

# 가스공급량 수요예측 모델개발

KUBIG CONFERENCE

분류&예측 프로젝트 1팀

12기 전지우 13기 김현지 전보민 14기 김유민 임혜리

# 00

목차

CONTENTS

01.

대회 및 데이터 소개

CONTENTS

02.

EDA 및 전처리

CONTENTS

03.

Feature  
Engineering

CONTENTS

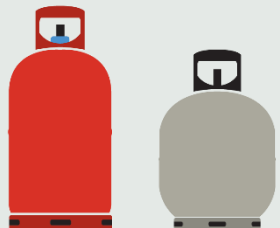
04.

Modeling

CONTENTS

05.

최종결과/의의 및 한계



# 01

## 대회 및 데이터 소개

### 참여 대회

## 가스공급량 수요예측 모델개발 대회

### 가스공급량 수요예측 모델개발

한국가스공사 | 스타트업 | 정형데이터 | 수요예측

💰 상금 : 총 3,250만원

🕒 2021.10.11 ~ 2021.12.10 23:00 [+ Google Calendar](#)

👤 488명 📅 마감



참여중

### 대회 목적

한국가스공사의 시간단위 공급량 내부데이터와 기상정보 및 가스 외 발전량 등 외부데이터를 포함한 데이터셋을 구축하여 90일 한도 일간 공급량을 예측하는 인공지능 모델 개발

### 사용 데이터

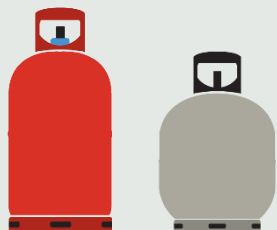
Total data : 2013년 1월 1일 ~ 2018년 12월 31일 가스 공급사 별 시간 별 공급량

Test data : 2019년 1월 1일 ~ 2019년 3월 31일 **‘일자|시간|공급사구분’** 가스공급량 예측

### 평가 방식

NMAE(Normalized Mean Absolute Error)

-정규화 평균 절대 오차 척도를 사용하여 평가



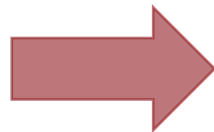
# 01

## 대회 및 데이터 소개

### 변수 설명

기본 제공 데이터에서 연도, 월, 일, 시간, 주말여부 추출

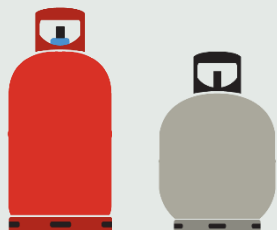
	연월일	시간	구분	공급량
0	2013-01-01	1	A	2497.129
1	2013-01-01	2	A	2363.265
2	2013-01-01	3	A	2258.505
3	2013-01-01	4	A	2243.969
4	2013-01-01	5	A	2344.105
5	2013-01-01	6	A	2390.961
6	2013-01-01	7	A	2378.457
7	2013-01-01	8	A	2518.921
8	2013-01-01	9	A	2706.481
9	2013-01-01	10	A	2832.057



연월일	시간	구분	공급량	year	month	day	weekday
2013-01-01	1	0	2497.129	2013	1	1	1
2013-01-01	2	0	2363.265	2013	1	1	1
2013-01-01	3	0	2258.505	2013	1	1	1
2013-01-01	4	0	2243.969	2013	1	1	1
2013-01-01	5	0	2344.105	2013	1	1	1
...	...	...	...	...	...	...	...
2018-12-31	20	6	681.033	2018	12	31	0
2018-12-31	21	6	669.961	2018	12	31	0
2018-12-31	22	6	657.941	2018	12	31	0
2018-12-31	23	6	610.953	2018	12	31	0
2018-12-31	24	6	560.896	2018	12	31	0



추후 외부데이터(전산업생산지수, 가스기름 상대가격, 가스전기 상대가격, 기온데이터 등) 추가 예정



# 02

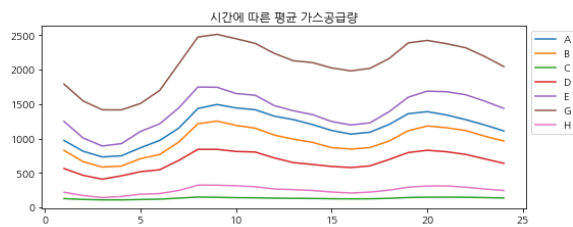
## EDA 및 전처리

### EDA

### EDA를 통해 인사이트 얻기

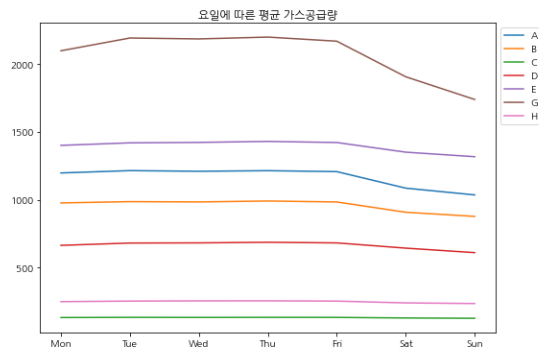
#### # 시간대별 가스공급량

```
[ ] 1 fig = plt.figure(figsize=(8,3))
2 for i in range(len(hour_mean)):
3     plt.plot(hour_mean.iloc[:,i], label=hour_mean.index[i])
4     plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left')
5 plt.tight_layout()
6 plt.title('시간에 따른 평균 가스공급량')
7 plt.show()
```



