ASHRAE -
Great Energy Predictor III

숙명여자대학교 컴퓨터과학전공 19학번 송민영

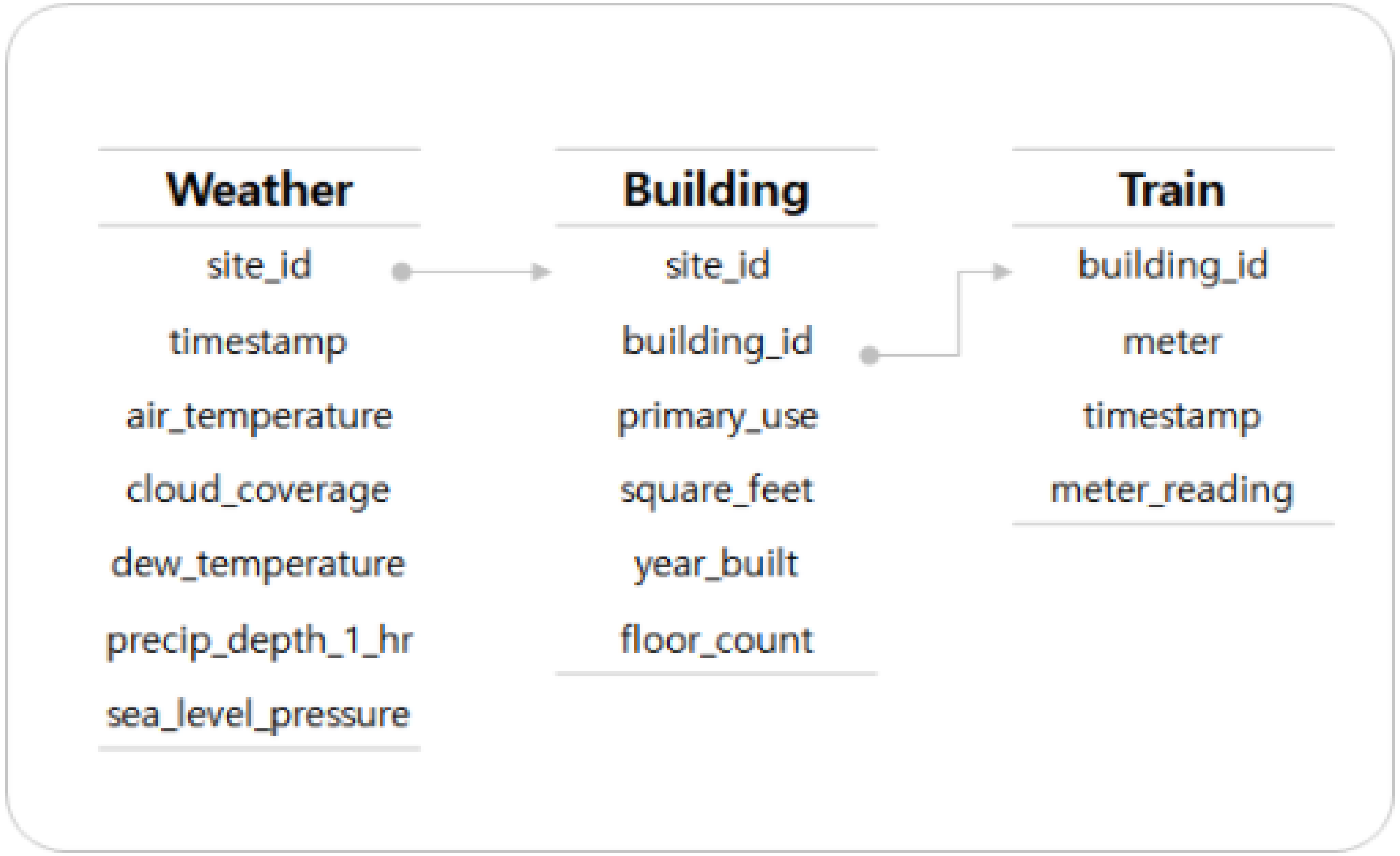
Introduction

2016년 데이터를 모델링하여 17년 1월 ~ 18년 6월까지의 에너지 사용량을 예측하는 대회
세계 각국의 100개가 넘는 건물에서 생성된 3년간의 Electricity/Chilledwater/Steam/Hotwater 영역에서의 사용량을 기반으로 모델링을 하는 대회

※ ASHRAE : 미국 냉난방 공조 협회 (American Society of Heating, Refrigerating and Air-Conditioning Engineers)는 난방, 환기, 냉방 및 냉장 시스템 설계 및 시공을 향상시키기 위해 노력하는 미국 전문가 협회

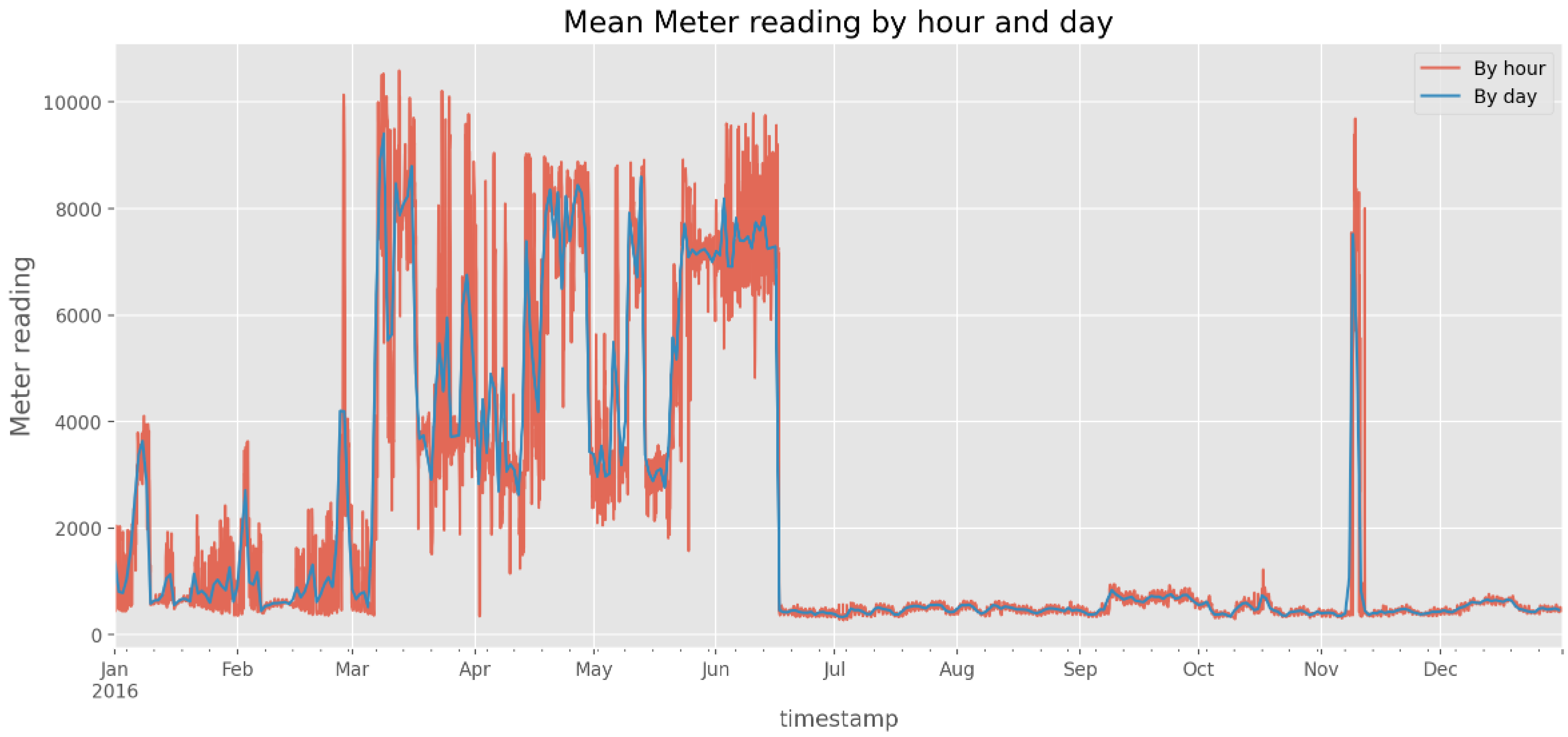
데이터 파악하기

Target 컬럼은 meter_reading이며 meter 컬럼에 의해 Electirity/Chilledwater/Steam/Hotwater로 구분된다.



