8. 에너지 부하 예측 실습

박경규

냉난방 부하 예측모델 개발

energy_efficiency_modeling.ipynb

냉난방 부하 예측모델

건물 에너지 소비는 전체 에너지 사용의 약 40%를 차지합니다. 설계 단계에서 건물의 냉난방 부하를 예측하는 것은 에너지 효율화를 위해 매우 중요합니다. 건물 에너지 성능에 대한 데이터셋을 기반으로 건물의 냉난방 부하를 예측하기 위해 다양한 모델을 개발합니다.

Input

- Relative Compactness
- Surface Area - m^2
- Wall Area - m^2
- Roof Area - m^2
- Overall Height - m
- Orientation - 2:North, 3:East, 4:South, 5:West
- Glazing Area - 0%, 10%, 25%, 40% (of floor area)
- Glazing Area Distribution (Variance)
  - 1:Uniform, 2:North, 3:East, 4:South, 5:West
- Heating Load - kWh
- Cooling Load - kWh

Output

- 난방부하
- 냉방부하

냉난방 부하 예측모델

```
[1]  import os
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     import warnings
     warnings.filterwarnings('ignore')
```

```
[2]  data = pd.read_csv('building_energy_efficiency.csv')
```

```
[3]  data.head()
```

	Relative Compactness	Surface Area	Wall Area	Roof Area	Overall Height	Orientation	Glazing Area	Glazing Area Distribution	Heating Load	Cooling Load
0	0.7638	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.9800	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.9800	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.9800	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.9000	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28

냉난방 부하 예측모델

```
[4]  data.shape
```

```
(768, 10)
```

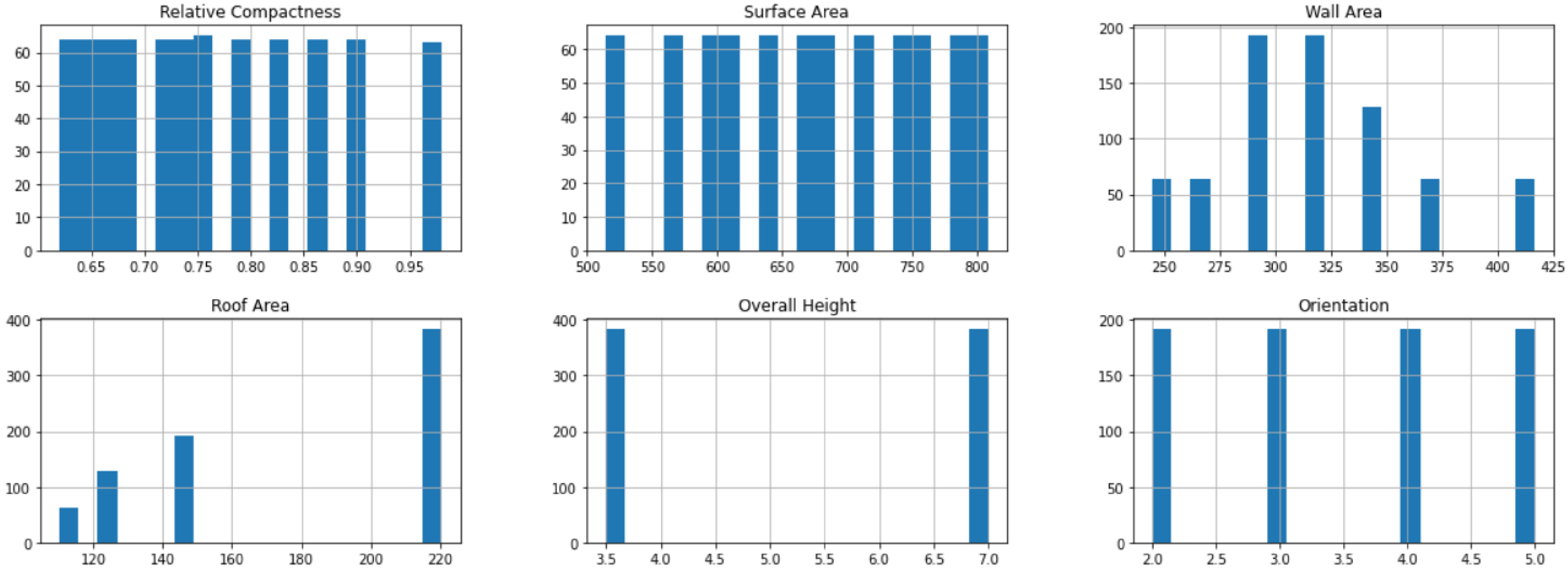
```
[5]  data.isnull().sum()
```

```
Relative Compactness         0
Surface Area                 0
Wall Area                    0
Roof Area                    0
Overall Height               0
Orientation                  0
Glazing Area                 0
Glazing Area Distribution    0
Heating Load                 0
Cooling Load                 0
dtype: int64
```

냉난방 부하 예측모델

각 데이터의 분포 체크

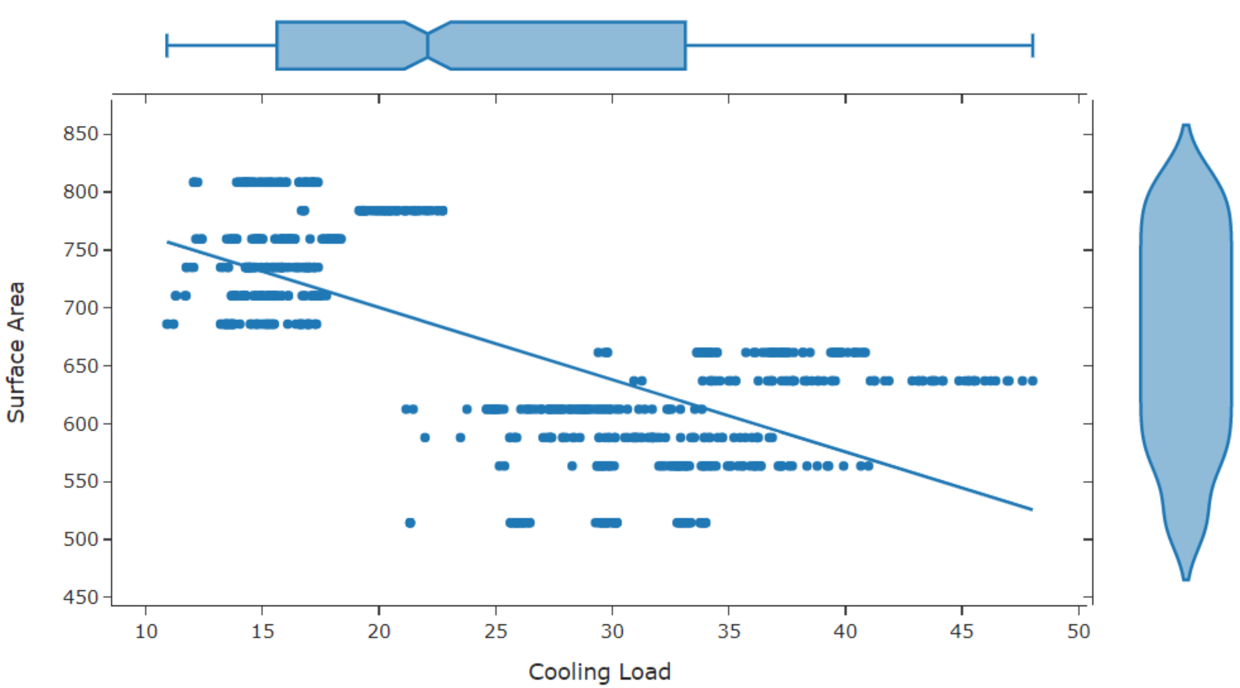
```
[6]  data.hist(bins=20, figsize=(20,15))
     plt.show()
```



냉난방 부하 예측모델

상관관계 분석

```
[7]  import plotly.express as px

     yprop = 'Surface Area'
     xprop = 'Cooling Load'
     h= None
     px.scatter(data, x=xprop, y=yprop, color=h, marginal_y="violin", marginal_x="box", trendline="ols",
```

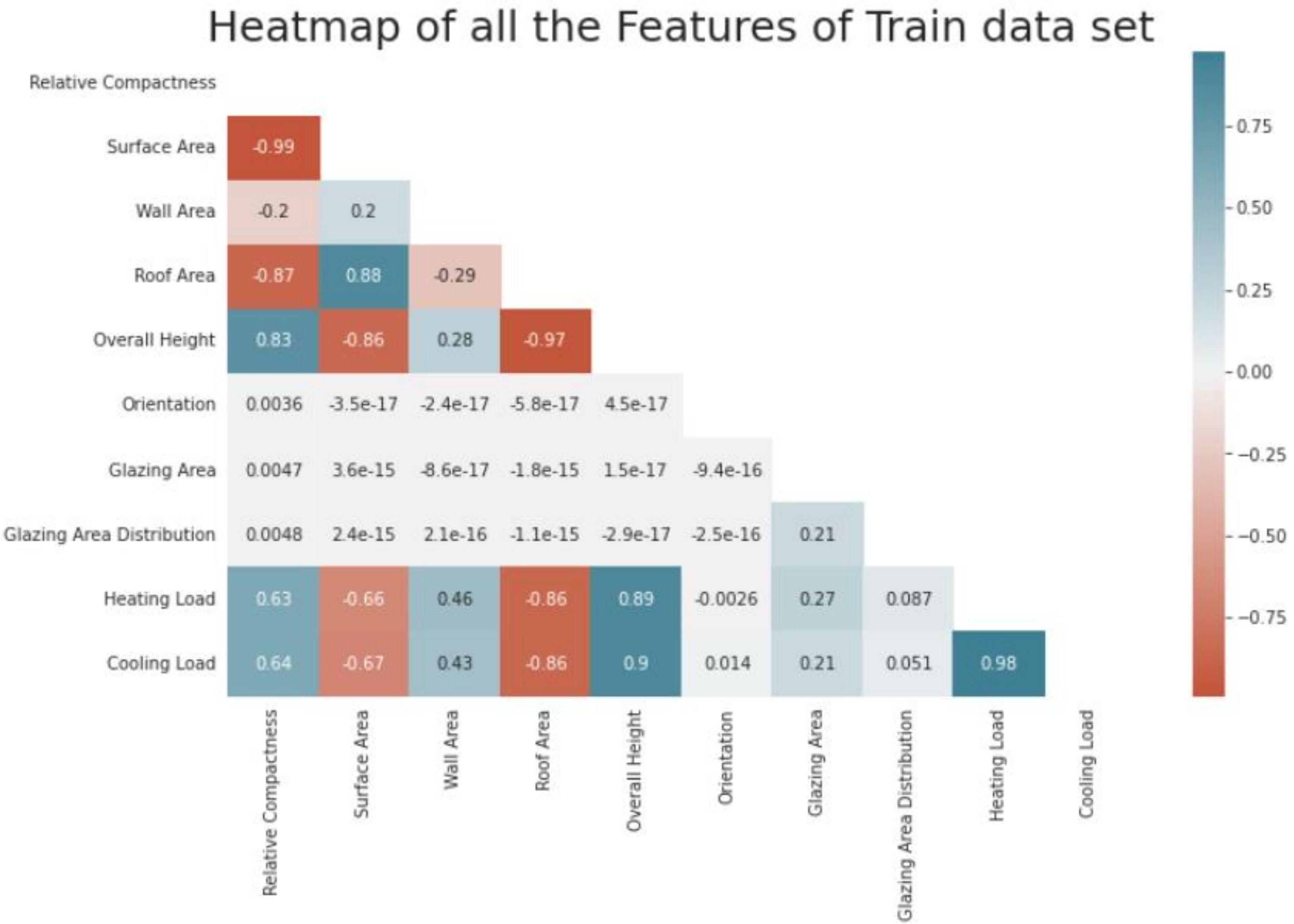


냉난방 부하 예측모델

명확한 상관관계를 찾기위해 상관관계 행렬 확인

```
[9]  import matplotlib.pyplot as plt
     import matplotlib.style as style
     import seaborn as sns
     style.use('ggplot')
     sns.set_style('whitegrid')

     plt.subplots(figsize = (12,7))
     ## Plotting heatmap. # Generate a mask for the upper triangle (taken from seabor
     mask = np.zeros_like(data.corr(), dtype=np.bool)
     mask[np.triu_indices_from(mask)] = True
     sns.heatmap(data.corr(), cmap=sns.diverging_palette(20, 220, n=200), annot=True,
     plt.title("Heatmap of all the Features of Train data set", fontsize = 25);
```

냉난방 부하 예측모델


Heatmap of all the Features of Train data set
Relative Compactness
Surface Area
Wall Area
Roof Area
Overall Height
Orientation
Glazing Area
Glazing Area Distribution
Heating Load
Cooling Load
-0.99
-0.2
0.2
-0.87
0.88
-0.29
0.83
-0.86
0.28
-0.97
0.0036
-3.5e-17
-2.4e-17
-5.8e-17
4.5e-17
0.0047
3.6e-15
-8.6e-17
-1.8e-15
1.5e-17
-9.4e-16
0.0048
2.4e-15
2.1e-16
-1.1e-15
-2.9e-17
-2.5e-16
0.21
0.63
-0.66
0.46
-0.86
0.89
-0.0026
0.27
0.087
0.64
-0.67
0.43
-0.86
0.9
0.014
0.21
0.051
0.98
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

