ML Project

Enefit - Predict Energy Behavior of Prosumers

5조
(권혁찬, 김창희, 문정의)

CONTENT

- Background
- Task Process
- Dataset
- Preprocessing
- EDA
- Feature Engineering
- Model
- Hyper-parameter Tuning
- Conclusion
- Lesson Learn

1. Background

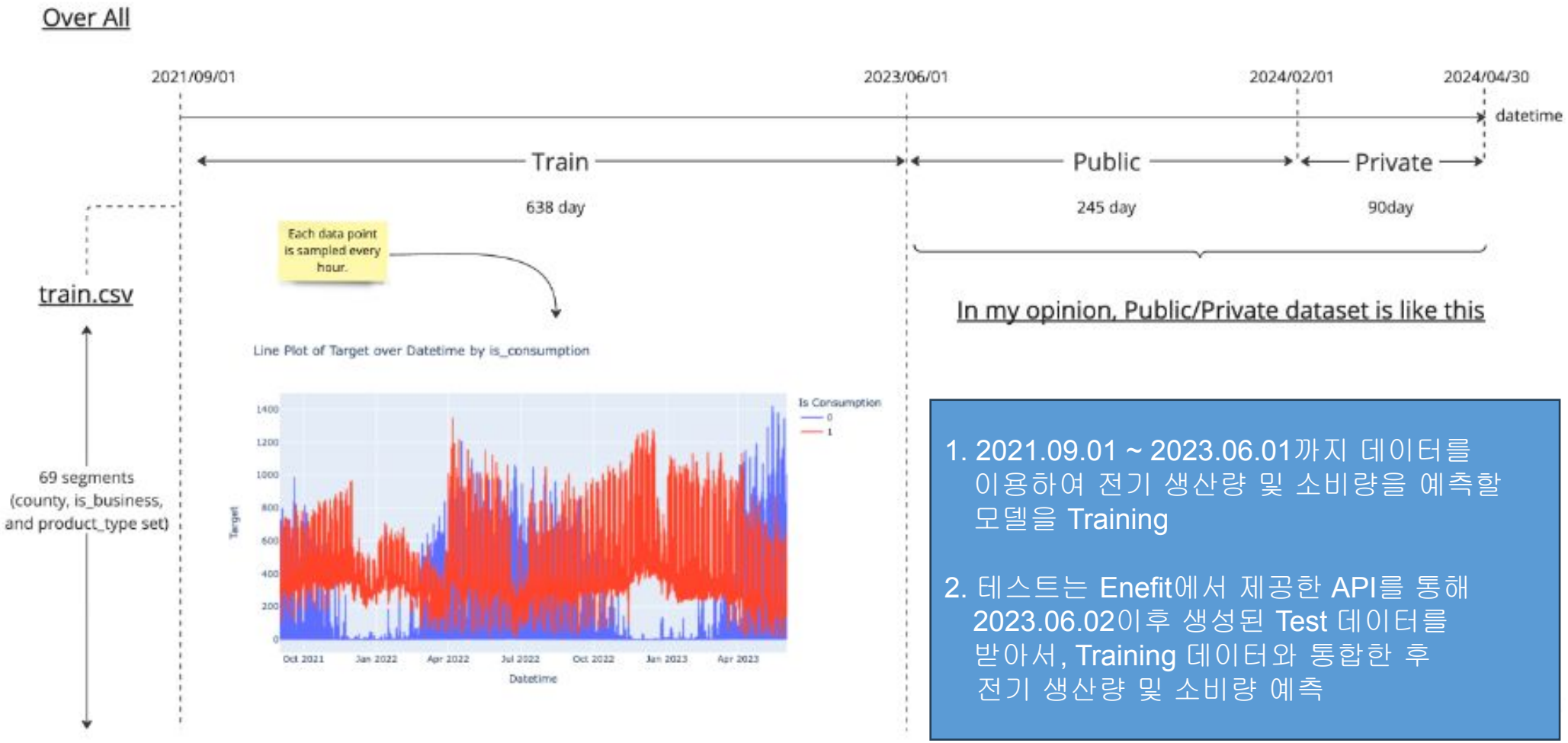
문제점

- 고객들에 의해서 소비 또는 생산될 것으로 예상되는 에너지 예측값이 실제 소비와 생산과 차이가 많음.
- 이로 인해 에너지 회사들에게 물류 및 재정적 문제가 발생하고 있음.

목표

- 에너지 생산 및 소비의 불균형 비용을 줄이기 위해 기존 예측 업체 보다 성능이 향상된 에너지 예측 모델을 구축 (에스토니아 에너지 고객들에 의해서 생산되고 소비되는 전기량을 예측)

1. Background



1. 2021.09.01 ~ 2023.06.01까지 데이터를 이용하여 전기 생산량 및 소비량을 예측할 모델을 Training

2. 테스트는 Enefit에서 제공한 API를 통해 2023.06.02이후 생성된 Test 데이터를 받아서, Training 데이터와 통합한 후 전기 생산량 및 소비량 예측

2. Task Process

1단계

EDA 또는 Baseline 노트북 참조

- Dataset 분석
- EDA 분석
- Base Model 분석

2단계

Public Score 높은 노트북 참조

- Feature Engineering 분석
- Best Model 분석

3단계

튜닝 및 모델개선

- Optuna를 이용한 Best Hyper-parameter 추출
- Submission 및 Public Score 향상
- Best Model 수정

4단계

EDA, Feature Engeering 반복 수행

- EDA 반복 수행
- Feature Engineering을 통한 New Feature 추출
- Hyper-parameter 추출 및 Submission

3. Dataset

- Dataset List

Dataset	description
train.csv	• 각 segment의 일시별 전기 생산량 또는 소비량 정보 (target) (2021-09-01 ~ 2023-05-31),메인 DataFrame • 2,018,352 rows x 9 columns
client.csv	• 각 segement(county, is_business, product_type)의 일별 eic_count, installed_capacitiy 데이터 (소비 포인트, 태양광 패널 설치 용량) • 41,919 rows x 7 columns
gas_prices.csv	• nature gas 일별 예측 가격 (최고, 최저 가격) • 637 rows x 5 columns
electricity_prices.csv	• 시간당 전기 Kw의 예측 가격 • 15,286 rows x 4 columns
Forecast_weather.csv	• 에스토니아 기상 예측 정보 (예측일로부터 48시간까지 예측) • 3,424,512 rows x 18 columns
Historical_weather.csv	• 에스토니아 과거 기상 정보 (매일 11시에 data_block_id 갱신) • 1,710,800 rows x 18 columns
weather_station_to_county_mapping.csv	• 위도 경도별 county 맵핑 데이터 • 112 rows x 4 columns

3. Dataset

3.1 train.csv

Column	description	Null count
county	0 ~ 15 자치주 코드	0
is_business	prosumer의 사업 여부	0
product_type	0 ~ 3까지 4가지의 계약 형태	0
target	전기 소비량 or 생산량	528
is_consumption	0 - 생산, 1 - 소비	0
datetime	에스토니아 시간(UTC+2)	0
data_block_id	데이터를 합칠 때 기준이 되는 id	0
row_id	row 고유 id, 0 ~ 2,018,351	0
prediction_unit_id	county, is_business, product_type의 조합에 대한 그룹 식별 코드 Segment이며 총 68개 존재함	0

3. Dataset

3.2 client.csv

Column	description	Null count
product_type	0 ~ 3까지 4가지의 계약 형태	0
county	0 ~ 15 자치주 코드	0
eic_count	집계된 소비 포인트 수	0
installed_capacity	설치된 태양광 패널의 용량(kW)	0
is_business	prosumer의 사업 여부	0
date	날짜	0
data_block_id	데이터를 합칠 때 기준이 되는 id	0

3. Dataset

3.3 gas_prices.csv

Column	description	Null count
forecast_date	예측 날짜 (1일 다음날)	0
lower_price_per_mwh	natural gas의 최저값	0
highest_price_per_mwh	natural gas의 최고값	0
origin_date	예측을 한 날짜	0
data_block_id	데이터를 합칠 때 기준이 되는 id	0