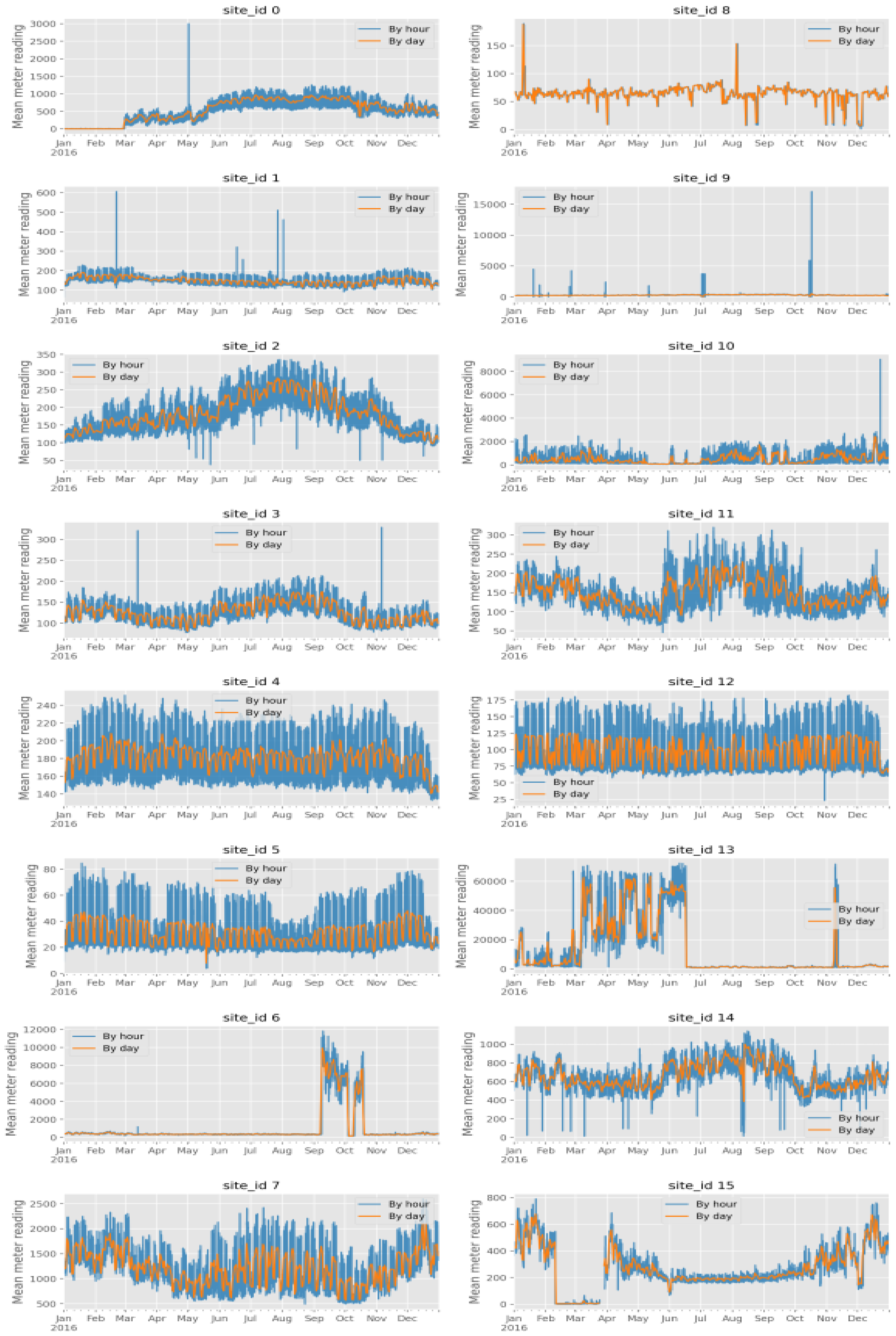
데이터 파악하기

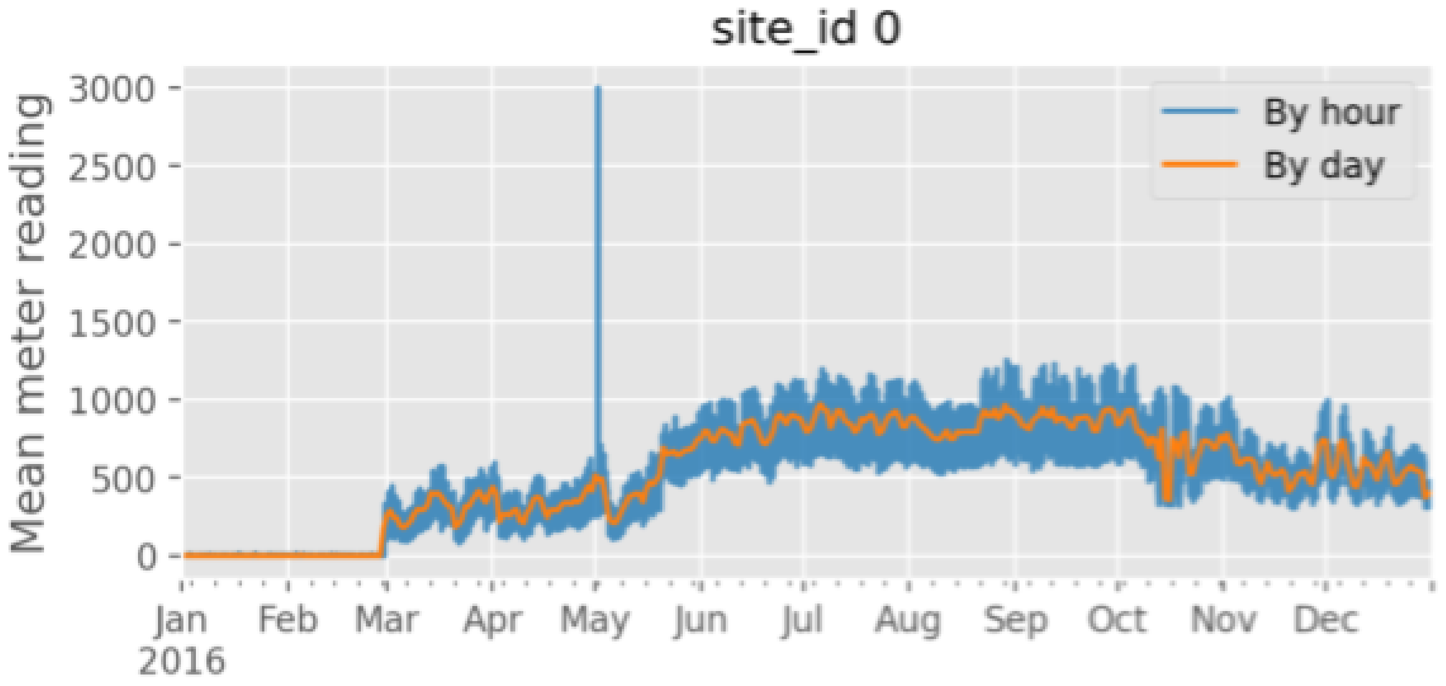
train dataset에 weather_train과 building_meta를 merge 후 plot그림