냉난방 부하 예측모델

```
[11] from scipy.stats import randint as sp_randint
     from catboost import CatBoostRegressor
     from sklearn.model_selection import GridSearchCV
     from keras.layers import Dense
     from keras.models import Sequential
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.neighbors import KNeighborsRegressor
     from sklearn.neural_network import MLPRegressor
     from sklearn.ensemble import GradientBoostingRegressor,AdaBoostRegressor
     from sklearn.ensemble import BaggingRegressor, RandomForestRegressor
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import train_test_split
     from sklearn.multioutput import MultiOutputRegressor
     from sklearn.preprocessing import MinMaxScaler
     from sklearn.svm import SVC
     from sklearn.svm import SVR
     from sklearn.metrics import accuracy_score, f1_score
     from sklearn.metrics import r2_score
     from sklearn.metrics import roc_auc_score
```

냉난방 부하 예측모델

```
[12]  X = data[['Relative Compactness', 'Surface Area', 'Wall Area',
               'Roof Area', 'Overall Height', 'Orientation',
               'Glazing Area', 'Glazing Area Distribution']]
      Y = data[['Heating Load', 'Cooling Load']]
      Y1= data[['Heating Load']]
      Y2= data[['Cooling Load']]
```

냉난방 부하 예측모델

- 데이터 세트를 훈련 및 테스트 세트로 분할.
- 특징 스케일링 또는 데이터 정규화는 데이터의 독립 변수 또는 특징의 범위를 정규화하는 데 사용되는 방법입니다.
- 따라서 독립 변수에서 값이 많이 다를 때 모든 값이 비교 가능한 범위에 유지되도록 특성 스케일링을 사용합니다.

```
[13] X_train, X_test, y1_train, y1_test, y2_train, y2_test = train_test_split(
         X, Y1, Y2, test_size=0.33, random_state = 20)

     MinMax = MinMaxScaler(feature_range= (0,1))
     X_train = MinMax.fit_transform(X_train)
     X_test = MinMax.transform(X_test)
```

냉난방 부하 예측모델

모델링

각 모델의 결과를 저장할 DataFrame을 만듭니다.

```
[14]  Acc = pd.DataFrame(index=None, columns=['model',
                                            'train_Heating','test_Heating',
                                            'train_Cooling','test_Cooling'])
```

```
[15]  regressors = [['SVR',SVR()],
                    ['DecisionTreeRegressor',DecisionTreeRegressor()],
                    ['KNeighborsRegressor', KNeighborsRegressor()],
                    ['RandomForestRegressor', RandomForestRegressor()],
                    ['MLPRegressor',MLPRegressor()],
                    ['AdaBoostRegressor',AdaBoostRegressor()],
                    ['GradientBoostingRegressor',GradientBoostingRegressor()]]
```

냉난방 부하 예측모델

[16]
```
for mod in regressors:
    name = mod[0]
    model = mod[1]

    model.fit(X_train, y1_train)
    actr1 = r2_score(y1_train, model.predict(X_train))
    acte1 = r2_score(y1_test, model.predict(X_test))

    model.fit(X_train, y2_train)
    actr2 = r2_score(y2_train, model.predict(X_train))
    acte2 = r2_score(y2_test, model.predict(X_test))

    Acc = Acc.append(pd.Series({'model':name, 'train_Heating':actr1,
                                'test_Heating':acte1,'train_Cooling':actr2,
                                'test_Cooling':acte2}),ignore_index=True )

Acc.sort_values(by='test_Cooling')
```

냉난방 부하 예측모델

	model	train_Heating	test_Heating	train_Cooling	test_Cooling
4	MLPRegressor	0.865059	0.872689	0.815817	0.835746
0	SVR	0.930662	0.910593	0.892578	0.887385
2	KNeighborsRegressor	0.945989	0.904490	0.926869	0.888896
5	AdaBoostRegressor	0.958150	0.955481	0.943954	0.941024
1	DecisionTreeRegressor	1.000000	0.997228	1.000000	0.948199
3	RandomForestRegressor	0.999405	0.997419	0.995561	0.964753
6	GradientBoostingRegressor	0.998173	0.997641	0.979423	0.976044

냉난방 부하 예측모델

모델 파라미터 튜닝

- Boosting machine learning 알고리즘은 단순한 알고리즘보다 더 나은 정확도를 제공하기 때문에 많이 사용됩니다.
- 알고리즘의 성능은 하이퍼파라미터에 따라 다릅니다.
- 최적의 parameter는 더 높은 정확도를 달성하는 데 도움이 될 수 있습니다.
- 수동으로 hyper parameter를 찾는 것은 지루하고 계산 비용이 많이 듭니다.
- 하이퍼파라미터 튜닝의 자동화가 중요합니다.
- RandomSearch, GridSearchCV, Bayesian optimization은 하이퍼파라미터를 최적화하는 데 사용됩니다.

냉난방 부하 예측모델

```
[17] param_grid = [{"learning_rate": [0.01, 0.02, 0.1],
                    "n_estimators":[150, 200, 250], "max_depth": [4, 5, 6],
                    "min_samples_split":[1, 2, 3], "min_samples_leaf":[2, 3],
                    "subsample":[1.0, 2.0]}]

     GBR = GradientBoostingRegressor()
     grid_search_GBR = GridSearchCV(GBR, param_grid, cv=10,
                                    scoring='neg_mean_squared_error')
     grid_search_GBR.fit(X_train, y2_train)

     print("R-Squared::{}".format(grid_search_GBR.best_score_))
     print("Best Hyperparameters::₩n{}".format(grid_search_GBR.best_params_))
```

```
R-Squared::-1.0560518941222068
Best Hyperparameters::
{'learning_rate': 0.1, 'max_depth': 5, 'min_samples_leaf': 3, 'min_samples_split': 3,
```

냉난방 부하 예측모델

A. Decision Tree Regressor parameters turning

```
[18]  DTR = DecisionTreeRegressor()
      param_grid = {"criterion": ["mse", "mae"],"min_samples_split": [14, 15, 16, 17],
                    "max_depth": [5, 6, 7],"min_samples_leaf": [4, 5, 6],
                    "max_leaf_nodes": [29, 30, 31, 32],}

      grid_cv_DTR = GridSearchCV(DTR, param_grid, cv=5)

      grid_cv_DTR.fit(X_train,y2_train)
      print("R-Squared::{}".format(grid_cv_DTR.best_score_))
      print("Best Hyperparameters::₩n{}".format(grid_cv_DTR.best_params_))
```

```
R-Squared::0.9599150110108299
Best Hyperparameters::
{'criterion': 'mse', 'max_depth': 6, 'max_leaf_nodes': 32, 'min_samples_leaf': 5
```

냉난방 부하 예측모델

```
[19] DTR = DecisionTreeRegressor()
     param_grid = {"criterion": ["mse", "mae"],"min_samples_split": [14, 15, 16, 17],
                   "max_depth": [5, 6, 7],"min_samples_leaf": [4, 5, 6],
                   "max_leaf_nodes": [29, 30, 31, 32],}

     grid_cv_DTR = GridSearchCV(DTR, param_grid, cv=5)

     grid_cv_DTR.fit(X_train,y2_train)
     print("R-Squared::{}".format(grid_cv_DTR.best_score_))
     print("Best Hyperparameters::₩n{}".format(grid_cv_DTR.best_params_))
```

```
R-Squared::0.9598595926917322
Best Hyperparameters::
{'criterion': 'mse', 'max_depth': 6, 'max_leaf_nodes': 31, 'min_samples_leaf': 5
```

냉난방 부하 예측모델
B. Tune Random Forests Parameters

[20]
```
from sklearn.model_selection import GridSearchCV
param_grid = [{'n_estimators': [350, 400, 450], 'max_features': [1, 2],
               'max_depth': [85, 90, 95]}]

RFR = RandomForestRegressor(n_jobs=-1)
grid_search_RFR = GridSearchCV(RFR, param_grid, cv=10,
                               scoring='neg_mean_squared_error')
grid_search_RFR.fit(X_train, y2_train)

print("R-Squared::{}".format(grid_search_RFR.best_score_))
print("Best Hyperparameters::\n{}".format(grid_search_RFR.best_params_))
```

```
R-Squared::-2.7105394110927175
Best Hyperparameters::
{'max_depth': 85, 'max_features': 1, 'n_estimators': 350}
```

냉난방 부하 예측모델

C. Gradient Boosting Regression - Hyperparameter Tuning

```
[22] GBR = GradientBoostingRegressor(learning_rate=0.1,n_estimators=250,
                                     max_depth=5, min_samples_split=3,
                                     min_samples_leaf=2, subsample=1.0)

     GBR.fit(X_train,y1_train)
     print("R-Squared on train dataset={}".format(GBR.score(X_test,y1_test)))

     GBR.fit(X_train,y2_train)
     print("R-Squaredon test dataset={}".format(GBR.score(X_test,y2_test)))
```

```
R-Squared on train dataset=0.9986725708412564
R-Squaredon test dataset=0.9915616795439752
```

냉난방 부하 예측모델

D. CatBoostRegressor

```
[23] import warnings
     warnings.filterwarnings("ignore")
     from sklearn import datasets
     from sklearn.model_selection import train_test_split
     from sklearn.model_selection import GridSearchCV
     from catboost import CatBoostRegressor

     model_CBR = CatBoostRegressor()
     parameters = {'depth':[8, 10],'iterations':[10000],'learning_rate':[0.02,0.03],
                   'border_count':[5],'random_state': [42, 45]}

     grid = GridSearchCV(estimator=model_CBR, param_grid = parameters, cv = 2, n_jobs=-1)
     grid.fit(X_train, y2_train)
     print(" Results from Grid Search " )
     print("₩n The best estimator across ALL searched params:₩n", grid.best_estimator_)
     print("₩n The best score across ALL searched params:₩n", grid.best_score_)
     print("₩n The best parameters across ALL searched params:₩n", grid.best_params_)
```

냉난방 부하 예측모델

E. MLPRegressor

```
[24] MLPR = MLPRegressor(hidden_layer_sizes = [180,100,20],activation ='relu',
                         solver='lbfgs',max_iter = 10000,random_state = 0)
     MLPR.fit(X_train,y1_train)
     print("R-Squared on train dataset={}".format(MLPR.score(X_test,y1_test)))

     MLPR.fit(X_train,y2_train)
     print("R-Squaredon test dataset={}".format(MLPR.score(X_test,y2_test)))
```

```
R-Squared on train dataset=0.9978104168656019
R-Squaredon test dataset=0.9905898640304825
```

냉난방 부하 예측모델

모델 성능 Summary

```
[25]  Acc1 = pd.DataFrame(index=None, columns=['model','train_Heating','test_Heating',
                                                 'train_Cooling','test_Cooling'])
```

```
[26]  regressors1 = [['DecisionTreeRegressor',
                      DecisionTreeRegressor(criterion= 'mse', max_depth= 6,
                                            max_leaf_nodes= 30, min_samples_leaf= 5,
                                            min_samples_split= 17)],
                     ['RandomForestRegressor',
                      RandomForestRegressor(n_estimators = 450, max_features = 1,
                                           max_depth= 90, bootstrap= True)],
                     ['MLPRegressor',
                      MLPRegressor(hidden_layer_sizes = [180,100,20],activation ='relu',
                                  solver='lbfgs',max_iter = 10000,random_state = 0)],
                     ['GradientBoostingRegressor',
                      GradientBoostingRegressor(learning_rate=0.1,n_estimators=250,
                                                max_depth=5, min_samples_split=2,
                                                min_samples_leaf=3, subsample=1.0)]]
```

냉난방 부하 예측모델

```
[27] for mod in regressors1:
         name = mod[0]
         model = mod[1]

         model.fit(X_train,y1_train)
         actr1 = r2_score(y1_train, model.predict(X_train))
         acte1 = r2_score(y1_test, model.predict(X_test))

         model.fit(X_train,y2_train)
         actr2 = r2_score(y2_train, model.predict(X_train))
         acte2 = r2_score(y2_test, model.predict(X_test))

         Acc1 = Acc1.append(pd.Series({'model':name, 'train_Heating':actr1,
                               'test_Heating':acte1,'train_Cooling':actr2,
                               'test_Cooling':acte2}),ignore_index=True )
     Acc1.sort_values(by='test_Cooling')
```

