Team Project 2nd - 으쌰으쌰 팀😎

팀장 가채원, 부팀장 김범중, 김진연, 윤진훈

[노션링크]

▼ **제출 점수(최고점)**

발표 전(public) : **0.1110990574(46등 2021-11-10)**

발표 후(public) : **0.1030211538(42등 2021-12-05 13:54:10)**

private : **0.1023986349(18등 2021-12-08 10:43:10)**

개요

1. 프로젝트 목표
2. 가설 세우고 확인하기
3. 채택하지 않은 가설
4. 머신러닝 모델 비교

5. GridSearch를 이용한 하이퍼파라미터 튜닝
6. 과대적합 해소를 위한 Feature Selection
7. 최종 발표까지 목표
8. 여러 모델 결과 합치기
9. Polynomial Features 적용하기
10. HyperOpt 적용하기
11. PyCaret 적용하기
12. 머신러닝 모델 결과 종합
13. 결론 - 배운 점과 팀 자체 평가

팀 구성 및 역할

팀장 가채원

- 프로젝트 계획 및 팀 조율
- 데이터 탐색 및 시각화
- 중간 결과 발표 자료 작성 (노션 정리 및 작성)
- 자료 리서치
- 데이터 탐색 및 시각화 (태블로 활용)

부팀장 김범중

- 기온 데이터 수집 및 전처리
- EDA
- 다양한 모델 비교
- 평가 지표(MSE, RMSE, 결정계수, MAE, NMAE) 활용한 평가
- PolynomialFeatures, 비선형 변환(log) 등 적용
- 예측 결과 종합
- 노션 정리 및 작성

팀원 김진연

- 가스 공급량, 기온 데이터 분석 및 태블로 시각화

- 머신러닝 모델 비교
- 머신러닝 모델 예측치 평균 내는 방법론 조사
- GridSearch를 이용한 하이퍼파라미터 튜닝
- HyperOpt 이용한 하이퍼파라미터 튜닝
- PyCaret 이용한 모델 비교 및 하이퍼파라미터 튜닝
- 노션 발표 자료 작성 및 정리

팀원 윤진훈

- 데이터 탐색 및 시각화
- 모델 구축 및 개선 (선형회귀, 의사결정트리, 앙상블)

한국가스공사 가스 사용량 예측하기

> ### 가스공급량 수요예측 모델개발

한국가스공사의 시간단위 공급량 내부데이터와 기상정보 및 가스외 발전량 등 외부데이터를 포함한 데이터셋을 구축하여 90일 한도 일간 공급량을 예측하는 인공지능 모델을 개발한다.

가설 세우기

> 💡 **기온이 낮을수록 도시가스 공급량이 늘어난다.**

- 가정용 도시가스는 난방에 사용된다.
- 기온이 낮으면 사람들은 난방을 더 많이 사용한다.

∴ 기온이 낮으면 도시가스 공급량이 늘어날 것이다.

가설 확인하기

1. 1, 4분기가 2, 3분기에 비해 기온이 낮다.
2. 1, 4분기에 2, 3분기보다 공급량이 많다.
3. 가스 공급량과 기온 데이터 중 이상치가 없는지 확인한다.

기본 데이터 분석

https://public.tableau.com/app/profile/.68963375/viz/211109_16364438010640/sheet2

https://public.tableau.com/views/211109_16364438010640/sheet2?:showVizHome=no&:embed=true

분석 내용

분기별 공급량 합계

[상대적] 1분기와 4분기의 공급량 합계 > 2분기와 3분기의 공급량 합계

⇒ '가설 확인하기-2' 확인 가능

공급사별 공급량 합계

공급사별 공급량 차이 존재

⇒ 공급사 : 공급량 예측의 중요 변수

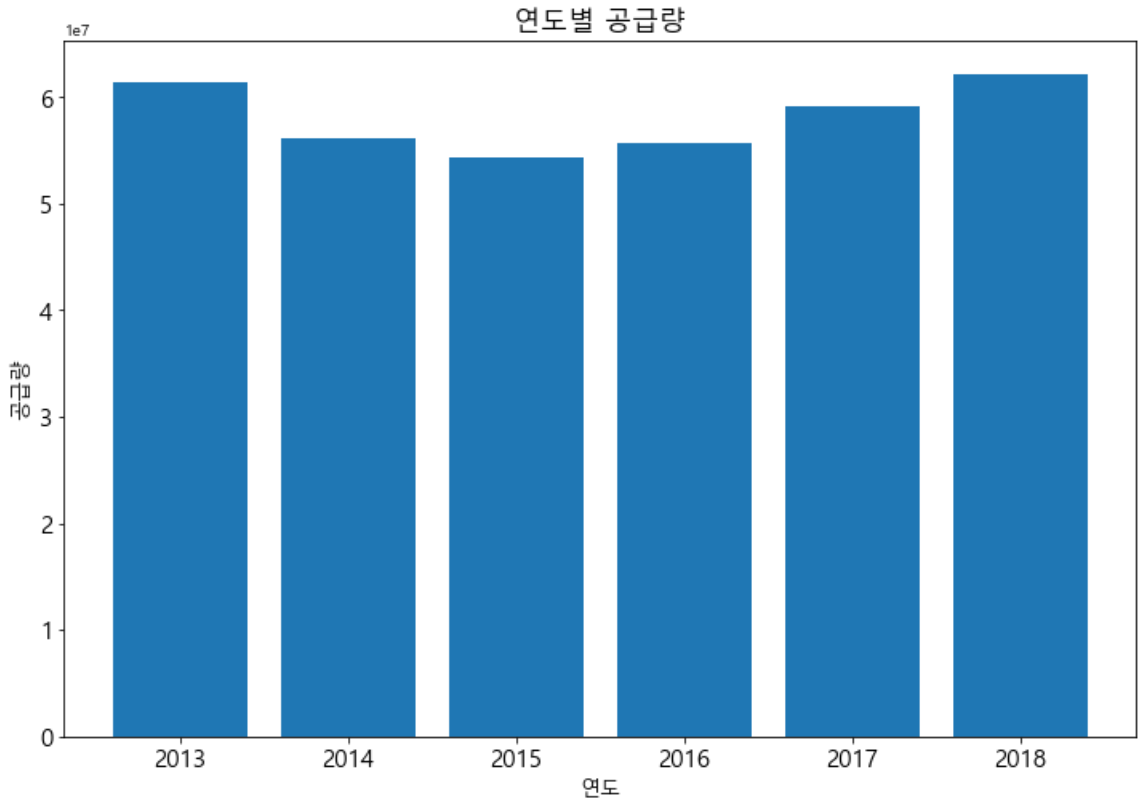
공급사별 월별 공급량 합계

월별 분석 결과, 가스 공급량 U자 그래프 확인 가능

⇒ IF 기온 그래프 A자 모양, 기온이 낮아질 시 가스 공급량 증가의 가설 신빙성 증가

▼ 기본 데이터 분석 내용

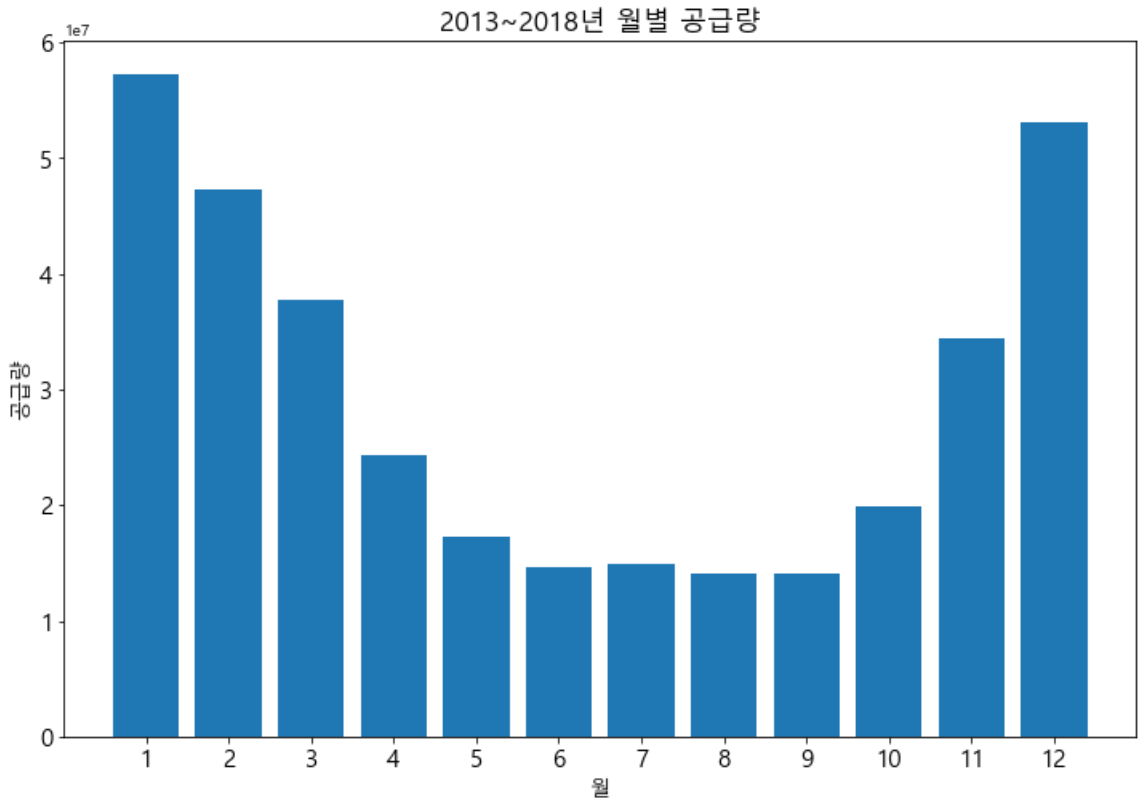
연도별 가스 공급량



- 가스공급량은 연도별로 큰 차이를 보이지 않는다.

⇒ 연도별로는 큰 차이를 보이지 않아 제외하고 훈련하는 것이 좀 더 좋은 성능을 보일 것으로 예상.

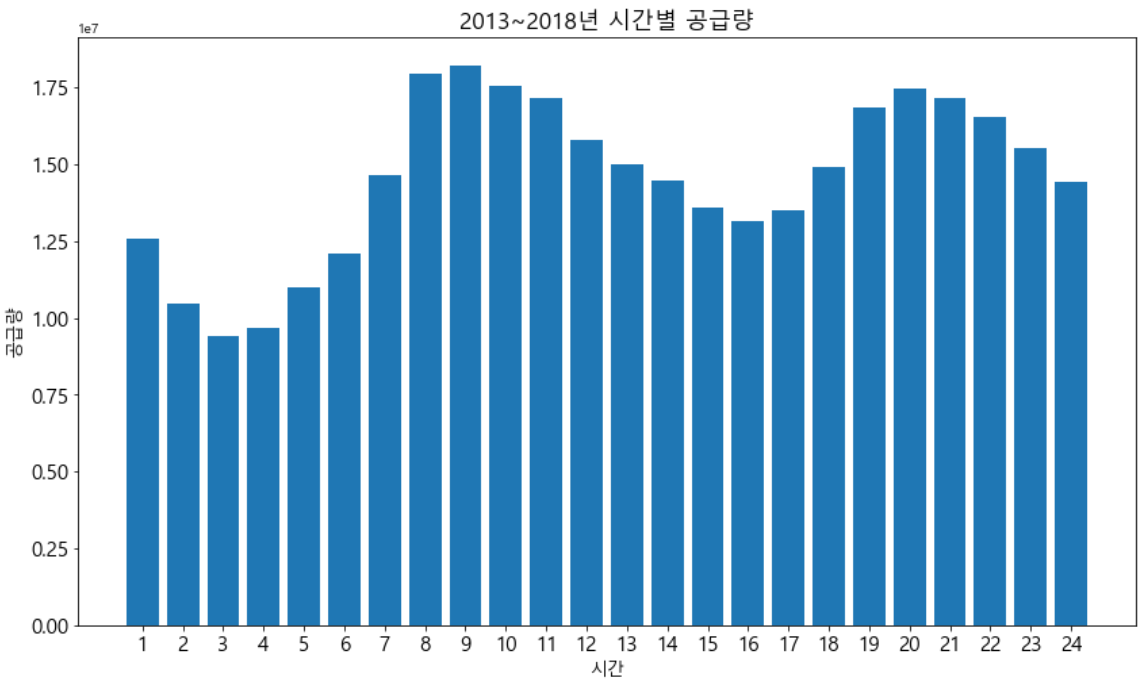
월별 가스공급량



- 추워지는 날씨에 가스공급량이 늘고, 더워지는 날씨에 감소한다.

⇒ 계절에 따른 예측이 유효 할 것.