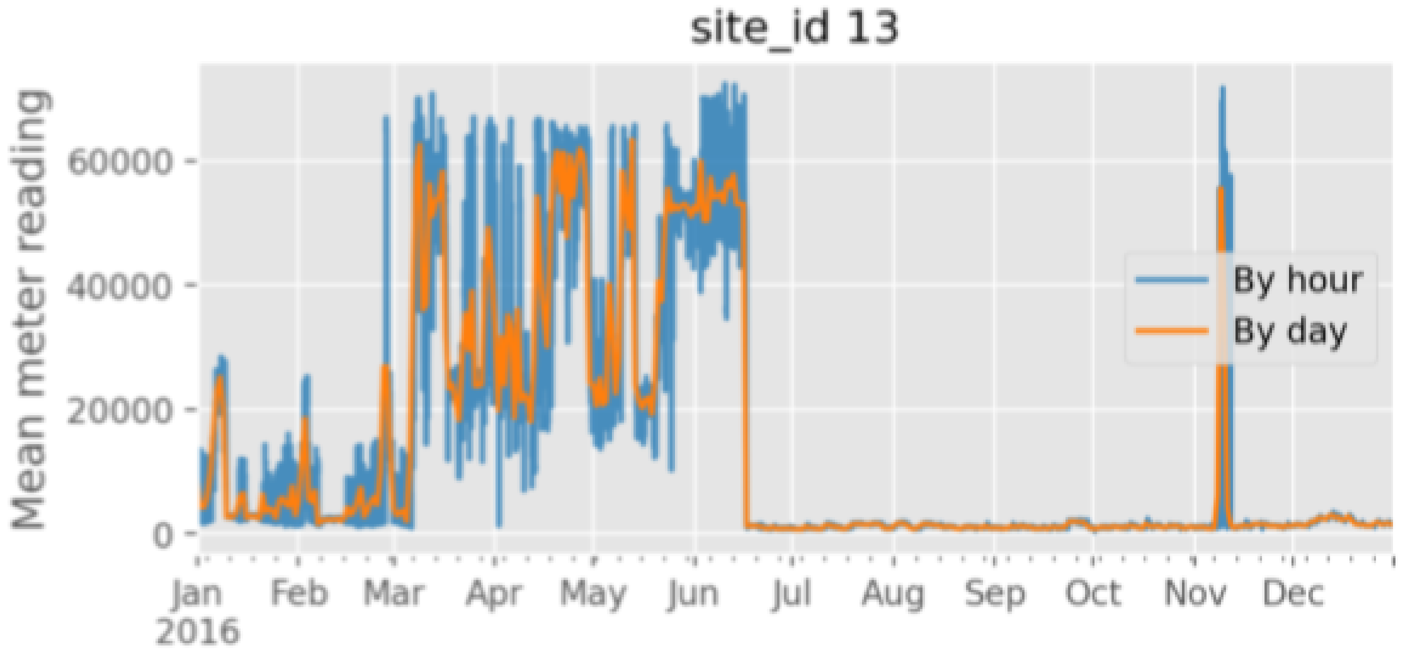


데이터 파악하기

site_id에 따라 plot 그리기



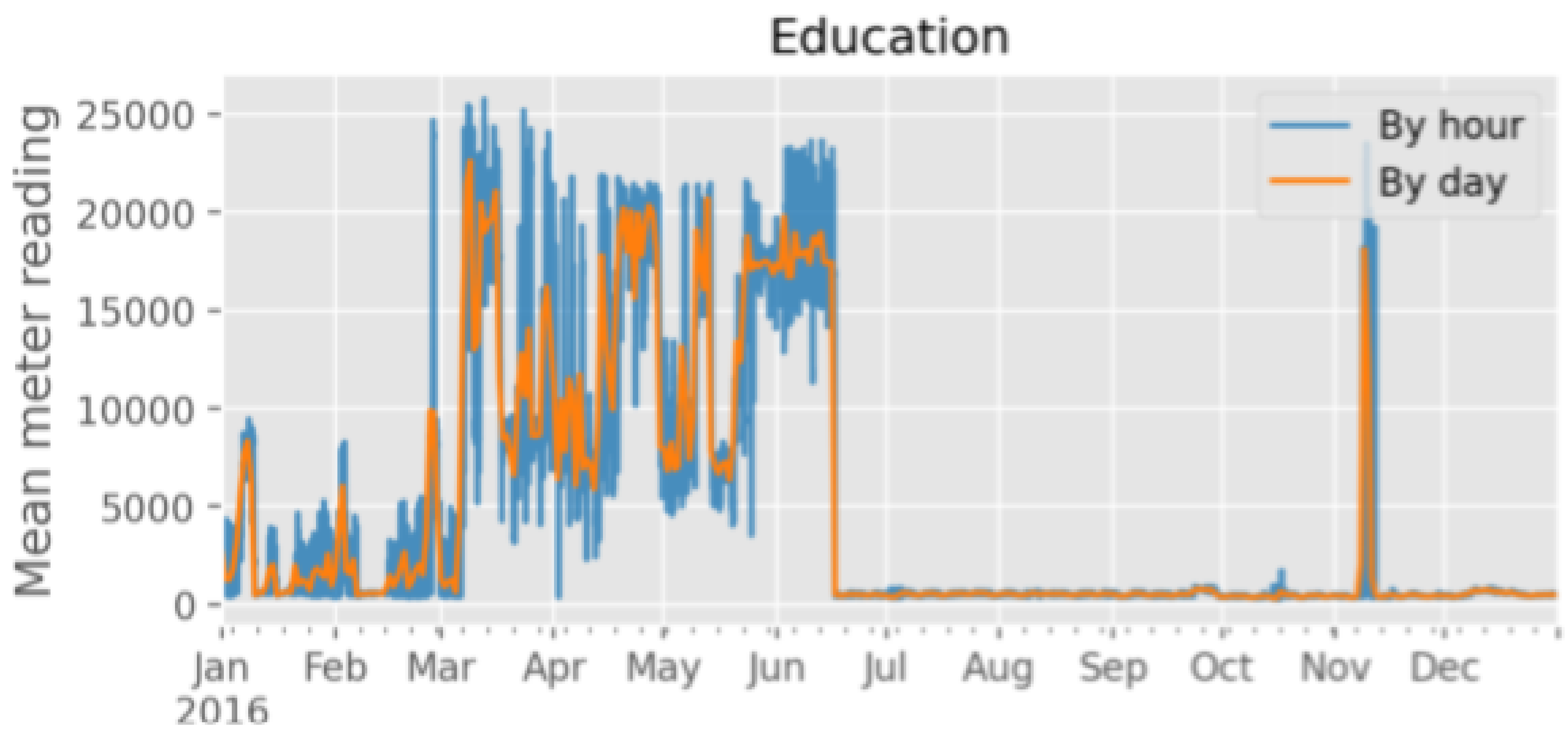


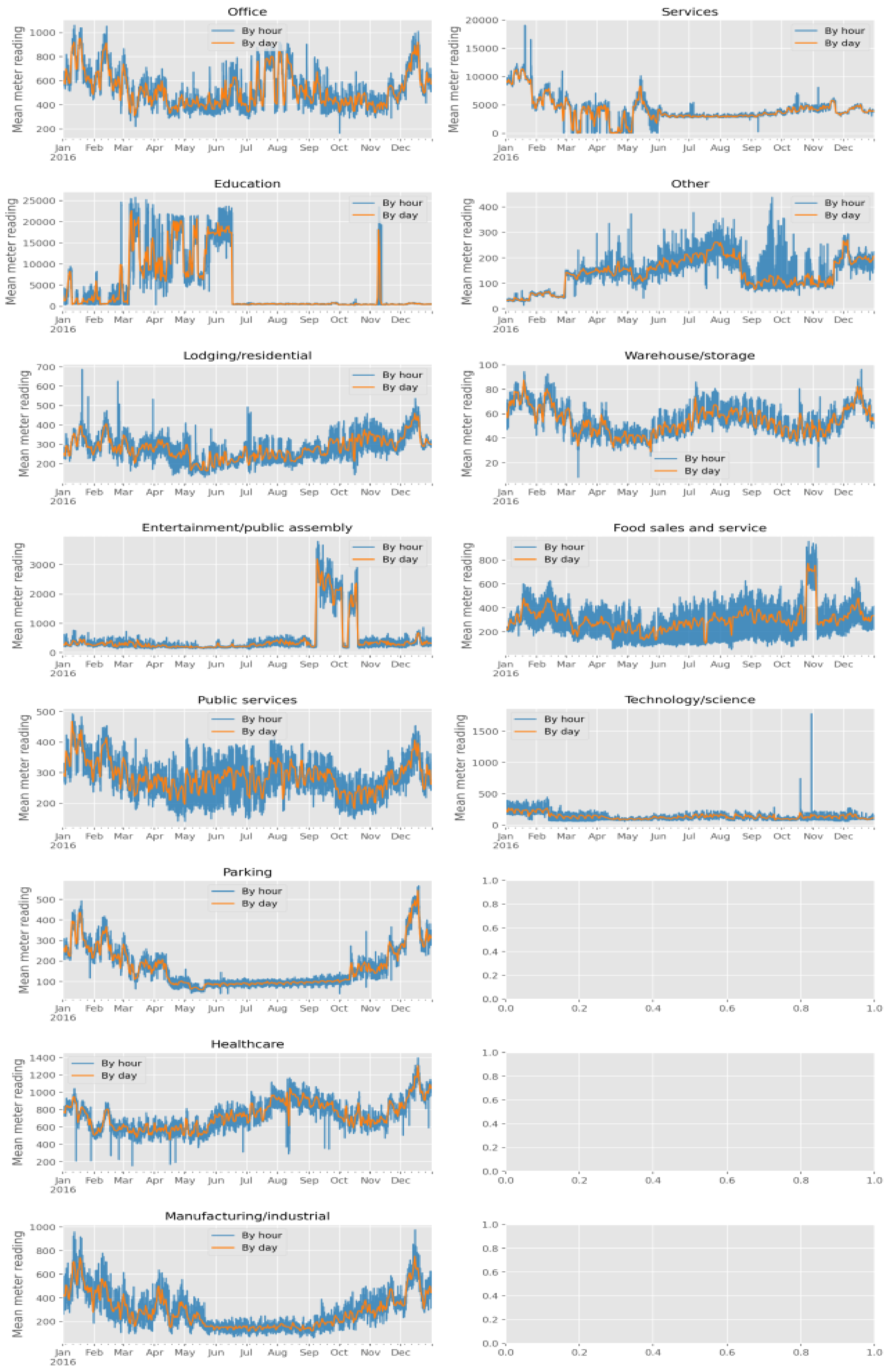


* site_id 0은 3월전까지는 측정값이 0이다. 해당 날짜이전의 특정 site _id 데이터는 필요 없다
* site_id 13이 전반적이 평균 meter_reading 하고 같다.

데이터 파악하기

site_id==13의 primary use에 따라 plot 그리기

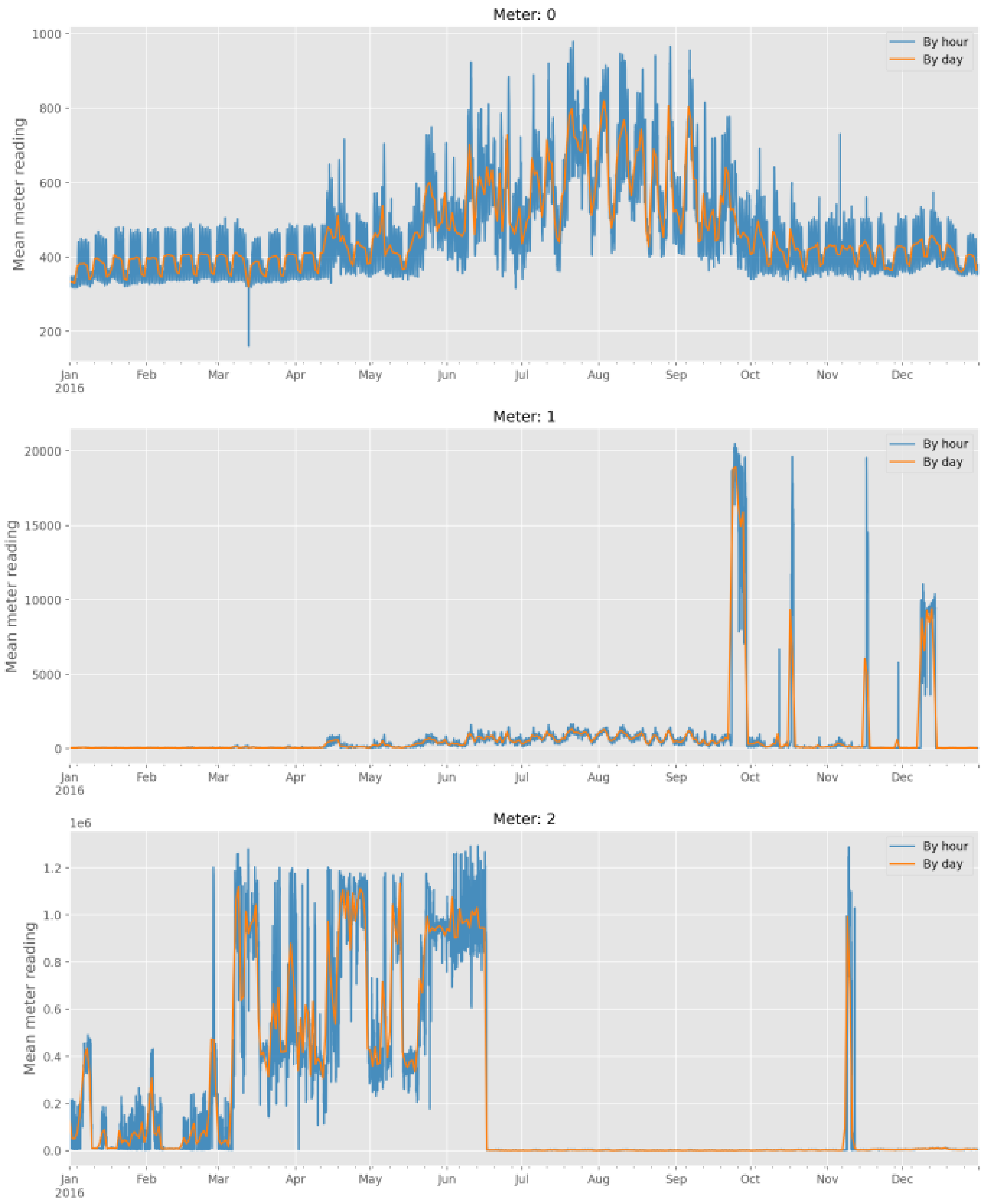




site_id == 13 및 primary_use == education은 앞에서 봤던 meter_reading의 일반적인 평균과 매우 비슷하다.