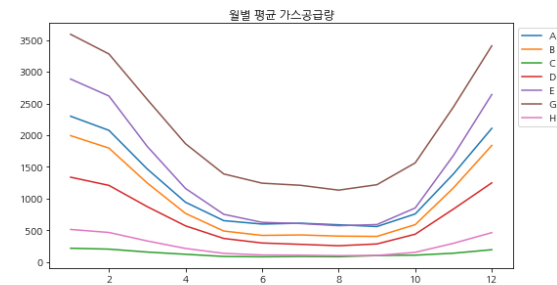
#### # 요일별 가스공급량

```
[ ] 1 fig = plt.figure(figsize=(8,5))
2 for i in range(len(weekday_mean)):
3     plt.plot(weekday_mean.iloc[:,i], label=weekday_mean.index[i])
4     plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left')
5 plt.tight_layout()
6 plt.title('요일에 따른 평균 가스공급량')
7 plt.show()
```



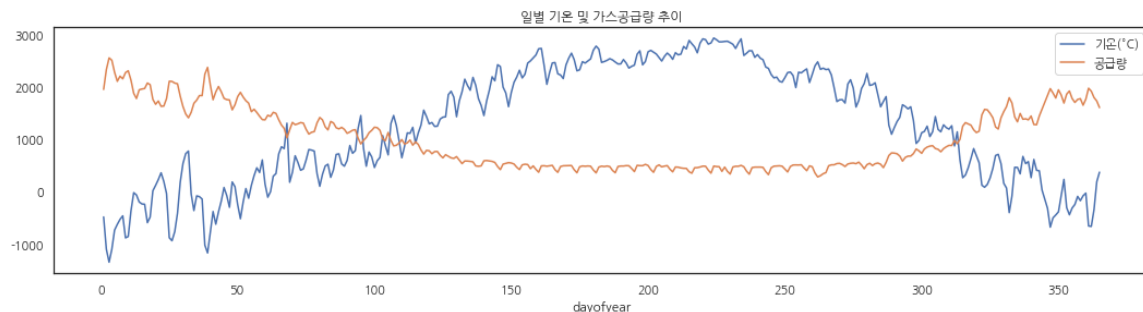
#### # 월별 가스공급량

```
[ ] 1 fig = plt.figure(figsize=(8,4))
2 for i in range(len(month_mean)):
3     plt.plot(month_mean.iloc[:,i], label=month_mean.index[i])
4     plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left')
5 plt.tight_layout()
6 plt.title('월별 평균 가스공급량')
7 plt.show()
```



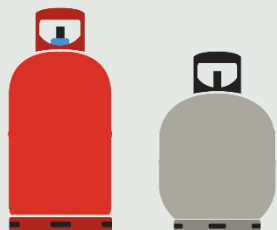
#### # 일별 기온 및 가스공급량 추이

```
[ ] 1 (weather_2013.groupby(['dayofyear'])['기온(°C)'].mean()+100).plot(figsize=(18,4))
2 total_2013.groupby(['dayofyear'])['공급량'].mean().plot(figsize=(18,4))
3 plt.legend(['기온(°C)', '공급량'])
4 plt.title('일별 기온 및 가스공급량 추이')
5 plt.show()
```



#### 가스공급량 변화 특징

- 1) 시간대, 월이 가스공급량에 영향을 미치는 것을 확인
- 2) 시간대와 월에 따라 가장 달라지는 기상 정보는 기온이라 생각되어 기온 정보 추가
- 3) 기온이 낮아질수록 공급량 증가



# 02

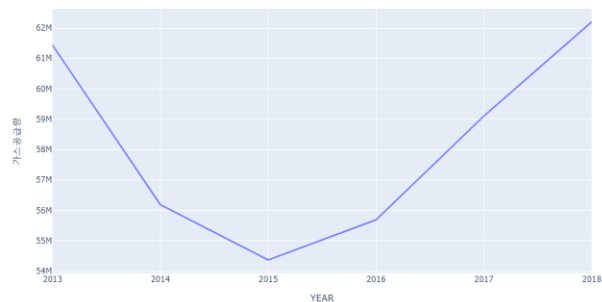
## EDA 및 전처리

### EDA

### EDA를 통해 인사이트 얻기

#### # 연도별 평균 가스공급량

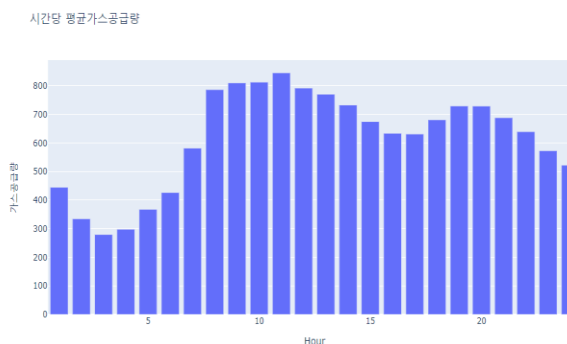
```
[17] 1 # plotting
2 fig = px.line(year,
3             x='year',
4             y='공급량')
5 fig.update_layout(xaxis_title='YEAR',
6                 yaxis_title='가스공급량')
7 fig.show()
```