시간별 가스공급량



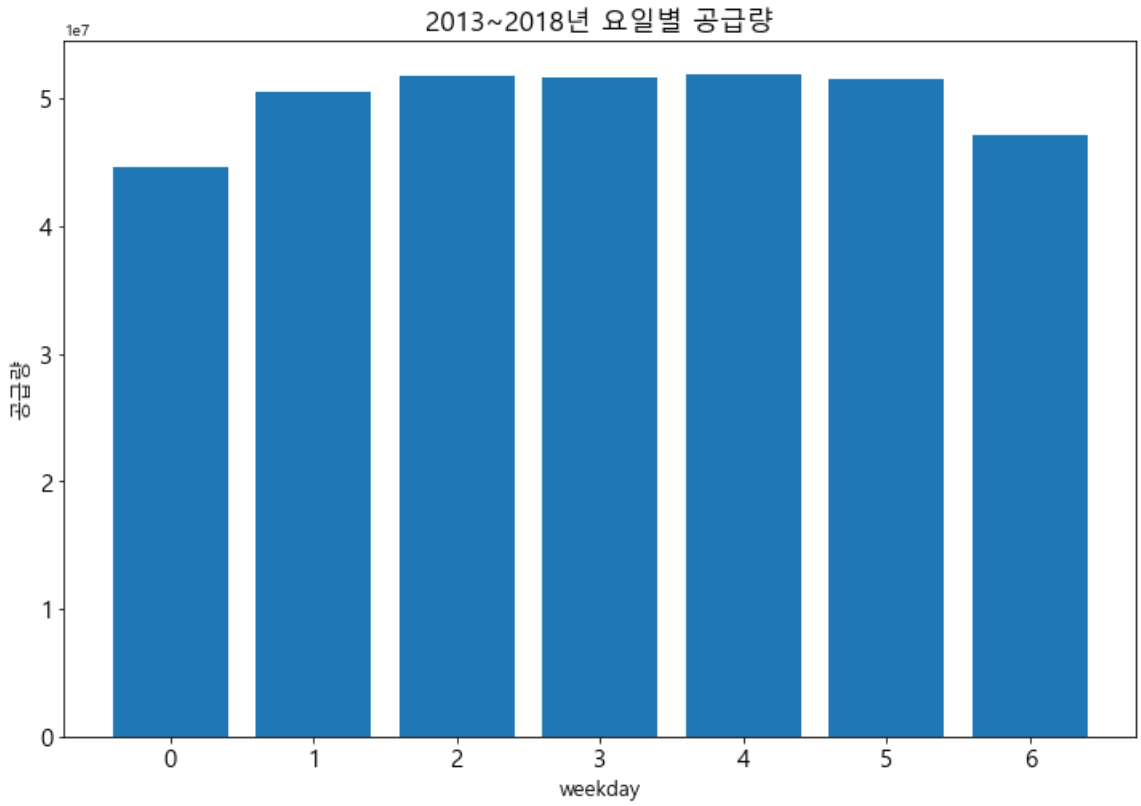
- 새벽 3시부터 오전 8시까지 가스 공급량이 증가했다가 오후 4시까지 낮아진다.
- 오후 4시 이후부터 8시까지 증가했다가 새벽 3시까지 감소한다.

해석

- 공급량이 변화하는 시간대로 보아, 자택의 거주 인원 변화와 기온에 따라 공급량 변화가 있을 것으로 예상된다.
- 가스 공급량의 변화가 기온의 변화에만 영향이 있지 않는 것으로 보여, 일반적인 직장인, 학생들의 활동과 연관이 있어보인다.
- 사람들의 생활 패턴에 영향이 있는 것으로 보인다.

⇒ 시간과 기온 데이터가 분석에 좋은 특성이 될 것.

요일별 가스공급량



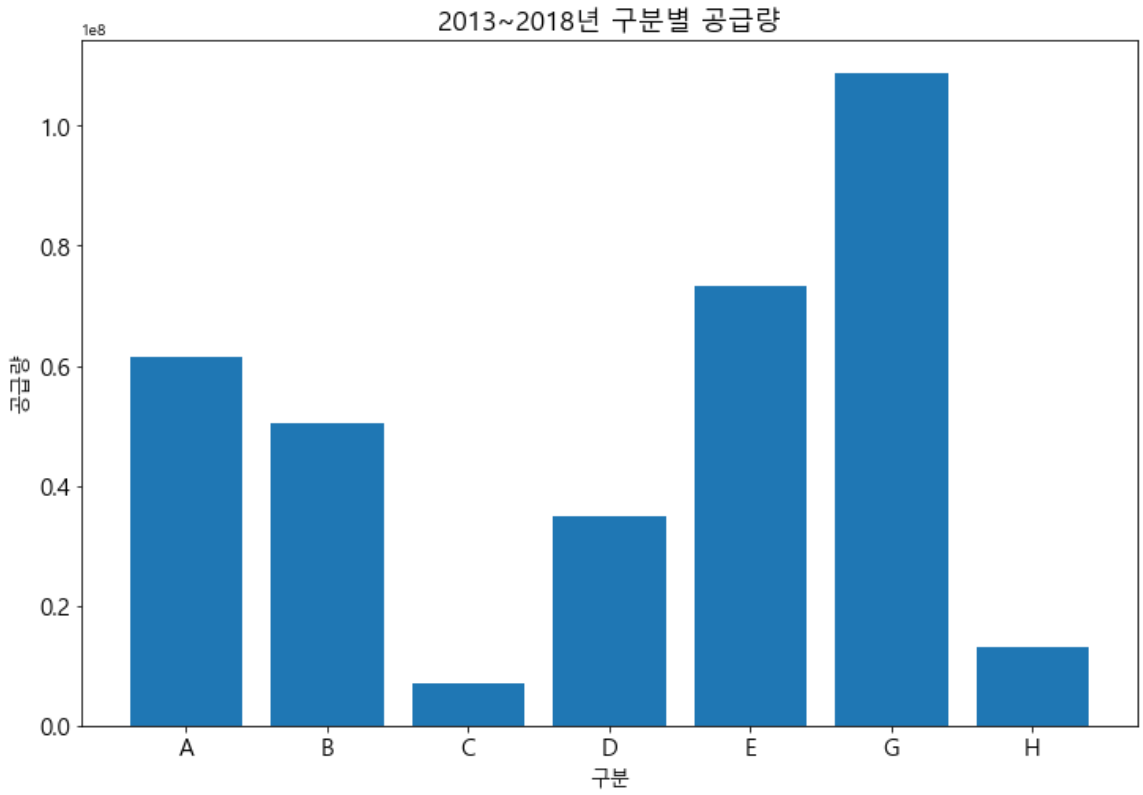
- 평일에는 비슷한 공급량을 보이지만 주말에는 감소하는 경향을 보인다.

해석

- 주말에는 외부 활동으로 가정 난방이 줄어든 것으로 예상한다.

⇒ 주말과 평일을 구분하면 좋은 예측이 가능할 것.

구분별 가스공급량



- 가스 공급사에 따라 공급량이 많은 차이를 보인다.

⇒ 공급량을 예측하는데 중요한 특성이 될 것.

대한민국 평균 기온 분석

출처: 기상청 기상자료개방포털 링크

https://public.tableau.com/app/profile/kimjinyeon/viz/_16347136091800/sheet0

https://public.tableau.com/views/20132018/sheet1?:showVizHome=no&:embed=true

분석 내용

월별 평균기온

⇒ 기온 그래프: A자 그래프 확인 가능

⇒ 2013 - 2018년 중 다른 년도 대비 이상치를 보이는 연도 X

분기별 평균기온

⇒ 분기별 가스 공급량 합계와 반대 모습(U↔A) 확인 가능

채택하지 않은 가설

천연가스 가격 상승 시, 가스 공급량 감소 예상

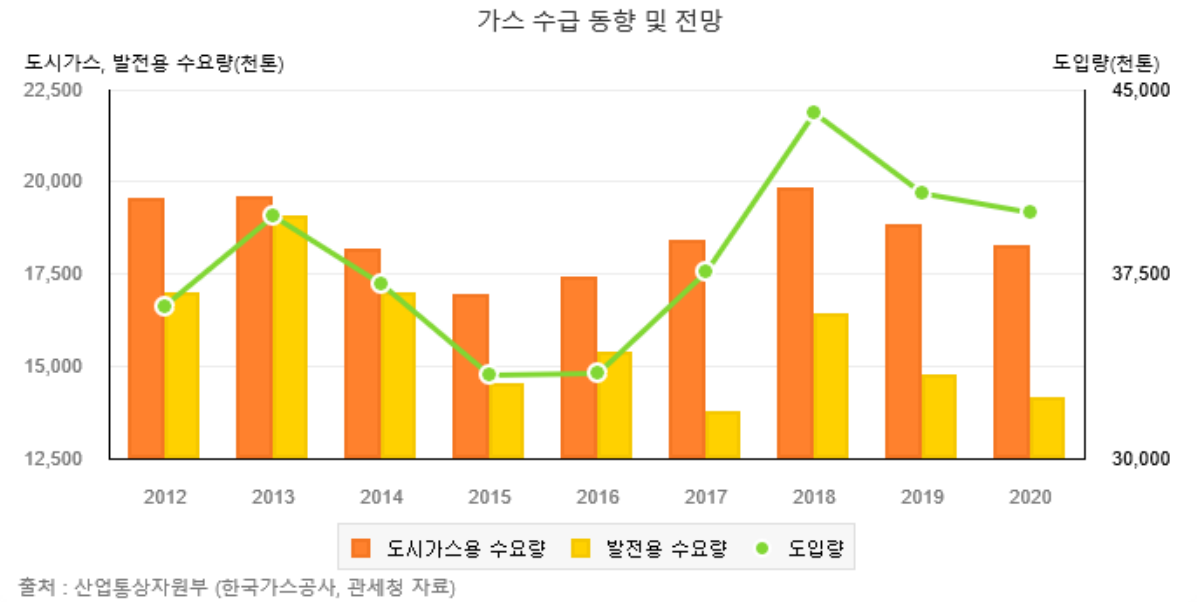
국제 천연가스 가격 상승 ⇒ 가스 공급에 차질 발생 가능 ⇒ 가스 공급량 감소

채택하지 않은 이유

e-나라지표 산업통상자원부 산하 한국가스공사 LNG 수급동향 정보

e-나라지표 지표조회상세

https://www.index.go.kr/potal/main/EachDtlPageDetail.do?idx_cd=1165



천연가스(LNG) 수급 동향 지표의 의의 및 활용도

- LNG 수요는 청정 고급 에너지의 선호에 따라 매년 큰 폭 증가

  ∴ LNG 중장기 도입계약 등을 통한 공급 안정성 확보 필수

- 연도별 물량의 변화

  ⇒ 당해년도 LNG 수요는 당해년도 LNG 공급가능물량(기초재고 + 도입물량)으로 충당

  [유의사항] 당해년도 LNG 수요량과 도입량 간 차이 발생

LNG 수급은 중장기 도입계약으로 이루어지며, 공급량은 당해년도 공급가능물량에 기반하고 있다.

LNG 수요량과 도입량에는 차이 발생 가능

> 💡 **국제시장에서 변동되는 천연가스 가격과 가스 공급량은 큰 관련성 X**

기온 데이터 수집하기

- 분기별 가스 공급량의 변화와 기온의 변화가 음의 상관관계를 가지며, 모델의 성능을 높이고자 기온 데이터 추가.

시간별 기온 데이터 수집하기

기상청 기상자료개방포털에서 제공하는 자료는 일별 기온 자료만 제공한다.

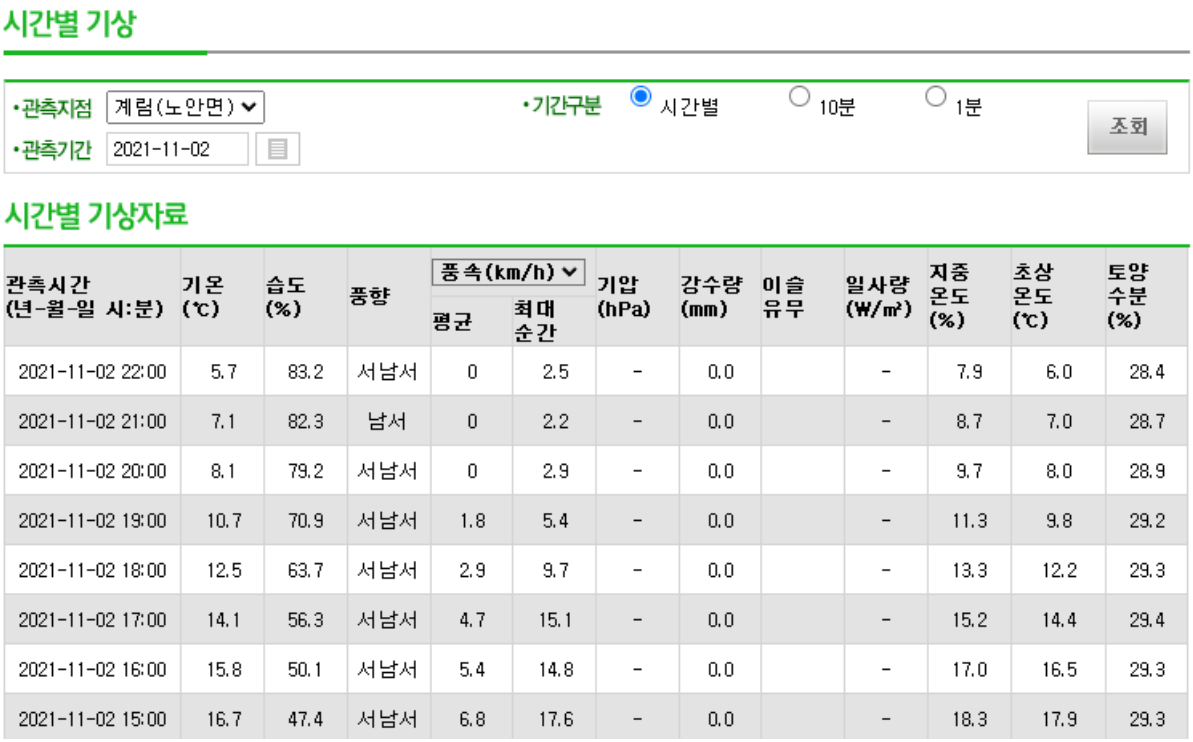
시간별 기온 자료는 따로 수집하여 전처리 할 필요가 있었다.