데이터 파악하기

site_id==13의 primary use==Education meter마다 plot 그리기

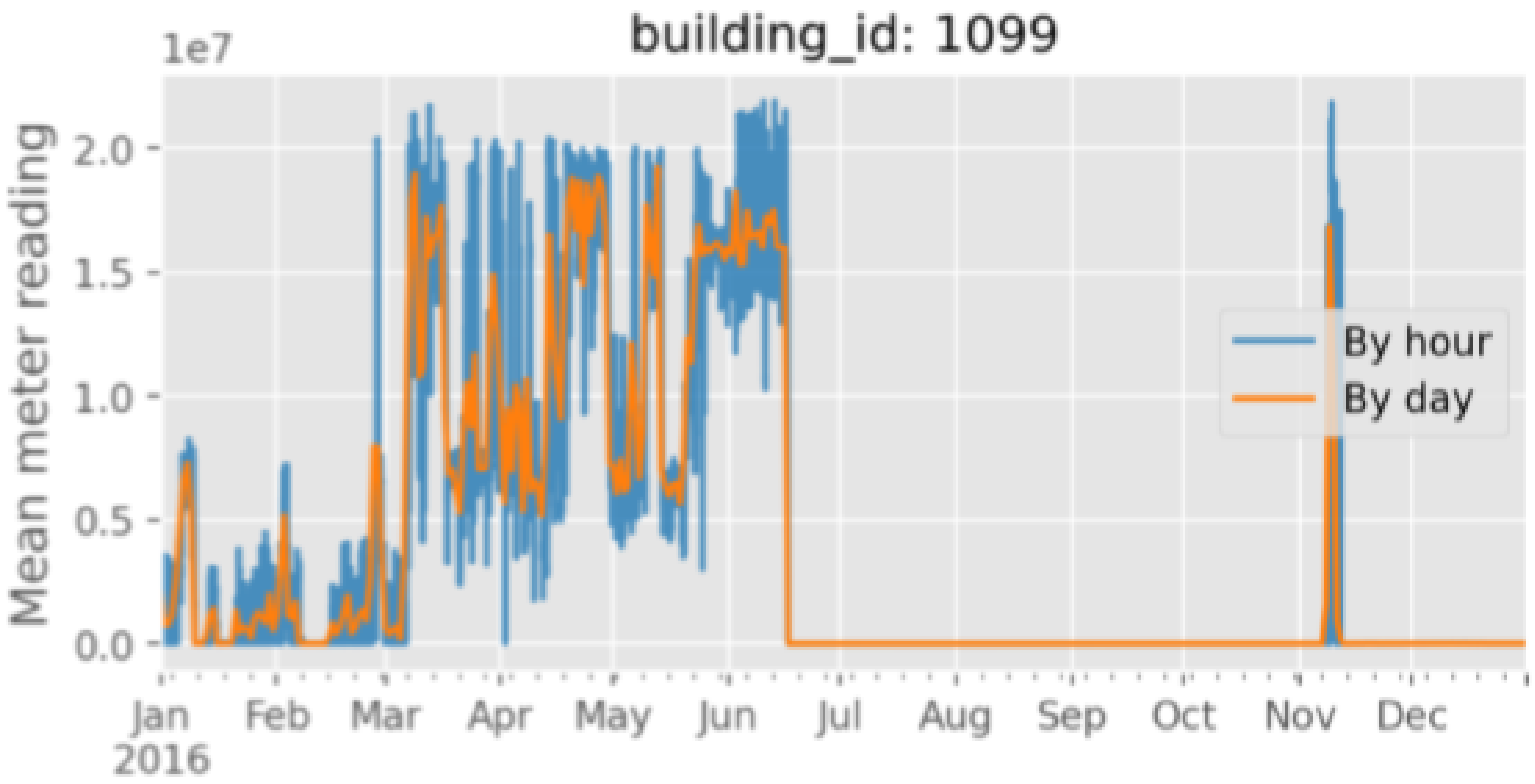


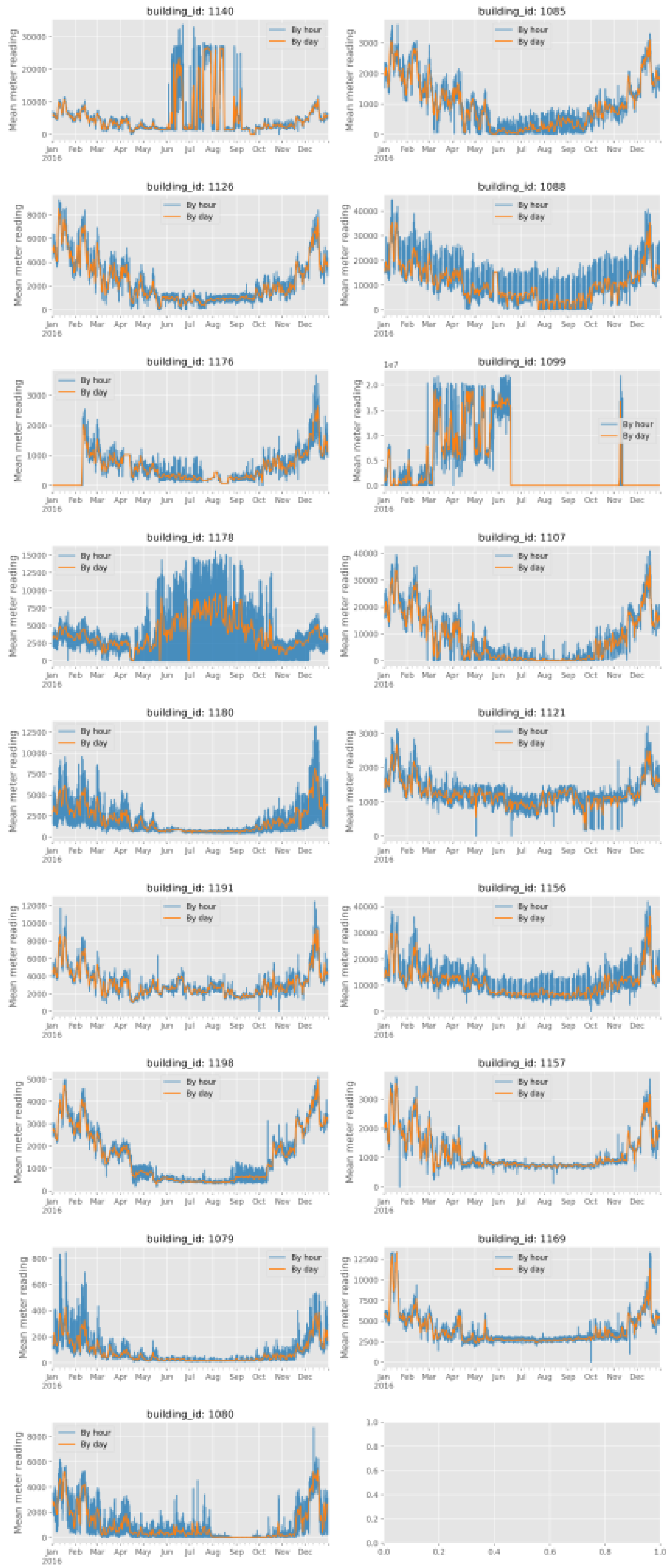
meter => {0: electricity, 1: chilledwater, 2: steam, 3: hot water}