2015년 이후 도시가스 수요가수가 증가하는 경향

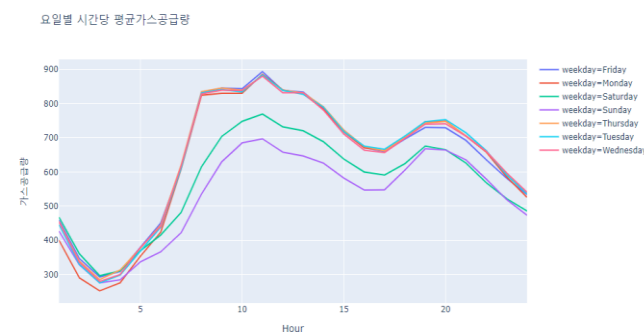
#### # 시간당 평균 가스공급량

```
[11] 1 # plotting
2 fig = px.bar(hour,
3             x='시간',
4             y='공급량',
5             title='시간당 평균가스공급량')
6 fig.update_layout(xaxis_title='Hour',
7                 yaxis_title='가스공급량')
8 fig.show()
```



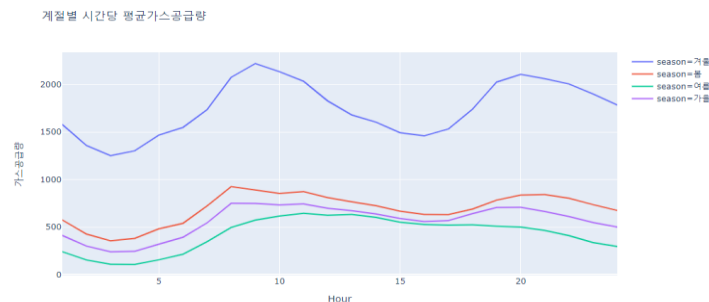
#### # 요일 별 시간당 평균 가스공급량

```
[13] 1 # plotting
2 fig = px.line(hour, weekday,
3             x='시간',
4             y='공급량',
5             color='weekday',
6             title='요일별 시간당 평균가스공급량')
7 fig.update_layout(xaxis_title='Hour',
8                 yaxis_title='가스공급량')
9 fig.show()
```



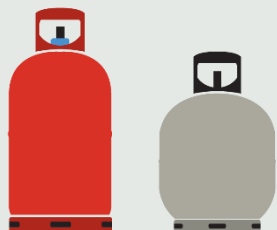
#### # 계절별 시간당 평균 가스공급량

```
1 # plotting
2 fig = px.line(hour, season,
3             x='시간',
4             y='공급량',
5             color='season',
6             title='계절별 시간당 평균가스공급량')
7 fig.update_layout(xaxis_title='Hour',
8                 yaxis_title='가스공급량')
9 fig.show()
```



#### 가스공급량 변화 특징

- 4) 주말보다 평일에 공급량 증가
- 5) 겨울(12~2월)이 다른 계절에 비해 공급량 압도적 많음
- 6) 밤 ~ 새벽 시간대에는 공급량 감소, 오전과 저녁 시간대에 크게 증가



# 02

## EDA 및 전처리

### 전처리

### 월별 + 구분별 이상치 처리

```
[ ] 1 def outliers_iqr(df):
2     quartile_1, quartile_3 = np.percentile(df, [25, 75])
3     iqr = quartile_3 - quartile_1
4     lower_bound = quartile_1 - (iqr * 1.5)
5     upper_bound = quartile_3 + (iqr * 1.5)
6     return lower_bound, upper_bound
```

•  
•  
•

```
[ ] 1 total['이상치처리_공급량'] = total.apply(lambda x: x['lower'] if x['공급량'] < x['lower'] else x['upper'] if x['공급량'] > x['upper'] else x['공급량'], axis=1)
```

```
1 total = total.drop(['lower', 'upper'], axis = 1)
2 total
```

	연월일	시간	구분	공급량	year	month	day	weekday	이상치처리_공급량
0	2013-01-01	1	0	2497.129	2013	1	1	1	2497.129
1	2013-01-01	2	0	2363.265	2013	1	1	1	2363.265
2	2013-01-01	3	0	2258.505	2013	1	1	1	2258.505
3	2013-01-01	4	0	2243.969	2013	1	1	1	2243.969
4	2013-01-01	5	0	2344.105	2013	1	1	1	2344.105
...	...	...	...	...	...	...	...	...	...
368083	2018-12-31	20	6	681.033	2018	12	31	0	681.033
368084	2018-12-31	21	6	669.961	2018	12	31	0	669.961
368085	2018-12-31	22	6	657.941	2018	12	31	0	657.941
368086	2018-12-31	23	6	610.953	2018	12	31	0	610.953
368087	2018-12-31	24	6	560.896	2018	12	31	0	560.896

