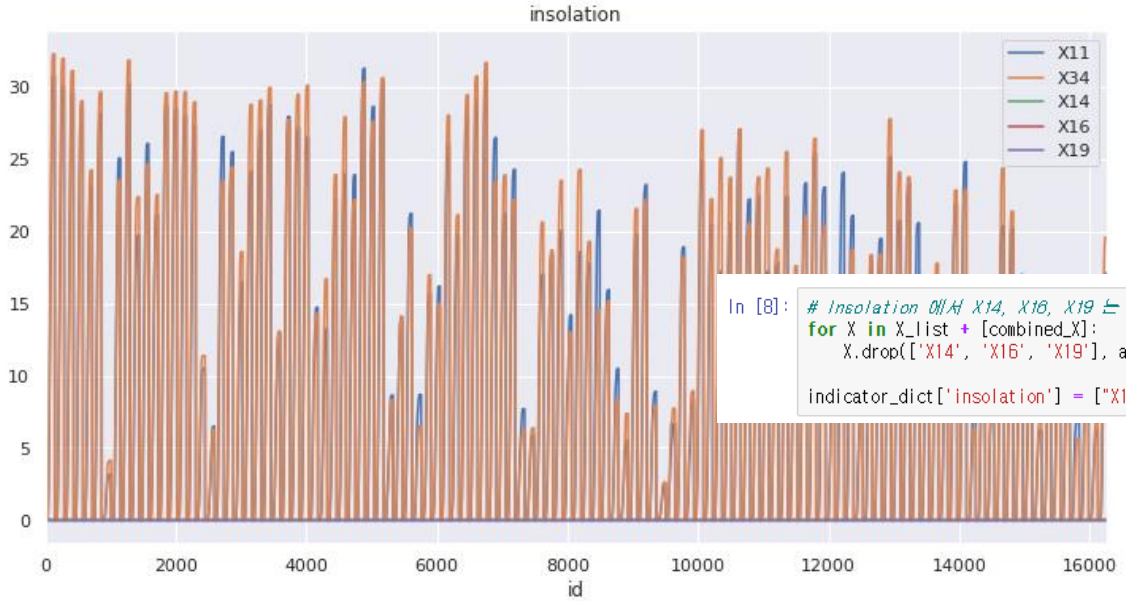
먼저 X컬럼을 여러분을 위해 나누어 드립니다. (솔직히 명적줘야 된다.)

n [2]:
```
temperature_name = ["X00","X07","X28","X31","X32"] #기온
localpress_name  = ["X01","X06","X22","X27","X29"] #현지기압
speed_name       = ["X02","X03","X18","X24","X26"] #풍속
water_name       = ["X04","X10","X21","X36","X39"] #일일 누적강수량
press_name       = ["X05","X08","X09","X23","X33"] #해면기압
sun_name         = ["X11","X14","X16","X19","X34"] #일일 누적일사량
humidity_name    = ["X12","X20","X30","X37","X38"] #습도
direction_name   = ["X13","X15","X17","X25","X35"] #풍향
```

1. EDA & 데이터 전처리 : Cleansing

2) Check Anomalies

In [7]:
```
for k, v in indicator_dict.items():
    combined_X.reset_index().plot(x='id', y=v, figsize=(12, 6))
    plt.gca().set_title(k)
```



In [8]:
```
# Insolation 에서 X14, X16, X19 는 관측자가 없음. => Drop!
for X in X_list + [combined_X]:
    X.drop(['X14', 'X16', 'X19'], axis=1, inplace=True)

indicator_dict['insolation'] = ["X11","X34"]
```

1. EDA & 데이터 전처리 : Feature Engineering

3) Time Series data의 특성을 활용해 date와 time 변수를 생성. (Feature Engineering을 위함)

In [9]:
```
for X in X_list + [combined_X]:
    X['date'] = X.index // 144
    X['time'] = X.index % 144
```

4) Modify Insolations and Precipitations

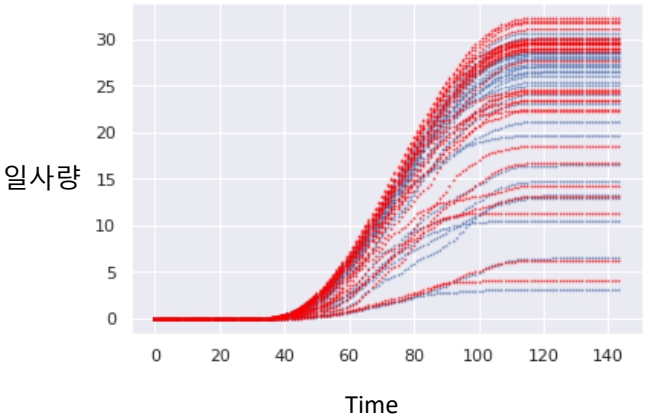
In [10]:
```
# 일일 누적치는 실시간 온도 추정에서 설명력이 없으므로 정해진 시간 동안의 Insolation & Precipitation을 구함.

# Modify the insolation (1시간) : 현재 값에서 1시간 이전(periods=6)의 값을 뺀 것으로 대치.
# 일사량은 일자가 바뀌는 자정 부근에 0이므로, shift로 인한 null 값은 0으로 fill.
for X in X_list:
    dates = X['date'].unique()
    for d in dates:
        day_idx = X.loc[X['date'] == d].index
        for f in indicator_dict['insolation']:
            X.loc[day_idx, f] = X.loc[day_idx, f] - X.loc[day_idx, f].shift(periods=6).fillna(0)
```

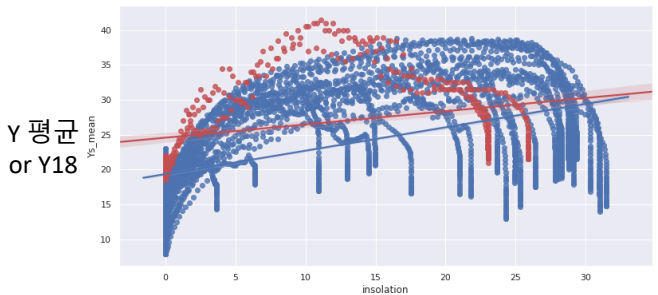
일일 누적 -> 1시간 누적

```
# Modify the Precipitation (6시간) : 현재 값에서 3시간 이전(periods=18)의 값을 뺀 것으로 대치.
# 강수량은 시간대와 관계없이 나타나므로, 날짜가 바뀌는 날에 0으로 reset됨. => 하루 전날의 누적 강수량을 더하여 값을 구함.
for X in X_list:
    dates = X['date'].unique()
    for f in indicator_dict['precipitation']:
        for d in dates[::-1]:
            day_idx = combined_X.loc[combined_X['date'] == d].index
            yes_day_idx = combined_X.loc[(combined_X['date'] == d-1)].index
            day_precip = combined_X.loc[(yes_day_idx | day_idx), f]

            # 편의를 위해 Traint, Train2, Test set 각각의 첫날(d=0)의 하루 전 누적 강수량은 0으로 고정함.
            yesterday_cum_precip = day_precip.loc[yes_day_idx].max() if d != 0 else 0
            day_precip.loc[day_idx] += yesterday_cum_precip
            day_precip_shift = day_precip.shift(periods=36).fillna(0)

            X.loc[day_idx, f] = day_precip.loc[day_idx] - day_precip_shift.loc[day_idx]
```

