토끼와두루미
후후니이
lch9826@gmail.com
나잼니
gkwlals6189@gmail.com
다콩이
dab2in79@gmail.com

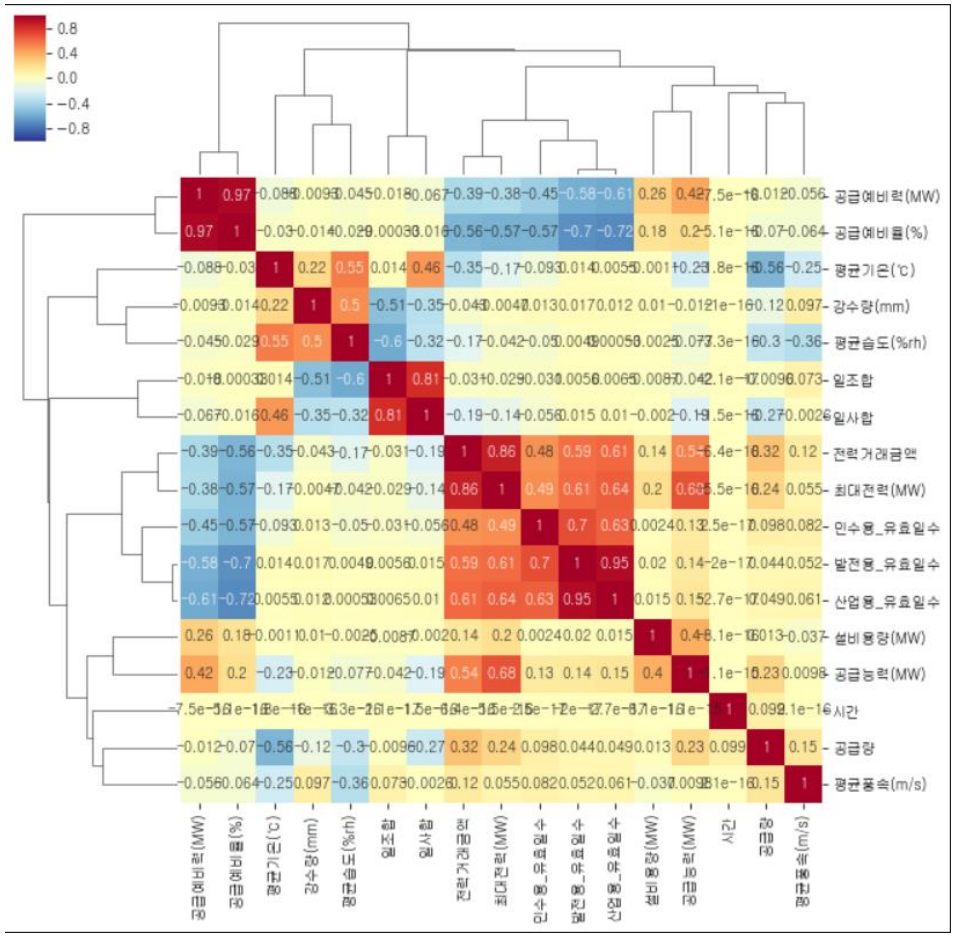
DACON
DATA TO VALUE

가스공급량 수요예측 모델개발

1. 개요

분석 개요

- 시간별 공급량 데이터를 Base로, 외부데이터(기상청, 한국가스공사 등)을 활용하려고 하였음.
  - 외부데이터를 활용해 데이터프레임을 생성하고 상관관계를 확인해보았으나, 상관계수 값이 너무 낮아, 외부데이터를 쓰지 않기로 함.
  - 실제로, VAR, LSTM과 같은 다변량 시계열 분석을 진행하였으나, score가 너무 낮게 나왔음.
- 이후, 시간별 가스 공급량 데이터만을 이용해 ARIMA 시계열 분석을 진행하려고 하였음.
  - 실제로 ARIMA 분석을 진행해보았지만, 추세성, 계절성, 차분 과정에서 적합하지 않다고 생각되어, Fbprophet 라이브러리를 활용해 분석을 진행하였음.
  - Fbprophet을 활용한 score는 나쁘지 않았음.
- 여러 방법을 활용한 후, 최종적으로는 score가 가장 높았던 RandomForest Regressor를 분석방법으로 선택함.



