

Likelion_ML_Project

☰ Github	https://github.com/Likelion-ML-Project/ML_Project
👤 이해관계자	명운 전명운 <small>민수</small> 김민수 <small>희경</small> Helena Jeong <small>규림</small> 이규림
🕒 작성일시	@July 16, 2021 3:52 PM
🕒 최종 편집일시	@August 5, 2021 10:47 PM

보노보노 없는 보노보노팀

팀원소개

Aa 이름	👤 태그	⌚ 캐릭터	☰ 역할
정희경(팀장)	희경 Helena Jeong	!	결측치 처리(지하철), 피처 생성, 발표
전명운(부팀장)	명운 전명운	!	결측치 처리(임대보증금, 임대료), 머신러닝 기술 지원
김민수	민수 김민수	!	결측치 처리(임대보증금, 임대료), 피처 생성 및 외부데이터 활용
이규림	규림 이규림	!	결측치 처리(지하철), 피처 생성 및 외부데이터 활용

▼ 🤝

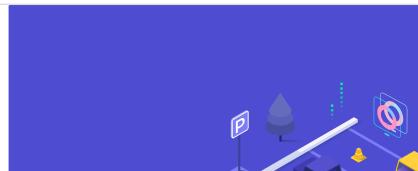


대회 자료 및 출처

주차수요 예측 AI 경진대회

출처 : DACON - Data Science Competition

<https://dacon.io/competitions/official/235745/data>



대회명: 주차수요 예측 AI 경진대회

- 웹 사이트 : <https://dacon.io/competitions/official/235745/data>
- 기간 : 2021.06.10 ~ 2021.07.30 18:00

주제

| 유형별 임대주택 설계 시 단지 내 적정 주차 수요를 예측

배경

| 아파트 단지 내 필요한 주차대수는 ①법정주차대수 ②장래주차수요 중 큰 값에 따라 결정하게되어 있어, 정확한 ②장래주차수요의 산정을 필요로 합니다. 현재 ②장래주차수요는 '주차원단위'와 '건축연면적'을 기초로하여 산출되고 있으며, '주차원단위'는 신규 건축예정 부지 인근의 유사 단지를 피크 시간대 방문하여 주차된 차량대수를 세는 방법으로 조사하고 있습니다. 이 경우 인력조사로 인한 오차발생, 현장조사 시점과 실제 건축시점과의 시간차 등의 문제로 과대 또는 과소 산정의 가능성을 배제할 수 없습니다.

DATA SET

- train.csv
- test.csv
- sample_submission.csv

- age_gender_info.csv

NULL값 처리

10분내 지하철수

▼ 전처리 과정 해결 1

▼ 그룹화 기준

```
all_df['10분내지하철수'].isnull().sum()
```

249

```
all_df.loc[ all_df['10분내지하철수'].isnull(), : ]
```

단지코드	총세대수	임대건물구분	지역	공급유형	전용면적	전용면적별세대수	공가수	자격유형	임대보증금	임대료	10분내지하철수	
86	C1312	518	아파트	충청남도	국민임대	39.72	60	12.0	A	17460000	122210	NaN
87	C1312	518	아파트	충청남도	국민임대	39.98	89	12.0	A	17460000	122210	NaN
88	C1312	518	아파트	충청남도	국민임대	41.55	225	12.0	A	19954000	130940	NaN
89	C1312	518	아파트	충청남도	국민임대	46.90	143	12.0	A	28687000	149660	NaN
90	C1874	619	아파트	충청남도	영구임대	26.37	294	2.0	C	3141000	69900	NaN
91	C1874	619	아파트	충청남도	영구임대	26.37	149	2.0	C	3141000	69900	NaN

```
grouped = all_df.groupby(['임대건물구분', '지역', '공급유형'])

group1 = grouped.get_group( ('아파트', '충청남도', '국민임대') )
group2 = grouped.get_group( ('아파트', '충청남도', '영구임대') )
group3 = grouped.get_group( ('아파트', '충청남도', '공공임대(50년') ) )
group4 = grouped.get_group( ('상가', '충청남도', '임대상가') ) 

group5 = grouped.get_group( ('아파트', '대전광역시', '국민임대') )
group6 = grouped.get_group( ('아파트', '대전광역시', '영구임대') )
group7 = grouped.get_group( ('아파트', '대전광역시', '공공임대(50년') ) )
group8 = grouped.get_group( ('아파트', '대전광역시', '공공임대(10년') ) )
group9 = grouped.get_group( ('아파트', '대전광역시', '공공분양') )
group10 = grouped.get_group( ('아파트', '대전광역시', '공공임대(분납') ) )
group11= grouped.get_group( ('상가', '대전광역시', '임대상가') )

group12 = grouped.get_group( ('아파트', '경상남도', '공공임대(10년') ) )
```

'임대건물구분', '지역', '공급유형'으로 그룹화해도 평균이 안나오는 값 (○ 표시)

→ 피쳐수를 줄여서 평균값 도출

```
grouped1 = all_df.groupby(['임대건물구분', '지역'])
group3 = grouped.get_group( ('아파트', '충청남도', '공공임대(50년') )
group7 = grouped.get_group( ('아파트', '대전광역시', '공공임대(50년') )
group9 = grouped.get_group( ('아파트', '대전광역시', '공공분양') )
```

```
grouped2 = all_df.groupby(['지역'])
group4 = grouped.get_group(('상가', '충청남도', '임대상가'))
```

▼ 문제 발생 사례

문제 1)

- fillna 함수를 사용

```
grouped = all_df.groupby(['임대건물구분', '지역', '공급유형'])
group1 = grouped.get_group(('아파트', '충청남도', '국민임대'))

group1['10분내지하철수'] = group1['10분내지하철수'].fillna(group1['10분내지하철수'].mean())
print(sum(group1['10분내지하철수'].isnull()))
print(all_df['10분내지하철수'].isnull().sum())
```

0
249

```
<ipython-input-173-2f9b3957fe11>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/ops.html
group1['10분내지하철수'] = group1['10분내지하철수'].fillna(group1['10분내지하철수'].mean())
```

type(group1)

pandas.core.frame.DataFrame

💡 새로운 데이터프레임으로 형성됨.

문제 2)

- 전체 데이터 프레임에 삽입

```
all_df['10분내지하철수'] = group1['10분내지하철수'].fillna(group1['10분내지하철수'].mean())
```

all_df['10분내지하철수'].isnull().sum()

3811

💡 데이터 프레임에 넣으려는 값 제외 나머지 값 전부가 null값 처리가 되어버림.

▼ 해결방안

💡 해결방안

- index 값으로 전체 데이터 프레임에 값 넣기

```
list =[('아파트', '충청남도', '국민임대'), ('아파트', '충청남도', '영구임대'),
('아파트', '대전광역시', '국민임대'), ('아파트', '대전광역시', '영구임대'),
('아파트', '대전광역시', '공공임대(10년)'), ('아파트', '대전광역시', '공공임대(분납)'),
('상가', '대전광역시', '임대상가'), ('아파트', '경상남도', '공공임대(10년)')]
```

```

for i in list :
    grouped = all_df.groupby(['임대건물구분', '지역', '공급유형'])
    group1 = grouped.get_group(i)
    group1['10분내지하철수'].mean()
    group1['10분내지하철수'].fillna(group1['10분내지하철수'].mean(), inplace=True)
    all_df.loc[group1.index, '10분내지하철수'] = group1['10분내지하철수']

```

```

grouped = all_df.groupby(['임대건물구분', '지역', '공급유형'])
grouped1 = all_df.groupby(['임대건물구분', '지역'])
grouped2 = all_df.groupby(['지역'])

group1 = grouped1.get_group(('아파트', '충청남도'))
group2 = grouped1.get_group(('아파트', '대전광역시'))

group3 = grouped.get_group(('아파트', '충청남도', '공공임대(50년') )
group4 = grouped.get_group(('아파트', '대전광역시', '공공임대(50년') )
group5 = grouped.get_group(('아파트', '대전광역시', '공공분양') )

group6 = grouped2.get_group(('충청남도' ))
group7 = grouped.get_group( ('상가', '충청남도', '임대상가') )

```

```

group3['10분내지하철수'].fillna(group1['10분내지하철수'].mean(), inplace=True)
all_df.loc[group3.index, '10분내지하철수'] = group3['10분내지하철수']

group4['10분내지하철수'].fillna(group2['10분내지하철수'].mean(), inplace=True)
all_df.loc[group4.index, '10분내지하철수'] = group4['10분내지하철수']

group5['10분내지하철수'].fillna(group2['10분내지하철수'].mean(), inplace=True)
all_df.loc[group5.index, '10분내지하철수'] = group5['10분내지하철수']

group6['10분내지하철수'].fillna(group6['10분내지하철수'].mean(), inplace=True)
all_df.loc[group7.index, '10분내지하철수'] = group7['10분내지하철수']

```

▼ 전처리 과정 해결 2

```
all_df['10분내지하철수'].isnull().sum()
```

249

```
all_df[all_df['10분내지하철수'].isnull()]['지역'].unique()
```

```
array(['충청남도', '대전광역시', '경상남도'], dtype=object)
```