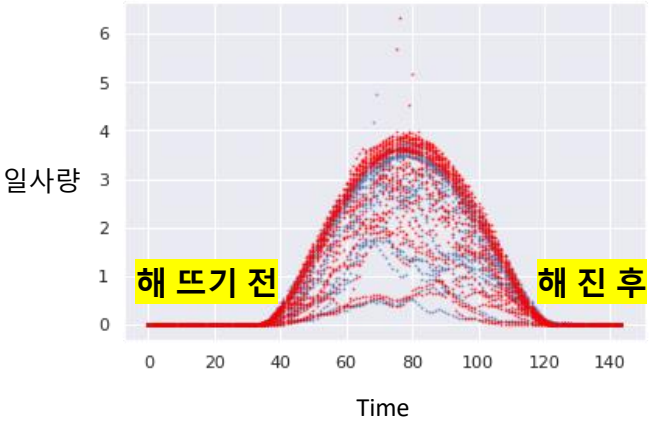
1. EDA & 데이터 전처리 : Feature Engineering

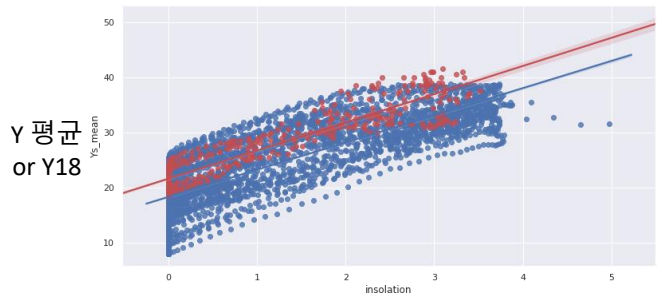
Before





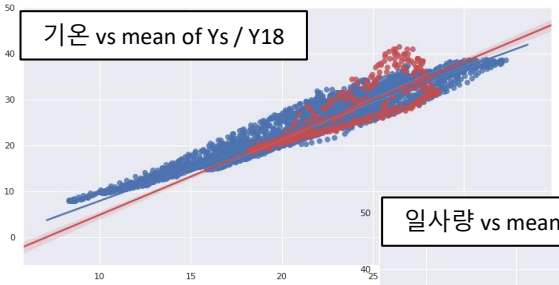
After

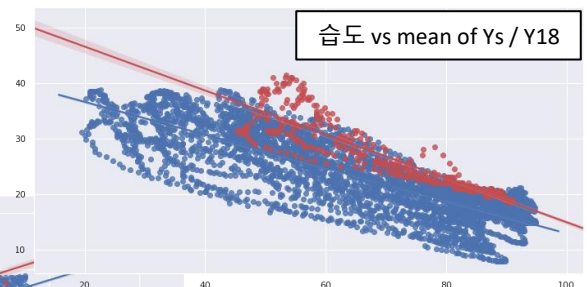


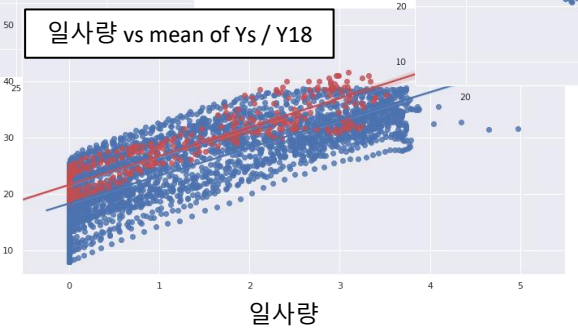


1. EDA & 데이터 전처리 : Feature Selection

```
In [10]: train1_Ys_mean = train1_Ys.mean(axis=1)
         for k, v  in indicator_dict.items():
             train1_feature_mean = train1_X[v].mean(axis=1)
             train2_feature_mean = train2_X[v].mean(axis=1)

             fig, ax = plt.subplots(figsize=(12, 6))
             sns.regplot(train1_feature_mean, train1_Ys_mean, ax=ax)
             sns.regplot(train2_feature_mean, train2_Y18, ax=ax, color='r')
             ax.set_xlabel(k)
             ax.set_ylabel('Ys_mean')
```

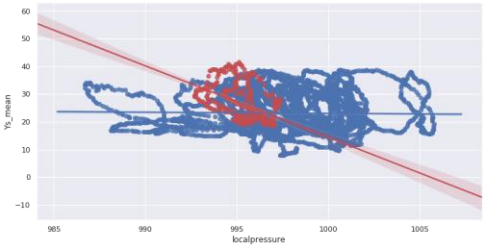


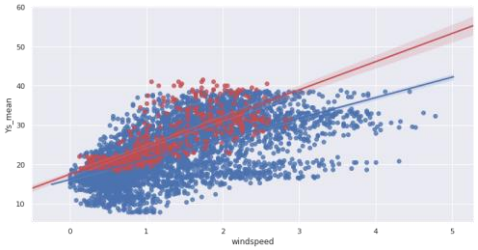


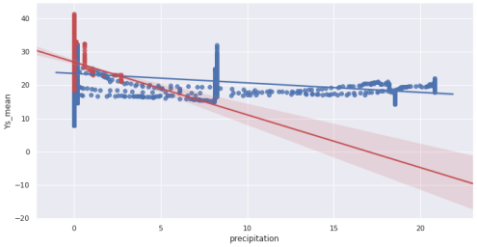


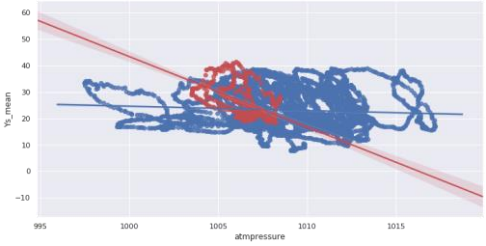
●파란색 : Train1 Data
●빨간색 : Train2 Data

1. EDA & 데이터 전처리 : Feature Selection









기온, 일사량, 습도 외
나머지 feature들은 모두 Drop!

In [12]:
```
# Train1과 Train2에서 consistent하게 linear한 상관관계가 있는 temperature, humidity, insolation 외에는 모두 drop 하기로 함.
for df in X_list:
    df.drop(indicator_dict['localpressure'], axis=1, inplace=True)
    df.drop(indicator_dict['atmpressure'], axis=1, inplace=True)
    df.drop(indicator_dict['windspeed'], axis=1, inplace=True)
    df.drop(indicator_dict['winddirection'], axis=1, inplace=True)
    df.drop(indicator_dict['precipitation'], axis=1, inplace=True)
indicator_dict = dict((k,indicator_dict[k]) for k in ['temperature', 'humidity', 'insolation'] if k in indicator_dict)
```

기온과 습도 각각에서 몇개의 변수들로 mean features를 생성

In [16]:
```
# Generate temp_mean and hum_mean
for X in X_list:
    X['temp_mean'] = X[["X07","X28","X31","X32"]].mean(1)
    X['hum_mean'] = X[["X30","X37","X38"]].mean(1)
indicator_dict['temperature'].append('temp_mean')
indicator_dict['humidity'].append('hum_mean')
```

<- X00 제외
<- X12, X20 제외

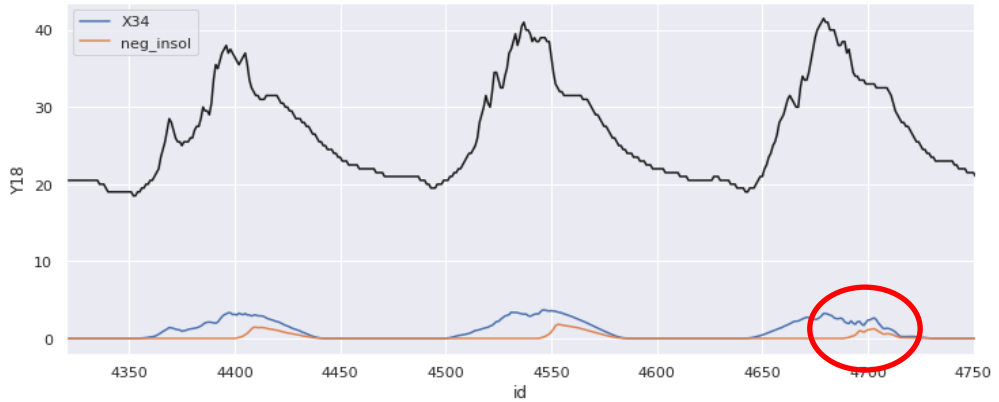
1. EDA & 데이터 전처리 : Feature Engineering

그림자?