외부데이터를 활용한 상관관계 히트맵

2. 데이터 핸들링

2.1 데이터 전처리

```
# 범주형 데이터 확인
total['구분'].unique()
```

```
array(['A', 'B', 'C', 'D', 'E', 'G', 'H'], dtype=object)
```

```
# 숫자 데이터로 변경
d_map = {}
for i, d in enumerate(total['구분'].unique()):
    d_map[d] = i
total['구분'] = total['구분'].map(d_map)
```

```
# 날짜 데이터 타입 확인
total['연월일'].dtype
```

```
dtype('O')
```

```
# 날짜 데이터 datetime 형식으로 변경
total['연월일'] = pd.to_datetime(total['연월일'])
```

```
# 날짜 데이터에서 연, 월, 일, 요일 데이터 생성
total['year'] = total['연월일'].dt.year
total['month'] = total['연월일'].dt.month
total['day'] = total['연월일'].dt.day
total['weekday'] = total['연월일'].dt.weekday
```

- 공급사를 나타내는 '구분'열을 수치형(numeric)으로 변환함.
- 날짜를 나타내는 '연월일'열을 datetime형식으로 변환함.

2. 데이터 핸들링

2.2 이상치 처리

- 공급사를 나타내는 '구분' 열을 기준으로 A, B, G, H에 해당하는 행의 이상치를 처리했음.

```
total_a.head()
```

```
total_aa = pd.DataFrame(data = {'ds' : total_a["연월일"], 'y' : total_a['공급량']})
total_aa.head()
```

```
plt.figure(figsize = (12,3))
plt.plot(total_aa["ds"], total_aa["y"])
plt.show()
```

```
# 이상치 데이터 결측치 처리
total_aa[2230:2231]['y'] = np.nan
```

```
total_aa[2230:2231]['y']
```

```
total_aa.isna().sum()
```

```
total_a['공급량' ] = total_aa['y']
```

A공급사 가스공급량 이상치 처리

결측치 보완 method를 적용하기 위해
이상치 데이터를 결측치로 변환함.

2. 데이터 핸들링

2.2 이상치 처리

```
total_bb = pd.DataFrame(data = {'ds' : total_b["연월일"], 'y' : total_b['공급량']})
total_bb.head()
```

```
plt.figure(figsize = (12,3))
plt.plot(total_bb["ds"], total_bb["y"])
plt.show()
```

```
plt.figure(figsize = (12,3))
plt.plot(total_bb[total_b["year"] == 2013]['ds'], total_bb[total_b["year"] == 2013]["y"])
plt.show()
```

```
# 이상치 데이터 확인
total_bb[(total_b['year'] == 2013) & (total_b['month'] == 7) & (total_b['공급량'] > 1000) \
      | (total_b['year'] == 2018) & (total_b['공급량'] > 3550)]
```

```
# 이상치 데이터 인덱스 저장
outlier_b = total_bb[(total_b['year'] == 2013) & (total_b['month'] == 7) & (total_b['공급량'] > 1000) \
                  | (total_b['year'] == 2018) & (total_b['공급량'] > 3550)].index
```

```
# 이상치 데이터 결측치 처리
for i in outlier_b:
    total_bb[i:i+1]['y'] = np.nan
```

```
total_b['공급량'] = total_bb['y']
```

```
total_b.isna().sum()
```

B공급사 가스공급량 이상치 처리

결측치 보완 method를 적용하기 위해
이상치 데이터를 결측치로 변환함.

2. 데이터 핸들링

2.2 이상치 처리

```
total_gg = pd.DataFrame(data = {'ds' : total_g["연월일"], 'y' : total_g['공급량']})
total_gg.head()
```

```
plt.figure(figsize = (12,3))
plt.plot(total_gg["ds"], total_gg["y"])
plt.show()
```

```
total_gg[total_gg['y'] > 7000]
```

```
# 이상치 데이터 인덱스 저장
outlier_g = total_gg[total_gg['y'] > 7000].index
outlier_g
```

```
# 이상치 데이터 결측치 처리
for i in outlier_g:
    total_gg[i:i+1]['y'] = np.nan
```

```
total_g['공급량'] = total_gg['y']
```

```
total_g.isna().sum()
```

G공급사 가스공급량 이상치 처리

결측치 보완 method를 적용하기 위해
이상치 데이터를 결측치로 변환함.