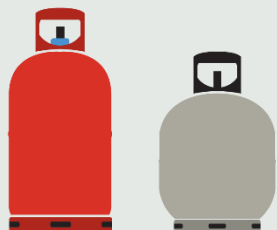
368088 rows × 9 columns



Inter Quartile Range(IQR)을 활용하여  
Q1-1.5\*IQR과 Q3+1.5\*IQR를 벗어난  
값을 이상치로 간주하여 제거



구분 별 학습 시 이상치처리\_공급량을 y값  
으로 두고 학습



# 03

## Feature Engineering

변수 생성

추가 변수 추출

1) 공급량\_평균

2) 공급량\_요일\_평균

3) 전년도\_공급량

4) sin\_time, cos\_time

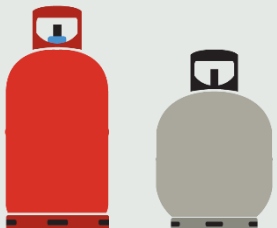
5) 휴일

6) 휴일+주말

Feature 추가

```
[ ] 1 # 공급량_평균
2 test['공급량_평균'] = test.progress_apply(lambda x: supply_mean.loc[(supply_mean.구분 == x['구분']) & (supply_mean.month == x['month']) & (supply_mean.day == x['day']) & (supply_mean.시간 == x['시간']), '공급량'].values[0], axis = 1)
3
4 # 공급량_요일_평균
5 test['공급량_요일_평균'] = test.progress_apply(lambda x: supply_weekday_mean.loc[(supply_weekday_mean.구분 == x['구분']) & (supply_weekday_mean.month == x['month']) & (supply_weekday_mean.weekday == x['weekday'])
6 & (supply_weekday_mean.시간 == x['시간']), '공급량'].values[0], axis = 1)
7
8 # 전년도_공급량
9 total_2018 = total[total.year == 2018]
10 test['전년도_공급량'] = test.progress_apply(lambda x: total_2018.loc[(total_2018.구분 == x['구분']) & (total_2018.month == x['month']) & (total_2018.day == x['day']) & (total_2018.시간 == x['시간']), '공급량'].values[0], axis = 1)
11
12 # sin_time, cos_time
13 test['sin_time'] = np.sin(2*np.pi*total.시간/24)
14 test['cos_time'] = np.cos(2*np.pi*total.시간/24)
15
```

```
49 # 휴일
50 ## 2019년 공휴일
51 holidays_2019 = pd.concat([pd.Series(np.array(SouthKorea()).holidays(2019))[1:, 0]), reset_index(drop=True)]
52 test['휴일'] = test['일자'].dt.date.isin(holidays_2019).astype(int)
53
54 # 주말
55 test['주말'] = test['weekday'].map({0:0, 1:0, 2:0, 3:0, 4:0, 5:1, 6:1})
56
57 # 휴일_주말
58 test['휴일_주말'] = (test['주말'] + test['휴일']).map({0:0, 1:1, 2:1})
```





