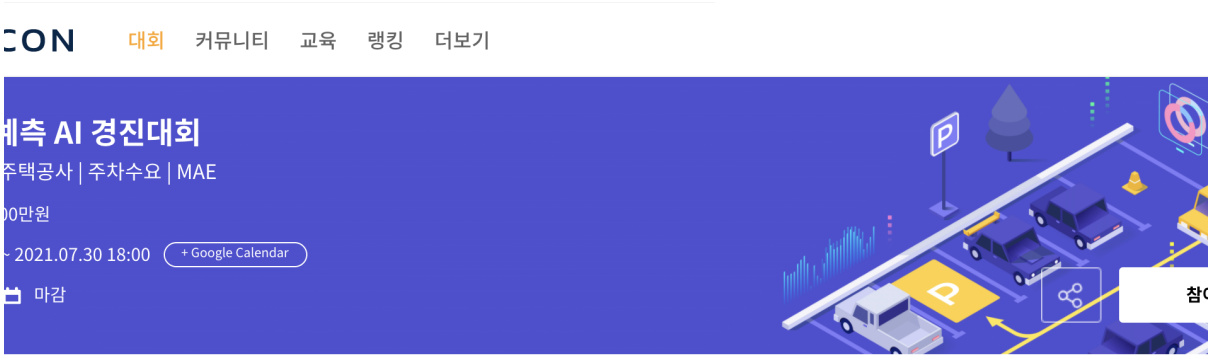


주차 수요 데이콘 AI 경진대회

82	Team_ENTER		110.75797	62	6일 전
35	Team_ENTER		92.58984	62	6일 전

2021.06.10 ~ 2021.07.30 18:00

정형 | 한국토지주택공사 | 주차수요 | MAE

목차

https://dacon.io/competitions/official/235745/overview/description

▼ 대회의 목적

> 💡 결국에 이 모델의 목적이 아파트 단지 설계 시 적절한 주차 수요에 맞는 '**단지내주차면수**'를 산출하기 위함인데 이미 데이터 컬럼에 존재한다. 찾아본 바로는 설계한 단지내 주차면수가 적절한지 판단하는것이 맞고, 이에 주차수요를 예측하여 설계한 주차면수가 과한지 부족한지를 판단하기 위함이다.

▼ 오류처리

1. 대회 오류 내용 공지를 확인한다
2. train, test 데이터에서 오류 처리한 데이터를 train_df_jm, test_df_jm의 csv 파일로 저장하였다.

- (관련 데이터) 아래와 같이 총 3개 단지 6개 코드에서 같은 유형의 오류가 확인되었습니다.

※ 동일한 단지에 코드가 2개로 부여된 단지 코드 (3쌍) : ['C2085', 'C1397'], ['C2431', 'C1649'], ['C1036', 'C2675']

- C2675 단지는 테스트셋, 나머지는 트레인셋 입니다.

3. 단지코드 등 기입 실수로 데이터 정제 과정에서 매칭 오류 발생

- (오류 내용) 단지코드 등 기입 실수로 총세대수가 주차면수에 비해 과하게 많거나 적은 경우가 발생하였고, 점검 결과 일부 데이터의 단지코드, 총세대수, 주차면수 등에서 오류가 검출되었습니다.

- (발생 원인) 원천데이터 수집 과정에서 단지 코드 등이 잘못 기입되었고 이를 인지하지 못한 채 데이터 정제를 하여 오류가 발생하였습니다.

- (관련 데이터) 아래와 같이 총 9개 단지에서 같은 문제가 확인되었습니다.

※ 실수가 발생한 단지 코드 (9개 단지) : ['C2335', 'C1327', 'C1095', 'C2051', 'C1218', 'C1894', 'C2483', 'C1502', 'C1988']

- C2335, C1327 단지는 테스트셋, 나머지는 트레인셋 입니다.

4. 오류 데이터 처리 방안

- 제공한 데이터상에 문제점이 발견되었으나 대회 중반에 들어선 시점에서 오류를 수정하여 재배포할 경우 혼란이 가중될 것이 우려되어 데이터 수정/재배포는 없을 예정입니다.

- 1번 오류의 경우 해당 사실을 감안하여 분석을 진행해 주시기 바라며, 2, 3번 오류의 경우에는 문제가 된 단지들을 트레인셋에서 제외하기를 권장드리고, 테스트셋에서는 평가 시 제외하고자 합니다.

- 테스트셋에서 평가 제외되는 데이터는 'C2675'(2번 사항에 해당), 'C2335', 'C1327'(3번 사항에 해당) 3개 단지입니다.

- 제출 양식은 변경되지 않으니 기존 제출 양식에 따라 제출 부탁드립니다.('C2675', 'C2335', 'C1327' 단지 예측 결과는 평가되지 않습니다.)

2, 3번 오류의 경우에는 문제가 된 단지들을 트레인셋에서 제외하기를 권장드리고, 테스트셋에서는 평가 시 제외하고자 합니다.

💡 정리를 하자면 전체적인 데이터 불일치, 오류가 일어났고, train, test 데이터에서 수정이 필요하다.

*오류단지들
train : ['C2085', 'C1397', 'C2431', 'C1649', 'C1036','C1095', 'C2051', 'C1218', 'C1894', 'C2483', 'C1502', 'C1988']
test : ['C2335', 'C1327', 'C2675']

대회측에서 문제가 되는 단지들을 제외하길 권장하므로 위에 있는 데이터를 제외시키는 전처리 작업 후 진행하는것이 좋을 듯 하다.

> 문제가 있다면 다른사람이 동일한 모델로 다르게 예측하여 제출해본 바에 의하면,
> **1) test 데이터의 오류단지 3개를 예측하여 예측치를 포함한 제출본**
> **2) test 데이터의 오류단지 3개를 예측하지 않고 0으로 입력한 제출본**
> 두개의 제출본에서 리더보드 상의 점수 차이가 존재함.
>
> 예측 결과는 평가되지 않는다는 공지가 PUBLIC SCORE가 아닌 PRIVATE SCORE에 한정된 것이라 하더라도, 대부분의 참가자들은 PUBLIC SCORE(리더보드)를 기준으로 모델을 개선하는 방향성을 잡는데,이렇게 되면 참가자들은 모델의 성능을 높이기 위해 잘못된 데이터까지도 잘 예측하는 모델을 만들어야 한다.
>
> 대회측 피드백) 아직 해당 데이터에 대한 예외처리가 이뤄지지 않았습니다. 금일 중에 예외 처리 및 기존 제출물 적용을 완료하도록 하겠습니다.
>
> 따라서 예외처리 적용이 완료되면 오류단지를 0으로 처리후 예측하는것이 좋다.

▼ EDA

1. 결측치 확인
2. **train 과 test 데이터의 평균 & 편차 크기 확인**
3. **train - test 차집합, subplot 그려보기**
4. **상관계수가 가장 높은 '단지내주차면수' & '총세대수' 분포도 확인**
5. **거주율 컬럼 추가후 등록차량수와의 상관관계 확인**

- **train & test 결측치 확인**

```
train.isnull().sum()
```

```
단지코드                                0
총세대수                                0
임대건물구분                              0
지역                                  0
공급유형                                0
전용면적                                0
전용면적별세대수                            0
공가수                                 0
자격유형                                0
임대보증금                             569
임대료                                569
도보 10분거리 내 지하철역 수(환승노선 수 반영)     207
도보 10분거리 내 버스정류장 수                   0
단지내주차면수                             0
등록차량수                               0
dtype: int64
```

```
test.isnull().sum()
```

```
단지코드                                0
총세대수                                0
임대건물구분                              0
지역                                  0
공급유형                                0
전용면적                                0
전용면적별세대수                            0
공가수                                 0
자격유형                                2
임대보증금                             180
임대료                                180
도보 10분거리 내 지하철역 수(환승노선 수 반영)      38
도보 10분거리 내 버스정류장 수                   0
단지내주차면수                             0
dtype: int64
```

- **train 과 test 데이터의 평균 & 편차가 큰 차이는 없는것으로 보인다.**

```
train.describe().T
```

	count	mean	std	min	25%	50%	75%	max
총세대수	2869.0	887.654235	517.795084	26.00	514.0	775.00	1105.00	2568.0
전용면적	2869.0	44.421394	32.072217	12.62	32.1	39.84	51.05	583.4
전용면적별세대수	2869.0	102.699895	133.287517	1.00	14.0	60.00	142.00	1865.0
공가수	2869.0	12.915999	10.679931	0.00	4.0	11.00	20.00	55.0
지하철수	2662.0	0.181818	0.432606	0.00	0.0	0.00	0.00	3.0
버정수	2869.0	3.709655	2.676486	0.00	2.0	3.00	4.00	20.0
단지내주차면수	2869.0	591.934472	391.292702	13.00	277.0	500.00	812.00	1798.0
등록차량수	2869.0	550.127571	430.465261	13.00	209.0	479.00	761.00	2550.0

```
test.describe().T
```