site_id가 13이고 Education으로 사용되는 building에서는 hotwater를 안쓰는것을 알 수 있다

meter 2를 보게 되면 위에서 봤던 전반적인 plot하고 같은것을 확인할 수 있다

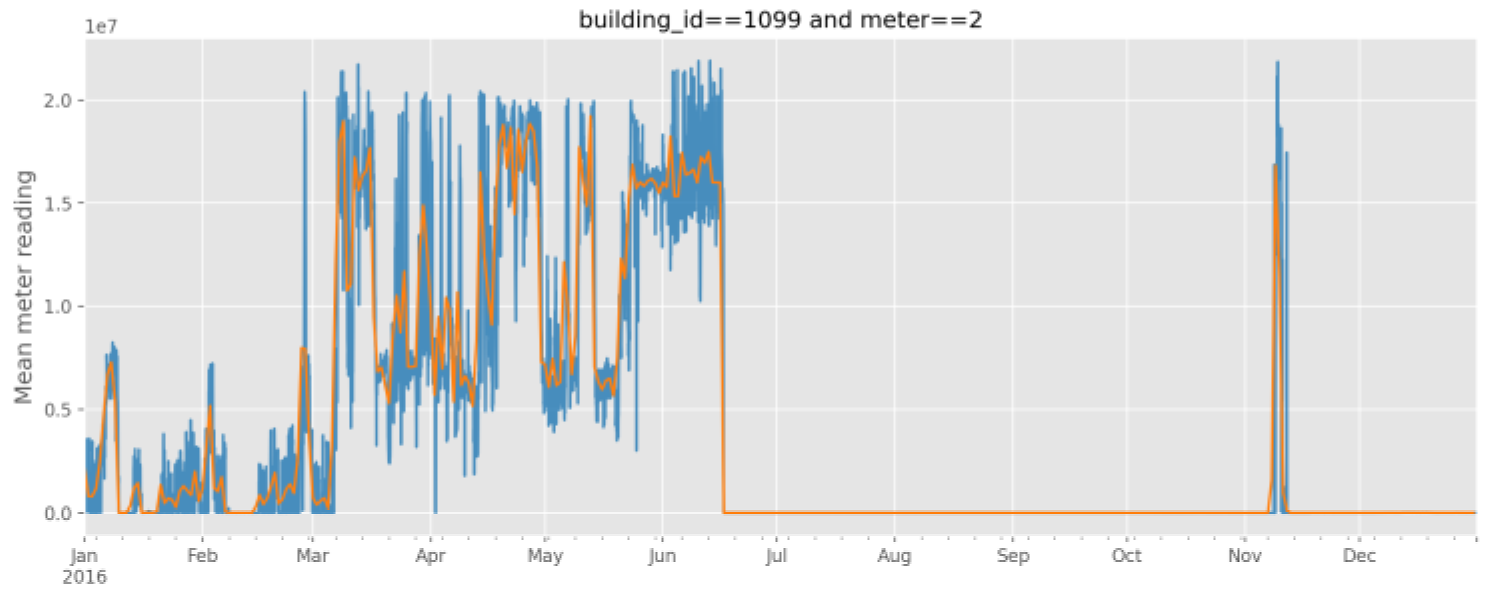
데이터 파악하기

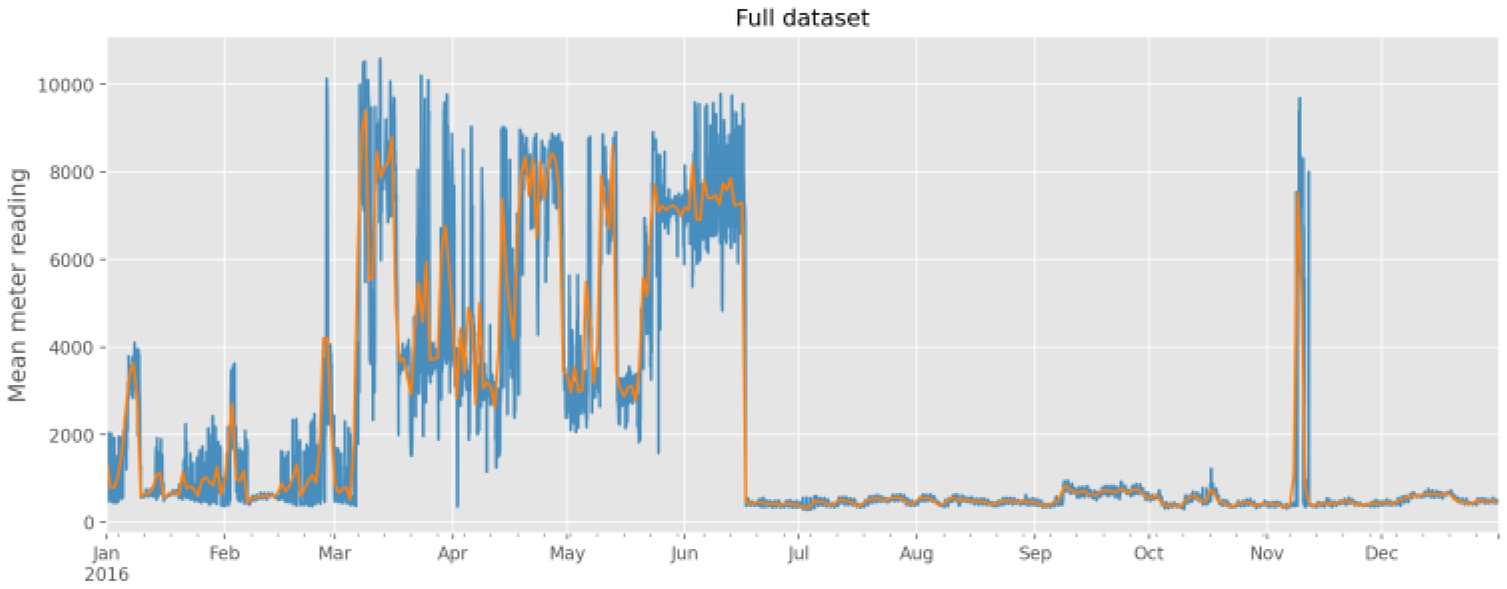

building_id: 1099
1e7
Mean meter reading
2.0
1.5
1.0
0.5
0.0
By hour
By day
Jan
2016
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec

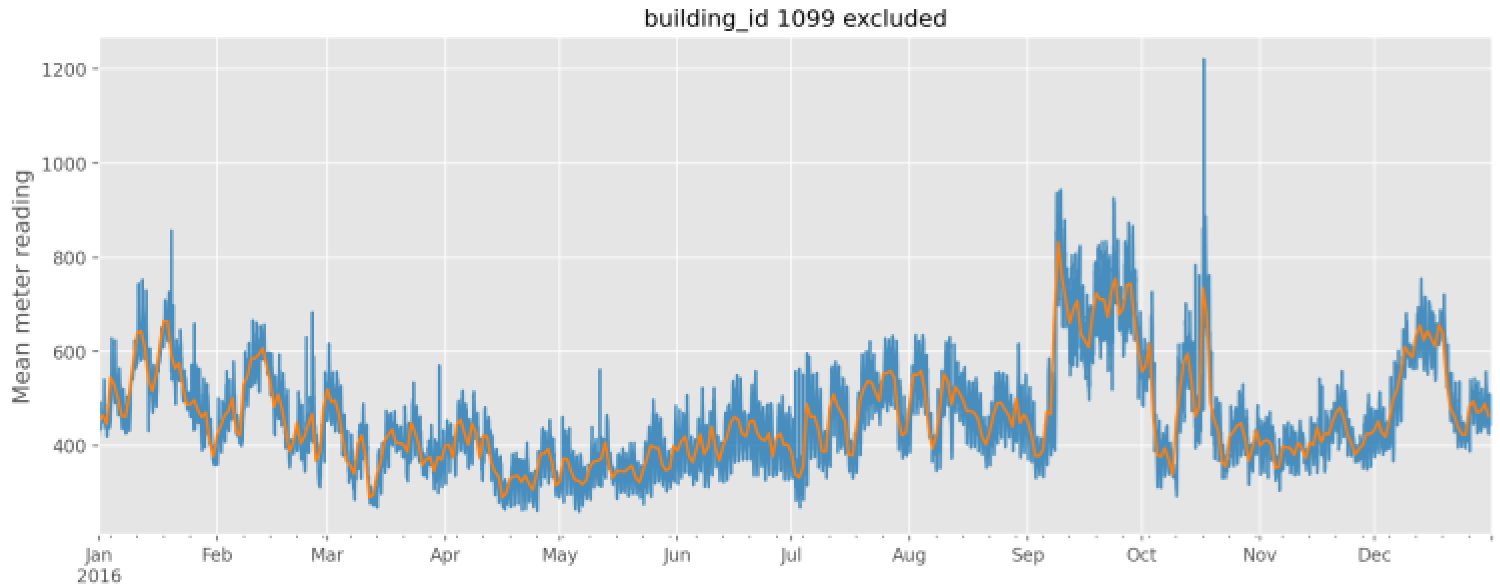

building_id: 1140
building_id: 1085
building_id: 1126
building_id: 1088
building_id: 1176
building_id: 1099
building_id: 1178
building_id: 1107
building_id: 1180
building_id: 1121
building_id: 1191
building_id: 1156
building_id: 1198
building_id: 1157
building_id: 1079
building_id: 1169
building_id: 1080

데이터 파악하기

building_id=1099가 outlier임을 알 수있다.