시간별 기온 데이터 출처: 나주시 농업기상정보시스템

:::나주시 농업기상정보시스템:::

나주시농업기상정보시스템

https://weather.naju.go.kr/agri_meteo/agri_time.html



시간별 기상

•관측지점 계림(노안면) •기간구분 시간별 10분 1분 조회

•관측기간 2021-11-02

시간별 기상자료

관측시간 (년-월-일 시:분)	기온 (℃)	습도 (%)	풍향	풍속(km/h) 평균	풍속(km/h) 최대순간	기압 (hPa)	강수량 (mm)	이슬 유무	일사량 (W/m²)	지중 온도 (%)	초상 온도 (℃)	토양 수분 (%)
2021-11-02 22:00	5.7	83.2	서남서	0	2.5	-	0.0		-	7.9	6.0	28.4
2021-11-02 21:00	7.1	82.3	남서	0	2.2	-	0.0		-	8.7	7.0	28.7
2021-11-02 20:00	8.1	79.2	서남서	0	2.9	-	0.0		-	9.7	8.0	28.9
2021-11-02 19:00	10.7	70.9	서남서	1.8	5.4	-	0.0		-	11.3	9.8	29.2
2021-11-02 18:00	12.5	63.7	서남서	2.9	9.7	-	0.0		-	13.3	12.2	29.3
2021-11-02 17:00	14.1	56.3	서남서	4.7	15.1	-	0.0		-	15.2	14.4	29.4
2021-11-02 16:00	15.8	50.1	서남서	5.4	14.8	-	0.0		-	17.0	16.5	29.3
2021-11-02 15:00	16.7	47.4	서남서	6.8	17.6	-	0.0		-	18.3	17.9	29.3

데이터 수집 및 전처리 과정 중 겪은 문제점

▼ 웹 크롤링 [코드]

▼ 주요 문제

- 사이트 날짜, 기온 데이터 결측치 존재
  - 해결1) 해당 날짜의 시간별 리스트 만들어 대조해서 확인
  - 해결2) 날짜가 없으면 해당 날짜 채우고, 기온이 없을 경우 NaN값으로 처리
- bs에 담은 자료에서 날짜, 기온의 데이터 인덱스 순서가 달라지는 문제
  - 해결1) 인덱스 규칙에 따라 함수 작성
  - 해결2) 조건문으로 적용

▼ 그 외 문제

- 달마다 날짜가 다른 문제
  - 윤년과 30, 31일 존재
    - 해결1) 특정 연, 월, 일 크롤링 함수 작성
    - 해결2) 조건문으로 적용

- 중복 날짜 데이터가 수집되고, 날짜가 내림차순 정렬되있는 문제
  - 해결) 날짜 데이터를 set으로 변환해 중복 제거, list로 변환해 오름차순 정렬
- 기온데이터에서 해당 날짜의 24시가 다음날 0시로 표현되는 문제
  - 해결) 해당 날짜의 마지막 0시를 하루 당겨 24시로 변경

▼ **데이터 전처리 [코드]**

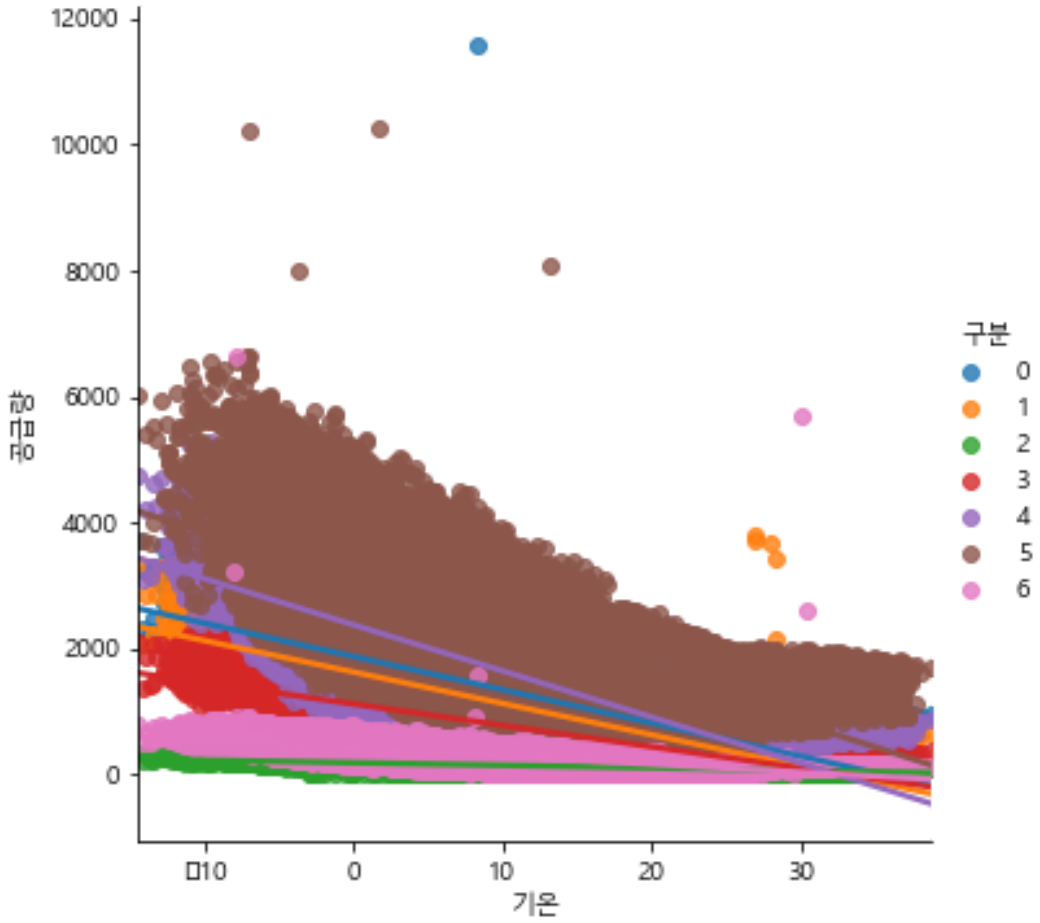
▼ **주요 문제**

- 특정 날짜, 오후, 오전 시간 때 기온 데이터 없는 문제
  - 해결) 같은 달, 일, 시간의 기온 평균으로 결측치 처리
- 매년 12월31일 24시 기온 데이터 없는 문제
  - 해결) 연도별 바로 전시간 23시 기온으로 결측치 처리

이슈 정리 표

▼ **데이터 분석 내용**

기온과 공급량 그래프



- 구분마다 이상치가 존재.
- 더 확실히 하기 위해 전체 데이터와 구분별 데이터 확인.

전체 데이터 정보

```
1 df.describe()
```

	시간	구분	공급량	year	month	day	weekday	기온
count	368088.000000	368088.000000	368088.000000	368088.000000	368088.000000	368088.000000	368088.000000	368088.000000
mean	12.500000	3.000000	948.100037	2015.500228	6.523962	15.726609	3.000000	13.385520
std	6.922196	2.000003	927.211578	1.707471	3.448424	8.798824	2.000003	10.434603
min	1.000000	0.000000	1.378000	2013.000000	1.000000	1.000000	0.000000	-14.500000
25%	6.750000	1.000000	221.973000	2014.000000	4.000000	8.000000	1.000000	4.600000
50%	12.500000	3.000000	637.014000	2016.000000	7.000000	16.000000	3.000000	13.900000
75%	18.250000	5.000000	1398.919000	2017.000000	10.000000	23.000000	5.000000	22.200000
max	24.000000	6.000000	11593.617000	2018.000000	12.000000	31.000000	6.000000	38.700000

- 공급량에서 이상치가 보임

구분0

```
df[df['구분']==0].describe()
```

	시간	구분	공급량	year	month	day	weekday	기온
count	52584.000000	52584.0	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000
mean	12.500000	0.0	1167.014483	2015.500228	6.523962	15.726609	3.000000	13.385520
std	6.922252	0.0	736.726376	1.707485	3.448452	8.798896	2.000019	10.434688
min	1.000000	0.0	114.097000	2013.000000	1.000000	1.000000	0.000000	-14.500000
25%	6.750000	0.0	628.659000	2014.000000	4.000000	8.000000	1.000000	4.600000
50%	12.500000	0.0	904.404000	2016.000000	7.000000	16.000000	3.000000	13.900000
75%	18.250000	0.0	1675.951000	2017.000000	10.000000	23.000000	5.000000	22.200000
max	24.000000	0.0	11593.617000	2018.000000	12.000000	31.000000	6.000000	38.700000

구분1

```
df[df['구분']==1].describe()
```

	시간	구분	공급량	year	month	day	weekday	기온
count	52584.000000	52584.0	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000
mean	12.500000	1.0	958.564621	2015.500228	6.523962	15.726609	3.000000	13.385520
std	6.922252	0.0	674.199480	1.707485	3.448452	8.798896	2.000019	10.434688
min	1.000000	1.0	67.973000	2013.000000	1.000000	1.000000	0.000000	-14.500000
25%	6.750000	1.0	469.506500	2014.000000	4.000000	8.000000	1.000000	4.600000
50%	12.500000	1.0	676.933000	2016.000000	7.000000	16.000000	3.000000	13.900000
75%	18.250000	1.0	1441.473000	2017.000000	10.000000	23.000000	5.000000	22.200000
max	24.000000	1.0	3798.613000	2018.000000	12.000000	31.000000	6.000000	38.700000

구분2

```
df[df['구분']==2].describe()
```

	시간	구분	공급량	year	month	day	weekday	기온
count	52584.000000	52584.0	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000
mean	12.500000	2.0	131.585481	2015.500228	6.523962	15.726609	3.000000	13.385520
std	6.922252	0.0	59.177756	1.707485	3.448452	8.798896	2.000019	10.434688
min	1.000000	2.0	1.378000	2013.000000	1.000000	1.000000	0.000000	-14.500000
25%	6.750000	2.0	94.978000	2014.000000	4.000000	8.000000	1.000000	4.600000
50%	12.500000	2.0	120.578000	2016.000000	7.000000	16.000000	3.000000	13.900000
75%	18.250000	2.0	171.778000	2017.000000	10.000000	23.000000	5.000000	22.200000
max	24.000000	2.0	358.654000	2018.000000	12.000000	31.000000	6.000000	38.700000

구분3

```
1 df[df['구분']==3].describe()
```

	시간	구분	공급량	year	month	day	weekday	기온
count	52584.000000	52584.0	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000
mean	12.500000	3.0	664.626452	2015.500228	6.523962	15.726609	3.000000	13.385520
std	6.922252	0.0	446.630583	1.707485	3.448452	8.798896	2.000019	10.434688
min	1.000000	3.0	36.196000	2013.000000	1.000000	1.000000	0.000000	-14.500000
25%	6.750000	3.0	326.131000	2014.000000	4.000000	8.000000	1.000000	4.600000
50%	12.500000	3.0	492.515000	2016.000000	7.000000	16.000000	3.000000	13.900000
75%	18.250000	3.0	996.105000	2017.000000	10.000000	23.000000	5.000000	22.200000
max	24.000000	3.0	2377.584000	2018.000000	12.000000	31.000000	6.000000	38.700000

구분4

```
1 df[df['구분']==4].describe()
```

	시간	구분	공급량	year	month	day	weekday	기온
count	52584.000000	52584.0	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000
mean	12.500000	4.0	1395.356673	2015.500228	6.523962	15.726609	3.000000	13.385520
std	6.922252	0.0	960.793541	1.707485	3.448452	8.798896	2.000019	10.434688
min	1.000000	4.0	108.101000	2013.000000	1.000000	1.000000	0.000000	-14.500000
25%	6.750000	4.0	683.440000	2014.000000	4.000000	8.000000	1.000000	4.600000
50%	12.500000	4.0	968.548000	2016.000000	7.000000	16.000000	3.000000	13.900000
75%	18.250000	4.0	2093.205000	2017.000000	10.000000	23.000000	5.000000	22.200000
max	24.000000	4.0	5301.451000	2018.000000	12.000000	31.000000	6.000000	38.700000

구분5

```
1 df[df['구분']==5].describe()
```

	시간	구분	공급량	year	month	day	weekday	기온
count	52584.000000	52584.0	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000
mean	12.500000	5.0	2070.685900	2015.500228	6.523962	15.726609	3.000000	13.385520
std	6.922252	0.0	1057.987317	1.707485	3.448452	8.798896	2.000019	10.434688
min	1.000000	5.0	355.413000	2013.000000	1.000000	1.000000	0.000000	-14.500000
25%	6.750000	5.0	1281.674750	2014.000000	4.000000	8.000000	1.000000	4.600000
50%	12.500000	5.0	1740.981000	2016.000000	7.000000	16.000000	3.000000	13.900000
75%	18.250000	5.0	2783.625000	2017.000000	10.000000	23.000000	5.000000	22.200000
max	24.000000	5.0	10271.437000	2018.000000	12.000000	31.000000	6.000000	38.700000

구분6

```
df[df['구분']==6].describe()
```

	시간	구분	공급량	year	month	day	weekday	기온
count	52584.000000	52584.0	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000	52584.000000
mean	12.500000	6.0	248.866646	2015.500228	6.523962	15.726609	3.000000	13.385520
std	6.922252	0.0	176.326173	1.707485	3.448452	8.798896	2.000019	10.434688
min	1.000000	6.0	2.756000	2013.000000	1.000000	1.000000	0.000000	-14.500000
25%	6.750000	6.0	122.196000	2014.000000	4.000000	8.000000	1.000000	4.600000
50%	12.500000	6.0	178.436000	2016.000000	7.000000	16.000000	3.000000	13.900000
75%	18.250000	6.0	371.345000	2017.000000	10.000000	23.000000	5.000000	22.200000
max	24.000000	6.0	6644.788000	2018.000000	12.000000	31.000000	6.000000	38.700000

- 확인 결과 각 구분마다 이상치 존재
- 이상치를 제거하거나 영향을 덜 받는 모델 선택.