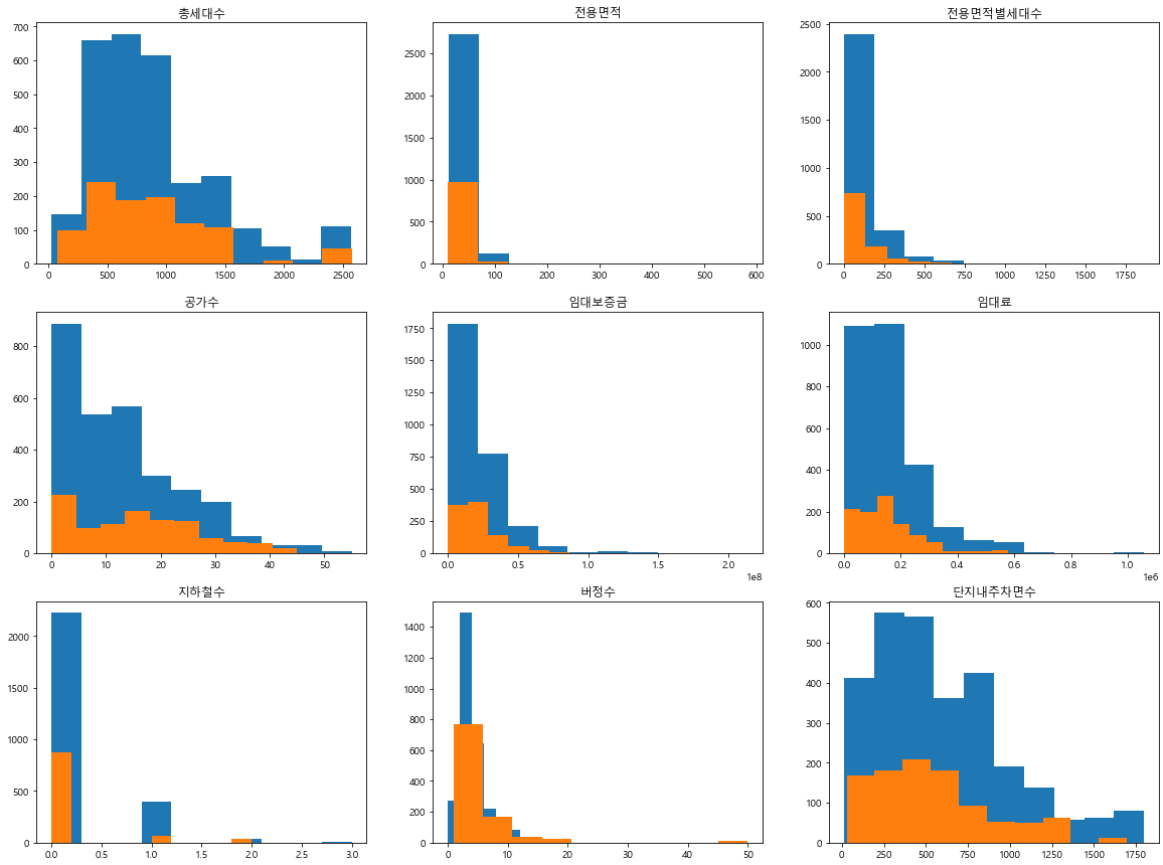
	count	mean	std	min	25%	50%	75%	max
총세대수	1008.0	869.378968	535.908018	75.00	492.50	754.00	1161.00	2572.0
전용면적	1008.0	43.788462	36.105647	9.96	33.15	39.72	47.41	583.4
전용면적별세대수	1008.0	101.093254	126.674450	1.00	14.00	60.00	140.00	1341.0
공가수	1008.0	15.630952	11.116013	0.00	6.00	16.00	23.00	45.0
지하철수	970.0	0.138144	0.437519	0.00	0.00	0.00	0.00	2.0
버정수	1008.0	4.597222	5.391566	1.00	2.00	3.00	5.00	50.0
단지내주차면수	1008.0	546.678571	341.278739	29.00	286.00	458.00	706.50	1696.0

- **train - test 차이가 나는 부분을 통하여 train에만 있는 유형이 무엇인지 파악하기**

```
columns =['임대건물구분','지역','공급유형','자격유형']

for i in columns:
    result = list(set(train[i].unique())-set(test[i].unique()))
    print('train - test 차집합:',i+":",result)
```

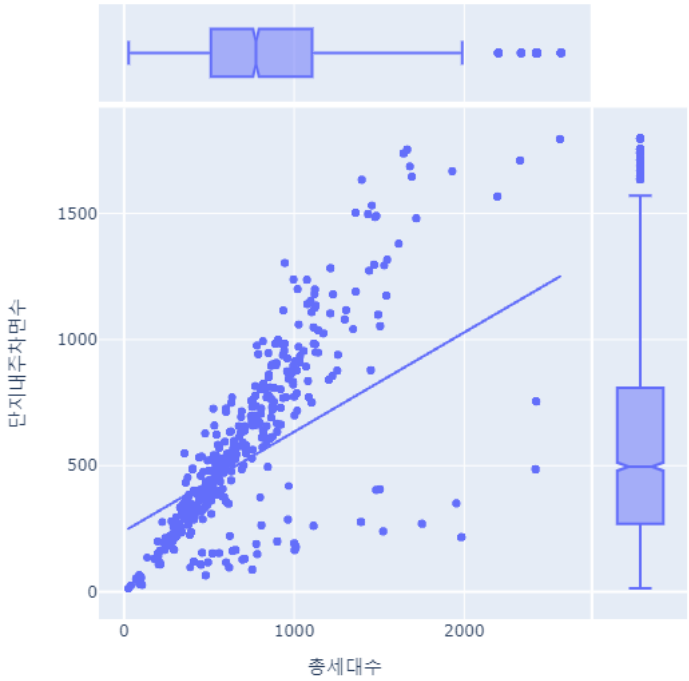
```
train - test 차집합: 임대건물구분: []
train - test 차집합: 지역: ['서울특별시']
train - test 차집합: 공급유형: ['장기전세', '공공임대(5년)', '공공분양']
train - test 차집합: 자격유형: ['F', 'O', 'B']
```