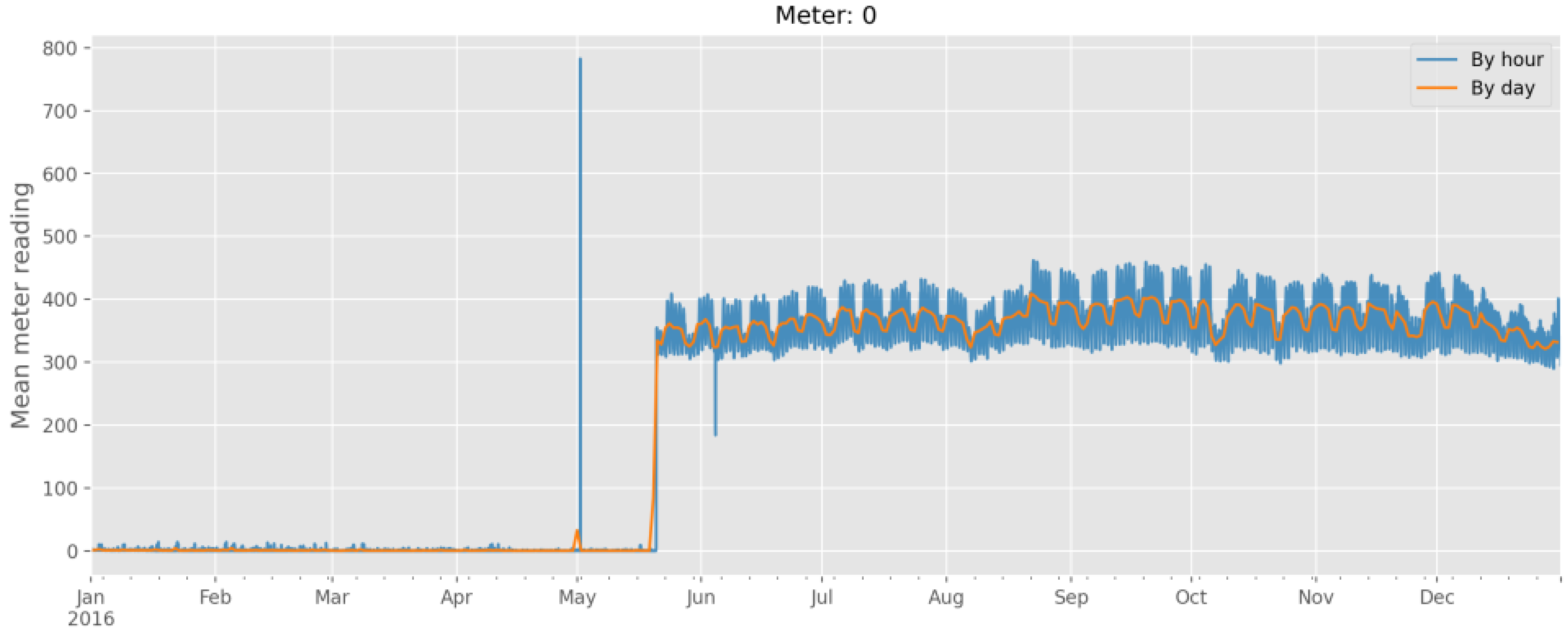
Import package & load data

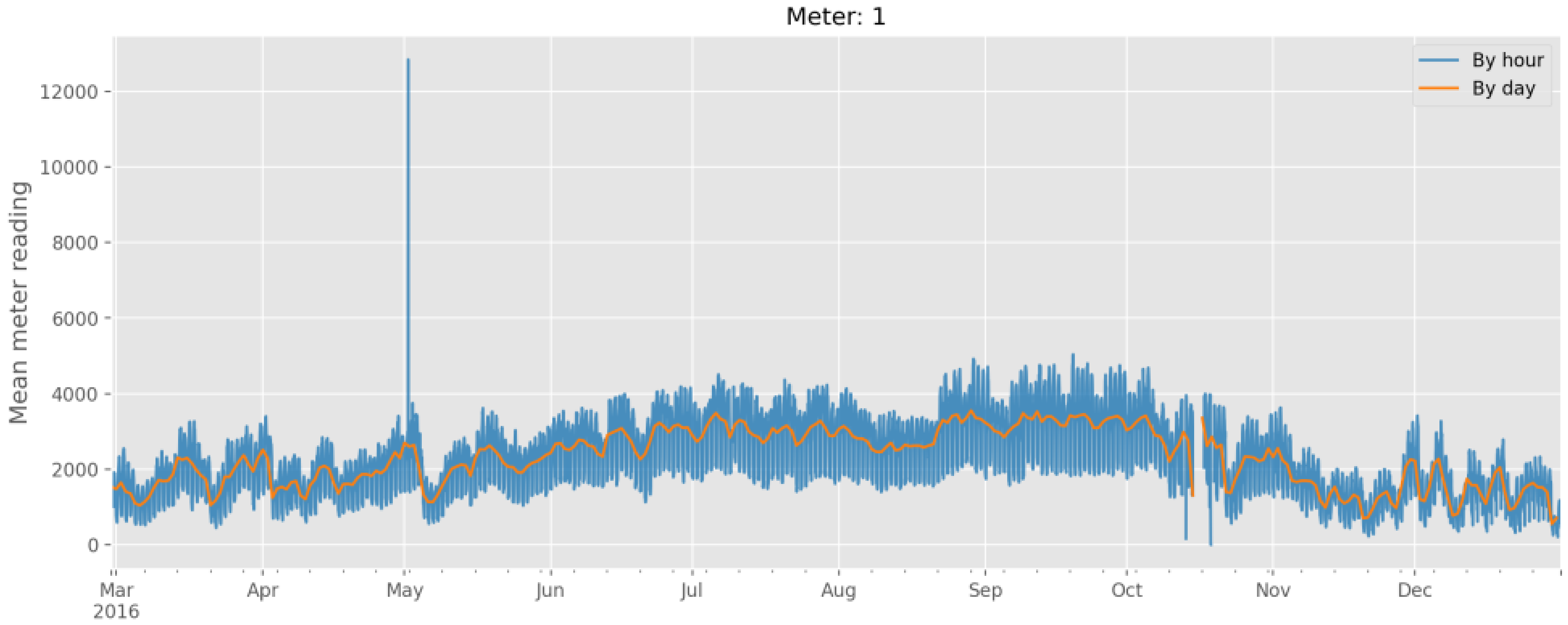
```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import lightgbm as lgb
     from sklearn.preprocessing import LabelEncoder
     from sklearn.model_selection import KFold
     import datetime
     import gc
```

```
[ ]  train_df = pd.read_csv('train.csv')
     weather_train_df = pd.read_csv('weather_train.csv')

     # Remove outliers
     train_df = train_df [ train_df['building_id'] != 1099 ]
     train_df = train_df.query('not (building_id <= 104 & meter == 0 & timestamp <= "2016-05-20")')

     building_meta_df = pd.read_csv('building_metadata.csv')
```

Import package & load data

site_id==0





	site_id	building_id	primary_use	square_feet	year_built	floor_count
0	0	0	Education	7432	2008.0	NaN
1	0	1	Education	2720	2004.0	NaN
2	0	2	Education	5376	1991.0	NaN
3	0	3	Education	23685	2002.0	NaN
4	0	4	Education	116607	1975.0	NaN
...	...	...	...	...	...	...
100	0	100	Lodging/residential	24456	1968.0	NaN
101	0	101	Office	18860	1986.0	NaN
102	0	102	Office	15876	1983.0	NaN
103	0	103	Education	21657	2016.0	NaN
104	0	104	Office	45330	2003.0	NaN

105 rows × 6 columns

https://www.kaggle.com/c/ashrae-energy-prediction/discussion/113054#656588

Utility Functions

reduce memory usage

데이터 프레임의 모든 열을 반복하고
데이터 타입을 수정하여 메모리 사용량을 줄인다

1. 모든 열에 대해 반복
2. datetime type이나 categorial type이면 skip
3. object 아닐 경우
   (1)최소값과 최대값 찾기
   (2)정수로 나타낼 수 있는지 확인
   (3)값 범위를 적합시킬 수 있는 가장 작은 데이터 유형 결정 및 적용
4. object일 경우 category type(범주형)으로 변경

```python
# Original code from https://www.kaggle.com/gemartin/load-data-reduce-memory-usage by @gemartin
# Modified to support timestamp type, categorical type
# Modified to add option to use float16 or not. feather format does not support float16.
from pandas.api.types import is_datetime64_any_dtype as is_datetime
from pandas.api.types import is_categorical_dtype

def reduce_mem_usage(df, use_float16=False):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        if is_datetime(df[col]) or is_categorical_dtype(df[col]):
            # skip datetime type or categorical type
            continue
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if use_float16 and c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))
```

https://www.kaggle.com/gemartin/load-data-reduce-memory-usage

Utility Functions

fill weather dataset

weather dataset에는 missing data가 많기 때문에 이를 채워줘야 한다.

```python
def fill_weather_dataset(weather_df):

    # Find Missing Dates
    time_format = "%Y-%m-%d %H:%M:%S"
    start_date = datetime.datetime.strptime(weather_df['timestamp'].min(),time_format)
    end_date = datetime.datetime.strptime(weather_df['timestamp'].max(),time_format)
    total_hours = int(((end_date - start_date).total_seconds() + 3600) / 3600)
    hours_list = [(end_date - datetime.timedelta(hours=x)).strftime(time_format) for x in range(total_hours)]

    missing_hours = []
    for site_id in range(16):
        site_hours = np.array(weather_df[weather_df['site_id'] == site_id]['timestamp'])
        new_rows = pd.DataFrame(np.setdiff1d(hours_list,site_hours),columns=['timestamp'])
        new_rows['site_id'] = site_id
        weather_df = pd.concat([weather_df,new_rows])

        weather_df = weather_df.reset_index(drop=True)

    # Add new Features
    weather_df["datetime"] = pd.to_datetime(weather_df["timestamp"])
    weather_df["day"] = weather_df["datetime"].dt.day
    weather_df["week"] = weather_df["datetime"].dt.week
    weather_df["month"] = weather_df["datetime"].dt.month

     # Reset Index for Fast Update
    weather_df = weather_df.set_index(['site_id','day','month'])

    air_temperature_filler = pd.DataFrame(weather_df.groupby(['site_id','day','month'])['air_temperature'].mean(),columns=["air_temperature"])
    weather_df.update(air_temperature_filler,overwrite=False)

    # Step 1
    cloud_coverage_filler = weather_df.groupby(['site_id','day','month'])['cloud_coverage'].mean()
    # Step 2
    cloud_coverage_filler = pd.DataFrame(cloud_coverage_filler.fillna(method='ffill'),columns=["cloud_coverage"])

    weather_df.update(cloud_coverage_filler,overwrite=False)

    due_temperature_filler = pd.DataFrame(weather_df.groupby(['site_id','day','month'])['dew_temperature'].mean(),columns=["dew_temperature"])
    weather_df.update(due_temperature_filler,overwrite=False)

    # Step 1
    sea_level_filler = weather_df.groupby(['site_id','day','month'])['sea_level_pressure'].mean()
    # Step 2
    sea_level_filler = pd.DataFrame(sea_level_filler.fillna(method='ffill'),columns=['sea_level_pressure'])

    weather_df.update(sea_level_filler,overwrite=False)
    wind_direction_filler =  pd.DataFrame(weather_df.groupby(['site_id','day','month'])['wind_direction'].mean(),columns=['wind_direction'])
    weather_df.update(wind_direction_filler,overwrite=False)

    wind_speed_filler =  pd.DataFrame(weather_df.groupby(['site_id','day','month'])['wind_speed'].mean(),columns=['wind_speed'])
    weather_df.update(wind_speed_filler,overwrite=False)

    # Step 1
    precip_depth_filler = weather_df.groupby(['site_id','day','month'])['precip_depth_1_hr'].mean()
    # Step 2
    precip_depth_filler = pd.DataFrame(precip_depth_filler.fillna(method='ffill'),columns=['precip_depth_1_hr'])

    weather_df.update(precip_depth_filler,overwrite=False)

    weather_df = weather_df.reset_index()
    weather_df = weather_df.drop(['datetime','day','week','month'],axis=1)

    return weather_df
```

https://www.kaggle.com/aitude/ashrae-missing-weather-data-handling

Utility Functions

fill weather dataset

(1)Missing Hours

이 csv에는 2016년 16개 지역에 대한 시간당 기상 정보가 있다.
(140,544개(16 x 24 x 366, 2016년은 윤년))
그러나 이 CSV에는 139,773개의 레코드가 있으므로 771시간의 데이터가 누락된다.

```
print(weather_train.shape)
```

```
(139773, 9)
```

```
# Find Missing Dates
time_format = "%Y-%m-%d %H:%M:%S"
start_date = datetime.datetime.strptime(weather_df['timestamp'].min(),time_format)
end_date = datetime.datetime.strptime(weather_df['timestamp'].max(),time_format)
total_hours = int(((end_date - start_date).total_seconds() + 3600) / 3600)
hours_list = [(end_date - datetime.timedelta(hours=x)).strftime(time_format) for x in range(total_hours)]

missing_hours = []
for site_id in range(16):
    site_hours = np.array(weather_df[weather_df['site_id'] == site_id]['timestamp'])
    new_rows = pd.DataFrame(np.setdiff1d(hours_list,site_hours),columns=['timestamp'])
    new_rows['site_id'] = site_id
    weather_df = pd.concat([weather_df,new_rows])

    weather_df = weather_df.reset_index(drop=True)
```

```
missing_statistics(weather_df)
```

	COLUMN NAME	MISSING VALUES	TOTAL ROWS	% MISSING
0	air_temperature	826	140544	0.59
1	cloud_coverage	69944	140544	49.77
2	dew_temperature	884	140544	0.63
3	precip_depth_1_hr	51060	140544	36.33
4	sea_level_pressure	11389	140544	8.10
5	site_id	0	140544	0.00
6	timestamp	0	140544	0.00
7	wind_direction	7039	140544	5.01
8	wind_speed	1075	140544	0.76

setdiff1d(x, y) : 첫번째 배열 x로 부터 두번째 배열 y를 뺀 차집합을 반환

Utility Functions

fill weather dataset
(2)Add Day,Week & Month Columns

```
# Add new Features
weather_df["datetime"] = pd.to_datetime(weather_df["timestamp"])
weather_df["day"] = weather_df["datetime"].dt.day
weather_df["week"] = weather_df["datetime"].dt.week
weather_df["month"] = weather_df["datetime"].dt.month
```

이 데이터 집합은 시간당 날씨 정보로 구성된다.
따라서 새로운 날짜 feature 바탕으로 결측값을 채울 것이다

Utility Functions

fill weather dataset

(3)Reset Index for Fast Update

```
 # Reset Index for Fast Update
weather_df = weather_df.set_index(['site_id','day','month'])

air_temperature_filler = pd.DataFrame(weather_df.groupby(['site_id','day','month'])['air_temperature'].mean(),columns=["air_temperature"])
weather_df.update(air_temperature_filler,overwrite=False)

# Step 1
cloud_coverage_filler = weather_df.groupby(['site_id','day','month'])['cloud_coverage'].mean()
# Step 2
cloud_coverage_filler = pd.DataFrame(cloud_coverage_filler.fillna(method='ffill'),columns=["cloud_coverage"])

weather_df.update(cloud_coverage_filler,overwrite=False)
```

<air temperature>missing air 온도를 해당 월의 평균 온도로 채운다.
계절에 따라 온도가 달라지고 이는 월마다 온도가 달라진다는 뜻

<cloud coverage>거의 50%의 데이터가 누락되었다. 따라서 먼저 해당 월의 평균 구름 범위를 계산한 다음 유효한 마지막 관측치로 나머지 결측값을 채운다.