↓

Negative Insolation (- insolation)

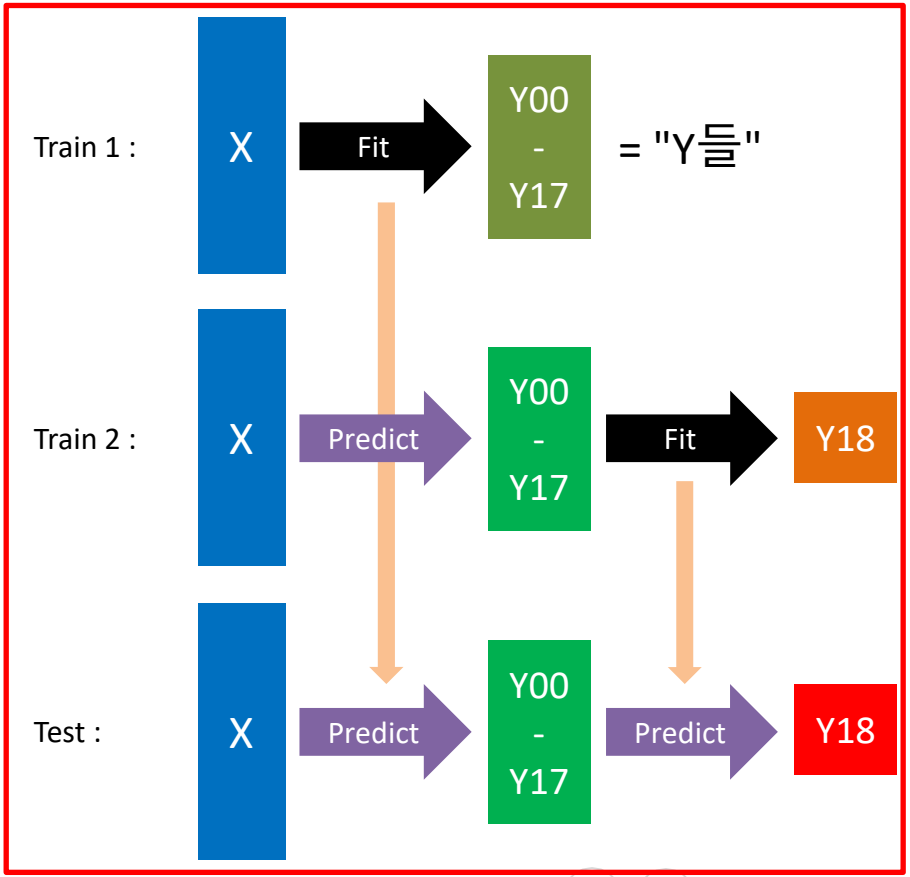
```
In [19]: # effect of Shadow or something after about 1pm.
for X in X_list:
    insol = X['X34']
    insol.index = X.index
    neg_insol = pd.Series(np.zeros_like(insol.values), index=X.index)

    time1_index = X.loc[X['time'] == 80].index  # 그림자가 지기 시작하는 시간
    time2_index = X.loc[X['time'] == 89].index  # 그림자가 완전히 진 시간
    end_index = X.loc[X['time'] == 143].index

    for t1, t2, end in zip(time1_index, time2_index, end_index):
        neg_insol.loc[t1:t2-1] = insol.loc[t1:t2-1] * (np.logspace(-1, 0, (t2-t1))-0.1)
        neg_insol.loc[t2:end] = insol.loc[t2:end]

    # 해가 질수록 (humidity가 상승할수록) neg insol의 영향력이 줄어들어야 한다고 판단
    # => neg_insol에 (1 - mean of humidity)를 곱해줌.
    humidity_mean = X[indicator_dict['humidity']].mean(1) / 100
    X['neg_insol'] = neg_insol * (1-humidity_mean)

indicator_dict['insolation'].append('neg_insol')
```



Noisy Feature를 찾아서 제거!

In [43]:
```
# Remove out noisy features
for X in X_list:
    X.drop(['X00', 'X11'], axis=1, inplace=True) # X00, X11 seems to be inconsistent with Y18,
indicator_dict['temperature'].remove('X00')
indicator_dict['insolation'].remove('X11')

# Remove date and time
for X in X_list:
    X.drop(['date', 'time'], axis=1, inplace=True)
```

2. 모델 구축 & 검증 : Single Model

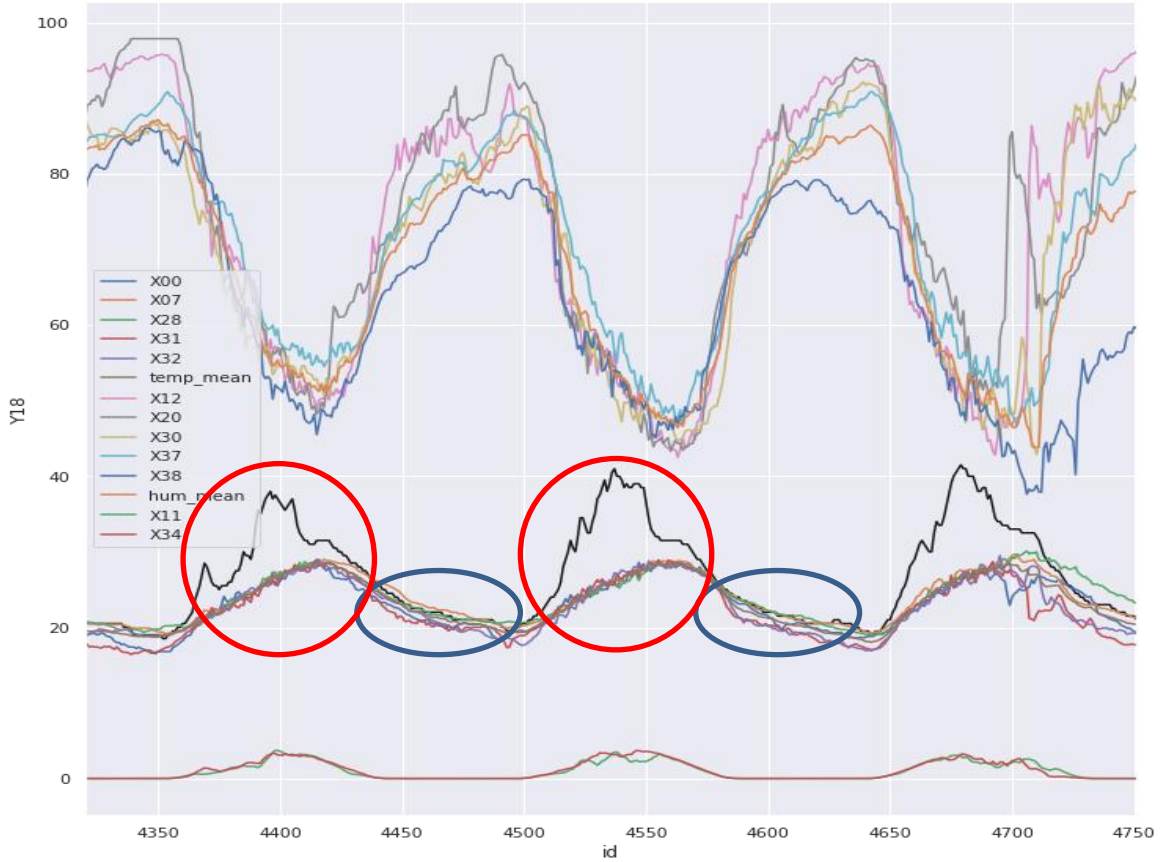


Learning Algorithm :
Lasso (LassoCV)

Evaluation :
Training score ,
Visualization,
Public score

단순한 구조!!