'10분내지하철수' column 중 null값이 있는 지역 확인

```
all_df[all_df['10분내지하철수'].isnull()]['공급유형'].unique()
```

```
array(['국민임대', '영구임대', '임대상가', '공공임대(50년)', '공공임대(10년)', '공공분양',
       '공공임대(분납)'), dtype=object)
```

'10분내지하철수' column 중 null값에 해당하는 공급유형 확인

```

grouped = all_df.groupby(all_df.loc[all_df['10분내지하철수'].isnull()]['지역'])
group1 = grouped.get_group('충청남도')
group2 = grouped.get_group('대전광역시')
group3 = grouped.get_group('경상남도')

```

null값이 있는 지역별로 그룹화

```
group1['공급유형'].unique() # 충청남도
```

```
array(['국민임대', '영구임대', '임대상가', '공공임대(50년)'], dtype=object)
```

충청남도 지역에서 '10분내지하철수' column의 null값이 포함된 공급유형 확인

```
grouped = group1.groupby(group1['공급유형'])
group11 = grouped.get_group('국민임대')
group12 = grouped.get_group('영구임대')
group13 = grouped.get_group('임대상가') o
group14 = grouped.get_group('공공임대(50년)') o
```

```
group11_mean = group11['10분내지하철수'].mean()
group12_mean = group12['10분내지하철수'].mean()
group13_mean = group13['10분내지하철수'].mean()
group14_mean = group14['10분내지하철수'].mean()
all_df.loc[(all_df['지역']=='충청남도')&(all_df['공급유형']=='국민임대')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group11_mean
all_df.loc[(all_df['지역']=='충청남도')&(all_df['공급유형']=='영구임대')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group12_mean
all_df.loc[(all_df['지역']=='충청남도')&(all_df['공급유형']=='임대상가')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group13_mean
all_df.loc[(all_df['지역']=='충청남도')&(all_df['공급유형']=='공공임대(50년)')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group14_mean
```

모든 값이 null값이 나온 충청남도 지역 임대상가, 공공임대(50년)에 대해서는 해당 지역의 '10분내지하철수'의 평균값으로 해결

```
group2['공급유형'].unique() # 대전광역시
```

```
array(['공공임대(50년)', '국민임대', '영구임대', '임대상가', '공공분양', '공공임대(10년)', '공공임대(분납)'], dtype=object)
```

대전광역시에서 '10분내지하철수' column의 null값이 포함된 공급유형 확인

```
grouped = group2.groupby(group2['공급유형'])
group21 = grouped.get_group('공공임대(50년') o
group22 = grouped.get_group('국민임대')
group23 = grouped.get_group('영구임대')
group24 = grouped.get_group('임대상가')
group25 = grouped.get_group('공공분양') o
group26 = grouped.get_group('공공임대(10년)')
group27 = grouped.get_group('공공임대(분납)')
```

```
group21_mean = group21['10분내지하철수'].mean()
group22_mean = group22['10분내지하철수'].mean()
group23_mean = group23['10분내지하철수'].mean()
group24_mean = group24['10분내지하철수'].mean()
group25_mean = group25['10분내지하철수'].mean()
group26_mean = group26['10분내지하철수'].mean()
group27_mean = group27['10분내지하철수'].mean()
all_df.loc[(all_df['지역']=='대전광역시')&(all_df['공급유형']=='공공임대(50년)')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group21_mean
all_df.loc[(all_df['지역']=='대전광역시')&(all_df['공급유형']=='국민임대')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group22_mean
all_df.loc[(all_df['지역']=='대전광역시')&(all_df['공급유형']=='영구임대')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group23_mean
all_df.loc[(all_df['지역']=='대전광역시')&(all_df['공급유형']=='임대상가')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group24_mean
all_df.loc[(all_df['지역']=='대전광역시')&(all_df['공급유형']=='공공분양')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group25_mean
all_df.loc[(all_df['지역']=='대전광역시')&(all_df['공급유형']=='공공임대(10년)')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group26_mean
all_df.loc[(all_df['지역']=='대전광역시')&(all_df['공급유형']=='공공임대(분납)')&(all_df['10분내지하철수'].isnull()), '10분내지하철수']=group27_mean
```

```
group3['공급유형'].unique() # 경상남도
```

```
array(['공공임대(10년)'), dtype=object)
```

경상남도 지역에서 '10분내지하철수' column의 null값이 포함된 공급유형 확인

```
group3['10분내지하철수'] = all_df[all_df['지역']=='경상남도'][all_df['공급유형']=='공공임대(10년)']['10분내지하철수'].mean()
```

```
group3_mean = group3['10분내지하철수'].mean()
all_df.loc[(all_df['지역']=='경상남도') & (all_df['공급유형']=='공공임대(10년)') & (all_df['10분내지하철수'].isnull()), '10분내지하철수']=group3_mean
```

```
all_df[all_df['10분내지하철수'].isnull()]
```

단지코드	총세대수	임대건물구분	지역	공급유형	전용면적	전용면적별세대수	공가수	자격유형	임대보증금	...	10분내지하철수	10분내버스정류장수	단지내주차면수	단지코드_Type	임대건물구분_lbl	지역_lbl	공급유형_lbl	자격유형_lbl	단지코드_lbl	총세대수

0 rows x 21 columns

💡 '10분내지하철수'의 null값이 처리된 것을 확인

10분내 지하철수 결측치 처리 결과

방안 1) lightgbm score : 0.02671188499

방안 2) lightgbm score : 0.02658467644

→ 방안 2가 더 낮은 mae값을 보이기에 방안 2로 채택

임대료 및 임대보증금

▼ ML 이용

임대보증금, 임대료의 상관관계 확인

- 임대보증금, 임대료와 관련이 있는 피처를 찾기 위해 결측치 버림
- 임대보증금, 임대료의 값이 '-'인 경우를 제외한 df 생성(temp_df1)
- 임대보증금, 임대료의 값이 '-'인 경우를 0으로 치환한 df 생성(temp_df2)

```
temp_df1 = all_df.dropna(subset=['임대보증금', '임대료'])
temp_df1 = temp_df1[ (temp_df1['임대보증금'] != '-') & (temp_df1['임대료'] != '-') ]

temp_df1['임대보증금'] = temp_df1['임대보증금'].astype(int)
temp_df1['임대료'] = temp_df1['임대료'].astype(int)

temp_df2 = all_df.dropna(subset=['임대보증금', '임대료'])
temp_df2.loc[ temp_df2['임대보증금'] == '-', '임대보증금' ] = 0
temp_df2.loc[ temp_df2['임대료'] == '-', '임대료' ] = 0
```

```

temp_df2['임대보증금'] = temp_df2['임대보증금'].astype(int)
temp_df2['임대료'] = temp_df2['임대료'].astype(int)

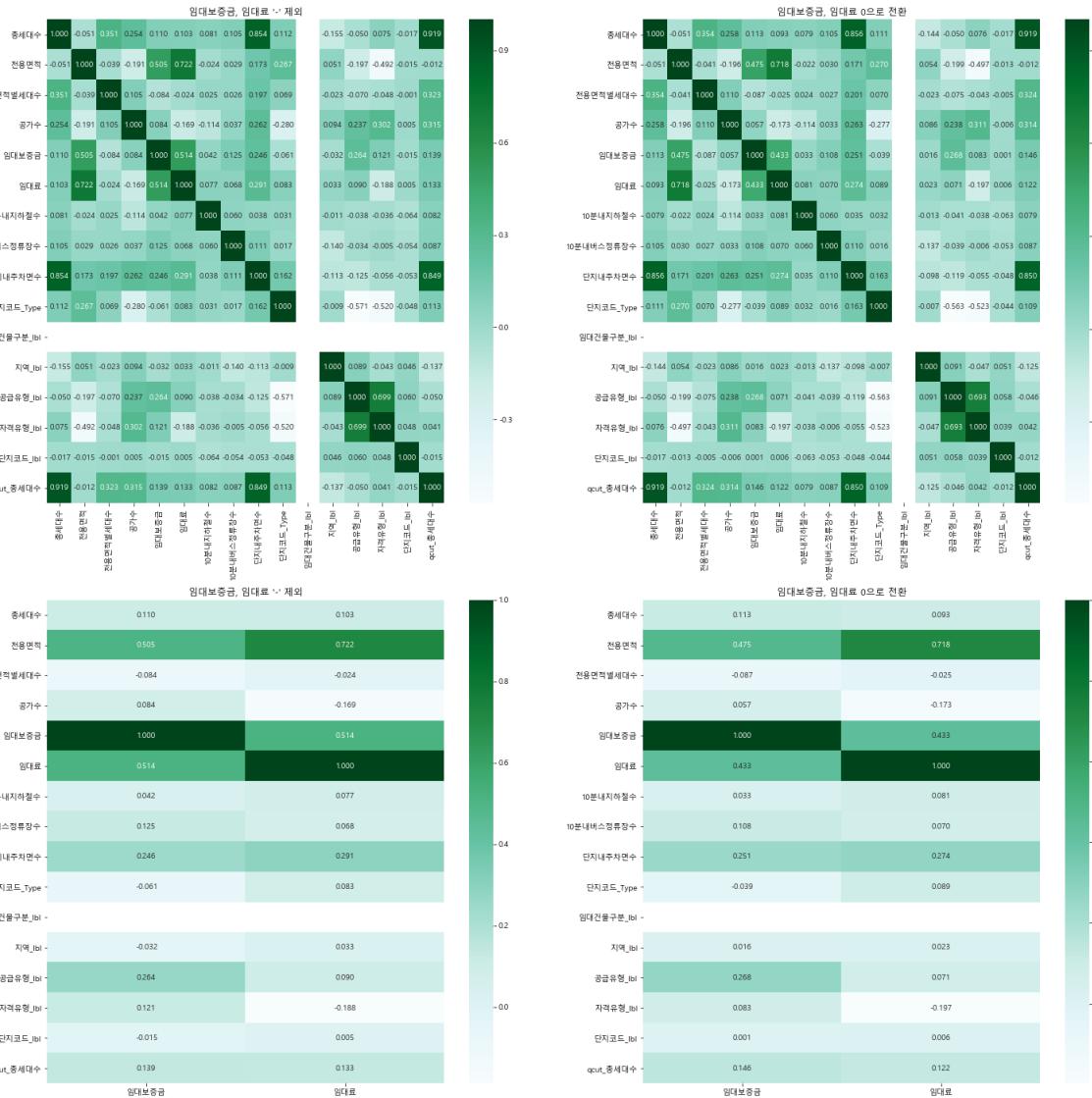
```