머신러닝 모델 비교

사용한 머신러닝 모델

- LinearRegression, Lasso, Ridge, KNeighborRegression, DecesionTreeRegressor
- 앙상블 모델(RandomForest, XGBRegressor, LGBMRegressor)

머신러닝 모델 비교

Aa 머신러닝 모델	# 결정계수	# 교차검증 결정계수 평균
LinearRegression	0.038	0.037
Lasso(alpha=0.01)	0.034	0.034
Ridge(alpha=0.0001)	0.034	0.034
KNeighborRegression	0.773	0.664
DecesionTreeRegressor	0.983	0.958
RandomForest	0.99	0.976
XGBRegressor	0.983	0.982
LGBMRegressor	0.971	0.97

기온 데이터 추가하기

2019년 기온 데이터를 머신러닝 모델로 추가,이를 기반으로 다시 공급량을 추정하는 모델 작성 코드

1. `XGBRegressor` 를 이용하여 2019년 기온 데이터 예측
    a. 기온 데이터 정확도: `0.951`
2. `XGBRegressor` 를 이용하여 공급량 추정하는 모델 작성 후 검증

머신러닝 모델 비교(기온 데이터 추가)

Aa 머신러닝 모델	# 결정계수	# 교차검증 결정계수 평균	# NMAE
LinearRegression	0.268	0.268	
Lasso(alpha=0.01)	0.263	0.263	4.53
Ridge(alpha=0.01)	0.263	0.263	4.53
KNeighborRegression	0.613	0.468	
DecesionTreeRegressor	0.967	0.951	
RandomForest	0.984	0.976	0.554
XGBRegressor	0.986	0.984	0.488
LGBMRegressor	0.979	0.978	
CatBoostRegressor	0.964	0.962	0.555

하이퍼파라미터 튜닝하기

```
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [3, 6, 9, 12],
    'min_samples_split': [0.01, 0.05, 0.1]
}

kf = KFold(random_state=30,
           n_splits=10,
           shuffle=True,
          )

grid_search = GridSearchCV(estimator=model_random,
                           param_grid=param_grid,
                           cv=kf,
                           n_jobs=-1,
                           verbose=2
                          )

grid_search.fit(X_train, y_train)
```

```
print(grid_search.best_params_)
### {'max_depth': 12, 'min_samples_split': 0.01, 'n_estimators': 200}
```

하이퍼파라미터 튜닝 후

Aa 머신러닝 모델	기온	튜닝	# 결정계수	# 교차검증 결정계수 평균
RandomForest	미포함	O	0.923	0.927
RandomForest	포함	O	0.923	0.927
RandomForest	미포함	X	0.99	0.976
RandomForest	포함	X	0.984	0.976

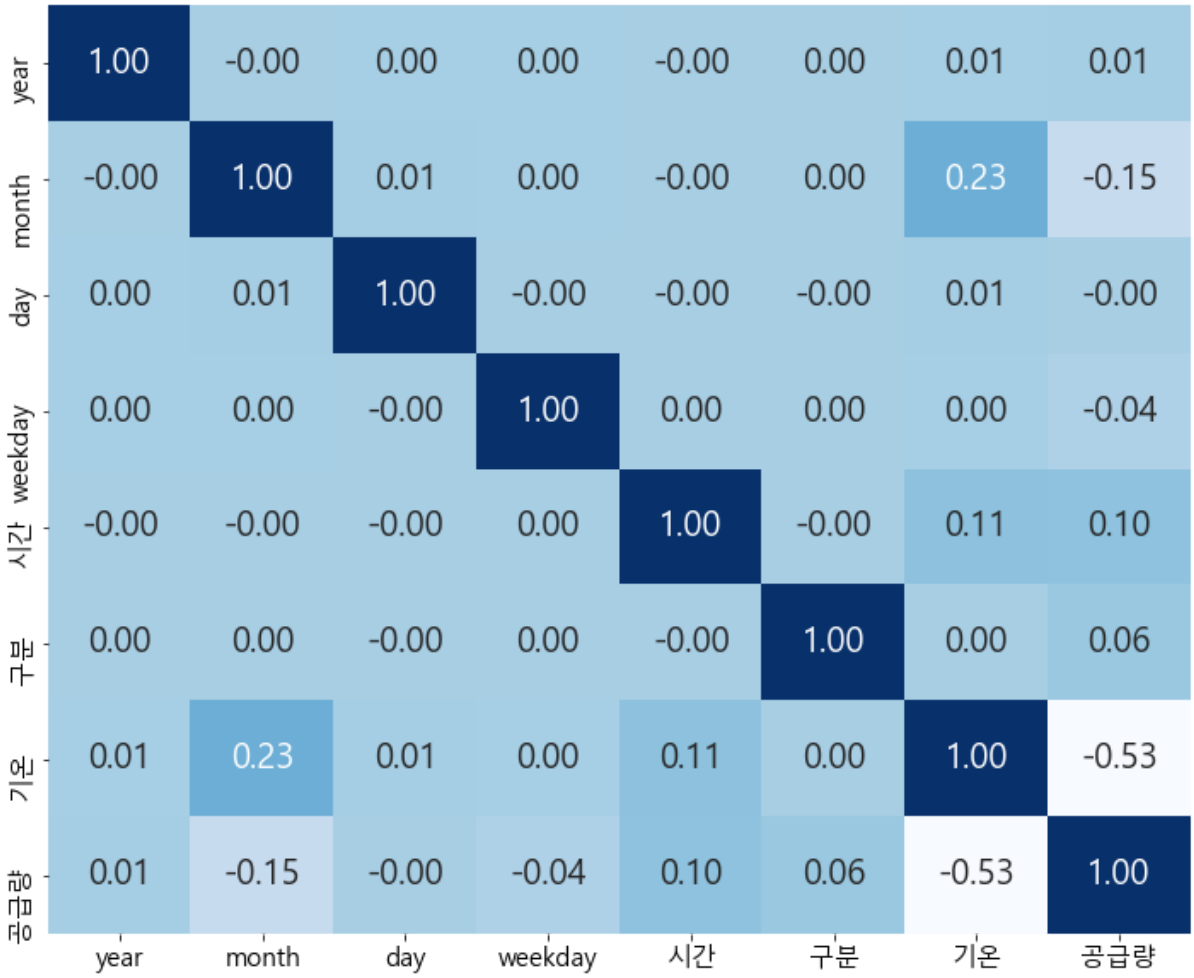
하이퍼파라미터 조정 후 결정계수가 낮아지는 결과, 하지만

> **실제로 제출해보니 점수는 0.1653877251로 시도해 본 것 중 가장 높았다.**

왜 결정계수가 낮은 모델이 더 높은 점수를 얻었을까?

💡 **기존에 시도하던 머신러닝 모델들이 과대적합되었다.**

과대적합을 해결하기 위한 특성 선택 변경



- **공급량과 상관관계가 큰 특성만 선택**
  - 'month', '시간', '구분', '기온'
    - 모델 : RandomForestRegressor
    - 제출 점수 : **0.1181514058 (56등-2021.11.04)**
- **모든 특성 사용 시보다 자체 결정 계수 점수는 떨어졌지만 제출 점수는 상승.**
  - 적절한 특성 선택이 과대적합을 해소한 것으로 판단
  - 자체 결정계수 점수(교차검증) 감소
    - 전 : 0.97 ➡️ 후 : 0.95
  - Dacon 제출 점수 증가
    - 전 : 0.1653 ➡️ 후 : 0.1181

최종 발표까지 목표

☑ ~~기존 머신러닝 모델들의 과적합 문제 해결~~

⇒ Feature Selection 통해서 해결

☑ ~~RandomForest 외 다른 모델들 하이퍼파라미터 튜닝~~

⇒ HyperOpt를 이용한 XGBRegressor 하이퍼파라미터 튜닝

☑ ~~polynomialfeatures 이용~~

⇒ 적용 후 결정 계수 확인 및 제출

☑ ~~딥러닝 모델로 학습~~

⇒ 시간 부족으로 제외

☑ ~~기온 데이터 외에 적용 가능한 외부 데이터 적용~~

⇒ 시간 부족으로 제외

여러 모델 결과 합치기

▼ 시도 배경

- 대회 상위 랭커들의 방법 중 여러 모델 예측의 평균으로 제출하는 방법 시도
- 결과가 좋은 모델의 예측 평균으로 시도

▼ 각 모델 제출 점수

- 선택 특성 : 'month', '시간', '구분', '기온'
- `RandomForestRegressor` : 0.1181514058 (56등-2021.11.04)
- `CatBoostRegressor` : 0.1201692248 (66등-2021.11.08)

▼ 제출 점수 : **0.1141754311 (52등-2021.11.09)**

- 예측에서 벗어난 값들이 평균으로 상호 보완되어 점수 향상된 것으로 판단

PolynomialFeatures(다항 특성) 적용

▼ 시도 배경

- 예측값과 연관있는 특성을 확대하여, 성능을 높이고자 시도

▼ 과정

- 선택 특성 : 'month', '시간', '구분', '기온'

- polynomial특성(degree=2, include_bias=False)

▼ 기온 예측 결정계수(교차검증)

- `RandomForestRegressor` : 0.8717(전)→ 0.9531(후) (증가)
- `CatBoostRegressor` : 0.8744(전) → 0.8738(후) (소폭 감소)

▼ 가스공급량 결정계수(교차검증)

- `RandomForestRegressor`: 0.9522(전) → 0.9531(후) (소폭증가)
- `CatBoostRegressor` : 0.9627(전) → 0.9626(후) (유사)

▼ 제출 점수 : 0.1151789265 (58등-2021.11.10)