2. 모델 구축 & 검증 : Single Model

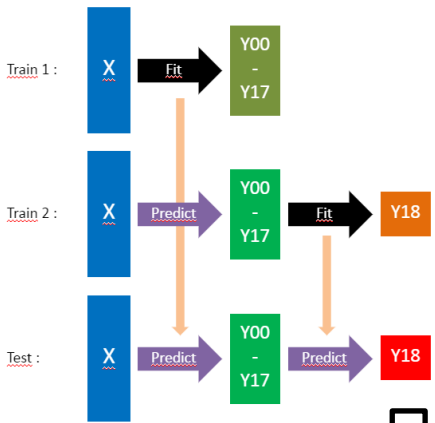


2. 모델 구축 & 검증 : Single Model

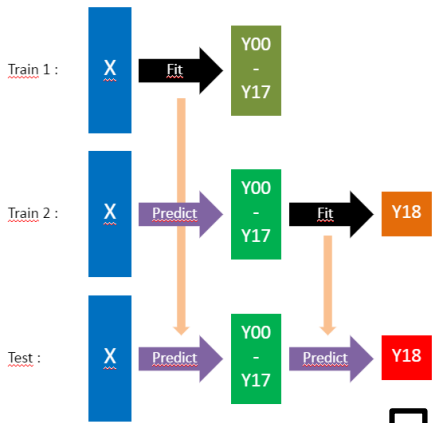
X['time'] = X.index % 144

Day index = **33** / Night index = **100**

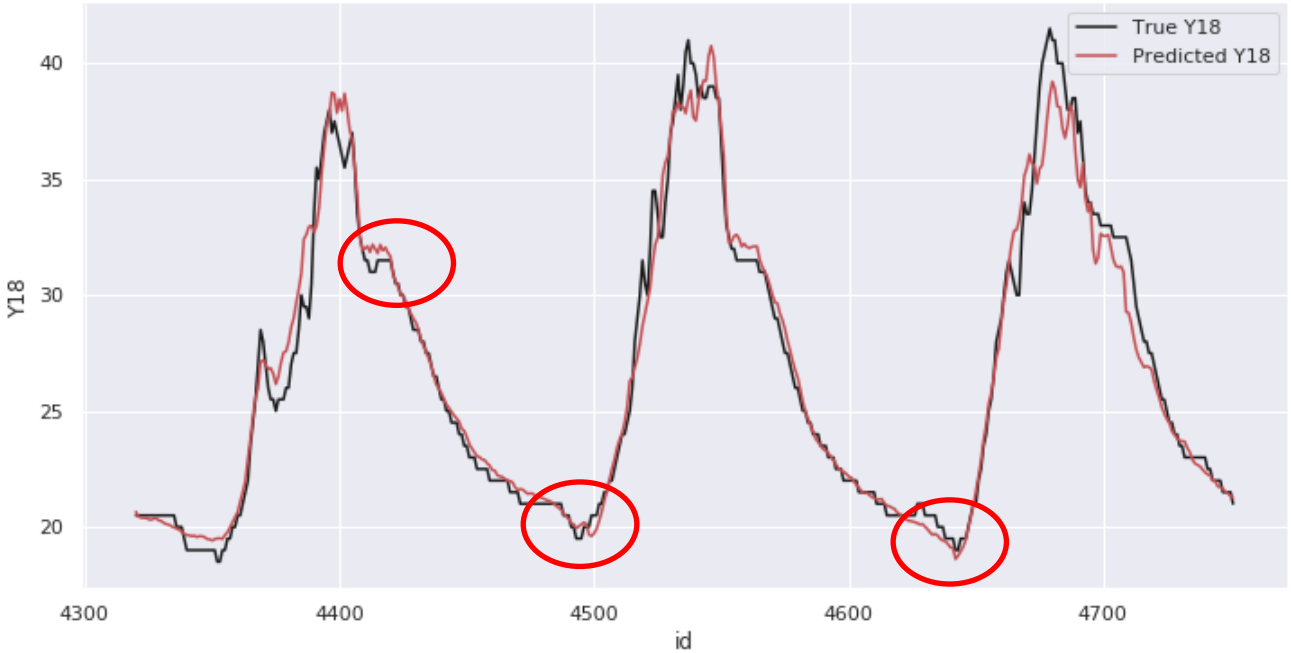
Day Time : [34 – 100]



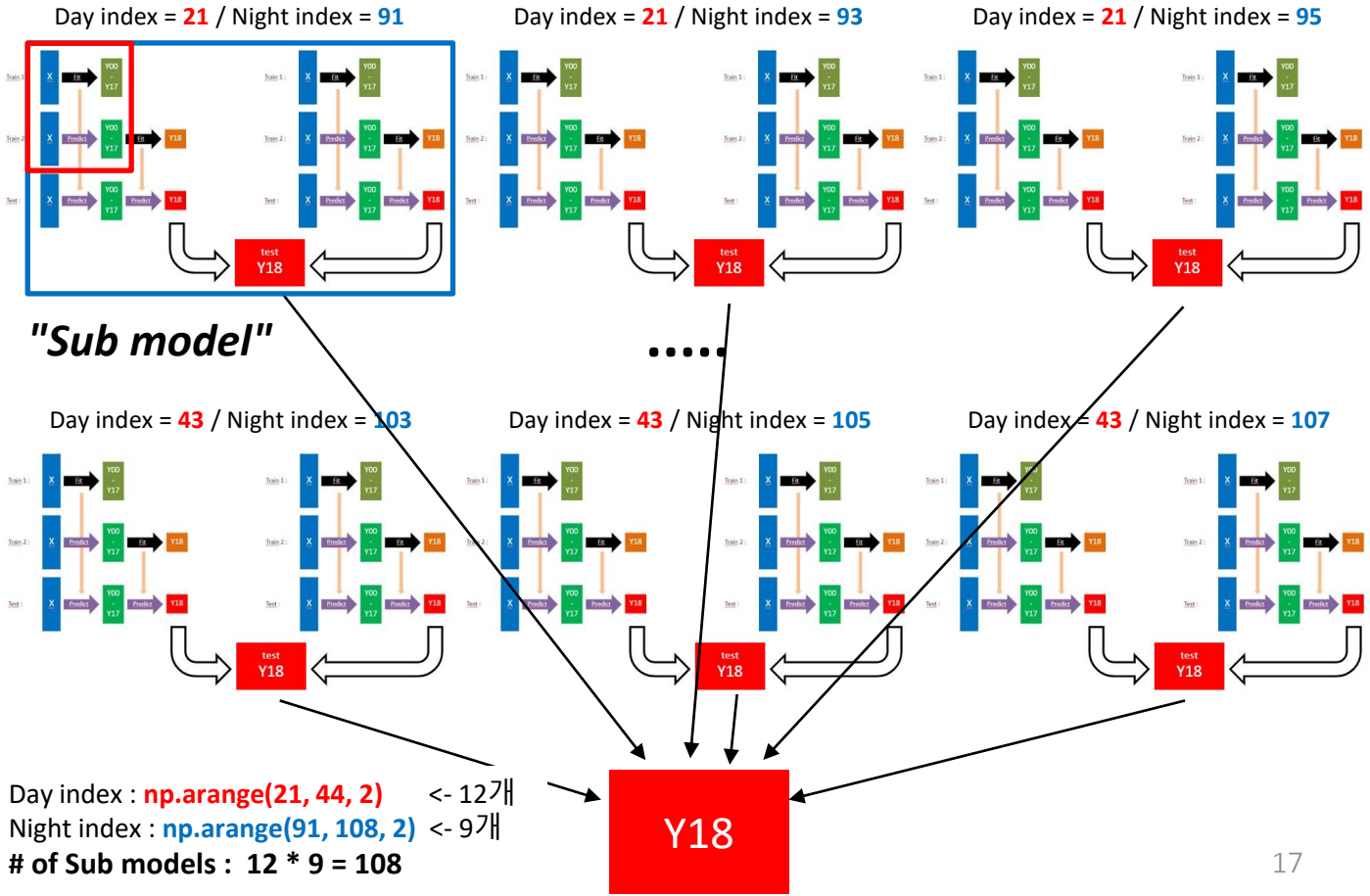
Night Time : [0 – 33] & [101 - 143]