- temp_df1의 상관관계
- temp_df2의 상관관계

	임대보증금	임대료
총세대수	0.109937	0.103770
전용면적	0.511991	0.720872
전용면적별세대수	-0.092141	-0.031308
공가수	0.088114	-0.157846
임대보증금	1.000000	0.516733
임대료	0.516733	1.000000
10분내지하철수	0.042535	0.076499
10분내버스정류장수	0.124462	0.070836
단지내주차면수	0.258849	0.299199
단지코드_Type	-0.048894	0.085303
임대건물구분_lbl	NaN	NaN
지역_lbl	-0.033143	0.030153
공급유형_lbl	0.264547	0.094094
자격유형_lbl	0.117971	-0.188225
단지코드_lbl	-0.016044	0.006841
qcut_총세대수	0.136655	0.131697

	임대보증금	임대료
총세대수	0.113191	0.094205
전용면적	0.481889	0.717161
전용면적별세대수	-0.094079	-0.032282
공가수	0.061411	-0.163042
임대보증금	1.000000	0.437005
임대료	0.437005	1.000000
10분내지하철수	0.033077	0.080238
10분내버스정류장수	0.107087	0.072495
단지내주차면수	0.262169	0.282598
단지코드_Type	-0.028787	0.091140
임대건물구분_lbl	NaN	NaN
지역_lbl	0.014110	0.020394
공급유형_lbl	0.268473	0.074955
자격유형_lbl	0.080636	-0.197368
단지코드_lbl	-0.000883	0.007945
qcut_총세대수	0.143574	0.121128

- 임대보증금, 임대료의 상관관계 히트맵



```

plt.figure(figsize=(25, 25))
plt.subplot(2, 2, 1)
sns.heatmap(temp_df1.corr(), annot=True, fmt=' .3f', cmap='BuGn')
plt.title("임대보증금, 임대료 '-' 제외")

plt.subplot(2, 2, 2)
sns.heatmap(temp_df2.corr(), annot=True, fmt=' .3f', cmap='BuGn')
plt.title("임대보증금, 임대료 0으로 전환")

plt.subplot(2, 2, 3)
sns.heatmap(temp_df1.corr()[['임대보증금', '임대료']], annot=True, fmt=' .3f', cmap='BuGn')
plt.title("임대보증금, 임대료 '-' 제외")

plt.subplot(2, 2, 4)
sns.heatmap(temp_df2.corr()[['임대보증금', '임대료']], annot=True, fmt=' .3f', cmap='BuGn')
plt.title("임대보증금, 임대료 0으로 전환")

```

ML을 이용한 임대보증금, 임대료의 결측치 처리 결과

▼ 임대보증금 Feature 별 Score / MAE

▼ 모든 Feature

'총세대수', '전용면적', '전용면적별세대수', '공가수', '임대료', '10분내지하철수', '10분내버스정류장수',
'단지내주차면수', '단지코드_Type', '임대건물구분_.lbl', '지역_.lbl', '공급유형_.lbl',
'자격유형_.lbl', '단지코드_.lbl', 'qcut_총세대수'

▼ RandomForest

'-' 제외

학습(score) : 0.9785363721940312
테스트(score) : 0.890618145409086
MAE 평균값 : 3402920.059501753

'-' → 0으로 전환

학습(score) : 0.9836532393522289
테스트(score) : 0.8747424314942299
MAE 평균값 : 3556572.4523075493

▼ GradientBoosting 

'-' 제외

학습(score) : 0.9065718212497454
테스트(score) : 0.8537010754143878
MAE 평균값 : 2574712.9624406938

'-' → 0으로 전환

학습(score) : 0.9206473548604717
테스트(score) : 0.7873041201901261
MAE 평균값 : 2832035.382666461

▼ LightGBM

'-' 제외

학습(score) : 0.9890259102913317
테스트(score) : 0.9185920345540931
MAE 평균값 : 2733488.1159100593

'-' → 0으로 전환

학습(score) : 0.9882971213306754
테스트(score) : 0.8914624143013029
MAE 평균값 : 3091486.7720562434

▼ 임대건물구분_.lbl 제외한 모든 Feature

'총세대수', '전용면적', '전용면적별세대수', '공가수', '임대료', '10분내지하철수', '10분내버스정류장수',
'단지내주차면수', '단지코드_Type', '지역_.lbl', '공급유형_.lbl', '자격유형_.lbl',
'단지코드_.lbl', 'qcut_총세대수'

▼ RandomForest

'-' 제외

학습(score) : 0.9805973146023333
테스트(score) : 0.8987723980441471
MAE 평균값 : 3404500.915268573

'-' → 0으로 전환

학습(score) : 0.9815537073896923
테스트(score) : 0.8685242728390795
MAE 평균값 : 3564153.863022326

▼ GradientBoosting 

'-' 제외

학습(score) : 0.9065718212497454
테스트(score) : 0.8533351685052134
MAE 평균값 : 2574781.102977841

'-' → 0으로 전환
학습(score) : 0.9206473548604717
테스트(score) : 0.7804047003692716
MAE 평균값 : 2836520.9180518356

▼ LightGBM

'-' 제외
학습(score) : 0.9890259102913317
테스트(score) : 0.9185920345540931
MAE 평균값 : 2733488.1159100593

'-' → 0으로 전환
학습(score) : 0.9882971213306754
테스트(score) : 0.8914624143013029
MAE 평균값 : 3091486.7720562434

▼ 상관관계 0.1 이상 Feature

'총세대수', '전용면적', '10분내버스정류장수', '단지내주차면수', '공급유형_lbl',
'자격유형_lbl', 'qcut_총세대수'

▼ RandomForest

'-' 제외
학습(score) : 0.9758143016666967
테스트(score) : 0.8636011186315141
MAE 평균값 : 3576357.898373588

'-' → 0으로 전환
학습(score) : 0.9694878300042289
테스트(score) : 0.8637037386371125
MAE 평균값 : 3900773.34723837

▼ GradientBoosting 

'-' 제외
학습(score) : 0.8653622680255605
테스트(score) : 0.8144103081097256
MAE 평균값 : 2844977.002398008
MAE 평균값 : 4061018.915705881

'-' → 0으로 전환
학습(score) : 0.8834988365813973
테스트(score) : 0.7881883289765126
MAE 평균값 : 3102508.112921849

▼ LightGBM

'-' 제외
학습(score) : 0.9714602466133712
테스트(score) : 0.8953699448849565
MAE 평균값 : 3289762.3437321214

'-' → 0으로 전환
학습(score) : 0.9714341685290594
테스트(score) : 0.8649435364369086
MAE 평균값 : 3780443.1688062563

▼ 상관관계 상위 5개 Feature

'전용면적', '임대료', '단지내주차면수', '공급유형_lbl', 'qcut_총세대수'

▼ RandomForest

'-' 제외

학습(score) : 0.9567174963672654
테스트(score) : 0.7405793249874688
MAE 평균값 : 4707915.568604569

'-' → 0으로 전환

학습(score) : 0.9656773136867853
테스트(score) : 0.7054652943410158
MAE 평균값 : 5101515.52427306

▼ GradientBoosting

'-' 제외

학습(score) : 0.744255521827103
테스트(score) : 0.6512031648327945
MAE 평균값 : 4389588.42126325

'-' → 0으로 전환

학습(score) : 0.7920168600277779
테스트(score) : 0.6118379074728817
MAE 평균값 : 4797132.757251739

▼ LightGBM

'-' 제외

학습(score) : 0.9004430657143366
테스트(score) : 0.7275834732335754
MAE 평균값 : 5108852.848834826

'-' → 0으로 전환

학습(score) : 0.9198584960844435
테스트(score) : 0.7009510059899682
MAE 평균값 : 5583782.308519905

▼ 임대료 Feature 별 Score / MAE

▼ 모든 Feature

'총세대수', '전용면적', '전용면적별세대수', '공가수', '임대료', '10분내지하철수', '10분내버스정류장수',
'단지내주차면수', '단지코드_Type', '임대건물구분_lbl', '지역_lbl', '공급유형_lbl',
'자격유형_lbl', '단지코드_lbl', 'qcut_총세대수'