- 불필요한 특성까지 확대되어 점수가 낮아진것으로 판단

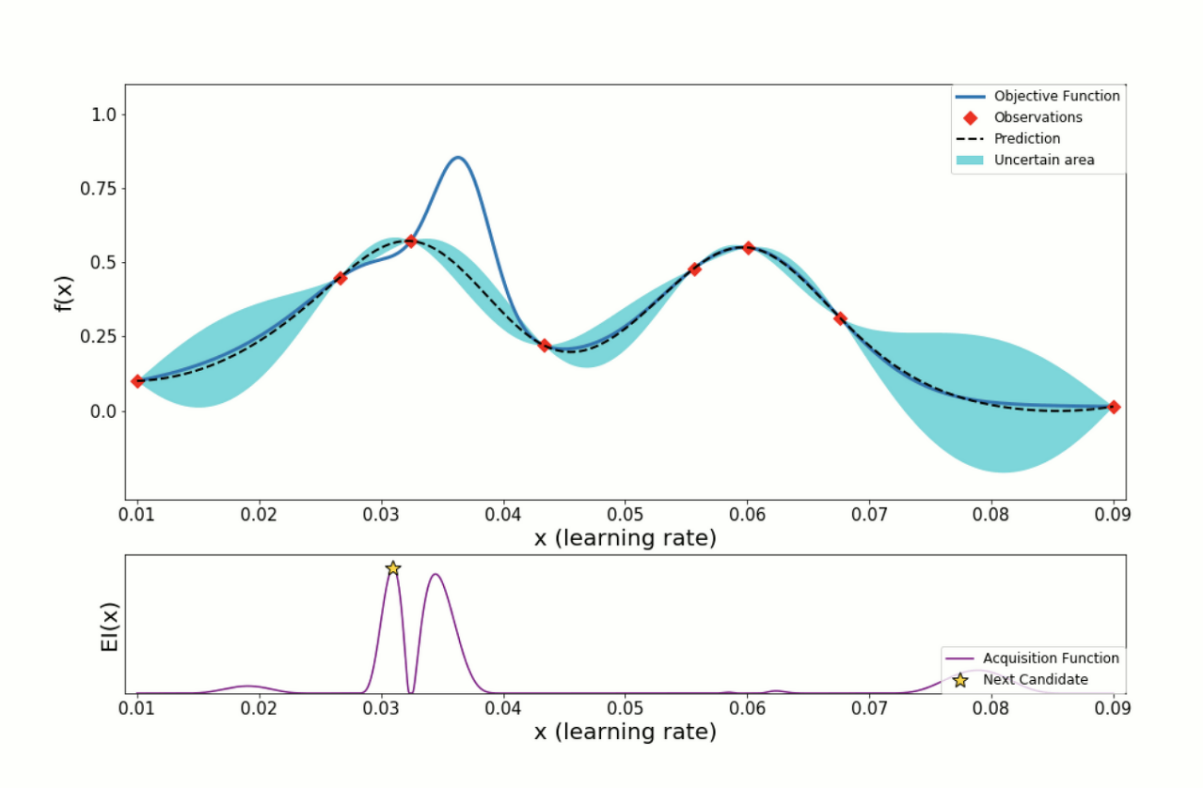
조건별 교차검증 점수

Aa 조건(특성 4개 : 'month', '시간', '구분', '기온')	기온	가스 공급량	# 제출 점수	제출날짜
RF+Cat(특성4개)	0.8717 / 0.8744	0.9522 / 0.9627	0.1141	2021-11-09
RF+Cat(특성4개, 표준화, 정규화, PolynomialFeatures, SelectPercentile 50% 적용)	0.8752 / 0.8753	0.1460 / 0.3535	1.137	2021-11-09
RF+Cat(특성4개, PolynomialFeatures 적용)	0.9531 / 0.8738	0.9531 / 0.9626	0.1151	2021-11-10
RF+Cat(특성4개+'weekday', PolynomialFeatures 적용)	0.8462 / 0.8751	0.9660 / 0.9720	0.1137	2021-11-11
RF+Cat(특성4개, log, PolynomialFeatures 적용)	0.8717 / 0.8738	0.8987 / 0.9197	0.1086	2021-11-11

HyperOpt를 이용한 하이퍼파라미터 튜닝

HyperOpt란?

베이지안 최적화에 기반하여 다음 파라미터 입력값을 추천받아 최적의 파라미터 조합을 찾아내는 하이퍼 파라미터 튜닝 라이브러리이다.



HyperOpt 공식 문서

Hyperopt Documentation

Install hyperopt from PyPI pip install hyperopt to run your first example # define an objective function def objective(args): case, val = args if case == 'case 1': return val else: return val ** 2 # define a search space from hyperopt import hp space = hp.choice('a', [('case 1',

http://hyperopt.github.io/hyperopt/

HyperOpt 시도 배경

중간발표까지 하이퍼파라미터 튜닝은 `GridSearch`를 이용했으나, 모든 경우의 수를 조사하는 `GridSearch` 특성상 시간이 너무 오래 걸렸다.

HyperOpt 도입 목표

- `GridSearch`보다 단축된 탐색 시간
- `GridSearch`와 비슷하거나 더 좋은 NMAE 지표
- 새로운 하이퍼파라미터 튜닝 방법 시도

HyperOpt 코드

```
from sklearn.metrics import mean_squared_error

def RMSE(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

### hp.quniform함수는 시작, 끝, step의 순서로 파라미터를 받아 실행한다.
### hp.quniform("파라미터명", 3, 12, 3)의 경우 3, 6, 9, 12와 동일하다.
space={'max_depth': hp.quniform("max_depth", 3, 12, 3),
       'subsample': hp.quniform('subsample', 0.6, 0.8, 0.1),
       'colsample_bytree' : hp.quniform('colsample_bytree', 0.3, 0.7, 0.1),
       'n_estimators': hp.quniform('n_estimators', 100, 250, 50)
      }

### 하이퍼파라미터 튜닝 함수(베이지안 최적화 목표 함수) 선언
def hyperparameter_tuning(space):
    model=XGBRegressor(n_estimators =int(space['n_estimators']),
                       max_depth = int(space['max_depth']),
                       subsample = space['subsample'],
                       colsample_bytree = space['colsample_bytree'],
                       random_state=777
                      )

    evaluation = [(X_train, y_train), (X_test, y_test)]

    model.fit(X_train, y_train,
              eval_set=evaluation,
              eval_metric="rmse",
              early_stopping_rounds=20,
              verbose=0)

    pred = model.predict(X_test)
    rmse= RMSE(y_test, pred)
    # 평가 방식 선정
    return {'loss':rmse, 'status': STATUS_OK, 'model': model}

# Trials 객체 선언합니다.
trials = Trials()
# best에 최적의 하이퍼 파라미터를 return 받습니다.
best = fmin(fn=hyperparameter_tuning,
            space=space,
            algo=tpe.suggest,
            max_evals=50, # 최대 반복 횟수를 지정합니다.
            trials=trials)

# 최적화된 결과를 int로 변환해야하는 파라미터는 타입 변환을 수행합니다.
best['max_depth'] = int(best['max_depth'])
best['n_estimators'] = int(best['n_estimators'])
print (best)

'''{'colsample_bytree': 0.7000000000000001,
 'max_depth': 12,
 'n_estimators': 250,
 'subsample': 0.8}'''
```

코드 참고 출처

결과

첫 번째 시도에서 **0.3133807529,**

튜닝 범위를 바꾼 두 번째 시도에서 **0.2746004941**로 좋지 않은 결과

603839	xgboost_hyperopt2.csv xgboost, 'month' '시간' '구분' '기온' 넣어서 hyperopt로 하이퍼 파라미터 튜닝 2번째 시도 edit	2021-11-08 17:02:08	0.2746004941
603815	xgboost_hyperopt.csv xgboost, 'month' '시간' '구분' '기온' 넣어서 hyperopt로 하이퍼 파라미터 튜닝 edit	2021-11-08 16:10:58	0.3133807529

왜 결과가 더 나빠졌을까?

점수가 낮아진 이유 추측

- 적절한 하이퍼 파라미터 선택 범위에서 벗어났다.
- 또는 하이퍼 파라미터 튜닝 과정이 충분하지 않았다.
- `GridSearch`가 느리지만 튜닝 성능이 더 뛰어나다.

PyCaret을 이용하여 모델 비교, 하이퍼파라미터 튜닝

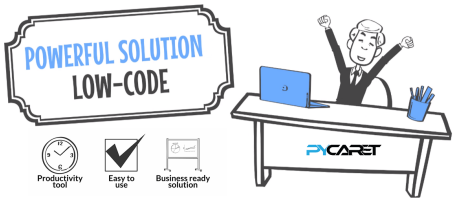
PyCaret이란?

오픈 소스, low-level 언어로 작성된 파이썬 머신러닝 라이브러리

Home - PyCaret

Whether its imputing missing values, transforming categorical data, feature engineering or even hyperparameter tuning of models, PyCaret automates all of it. It orchestrates the entire

https://pycaret.org/



PyCaret 시도 배경

`HyperOpt`가 `GridSearch`보다 빨랐지만 직접 코드를 작성해 하이퍼파라미터를 튜닝해야 했다.

`PyCaret`은 `HyperOpt`와 다르게 여러 머신러닝 모델을 비교하고 하이퍼파라미터를 튜닝하는 과정이 메소드로 구현되어 있다.

PyCaret 코드

```
### 데이터 나누기
data = total.sample(frac=0.9)
data_unseen = total.drop(data.index)
data.reset_index(drop=True, inplace=True)
data_unseen.reset_index(drop=True, inplace=True)

print('Data for Modeling: ' + str(data.shape))
print('Unseen Data For Predictions: ' + str(data_unseen.shape))

### pycaret setup
exp_reg101 = setup(data = data, target = '공급량', session_id=123, use_gpu = True)

### 제공하는 모든 모델을 비교하여 좋은 성능을 보이는 순서대로 표기
best = compare_models(exclude = ['ransac'])

### 모델 생성
lightgbm = create_model('lightgbm')
knn = create_model('knn')

### 모델 튜닝
tuned_lightgbm = tune_model(lightgbm)
tuned_knn = tune_model(knn)

### 모델 생성 마무리
final_lightgbm = finalize_model(tuned_lightgbm)
final_knn = finalize_model(tuned_knn)

### 내부 test 데이터로 검증
from pycaret.utils import check_metric

unseen_predictions_lightgbm = predict_model(final_lightgbm, data=data_unseen)
unseen_predictions_knn = predict_model(final_knn, data=data_unseen)

check_metric(unseen_predictions_lightgbm.공급량, unseen_predictions_lightgbm.Label, 'R2')
check_metric(unseen_predictions_lightgbm.공급량, unseen_predictions_lightgbm.Label, 'MAE')
check_metric(unseen_predictions_knn.공급량, unseen_predictions_knn.Label, 'R2')
check_metric(unseen_predictions_knn.공급량, unseen_predictions_knn.Label, 'MAE')

### submission 예측, X_sub는 submission 열을 나누고 기온 추가한 DataFrame
pred_lightgbm = predict_model(final_lightgbm, data=X_sub)
pred_knn = predict_model(final_knn, data=X_sub)

### 두 모델의 평균을 공급량으로 취한다
sub["공급량"] = (pred_lightgbm['Label'] + pred_knn['Label']) / 2
```

PyCaret Regressor Tutorial 참고

결과

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lightgbm	Light Gradient Boosting Machine	110.6273	33283.6340	182.3743	0.9613	0.3747	0.5623	1.240
knn	K Neighbors Regressor	116.4040	40355.7594	200.8320	0.9531	0.3563	0.5546	2.679
rf	Random Forest Regressor	119.7998	41994.5683	204.8871	0.9512	0.3596	0.5535	64.078
et	Extra Trees Regressor	130.0865	49843.7558	223.2207	0.9421	0.4032	0.5691	60.800
gbr	Gradient Boosting Regressor	153.5424	52784.6388	229.7182	0.9386	0.5192	0.7187	15.146
dt	Decision Tree Regressor	139.4984	58001.3100	240.8098	0.9326	0.4370	0.5765	0.898
lr	Linear Regression	332.4068	190083.3344	435.9771	0.7790	0.8354	2.4901	0.143
ridge	Ridge Regression	332.4035	190083.3328	435.9770	0.7790	0.8354	2.4899	0.096
lar	Least Angle Regression	332.4088	190083.3754	435.9771	0.7790	0.8354	2.4903	0.083
br	Bayesian Ridge	332.4041	190083.3302	435.9770	0.7790	0.8354	2.4899	0.309
lasso	Lasso Regression	331.6280	190627.3453	436.6008	0.7784	0.8316	2.4056	0.225
huber	Huber Regressor	319.1961	201916.4701	449.3392	0.7653	0.7567	1.6373	6.669
par	Passive Aggressive Regressor	416.7429	291146.9256	527.3798	0.6613	0.9874	3.4200	0.408
ada	AdaBoost Regressor	473.0746	354889.9254	595.2883	0.5874	1.0728	4.0457	5.072
omp	Orthogonal Matching Pursuit	516.3310	445992.3599	667.8176	0.4815	1.1608	3.5957	0.073
en	Elastic Net	511.9236	470554.6156	685.9571	0.4529	1.0507	3.4746	0.105
llar	Lasso Least Angle Regression	719.9870	857624.9777	926.0672	0.0029	1.3181	6.8854	0.075

lightgbm, knn이 MAE 110.6273, 116.4040으로 가장 좋은 결과를 보였다.

`PyCaret`에서 NMAE를 따로 지원하지 않아, MAE로 비교하였다.

위의 결과에 따라서 `lightgbm`과 `knn`모델만 하이퍼파라미터를 튜닝하고 결과를 예측하였다.

MAE, R2 비교표

Aa 이름	# MAE	# R2
finalized lightgbm	107.4415	0.962
finalized knn	111.8792	0.9561

lightgbm, knn 하이퍼파라미터 튜닝 후 제출

`lightgbm`과 `knn` 하이퍼파라미터 튜닝 후, 각각 예측하여 예측값의 평균 제출

💡 여러 모델 결과 합치기가 성능이 좋았으니, PyCaret에도 적용

제출 결과

제출 점수: **0.111891712**(46등 2021-11-10)으로 좋은 결과

모든 결과 종합

여러 모델 결과 합치기에서 사용했던 `RandomForest`와 `Catboost` (**0.1141754311**)

PyCaret에서 사용했던 `lightgbm`과 `knn` (**0.111891712**)

4 가지 모델 결과를 합쳐서 제출

제출 결과

제출 점수: **0.1110990574**(**46등 2021-11-10**)으로 PyCaret의 **0.111891712**에 비해 소폭 상승

결론

프로젝트 중 점수를 올릴 수 있었던 계기 3가지

좋은 특성을 선택하는 것

중요한 것과 중요하지 않은 것을 구별해낼 수 있는 **도메인 지식**이 중요

도메인 지식을 바탕으로 Feature Engineering을 철저히 하지 않으면 좋은 결과를 얻을 수 없다.