2. 모델 구축 & 검증 : Single Model



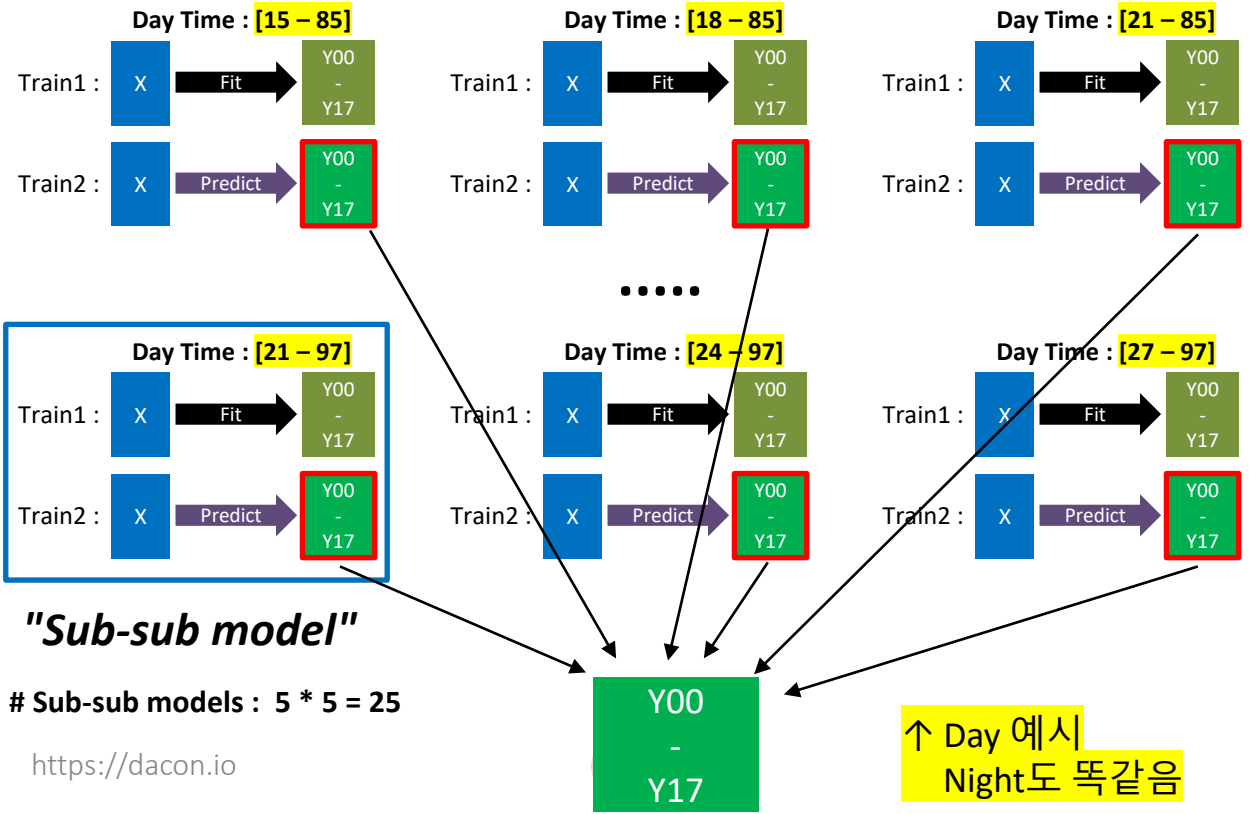
2. 모델 구축 & 검증 : Multi Model



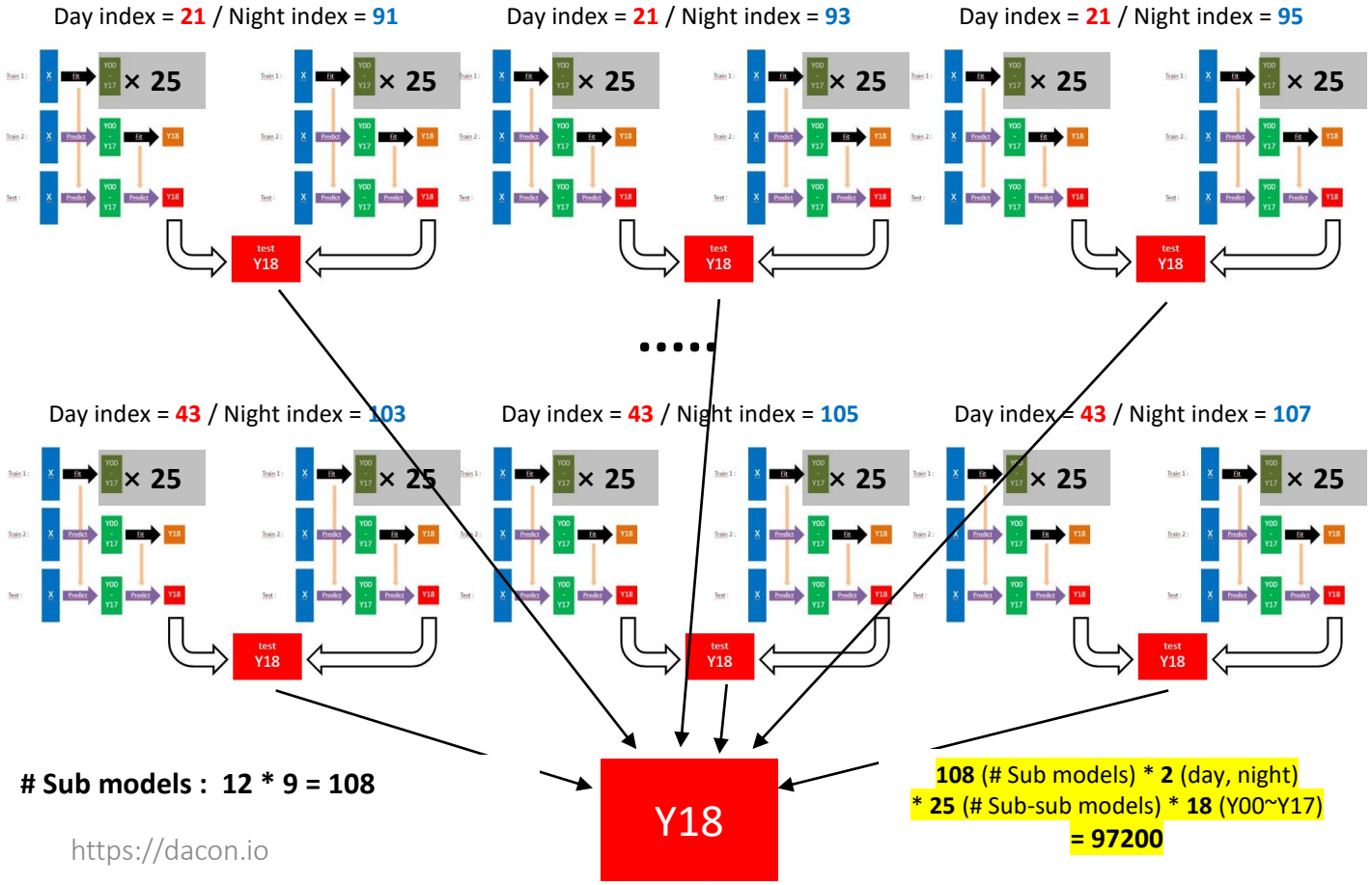
2. 모델 구축 & 검증 : Single Model

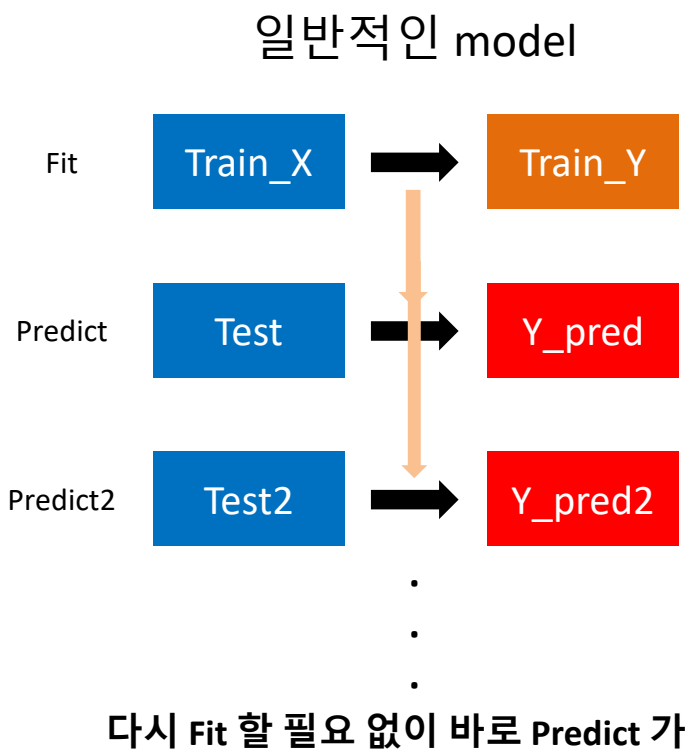
Day index extended = [15, 18, 21, 24, 27] / Night index extended = [85, 88, 91, 94, 97]