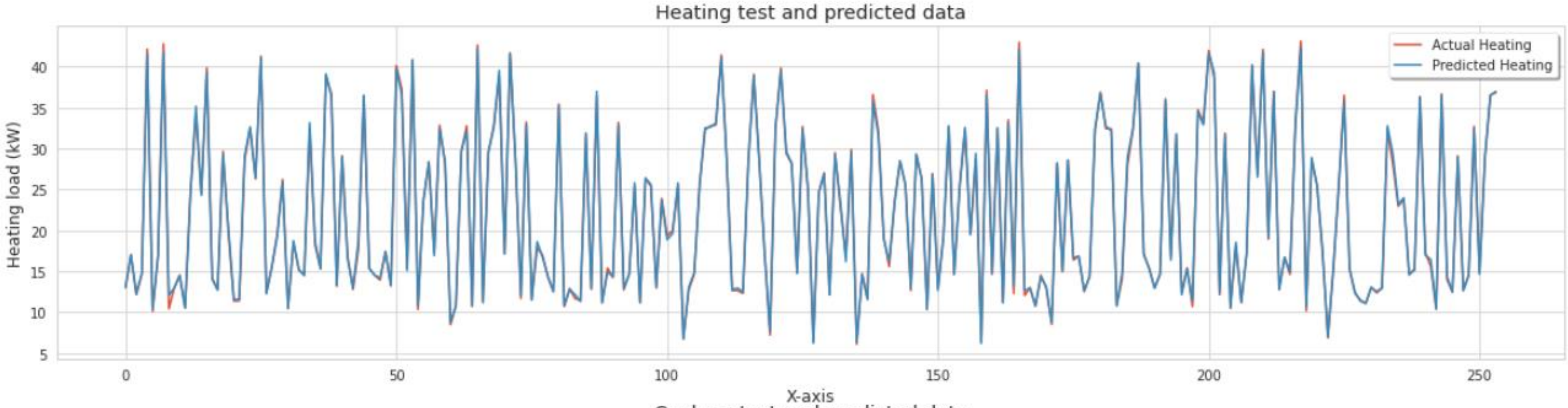
냉난방 부하 예측모델

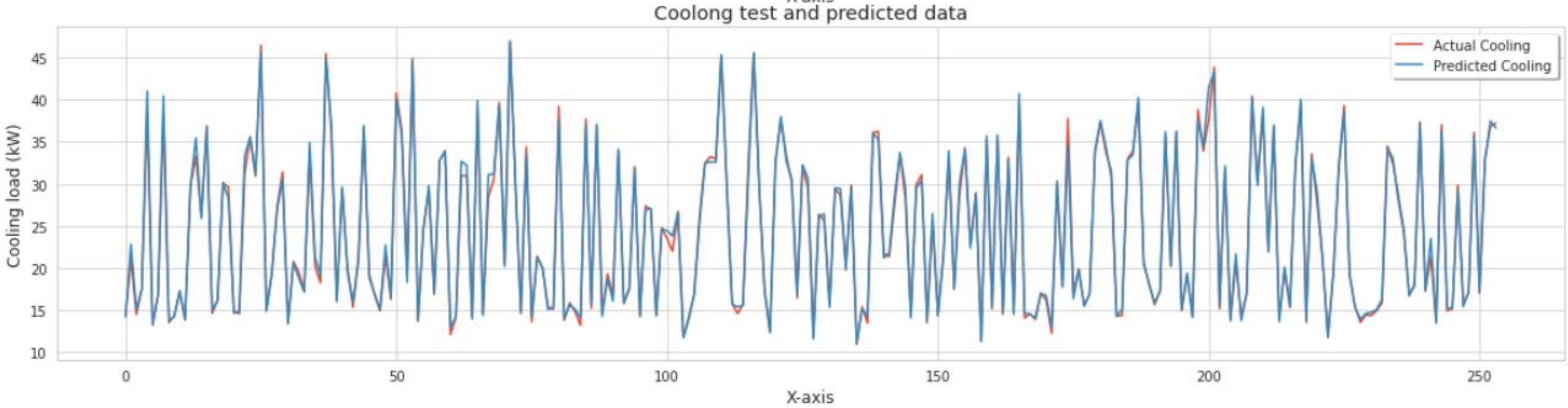
	model	train_Heating	test_Heating	train_Cooling	test_Cooling
0	DecisionTreeRegressor	0.994803	0.995168	0.967655	0.959112
1	RandomForestRegressor	0.998754	0.991773	0.996188	0.974144
2	MLPRegressor	0.999816	0.997810	0.999640	0.990590
3	GradientBoostingRegressor	0.999735	0.998553	0.998988	0.991933

냉난방 부하 예측모델

```
[30]  x_ax = range(len(y1_test))
      plt.figure(figsize=(20,10))
      plt.subplot(2,1,1)
      plt.plot(x_ax, y1_test, label="Actual Heating")
      plt.plot(x_ax, y1_pred, label="Predicted Heating")
      plt.title("Heating test and predicted data")
      plt.xlabel('X-axis')
      plt.ylabel('Heating load (kW)')
      plt.legend(loc='best',fancybox=True, shadow=True)
      plt.grid(True)

      plt.subplot(2,1,2)
      plt.plot(x_ax, y2_test, label="Actual Cooling")
      plt.plot(x_ax, y2_pred, label="Predicted Cooling")
      plt.title("Coolong test and predicted data")
      plt.xlabel('X-axis')
      plt.ylabel('Cooling load (kW)')
      plt.legend(loc='best',fancybox=True, shadow=True)
      plt.grid(True)
```

냉난방 부하 예측모델

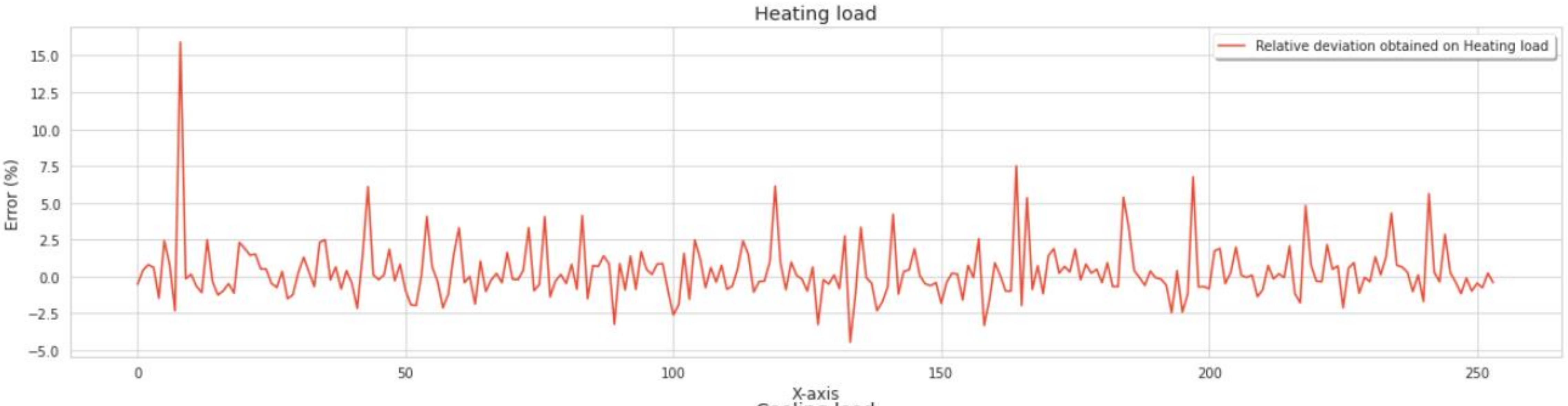

Heating test and predicted data
Actual Heating
Predicted Heating
Heating load (kW)
40
35
30
25
20
15
10
5
0
50
100
150
200
250
X-axis


Coolong test and predicted data
Actual Cooling
Predicted Cooling
Cooling load (kW)
45
40
35
30
25
20
15
10
0
50
100
150
200
250
X-axis

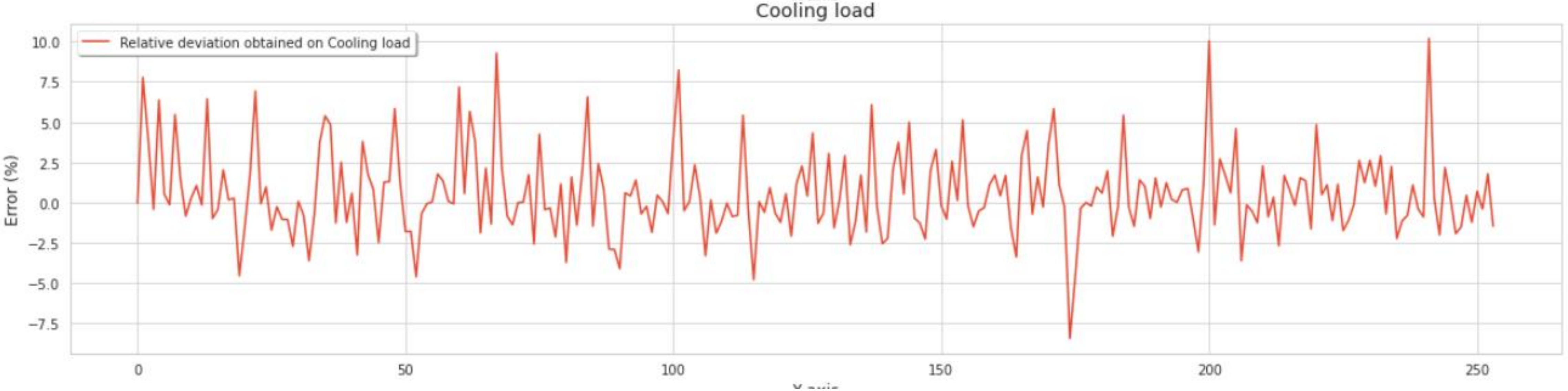
냉난방 부하 예측모델

```
[31] def AAD(y1_test, y1_pred):
         AAD =[]
         for i in range(len(y1_pred)):
             AAD.append((y1_pred[i] - y1_test.values[i])/y1_test.values[i]*100)
         return AAD

     x_ax = range(len(y1_test))
     plt.figure(figsize=(20,10))
     plt.subplot(2,1,1)
     plt.plot(x_ax, AAD(y1_test, y1_pred), label="Relative deviation obtained on Heating load")
     plt.title("Heating load")
     plt.xlabel('X-axis')
     plt.ylabel('Error (%)')
     plt.legend(loc='best',fancybox=True, shadow=True)
     plt.grid(True)

     plt.subplot(2,1,2)
     plt.plot(x_ax, AAD(y2_test, y2_pred), label="Relative deviation obtained on Cooling load")
     plt.title("Cooling load")
     plt.xlabel('X-axis')
     plt.ylabel('Error (%)')
     plt.legend(loc='best',fancybox=True, shadow=True)
     plt.grid(True)
```

냉난방 부하 예측모델



Heating load
Relative deviation obtained on Heating load
Error (%)
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
50
100
150
200
250
X-axis



Cooling load
Relative deviation obtained on Cooling load
Error (%)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0
50
100
150
200
250

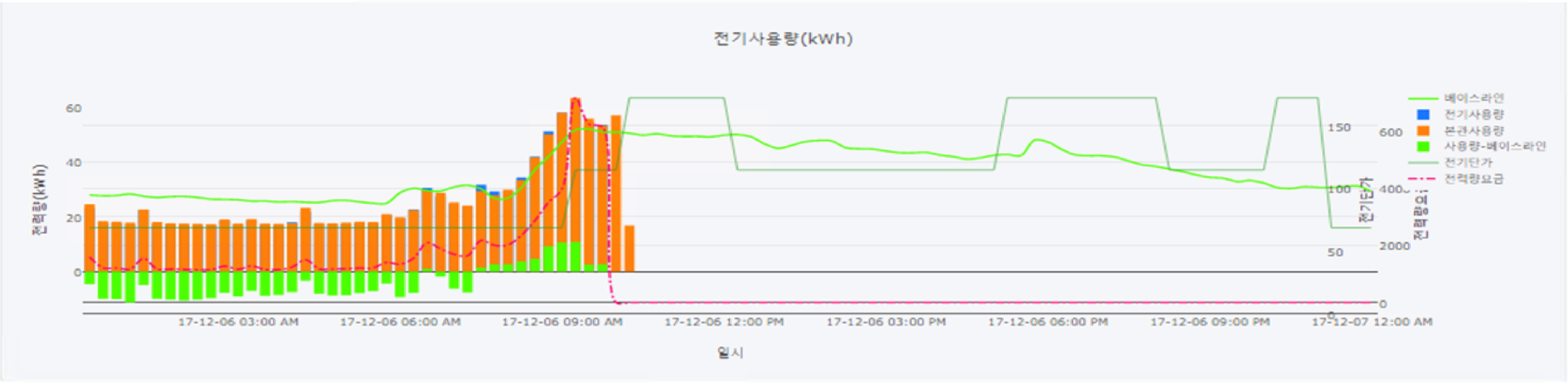
에너지 사용량 예측모델 개발

energy_usage_prediction.ipynb

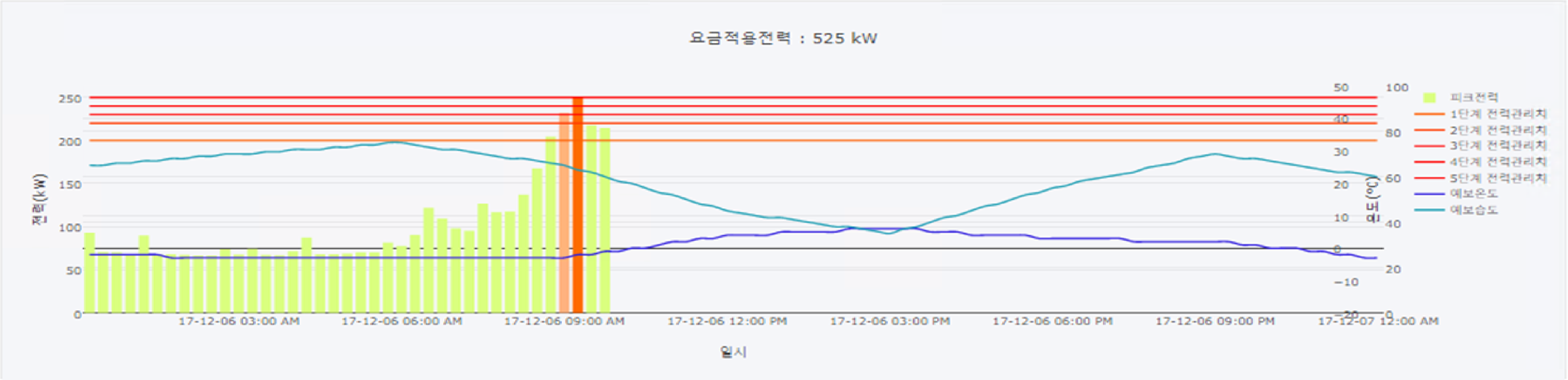
에너지 사용량 예측모델

에너지 사용량 예측은 건물 에너지 최적화에 필수적인 기본기능입니다.