▼ RandomForest

'-' 제외

학습(score) : 0.9831591437762538
테스트(score) : 0.9060798930112266
MAE 평균값 : 22847.569398884

'-' → 0으로 전환

학습(score) : 0.9815908414474658
테스트(score) : 0.9019130946054446
MAE 평균값 : 23006.671517498697

▼ GradientBoosting

'-' 제외

학습(score) : 0.89601064140107
테스트(score) : 0.8718547070978245
MAE 평균값 : 17212.855084341838

'-' → 0으로 전환

학습(score) : 0.8908403845029009

테스트(score) : 0.8581878515964133
MAE 평균값 : 17766.635299570808

▼ LightGBM

'-' 제외
학습(score) : 0.9915813848560712
테스트(score) : 0.9512606026350369
MAE 평균값 : 18490.8235112282
'-' → 0으로 전환
학습(score) : 0.9924134738632072
테스트(score) : 0.9363247930668442
MAE 평균값 : 19191.38887432587

▼ 임대건물구분_lbl 제외한 모든 Feature

'총세대수', '전용면적', '전용면적별세대수', '공가수', '임대료', '10분내지하철수', '10분내버스정류장수',
'단지내주차면수', '단지코드_Type', '지역_lbl', '공급유형_lbl', '자격유형_lbl',
'단지코드_lbl', 'qcut_총세대수'

▼ RandomForest

'-' 제외
학습(score) : 0.9819528769316641
테스트(score) : 0.9073895818565239
MAE 평균값 : 22869.66010229357
'-' → 0으로 전환
학습(score) : 0.9819985113269603
테스트(score) : 0.9078894042110826
MAE 평균값 : 22986.49857072895

▼ GradientBoosting 

'-' 제외
학습(score) : 0.89601064140107
테스트(score) : 0.8720190764384371
MAE 평균값 : 17277.503168745243
'-' → 0으로 전환
학습(score) : 0.8908403845029009
테스트(score) : 0.8588417509897338
MAE 평균값 : 17821.490619891734

▼ LightGBM

'-' 제외
학습(score) : 0.9915813848560712
테스트(score) : 0.9512606026350369
MAE 평균값 : 18490.8235112282
'-' → 0으로 전환
학습(score) : 0.9924134738632072
테스트(score) : 0.9363247930668442
MAE 평균값 : 19191.38887432587

▼ 상관관계 0.1 이상 Feature

'총세대수', '전용면적', '10분내버스정류장수', '단지내주차면수', '공급유형_lbl',
'자격유형_lbl', 'qcut_총세대수'

▼ RandomForest

'-' 제외

학습(score) : 0.9787341616625912
테스트(score) : 0.8866414985513585
MAE 평균값 : 25799.856898008446

'-' → 0으로 전환

학습(score) : 0.9807949363810026
테스트(score) : 0.8925908019747947
MAE 평균값 : 25666.59595774691

▼ GradientBoosting

'-' 제외

학습(score) : 0.866013112980877
테스트(score) : 0.8324566018708346
MAE 평균값 : 19484.07134743729

'-' → 0으로 전환

학습(score) : 0.8535624941775708
테스트(score) : 0.8200042016251071
MAE 평균값 : 19941.161726166036

▼ LightGBM

'-' 제외

학습(score) : 0.9772977375654344
테스트(score) : 0.9356821642853483
MAE 평균값 : 22517.32419390624

'-' → 0으로 전환

학습(score) : 0.9776822833701081
테스트(score) : 0.9141046806044756
MAE 평균값 : 23044.12845961888

▼ 상관관계 상위 5개 Feature

'전용면적', '임대료', '단지내주차면수', '공급유형_lbl', 'qcut_총세대수'

▼ RandomForest

'-' 제외

학습(score) : 0.9747055911206635
테스트(score) : 0.8587252153451508
MAE 평균값 : 30108.23303606257

'-' → 0으로 전환

학습(score) : 0.9703431516869858
테스트(score) : 0.8592461175766163
MAE 평균값 : 30092.07488674039

▼ GradientBoosting

'-' 제외

학습(score) : 0.8320087880601525
테스트(score) : 0.792557387424434
MAE 평균값 : 24972.85085069869

'-' → 0으로 전환

학습(score) : 0.8191147402582868
테스트(score) : 0.7846561705821546
MAE 평균값 : 25629.697952471477

▼ LightGBM

```
'-' 제외
학습(score) : 0.9527491912062102
테스트(score) : 0.8850601703192005
MAE 평균값 : 30054.689492801328

'-' → 0으로 전환
학습(score) : 0.9498364368258596
테스트(score) : 0.8674203291056329
MAE 평균값 : 31251.292197166757
```

▼ 0, 평균값, 중앙값으로 처리 및 정규화

임대보증금, 임대료의 결측치를 0, 평균, 중앙값으로 처리

동일한 과정이 반복되어 한번에 처리

Train, Test 데이터 병합 후 '임대보증금'과 '임대료'의 '-' 값을 0으로 변경

```
all_df = pd.concat([train, test], ignore_index=True)

all_df.loc[ all_df['임대보증금'] == '-', '임대보증금'] = 0
all_df.loc[ all_df['임대료'] == '-', '임대료'] = 0
```

임대보증금의 결측치가 없는 temp_df를 생성하여 임대보증금과 임대료의 평균값, 중앙값 산출

```
temp_df = all_df.loc[ ~all_df['임대보증금'].isna() ]           # 임대보증금의 Nan이 없는 DF 생성
fee_mean1 = temp_df['임대보증금'].astype(int).mean()            # 임대보증금 평균값
fee_mean2 = temp_df['임대료'].astype(int).mean()                # 임대료 평균값

fee_median1 = temp_df['임대보증금'].astype(int).median()         # 임대보증금 중앙값
fee_median2 = temp_df['임대료'].astype(int).median()              # 임대료 중앙값
```

결측치를 처리하여 저장할 각 DF 생성 후 결측치 처리

```
me_df1 = all_df.copy()   # 임대보증금, 임대료 0
me_df2 = all_df.copy()   # 임대보증금, 임대료 평균값
me_df3 = all_df.copy()   # 임대보증금, 임대료 중앙값

me_df1.loc[ me_df1['임대보증금'].isna(), '임대보증금' ] = 0
me_df1.loc[ me_df1['임대료'].isna(), '임대료' ] = 0

me_df2.loc[ me_df2['임대보증금'].isna(), '임대보증금' ] = fee_mean1
me_df2.loc[ me_df2['임대료'].isna(), '임대료' ] = fee_mean2

me_df3.loc[ me_df3['임대보증금'].isna(), '임대보증금' ] = fee_median1
me_df3.loc[ me_df3['임대료'].isna(), '임대료' ] = fee_median2
```

수치형 데이터 RobustScaler 이용하여 정규화

결측치 처리된 DF의 정규화하여 temp_me_df 저장

```
scaler = RobustScaler()

scal_cols = ['총세대수', '전용면적', '전용면적별세대수', '공가수',
             '임대보증금', '임대료', '10분내지하철수', '10분내버스정류장수',
             '단지내주차면수', '단지당 자동차 보유 가능 인구비율', '교통편의성',
             '총세대수주차면수']

scaler.fit(me_df1[scal_cols])
temp_me_df1_scal = pd.DataFrame(scaler.transform(me_df1[scal_cols]))

scaler.fit(me_df2[scal_cols])
temp_me_df2_scal = pd.DataFrame(scaler.transform(me_df2[scal_cols]))

scaler.fit(me_df3[scal_cols])
temp_me_df3_scal = pd.DataFrame(scaler.transform(me_df3[scal_cols]))
```

정규화된 임시DF 를 Column명 정리 후 각 DF(me_df_scal)에 저장

```
# 임시DF Column 정리
temp_me_df1_scal.columns = scal_cols
temp_me_df2_scal.columns = scal_cols
temp_me_df3_scal.columns = scal_cols

# 정규화된 내용 담을 DF 생성
me_df1_scal = me_df1.copy()
me_df2_scal = me_df2.copy()
me_df3_scal = me_df3.copy()

# 정규화된 내용 DF에 입력
me_df1_scal.loc[:, scal_cols] = temp_me_df1_scal[scal_cols]
me_df2_scal.loc[:, scal_cols] = temp_me_df2_scal[scal_cols]
me_df3_scal.loc[:, scal_cols] = temp_me_df3_scal[scal_cols]
```