2. 데이터 핸들링

2.2 이상치 처리

```
total_hh = pd.DataFrame(data = {'ds' : total_h["연월일"], 'y' : total_h['공급량']})
total_hh.head()
```

```
plt.figure(figsize = (12,3))
plt.plot(total_hh["ds"], total_hh["y"])
plt.show()
```

```
total_hh[total_hh['y'] > 1000]
```

```
# 이상치 데이터 인덱스 저장 및 결측치 처리
outlier_h = total_hh[total_hh['y'] > 1000].index

for i in outlier_h:
    total_hh[i:i+1]['y'] = np.nan
```

```
total_h['공급량'] = total_hh['y']
```

```
total_h.isna().sum()
```

H공급사 가스공급량 이상치 처리

결측치 보완 method를 적용하기 위해
이상치 데이터를 결측치로 변환함.

2. 데이터 핸들링

2.3 데이터 병합 및 훈련/검증 데이터 셋 분할

```
total = pd.concat([total_a, total_b, total_c, total_d, total_e, total_g, total_h], ignore_index = True)
total.head()
```

```
total.isna().sum()
```

```
# 결측치 데이터를 그 이전의 값으로 채우는 'bfill' 적용
total = total.fillna(method='bfill')
```

```
total.isna().sum()
```

```
# 훈련에 사용할 train set, 점수 확인을 위한 validation set 분할
train_years = [2013,2014,2015,2016,2017]
val_years = [2018]
```

```
train = total[total['year'].isin(train_years)]
val = total[total['year'].isin(val_years)]
```

```
# 독립변수, 종속변수 분할
features = ['구분', 'month', 'day', 'weekday', '시간']
train_x = train[features]
train_y = train['공급량']

val_x = val[features]
val_y = val['공급량']
```

결측치 데이터를 그 이전의 값으로 채우는 bfill method 적용

3. 모델 생성 및 학습

3.1 파라미터 선정

```
# train 및 val 설명력 결과 저장
train_score = []
val_score = []

# n_estimators : 트리 수 변경
para_n_tree = [n_tree * 10 for n_tree in range(1, 11)]

for v_n_estimators in para_n_tree:
    rf = RandomForestRegressor(n_estimators = v_n_estimators, random_state = 1234)
    rf.fit(train_x, train_y)
    train_score.append(rf.score(train_x, train_y))
    val_score.append(rf.score(val_x, val_y))

# 결과 저장
df_score_n = pd.DataFrame()
df_score_n["n_estimators"] = para_n_tree
df_score_n["TrainScore"] = train_score
df_score_n["ValScore"] = val_score
```

```
# 모델 설명력 확인
df_score_n.round(3)
```

```
plt.plot(para_n_tree, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_n_tree, val_score, linestyle = "--", label = "Val Score")
plt.ylabel('score'); plt.xlabel("n_estimators")
plt.legend()
plt.show()
```

```
# 과대적합 방지를 위해 트리 수는 30으로 선택
n_estimators = 30
```

① n_estimators: 트리 개수

- 랜덤 포레스트에 들어가는 의사결정나무 개수를 설정
- 여러 트리 모델의 결과를 선형 결합해 최종 모델을 만드는데 트리 개수를 감소시키면 일반적으로 과대 적합 방지
- 과대적합 방지를 위해 트리 수는 30으로 선택