전기 사용량 : 베이스라인 대비 증감 -170kWh

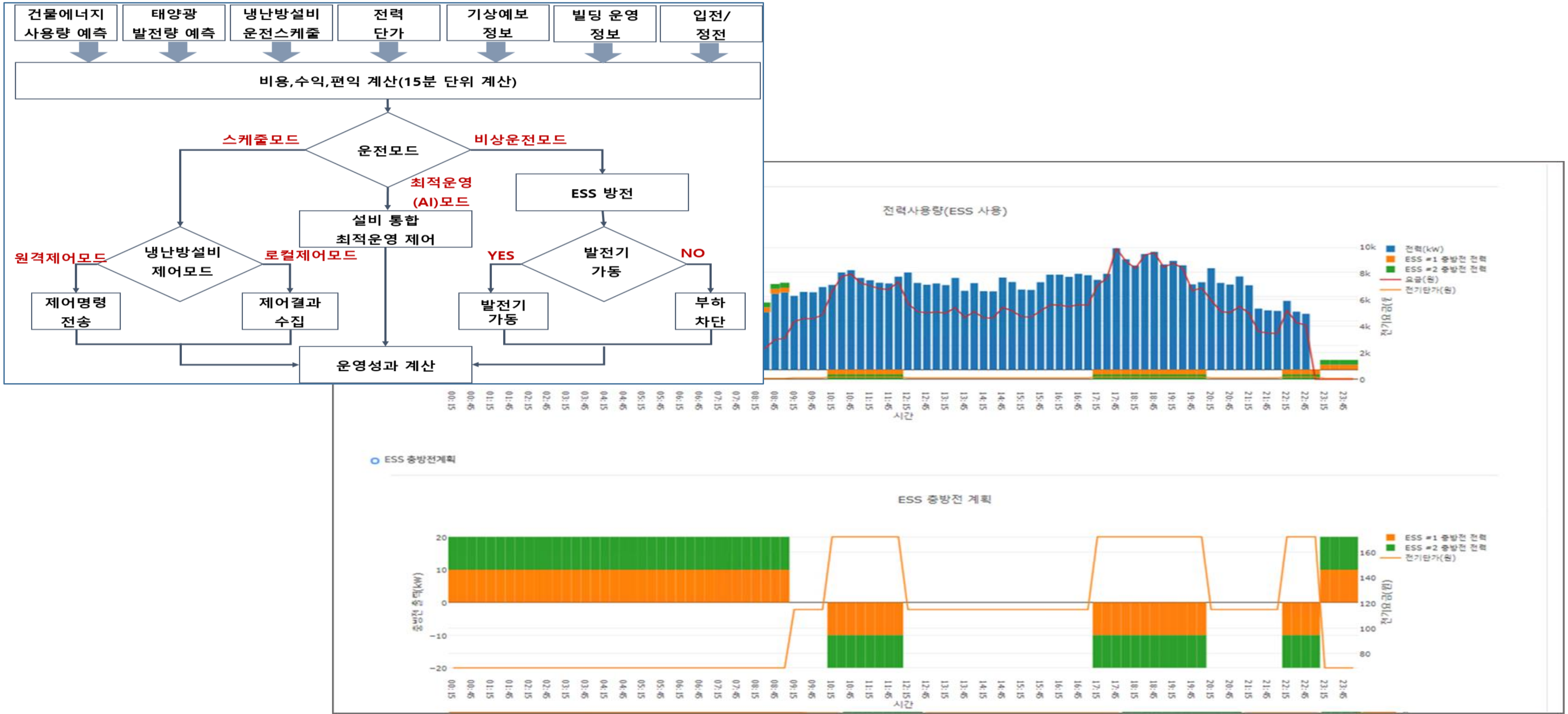


피크전력



에너지 사용량 예측

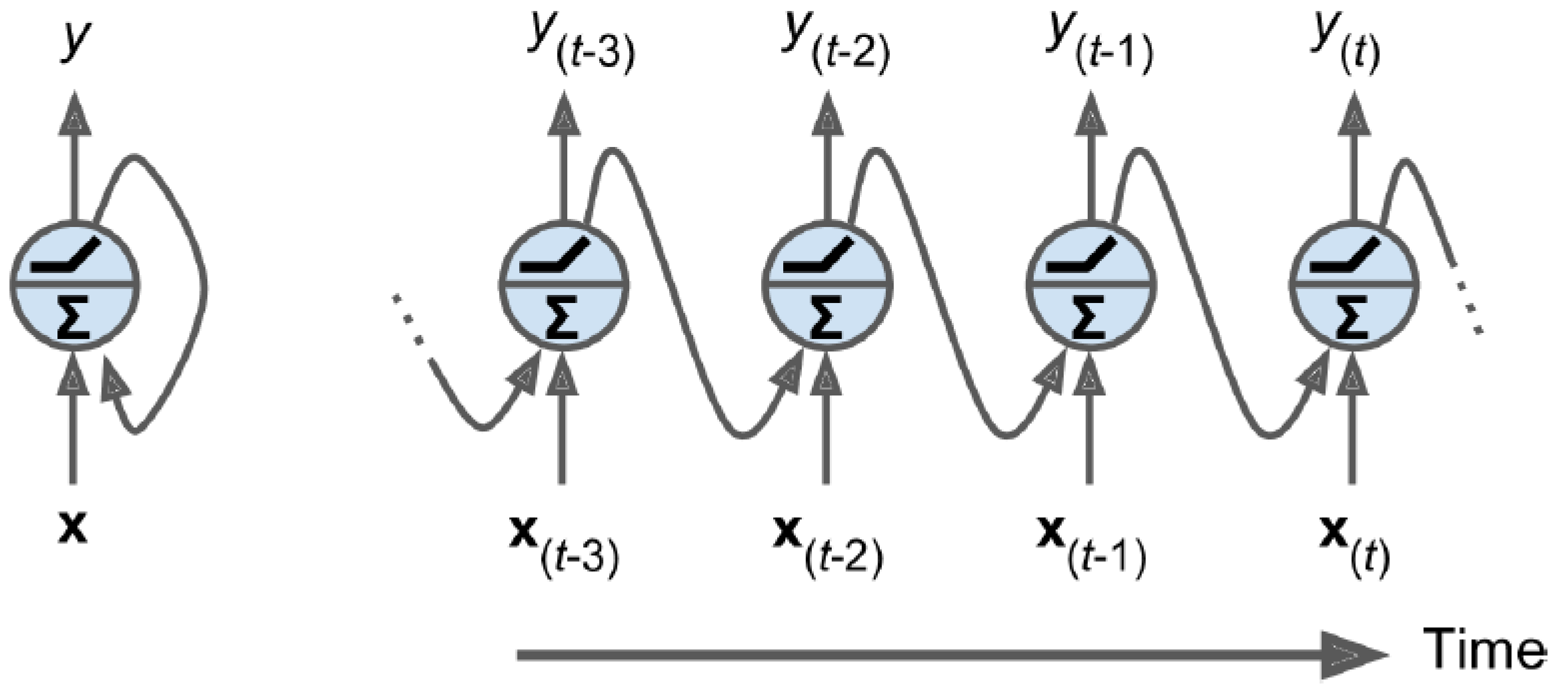
에너지 사용량 예측은 건물 에너지 최적화에 필수적인 기본기능입니다.
예측 모델은 대부분의 운영 최적화, 스케줄링에 필요하므로 다양한 분야에 활용이 가능합니다.



예측 모델 - RNN(Recurrent Neural Network)

순환신경망은 고정 길이 입력이 아닌 임의 길이를 가진 시퀀스를 다룰 수 있습니다. 순환신경망은 시계열 데이터를 분석해서 미래값을 예측하고 문장, 오디오를 입력으로 받아 자동번역, 자연어처리에 유용합니다.

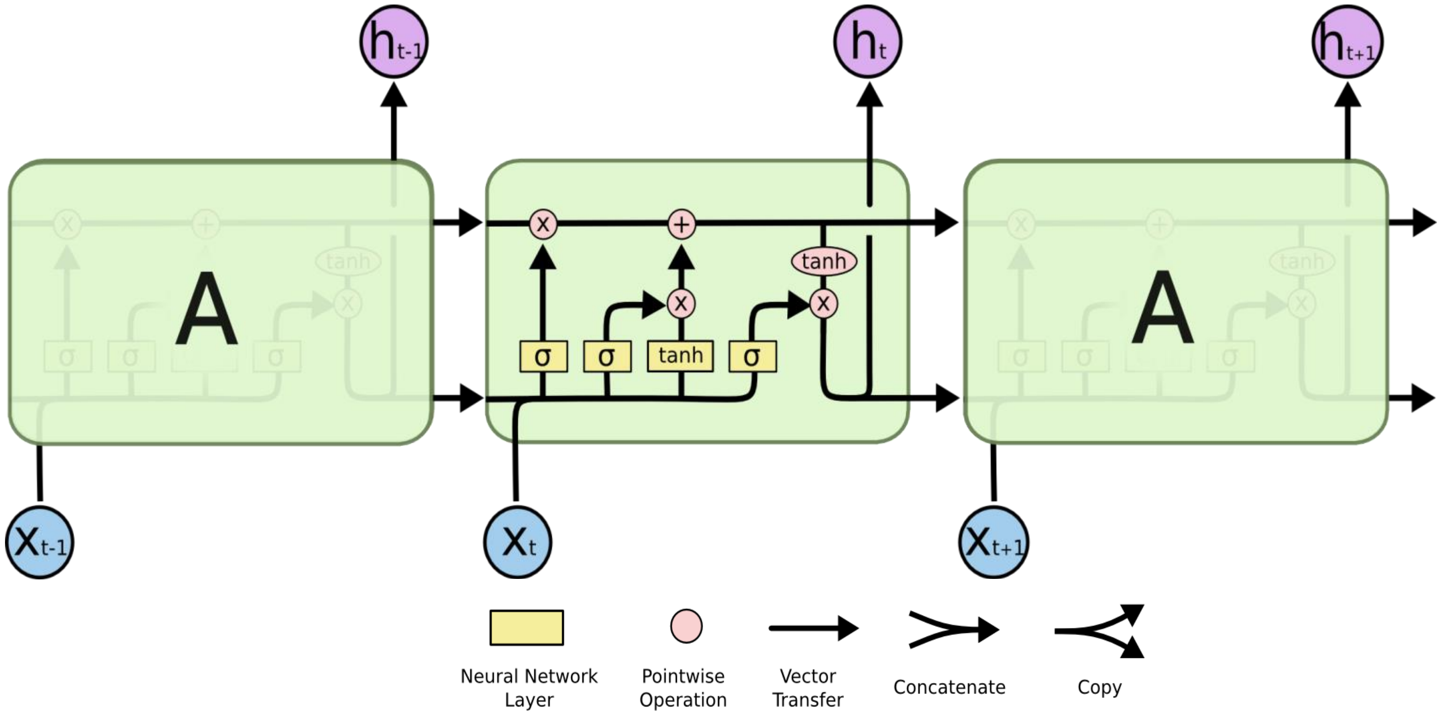
■ 순환뉴런 ■ 순환뉴런을 타임 스텝으로 펼친 모습



출처 : 도서, 핸즈온 머신러닝 2판

예측 모델 - LSTM(Long Short-Term Memory)

LSTM 네트워크는 장기적인 종속성을 학습할 수 있는 특수한 종류의 RNN입니다.
LSTM은 RNN과 동일하게 입력과 출력사이 신경망이 재귀하는 구조를 갖고 있습니다.
그러나 RNN은 재귀를 통한 정보전이 및 전파가 하나의 레이어로 제어되는 반면
LSTM은 Forget gate, Input gate, Output gate를 통한 정보전이 및 전파를 제어합니다.

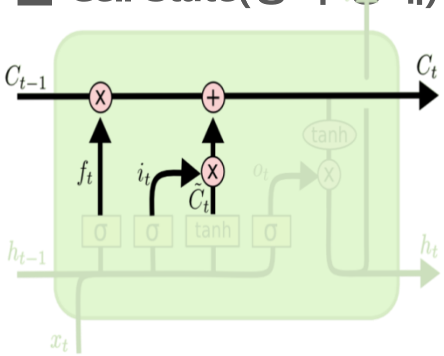


출처 : http://colah.github.io/posts/2015-08-Understanding-LSTMs/

예측 모델 - LSTM(Long Short-Term Memory)

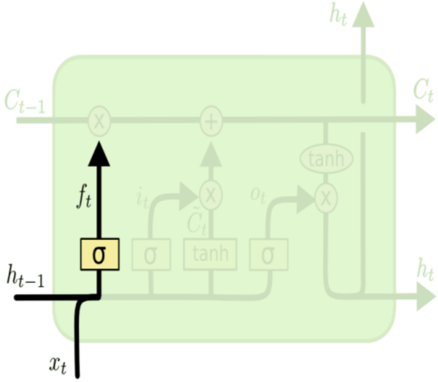
LSTM 구조

Cell State(장기 상태)



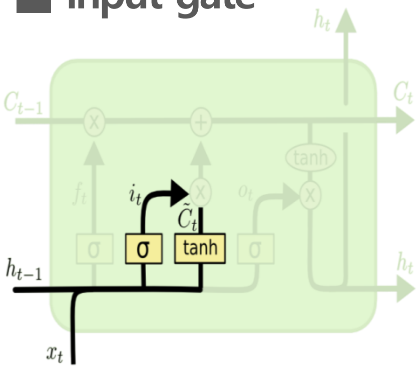
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

forget gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

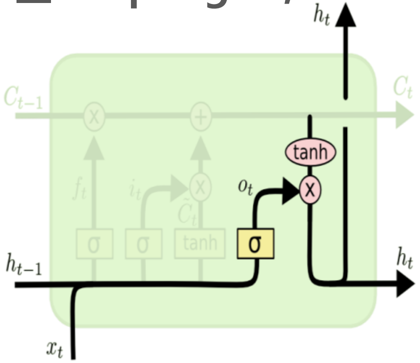
input gate



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

output gate, hidden state(단기 상태)