또한 특성 선택에 따라서 좋은 성과를 내는 모델이 달라졌다.

> ⇒ 모든 특성을 대상으로 하면 `RandomForest`, `Extra Tree`가 좋은 성능을 보였다.
>
> 트리 성장 절차 중에 영향이 낮은 특성을 걸러낸 것이 좋은 영향을 주었다고 판단.
>
> ⇒ 일부 특성을 대상으로 하면 `lightgbm`, `knn`이 좋은 성능을 보였다.
>
> 영향이 낮은 특성을 미리 걸러냄으로써 `RandomForest`, `Extra Tree`보다 좋은 성능을 보인 것으로 판단.

모델 조정시 경험과 꼼꼼함 중요

어떤 파라미터를 튜닝할지, 어느 범위 안에서 튜닝할지 정하는 것이 어려웠다.

수치에 기반하지 않고, 적당히 정할 경우 **점수가 크게 떨어지기 쉬웠다.**

어떤 파라미터가 민감한 파라미터인지, 어느 정도 범위 안에서 튜닝해야 하는지 알아야 잘 할 수 있다.

관련 경험을 충분히 쌓아야 하는 분야라고 판단된다.

다양한 모델의 결과 종합으로 예측 점수 향상 가능

한 가지 모델에 의존하기보다, **다양한 모델을 시도해보고 결과치를 종합**해보는 것이 의미있다.

여러 모델을 종합하여 잔차를 최소화함으로써 점수를 더 올릴 수 있었다.

으쌰으쌰 자체 평가

팀장 가채원

- 각자 맡은 역할에 너무나 최선을 다해 주었다.
- 팀원들의 학습 및 프로젝트에 대한 능동성, 적극성이 긍정적으로 발현되었다.
- IT 분야 굵직한 대회에 참가할 수 있는 좋은 기회였고, 팀원 간 상호 교류를 통해 성장할 수 있는 기회가 되었다.

부팀장 김범중

- 다양한 자료는 아니지만 핵심적인 자료를 활용한 점이 좋았다.
- 자료를 모으고 분석하면서 수업에서 배운 많은 내용을 다시 확인하고 단단히 할 수 있었다.
- 팀원들이 서로 다양한 아이디어와 기술을 보고 배울 수 있어 유익했다.

팀원 김진연

- 팀원들이 의견 제시와 새로운 의견 수용에 적극적이어서 팀 프로젝트가 액션 중심으로 활발하게 이루어졌다.
- 단순히 프로젝트를 분업하고 분업한 작업을 합치는 것이 아니라, 서로의 결과물에 대해서 토의하고 보완해나가는 일이 잘 이루어져서 팀으로서 시너지가 발휘되었다.
- 팀으로서 일하는 법, 머신러닝 대회에서의 노하우 두 가지를 많이 배워갈 수 있었다.

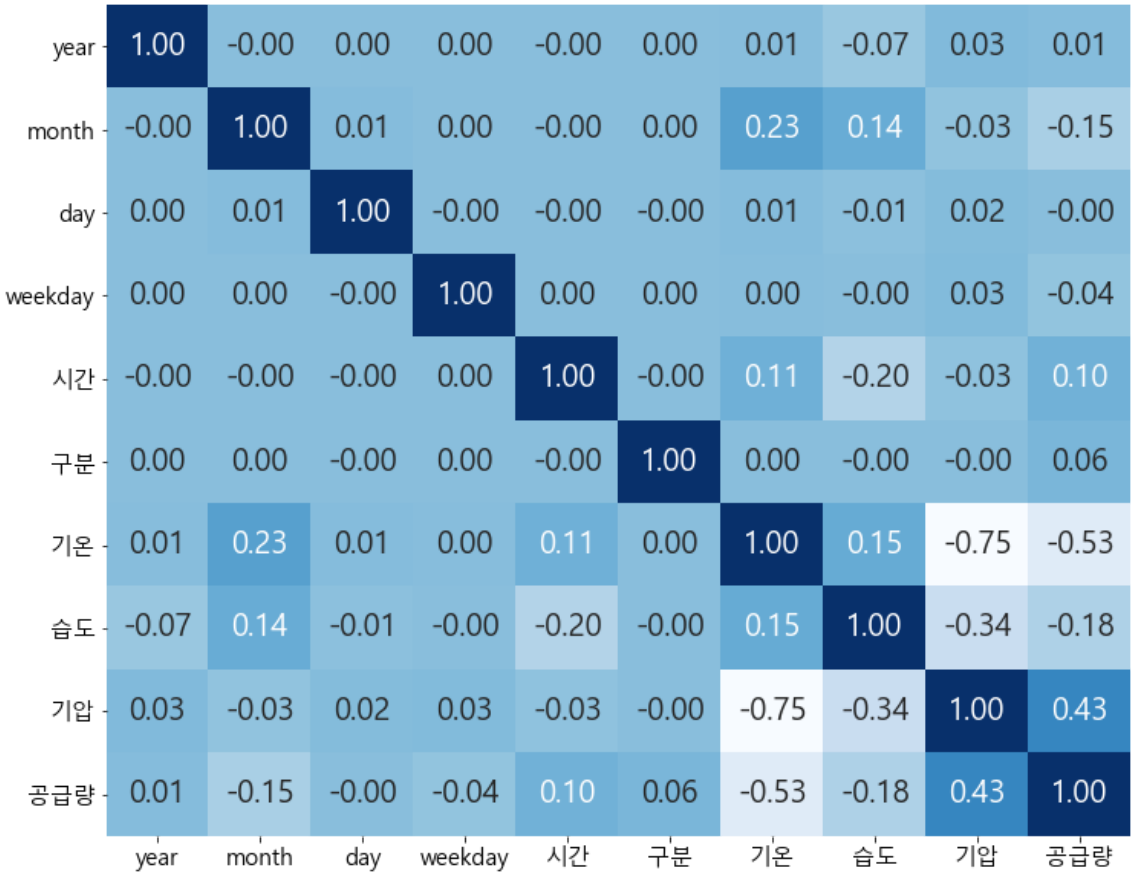
팀원 윤진훈

- 데이콘이라는 의미 있는 대회의 분위기를 알 수 있었고, IT 쪽은 코드 공유의 문화가 권장된다는 걸 확인할 수 있었다.
- 협업의 시너지가 중요하다는 걸 체감할 수 있었다.
- 팀에 기여를 못한 것 같아서 팀원 분들께 죄송하다. 기본이 많이 부족함을 느꼈다.