np.meshgrid([15, 18, 21, 24, 27], [85, 88, 91, 94, 97]) <- 총 25개



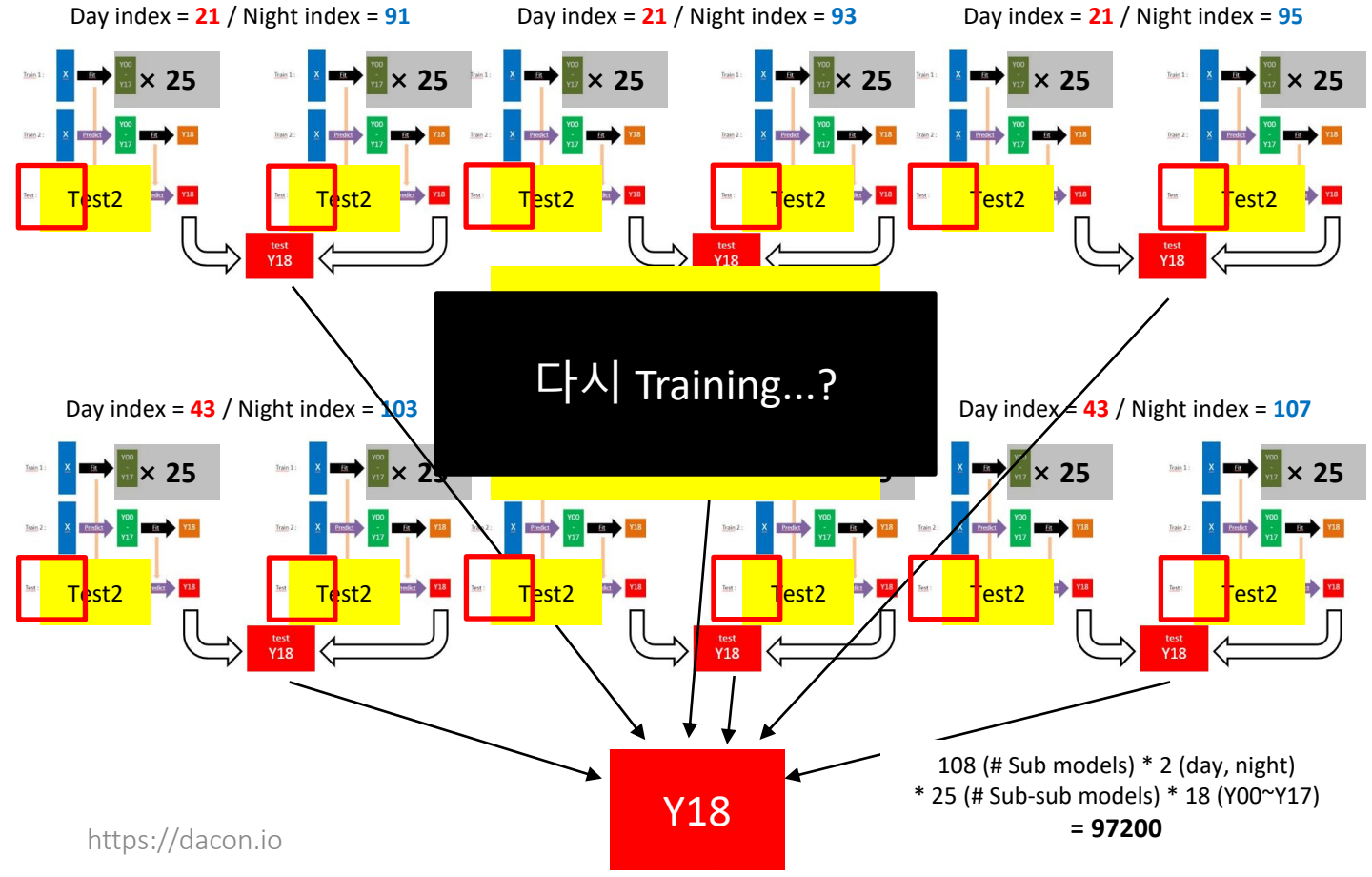
2. 모델 구축 & 검증 : Multi Model





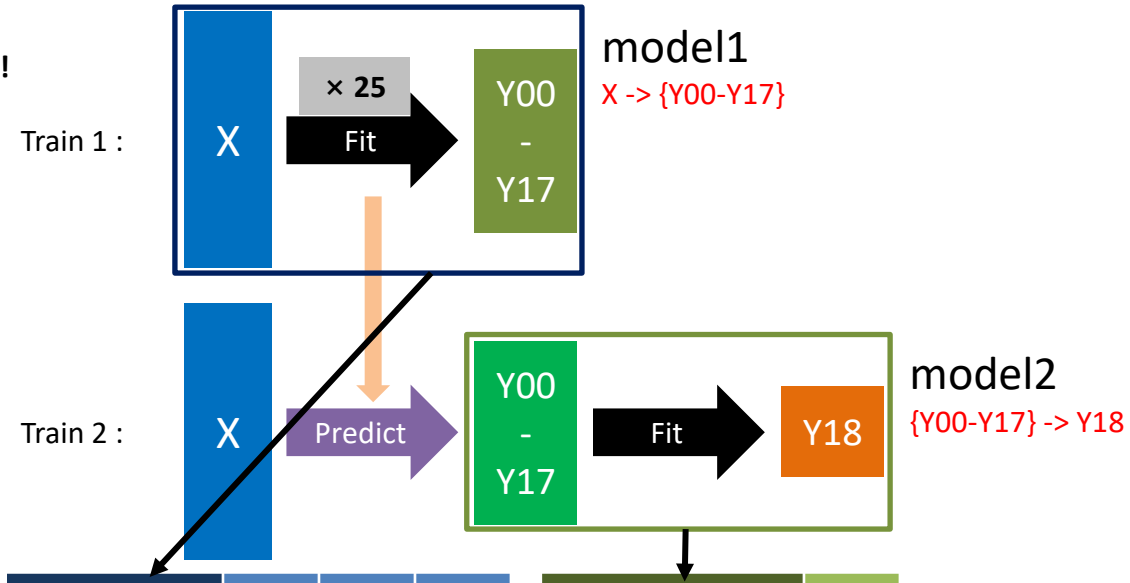
다시 Fit 할 필요 없이 바로 Predict 가능

2. 모델 구축 & 검증 : Multi Model



2. 모델 구축 & 검증 : Multi Model

결과값이 아닌
Coefficients의 평균!



model1.coef_	Y00	Y01	Y02
X00	0.1	2.1	1.5
X01	2.7	3.4	5.1
X02	6.3	5.7	3.3
Ys_Intercept	15.0	13.3	16.8

X -> {Y00-Y17}

model2.coef_	Y18
Y00	0.5
Y01	1.2
Y02	-1.3
Y18_Intercept	2.1

{Y00-Y17} -> Y18

2. 모델 구축 & 검증 : Multi Model

[model1 + model2]