3. Dataset

3.4 electricity_prices.csv

Column	description	Null count
forecast_date	예측 날짜 (1일 다음날)	0
euros_per_mwh	전기 가격(유로)	0
origin_Date	예측을 한 날짜	0
data_block_id	데이터를 합칠 때 기준이 되는 id	0

3. Dataset

3.5 forecast_weather.csv

Column	description	Null count
latitude	위도	0
longitude	경도	0
origin_datetime	예측을 한 날짜	0
hours_ahead	forcast_datetime과 orign_datetime의 시간차이	0
temperature	온도	0
dewpoint	이슬점	0
cloudover(high, mid, low, total)	구름에 의해 하늘이 가려진 비율	0
10_metre_[u/v]_wind_component	10미터 상공의 풍속	0
data_block_id	데이터를 합칠 때 기준이 되는 id	0
forecast_datetime	예측 날짜	0
direct_solar_radiation	직달일사량	0
surface_solar_radiation_downwards	지표면의 태양 복사량	0
snowfall	적설량	0
Total_precipitation	총 강수량	0

3. Dataset

3.6 historical_weather.csv

Column	description	Null count
latitude / longitude	위도 / 경도	0
diffuse_radiation	산란일사량	0
datetime	날짜	0
shortwave_radiation	단파복사량	0
temperature	온도	0
dewpoint	이슬점	0
cloudover(high, mid, low, total)	구름에 의해 하늘이 가려진 비율	0
winddirection_10m	10미터 상공의 풍향	0
data_block_id	데이터를 합칠 때 기준이 되는 id	0
windspeed_10m	10미터 상공의 풍속	0
direct_solar_radiation	직달일사량	0
surface_pressure	지표면 기압	0
snowfall	적설량	0
rain	강수량	0

3. Dataset

3.7 weather_station_to_county_mapping.csv

Column	description	Null count
county_name	county명	63
longtitude	경도	0
latitude	위도	0
county	county code	0

4. Data Preprocessing

- 데이터셋 병합(Merging)

```python
def merge_dataset(df_data, df_client, df_gas, df_electricity, df_forecast, df_historical, df_location):
    df_data = (
            df_data
            .with_columns(
                pl.col("datetime").cast(pl.Date).alias("date"),
            )
        )

    df_client = (
        df_client
        .with_columns(
            (pl.col("date") + pl.duration(days=2)).cast(pl.Date)
        )
    )

    df_gas = (
        df_gas
        .rename({"forecast_date": "date"})
    )

    df_electricity = (
            df_electricity
            .rename({"forecast_date": "datetime"})
            .with_columns(
                pl.col("datetime") + pl.duration(days=1)
            )
    )

    df_location = (
        df_location
        .with_columns(
            pl.col("latitude").cast(pl.datatypes.Float32),
            pl.col("longitude").cast(pl.datatypes.Float32)
        )
    )

    df_forecast = (
            df_forecast
            .rename({"forecast_datetime": "datetime"})
            .with_columns(
                pl.col("latitude").cast(pl.datatypes.Float32),
                pl.col("longitude").cast(pl.datatypes.Float32),
                pl.col("datetime").dt.convert_time_zone("Europe/Bucharest")
            )
            .join(df_location, how="left", on=["longitude", "latitude"])
            .drop("longitude", "latitude")
    )
```

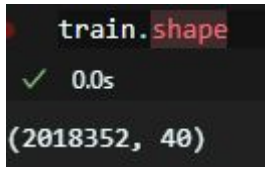
```python
    df_historical = (
        df_historical
        .with_columns(
            pl.col("latitude").cast(pl.datatypes.Float32),
            pl.col("longitude").cast(pl.datatypes.Float32),
            pl.col("datetime") + pl.duration(hours=37)
        )
        .join(df_location, how="left", on=["longitude", "la
        .drop("longitude", "latitude")
    )

    df_historical_date = (
        df_historical
        .group_by("datetime").mean()
        .drop("county")
    )

    df_data = (
        df_data
        .join(df_gas, on="date", how="left")
        .join(df_client, on=["county", "is_business", "product_type", "date"], how="left")
        .join(df_electricity, on="datetime", how="left")
        .join(df_forecast_date, on="datetime", how="left", suffix="_fd")
        .join(df_historical_date, on="datetime", how="left", suffix="_hd")
    )
    return df_data
```

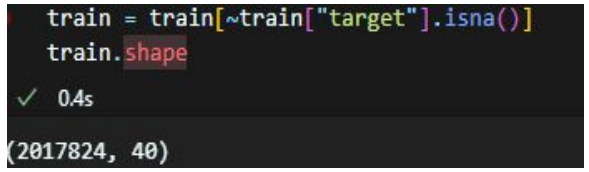
- Train
- Client
- Gas_prices
- Electricity_prices
- Location
- Forcast_weather
- Historical_weather