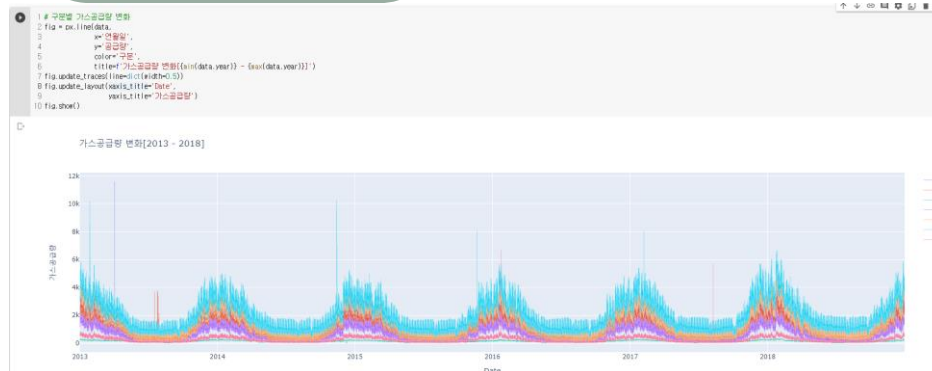
# 03

## Feature Engineering

### 변수 생성

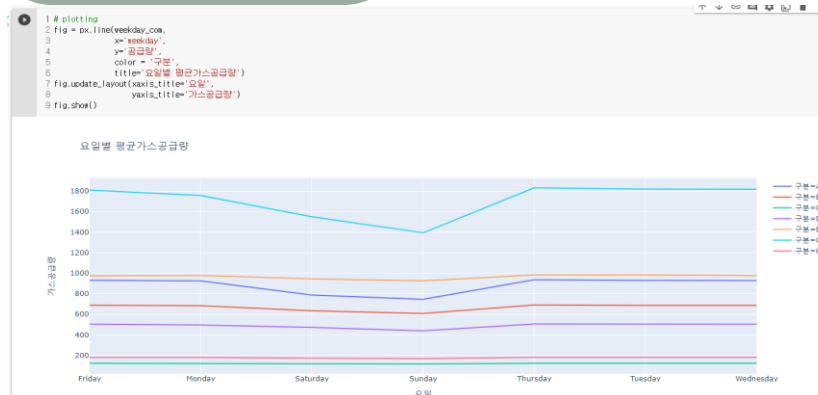
### 추가 변수 추출

#### 7) G공급사 더미변수



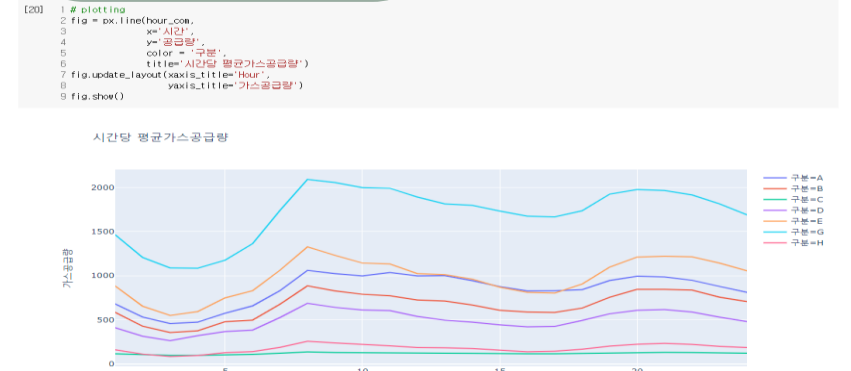
가스를 가장 많이 공급하는 G 공급사에 가중치

#### 9) 요일 더미변수



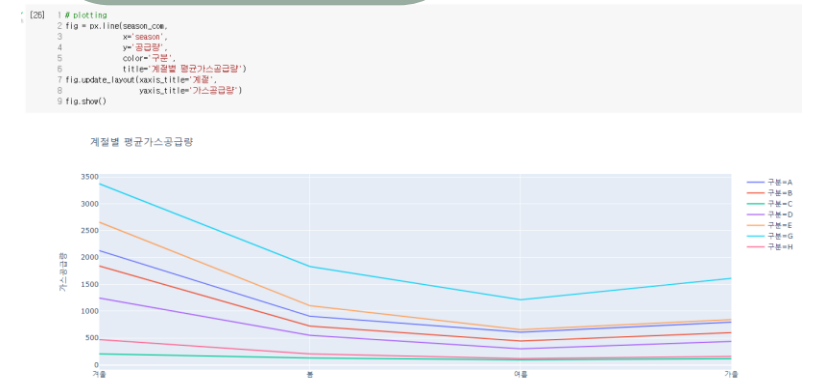
공급사 모두 주말(토,일)엔 가스공급량 감소=> 평일에 가중치

#### 8) 특정 시간 더미변수

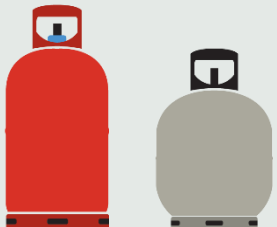


밤~새벽 시간대 감소 & 오전, 저녁 시간대 증가하는 성향 O  
증가하는 시간대에 가중치

#### 10) 겨울 더미변수



가스공급량은 겨울>봄>가을>여름 순. => 겨울에 가중치



# 03

## Feature Engineering

변수 생성

외부데이터 사용

### 11) 전산업생산지수

: KOSIS 국가통계포털 전산업생산지수(원지수)

### 12) 가스기름 상대 가격

: 한국가스공사\_도시가스 산업용 월별 상대가격지수  
: 산업에서 가스의 대체제는 기름(석유)이므로

### 13) 가스전기 상대 가격

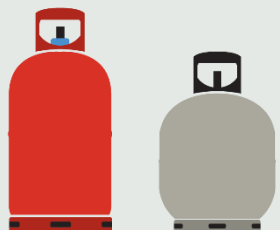
: 한국가스공사\_도시가스 민수용 월별 상대가격지수  
: 민간에서 가스의 대체제는 전기이므로

### 14) 2013~2018 기온

: 기상청 제공 일 별 기온 데이터

\* 2018년 12월 21일에 배포된 2019년 1월~3월 월평균 기온 전망 정보를 이용해  
test 데이터의 월평균 기온 변수 생성

```
16 # 겨울_더미
17 test['계절'] = test['month'].apply(lambda month_number: (month_number*12+3)//3)
18 winter = [1]
19 test['겨울_더미'] = test['계절'].apply(lambda x: 1 if x in winter else 0)
20 test = test.drop(['계절'], axis=1)
21
22 # 특정시간_더미
23 test['특정시간_더미'] = test['시간'].apply(lambda x: 0 if x in hour_list else 1)
24
25 # 평일
26 weekend = [5, 6]
27 test['평일'] = test['weekday'].apply(lambda x: 0 if x in weekend else 1)
28
29 # 6.공급사
30 the_most = [5]
31 test['6.공급사'] = test['구분'].apply(lambda x: 1 if x in the_most else 0)
32
33 # 가스기름상대가격
34 test['가스기름상대가격'] = test.progress_apply(lambda x: 60_price.loc[(60_price.year == 2018) & (60_price.month == 12), '상대가격'].values[0], axis=1)
35
36 # 가스전기상대가격
37 test['가스전기상대가격'] = test.progress_apply(lambda x: 6E_price.loc[(6E_price.year == 2018) & (6E_price.month == 12), '상대가격'].values[0], axis=1)
38
39 # 전산업생산지수
40 test['전산업생산지수'] = test.progress_apply(lambda x: IAIP.loc[(IAIP.year == 2018) & (IAIP.month == 12), '전산업생산지수(농림어업 제외)'].values[0], axis=1)
41
42 # 가스수요산업지수
43 total_2018_1 = total[(total.year == 2018) & (total.month == 1)]
44 total_2018_2 = total[(total.year == 2018) & (total.month == 2)]
45 total_2018_3 = total[(total.year == 2018) & (total.month == 3)]
46 total_1to3 = pd.concat([total_2018_1, total_2018_2, total_2018_3])
47 test['가스수요산업지수'] = total_1to3['가스수요산업지수'].values
48
```