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

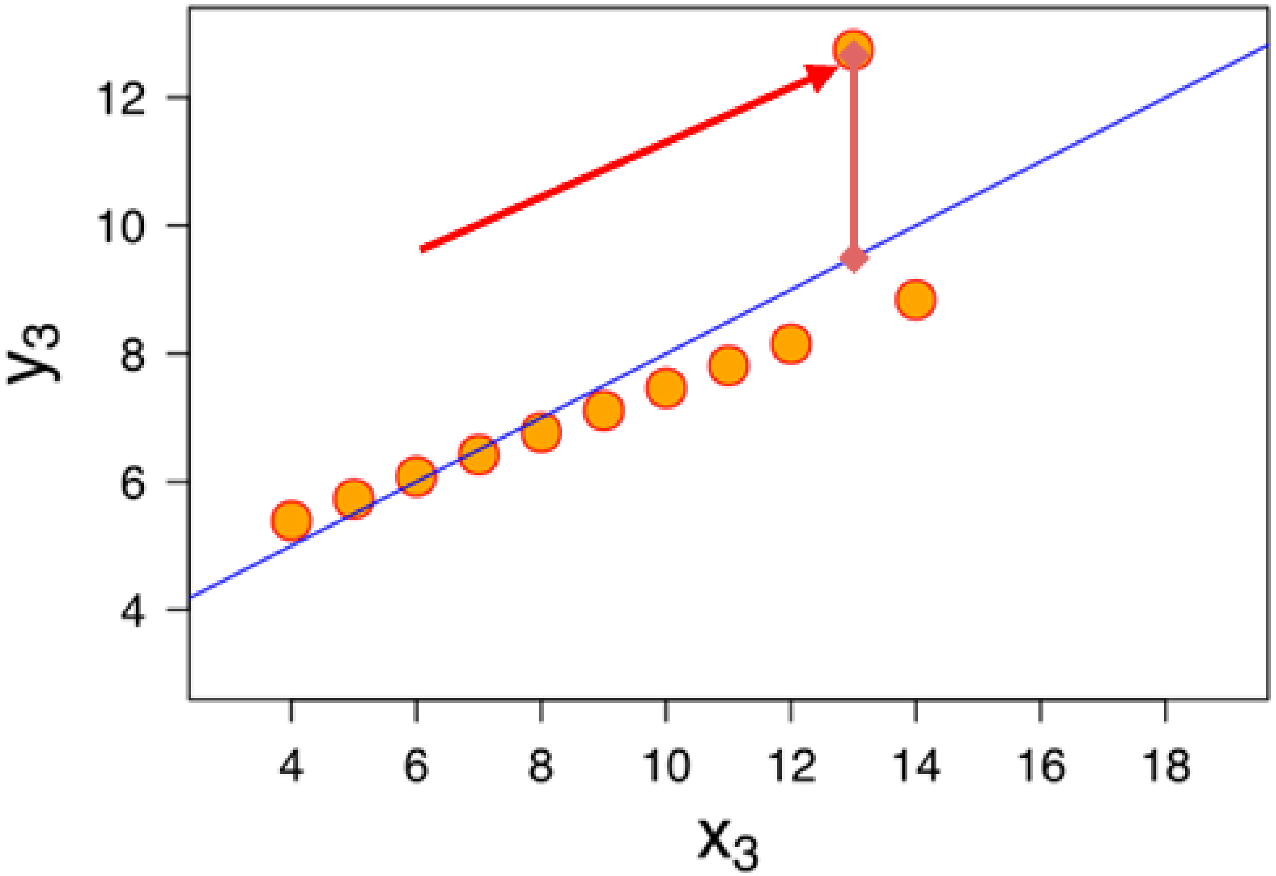
$$h_t = o_t * \tanh\left(C_t\right)$$

출처 : http://colah.github.io/posts/2015-08-Understanding-LSTMs/

예측 모델 - 성능측정

예측 모델의 성능 측정은 MSE를 사용하며, 함수로 학습을 진행하면서 지속적으로 측정을 합니다.

■ MSE(Mean Squared Error)



$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

레이블 값
(실제값)

y_hat
(모델이 예측한 값)

예측모델 개발 방법

1. 데이터분석, 딥러닝 라이브러리 임포트
2. 시계열 데이터 로드
3. 탐색적 데이터 분석
(EDA : Exploratory Data Analysis)
4. 시계열 데이터 정규화
5. 시계열 데이터를 시퀀스 데이터로 변환
6. 딥러닝 모델 구성 및 학습과정 설정
7. 훈련 데이터셋으로 딥러닝 모델 학습 시키기
8. 테스트 데이터셋으로 딥러닝 모델을 평가
9. 예측결과 시각화
10. 딥러닝 모델/가중치 저장

예측모델 사용 방법

1. 딥러닝, 데이터처리 라이브러리 임포트

↓

2. 데이터 로딩 및 포맷변환

↓

3. 학습된 신경망 모델 로딩 및 사용

실습 데이터셋

■ 데이터 파일 : e_usage_train.csv, e_usage_test.csv

- 데이터 설명 : ABC 빌딩의 15분 전기에너지 사용량 데이터
- e_usage_train.csv : 모델 학습(Train) 데이터, 70,000개
- e_usage_test.csv : 모델 성능 테스트(Test) 데이터, 35,040개
- 빌딩의 전기에너지 검침 주기가 15분으로,
- 1시간에는 4개의 데이터, 하루에는 96개(4개x24시) 데이터,
- 1년 기간에는 35,040개(4개x24시x365일)의 데이터가 있습니다.

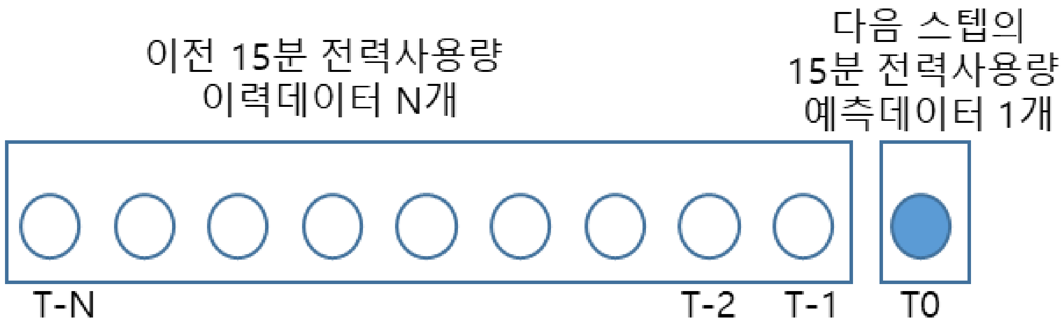
■ 데이터 컬럼명

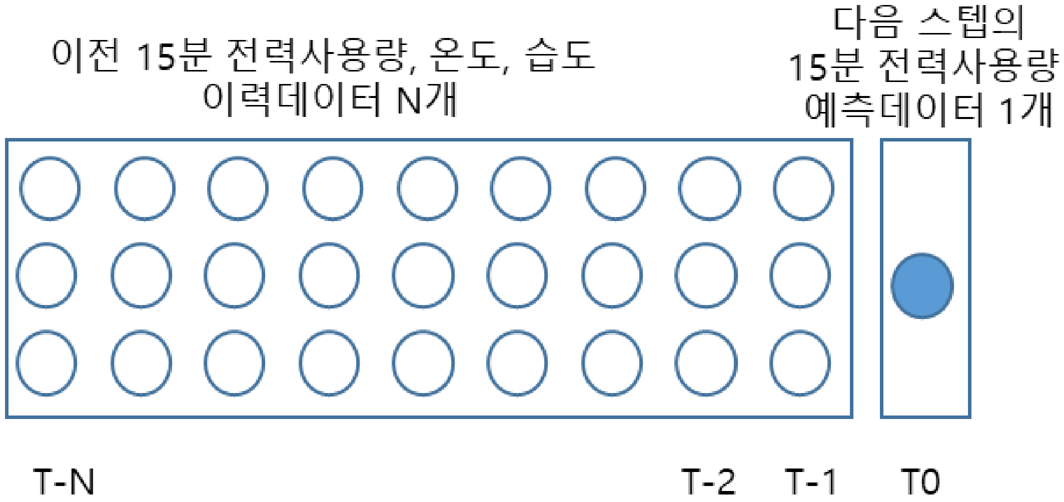
- b_name : 빌딩 이름
- daq_time : 데이터 수집 시간
- wday : 요일 구분
- day_type : 일 구분
  1 - 평일, 2 - 토요일, 3 - 일요일, 휴일
- temp : 온도(°C)
- rh : 상대습도(%)

b_name	daq_time	wday	day_type	hour	temp	rh	p_usage
ABC	2016-01-01 0:15	5	3	1	-2.5	99	229
ABC	2016-01-01 0:30	5	3	1	-2.5	99	231
ABC	2016-01-01 0:45	5	3	1	-2.5	99	231
ABC	2016-01-01 1:00	5	3	1	-3.1	100	226
ABC	2016-01-01 1:15	5	3	2	-3.1	100	229
ABC	2016-01-01 1:30	5	3	2	-3.1	100	223
ABC	2016-01-01 1:45	5	3	2	-3.1	100	233
ABC	2016-01-01 2:00	5	3	2	-3.1	100	234
ABC	2016-01-01 2:15	5	3	3	-3.1	100	230
ABC	2016-01-01 2:30	5	3	3	-3.1	100	228
ABC	2016-01-01 2:45	5	3	3	-3.1	100	224
ABC	2016-01-01 3:00	5	3	3	-2.9	100	226
ABC	2016-01-01 3:15	5	3	4	-2.9	100	234