4. Data Preprocessing

- Drop target null

- target이 null인 528개의 row 제외

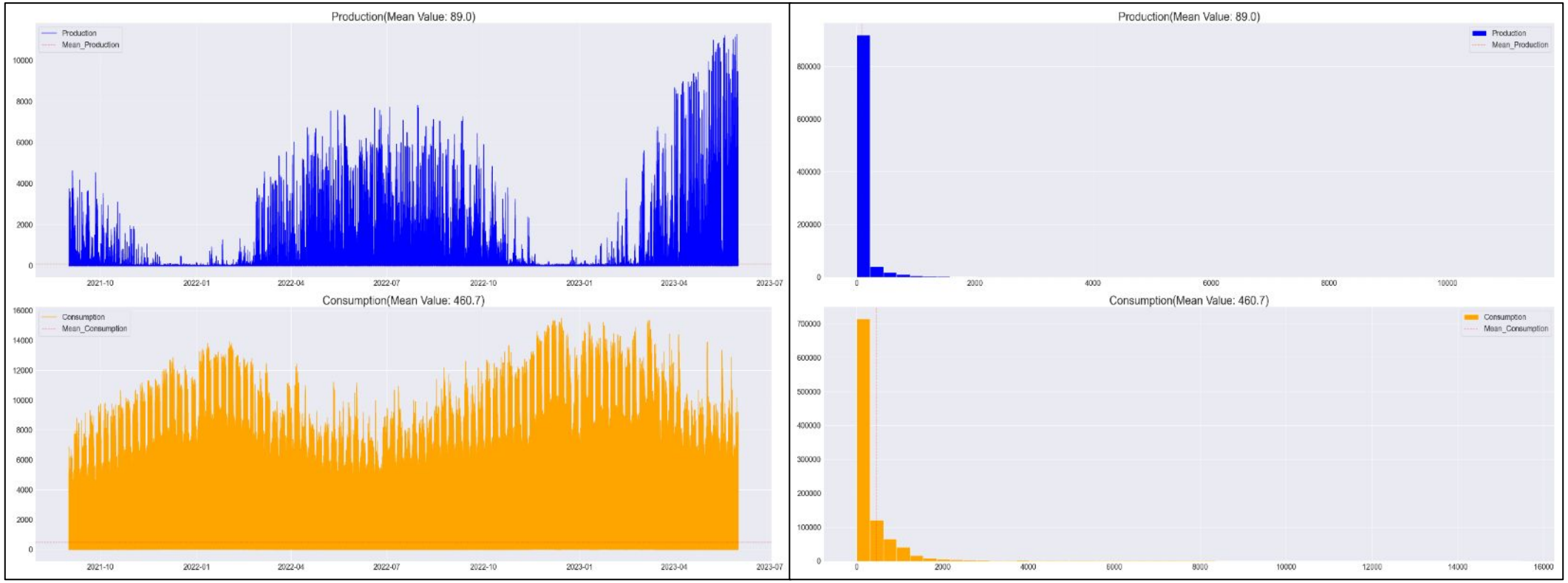




5. EDA

5.1 전력 생산량과 소비량 추이

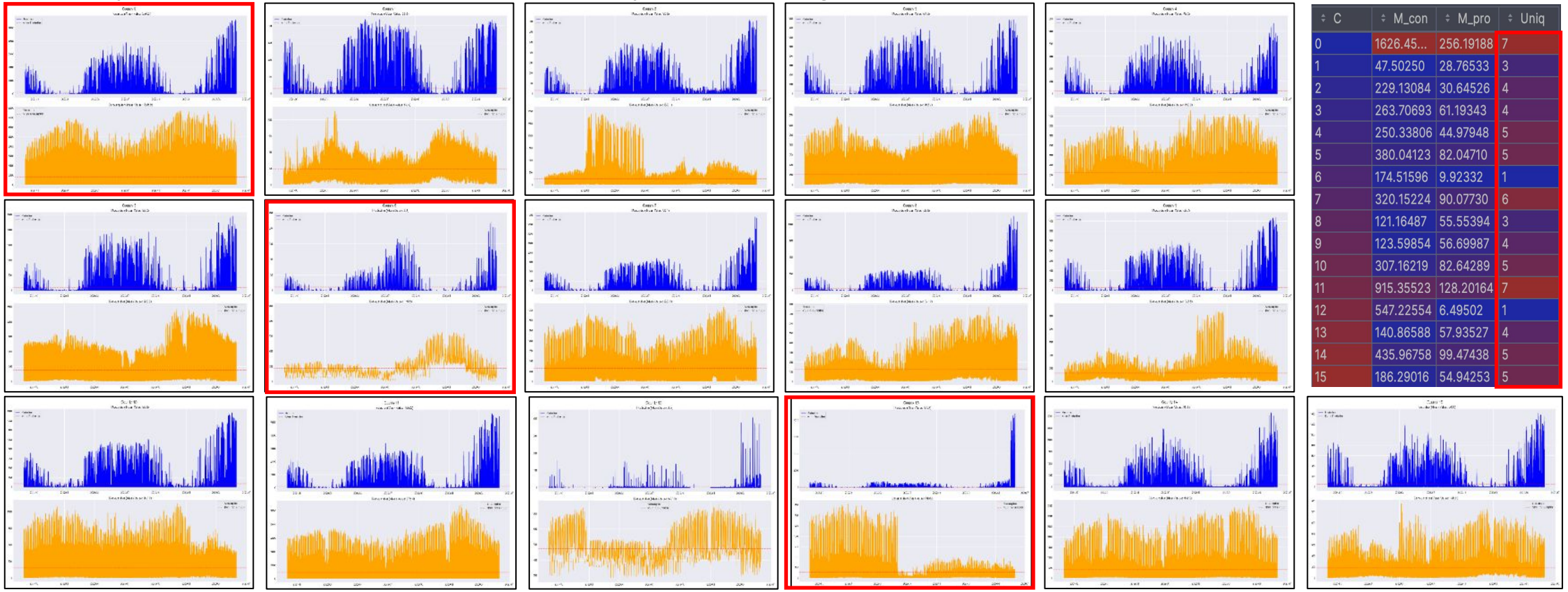
- 전력 생산량과 소비량이 일자별로 변동성이 존재하며, 현재 날짜로 갈수록 증가함



5. EDA

5.2 County별 Segment개수, 평균 전력 생산량, 평균 전력 소비량

- County마다 생산량과 소비량이 다르며, County에 포함된 Segment 수량도 다름

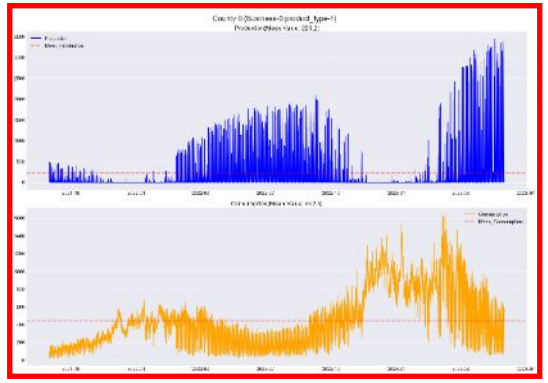
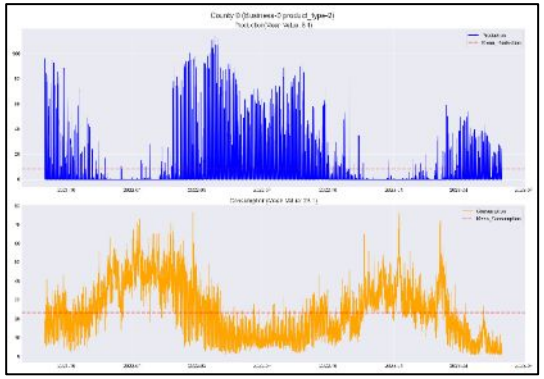
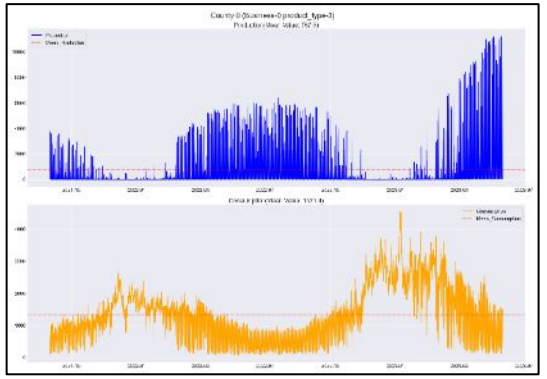


C	M_con	M_pro	Uniq
0	1626.45...	256.19188	7
1	47.50250	28.76533	3
2	229.13084	30.64526	4
3	263.70693	61.19343	4
4	250.33806	44.97948	5
5	380.04123	82.04710	5
6	174.51596	9.92332	1
7	320.15224	90.07730	6
8	121.16487	55.55394	3
9	123.59854	56.69987	4
10	307.16219	82.64289	5
11	915.35523	128.20164	7
12	547.22554	6.49502	1
13	140.86588	57.93527	4
14	435.96758	99.47438	5
15	186.29016	54.94253	5

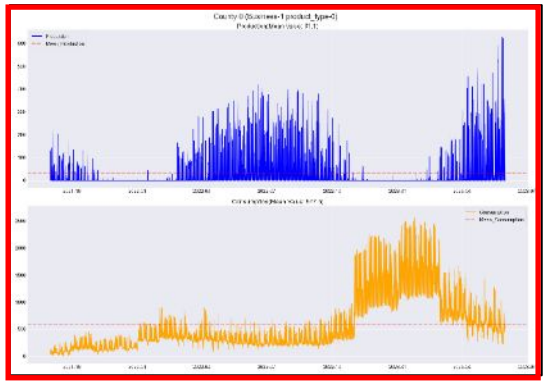
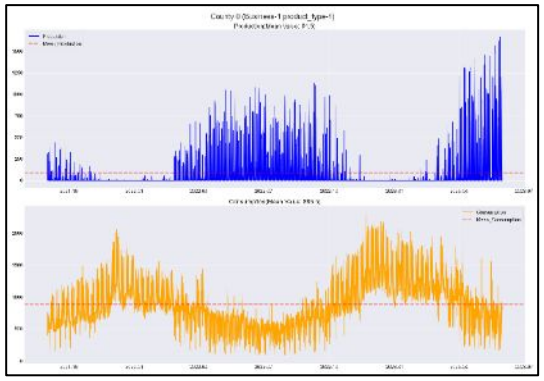
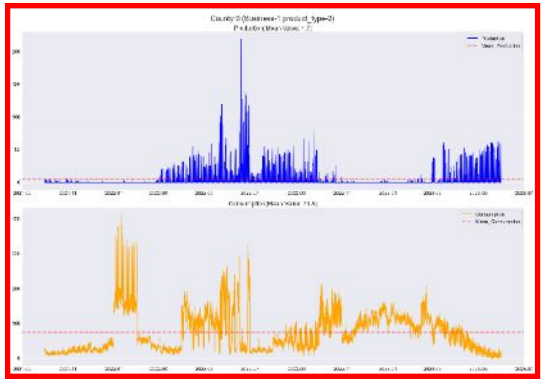
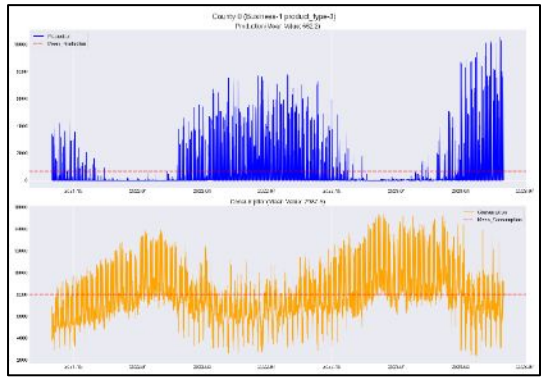
5. EDA