DataFrame. update (other , join = 'left' , overwrite = True , filter_func = None , errors = 'ignore')
다른 data frame에서의 non-NA 값을 가지고 update
other : 원본 DataFrame과 일치하는 인덱스 / 열 레이블이 하나 이상 있어야함
overwrite = False : 원래 DataFrame에서 NA 인 값만 업데이트

Utility Functions

fill weather dataset

(3)Reset Index for Fast Update

```
due_temperature_filler = pd.DataFrame(weather_df.groupby(['site_id','day','month'])['dew_temperature'].mean(),columns=["dew_temperature"])
weather_df.update(due_temperature_filler,overwrite=False)

# Step 1
sea_level_filler = weather_df.groupby(['site_id','day','month'])['sea_level_pressure'].mean()
# Step 2
sea_level_filler = pd.DataFrame(sea_level_filler.fillna(method='ffill'),columns=['sea_level_pressure'])

weather_df.update(sea_level_filler,overwrite=False)
wind_direction_filler =  pd.DataFrame(weather_df.groupby(['site_id','day','month'])['wind_direction'].mean(),columns=['wind_direction'])
weather_df.update(wind_direction_filler,overwrite=False)

wind_speed_filler =  pd.DataFrame(weather_df.groupby(['site_id','day','month'])['wind_speed'].mean(),columns=['wind_speed'])
weather_df.update(wind_speed_filler,overwrite=False)

# Step 1
precip_depth_filler = weather_df.groupby(['site_id','day','month'])['precip_depth_1_hr'].mean()
# Step 2
precip_depth_filler = pd.DataFrame(precip_depth_filler.fillna(method='ffill'),columns=['precip_depth_1_hr'])

weather_df.update(precip_depth_filler,overwrite=False)

weather_df = weather_df.reset_index()
weather_df = weather_df.drop(['datetime','day','week','month'],axis=1)

return weather_df
```

Utility Functions

features engineering

```
def features_engineering(df):


    # Add more features
    df["timestamp"] = pd.to_datetime(df["timestamp"],format="%Y-%m-%d %H:%M:%S")
    df["hour"] = df["timestamp"].dt.hour
    df["weekend"] = df["timestamp"].dt.weekday
    df['square_feet'] =  np.log1p(df['square_feet'])

    # Remove Unused Columns
    drop = ["timestamp","sea_level_pressure", "wind_direction", "wind_speed","year_built","floor_count"]
    df = df.drop(drop, axis=1)
    gc.collect()

    # Encode Categorical Data
    le = LabelEncoder()
    df["primary_use"] = le.fit_transform(df["primary_use"])

    return df
```

LabelEncoder :
범주형 변수(primary_use)를 처리.
레이블 인코딩은 각 고유 값을 다른 정수에 할당

Train

```
weather_train_df = fill_weather_dataset(weather_train_df)
```

```
train_df = reduce_mem_usage(train_df,use_float16=True)
building_meta_df = reduce_mem_usage(building_meta_df,use_float16=True)
weather_train_df = reduce_mem_usage(weather_train_df,use_float16=True)
```

```
Memory usage of dataframe is 757.31 MB
Memory usage after optimization is: 322.24 MB
Decreased by 57.4%
Memory usage of dataframe is 0.07 MB
Memory usage after optimization is: 0.02 MB
Decreased by 73.8%
Memory usage of dataframe is 9.65 MB
Memory usage after optimization is: 2.66 MB
Decreased by 72.5%
```

```
train_df = features_engineering(train_df)
```

```
train_df = train_df.merge(building_meta_df, left_on='building_id',right_on='building_id',how='left')
train_df = train_df.merge(weather_train_df,how='left',left_on=['site_id','timestamp'],right_on=['site_id','timestamp'])
del weather_train_df
gc.collect()
```

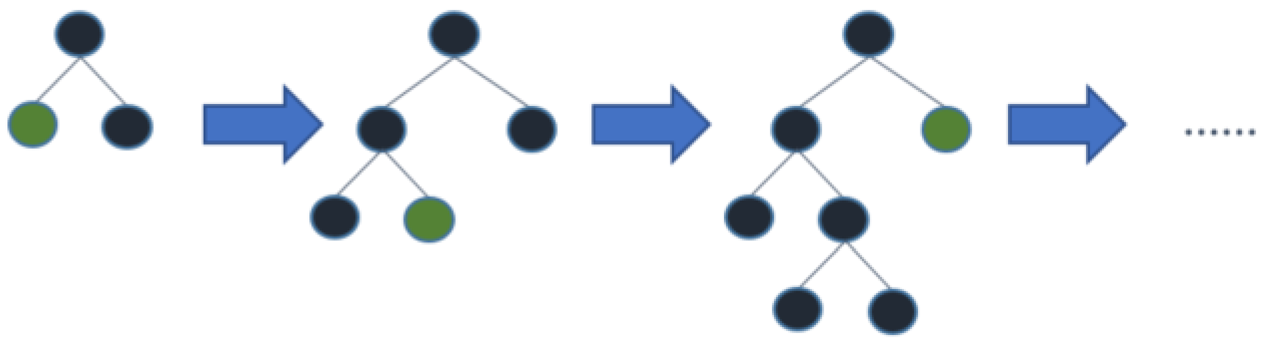
```
target = np.log1p(train_df["meter_reading"])
features = train_df.drop('meter_reading', axis = 1)
del train_df
gc.collect()
```

KFOLD LIGHTGBM Model

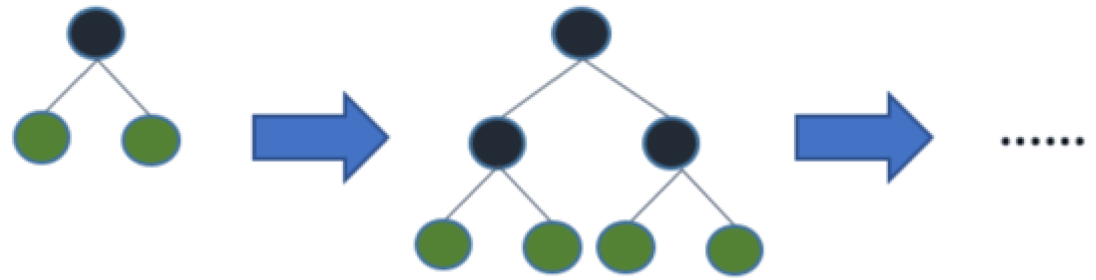
```
categorical_features = ["building_id", "site_id", "meter", "primary_use", "weekend"]
params = {
    "objective": "regression",
    "boosting": "gbdt",
    "num_leaves": 1280,
    "learning_rate": 0.05,
    "feature_fraction": 0.85,
    "reg_lambda": 2,
    "metric": "rmse",
}
kf = KFold(n_splits=3)
models = []
for train_index,test_index in kf.split(features):
    train_features = features.loc[train_index]
    train_target = target.loc[train_index]

    test_features = features.loc[test_index]
    test_target = target.loc[test_index]

    d_training = lgb.Dataset(train_features, label=train_target,categorical_feature=categorical_features, free_raw_data=False)
    d_test = lgb.Dataset(test_features, label=test_target,categorical_feature=categorical_features, free_raw_data=False)

    model = lgb.train(params, train_set=d_training, num_boost_round=1000, valid_sets=[d_training,d_test], verbose_eval=25, early_stopping_rounds=50)
    models.append(model)
    del train_features, train_target, test_features, test_target, d_training, d_test
    gc.collect()
```

KFOLD LIGHTGBM Model

LightGBM :
LightGBM은 리프 중심 트리 분할(Leaf Wise) 방식을 사용한다.
Light GBM은 Tree가 수직적으로 확장되는 반면에 다른 알고리즘은 Tree가 수평적으로 확장된다, 즉 Light GBM은 leaf-wise 인 반면 다른 알고리즘은 level-wise. 확장하기 위해서 max delta loss를 가진 leaf를 선택하게 되는 것. 동일한 leaf를 확장할 때, leaf-wise 알고리즘은 level-wise 알고리즘보다 더 많은 loss, 손실을 줄일 수 있다.