- **train & test 데이터 subplot으로 겹치게 그리기**

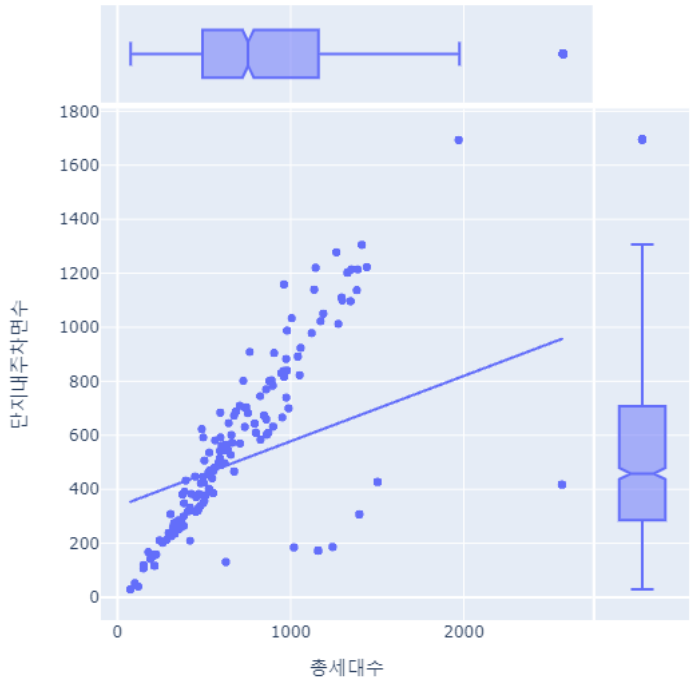


- **상관계수가 가장 높은 '단지내주차면수' & '총세대수' 분포도 확인해보기**

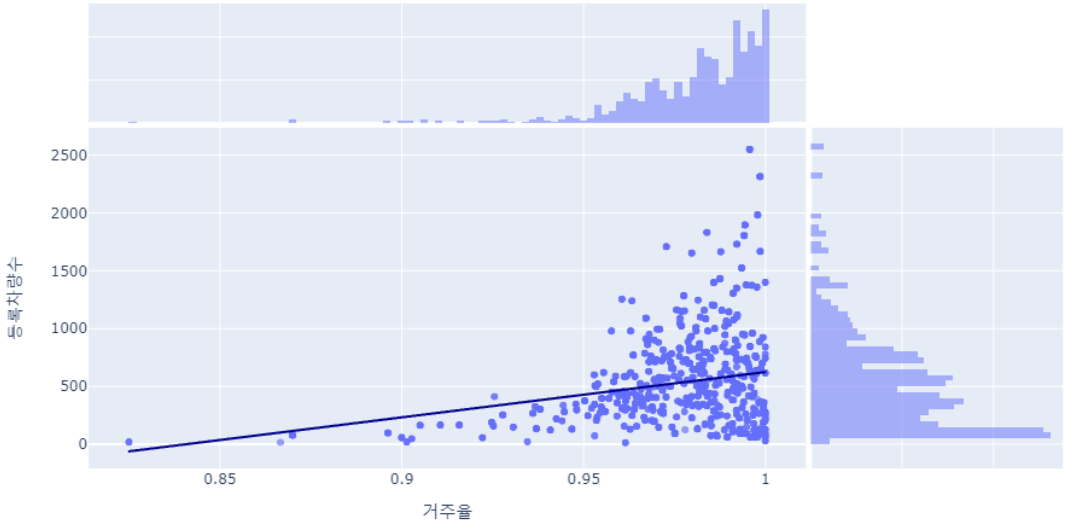
1. train 데이터



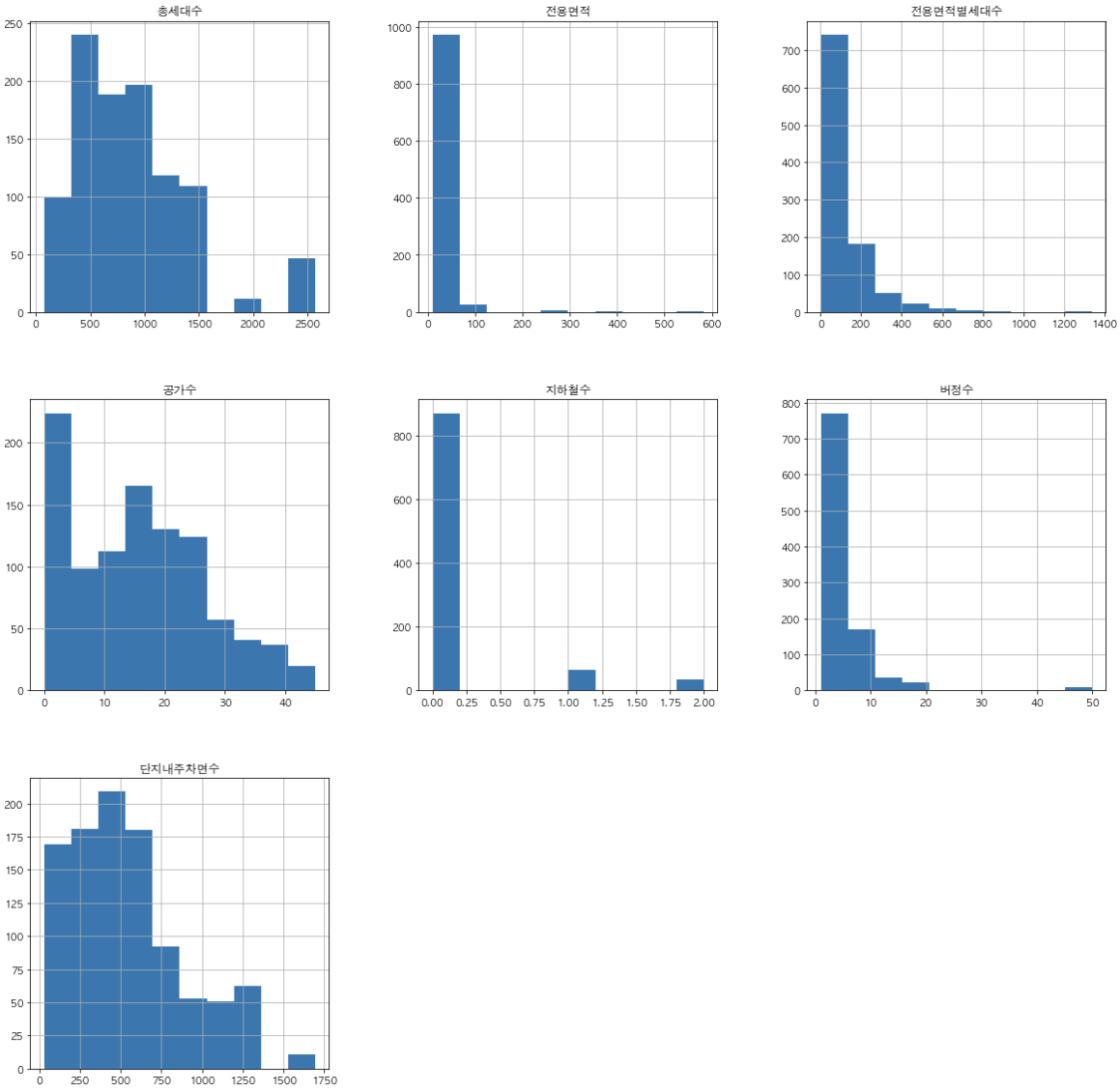
2. test 데이터



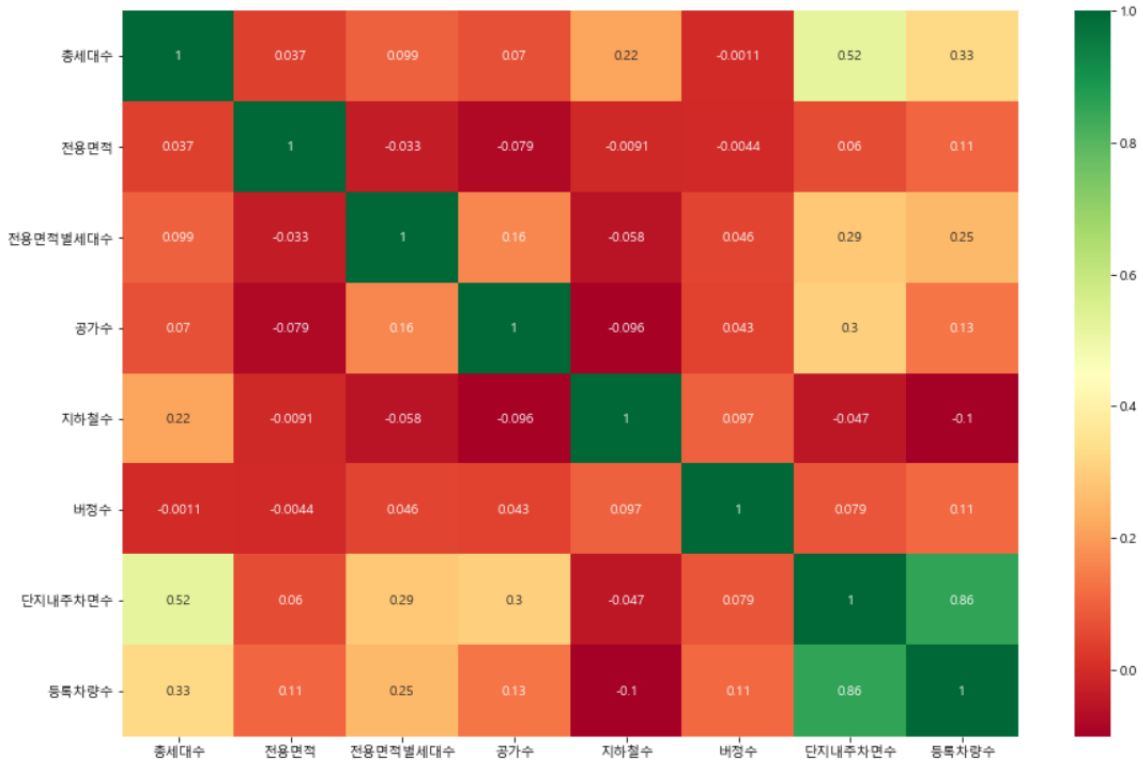
- **거주율 컬럼 추가후 등록차량수와의 상관관계 그리기**



- **전체 Feature와 등록차량수와의 상관관계 그리기**

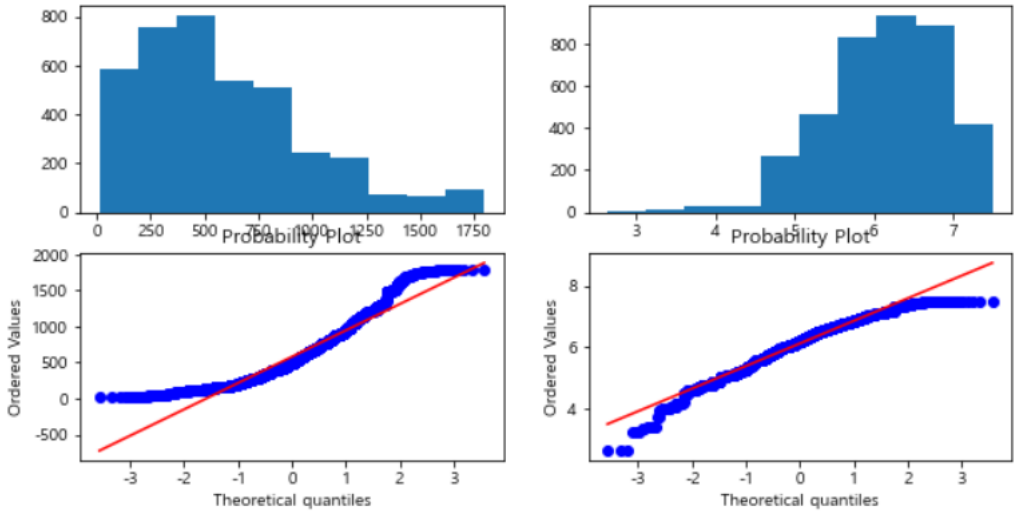


- **상관관계 확인 - heatmap**
  > 등록차량수와 상관관계가 높은 총세대수 , 전용면적별세대수, 단지내주차면수 3가지 feature에 관하여 log 처리