5.3 0번 County의 Segment별 평균 전력 생산량, 평균 전력 소비량

- Segment마다 생산량과 소비량이 다름

Id	M_con	M_pro
b0-p0	nan	nan
b0-p1	442.32652	223.15992
b0-p2	23.10970	8.36996
b0-p3	1321.375...	767.50087
b1-p0	577.54720	31.09730
b1-p1	886.50634	84.54514
b1-p2	73.81563	4.65696
b1-p3	7987.50...	662.18228

5. EDA

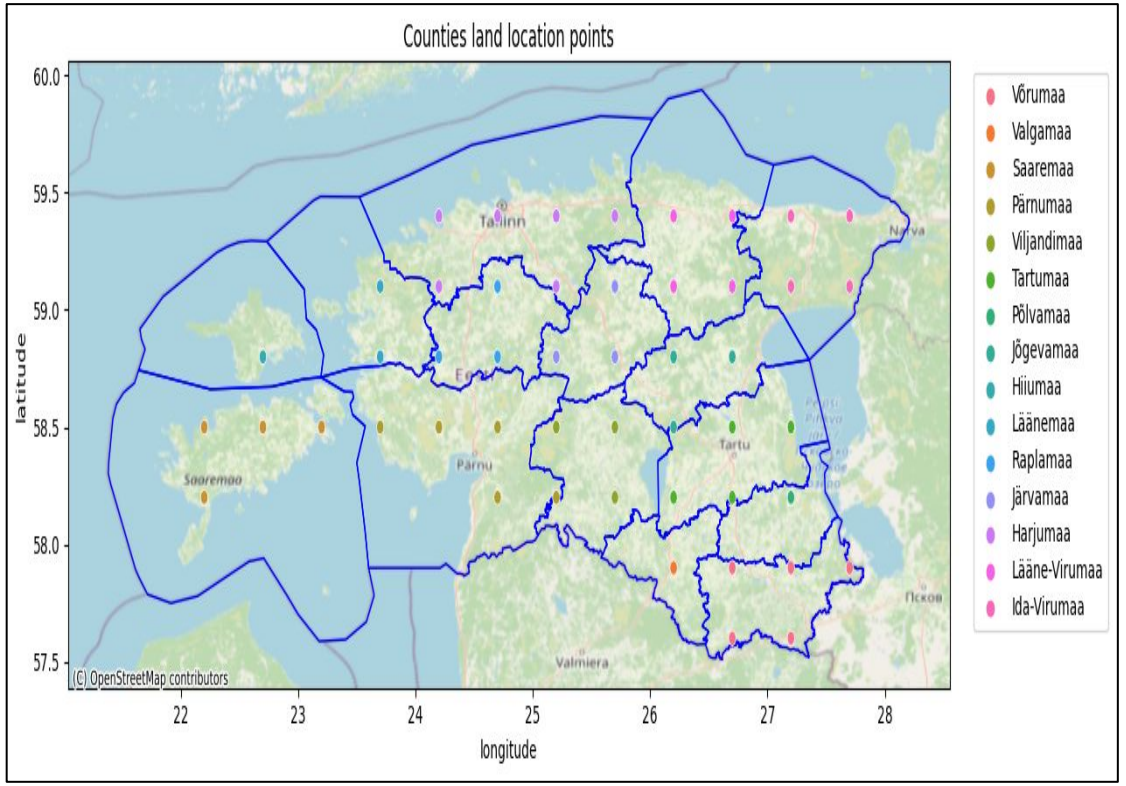
5.4 특정 segment들의 생산량과 소비량

- 특정 Segment의 경우 데이터가 누락되거나 이상치가 존재함

5. EDA

5.5 County 별 지역 정보

- County 12를 제외한 각 County에 속한 모든 Station이 경도와 위도에 따라 표시되고 있음



5. EDA

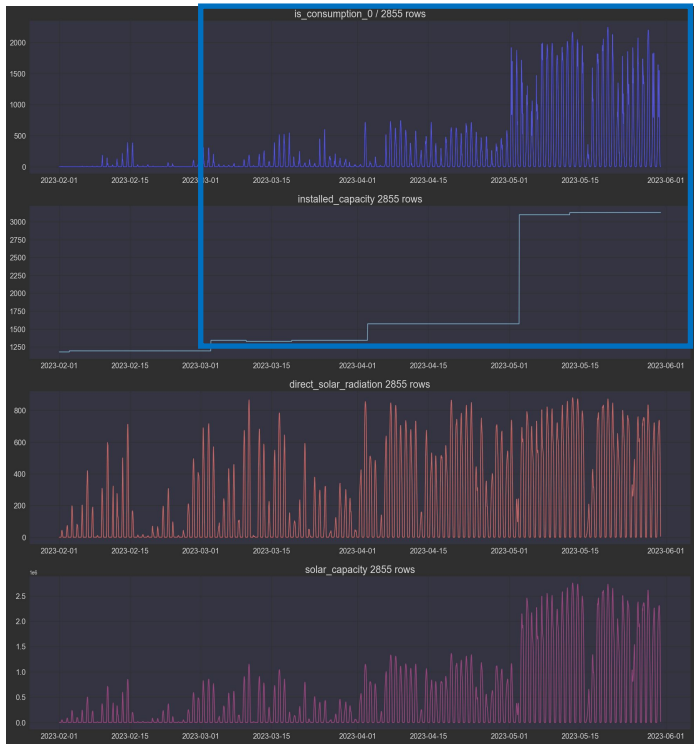
5.6 연간 전력 생산량과 소비량

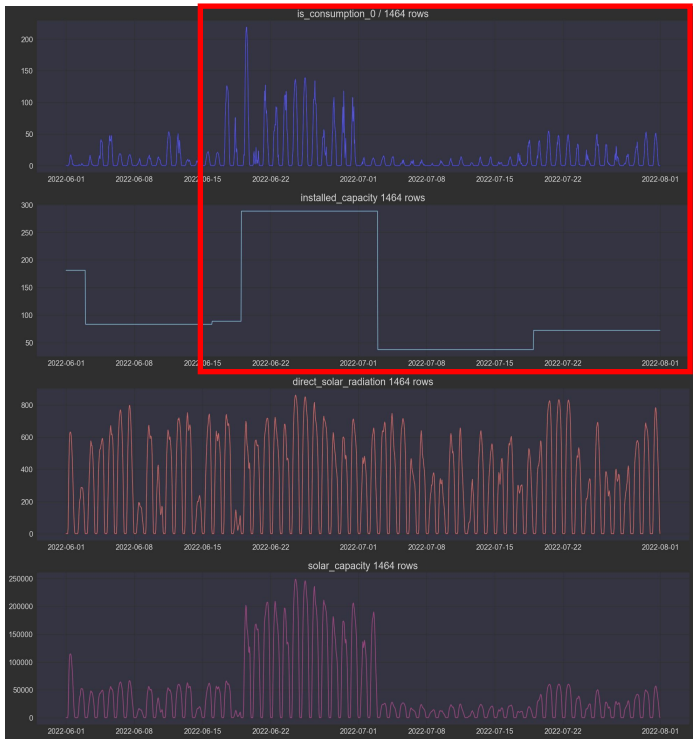
- 생산량의 경우 여름철과 겨울철 간의 차이가 매우 크며, 소비량도 생산량의 수량 만큼 차이가 존재하고 있음