각 DF를 Train, Test 로 분할 후 Test에 '등록차량수', 'log_등록차량수'를 제외

```
# 임대보증금, 임대료 0 처리된 DF
me_df1_train = me_df1.iloc[0:2896,:]
me_df1_test = me_df1.iloc[2896,:,:]

me_df1_test.drop(['등록차량수', 'log_등록차량수'], inplace=True, axis=1)

# 임대보증금, 임대료 평균값 처리된 DF
me_df2_train = me_df2.iloc[0:2896,:]
me_df2_test = me_df2.iloc[2896,:,:]

me_df2_test.drop(['등록차량수', 'log_등록차량수'], inplace=True, axis=1)

# 임대보증금, 임대료 중앙값 처리된 DF
me_df3_train = me_df3.iloc[0:2896,:]
me_df3_test = me_df3.iloc[2896,:,:]

me_df3_test.drop(['등록차량수', 'log_등록차량수'], inplace=True, axis=1)

# 임대보증금, 임대료 0 및 정규화 처리된 DF
me_df1_scal_train = me_df1_scal.iloc[0:2896,:]
me_df1_scal_test = me_df1_scal.iloc[2896,:,:]

me_df1_scal_test.drop(['등록차량수', 'log_등록차량수'], inplace=True, axis=1)

# 임대보증금, 임대료 평균값 및 정규화 처리된 DF
me_df2_scal_train = me_df2_scal.iloc[0:2896,:]
me_df2_scal_test = me_df2_scal.iloc[2896,:,:]

me_df2_scal_test.drop(['등록차량수', 'log_등록차량수'], inplace=True, axis=1)

# 임대보증금, 임대료 중앙값 및 정규화 처리된 DF
me_df3_scal_train = me_df3_scal.iloc[0:2896,:]
me_df3_scal_test = me_df3_scal.iloc[2896,:,:]

me_df3_scal_test.drop(['등록차량수', 'log_등록차량수'], inplace=True, axis=1)
```

임대보증금, 임대료의 결측치 처리 후 CatBoost Score 측정 결과

▼ 사용 Feature

'총세대수', '전용면적', '공가수', '단지내주차면수','qcut_총세대수', '자격유형_lbl',
'전용면적별세대수', '10분내버스정류장수', '10분내지하철수', '공급유형_lbl',
'지역_lbl', '단지코드_lbl','단지코드_Type', '교통편의성','총세대수주차면수','임대료',
'임대보증금', '단지당 자동차 보유 가능 인구비율'

▼ ML(Gradient Boosting Regressor) 이용

Train : 0.9671082141232954

Test : 0.9596922637834391

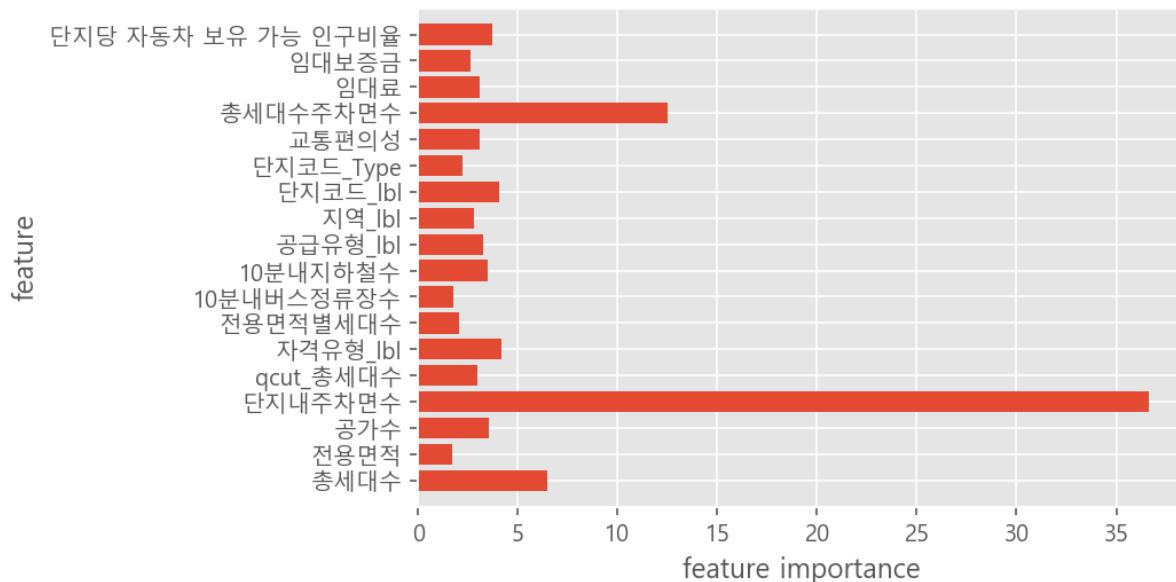
MAE 평균값 : 0.10439695701113991

▼ 0으로 처리

Train : 0.9660290240496178
Test : 0.9582228452461531
MAE 평균값 : 0.09814497306425582

▼ 0으로 처리 (정규화) 

Train : 0.9660290240496178
Test : 0.9582228452461531
MAE 평균값 : 0.09813813832407789



▼ 평균값으로 처리

Train : 0.9702025638528791
Test : 0.9606845258036387
MAE 평균값 : 0.09851047157896405

▼ 평균값으로 처리 (정규화)

Train : 0.9702025638528791
Test : 0.9606845258036387
MAE 평균값 : 0.09851094203323238

▼ 중앙값으로 처리

Train : 0.9716406430914867
Test : 0.9630310883892691
MAE 평균값 : 0.10030674181147008

▼ 중앙값으로 처리 (정규화)

Train : 0.9716406430914867
Test : 0.9630310883892691
MAE 평균값 : 0.10031003787876525

임대보증금, 임대료 결측치 처리 결과

자체 TEST 결과) 0으로 처리하여 정규화한 모델 : MAE 가장 낮음

DACON 제출) 0으로 처리하고 정규화하지 않은 모델 : MAE 가장 낮음

FEATURE 추가

기존 데이터 활용

- ▼ 교통편의성, 총세대수주차면수

🤔 주변 대중교통이 편의성과 단지내 주차편의성은 자동차 등록대수와 상관관계가 있을까?

```
all_df['교통편의성'] = all_df['10분내지하철수'] + all_df['10분내버스정류장수']
all_df['총세대수주차면수'] = all_df['단지내주차면수'] / all_df['총세대수']
```

```
<등록차량수 상관계수>
교통편의성          0.081256
총세대수주차면수      0.600569
```

```
<log_등록차량수 상관계수>
교통편의성          0.053589
총세대수주차면수      0.735821
```

기존 데이터 활용 결과

CatBoost Score : 0.104347 / DACON Score 97.45656313

AGE 데이터 활용

- ▼ 방법 1 : 단지별 자동차 보유 인구비율

🤔 단지별 자동차 보유 인구와 자동차 등록대수는 상관관계가 있지 않을까?

외부데이터 사용 : 2021년_06월_자동차_등록자료_통계

```
국토교통부 통계누리
국토교통부 통계누리 홈페이지입니다.
http://stat.molit.go.kr/portal/cate/statFileView.do?hRsId=58&hFormId=5409&hKeyWord=자동차등록&hTotalFlag=Y
```

2021년_06월_자동차_등록자료_통계 중 04. 성별_연령별 시트 사용

```
car = pd.read_excel('./Data/2021년_06월_자동차_등록자료_통계.xlsx',
                     sheet_name=3,
                     skiprows=2,
                     usecols=range(0, 20))
```

데이터 확인 및 결측치 처리

	성별	연령/시도	총계	서울	부산	대구	인천	광주	대전	울산	세종	경기	강원	충북	충남	전북	전남	경북	경남	제주
0	남성	10대 이하	7437	1371	446	198	389	153	276	189	75	2140	181	267	277	294	280	184	564	153
1	NaN	20대	356992	34205	18101	16904	22269	12628	12114	8930	2209	97857	14409	16590	20507	14402	15546	20746	24485	5090
2	NaN	30대	2159536	302499	118012	101979	127869	63873	66700	53399	22189	629142	63712	77251	111205	67031	70872	112663	143393	27747
3	NaN	40대	3912264	508214	211040	189715	215684	117530	113270	99745	42365	1109019	118716	132873	197513	142332	145890	221831	286281	60246
4	NaN	50대	4417099	535067	225506	226919	231824	130177	129276	126736	31715	1138075	156776	167229	219242	183433	203100	300922	341419	69683
5	NaN	60대	3450897	433997	203219	172957	173957	84641	96344	97069	19851	788009	142611	141150	177831	152936	166940	272794	277696	48895
6	NaN	70대	1088779	158200	67062	55435	44577	26622	28079	23543	5163	217137	47229	42168	60042	55916	62158	94595	83319	17534
7	NaN	80대	193790	30506	10403	9524	6641	4423	4795	2808	837	41212	10213	7868	11054	10473	10352	16589	12406	3686
8	NaN	90대 이상	16671	3403	1012	746	627	382	384	222	65	3482	658	504	859	957	978	1165	971	256
9	NaN	계	15603465	2007462	854801	774377	823837	440429	451238	412641	124469	4026073	554505	585900	798530	627774	676116	1041489	1170534	233290
10	여성	10대 이하	3689	587	242	102	234	75	148	83	39	1023	106	144	152	142	148	117	276	71
11	NaN	20대	186048	18187	8703	8861	10915	8067	6911	4183	1542	49210	6969	8661	10559	8249	8015	11038	12403	3575
12	NaN	30대	864601	115444	45062	40177	50428	31412	28244	18743	10623	242719	27253	31771	41818	32167	31064	44449	57703	15524
13	NaN	40대	1444970	186445	77937	71816	80957	50143	46477	32357	15840	401853	48248	50214	66009	55040	52420	77929	103389	27896
14	NaN	50대	1652221	196321	87790	88535	90322	53551	51328	36518	11214	441516	61910	61846	77996	69142	68341	101304	122361	32226
15	NaN	60대	1051197	135119	61218	54153	54169	28490	30836	21786	6202	257824	45510	40241	52508	46671	45864	72637	78265	19704
16	NaN	70대	225936	36810	13412	11973	9511	6005	6061	3856	1092	54033	9526	7796	10799	10853	10175	15019	14601	4414
17	NaN	80대	49602	7952	2672	2254	1984	1288	1431	733	245	11378	1856	1789	2558	3077	3053	3274	3241	817
18	NaN	90대 이상	11440	2041	547	441	442	344	352	187	55	2464	538	304	544	824	754	665	734	204
19	NaN	계	5489704	698906	297583	278312	298962	179375	171788	118446	46852	1462020	201916	202766	262943	226165	219834	326432	392973	104431
20	기타	법인 및 사업자	3549082	456617	295556	165075	554335	74535	64052	47169	8528	619306	61462	89844	100941	99758	242927	119784	248768	300425
21	합계	NaN	24642251	3162985	1447940	1217764	1677134	694339	687078	578256	179849	6107399	817883	878510	1162414	953697	1138877	1487705	1812275	638146