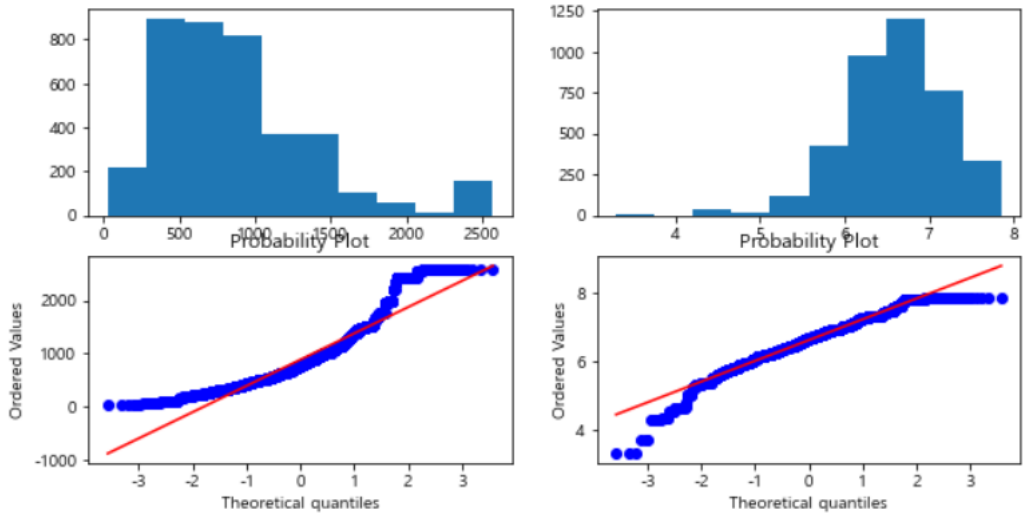


단지내주차면수, 총세대수, 전용면적별세대수 log 정규화 확인

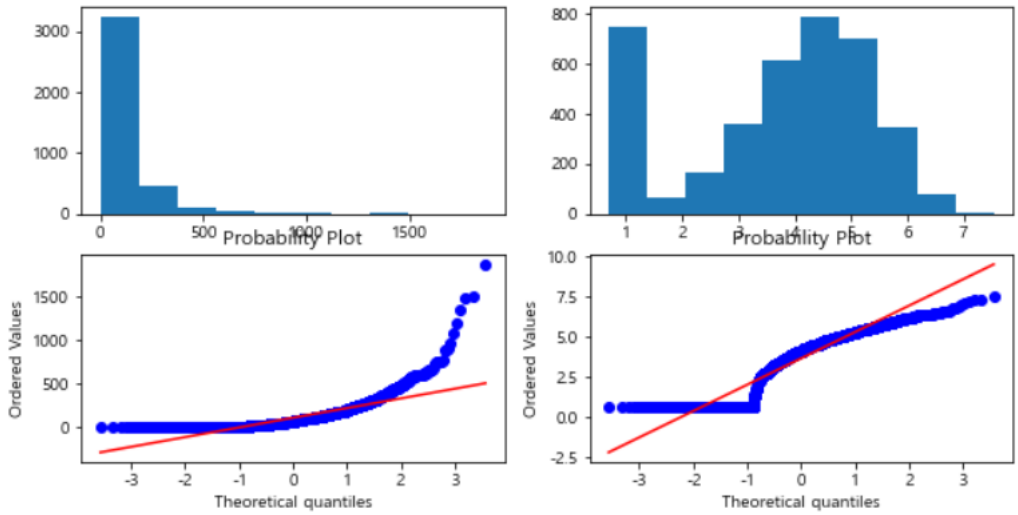
- **단지내주차면수(오른쪽이 log 씌운값) > 넣기**



- **총 세대수(오른쪽이 log 씌운값) >넣기**



- **전용면적별세대수(오른쪽이 log 씌운값) > 넣지 않기**



결론 :

- train 데이터와 test 데이터가 최대한 서로 비슷해야 훈련을 잘 할 수 있으므로 스코어가 잘 나올 것이다.
- 두 데이터의 모양, 평균편차를 확인해보니 크게 다른 점이 보이는 것이 없어 결측치를 올바르게 처리해주고 컬럼들간의 상관관계를 보고 변수를 잘 선택해야겠다.

▼ 데이터전처리

1. 결측치 처리 (임대보증금&임대료 - 0처리, 지하철수 DecisionTree 예측)
2. train 데이터에만 있는 '서울특별시' 데이터 삭제
3. 공급유형/자격유형 묶기
4. age_gender 병합시키기
5. 주차면수대비총세대수비율 & 거주율 추가
6. 표준화 및 이상치 제거

7. Feature 중요도 뽑고 변수 선택하기

결측값처리

- 임대보증금 & 임대료 - 결측값 0처리 (등록차량수를 구하는데 지장이 없다고 판단)

http://www.reacademy.org/rboard/data/krea2_new/71_10.pdf

```
train.loc[train.임대보증금=='-', '임대보증금'] = np.nan
test.loc[test.임대보증금=='-', '임대보증금'] = np.nan
train['임대보증금'] = train['임대보증금'].astype(float)
test['임대보증금'] = test['임대보증금'].astype(float)

train.loc[train.임대료=='-', '임대료'] = np.nan
test.loc[test.임대료=='-', '임대료'] = np.nan
train['임대료'] = train['임대료'].astype(float)
test['임대료'] = test['임대료'].astype(float)

train[['임대보증금', '임대료']] = train[['임대보증금', '임대료']].fillna(0)
test[['임대보증금', '임대료']] = test[['임대보증금', '임대료']].fillna(0)
```

- 지하철 결측값 예측 - 결측값이 있는 행을 y로 놓고 나머지행 X를 이용해 값을 예측(Decision tree)

—> 결론은 안좋아서 0처리,,🤔 이것도 의문

```
from sklearn.tree import DecisionTreeClassifier
sel = ['총세대수','임대건물구분','지역',
'공급유형', '공가수','자격유형', '버정수','단지내주차면수']

X_train = train_df[sel]
y_train = train_df['지하철수']
X_test = test_df[sel]
```

```
model = DecisionTreeClassifier(max_depth=7, random_state=0)
model.fit(X_train, y_train)
pred = model.predict(X_test)

print("학습(score) :", model.score(X_train, y_train) ) # 결정계수
```

---

train 에만 있는 데이터 유형 처리

- train과 test차이

```
train - test 차집합: 임대건물구분: []
train - test 차집합: 지역: ['서울특별시']
train - test 차집합: 공급유형: ['장기전세', '공공임대(5년)', '공공분양']
train - test 차집합: 자격유형: ['F', 'O', 'B']
```

- 서울은 해결 불가능 - train에서 제외시킨다 →🤔 점수가 좋진 않았다. 왜???

```
train = train.loc[~(train['지역'] =='서울특별시'),:]
```

- 공급유형 묶기

💡 **국민임대와 공공임대의 차이**

- LH공사와 SH공사에서 제공하는 국민임대는 국가 재정과 국민주택기금을 지원받아 국가, 지방자치단체, 한국토지주택공사 또는 지방공사가 건설, 공급하는 주택을 의미한다. 저렴한 임대료를 지불하고 장기(30년) 임대가 가능하다. 장기 거주는 가능하지만 해당 주거지를 매입할 수 없다는 단점이 있다.

- 반면 공공임대는 5년 또는 10년 후 임대기간이 종료되었을 때 입주자에게 우선 분양전환혜택을 준다.

```
train.loc[train.공급유형.isin(['공공임대(5년)', '공공분양', '공공임대(10년)', '공공임대(분납)']), '공급유형'] = '공공임대(5년/10년/분납/분양)'
test.loc[test.공급유형.isin(['공공임대(5년)', '공공분양', '공공임대(10년)', '공공임대(분납)']), '공급유형'] = '공공임대(5년/10년/분납/분양)'
train.loc[train.공급유형.isin(['장기전세', '국민임대']), '공급유형'] = '국민임대/장기전세'
test.loc[test.공급유형.isin(['장기전세', '국민임대']), '공급유형'] = '국민임대/장기전세'
train.공급유형.unique()
```

```
array(['국민임대/장기전세', '공공임대(50년)', '영구임대', '임대상가', '공공임대(5년/
10년/분납/분양)',
       '행복주택'], dtype=object)
```

- 자격유형 묶기

💡 **공공임대주택 공급대상**

영구임대: 생계급여 또는 의료급여 수급자 등[소득 1분위] - 1
국민임대: 무주택세대구성원[소득 2~4분위] - 3
장기전세: 무주택세대구성원[소득 3~4분위] - 2
공공임대(5년/10년/분납): 무주택세대구성원[소득 3~5분위] - 3
행복주택: 무주택세대구성원/무주택자[소득 2~5분위] - 4

의문점🤔: 소득분위를 나누면 13개 항목인데 왜 자격유형의 항목수는 15개일까??
모르겠다.. 그냥 묶어버리쟈!

```
train.loc[train.공급유형.isin(['국민임대/장기전세']), '자격유형'].value_counts() ## 5
```

```
A    1511
H     155
E      34
B      21
G       9
Name: 자격유형, dtype: int64
```

```
train.loc[train.공급유형.isin(['공공임대(50년)']), '자격유형'].value_counts()
```

```
A    31
Name: 자격유형, dtype: int64
```

```
train.loc[train.공급유형.isin(['영구임대']), '자격유형'].value_counts()
```

```
C    95
I    49
F     3
E     3
A     2
Name: 자격유형, dtype: int64
```

```
train.loc[train.공급유형.isin(['임대상가']), '자격유형'].value_counts()
```

```
D    562
Name: 자격유형, dtype: int64
```

```
train.loc[train.공급유형.isin(['공공임대(5년/10년/분납/분양)']), '자격유형'].value_cou
```

```
A    185
D      7
Name: 자격유형, dtype: int64
```

```
train.loc[train.공급유형.isin(['행복주택']), '자격유형'].value_counts()
```

```
J    103
L     33
K     33
N     30
M      2
O      1
Name: 자격유형, dtype: int64
```

💡 **공급유형이 겹치는 자격유형을 제외한 자격유형 묶기**

E 의 경우 영구임대와 국민임대/장기전세 항목과 겹치지만 국민임대/장기전세쪽이 많기 때문에 같이 묶음
기존에 train에만 있는 공급유형과 자격유형을 제거해주었지만 묶음으로서 해결됨

```
train.loc[train.자격유형.isin(['H', 'B', 'E', 'G']), '자격유형'] = '국민임대/장기전세_공급대상'
test.loc[test.자격유형.isin(['H', 'B', 'E', 'G']), '자격유형'] = '국민임대/장기전세_공급대상'

train.loc[train.자격유형.isin(['C', 'I', 'F']), '자격유형'] = '영구임대_공급대상'
test.loc[test.자격유형.isin(['C', 'I', 'F']), '자격유형'] = '영구임대_공급대상'

train.loc[train.자격유형.isin(['J', 'L', 'K', 'N', 'M', 'O']), '자격유형'] = '행복주택_공급대상'
test.loc[test.자격유형.isin(['J', 'L', 'K', 'N', 'M', 'O']), '자격유형'] = '행복주택_공급대상'
```

- 자격유형을 묶고나서 자격유형에 있었던 2개의 결측값을 해결

```
test.loc[test['자격유형'].isnull() & test['단지코드'].isin(['C2411']) ,'자격유형'] = 'A'
test.loc[test['자격유형'].isnull() & test['단지코드'].isin(['C2253']) ,'자격유형'] = '영구임대_공급대상'
```

예측에 도움이 될만한 컬럼추가

- age_gender 병합시키기

```
train = train.merge(age_gender, how='left')
test = test.merge(age_gender, how='left')
```

- 주차면수대비총세대수비율 & 거주율 추가

```
train['주차면수대비총세대수비율'] = train['총세대수']/train['단지내주차면수']
train['거주율'] =1 - (train['공가수']/train['총세대수'])

test['주차면수대비총세대수비율'] = test['총세대수']/test['단지내주차면수']
test['거주율'] =1 - (test['공가수']/test['총세대수'])
```