5. EDA

5.7 Segment의 Installed_capacity 증가/감소 에 따른 전력량

- installed_capacity의 증감에 따라 생산되는 전력량의 증가와 감소가 있음

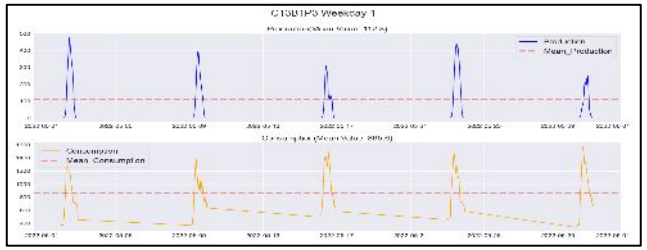


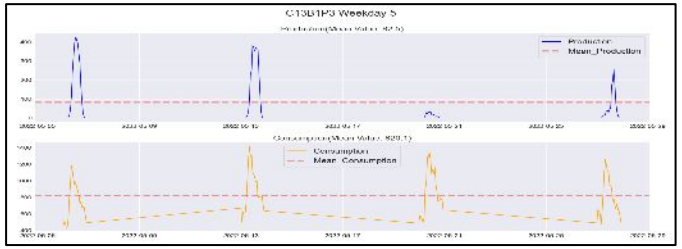


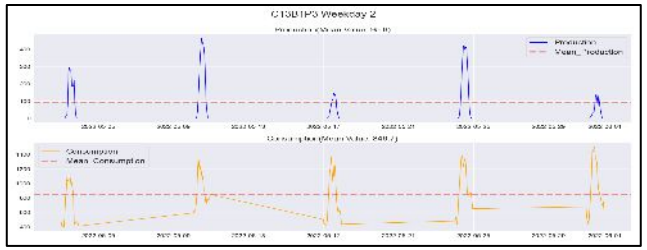
5. EDA

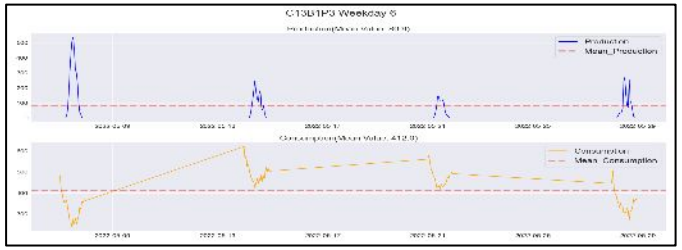
5.8 요일별 생산량과 소비량

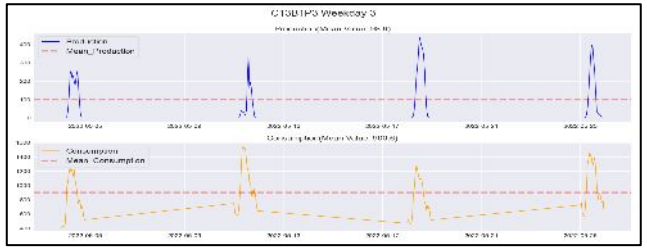
- 소비량의 경우 주말에 사용량이 급격히 낮아지나, 생산량의 경우 요일별 패턴이 나타나지 않으며, 날씨와 상관 관계가 있는 것으로 예상됨

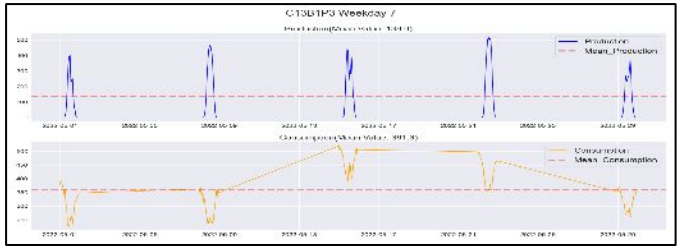


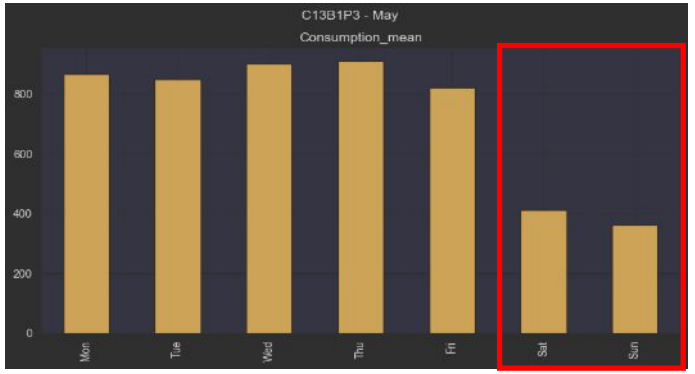












6. Feature Engineering

- 평균 weather (date, local)

- 전국날씨와 county 별로 날씨가 상이할 수 있기 때문에 전국 날씨와 County 날씨를 별도로 생성

- forecast_date : 에스토니아 전국 예측 날씨의 평균 정보
- forecast_local : county 별 예측 날씨의 평균 정보
- historical_date : 에스토니아 전국 과거 날씨의 평균 정보
- historical_local : county 별 과거 날씨의 평균 정보

```
df_forecast_date = (
    df_forecast
    .group_by("datetime").mean()
    .drop("county")
)
```

```
df_forecast_local = (
    df_forecast
    .filter(pl.col("county").is_not_null())
    .group_by("county", "datetime").mean()
)
```

```
df_historical_date = (
    df_historical
    .group_by("datetime").mean()
    .drop("county")
)
```

```
df_historical_local = (
    df_historical
    .filter(pl.col("county").is_not_null())
    .group_by("county", "datetime").mean()
)
```

6. Feature Engineering

- 과거의 날씨 정보

- 예측하는 날의 기상정보와 과거의 기상정보가 관련이 있을 수 있다고 추론함

- 전국 날씨의 2일전과 7일전 정보
- County별 날씨의 2일전과 7일전 정보

```
.join(df_forecast_date.with_columns(pl.col("datetime") + pl.duration(days=2)), on="datetime", how="left", suffix="_fd2")
.join(df_forecast_local.with_columns(pl.col("datetime") + pl.duration(days=2)), on=["county", "datetime"], how="left", suffix="_fl2")
.join(df_historical_date.with_columns(pl.col("datetime") + pl.duration(days=2)), on="datetime", how="left", suffix="_hd2")
.join(df_historical_local.with_columns(pl.col("datetime") + pl.duration(days=2)), on=["county", "datetime"], how="left", suffix="_hl2")

.join(df_forecast_date.with_columns(pl.col("datetime") + pl.duration(days=7)), on="datetime", how="left", suffix="_fd7")
.join(df_forecast_local.with_columns(pl.col("datetime") + pl.duration(days=7)), on=["county", "datetime"], how="left", suffix="_fl7")
.join(df_historical_date.with_columns(pl.col("datetime") + pl.duration(days=7)), on="datetime", how="left", suffix="_hd7")
.join(df_historical_local.with_columns(pl.col("datetime") + pl.duration(days=7)), on=["county", "datetime"], how="left", suffix="_hl7")
```

6. Feature Engineering

- 과거의 가스, 전기 가격

- 예측하는 날의 가스와 전기 가격이 과거의 가격 정보와 관련이 있을 수 있다고 추론함

- 가스 가격의 2일전과 7일전 정보
- 전기 가격의 2일전과 7일전 정보

```
.join(df_gas.with_columns(pl.col("date") + pl.duration(days=2)).rename({"lowest_price_per_mwh": "lowest_price_per_mwh_2_days_ago", "highest_price_per_mwh" : "hi
.join(df_gas.with_columns(pl.col("date") + pl.duration(days=7)).rename({"lowest_price_per_mwh": "lowest_price_per_mwh_7_days_ago", "highest_price_per_mwh" : "hi

.join(df_electricity.with_columns(pl.col("datetime") + pl.duration(days=2)).rename({"euros_per_mwh": "euros_per_mwh_2_days_ago"}), on=["datetime"], how="left")
.join(df_electricity.with_columns(pl.col("datetime") + pl.duration(days=7)).rename({"euros_per_mwh": "euros_per_mwh_7_days_ago"}), on=["datetime"], how="left")
```