X -> {Y00-Y17} -> Y18

model1.coef_	Y00	Y01	Y02
X00	0.1	2.1	1.5
X01	2.7	3.4	5.1
X02	6.3	5.7	3.3
Ys_Intercept	15.0	13.3	16.8

model2.coef_	Y18
Y00	0.5
Y01	1.2
Y02	-1.3
Y18_Intercept	2.1

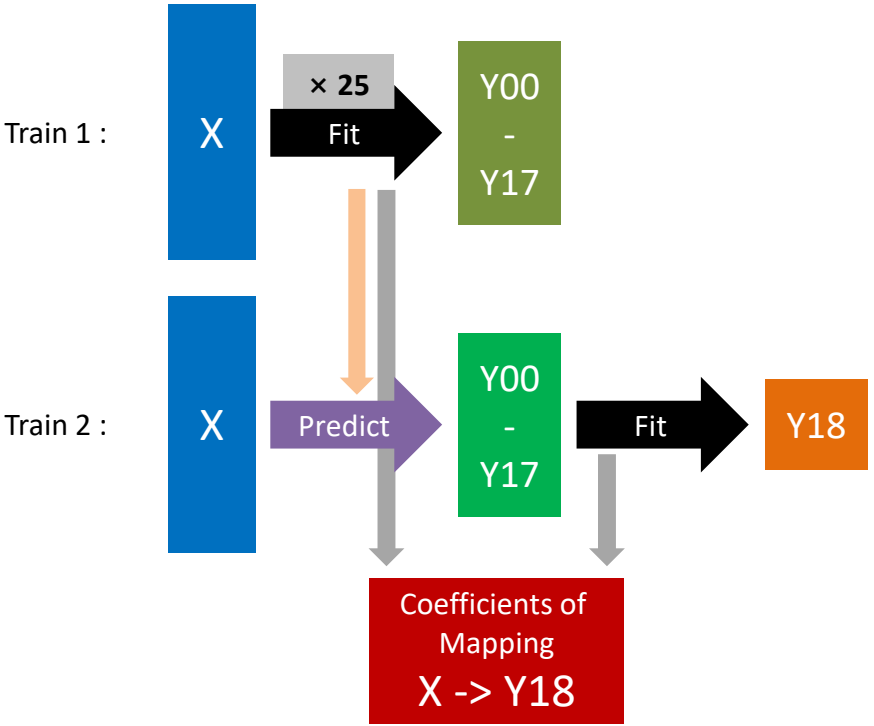
× 0.5 × 1.2 × -1.3

model1.coef_	Y00	Y01	Y02
X00	0.05	2.52	-1.95
X01	1.35	4.08	-6.63
X02	3.15	6.84	-4.29
Ys_Intercept	7.5	15.96	-21.84

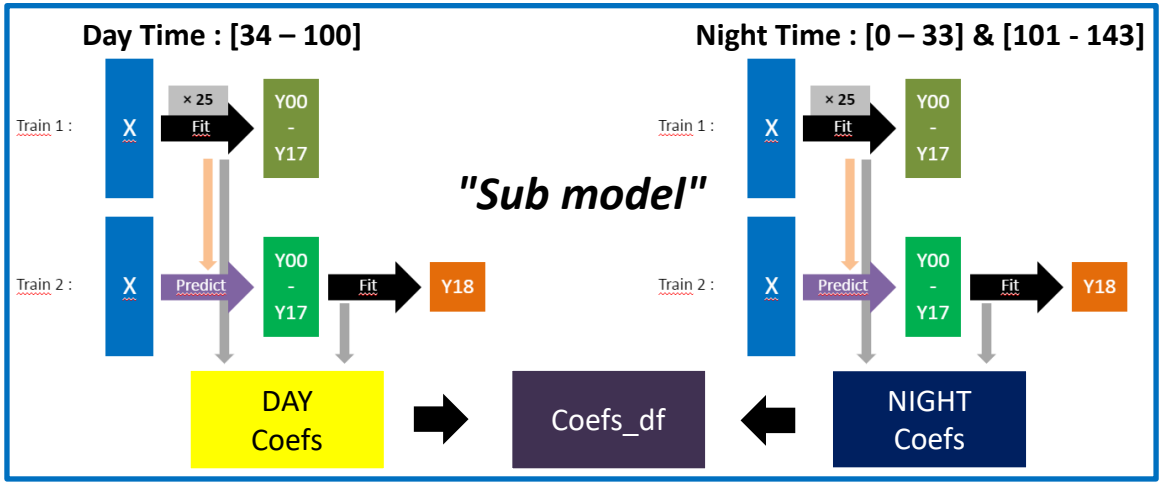
model1 + model2	Y18
X00	0.62
X01	2
X02	7
Ys_Intercept	2

Coefficients of Mapping
X -> Y18

2. 모델 구축 & 검증 : Multi Model



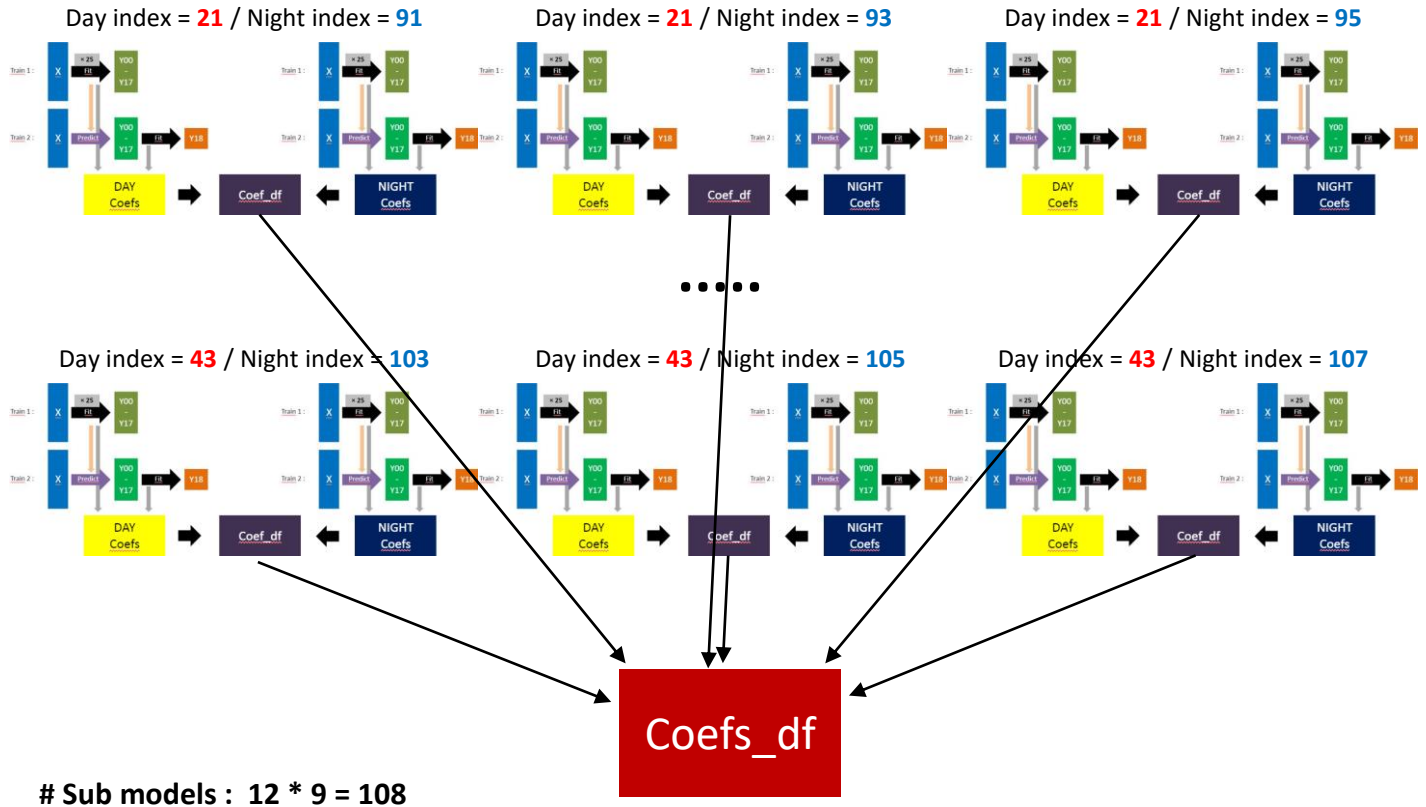
2. 모델 구축 & 검증 : Multi Model



Coef_df :