# 04

## Modeling

modeling

모델1

**XGBoost: 전체 학습 및 예측  
& 구분별 학습 후 구분별 예측**

'구분'별로 이상치를 처리한 이상치처리\_공급량을 y값으로 설정하여  
구분별로 나눠서 학습  
→ 0.6:0.4로 앙상블

모델2

**LGBM & XGBoost**

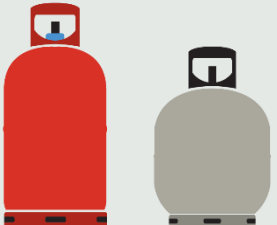
→ 0.6:0.4로 앙상블

모델3

**ExtraTree Regressor**

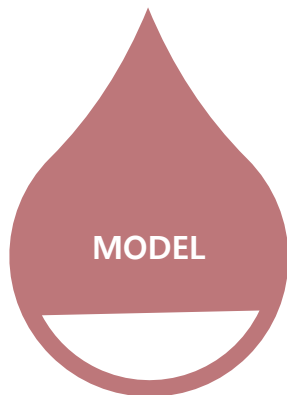
최종 앙상블

5 : 3 : 2



# 04

## Modeling

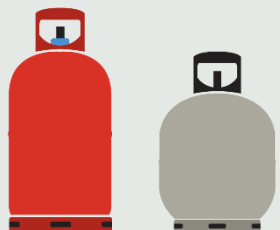


## XGBoost

Gradient Boosting 알고리즘(여러 개의 약한 Decision Tree를 Ensemble)을 분산환경에서도 실행할 수 있도록 구현해 놓은 라이브러리 (병렬 학습 지원)

### 장점

1. 병렬 처리로 학습 및 분류 속도가 빠름
2. 자체 과적합 규제 기능으로 강한 내구성을 지님
3. Early Stopping(조기 종료) 기능 존재
4. 다양한 옵션을 제공해 customizing 용이



# 04

## Modeling

# Hyper parameter Tuning – XGBoost

2단계: gamma 튜닝

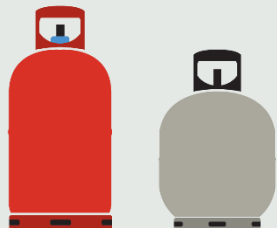
```
def objectiveXGB(trial, train_x = train_x, train_y = train_y, val_x = val_x, val_y = val_y):  
    param = {  
        'eta': 0.05,  
        'num_boost_rounds': 3000,  
        'eval_metric': 'mae',  
        'seed': 1,  
        'max_depth': 10,  
        'min_child_weight': 164,  
        'gamma': trial.suggest_categorical('gamma', [0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]),  
        'sub_sample': 0.8,  
        'colsample_bytree': 0.8  
    }  
  
    dtrain = xgb.DMatrix(data=train_x, label=train_y)  
    dval_x = xgb.DMatrix(val_x)  
  
    model = xgb.train(param, dtrain)  
  
    preds = model.predict(dval_x)  
  
    score = mean_absolute_error(val_y, preds)  
  
    return score
```

```
study = optuna.create_study(direction='minimize')  
study.optimize(objectiveXGB, n_trials = 50)  
print('Best trial: score {},\nparams {}'.format(study.best_trial.value, study.best_trial.params))
```

3단계: sub\_sample & colsample\_bytree

```
def objectiveXGB(trial, train_x = train_x, train_y = train_y, val_x = val_x, val_y = val_y):  
    param = {  
        'eta': 0.05,  
        'num_boost_rounds': 3000,  
        'eval_metric': 'mae',  
        'seed': 1,  
        'sub_sample': trial.suggest_categorical('sub_sample', [0.5, 0.6, 0.7, 0.8, 0.9]),  
        'colsample_bytree': trial.suggest_categorical('colsample_bytree', [0.5, 0.6, 0.7, 0.8, 0.9])  
    }
```

단계별 파라미터 튜닝 진행



# 04

## Modeling

### Hyper parameter Tuning – XGBoost

#### max\_depth

default = None

트리의 최대 깊이

값이 증가함에 따라 초반에는 test set에 대한 성능이 증가하지만 특정 시점 이후에는 train set에 과적합이 발생해 성능 감소

#### min\_child\_weight

min\_samples\_leaf와 유사

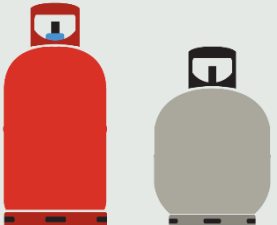
관측치에 대한 가중치 합을 최소를 말하지만  
GBM에서는 관측치 수에 대한 최소를 의미  
-과적합 조절 용도