6. Feature Engineering

- Lagged target

- Target(전기량)의 값이 7일 마다 주기성이 있음

- Target의 2일, 3일, 4일, 5일, 6일, 7일, 14일전 정보

```
.join(df_target.with_columns(pl.col("datetime") + pl.duration(days=2)).rename({"target": "target_2_days_ago"}), on=["county", "is_business", "product_type", "is_consumption", "datetime"], how="left")
.join(df_target.with_columns(pl.col("datetime") + pl.duration(days=3)).rename({"target": "target_3_days_ago"}), on=["county", "is_business", "product_type", "is_consumption", "datetime"], how="left")
.join(df_target.with_columns(pl.col("datetime") + pl.duration(days=4)).rename({"target": "target_4_days_ago"}), on=["county", "is_business", "product_type", "is_consumption", "datetime"], how="left")
.join(df_target.with_columns(pl.col("datetime") + pl.duration(days=5)).rename({"target": "target_5_days_ago"}), on=["county", "is_business", "product_type", "is_consumption", "datetime"], how="left")
.join(df_target.with_columns(pl.col("datetime") + pl.duration(days=6)).rename({"target": "target_6_days_ago"}), on=["county", "is_business", "product_type", "is_consumption", "datetime"], how="left")
.join(df_target.with_columns(pl.col("datetime") + pl.duration(days=7)).rename({"target": "target_7_days_ago"}), on=["county", "is_business", "product_type", "is_consumption", "datetime"], how="left")
.join(df_target.with_columns(pl.col("datetime") + pl.duration(days=14)).rename({"target": "target_14_days_ago"}), on=["county", "is_business", "product_type", "is_consumption", "datetime"], how="left")
```

6. Feature Engineering

- installed_capacity, eic_count ratio

- installed_capacity과 eic_count 값이 target에 높은 상관관계가 있다고 판단하여 과거 target 대비 installed_capacity, eic_count 비율을 변수로 생성

- installed_capacity의 2일, 3일, 4일, 5일, 6일, 7일, 14일 비율 정보
- eic_count 의 2일, 3일, 4일, 5일, 6일, 7일, 14일 비율 정보

```
for day in [2, 3, 4, 5, 6, 7, 14]:
    df[f'target_{day}_days_ago_capacity_ratio'] = df[f'target_{day}_days_ago'] / df['installed_capacity']
    df[f'target_{day}_days_ago_eic_count_ratio'] = df[f'target_{day}_days_ago'] / df['eic_count']
```

6. Feature Engineering

- weekend

- Target(전기량)의 소비량이 평일과 주말 간에 차이가 있음

  - 평일과 주말의 구분 정보

    (6보다 작으면 평일[1] 크면 주말[0] – 월요일[1] ~ 일요일[7])

```
def is_weekend(day_of_week):
    return 1 if day_of_week <= 5 else 0
```

6. Feature Engineering

- 주기 데이터

- 시계열 데이터에는 계절적인 패턴이 존재하므로 주기데이터를 이용해 성능 향상시킬 수 있다고 판단

- Year의 sin, cos 정보
- Month의 sin, cos 정보
- Weekday의 sin, cos 정보
- Hour 의 sin, cos 정보

```
.with_columns(
    pl.col("weekday").apply(is_weekend).alias("isweekday"),
    (np.pi * pl.col("dayofyear") / 182.5).sin().alias("sin(dayofyear)"),
    (np.pi * pl.col("dayofyear") / 182.5).cos().alias("cos(dayofyear)"),
    (np.pi * pl.col("weekday") / 3.5).sin().alias("sin(weekday)"),
    (np.pi * pl.col("weekday") / 3.5).cos().alias("cos(weekday)"),
    (np.pi * pl.col("month") / 6).sin().alias("sin(month)"),
    (np.pi * pl.col("month") / 6).cos().alias("cos(month)"),
    (np.pi * pl.col("hour") / 12).sin().alias("sin(hour)"),
    (np.pi * pl.col("hour") / 12).cos().alias("cos(hour)"),
)
```

6. Feature Engineering

- Rolling statistics

- Target의 추세와 계절성 같은 패턴과 이상치를 파악하기 위한 이동 통계 feature 생성

- Target의 2일, 3일, 4일, 5일, 6일, 7일, 14일 평균
- Target의 2일, 3일, 4일, 5일, 6일, 7일, 14일 표준편차

```
df["target_mean"] = df[[f"target_{i}_days_ago" for i in [2, 3, 4, 5, 6, 7, 14]]].mean(1)
df["target_std"] = df[[f"target_{i}_days_ago" for i in [2, 3, 4, 5, 6, 7, 14]]].std(1)
```

7. Model

7.1 Segment별 생산과 소비에 따른 개별 모델 생성 (1/2)

- County, Business, product, consumption로 구분된 데이터셋을 이용하여 LGBM과 Best 하이퍼파라메터를 이용하여 모델들을 만들어 냄

- fit_model(df, parameter)

  - 데이터셋과 파라메터를 입력받아 LGBM 모델을 생성하고 학습을 시킨후 학습된 모델을 넘겨줌

- Execute_func_per_users(func, df, params=None)

  - 반복작업을 수행하여 모든 Segment별 생산과 소비에 관련된 모델을 학습시킨 후 학습된 모델을 넘겨줌

  1) func(fit_model)과 전체 데이터프레임과 하이퍼파라메터를 입력받음

  2) 하나의 Segment에 대해서 생산과 소비로 구분된 데이터프레임을 추출함

  3) fit_model 함수에 추출된 데이터프레임과 파라메터를 넘겨주고 모델을 학습시킴

  4) 2)와 3)을 모든 Segment에 대해서 수행함

  5) 모든 훈련된 모델을 넘겨줌

7. Model

7.1 Segment별 생산과 소비에 따른 개별 모델(136개) 생성 (2/2)

```python
def execute_func_per_users(func, df, params=None):
    result_dict = {}
    if params==None:
        counties = pd.unique(df['county'])
        products = pd.unique(df['product_type'])
        is_business = pd.unique(df['is_business'])
    else:
        counties = params['county']
        products = params['products']
        is_business = params['business']
    for c in counties:
        for p in products:
            for b in is_business:
                user_df = df[(df['county']==c)&(df['product_type']==p)&(df['is_business']==b)]
                if user_df.shape[0]==0:continue
                con_0 = (user_df[user_df['is_consumption']==0].drop(columns=["target"]))
                con_1 = (user_df[user_df['is_consumption']==1].drop(columns=["target"]))
                model_0 = func(con_0)
                model_1 = func(con_1)
                result_dict[f'c{c}p{p}b{b}'] = {'model_0':model_0, 'model_1':model_1}
                print(f'c{c}p{p}b{b}')
    return result_dict
```

```
p1={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.05073909898961407,
 'colsample_bytree': 0.726023996436955, 'colsample_bynode': 0.5803681307354022, 'lambda_l1': 8.562963348932286,
 'lambda_l2': 4.893256185259296, 'min_data_in_leaf': 115, 'max_depth': 23, 'max_bin': 898}
```

```python
def fit_model(df, parameter=p1):
    model = lgb.LGBMRegressor(**parameter, random_state=42)
    model.fit(
        X=df.drop(columns=["target"]),
        y=df["target"])
    return model
```

```python
models = execute_func_per_users(fit_model, df_train)
```