엑셀 상에서 셀 병합된 부분이 결측치로 확인되므로 동일한 데이터를 넣음

```
car.iloc[1:10, 0] = '남성'
car.iloc[11:20, 0] = '여성'
car.iloc[21, 1] = '합계'
```

Column명 정리 및 지역명을 원말로 변경

```
car.columns = ['성별', '연령', '총계', '서울특별시', '부산광역시',
               '대구광역시', '인천광역시', '광주광역시', '대전광역시', '울산광역시',
               '세종특별자치시', '경기도', '강원도', '충청북도', '충청남도',
               '전라북도', '전라남도', '경상북도', '경상남도', '제주특별자치도']
```

'성별'과 '연령'을 합친 '성별연령' Column 생성

```
car['성별연령'] = car['성별'] + ' ' + car['연령']
```

사용하지 않는 '합계'와 '기타항목' 제거

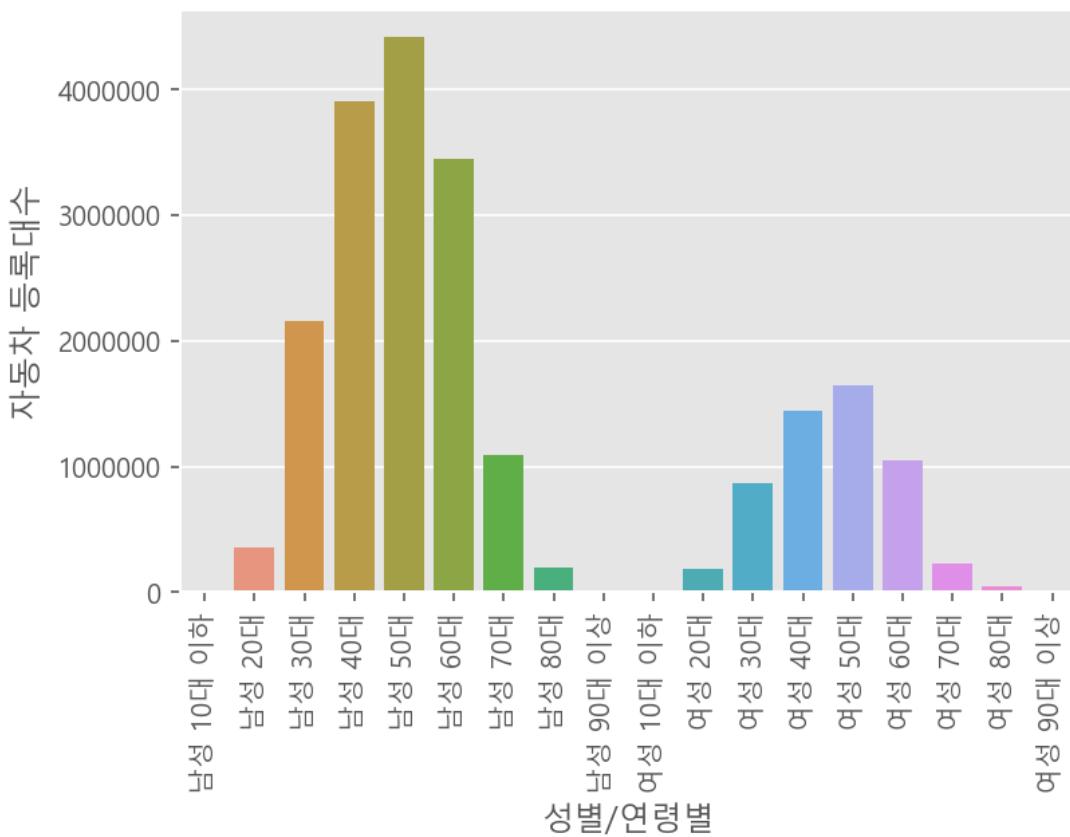
```
grap_car = car.drop([9, 19, 20, 21]).drop('총계', axis=1)
```

남성과 여성의 연령대별 자동차 등록대수 시각화

```
grap_car_age = pd.DataFrame(grap_car.set_index('성별연령').sum(axis=1)).T

plt.xticks(rotation = 90 )
sns.barplot(data=grap_car_age, ci=None)

plt.xlabel("성별/연령별")
plt.ylabel("자동차 등록대수")
```



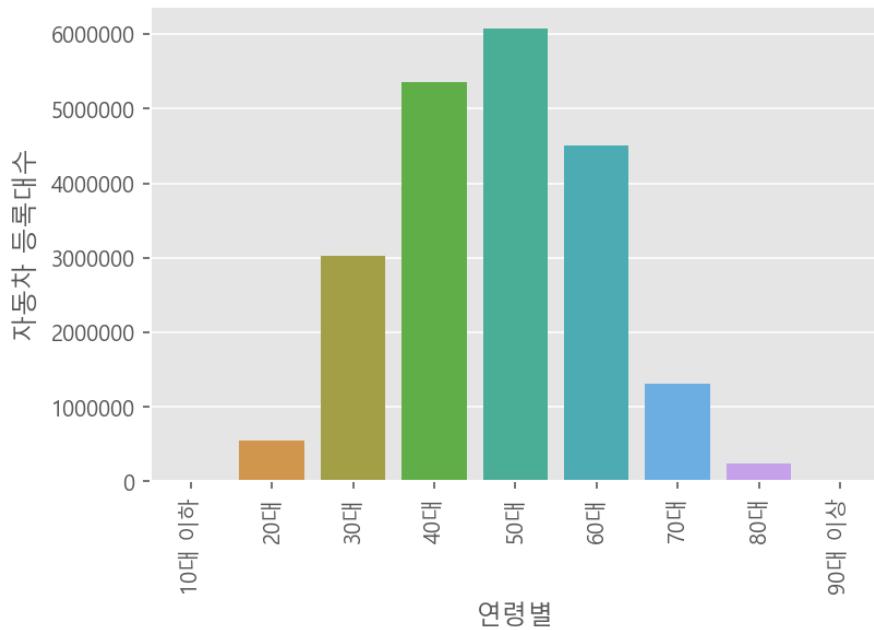
남성과 여성을 합친 연령대별 자동차 등록대수를 시각화

```

grouped = grp_car.groupby('연령')
group = pd.DataFrame(grouped.sum().sum(axis=1)).T

plt.xticks(rotation = 90 )
sns.barplot(data=group, ci=None)
plt.xlabel("연령별")
plt.ylabel("자동차 등록대수")

```



20 ~ 70의 자동차 등록대수가 많으므로 지역별 임대주택 인구비율에서 20 ~ 70대의 정보를 이용

20 ~ 70대 인구 비율을 이용하여 단지당 자동차 보유 가능 인구비율 구하기

- 가설 : 자동차 보유 인구를 20대에서 70대로 가정하며, 해당 인구는 차량이 모두 있다고 가정
- 공식 : 지역별 자동차 보유 가능 연령비율 / 지역별 총 세대수 * 단지별 세대수

Train 데이터와 Test 데이터를 통합하여 입력

```
all_df = pd.concat([train, test], ignore_index=True)
```

단지당 자동차 보유 가능 인구비율을 구하는 함수 생성

```
def car_calc(data) :
    loc = data['지역']

    total_house = group.loc[loc, '총세대수'].sum()
    loc_house = data['총세대수']
    popul = age.loc[age['지역'] == loc, '20-70대'].values[0]

    result = (popul / total_house) * loc_house

    return result
```

단지코드에 중복이 있으므로 Group으로 묶은 후 '지역', '단지코드'별 평균값을 가진 DF 생성

단지당 자동차 보유 가능 인구비율 계산

```
grouped = all_df.groupby(['지역', '단지코드'])
group = grouped.mean()

all_df['단지당 자동차 보유 가능 인구비율'] = all_df.apply(car_calc, axis=1)
```

Train, Test 데이터 분리 및 test 데이터의 '등록차량수', 'log_등록차량수' 제거

```
train = all_df.iloc[0:2896,:]
test = all_df.iloc[2896:,:]

test.drop(['등록차량수', 'log_등록차량수'], inplace=True, axis=1)
```