표준화

- 표준화 전 feature들의 분산값 확인

```
print('feature 들의 평균 값')
print(train_df.mean())
print('\nfeature 들의 분산 값')
print(train_df.var())
```

```
feature 들의 평균 값
총세대수              8.892529e+02
임대건물구분            8.006385e-01
지역                5.397304e+00
공급유형              2.569351e+00
전용면적              4.432630e+01
전용면적별세대수          1.030905e+02
공가수               1.297198e+01
자격유형              7.800639e-01
임대보증금             1.967736e+07
임대료               1.477451e+05
버정수               3.697410e+00
단지내주차면수           5.906552e+02
지하철수              1.837531e-01

feature 들의 분산 값
총세대수              2.720567e+05
임대건물구분            1.596731e-01
지역                1.962847e+01
공급유형              1.237471e+00
전용면적              1.038610e+03
전용면적별세대수          1.791566e+04
공가수               1.149641e+02
자격유형              1.440610e+00
임대보증금             3.221121e+14
임대료               1.590992e+10
버정수               7.206846e+00
단지내주차면수           1.550450e+05
지하철수              1.862370e-01
```

- 표준화 진행

```
scaler = StandardScaler()
scaler.fit(train_df)
train_scaled = scaler.transform(train_df)
```

- 표준화 후 분산값 확인

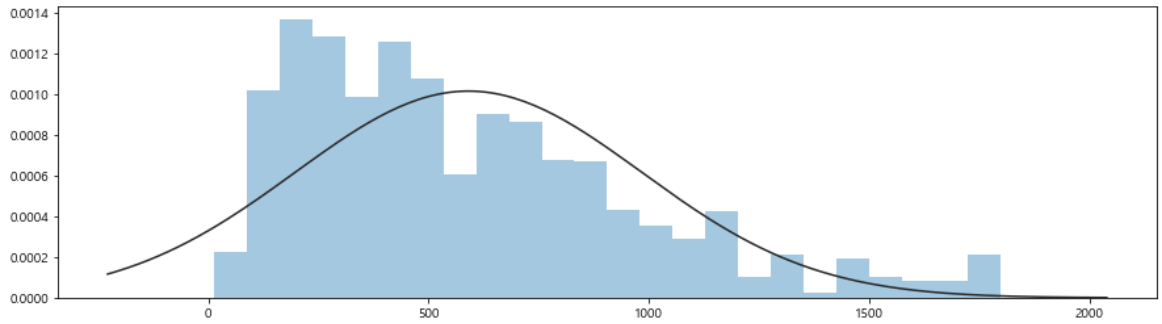
```
print("\n=========== 표준화 ==============\n")

train_df_scaled = pd.DataFrame(data=train_scaled, columns=train_df.columns)
```

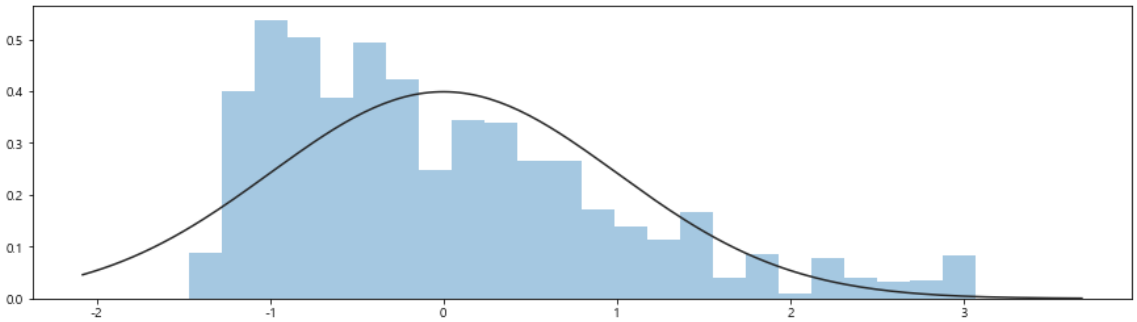
```
print('feature 들의 평균 값')
print(train_df_scaled.mean())
print('\nfeature 들의 분산 값')
print(train_df_scaled.var())
```

```
========== 표준화 =============

feature 들의 평균 값
총세대수           -4.204591e-16
임대건물구분         -1.126138e-14
지역             -4.110070e-16
공급유형           -1.282644e-15
전용면적            1.105103e-16
전용면적별세대수       -2.722193e-16
공가수            3.082159e-16
자격유형           -1.476096e-15
임대보증금           8.481450e-16
임대료            -1.180750e-15
버정수            -2.904933e-16
단지내주차면수        -6.410071e-16
지하철수           -1.819994e-15

 feature 들의 분산 값
 총세대수            1.000355
 임대건물구분          1.000355
 지역              1.000355
 공급유형            1.000355
 전용면적            1.000355
 전용면적별세대수        1.000355
 공가수             1.000355
 자격유형            1.000355
 임대보증금           1.000355
 임대료             1.000355
 버정수             1.000355
 단지내주차면수         1.000355
 지하철수            1.000355
```

- 정규 분포 모양 비교

```
f,ax = plt.subplots(2,1,figsize=(15,9))

# 표준화 전 정규분포
x0 = train_df['단지내주차면수'].dropna().values
sns.distplot(x0, kde=False, rug=False, fit =sp.stats.norm, ax = ax[0]) #커널 밀도 함수, 러그밀도함수 는 안보이고(False) / 정규분포만 보이게(Fi

# 표준화 이후 정규 분포
x1 = train_df_scaled['단지내주차면수'].values
sns.distplot(x1, kde=False, rug=False, fit =sp.stats.norm, ax = ax[1])
plt.show()
```





- 이상치 발견 및 제거

```
df_Zscore = pd.DataFrame()
outlier_dict = {}
outlier_idx_list = []

for one in train_df_scaled.columns:
    df_Zscore[f'{one}_Zscore'] = sp.stats.zscore(train_df_scaled[one])
    outlier_dict[one] = df_Zscore[f'{one}_Zscore'][(df_Zscore[f'{one}_Zscore']>2)|(df_Zscore[f'{one}_Zscore']<-2)]
    outlier_idx_list.append(list(outlier_dict[one].index))
    if len(outlier_dict[one]):
        print(one,'이상치값들\n',outlier_dict[one])
    else:
        print(one,'이상치가 없다')
    print()
```

```
임대건물구분 이상치값들
 80    -2.004
81    -2.004
82    -2.004
83    -2.004
93    -2.004
      ...
826   -2.004
827   -2.004
828   -2.004
829   -2.004
830   -2.004
Name: 임대건물구분_Zscore, Length: 562, dtype: float64

지역 이상치가 없다
```

- 이상치를 최대 & 최소값에 맞춤

```
cols = ['총세대수', '임대건물구분', '지역', '공급유형', '전용면적', '전용면적별세대수', '공가수',
       '자격유형', '임대보증금', '임대료', '버정수', '단지내주차면수',  '지하철수', '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)', '주차면수대비총세대수비율', '거주율']
for i in cols:
    idx = train_df_scaled[train_df_scaled[i]>2].index
```

```
    idx2 = train_df_scaled[train_df_scaled[i]<-2].index
    train_df_scaled.loc[idx,[i]] = 2
    train_df_scaled.loc[idx2,[i]] = -2
```

> 대회가 다끝나고 강사님 강의를 듣다가 안 사실이지만 표준화를 진행하고 나서 모델 스코어가 나아지지 않았는데 그 이유가 모델의 표준화를 시킬때 동일한 기준을 가지고 적용시켜야 했다. 하지만 우리는 train과 test를 각각의 데이터를 기준으로 표준화를 진행했다.ㅠ 흑흑

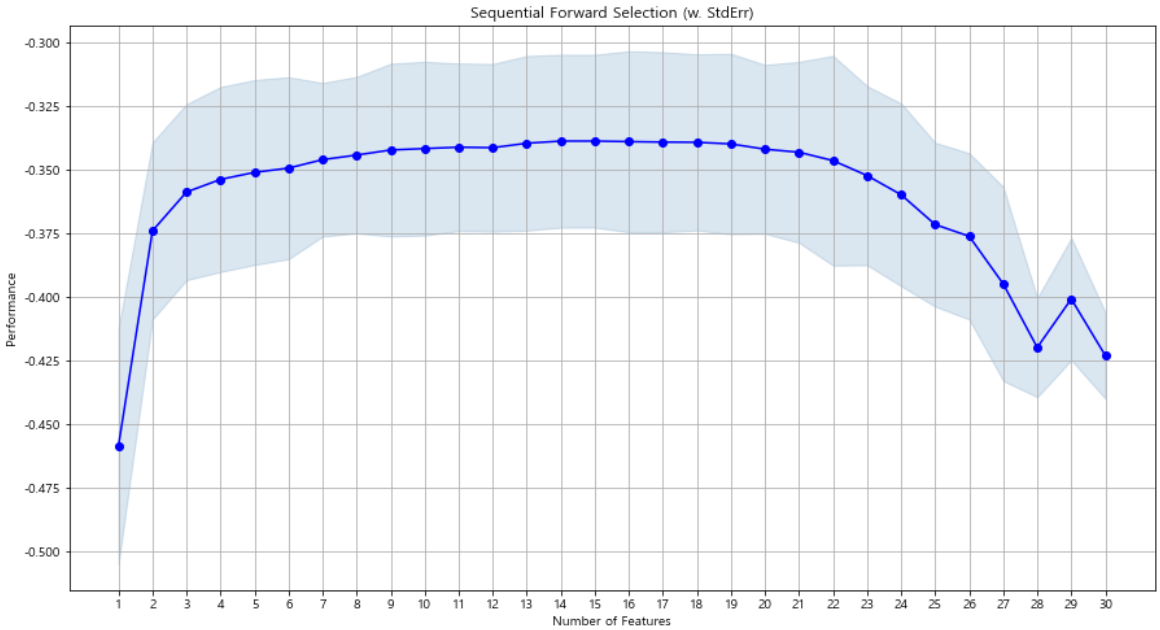
표준화도 시키고 정규화도 해주었는데 그게 맞는지 잘 모르겠다?