Leaf-wise tree growth



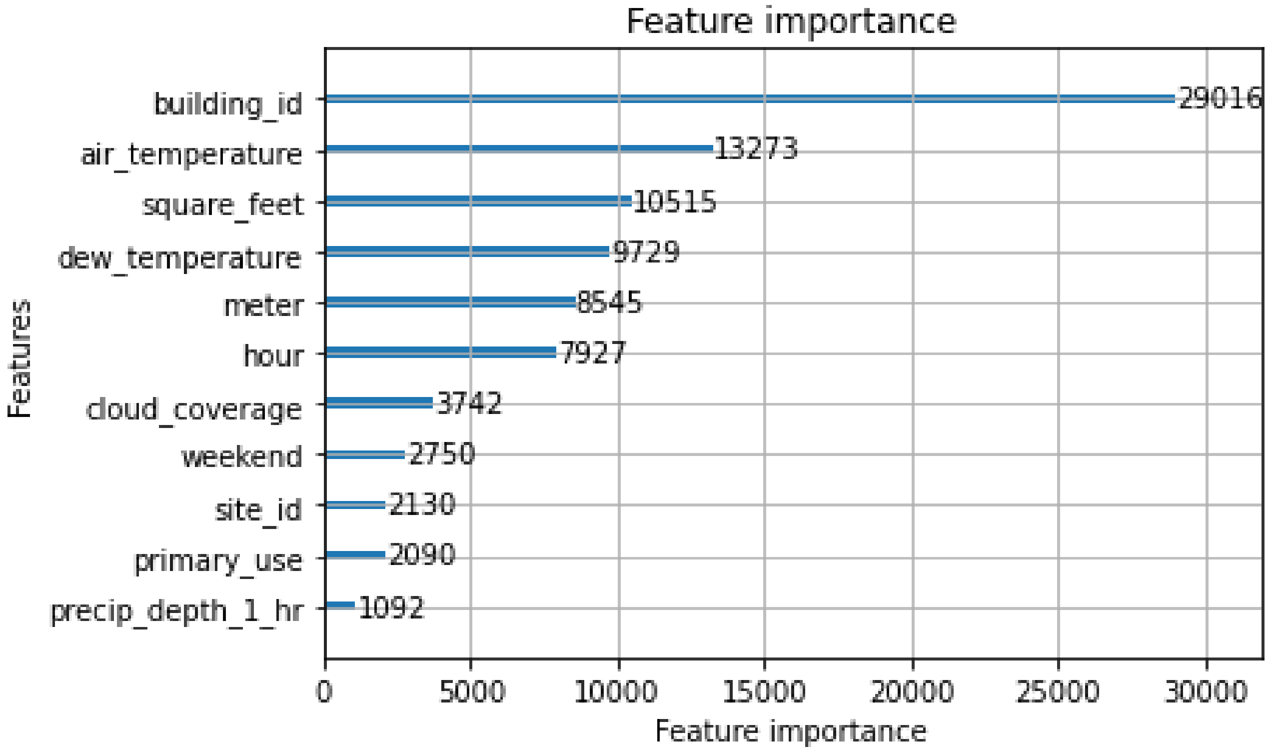
Level-wise tree growth

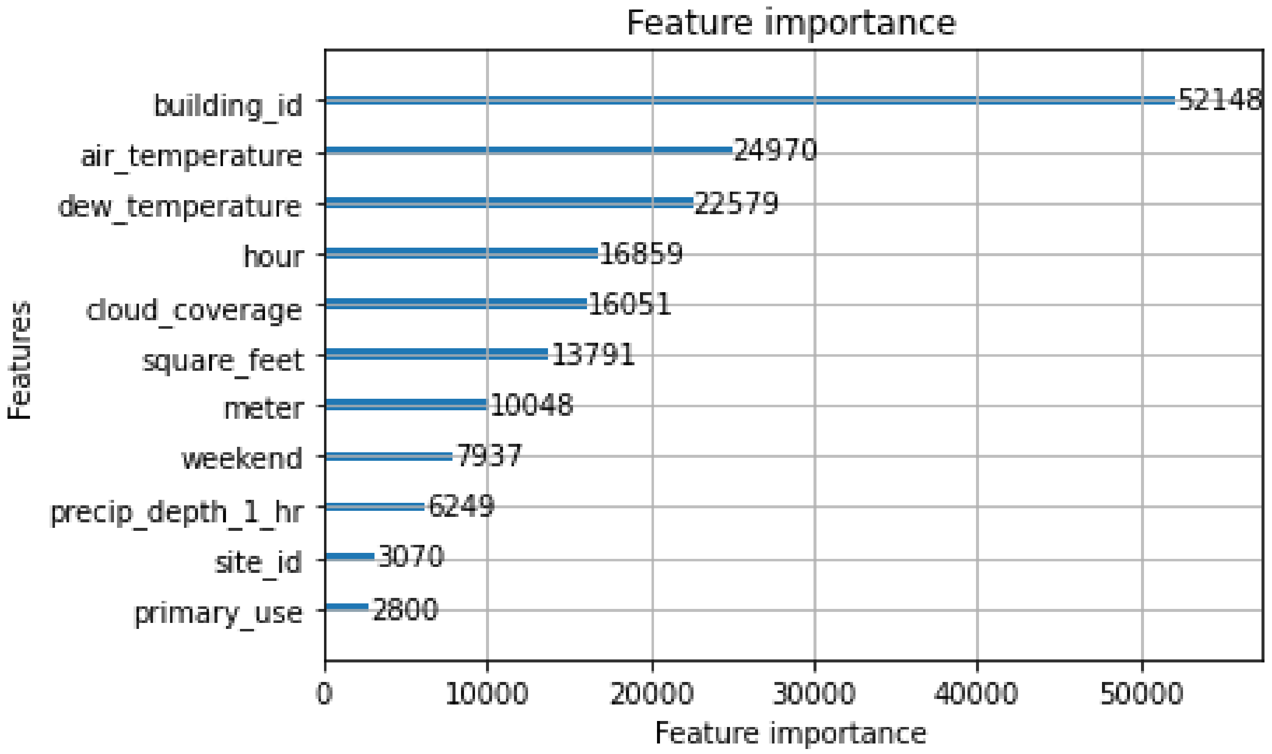
- n_estimators : 반복하려는 트리의 개수
- learning_rate : 학습률
- max_depth : 트리의 최대 깊이
- min_child_samples : 리프 노드가 되기 위한 최소한의 샘플 데이터 수
- num_leaves : 하나의 트리가 가질 수 있는 최대 리프 개수
- feature_fraction : 트리를 학습할 때마다 선택하는 feature의 비율
- reg_lambda : L2 regularization
- reg_alpha : L1 regularization

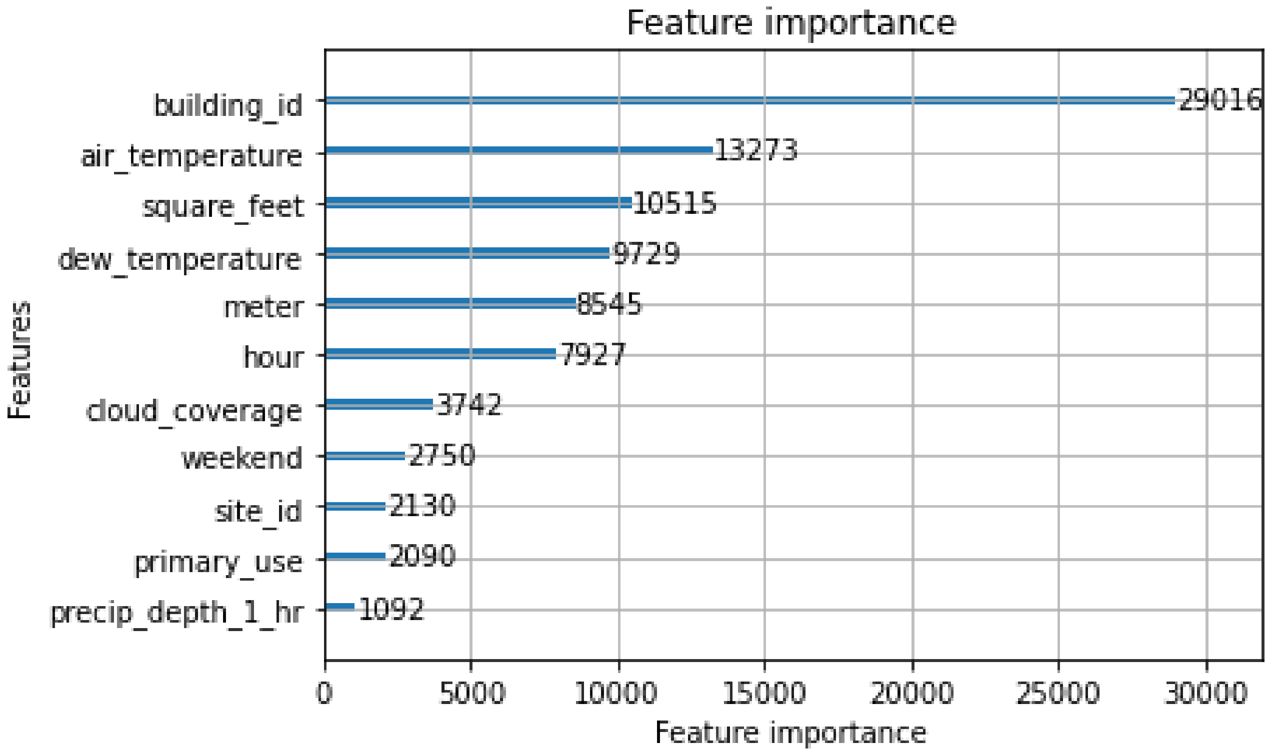
https://www.kaggle.com/aitude/ashrae-hyperparameter-tuning

Important Features

```
for model in models:
    lgb.plot_importance(model)
    plt.show()
```







Test

```
test_df = pd.read_csv('test.csv')
row_ids = test_df["row_id"]
test_df.drop("row_id", axis=1, inplace=True)
test_df = reduce_mem_usage(test_df)
```

```
test_df = test_df.merge(building_meta_df,left_on='building_id',right_on='building_id',how='left')
del building_meta_df
gc.collect()
```

```
weather_df = pd.read_csv('weather_test.csv')
weather_df = fill_weather_dataset(weather_df)
weather_df = reduce_mem_usage(weather_df)
```

```
test_df = test_df.merge(weather_df,how='left',on=['timestamp','site_id'])
del weather_df
gc.collect()
```

```
test_df = features_engineering(test_df)
```

Prediction

```
# results = []
# for model in models:
#     if  results == []:
#         results = np.expm1(model.predict(test_df, num_iteration=model.best_iteration)) / len(models)
#     else:
#         results += np.expm1(model.predict(test_df, num_iteration=model.best_iteration)) / len(models)
#     del model
#     gc.collect()



stepsize = 1000000
results = np.zeros(test_df.shape[0])
for model in models:
  predictions = []
  for i in range(0, test_df.shape[0], stepsize):
    predictions.append(np.expm1(model.predict(test_df.loc[i:i+stepsize-1,:], num_iteration=model.best_iteration)))
  results += (1 / len(models)) * np.concatenate(predictions, axis=0)
  del model
```

Submission

```
# assert(results.shape[0] == test_.shape[0])
results_df = pd.DataFrame({"row_id": row_ids, "meter_reading": np.clip(results, 0, None)})
results_df.to_csv("submission.csv", index=False)

# results_df = pd.DataFrame({"row_id": row_ids, "meter_reading": np.clip(results, 0, a_max=None)})
# del row_ids,results
# gc.collect()
# results_df.to_csv("submission.csv", index=False)
```

Reference

- 사자처럼 우아하게
- ASHRAE -Start Here: A GENTLE Introduction
- ASHRAE- KFold LightGBM - without leak (1.08)
- https://github.com/YongseonKim/Kaggle
- ASHRAE: Training LGBM by meter type