#### gamma

리프노드의 추가분할을 결정할 최소손실 감소값  
해당값보다 손실이 크게 감소할 때 분리  
값이 클수록 과적합 감소효과

#### sub\_sample

GBM의 subsample과 동일  
데이터 샘플링 비율 지정(과적합 제어)  
일반적으로 0.5~1 사이의 값을 사용  
범위: 0 ~ 1



# 04

## Modeling

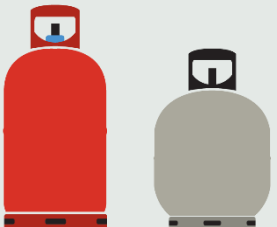
### Hyper parameter Tuning – XGBoost

#### alpha

L1 Regularization 적용 값  
피쳐 개수가 많을 때 적용을 검토  
클수록 과적합 감소 효과

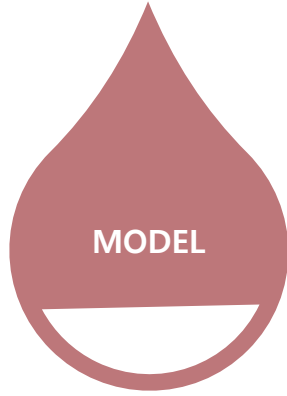
#### lambda

L2 Regularization 적용 값  
피쳐 개수가 많을 때 적용을 검토  
클수록 과적합 감소 효과



# 04

## Modeling

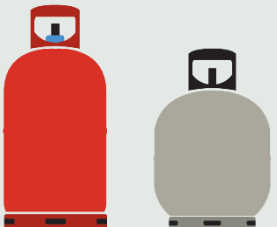


## ExtraTree Regressor

데이터 샘플 수, 트리 형성을 위한 feature 선택까지  
무작위성을 부여해 Random Forest보다 더욱 극단적으로  
랜덤하게 만들어진 모델

### 장점

1. Bootstrapping을 쓰지 않고 Whole Origin Data를 사용하므로 Random Forest에 비해 Bias를 낮출 수 있음
2. Split point를 랜덤하게 선택해 Variance를 줄일 수 있음
3. 연산 속도 개선





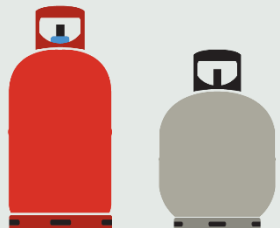
# 04

## Modeling

# Hyper parameter Tuning - ExtraTree Regressor

```
def optimi_maxdepth (algorithm, algorithm_name, x_train, y_train, x_test, y_test, depth_min, depth_max, n_estimator):  
    train_score = []; test_score = []  
    para_depth = [depth for depth in range(depth_min, depth_max)]  
  
    for v_max_depth in para_depth:  
        if algorithm == DTC:  
            model = algorithm(max_depth = v_max_depth,  
                              random_state=1234)  
        else:  
            model = algorithm(max_depth = v_max_depth,  
                              n_estimators = n_estimator,  
                              random_state=1234)  
        model.fit(x_train, y_train)  
        train_score.append(model.score(x_train, y_train))  
        test_score.append(model.score(x_test, y_test))  
    df_score_n = pd.DataFrame({'depth': para_depth, 'TrainScore': train_score, 'TestScore': test_score})  
    optimi_visualization(algorithm_name, para_depth, train_score, test_score, "The number of depth", "n_depth")  
    print(round(df_score_n, 4))
```

하이퍼 파라미터 튜닝을 위해 함수 정의하여 진행



# 04

## Modeling

# Hyper parameter Tuning - ExtraTree Regressor

## n\_estimator

default = 100

트리의 개수 결정

값이 클수록 일반화된 결과 생성이  
가능하지만 소요 시간이 증가

## max\_depth

default = None

트리의 최대 깊이

값이 증가함에 따라 초반에는 test set에 대한 성능이  
증가하지만 특정 시점 이후에는 train set에 과적합이  
발생해 성능 감소

## min\_samples\_split

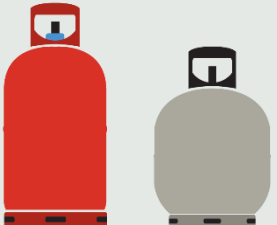
default = 2

노드를 분할하는데 필요한 최소 샘플 수  
작게 설정할수록 분할 노드가 많아져  
과적합 가능성 증가

## min\_samples\_leaf

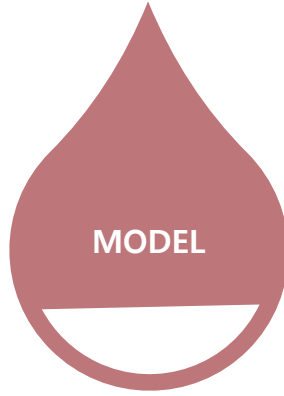
default = 1

leaf 노드를 만들기 위해 필요한 최소 샘플 leaf  
값이 증가함에 따라 과적합 방지



# 04

## Modeling

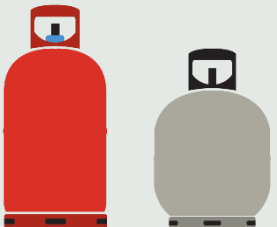


# LGBM (Light Gradient Boosting Machine)

Tree구조가 수평적으로 확장하는 다른 기존 Tree기반의 알고리즘에 비해 수직적으로 확장을 한다. (leaf-wise 방식)