Feature 중요도 뽑기 - SFS

http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/

- 적절한 feature의 개수

```
sfs = SFS(LinearRegression(),
          k_features=30,
          forward=True,
          floating=False,
          scoring='neg_mean_absolute_error',
          cv=5)

sfs = sfs.fit(X, y)
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err', figsize=(15,8))

plt.title('Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()
```



14~16개의 feature를 사용하는것이 적절해 보인다.

> 💡 **순방향선택**
>
> 하나의 예측 변수로 시작하여 더 반복적으로 추가합니다. 각 후속 반복에서 성능 기준에 따라 나머지 원래 예측 변수 중 가장 좋은 것이 추가됩니다.

```
#Define Sequential Forward Selection (sfs)
sfs = SFS(LinearRegression(),
           k_features=14,
           forward=True,
           floating=False,
           scoring = 'r2',
           cv = 0)
#Use SFS to select the top 5 features
sfs.fit(X, y)

#Create a dataframe for the SFS results
df_SFS_results = pd.DataFrame(sfs.subsets_).transpose()
df_SFS_results
```

	feature_idx	cv_scores	avg_score	feature_names
1	(9,)	[0.6430981028552802]	0.643098	(단지내주차면수,)
2	(9, 32)	[0.7296502558565444]	0.72965	(단지내주차면수, 임대건물구분_lbl)
3	(9, 32, 40)	[0.7526811097756674]	0.752681	(단지내주차면수, 임대건물구분_lbl, 거주율)
4	(9, 32, 39, 40)	[0.7677044903888961]	0.767704	(단지내주차면수, 임대건물구분_lbl, 주차면수대비층세대수비율, 거주율)
5	(9, 32, 33, 39, 40)	[0.7796403290829406]	0.77964	(단지내주차면수, 임대건물구분_lbl, 지역_lbl, 주차면수대비층세대수비율, 거주율)
6	(6, 9, 32, 33, 39, 40)	[0.7874582153276072]	0.787458	(임대료, 단지내주차면수, 임대건물구분_lbl, 지역_lbl, 주차면수대비층세대수비...
7	(6, 9, 32, 33, 34, 39, 40)	[0.7917222495908798]	0.791722	(임대료, 단지내주차면수, 임대건물구분_lbl, 지역_lbl, 공급유형_lbl, 주...
8	(6, 9, 32, 33, 34, 36, 39, 40)	[0.7964974041965527]	0.796497	(임대료, 단지내주차면수, 임대건물구분_lbl, 지역_lbl, 공급유형_lbl, q...
9	(3, 6, 9, 32, 33, 34, 36, 39, 40)	[0.7986059483931915]	0.798606	(공가수, 임대료, 단지내주차면수, 임대건물구분_lbl, 지역_lbl, 공급유형_l...
10	(3, 5, 6, 9, 32, 33, 34, 36, 39, 40)	[0.8000460209519832]	0.800046	(공가수, 임대보증금, 임대료, 단지내주차면수, 임대건물구분_lbl, 지역_lbl,...

💡 **단계적 선택**

정방향 선택과 역방향 제거의 조합을 기반으로 하는 양방향. 제거된 모델에 예측 변수를 다시 추가하는 것을 재고하기 때문에 이전 두 절차보다 덜 탐욕스러운 것으로 간주됩니다(반대의 경우도 마찬가지). 그럼에도 불구하고 주어진 반복에서 로컬 최적화를 기반으로 고려가 이루어집니다.

```
#Define Sequential Forward Selection (sfs)
sffs = SFS(LinearRegression(),
           k_features=16,
           forward=True,
           floating=True,
           scoring = 'r2',
           cv = 0)
#Use SFS to select the top 5 features
feature_names=sel
sffs.fit(X, y, custom_feature_names=feature_names)

#Create a dataframe for the SFS results
df_SFFS_results2 = pd.DataFrame(sffs.subsets_).transpose()
df_SFFS_results2
```

	feature_idx	cv_scores	avg_score	feature_names
1	(41,)	[0.7698604572968635]	0.76986	(log_단지내주차면수,)
2	(0, 41)	[0.8131095709590975]	0.81311	(층세대수, log_단지내주차면수)
3	(0, 6, 41)	[0.8249525100826771]	0.824953	(층세대수, 임대료, log_단지내주차면수)
4	(0, 3, 6, 41)	[0.828838219005085]	0.828838	(층세대수, 공가수, 임대료, log_단지내주차면수)
5	(0, 3, 6, 41, 44)	[0.8342843250663394]	0.834284	(층세대수, 공가수, 임대료, log_단지내주차면수, log_공가수)
6	(0, 3, 6, 33, 41, 44)	[0.8390899874398635]	0.83909	(층세대수, 공가수, 임대료, 지역_lbl, log_단지내주차면수, log_공가수)
7	(0, 3, 6, 32, 33, 41, 44)	[0.8430925885324062]	0.843093	(층세대수, 공가수, 임대료, 임대건물구분_lbl, 지역_lbl, log_단지내주차...
8	(0, 3, 6, 32, 33, 39, 41, 44)	[0.8468638546528854]	0.846864	(층세대수, 공가수, 임대료, 임대건물구분_lbl, 지역_lbl, 주차면수대비층세대...
9	(0, 3, 6, 7, 32, 33, 39, 41, 44)	[0.850484014054494]	0.850484	(층세대수, 공가수, 임대료, 10분내지하철수, 임대건물구분_lbl, 지역_lbl,...
10	(0, 3, 6, 7, 8, 32, 33, 39, 41, 44)	[0.8526740384165014]	0.852674	(층세대수, 공가수, 임대료, 10분내지하철수, 10분내버스정류장수, 임대건물구분_...

💡 **역방향 제거**

모든 예측 변수로 시작하여 하나씩 반복적으로 제거합니다. 가장 인기 있는 알고리즘 중 하나는 기능 중요도 순위를 기반으로 덜 중요한 예측 변수를 제거하는 RFE(재귀적 기능 제거)입니다.

```
#Build a logistic regression model
model = LinearRegression()
#Define RFE
rfe = RFE(model, 5)
#Use RFE to select the top 5 features
fit = rfe.fit(X, y)

#Create a dataframe for the results
df_RFE_results = []
for i in range(X.shape[1]):
    df_RFE_results.append(
        {
            'Feature_names': sel[i],
            'Selected':  rfe.support_[i],
            'RFE_ranking':  rfe.ranking_[i],
        }
    )

df_RFE_results = pd.DataFrame(df_RFE_results)
df_RFE_results.index.name='Columns'
df_RFE_results
```

Columns	Feature_names	Selected	RFE_ranking
0	총세대수	False	40
1	전용면적	False	34
2	전용면적별세대수	False	35
3	공가수	False	28
4	자격유형	False	32
5	임대보증금	False	38
6	임대료	False	37
7	10분내지하철수	False	25
8	10분내버스정류장수	False	31
9	단지내주차면수	False	36
10	10대미만(여자)	False	9

설명

Sequential Feature Selector

1. Sequential Forward Selection(SFS) - 순방향 선택
2. Sequential Backward Selection(SBS)
3. Sequential Forward Floating Selection(SFFS) - 단계적 선택
4. Sequential Backward Floating Selection(SBFS)
5. RFE(Recursive Feature Elimination) - 역방향 제거

RFE는 가중치 계수(선형모델) 와 feature importance 를 사용하여 기능을 재귀적으로 제거한다. SFS는 사용자 정의 분류기/회귀 성능 메트릭을 기반으로 feature를 제거 혹은 추가합니다.

결론 :

- Feature 중요도를 뽑아보았지만 결론적으로는 스코어가 올라가서 좋지 않았음. 이유를 잘 모르겠다.

▼ 모델 검증 및 평가

모델 검증 및 평가

Aa 모델	피처	하이퍼파라미터	평가	최종 SCORE
Lasso	['총세대수', '전용면적', '전용면적별세대수', '공가수', '자격유형', '10분내버스정류장수', '단지내주차면수', '임대건물구분_lbl', '지역_lbl', '공급유형_lbl','단지코드_lbl']	alpha: 0.5	MAE: 147.95215377675154	115.3201975854
LGBM	[자격유형', '전용면적별세대수', '전용면적', '공가수', '10분내버스정류장수', 'log_단지내주차면수', 'log_총세대수', 'qcut_총세대수', '임대건물구분_lbl', '공급유형_lbl', '지역_lbl', '단지코드_lbl', '단지코드_Type']	n_estimators = 2000	0.030599322622307225	103.57818
XGBoost	['총세대수', '전용면적', '공가수', '자격유형', '10분내지하철수', '10분내버스정류장수', '단지내주차면수', '10대미만(여자)', '10대미만(남자)', '30대(여자)', '30대(남자)','지역_lbl', '공급유형_lbl', '단지코드_lbl','단지코드_Type']	n_estimators=50000, learning_rate=0.05	0.01477507786621523	94.0385838723

모델	피처	하이퍼파라미터	평가	최종 SCORE
CATBOOST	['총세대수', '전용면적', '공가수', '자격유형', '10분내지하철수', '10분내버스정류장수', '단지내주차면수', '10대미만(여자)', '10대미만(남자)', '30대(여자)', '30대(남자)','지역_lbl', '공급유형_lbl', '단지코드_lbl','단지코드_Type']	n_estimators=200000, learning_rate=0.05	0.04692451270019151	**92.58984 (Public) 110.75797 (Private)**
제목 없음	🤔 리더보드상 좋은 컬럼을 선택하였기 때문에 Private 점수에 어떤 영향을 끼쳤을지는 모르겠지만 과적합을 더 줄일 수 있지 않았을까			
제목 없음				
제목 없음				