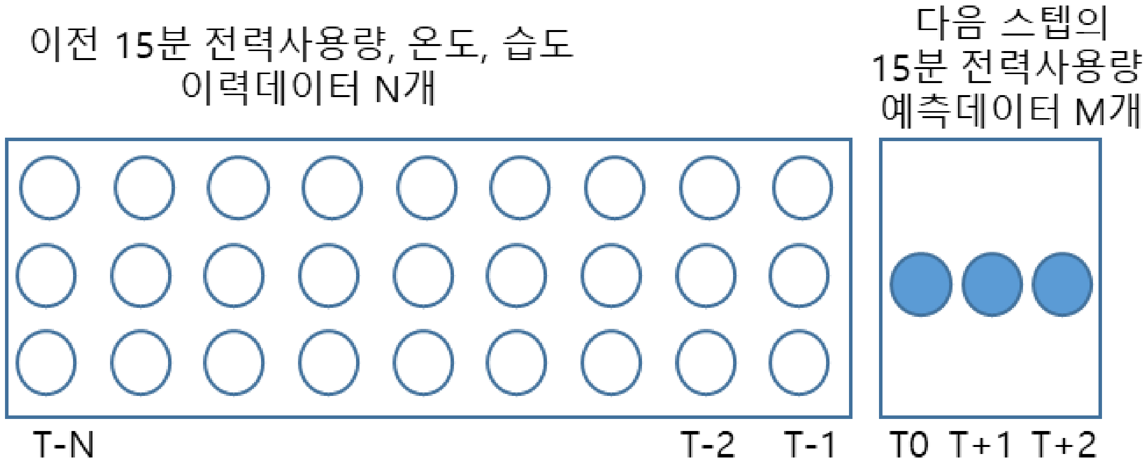
시퀀스 데이터 구성방법

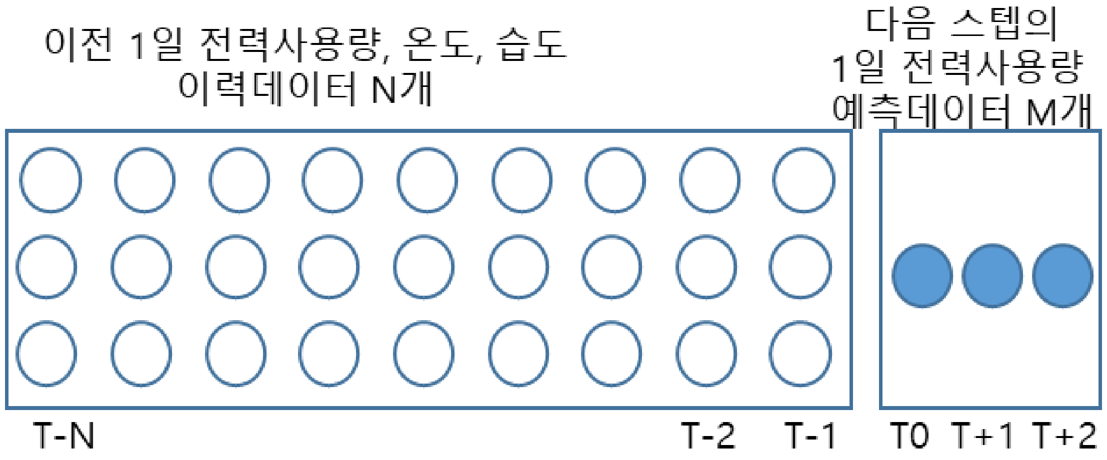
■ 싱글스텝





■ 멀티스텝

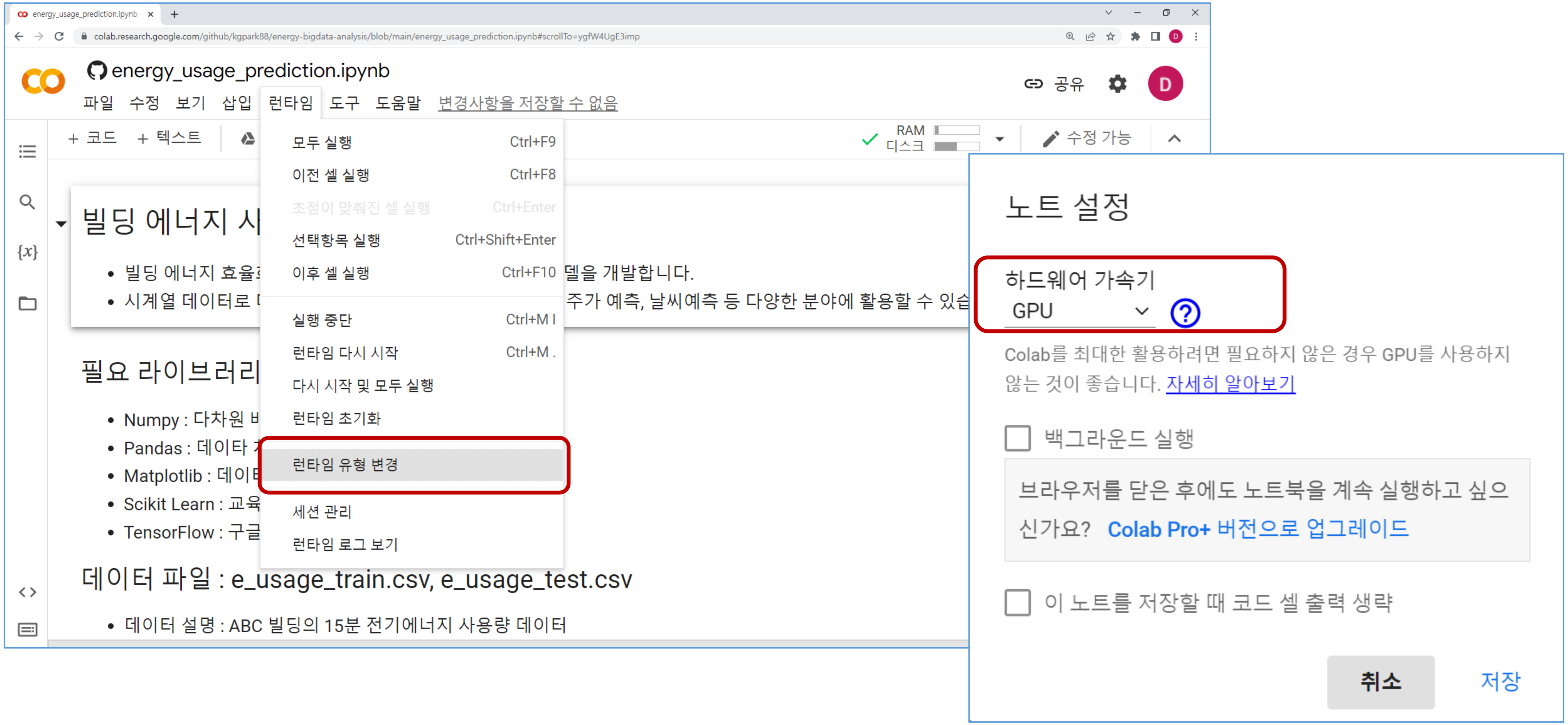




에너지 사용량 예측모델 개발

energy_usage_prediction.ipynb

Open in Colab



에너지 사용량 예측모델 개발

STEP 1. 라이브러리 import

```
[1]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.preprocessing import MinMaxScaler

     import tensorflow as tf
     from tensorflow.keras.models import Sequential, load_model
     from tensorflow.keras.layers import Dense, LSTM, Activation, Dropout
     from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

에너지 사용량 예측모델 개발

STEP 2. 시계열 데이터 처리

csv 파일에서 Train 데이터를 로드합니다.

```
[2]  df = pd.read_csv('e_usage_train.csv', header = 0, delimiter = ',')
```

데이터를 확인합니다.

```
[3]  df.head()
```

	b_name	daq_time	wday	day_type	hour	temp	rh	p_usage
0	ABC	2016-01-01 0:15	5	3	1	-2.5	99.0	229
1	ABC	2016-01-01 0:30	5	3	1	-2.5	99.0	231
2	ABC	2016-01-01 0:45	5	3	1	-2.5	99.0	231

에너지 사용량 예측모델 개발

데이터셋을 입력시퀀스데이터와 타깃데이터로 분리하는 함수입니다.

- 시계열 데이터를 시퀀스 데이터로 변환
- 입력데이터는 시퀀스이고, 출력은 고정크기의 벡터나 스칼라인 다대일(many-to-one) 구조로 데이터 변환

```
[7] def split_multivariate_data(dataset, target, start_index, end_index, hist_data_size, target_size, step, single_step=False):
        data = []
        labels = []

        start_index = start_index + hist_data_size
        if end_index is None:
            end_index = len(dataset) - target_size

        for i in range(start_index, end_index):
            indices = range(i-hist_data_size, i, step)
            data.append(dataset[indices])

            if single_step:
                labels.append(target[i+target_size])
            else:
                labels.append(target[i:i+target_size])

        return np.array(data), np.array(labels)
```

에너지 사용량 예측모델 개발

입력시퀀스데이터, 타깃데이터, 예측데이터를 그래프에 출력하는 함수입니다.

```
[8]  def plot_series(series, y=None, y_pred=None, x_label="$t$", y_label="$x(t)$"):
         n_steps = len(series)
         plt.plot(series, ".-")
         if y is not None:
             plt.plot(n_steps, y, "bx", markersize=10)
         if y_pred is not None:
             plt.plot(n_steps, y_pred, "ro")
         plt.grid(True)
         if x_label:
             plt.xlabel(x_label, fontsize=16, rotation=90)
         if y_label:
             plt.ylabel(y_label, fontsize=16, rotation=0)
```

에너지 사용량 예측모델 개발

전력사용량, 온도, 상대습도를 입력데이터(Feature)로 사용합니다.

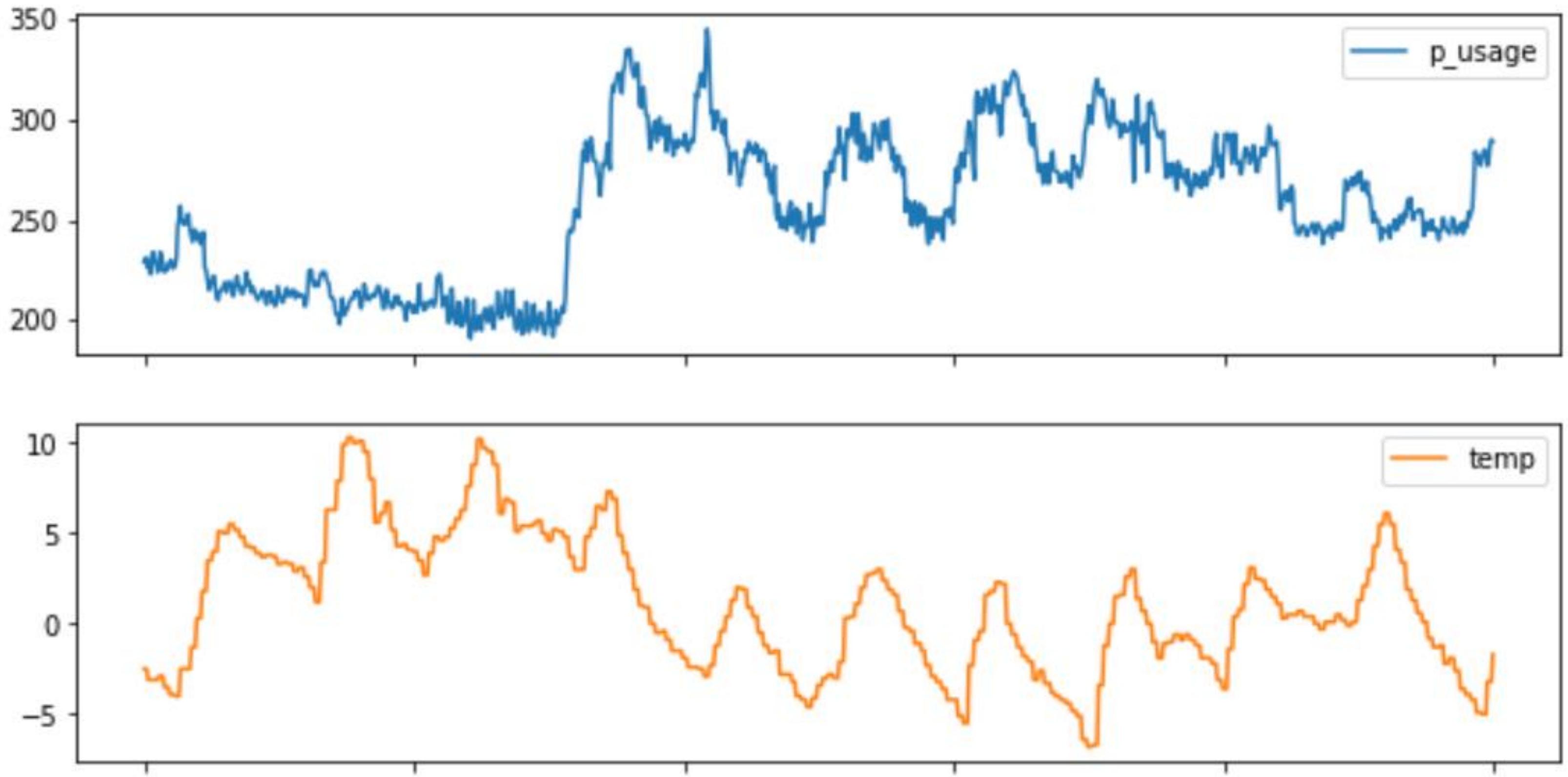
```
[9]  features = ['p_usage', 'temp', 'rh']
     features_data = df[features]
     features_data.index = df['daq_time']
     features_data.head()
```

	p_usage	temp	rh
daq_time			
2016-01-01 0:15	229	-2.5	99.0
2016-01-01 0:30	231	-2.5	99.0
2016-01-01 0:45	231	-2.5	99.0

에너지 사용량 예측모델 개발

시계열 데이터의 패턴을 확인합니다.

```
[10] features_data[0:1000].plot(subplots=True, figsize=(10, 8))
```



에너지 사용량 예측모델 개발

데이터셋의 피처(Feature)를 정규화(Scaling)합니다.

```
[12]  TRAIN_SPLIT = 60000
      HISTORY_DATA_SIZE = 20
      FUTURE_TARGET = 0
      STEP = 1
```

```
[13]  scaler = MinMaxScaler()
      dataset  = scaler.fit_transform(dataset)
```

```
[14]  dataset
```

```
array([[0.43126177, 0.26245211, 0.98876404],
       [0.43502825, 0.26245211, 0.98876404],
       [0.43502825, 0.26245211, 0.98876404],
       ...,
```

에너지 사용량 예측모델 개발

Train 데이터셋과 Validation 데이터셋을 만듭니다.

```
[15]  X = dataset
      y = dataset[:,0]

      X_train, y_train = split_multivariate_data(X, y,
                                                  0, TRAIN_SPLIT,
                                                  HISTORY_DATA_SIZE, FUTURE_TARGET,
                                                  STEP, True)
      X_valid, y_valid = split_multivariate_data(X, y,
                                                  TRAIN_SPLIT, None,
                                                  HISTORY_DATA_SIZE, FUTURE_TARGET,
                                                  STEP, True)
```

에너지 사용량 예측모델 개발

split_multivariate_data 함수가 반환하는 내용입니다.