'단지당 자동차 보유 가능 인구비율'의 상관관계 및 시각화

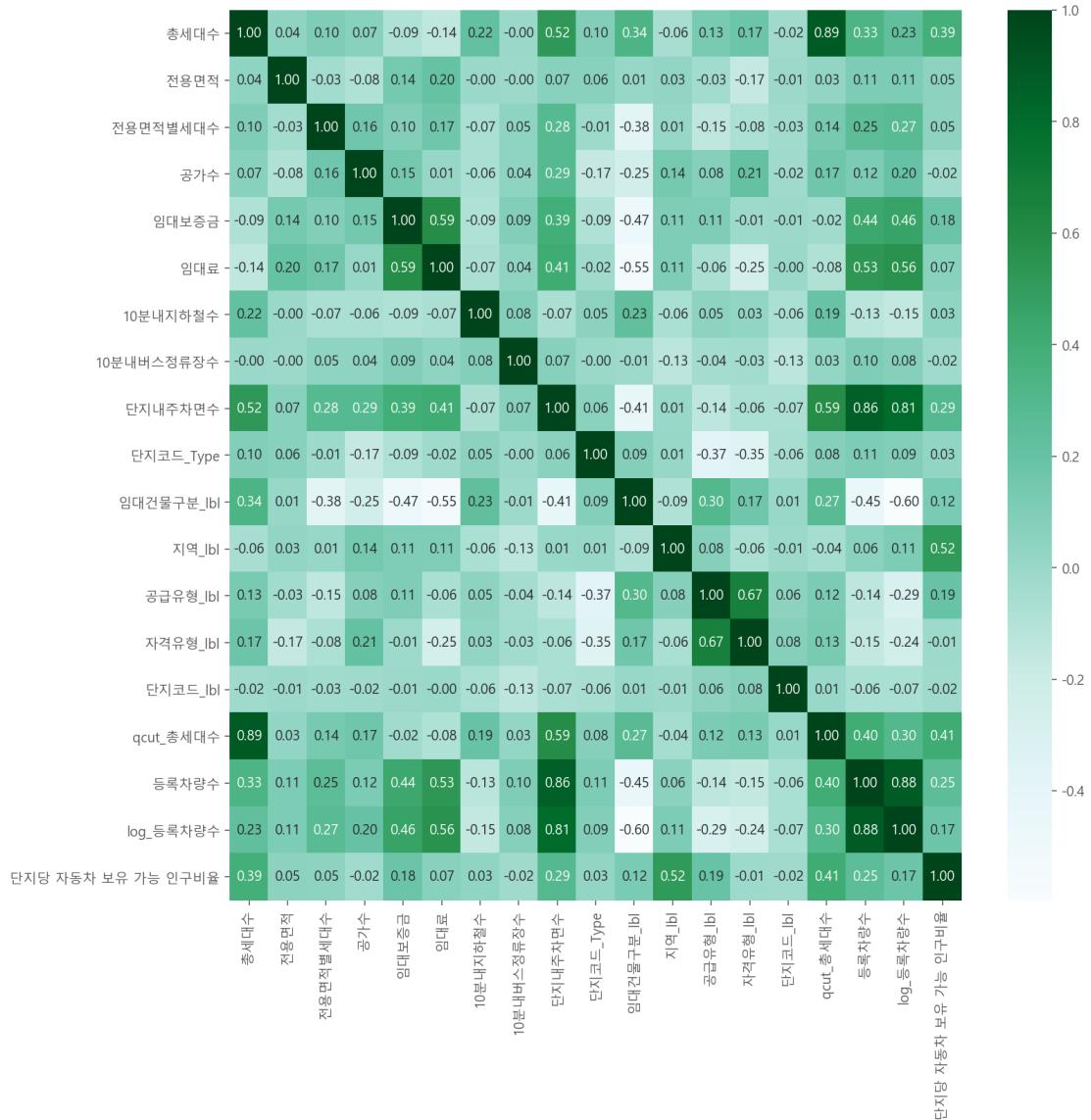
상관관계

```
train.corr()['단지당 자동차 보유 가능 인구비율'].sort_values()
```

```
공가수           -0.023817
단지코드_lbl      -0.021588
10분내버스정류장수   -0.016801
자격유형_lbl      -0.014882
단지코드_Type       0.030354
10분내지하철수        0.033987
전용면적별세대수      0.048343
전용면적            0.053147
임대료             0.067094
임대건물구분_lbl     0.122622
log_등록차량수        0.174055
임대보증금          0.177486
공급유형_lbl          0.191996
등록차량수          0.254782
단지내주차면수        0.290083
총세대수            0.388174
qcut_총세대수         0.412037
지역_lbl             0.519520
단지당 자동차 보유 가능 인구비율    1.000000
Name: 단지당 자동차 보유 가능 인구비율, dtype: float64
```

시각화

```
plt.figure(figsize=(15, 15))
sns.heatmap(data=train.corr(), annot=True, fmt='.3f', cmap='BuGn')
```



▼ 방법 2 : 지역별 단지내주차면수 비율

 인구비율과 지역별 단지내주면수의 상관관계가 있지 않을까?

AGE 데이터 : 경제활동인구비율

남여 데이터 통합

```

age['10대 미만'] = age['10대미만(여자)'] + age['10대미만(여자)']
age['10대'] = age['10대(여자)'] + age['10대(남자)']
age['20대'] = age['20대(여자)'] + age['20대(남자)']
age['30대'] = age['30대(여자)'] + age['30대(남자)']
age['40대'] = age['40대(여자)'] + age['40대(남자)']
age['50대'] = age['50대(여자)'] + age['50대(남자)']
age['60대'] = age['60대(여자)'] + age['60대(남자)']
age['70대'] = age['70대(여자)'] + age['70대(남자)']
age['80대'] = age['80대(여자)'] + age['80대(남자)']
age['90대'] = age['90대(여자)'] + age['90대(남자)']
age['100대'] = age['10대(여자)'] + age['10대(남자)']

```

```
age.drop(['10대미만(여자)', '10대미만(남자)', '10대(여자)', '10대(남자)', '20대(여자)',
          '20대(남자)', '30대(여자)', '30대(남자)', '40대(여자)', '40대(남자)', '50대(여자)',
          '50대(남자)', '60대(여자)', '60대(남자)', '70대(여자)', '70대(남자)', '80대(여자)',
          '80대(남자)', '90대(여자)', '90대(남자)', '100대(여자)', '100대(남자)', '10대 미만'], axis=1, inplace=True)
```

지역별 경제활동 인구비율을 이용하여 지역별 인구대비 단지내 주차면수 총합 비율과 상관관계 구하기

- 가설 : 경제활동 인구는 20-60대로 분류하며, 경제활동인구는 자동차 보유할 확률이 높다고 가정
- 공식: 지역별 단지내주차면수총합 / 지역별 인구수

```
# 경제활동인구(20-60대) 데이터 추출

age1 = age1.T
age1 = age1.rename(columns=age1.iloc[0])
age1 = age1.drop(age1.index[0])
age2 = age1.iloc[1:6]
age2
```

```
# 지역별 경제활동인구비율 데이터 추출

age_sum = pd.DataFrame(age2.sum())
age_sum= age_sum.reset_index(drop=False)
age_sum.columns=['지역','인구비율']

age_sum = age_sum.sort_values(by=['지역'], axis=0)
age_sum= age_sum.reset_index(drop=True)
age_sum
```

Train 데이터: 지역별 단지내 주차면수 총합

```
region = train[['지역','단지내주차면수']].drop_duplicates().reset_index(drop=True)
```

```
regions = region['지역'].unique()
region1 = []
for a in regions:
    region_sum = region[ region['지역'] == a ]['단지내주차면수'].sum()
    region1.append(region_sum)

regions = pd.Series(regions)
region1 = pd.Series(region1)
r_sum = pd.concat([regions,region1],axis=1)

r_sum.columns=['지역','단지내주차면수합']
r_sum = r_sum.sort_values(by=['지역'], axis=0)
r_sum= r_sum.reset_index(drop=True)
r_sum
```

```
# 합치기

age_r = pd.merge(age_sum,r_sum,on="지역")

age_r=age_r.sort_values(by=['지역']).reset_index(drop=True)
age_r
```

	지역	인구비율	단지내주차면수합
0	강원도	0.699757	12904
1	경기도	0.695835	61386
2	경상남도	0.710398	28096
3	경상북도	0.693101	11687
4	광주광역시	0.705336	14975
5	대구광역시	0.734845	12987
6	대전광역시	0.716473	18632
7	부산광역시	0.676011	15731
8	서울특별시	0.672458	5271
9	세종특별자치시	0.744909	4331
10	울산광역시	0.703001	2827
11	전라남도	0.692963	11612
12	전라북도	0.679325	9872
13	제주특별자치도	0.678012	5746
14	충청남도	0.701710	6227
15	충청북도	0.693262	17890

외부데이터 사용 : 지역별 인구수

KOSIS

age_r 데이터 형식에 맞춘 데이터프레임 생성

```

po = nums[['소재지(시군구)별(1)', '2020']]
po.columns=['지역', '인구수']

a = po.iloc[3:6]
b = po.iloc[7:11]
c = po.iloc[12:]

pop= pd.concat([a,b,c])
pop=pop.sort_values(by=['지역'], axis=0)
pop=pop.reset_index(drop=True)
pop