3. 모델 생성 및 학습

3.1 파라미터 선정

```
# train 및 val 설명력 결과 저장
train_score = []
val_score = []

# min_samples_leaf : 잎사귀 최소 자료 수
para_leaf = [n_leaf for n_leaf in range(1, 21)]

for v_min_samples_leaf in para_leaf:
    rf = RandomForestRegressor(n_estimators = 30, random_state = 1234,
                               min_samples_leaf = v_min_samples_leaf)
    rf.fit(train_x, train_y)
    train_score.append(rf.score(train_x, train_y))
    val_score.append(rf.score(val_x, val_y))

# 결과 저장
df_score_leaf = pd.DataFrame()
df_score_leaf["MinSamplesLeaf"] = para_leaf
df_score_leaf["TrainScore"] = train_score
df_score_leaf["ValScore"] = val_score
```

```
# 모델 설명력 확인
df_score_leaf.round(3)
```

```
plt.plot(para_leaf, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_leaf, val_score, linestyle = "--", label = "Val Score")
plt.ylabel('score'); plt.xlabel("min samples leaf")
plt.legend()
plt.show()
```

```
# 잎사귀 노드 최소 자료 수 증가에 따라 설명력은 감소,
# val데이터 정확도 변화를 고려해 6 선택
min_samples_leaf = 6
```

② min_samples_leaf: 잎사귀 노드 최소 자료 수

- 최소 자료 수를 증가시키면 분리 조건이 엄격해져 과대적합이 방지됨
- 하지만 잎사귀 노드 최소 자료 수 증가에 따라 모델의 설명력은 감소
- val 데이터 정확도 변화를 고려해 6 선택

3. 모델 생성 및 학습

3.1 파라미터 선정

```
# train 및 val 설명력 결과 저장
train_score = []
val_score = []

# min_samples_split : 분할하기 위한 노드 최소 자료 수
para_split = [n_split*2 for n_split in range(2, 21)]

for v_min_samples_split in para_split:
    rf = RandomForestRegressor(n_estimators = 30, random_state = 1234,
                               min_samples_leaf = 6,
                               min_samples_split = v_min_samples_split)
    rf.fit(train_x, train_y)
    train_score.append(rf.score(train_x, train_y))
    val_score.append(rf.score(val_x, val_y))

# 결과 저장
df_score_split = pd.DataFrame()
df_score_split["MinSamplesSplit"] = para_split
df_score_split["TrainScore"] = train_score
df_score_split["ValScore"] = val_score
```

```
# 모델 설명력 확인
df_score_split.round(3)
```

```
plt.plot(para_split, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_split, val_score, linestyle = "--", label = "Val Score")
plt.ylabel('score'); plt.xlabel("min samples split")
plt.legend()
plt.show()
```

```
# 분리 노드 최소 자료 수 증가에 따라 모델의 설명력 감소,
# train/val 데이터 성능 변화를 고려해 13 선택
min_samples_split = 13
```

③ min_samples_split: 분리 노드 최소 자료 수

- 분리 노드 최소 자료 수 증가에 따라 모델의 설명력은 감소
- train/val 데이터 성능 변화를 고려해 13 선택

3. 모델 생성 및 학습

3.1 파라미터 선정

```
# train 및 val 설명력 결과 저장
train_score = []
val_score = []

# max_depth : 최대 깊이 변경
para_depth = [depth for depth in range(1, 21)]
for v_max_depth in para_depth:
    rf = RandomForestRegressor(n_estimators = 30, random_state = 1234,
                               min_samples_leaf = 6, min_samples_split = 13,
                               max_depth = v_max_depth)
    rf.fit(train_x, train_y)
    train_score.append(rf.score(train_x, train_y))
    val_score.append(rf.score(val_x, val_y))

# 결과 저장
df_score_depth = pd.DataFrame()
df_score_depth["Depth"] = para_depth
df_score_depth["TrainScore"] = train_score
df_score_depth["ValScore"] = val_score
```

```
# 모델 설명력 확인
df_score_depth.round(3)
```

```
plt.plot(para_depth, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_depth, val_score, linestyle = "--", label = "Val Score")
plt.ylabel('score'); plt.xlabel("min samples split")
plt.legend()
plt.show()
```

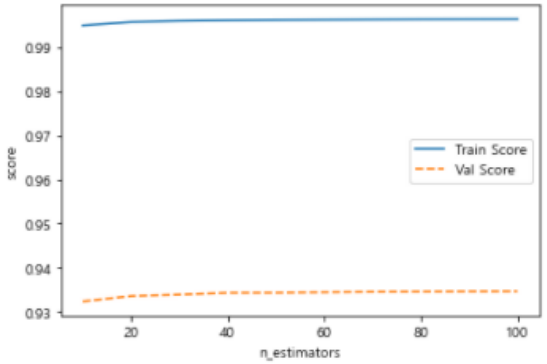
```
# 분리 노드 최소 자료 수 증가에 따라 모델의 설명력 감소,
# train/val 데이터 성능 변화를 고려해 16 선택
max_depth = 16
```