```
[16] print ('Single window of past history : {}'.format(X_train[0].shape))
```

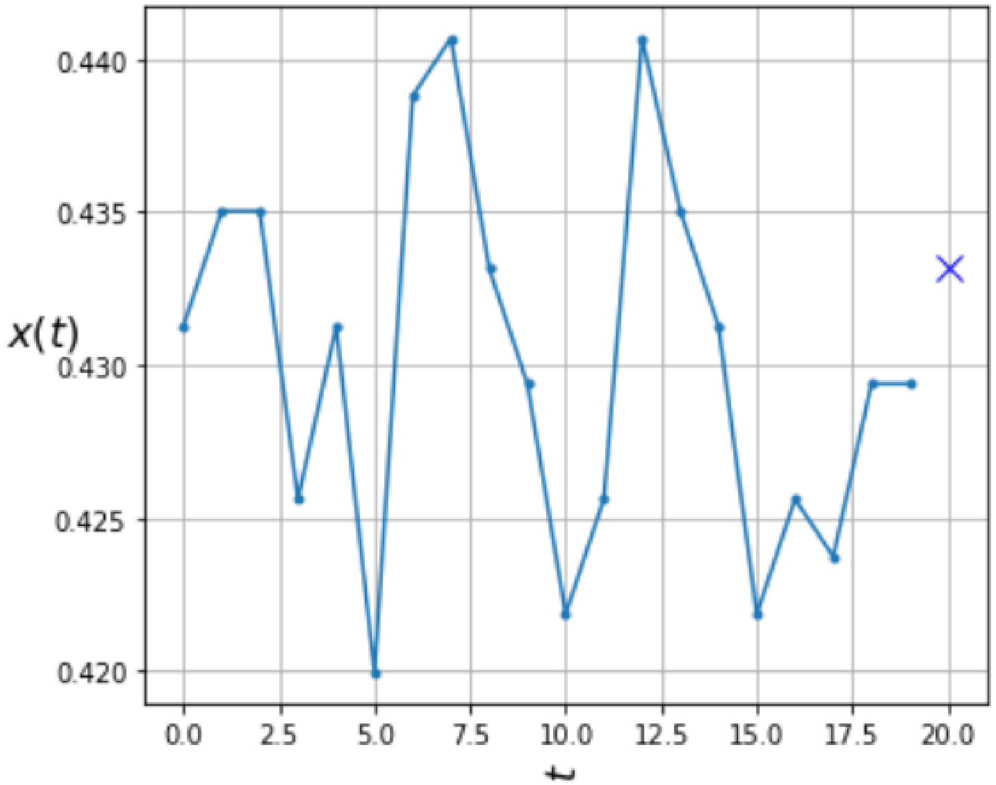
```
Single window of past history : (20, 3)
```

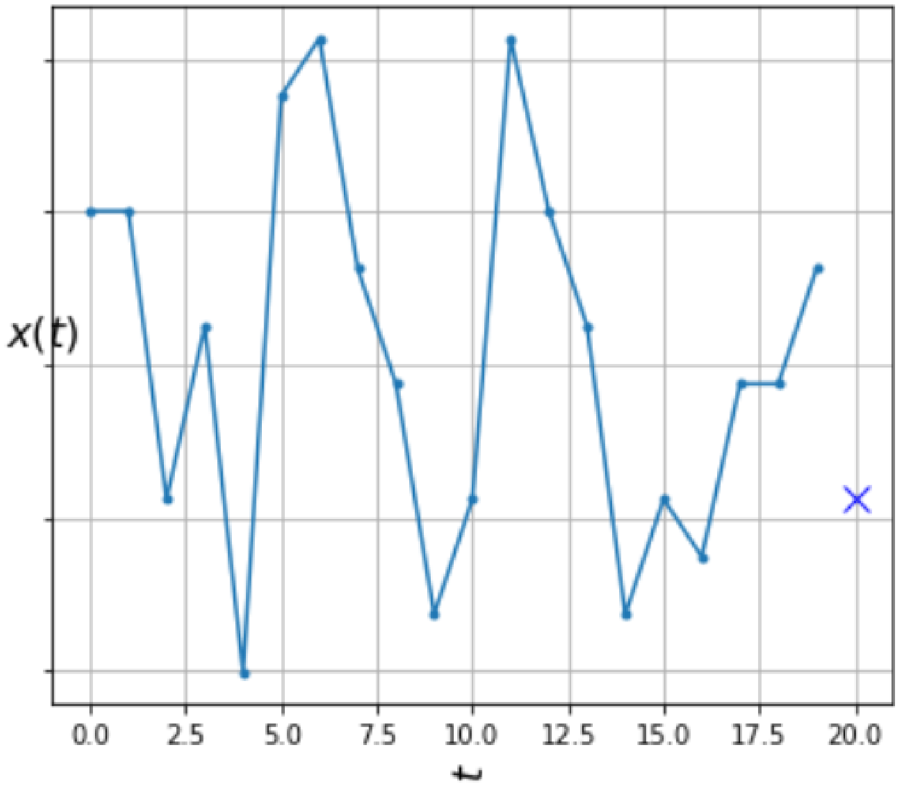
```
[17] print ('입력 데이터')
     print (X_train[0])
     print ('타겟 데이터')
     print (y_train[0])
```

```
입력 데이터
[[0.43126177 0.26245211 0.98876404]
 [0.43502825 0.26245211 0.98876404]
 [0.43502825 0.26245211 0.98876404]
 [0.42561205 0.25095785 1.        ]
 [0.43126177 0.25095785 1.        ]
 [0.41996234 0.25095785 1.        ]
 [0.43879473 0.25095785 1.        ]
 [0.44067797 0.25095785 1.        ]
 [0.43314501 0.25095785 1.        ]
 [0.42937853 0.25095785 1.        ]
 [0.42184557 0.25095785 1.        ]
 [0.42561205 0.25478927 1.        ]
 [0.44067797 0.25478927 1.        ]
 [0.43502825 0.25478927 1.        ]
 [0.43126177 0.25478927 1.        ]
 [0.42184557 0.24329502 1.        ]
 [0.42561205 0.24329502 1.        ]
 [0.42372881 0.24329502 1.        ]
 [0.42937853 0.24329502 1.        ]
 [0.42937853 0.23563218 1.        ]]
타겟 데이터
0.4331450094161958
```

에너지 사용량 예측모델 개발

```
[18] cols = 3
     fig, axes = plt.subplots(nrows=1, ncols=cols, sharey=True, figsize=(20, 5))
     for i in range(cols):
         plt.sca(axes[i])
         plot_series(X_train[i, :, 0], y_train[i])
     plt.show()
```





에너지 사용량 예측모델 개발

STEP 3. 딥러닝 모델 구현

데이터가 순차데이터(Sequence Data)인 시계열(Time Series) 이므로
다양한 길이의 순차데이터 처리에 적합한 RNN 기반의 LSTM 모델을 사용합니다.

```
[19] HIDDEN_SIZE = 10
     DROP_OUT = 0.3

     model = Sequential()
     model.add(LSTM(HIDDEN_SIZE, input_shape=[20, 3], return_sequences=False))
     model.add(Dropout(DROP_OUT))
     model.add(Dense(1))
```

에너지 사용량 예측모델 개발

모델 구성 확인

```
[20]  model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10)                560

 dropout (Dropout)           (None, 10)                0

 dense (Dense)               (None, 1)                 11

=================================================================
Total params: 571
Trainable params: 571
Non-trainable params: 0
_________________________________________________________________
```

에너지 사용량 예측모델 개발

모델 컴파일

```
[22]  model.compile(optimizer='adam', loss='mse')
```

모델 학습(Train) 조기종료, 체크포인트 설정

```
[23]  early_stop = EarlyStopping(monitor='val_loss', mode='min',
                                verbose=1, patience=5)
      check_point = ModelCheckpoint('best_model.h5', verbose=1,
                            monitor='val_loss', mode='min', save_best_only=True)
```

에너지 사용량 예측모델 개발

모델 학습(Train)

전력사용량 데이터는 일(Day) 단위 패턴이 있으므로 BATCH_SIZE를 96(15분*24시간=96)

```
[24] EPOCHS = 50
     BATCH_SIZE=96

     history = model.fit(x=X_train, y=y_train,
                         epochs=EPOCHS,
                         batch_size=BATCH_SIZE,
                         verbose=1,
                         validation_data=(X_valid, y_valid),
                         callbacks=[early_stop, check_point])
```

에너지 사용량 예측모델 개발

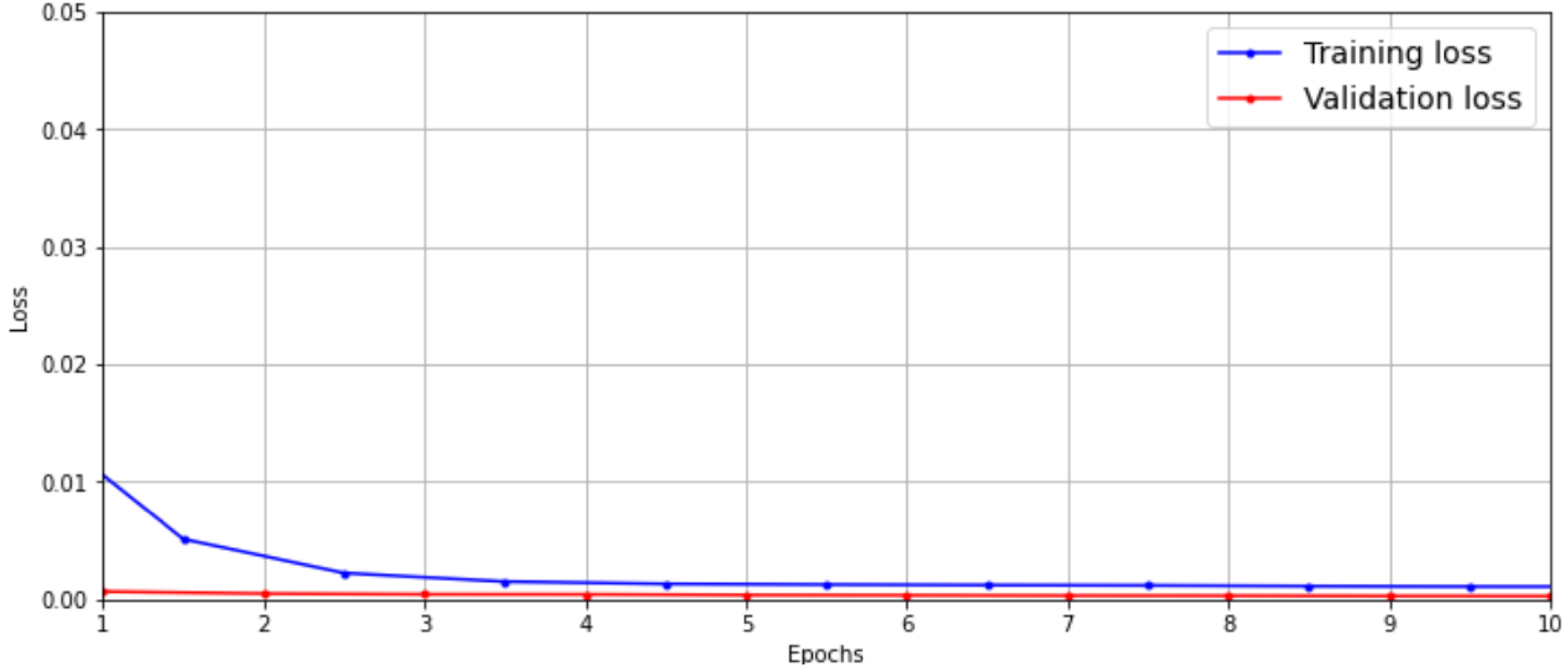
```
Epoch 1/50
616/625 [============================>.] - ETA: 0s - loss: 0.0161
Epoch 1: val_loss improved from inf to 0.00067, saving model to best_model.h5
625/625 [==============================] - 6s 5ms/step - loss: 0.0160 - val_loss: 6.6692e-04
Epoch 2/50
618/625 [============================>.] - ETA: 0s - loss: 0.0051
Epoch 2: val_loss improved from 0.00067 to 0.00049, saving model to best_model.h5
625/625 [==============================] - 3s 4ms/step - loss: 0.0051 - val_loss: 4.8703e-04
Epoch 3/50
612/625 [============================>.] - ETA: 0s - loss: 0.0022
Epoch 3: val_loss improved from 0.00049 to 0.00043, saving model to best_model.h5
625/625 [==============================] - 3s 4ms/step - loss: 0.0022 - val_loss: 4.2789e-04
Epoch 4/50
617/625 [============================>.] - ETA: 0s - loss: 0.0015
Epoch 4: val_loss improved from 0.00043 to 0.00042, saving model to best_model.h5
625/625 [==============================] - 3s 4ms/step - loss: 0.0015 - val_loss: 4.2062e-04
Epoch 5/50
616/625 [============================>.] - ETA: 0s - loss: 0.0013
Epoch 5: val_loss improved from 0.00042 to 0.00037, saving model to best_model.h5
```

에너지 사용량 예측모델 개발

모델의 Training Loss와 Validation Loss를 출력하는 함수입니다.

```
[25] def plot_learning_curves(loss, val_loss):
         plt.figure(figsize=(12, 5))
         plt.plot(np.arange(len(loss)) + 0.5, loss, "b.-", label="Training loss")
         plt.plot(np.arange(len(val_loss)) + 1, val_loss, "r.-", label="Validation loss")
         plt.axis([1, 10, 0, 0.05])
         plt.legend(fontsize=14)
         plt.xlabel("Epochs")
         plt.ylabel("Loss")
         plt.grid(True)

     plot_learning_curves(history.history["loss"], history.history["val_loss"])
     plt.show()
```

에너지 사용량 예측모델 개발



모델 저장

```
[26] model.save('my_model.h5')
```

에너지 사용량 예측모델 사용

STEP 4. 딥러닝 모델 사용

csv 파일에서 Test 데이터를 로드합니다.

```
[27]  df = pd.read_csv('e_usage_test.csv', header = 0, delimiter = ',')
```

데이터를 확인합니다.

```
[28]  df.head()
```

	b_name	daq_time	wday	day_type	hour	temp	rh	p_usage
0	ABC	2018-01-01 0:15	1	3	1	-1.8	43	283
1	ABC	2018-01-01 0:30	1	3	1	-1.8	43	279

에너지 사용량 예측모델 사용

전력사용량, 온도, 상대습도를 입력데이터(Feature)로 사용합니다.

```
[31] features = ['p_usage', 'temp', 'rh']
     features_data = df[features]
     features_data.index = df['daq_time']
     dataset = features_data.values
```

데이터를 정규화(Scaling) 합니다.

```
[32] scaled_dataset = scaler.transform(dataset)
```

```
[33] X = scaled_dataset
     y = scaled_dataset[:,0]

     X_test, y_test = split_multivariate_data(X, y,
                                              0, None,
                                              HISTORY_DATA_SIZE, FUTURE_TARGET,
                                              STEP, True)
```

에너지 사용량 예측모델 사용

저장한 모델을 로드합니다.

```
[34]  model = load_model('best_model.h5')
```

테스트 데이터셋으로 모델 성능을 평가합니다.