7. Model

7.2 Voting regressor를 이용한 Ensemble 모델 생성

- 8개의 LGBM모델에 8개의 최적화된 하이퍼파라메터를 이용하여 모델을 학습시킴

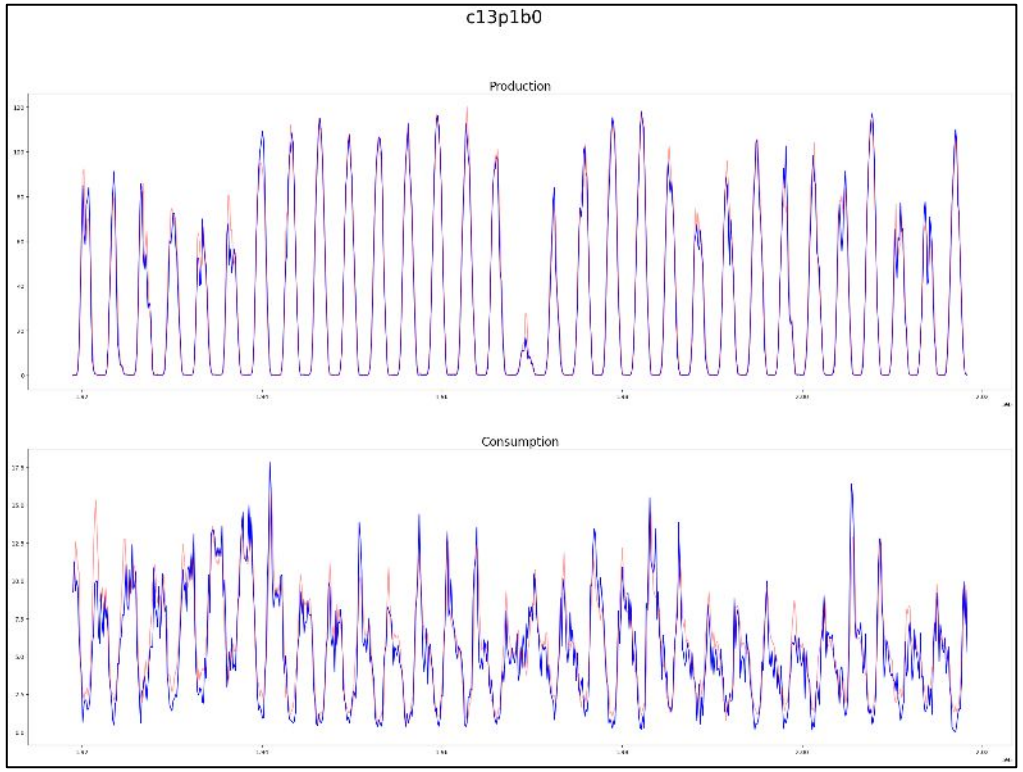
```
model = VotingRegressor([
    ('lgb_1', lgb.LGBMRegressor(**p1, random_state=42)),
    ('lgb_2', lgb.LGBMRegressor(**p2, random_state=42)),
    ('lgb_3', lgb.LGBMRegressor(**p3, random_state=42)),
    ('lgb_4', lgb.LGBMRegressor(**p4, random_state=42)),
    ('lgb_5', lgb.LGBMRegressor(**p5, random_state=42)),
    ('lgb_6', lgb.LGBMRegressor(**p6, random_state=42)),
    ('lgb_7', lgb.LGBMRegressor(**p7, random_state=42)),
    ('lgb_8', lgb.LGBMRegressor(**p8, random_state=42)),
],weights=[0.16,0.13,0.12,0.11,0.12,0.11,0.14,0.11])

model.fit(
    X=df_train.drop(columns=["target"]),
    y=df_train["target"]
)
```

```
p1={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.05073909898961407,
 'colsample_bytree': 0.726023996436955, 'colsample_bynode': 0.5803681307354022, 'lambda_l1': 8.562963348932286,
 'lambda_l2': 4.893256185259296, 'min_data_in_leaf': 115, 'max_depth': 23, 'max_bin': 898}
p2={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.05670084478292278,
 'colsample_bytree': 0.6440444070196796, 'colsample_bynode': 0.637635804565811, 'lambda_l1': 6.29090474401462,
 'lambda_l2': 6.775341543233317, 'min_data_in_leaf': 95, 'max_depth': 9, 'max_bin': 630}
p3={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.0632167263149817,
 'colsample_bytree': 0.6958033941948067, 'colsample_bynode': 0.6030801666196094, 'lambda_l1': 7.137580620471935,
 'lambda_l2': 9.348169401713742, 'min_data_in_leaf': 74, 'max_depth': 11, 'max_bin': 530}
p4={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.061236402165228264,
 'colsample_bytree': 0.81427095118471, 'colsample_bynode': 0.6097376843527067, 'lambda_l1': 6.360490880385201,
 'lambda_l2': 9.954136008333839, 'min_data_in_leaf': 238, 'max_depth': 16, 'max_bin': 649}
p5={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.08753282378023663,
 'colsample_bytree': 0.7508715107428325, 'colsample_bynode': 0.6831819500325418, 'lambda_l1': 8.679353563755722,
 'lambda_l2': 6.105008696961338, 'min_data_in_leaf': 198, 'max_depth': 15, 'max_bin': 835}
p6={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.05929380742257108,
 'colsample_bytree': 0.6101576947777211, 'colsample_bynode': 0.6052639518604396, 'lambda_l1': 8.087311995794915,
 'lambda_l2': 6.067361158677095, 'min_data_in_leaf': 122, 'max_depth': 9, 'max_bin': 797}
p7={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.05689066836106983,
 'colsample_bytree': 0.8915976762048253, 'colsample_bynode': 0.5942203285139224, 'lambda_l1': 7.6277555139102864,
 'lambda_l2': 6.6591278779517808, 'min_data_in_leaf' : 156, 'max_depth': 11, 'max_bin': 813}
p8={'n_iter': 2000,'verbose': -1,'objective': 'l2','metric': 'mae','learning_rate': 0.06210133914728566,
 'colsample_bytree': 0.9394149364406023, 'colsample_bynode': 0.6136449922460668, 'lambda_l1': 6.8170120783290963,
 'lambda_l2': 6.941392509816262S, 'min_data_in_leaf': 100, 'max_depth': 12, 'max_bin': 749}
```

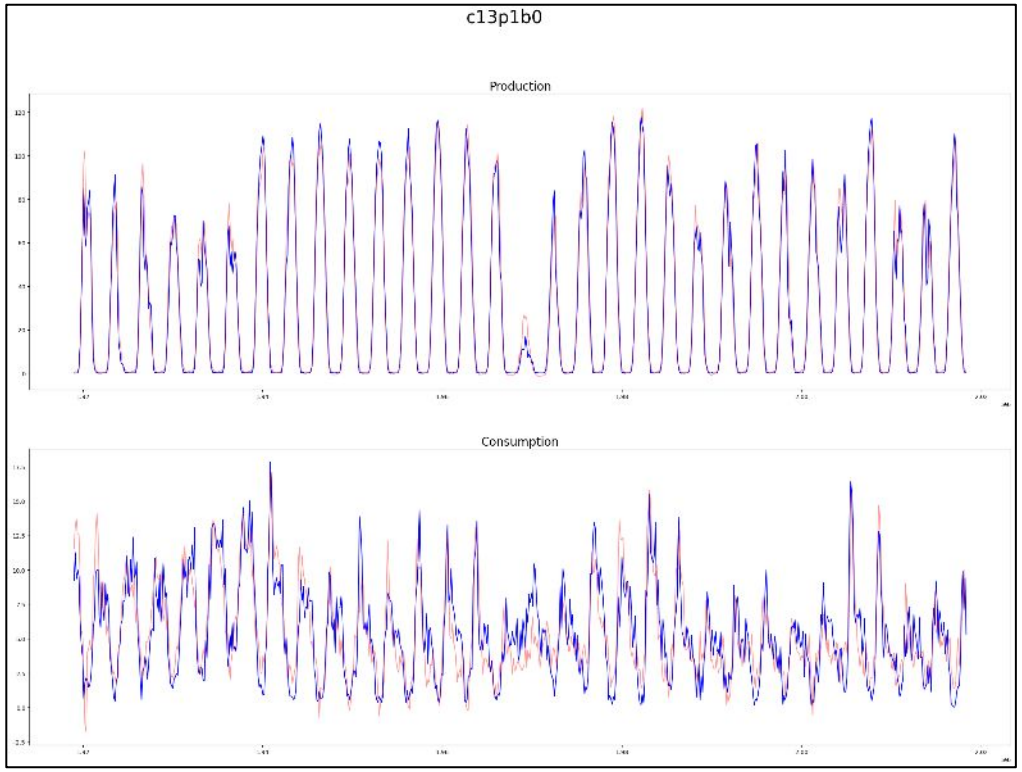
7. Model

7.3 각각의 모델 예측값 확인

개별 RGBM 모델 예측값



Voting 모델 예측값



7. Model

7.4 MAE 결과값

- County별 LGBM 모델을 학습할 경우 모델마다 MAE 결과값의 차이가 많이 나타남

County별 RGBM 모델

Voting 모델

Train(95%)