④ max_depth: 최대 깊이 변경에 따른 모델 성능

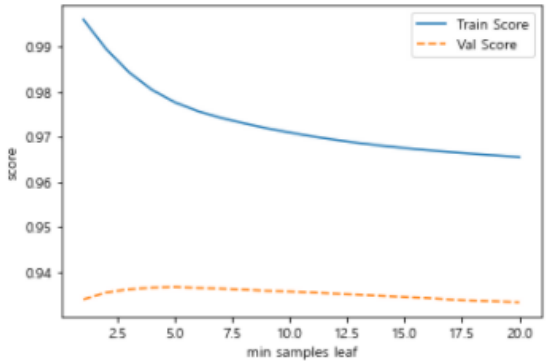
- 최대 깊이를 감소시키면 깊이 제약으로 과대적합이 방지됨
- 최대 깊이 증가에 따라 모델의 설명력은 증가함
- train/val 데이터의 정확도 변화를 고려해 16 선택

3. 모델 생성 및 학습

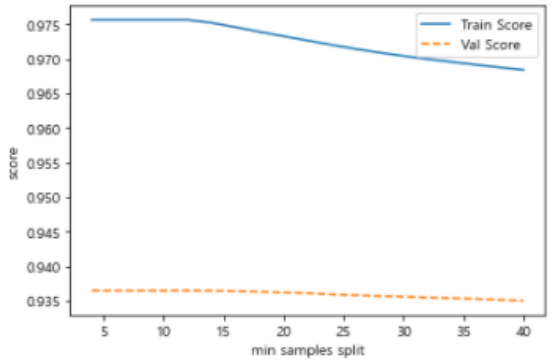
3.1 파라미터 선정



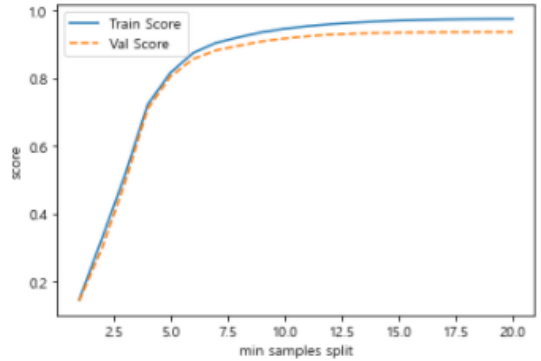
① n_estimators
트리 개수



② min_samples_leaf
잎사귀 노드 최소 자료 수



③ min_samples_split
분리 노드 최소 자료 수



④ max_depth
최대 깊이 변경에 따른 모델 성능

3.2 최적의 파라미터로 모델 훈련 및 검증

```
rf = RandomForestRegressor(max_depth = 16, min_samples_split = 13,
                           min_samples_leaf = 6, n_estimators = 30, random_state = 1234)
rf.fit(train_x, train_y)
```

```
print("training set 점수 : {:.3f}".format(rf.score(train_x, train_y)))
print("val set 점수 : {:.3f}".format(rf.score(val_x, val_y)))
```

```
training set 점수 : 0.972
val set 점수 : 0.936
```

4. 추론 및 결과 도출

4.1 훈련 데이터 생성

```
# 실제 훈련을 위한 데이터 생성
train_all_years = [2013,2014,2015,2016,2017,2018]
```

```
train_all = total[total['year'].isin(train_all_years)]
```

```
features = ['구분', 'month', 'day', 'weekday', '시간']
train_xx = train_all[features]
train_yy = train_all['공급량']
```

- 훈련 데이터의 시간적 범위: 2013 – 2018년

4.2 모델 훈련

```
rf = RandomForestRegressor(max_depth = 16, min_samples_split = 13,
                           min_samples_leaf = 6, n_estimators = 30, random_state = 1234)
rf.fit(train_xx, train_yy)
```