```
[35]  model.evaluate(X_test, y_test)
```

```
1095/1095 [==============================] - 3s 2ms/step - loss: 5.8708e-04
0.0005870836903341115
```

에너지 사용량 예측모델 사용

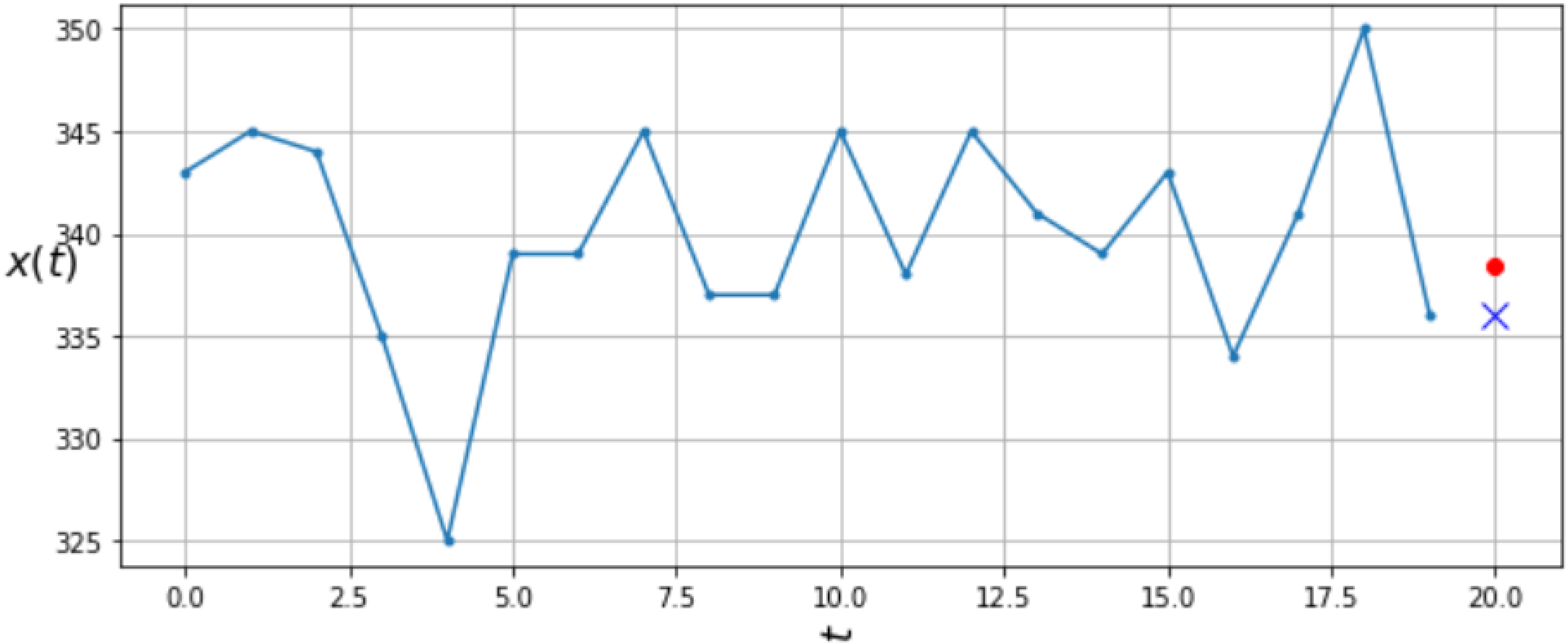
AI모델로 에너지 사용량을 예측합니다.

```
[36] for TIME_STEP in range(1000,1110):
         p_usage_hist = scaler.inverse_transform(X_test[TIME_STEP])
         p_usage_hist = p_usage_hist[:, 0]

         p_usage_real  = dataset[TIME_STEP + HISTORY_DATA_SIZE][0]

         pred = model.predict(
             X_test[TIME_STEP].reshape(1, HISTORY_DATA_SIZE, -1))
         pred = pred[0][0]
         p_usage_pred = scaler.inverse_transform([[pred, 0, 0]])[0][0]

         error = abs(p_usage_pred - p_usage_real)
         error_rate = error/p_usage_real*100
```

에너지 사용량 예측모델 사용

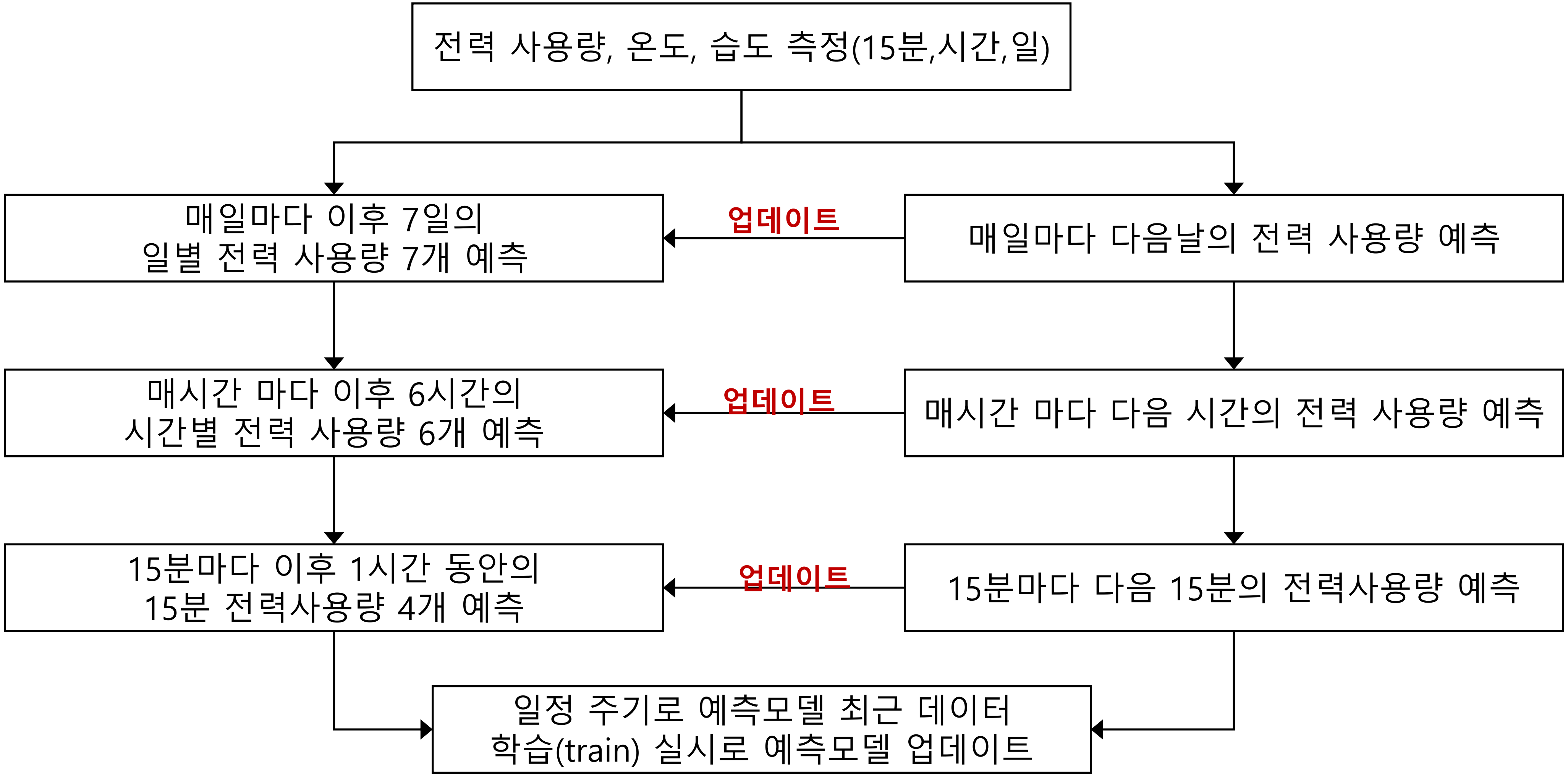
```
fig, axes = plt.subplots(nrows=1, ncols=1,
                         sharey=True, figsize=(10, 4))
plot_series(p_usage_hist, p_usage_real, p_usage_pred)
plt.show()

print(f'입력데이터 : {p_usage_hist}')
print(f'실제값 : {p_usage_real:.2f}')
print(f'예측값 : {p_usage_pred:.2f}')
print(f'오차 : {error:.2f}')
print(f'오차율 : {error_rate:.2f}%')
```

에너지 사용량 예측모델 사용



입력데이터 : [343. 345. 344. 335. 325. 339. 339. 345. 337. 337. 345. 338. 345. 341.
339. 343. 334. 341. 350. 336.]
실제값 : 336.00
예측값 : 338.46
오차 : 2.46
오차율 : 0.73%

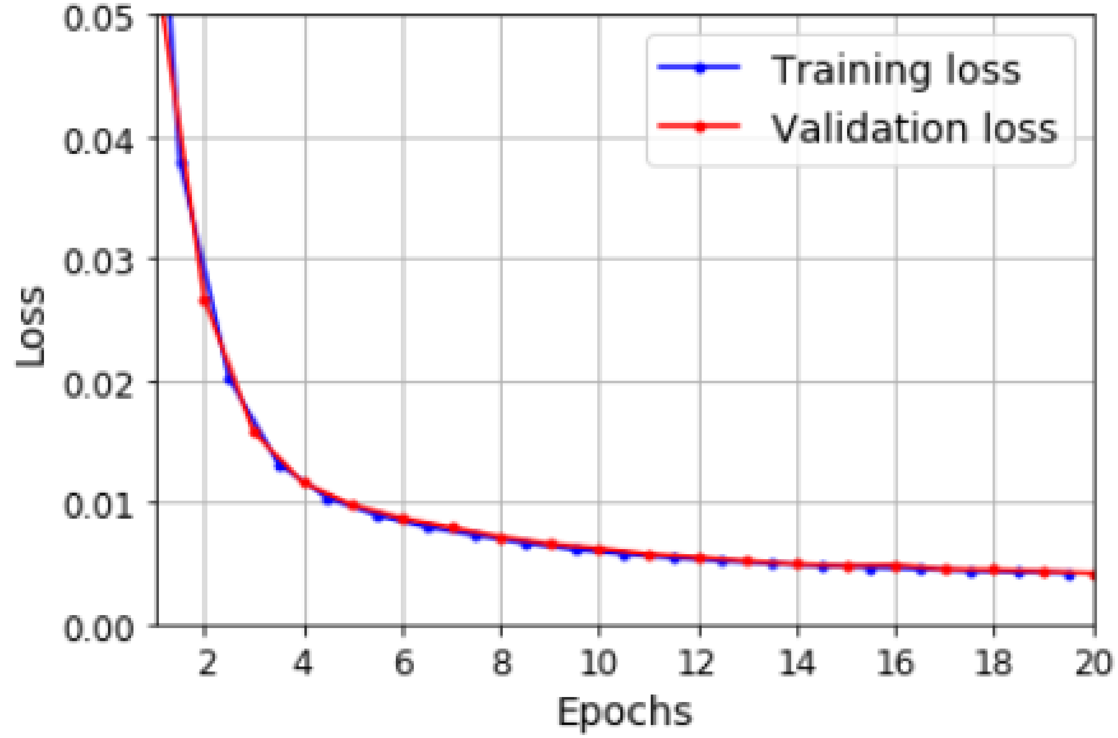
에너지 사용량 예측모델 사용 전략

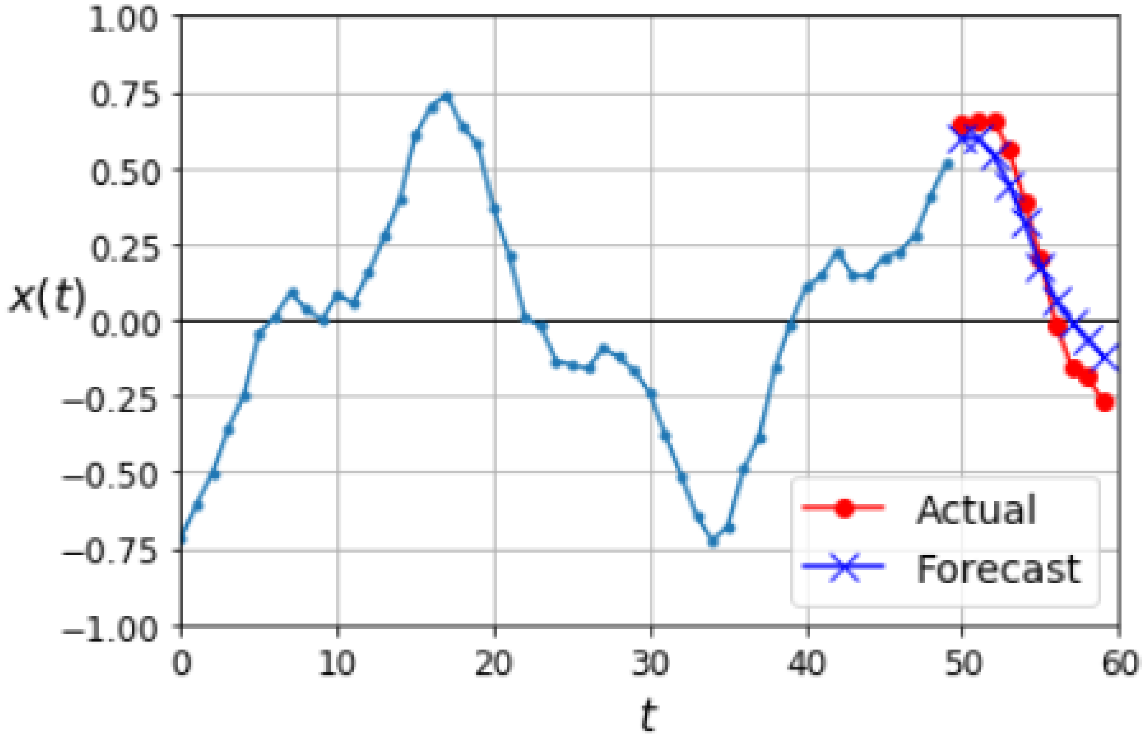


AI 예측모델 - Further Study

https://github.com/rickiepark/handson-ml2/blob/master/15_processing_sequences_using_rnns_and_cnns.ipynb

Run in Google Colab





Thank you

kgpark88@gmail.com