▼ 최종 정리

1. 시행착오 및 아쉬운 점

- 표준화 과정에서 대회가 끝나고서야 기준을 잘못 설정한 것을 깨달았다. 다음 부터는 Scaler를 할 때에 기준을 동일하게 두고 표준화하여 모델 스코어가 얼마나 좋아지는지 확인해보아야겠다.
- 데이터 전처리를 하면 할수록 스코어가 좋아지지 않았다. 🤔

  —> 이유를 정확하게 분석할 시간이 부족했다. 리더보드상의 성적을 올리기 위해 나중에는 모델검증, 피쳐 선택에만 급급했지만 결론적으로 Private 성적이 오른 걸 보아 데이터 전처리를 하는 부분으로 했으면 Private 성적이 좋았을지도 모르겠다는 생각이 들었다.
- 전처리 과정에 집중하였지만, 중간 점검 팀장회의에서 데이터는 베이스라인의 기초로 가고 모델 성능과 파라미터 개선으로 스코어를 낮춘 다른 팀장님들의 팁을 들었다. 그 후, 베이스라인을 갈아엎고 Feature 선택과 모델 검증에 시간을 쏟게 되었다.
- LASSO → LGBM(점수침체기ㅠ) → XGBOOST → CATBOOST 순으로 검증을 시도하고 score가 최종적으로 103 → 94 → 92 로 개선되었다. 대회가 끝나고 Private 점수가 높아진 것을 보며 CATBOOST는 과적합의 단점을 가진 것으로 보인다.
- 계속 LGBM으로 모델만들어서 학습시키다가 LGBM은 적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉽다는 단점이 있어서 XGBoost로 모델 변경했다.
- Private, Public 점수의 차이가 많이 나는 이유가 과적합(Overfitting)이라고 하는데 이유를 잘 모르겠다. 같은 데이터내에서 스코어를 낸 Public, Private인데, 나머지 66%는 과적합이고 33%는 아닌 건가?

데이터 전처리

열 1	중간점검	최종	이름	lgbm score 변화 (열 1)에 관계됨	태그	역할 분담 (속성)에 관계됨
	train_df_errno.csv	train_df_jm.csv +(train 데이터에도 오류처리)	+ 2. 'age_gender_info' 적용하기	🤩 양효진	☑	
	'등록차량수' 정규화	'등록차량수' 정규화	+ 1. log변수 추가 및 오류처리 파일 변경	🤩 양효진	☑	
		+train 데이터에만 있는 '서울특별시' 데이터 삭제	제목 없음		☐	

열 1	중간점검	최종	이름	lgbm score 변화 (열 1)에 관계됨	태그	역할분담 (속성)에 관계됨
결측치처리	'10분내버스정류장수'의 평균으로 결측치 처리	'10분내버스정류장수'의 평균으로 결측치 처리	* '10분내지하철수' 결측치 처리	😎 이준명	☑	
		+임대보증금&임대료 - 0처리	제목 없음		☐	
		+지하철수 DecisionTree 예측	제목 없음		☐	
라벨인코딩	'자격유형' mapping	공급유형/자격유형 묶기	제목 없음		☑	
	'임대건물구분', '지역', '공급유형' mapping		*임대건물구분, 지역, 공급유형,단지코드 _lbl 제거	🧐 양소연	☑	
새로운컬럼추가		+age_gender 병합시키기	제목 없음		☐	
		+주차면수대비총세대수비율 & 거주율 추가	제목 없음		☐	
표준화,이상치제거		+표준화 및 이상치 제거	제목 없음		☐	
	'총세대수' qcut		* 총세대수 이상치제거	😃 김혜린	☑	
모델링	lgbm	lgbm, xgboost, catboost, lasso	learning_rate = 0.5, n_estimators=2000		☑	
피쳐선택	['총세대수', '전용면적', '전용면적별세대수', '공가수', '자격유형', '10분내버스정류장수', '단지내주차면수', '임대건물구분_lbl', '지역_lbl', '공급유형_lbl','단지코드_lbl']	['총세대수', '전용면적', '공가수', '자격유형', '10분내지하철수', '10분내버스정류장수', '단지내주차면수', '10대미만(여자)', '10대미만(남자)', '30대(여자)', '30대(남자)','지역_lbl', '공급유형_lbl', '단지코드_lbl','단지코드_Type']	제목 없음		☐	
	score = 103.57818	score = **92.58984**	제목 없음		☐	

역할분담

이름	중간점검	최종	속성

Aa 이름	중간점검	최종	속성
😃 김혜린	* 총세대수 이상치 제거	+ 전체 Notion 정리 및 발표 + CATBoost, LGBM 모델 스코어 기록 + 파라미터, 변수, 모델 변경 및 기록	210722_baseline_Hyerin(2).html
😎 이준명	* '10분내지하철수' 결측치 처리	+ 표준화, 이상치 제거 + 컬럼 생성 + 라벨 인코딩 + 파라미터, 변수, 모델 변경 및 기록	0722_dacon_parking_demand_subway_score.html 0722_dacon_parking_demand_test_subwa
🤩 양효진	+ 1. log변수 추가 및 오류처리 파일 변경, + 2. 'age_gender_info' 적용하기	+ age_gender 상관관계 그래프 그리기, merge + 피쳐 상관관계 그래프 분석 + Lasso, XGBoost 모델 스코어 기록 + 파라미터, 변수, 모델 변경 및 기록	0722_9th_10th_11th_준명님 파일로 바꿈.html
🧐 양소연	*임대건물구분, 지역, 공급유형,단지코드 _ lbl 제거	+ 변수들 간의 상관관계 분석 + 정규화된 변수들 상관관계 분석 + 추가된 변수들 상관관계 분석 + 파라미터, 변수, 모델 변경 및 기록	210722_baseline_edit.html

Github - 코드

KimHyerin25/LikeLion_2nd_TeamProject_-AI-

2팀) 김혜린, 이준명, 양효진, 양소연, 박상엽. Contribute to KimHyerin25/LikeLion_2nd_TeamProject_-AI-development by creating an account on GitHub.

https://github.com/KimHyerin25/LikeLion_2nd_TeamProject_-AI-

▼ ~~중간점검발표_데이터전처리~~

변수 선택 및 데이터 전처리

> 스코어를 낮추기 위해서는 상관관계가 높은 변수를 선택하여 예측에 도움이 되도록 하고 최대한 train data와 test data가 비슷해야 한다고 생각할 수 있었다.

	단지코드	총세대수	임대건물구분	지역	공급유형	전용면적	전용면적별세대수	공가수	자격유형	임대보증금	임대료	10분내지하철수	10분내버스정류장수	단지내주차면수	임대건물구분_lbl	지역_lbl
0	C2515	545	아파트	경상남도	국민임대	33.48	276	17.0	1	9216000	82940	0.0	3.0	624.0	1	1
1	C2515	545	아파트	경상남도	국민임대	39.60	60	17.0	1	12672000	107130	0.0	3.0	624.0	1	1
2	C2515	545	아파트	경상남도	국민임대	39.60	20	17.0	1	12672000	107130	0.0	3.0	624.0	1	1
3	C2515	545	아파트	경상남도	국민임대	46.90	38	17.0	1	18433000	149760	0.0	3.0	624.0	1	1
4	C2515	545	아파트	경상남도	국민임대	46.90	19	17.0	1	18433000	149760	0.0	3.0	624.0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3899	C1267	675	아파트	경상남도	행복주택	36.77	126	38.0	12	-	-	0.0	1.0	467.0	1	1
3900	C2189	382	아파트	전라북도	국민임대	29.19	96	45.0	8	6872000	106400	0.0	2.0	300.0	1	4