발표 후 자료 정리

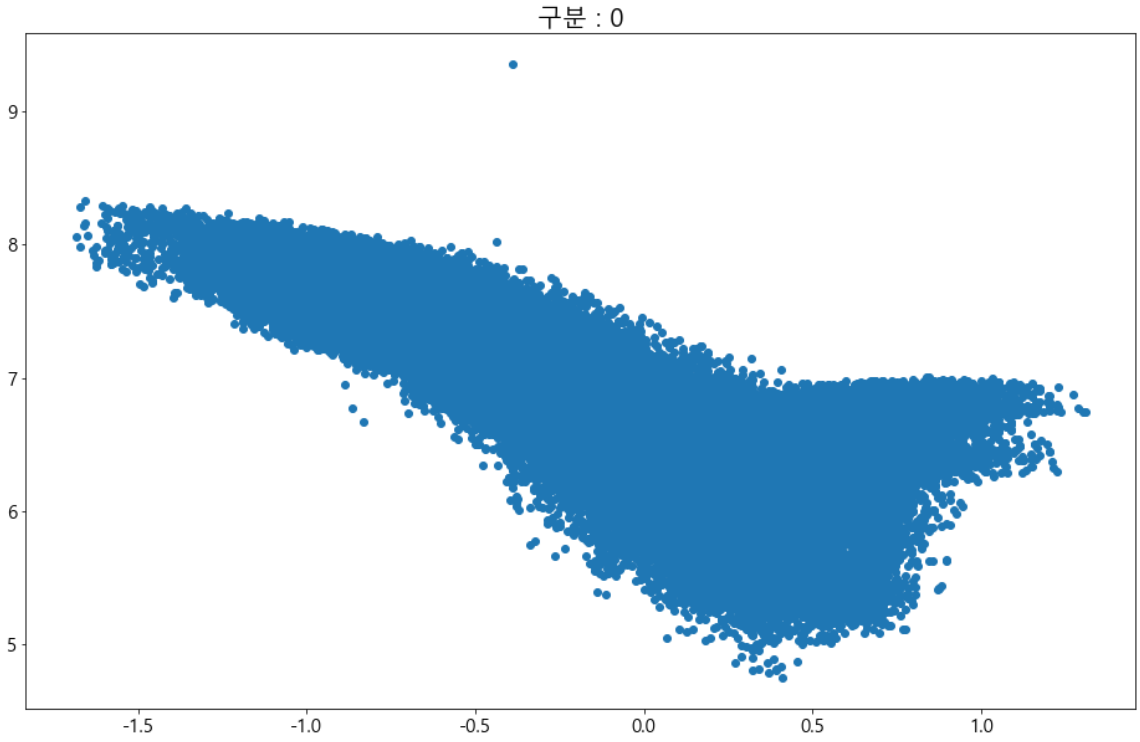
▼ 자료 분석

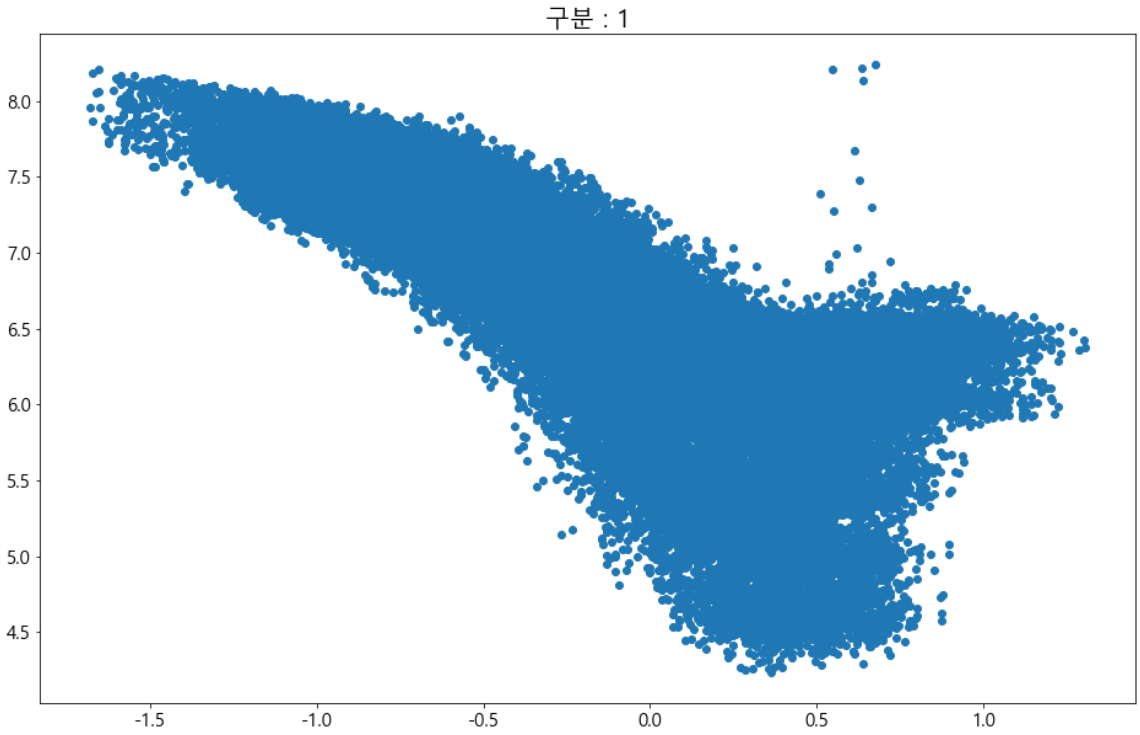
외부데이터와 기존 데이터 상관계수

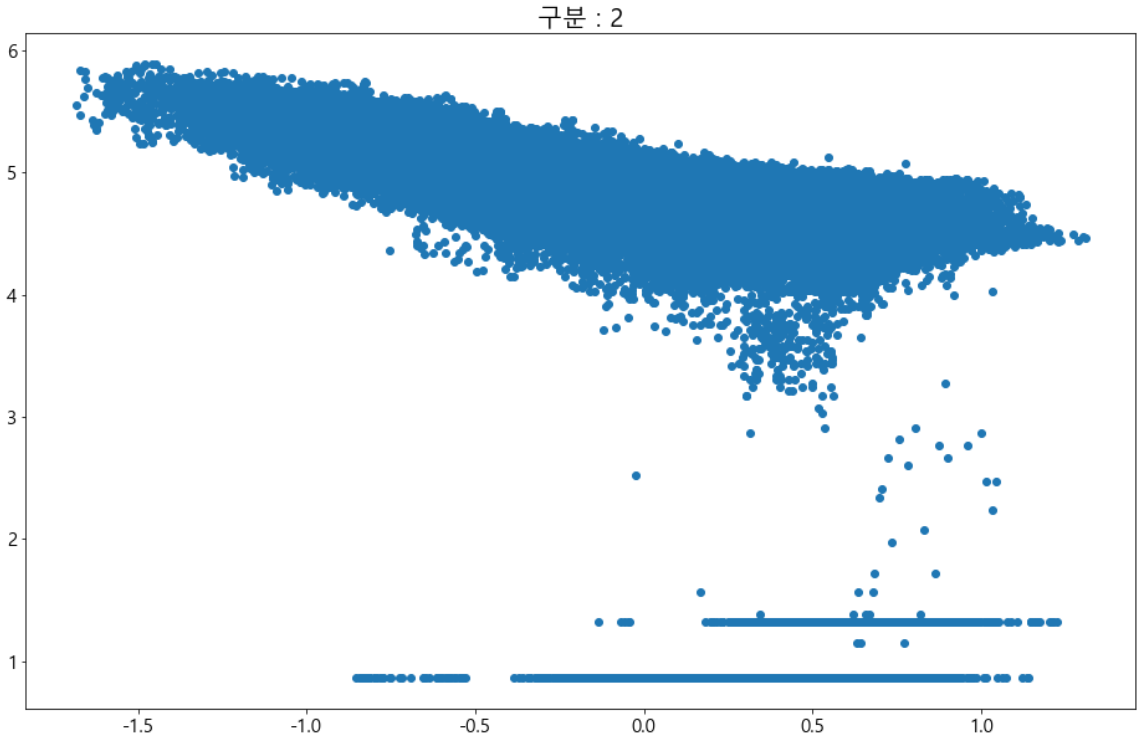


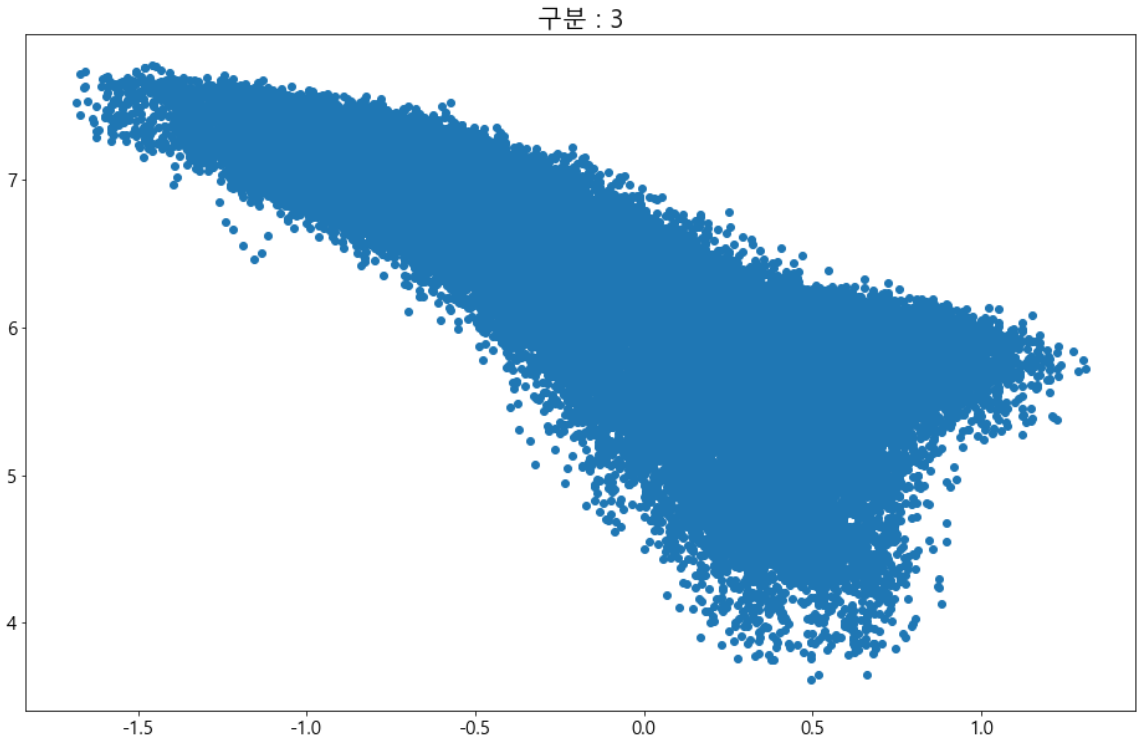
공급량과 상관계수가 높은 특성 : 기온(-0.53), 기압(0.43), 습도(-0.18)

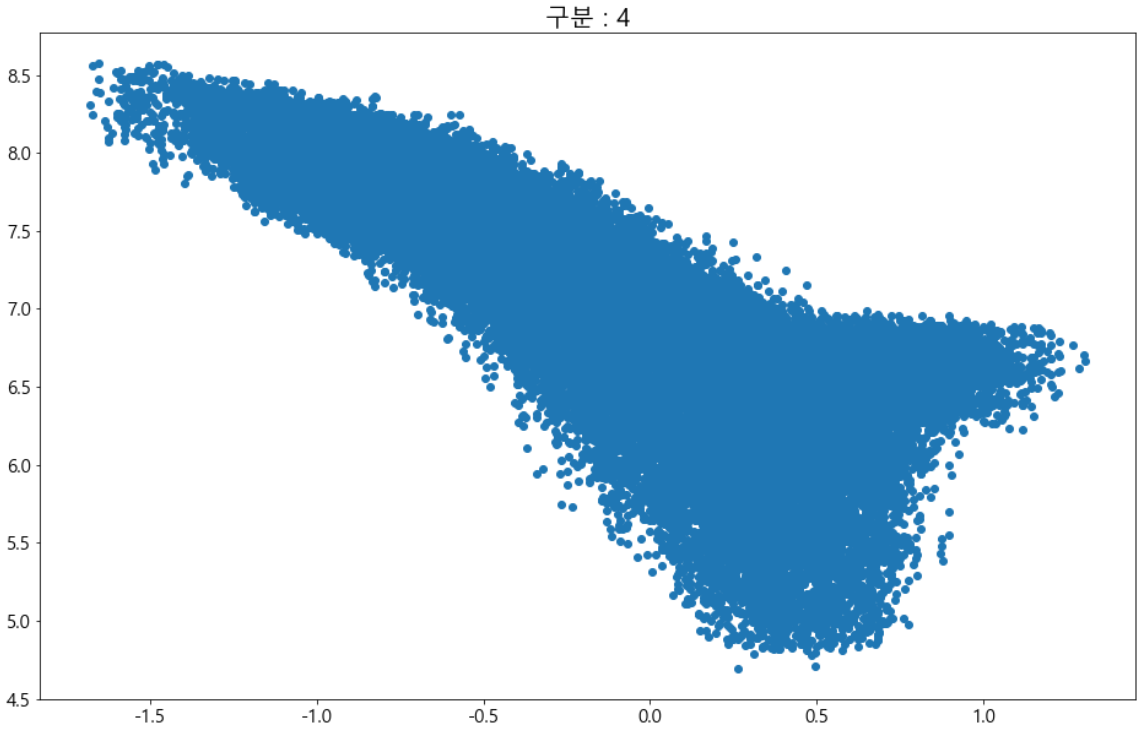
Robust한 기온과 log적용한 공급량 산점도

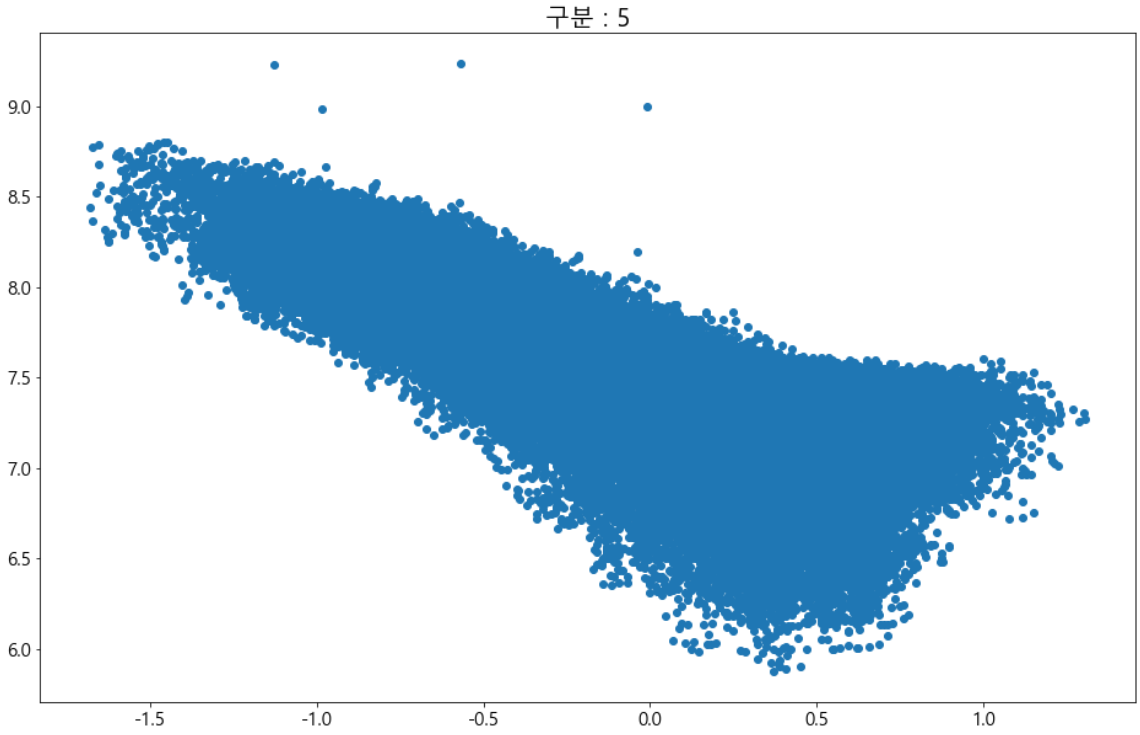

구분 : 0
9
8
7
6
5
-1.5
-1.0
-0.5
0.0
0.5
1.0

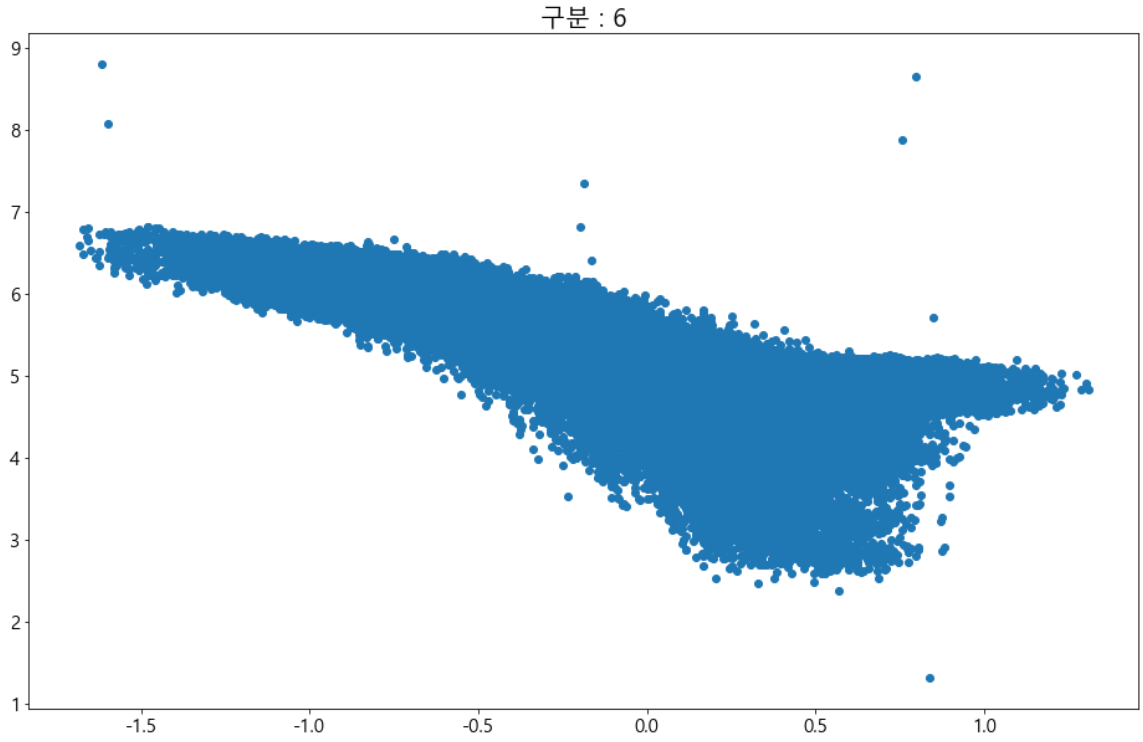

구분 : 1
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
-1.5
-1.0
-0.5
0.0
0.5
1.0