## 장점

1. 학습하는데 시간이 적게 걸린다
2. 메모리 사용량이 상대적으로 적은 편이다
3. 카테고리형 피처들의 자동 변환과 최적 분할
4. 확장을 위해 max delta loss를 가진 leaf를 선택하게 되므로 level-wise 알고리즘과 비교하여 더 많은 손실을 줄일 수 있다
5. 최신의 것이기에 기능상의 다양성이 더 많다



# 04

## Modeling

### Hyper parameter Tuning - LGBM

#### num\_iterations

default = 100

반복 수행하려는 트리의 개수를 지정  
너무 크게 설정하면 과적합 발생

#### learning\_rate

default = 0.1

부스팅을 반복 수행할 때 업데이트 되는 학습률  
0~1 사이의 값을 지정

#### max\_depth

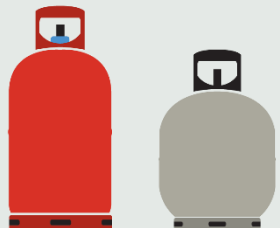
default = 1

트리의 최대 깊이  
0보다 작은 값을 입력하면 깊이에 제한 없음

#### min\_data\_in\_leaf

default = 20

의사결정나무의 min\_samples\_leaf와  
같은 파라미터  
과적합을 제어해주는 파라미터로 사용



# 04

## Modeling

# Hyper parameter Tuning - LGBM

## num\_leaves

default = 3

하나의 트리가 가질 수 있는 최대 리프 개수  
개수를 높이면 정확도가 높아지지만 모델이 너무 복잡해지면 과  
적합 발생 가능성이 커진다

## Bagging\_fraction

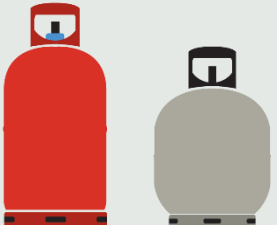
default = 1.0

데이터를 샘플링 하는비율  
과적합을 제어하기 위해 사용

## Feature\_fraction










default = 1.0

개별 트리를 학습할 때  
무작위로 선택하는 피처의 비율



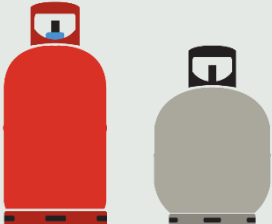
# 05

최종결과/  
의의 및 한계

최종 결과					
30	국경원 요원		0.10828	1	24일 전
31	Flada		0.10831	1	25일 전
32	야성적 눈썹		0.10898	13	21일 전
33	kchanis1223		0.10985	1	23일 전
34	유&강		0.11043	8	21일 전
35	쿠(KU)스트라다무스		0.11121	15	21일 전
36	GGASBARY		0.11126	3	21일 전
37	돈가스게임		0.11188	13	21일 전
38	세월을 낚는 낚시꾼		0.11335	1	25일 전

NMAE 척도 기준 - 0.11121

데이콘 리더보드 35위!!



# 05

## 최종결과/ 의의 및 한계

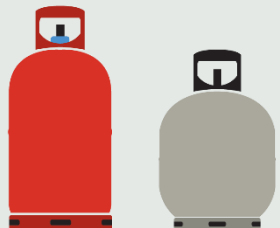
# 의의 및 한계

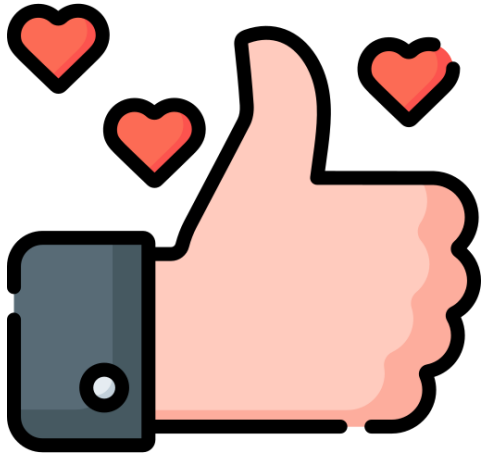
### 의의

- 한국가스공사가 제공하는 실제 데이터를 다루어 볼 수 있었다.
- 다양한 앙상블을 진행해봄으로써 여러 모델 기법을 공부하고 사용해볼 수 있었다.

### 한계

- 시계열과 관련된 모델을 사용해보지 못했다.
- Data Leakage에 대한 우려 & 주어진 데이터가 적어서 파생 변수 생성에 어려움이 있었다.





**들어주셔서 감사합니다.**