	mae_0	mae_1	mae_mean
0	189.77931	41.56843	115.67387
1	26.61089	2.36717	14.48903
2	30.03978	8.45656	19.24817
3	59.95147	8.65811	34.30479
4	50.77046	16.21294	33.49170
5	94.71215	16.69950	55.70583
6	79.09921	4.86955	41.98438
7	99.23200	19.99436	59.61318
8	48.61094	4.47858	26.54476
9	45.00158	6.12642	25.56400
10	56.18615	10.07074	33.12844
11	92.27594	29.23385	60.75490
12	371.89986	9.72039	190.81013
13	36.50008	6.21627	21.35818
14	98.33485	15.98758	57.16122
15	47.78380	11.24437	29.51409
mean	89.17428	13.24405	51.20917

Test(5%)

	mae_0	mae_1	mae_mean
0	563.59338	149.81165	356.70252
1	52.39001	5.06167	28.72584
2	46.21373	20.04132	33.12752
3	136.55461	25.77560	81.16511
4	101.81574	42.58833	72.20204
5	173.84637	47.67044	110.75841
6	59.33409	26.91822	43.12616
7	263.94750	67.73587	165.84168
8	204.85910	22.58773	113.72341
9	112.16283	15.57541	63.86912
10	168.48001	31.64031	100.06016
11	242.82953	102.91423	172.87188
12	352.83736	67.86501	210.35119
13	345.88041	17.55535	181.71788
14	144.13631	47.06650	95.60140
15	97.79599	31.87738	64.83669
mean	191.66731	45.16781	118.41756

```
train(95%) :  21.368711891470596
test(5%) :  61.703621777675956
my_best_parmams
CPU times: user 3min 58s, sys: 5.1 s, total: 4min 3s
Wall time: 48.3 s
```

성능이 낮은 개별 모델들에 대한 개선을 통해서 전체적인 성능을 향상시키는 것도 성능향상 방법으로 예상함

8. Hyper-parameter Tuning

7.1 Hyper-parameter (최종)

- Public Score에 좋은 성적을 나타내는 MAE가 61 ~ 64까지 분포하는 하이퍼파라메터를 사용함

name	p1	p2	p3	p4	p5	p6
n_iter	1500	1500	1500	1500	1500	1500
verbose	-1	-1	-1	-1	-1	-1
objective	l2	l2	l2	l2	l2	l2
learning_rate	0.056700845	0.059336649	0.061236402	0.087532824	0.053339669	0.056890668
colsample_bytree	0.644044407	0.994697399	0.814270951	0.750871511	0.899214956	0.891597676
colsample_bynode	0.637635805	0.522615629	0.609737684	0.68318195	0.572272354	0.594220329
lambda_l1	6.290904744	3.935758192	6.36049088	8.679353564	9.773166365	3.627755514
lambda_l2	6.775341543	4.632340933	9.954136008	6.105008697	1.40377015	1.659127878
min_data_in_leaf	95	186	238	198	222	186
max_depth	9	9	16	15	8	9
max_bin	630	912	649	835	1012	813
MAE	**62.768**	**62.937**	**62.294**	**64.004**	**61.710**	**63.223**

8. Hyper-parameter Tuning

7.2 Tuning 결과

- LGBM이 Catboost보다 더 좋은 성능을 나타냄. 하이퍼파라미터의 경우 Local에서 좋은 성능을 나타내나 PS에서 성능이 낮아지는 경우가 자주 존재함. Feature 추가에 따른 성능 증감이 있음.

Model	차수	Public Score	P1	P2	P3	P4	P5	P6	증감	비고
Catboost	1차	**94.54**							**0**	EDA + Base Model
Catboost	2차	**89.09**							**-5.45**	LGBM 대비 성능 차이가 많음
LGBM + Voting	1차	**69.83**	62.768	63.159	62.294	64.004	65.382	63.223	**0**	Base Model
LGBM + Voting	2차	**69.90**	62.768	62.070	62.294	64.004	65.382	63.223	**0.07**	
LGBM + Voting	3차	**69.80**	62.768	62.937	62.294	64.004	65.382	63.223	**-0.03**	
LGBM + Voting	4차	**69.73**	62.768	62.937	62.294	64.004	61.710	63.223	**-0.10**	하이퍼파라메터 변경 후 PB 성능 향상
LGBM + Voting	5차	**69.96**	62.768	62.937	62.294	62.371	61.710	63.223	**0.13**	하이퍼파라메터 변경 후 PB 성능 저하
LGBM + Voting	6차	**70.27**	62.768	62.937	62.294	62.371	61.710	63.223	**0.44**	VotingRegressor에 성능 좋은 3개의 하이퍼파라미터 적용 후 PB 성능 저하
LGBM + Voting	7차	**69.92**	62.768	62.937	62.294	61.837	61.710	63.223	**0.09**	
LGBM + Voting	8차	**69.89**	62.768	62.937	62.294	64.004	61.710	61.837	**0.06**	
LGBM + Voting	9차	**69.93**	62.768	62.937	62.294	62.228	61.710	63.223	**0.10**	
LGBM + Voting	10차	**69.91**	62.768	62.937	62.294	64.004	61.710	63.223	**0.08**	4차와 동일 하이퍼파라미터 사용하였으나, 다른 결과 나옴
LGBM + Voting	11차	**69.63**	62.768	62.937	62.294	64.004	61.710	63.223	**-0.20**	weekend 추가 후 성능 향상
LGBM + Voting	12차	**69.67**	62.768	62.937	62.294	64.004	61.710	63.223	**-0.16**	week, installed_target 추가
LGBM + Voting	13차	**69.81**	62.768	62.937	62.294	64.004	61.710	63.223	**-0.02**	week_weight 추가
LGBM + Voting	14차	**71.21**	62.768	62.937	62.294	64.004	61.710	63.223	**1.38**	3개의 VotingRegressor를 적용

9. Conclusion

Leaderboard	▪ 김창희 : 순위 XX, OO.OO점 (동메달) ▪ 권혁찬 : 순위 XX, OO.OO점 (동메달) ▪ 문정의 : 순위 148, 69.62점
Features	▪ 총 134 ~ XXX 개
Model	▪ 6개(또는 8개)의 하이퍼파라미터를 이용하여 LGBMRegressor 6개(또는 8개) 모델을 학습 ▪ VotingRegressor 사용하여 전체 데이터로 학습하는 모델 ▪ VotingRegressor 사용하여 전력 생산 데이터로 학습하는 모델 (is_consumption == 0)
Hyper-parameter	▪ 6개(또는 8개)의 각기 다른 Best Hyper-parameter

10. Lesson Learn

EDA	▪ 최적의 하이퍼파라미터를 적용 후 더 이상 성능 향상이 없을 경우 EDA를 다시 수행해야 함
Feature Engineering	▪ 성능 향상이 없을 경우 새로운 Feature 추출하고 적용시 성능 향상이 있었음
Model	▪ LGBM을 사용하는 것이 Catboost를 사용하는 것보다 성능 향상이 우수함 ▪ VotingRegressor를 사용하는 것이 단일 Model을 사용한 것보다 성능 향상이 있었음
Hyper-parameter	▪ 최적의 하이퍼파라미터를 적용하는 것만으로도 Public Score를 향상시킬 수 있음을 확인함 ▪ 학습 결과 더 좋은 하이퍼파라미터가 public score에는 반영되지 않는 경우가 종종 발생함