구분 : 2
6
5
4
3
2
1
-1.5
-1.0
-0.5
0.0
0.5
1.0


구분 : 3
7
6
5
4
-1.5
-1.0
-0.5
0.0
0.5
1.0


구분 : 4
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
-1.5
-1.0
-0.5
0.0
0.5
1.0


구분 : 5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
-1.5
-1.0
-0.5
0.0
0.5
1.0



각 구분마다 이상치가 존재. 그 중 구분2의 이상치가 가장 많아보임.

```
1  # 전체
2  df[df['공급량'] < 3]
```

	연월일	year	month	day	weekday	시간	구분	공급량	기온	습도	기압	공급량차	기온차	습도차	기압차
110535	2013-08-12	2013	8	12	0	16	2	1.378	32.2	53.0	1000.0	-52.00	0.3	-1.0	0.0
110536	2013-08-12	2013	8	12	0	17	2	1.378	31.7	57.0	1000.0	0.00	-0.5	4.0	0.0
110537	2013-08-12	2013	8	12	0	18	2	1.378	31.3	60.0	999.7	0.00	-0.4	3.0	-0.3
110538	2013-08-12	2013	8	12	0	19	2	1.378	30.3	67.0	999.7	0.00	-1.0	7.0	0.0
110539	2013-08-12	2013	8	12	0	20	2	1.378	29.2	71.0	1000.0	0.00	-1.1	4.0	0.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
155795	2018-10-11	2018	10	11	3	12	2	2.756	13.2	31.0	1012.6	0.00	1.5	-5.0	-0.4
155796	2018-10-11	2018	10	11	3	13	2	2.756	13.5	32.0	1012.0	0.00	0.3	1.0	-0.6
155797	2018-10-11	2018	10	11	3	14	2	2.756	13.4	32.0	1011.6	0.00	-0.1	0.0	-0.4
155798	2018-10-11	2018	10	11	3	15	2	2.756	13.0	39.0	1011.5	0.00	-0.4	7.0	-0.1
337022	2015-06-16	2015	6	16	1	15	6	2.756	30.4	47.0	993.7	-300.24	-0.2	2.0	-0.4

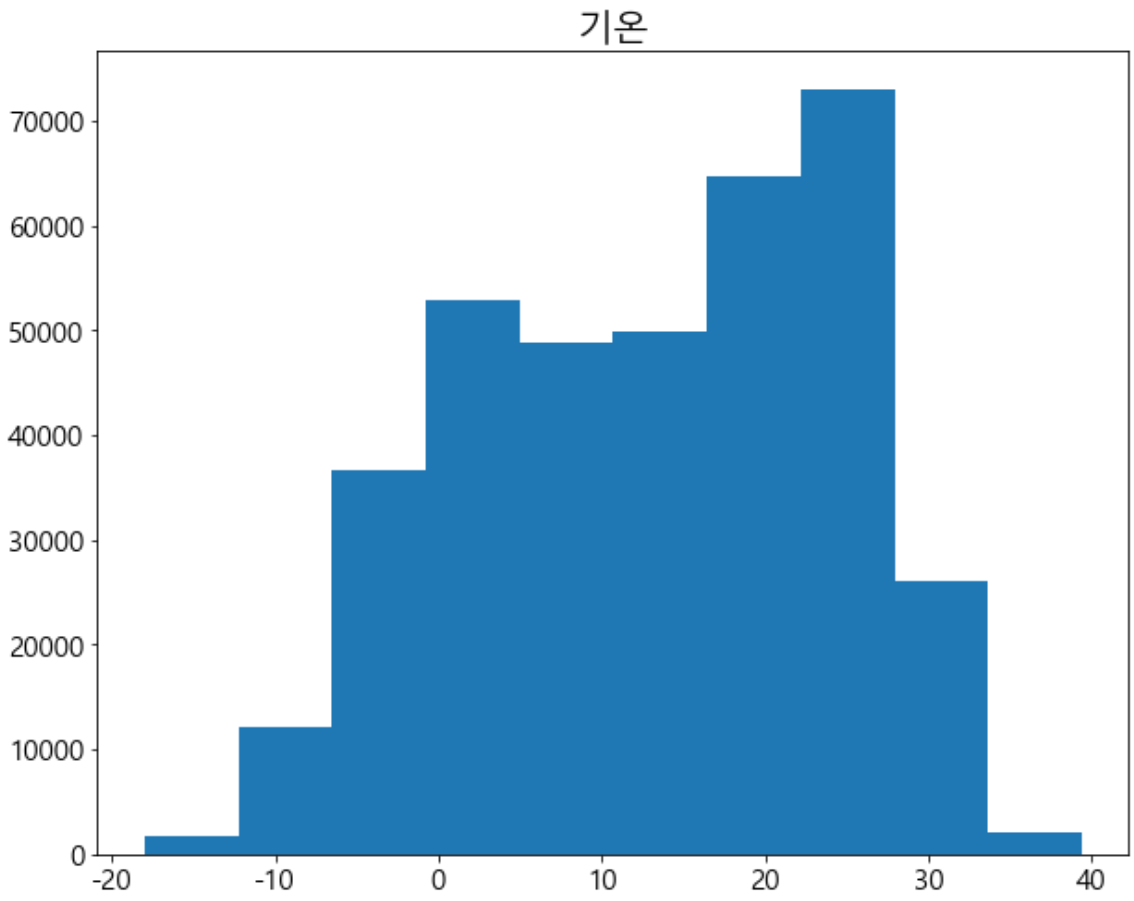
2756 rows × 15 columns

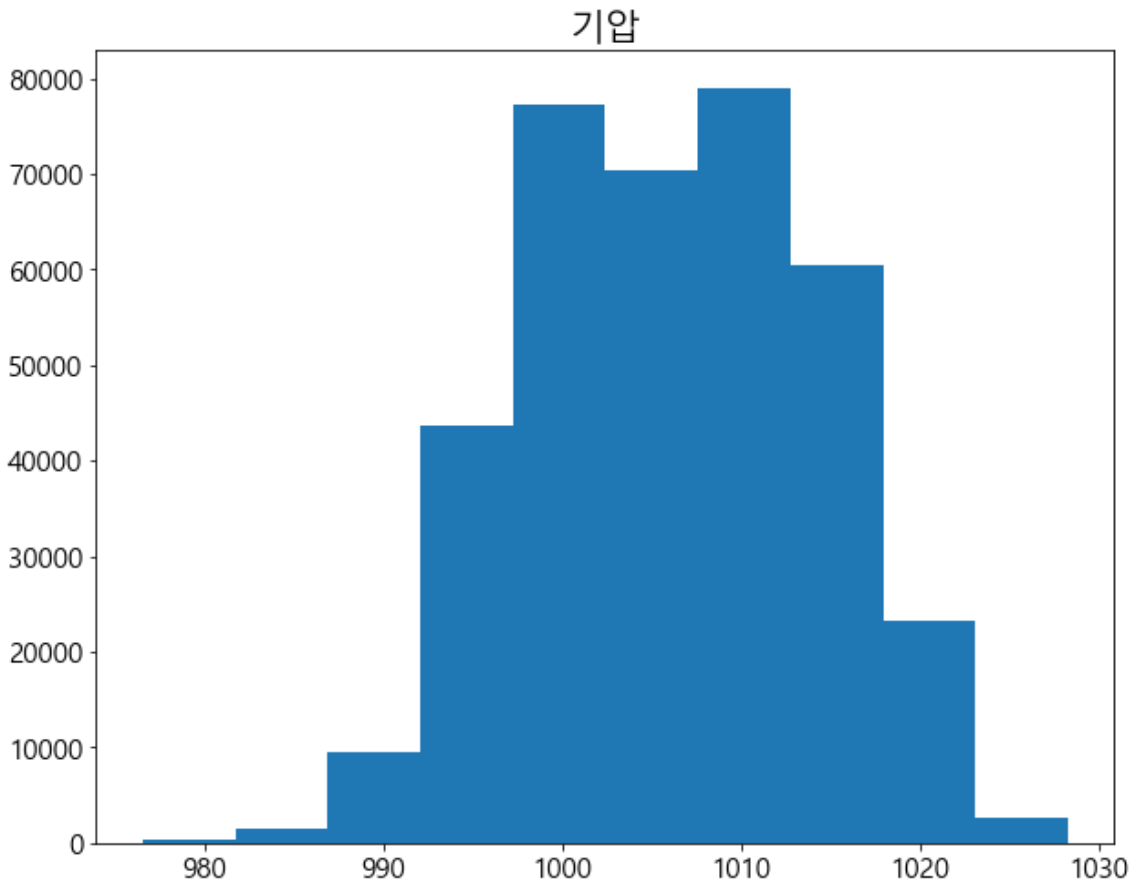
같은 값으로 채워진 이상치가 2700여개 존재. 이상치 처리함.

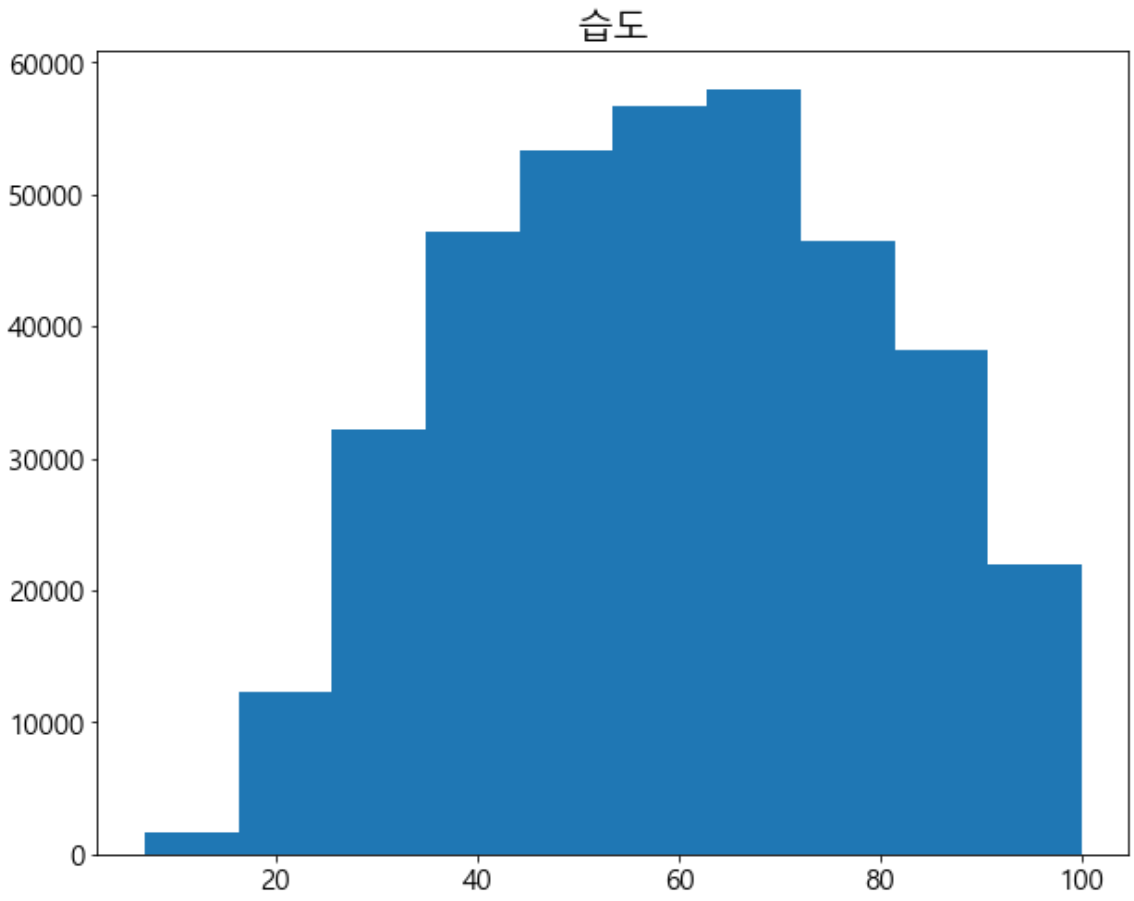
외부 데이터 분포 확인

분포상 표준정규분포 형태를 따르고 있어 다른 처리는 필요치 않아보임.

스케일 조정만 필요.


기온
70000
60000
50000
40000
30000
20000
10000
0
-20
-10
0
10
20
30
40


기압
80000
70000
60000
50000
40000
30000
20000
10000
0
980
990
1000
1010
1020
1030



▼ 사용 기술

▼ 전처리

공급량 이상치 제거 : 각 구분마다 반복되는 이상치 직접 제거. 동일한 숫자로 채워진 이상치 값이 2700여개가 넘어 특정 구분의 예측 정확도가 떨어지는 문제 발생.

RobustScaler : 기온, 기압, 습도 값의 자리수가 달라 스케일 조정 필요. 이상치에 영향 받지 않는 방법 적용.

target transformation : 훈련 데이터와 입력 데이터 모두 표준 정규화하여 모델의 예측 정확도를 향상시키고자 적용.

조건별 점수 기록

private 점수