```

🚫 주의) 기존 데이터에는 인천광역시의 데이터가 없다!

```

# 합치기

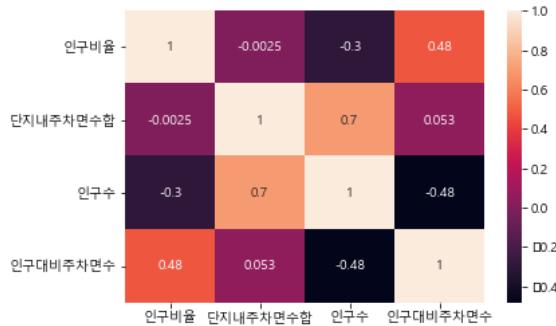
df=pd.concat([age_r,pop],axis=1)
df = df.iloc[0:,1:]
df=df[['지역','인구비율','단지내주차면수합','인구수']]
df

```

	지역	인구비율	단지내주차면수합	인구수
0	강원도	0.699757	12904	1542840
1	경기도	0.695835	61386	13427014
2	경상남도	0.710398	28096	3340216
3	경상북도	0.693101	11687	2639422
4	광주광역시	0.705336	14975	1450062
5	대구광역시	0.734845	12987	2418346
6	대전광역시	0.716473	18632	1463882
7	부산광역시	0.676011	15731	3391946
8	서울특별시	0.672458	5271	9668465
9	세종특별자치시	0.744909	4331	355831
10	울산광역시	0.703001	2827	1136017
11	전라남도	0.692963	11612	1851549
12	전라북도	0.679325	9872	1804104
13	제주특별자치도	0.678012	5746	674635
14	충청남도	0.701710	6227	2121029
15	충청북도	0.693262	17890	1600837

경제활동인구비율과 인구대비주차면수의 상관관계

```
# 상관관계 피쳐추가
df['인구대비주차면수']=df['단지내주차면수합']/df['인구수'] -> 0.48
```



인구대비주차면수와 인구비율의 상관관계를 나타내는 피쳐추가

→ 낮은 상관계수를 나타냄

```
df['인구관련지표'] =df['인구대비주차면수']/df['인구비율']

# log_등록차량수 상관계수
인구수          0.306544
인구대비주차면수 -0.076051
```

주차면수대비	0.106292
인구관련지표	-0.081743

AGE 데이터 활용 결과

방법 1 (단지별 자동차 보유 인구비율)

CatBoost Score : 0.100936 / DACON Score 105 → 방법 1 사용 ✗

방법2 (지역별 단지내주차면수 비율)

CatBoost Score : 0.166578/ DACON Score 108.6 → 방법 2 사용 ✗

모델(Model) 사용이력

기존 부스팅 기법

1. 실제 값들의 평균과 실제 값의 차이인 잔차(Residual)를 구한다.
2. 데이터로 이 잔차들을 학습하는 모델 생성
3. 생성모델 기반 예측하며, 예측 값에 Learning-rate 를 곱하여 업데이트 한다.

기존 부스팅 기법 문제점

1. 실제 값들의 평균과 실제 값의 차이인 잔차(Residual)를 구한다.
2. 데이터로 이 잔차들을 학습하는 모델 생성

Boost 특징

Aa 컬럼	XGBoost	light GBM	Catboost
사용 트리	Level-wise(넓이 우선순회 트리)	Leaf-wise(깊이 우선순회 트리)	Level-wise(넓이 우선순회 트리)

▼ RandomForest

▼ 장점



의사결정트리 여러 개를 결합한 모델로 단일 의사결정트리에 비해 예측력이 개선됨

▼ 모델

```
sel = [ '총세대수', '전용면적', '공가수', '단지내주차면수','qcut_총세대수', '자격유형', '전용면적별세대수',
'10분내버스정류장수', '임대건물구분_lbl', '공급유형_lbl', '지역_lbl', '단지코드_lbl','단지코드_Type']
lable_name = 'log_등록차량수'
X = train_df[sel]
y = train_df[lable_name]
test_X = test_df[sel]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
```

```
model = RandomForestRegressor(n_jobs=-1)
model.fit(X_train, y_train)
```

```

pred = model.predict(X_test)
print("학습(score) :", model.score(X_train, y_train) ) # 결정계수
print("테스트(score) :", model.score(X_test, y_test) ) # 결정계수

학습(score) : 0.9991547092426224
테스트(score) : 0.9986196764758375

```

```

model_RF = RandomForestRegressor(n_estimators = 1000, random_state=0, n_jobs=-1) # 랜덤포레스트 인자 변경
model_RF.fit(X_train, y_train)
score = cross_val_score(model_RF, X_train, y_train,
                       cv=5, scoring="neg_mean_absolute_error") # neg_mean_squared_error
m_score = np.abs(score.mean())
print("RandomForestRegressor Score : {}".format(m_score))

RandomForestRegressor Score : 0.0325729209923925

```

▼ 제출

```

model_last = RandomForestRegressor(n_estimators = 1000, random_state=0, n_jobs=-1)
model_last.fit(X_train, y_train)
pred = model_last.predict(test_X)

public score (Dacon) : 116.24762

```



랜덤 포레스트 모델을 사용한 결과 Dacon 스코어를 높이기 위해 다른 머신러닝 모델을 사용해보기로 했다.

▼ lightGBM

▼ 장점



Gradient Boosting Model을 기반으로 하되 학습 시간이 더 짧고 과적합을 방지 기능이 뛰어나 캐글 경진대회에서 활용되는 모델

▼ 모델



'교통편의성', '총세대수주차면수' 변수 추가

```

all_df['교통편의성'] = all_df['10분내지하철수'] + all_df['10분내버스정류장수']

all_df['총세대수주차면수'] = all_df['단지내주차면수']/all_df['총세대수']

```

```

sel = [ '총세대수', '전용면적', '공가수', '단지내주차면수', 'qcut_총세대수', '자격유형_lbl', '전용면적별세대수',
       '10분내버스정류장수', '10분내지하철수', '임대건물구분_lbl', '공급유형_lbl', '지역_lbl', '단지코드_lbl', '단지코드_Type',
       '교통편의성', '총세대수주차면수' ]
lable_name = 'log_등록차량수'
X = train_df[sel]
y = train_df[lable_name]
test_X = test_df[sel]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)

```

```

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
import lightgbm as lgb

```

▼ 하이퍼파라미터

```
boosting_type': 'gbdt',
    'colsample_bytree': 0.7,
    'is_unbalance': False,
    'learning_rate': 0.02,
    'min_child_samples': 20,
    'num_leaves': 56,
    'subsample': 0.5233384321711397,
    'n_estimators': 2000
```

▼ 제출

```
now_time = time.time()
m_lgbm1 = lgb.LGBMRegressor(**hyperparameters)
m_lgbm1.fit(X_train, y_train)
score = cross_val_score(m_lgbm1, X_train, y_train,
                        cv=5, scoring="neg_mean_absolute_error")
m_score = np.abs(score.mean())
pro_time = time.time() - now_time
print(pro_time)
print("LightGBM Score : {}".format(m_score))

LightGBM Score : 0.02658467643597
(DACON) 100.2721826176
```

▼ Catboost

▼ 장점



범주형 변수를 자동으로 전처리, 모델 튜닝이 간소화

▼ 최종

```
from catboost import CatBoostRegressor
model = CatBoostRegressor(
    loss_function='MAE',
    n_estimators=500,
    learning_rate=0.1,
    random_state=42)

model.fit(X, y)
pred = model.predict(test_X)
print("Score : {}".format(model.score(X, y)))

Score: 0.9623954443283291 public: 97.4565631289
→ private: 109.87482 (65등)
```

▼ 하이퍼파라미터 조정

```
random_state=50 | Score: 0.9262627160593916 public: 101.8090432
n_estimators=600 | Score: 0.9315280869247932 public: 102.9365546
learning_rate=0.1 | Score: 0.9623954443283291 public: 96.91873251
```

▼ 모델 이력 및 최종 선택

이력

Aa 모델이름	Score	public
<u>lightGBM</u>	0.029764732384683697	100.2721826176
<u>catboost</u>	0.9623954443283291	97.4565631289
<u>Untitled</u>		

결론

▼ 아쉬웠던 부분

- 💡 6월 10일부터 시작된 대회였지만, 7월 30일 최종 코드 제출까지 실제 참여기간은 2주 남짓이었기 때문에 짧았던 수행기간이 아쉬움
- 💡 스코어의 기준이 무엇인지 아직도 감을 못잡았다는 게 참 답답함
- 💡 제한된 분석 수행기간 안에 경진대회에 적용해 볼 수 있을 여러 머신러닝 모델을 수행하지 못 한 것이 아쉬움

▼ 좋았던 부분

- 💡 다양한 머신러닝 기법들을 알 수 있는 기회가 되었으며, 팀원들과 소통하는 시간이 되었음
- 💡 대회에 참가하며 scikit-learn, pandas, seaborn 등 파이썬 라이브러리와 코드에 익숙해지는 계기가 되었다고 생각함
- 💡 짧은 참여기간이었고 스코어 개선에 끼친 영향이 미미한 적도 있었지만, 데이터 분석 경진대회에서 활용해 볼 수 있는 분석 노하우(자료형 변환 또는 qcut으로 새로운 feature 생성, 변수 간의 관계를 활용한 새로운 feature 생성)를 궁리하고 시도해 볼 수 있었음