	Time	X00	X01	…	neg_insol	Ys_intercept	Y18_intercept
Night coefs	0	1.3	0.0	…	0.0	3.8	1.2
	1	1.3	0.0	…	0.0	3.8	1.2
	…	1.3	0.0	…	0.0	3.8	1.2
Day coefs	34	0.0	5.5	…	-2.4	1.3	0.5
	…	0.0	5.5	…	-2.4	1.3	0.5
	100	0.0	5.5	…	-2.4	1.3	0.5
Night coefs	…	1.3	0.0	…	0.0	3.8	1.2
	143	1.3	0.0	…	0.0	3.8	1.2

2. 모델 구축 & 검증 : Multi Model



 # Sub models : 12 * 9 = 108

2. 모델 구축 & 검증 : Multi Model

Coefs_df :

Time	X00	X01	...	neg_insol	Ys_intercept	Y18_intercept
0	1.3	0.0	···	0.0	3.8	1.2
1	1.3	0.0	···	0.0	3.8	1.2
···	1.3	0.0	···	0.0	3.8	1.2
34	0.3	7.8	···	-0.6	2.5	1.1
···	0.0	5.5	···	-2.4	1.3	0.5
100	0.9	4.2	···	-0.3	3.1	0.8
···	1.3	0.0	···	0.0	3.8	1.2
143	1.3	0.0	···	0.0	3.8	1.2

Test:
(after preprocessing)

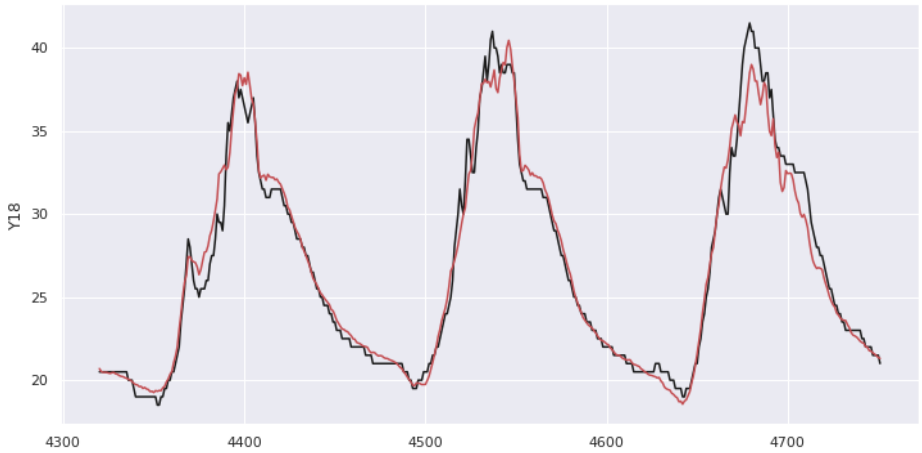
time	id	X00	X01	···	neg_insol	Ys_intercept	Y18_intercept
100	12340	26.1	45.0	···	1.1	1.0	1.0
101	12341	26.2	44.8	···	1.0	1.0	1.0
102	12342	26.4	44.2	···	0.8	1.0	1.0

Dot Product!

3. 결과 및 결언 : Conclusion

```
In [50]: %%time
         # day_start, day_ends, night_start, night_end 값은 리더보드 점수가 가장 높은 것으로 선택함.
         train2_Y18_pred, test_Y18, multi_X_Y18_coefs = multi_model(day_start=21, day_end=45, night_start=91, night_end=109, predict=True,
                                                                    train1_X=train1_X, train1_Ys=train1_Ys,
                                                                    train2_X=train2_X, train2_Y18=train2_Y18, test_X=test_X)

         CPU times: user 2h 20min 7s, sys: 2h 50min 18s, total: 5h 10min 25s
         Wall time: 38min 52s

In [51]: # Training error
         mean_squared_error(train

Out[51]: 1.1617327416529861
```

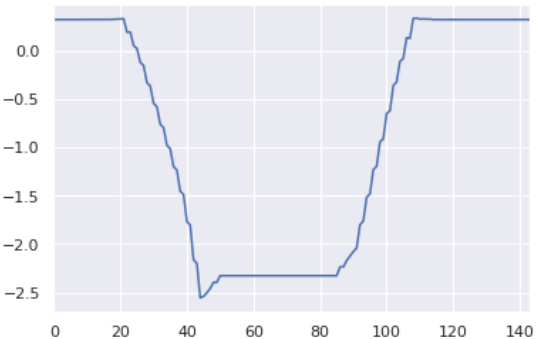
```
In [52]: # Visualization : train2_Y18 prediction
         fig, ax = plt.subplots(figsize=(12, 6))
         sns.lineplot(x=train2_Y18.index, y=train2_Y18, ax=ax, c='k')
         sns.lineplot(x=train2_Y18.index, y=train2_Y18_pred, ax=ax, c='r')

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb8fc8b1ed0>
```



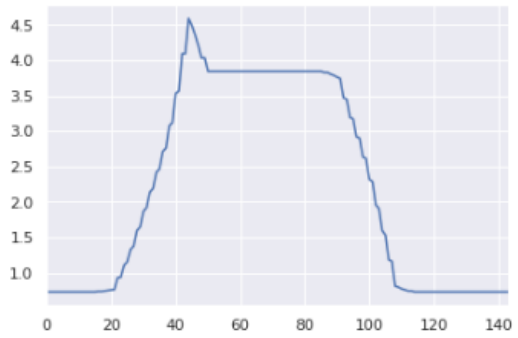
Public score : 1.02845
Private score : 2.67974

3. 결과 및 결언 : Conclusion

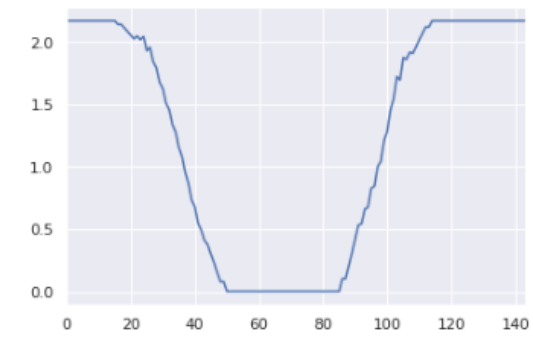
```
In [33]: multi_X_Y18_coefs['neg_insol'].plot()
Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2a97cf28d0>
```



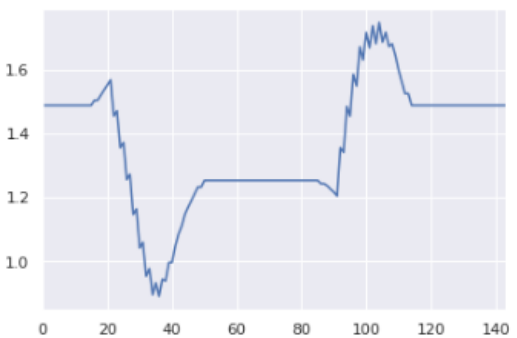

```
In [34]: multi_X_Y18_coefs['X34'].plot()
Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2a
```




```
In [35]: multi_X_Y18_coefs['temp_mean'].plot()
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2a
```




```
In [36]: multi_X_Y18_coefs['X07'].plot()
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7f619
```



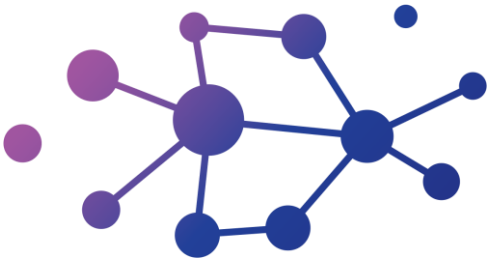
3. 결과 및 결언 : Suggestion

다양한 알고리즘의 시도

+ Transfer Learning ?

맑은 날 vs 흐린 날을 구분할 수 있는

좀 더 정교한 feature를 도입했더라면...



THANK YOU