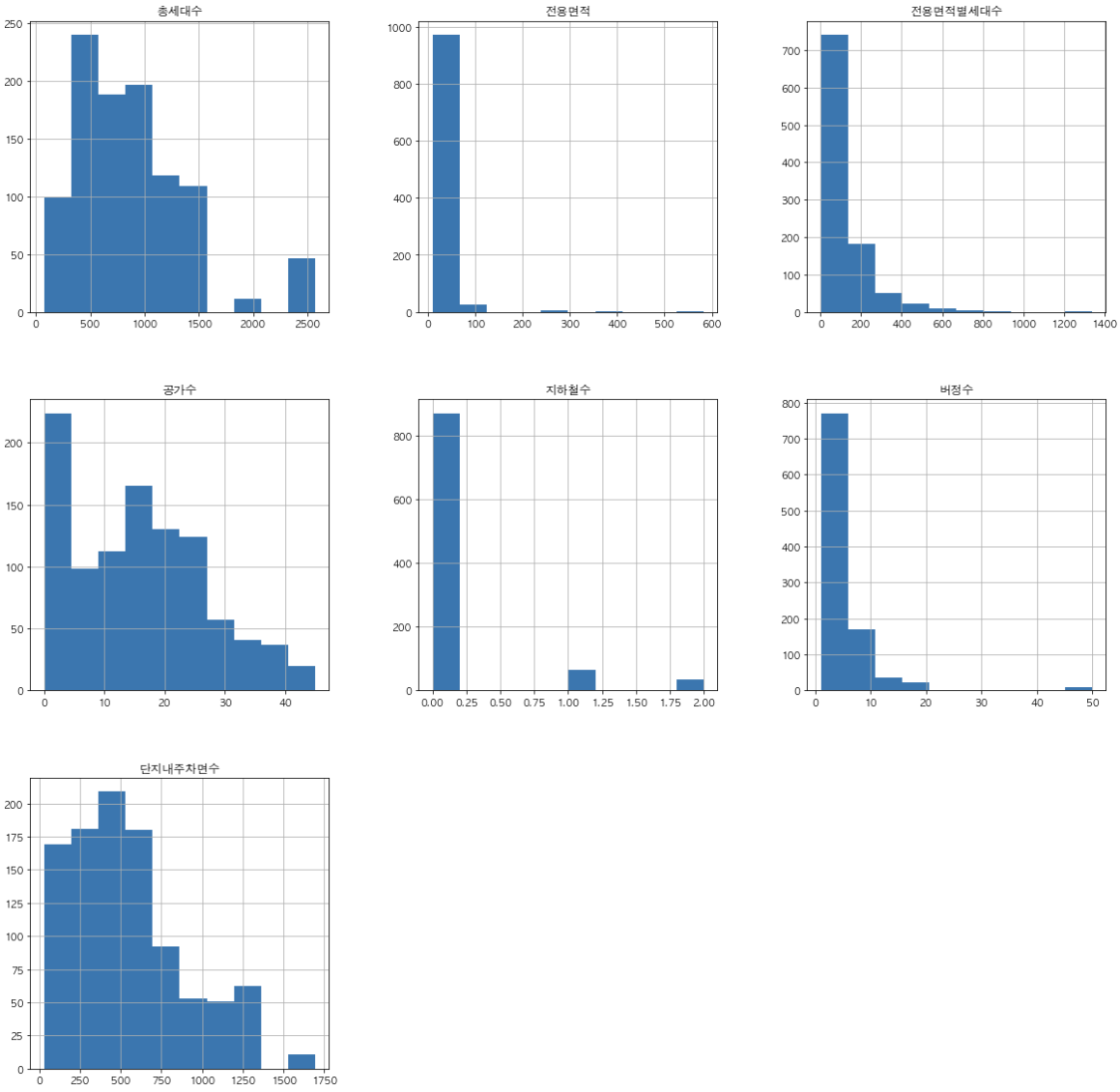
- train data에만 있고 test data에 없는 '지역'의 '**서울특별시**'를 뺐더니 126—> 121점으로 스코어가 내려갔다. ❗ 나머지 train data에만 존재하는 항목들(어떤거)을 다 뺐더니 더 내려갔음 (얼마로?)

```
train_지역list :  ['경상남도' '대전광역시' '경기도' '전라북도' '강원도' '광주광역시' '충청남도' '부산광역시' '제주특별자치도'
 '울산광역시' '충청북도' '전라남도' '경상북도' '대구광역시' '서울특별시' '세종특별자치시']
test_지역list :  ['경기도' '부산광역시' '전라북도' '경상남도' '충청남도' '대전광역시' '제주특별자치도' '강원도' '울산광역시' '경상북도'
 '충청북도' '광주광역시' '전라남도' '대구광역시' '세종특별자치시']
```

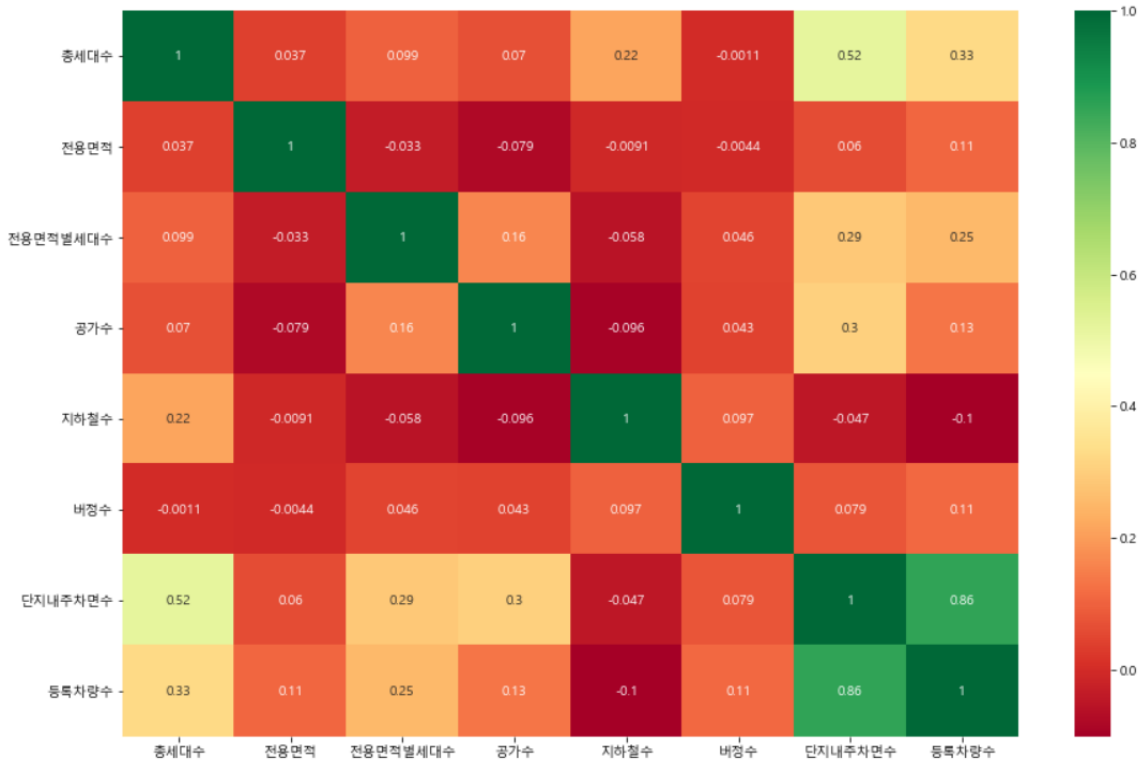
- 모델을 릿지, 라쏘의 스코어 차이 : alpha값을 올렸더니 스코어가 올라갔다. ❗ alpha가 규제를 세게 했더니 오히려 올랐다. 이유는 데이터가 양이 많지 않아서 @이준명

- 여러 feature의 히스토그램에서 그래프가 **왼쪽**으로 치우쳐져있는 것을 보고, **정규화** 시키거나 **이상치를 제거**하면 score가 내려가지 않을까? lgbm score가 0.03 —> 0.2로 올라갔다. ❗ @hyojin yang @Hyerin Kim

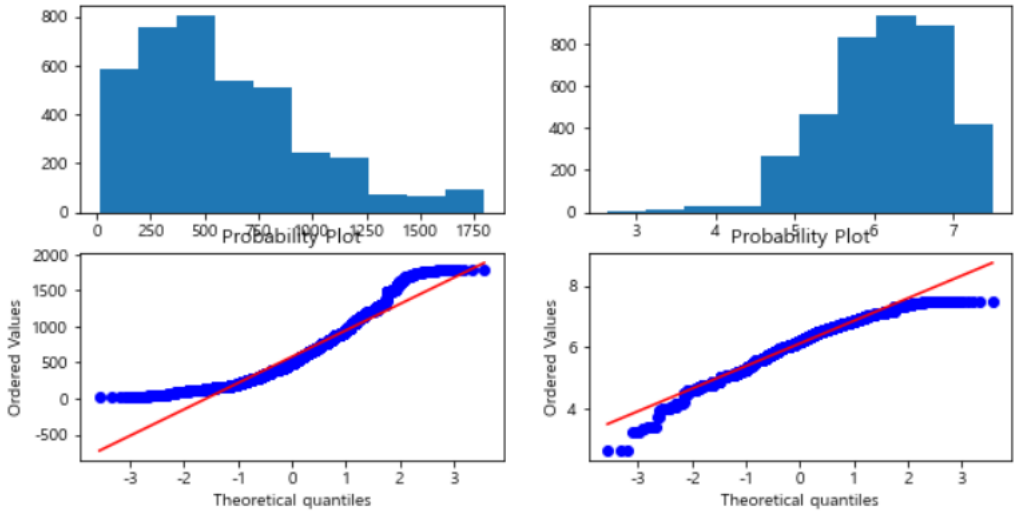
⇒ 앞으로 해야할 것 - 상관관계가 높은 칼럼들을 뽑아 정규화 시켜보고 검증해보았지만 score 가 내려가진 않았다. **Z-점수 정규화**를 할 예정 @hyojin yang



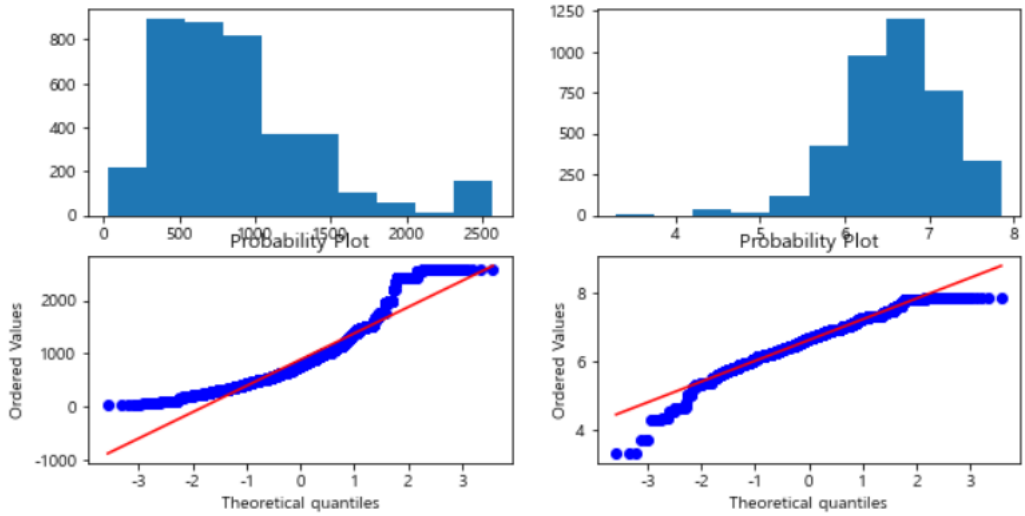
- 상관관계 확인
  > 등록차량수와 상관관계가 높은 총세대수 , 전용면적별세대수, 단지내주차면수 3가지 feature에 관하여 log 처리



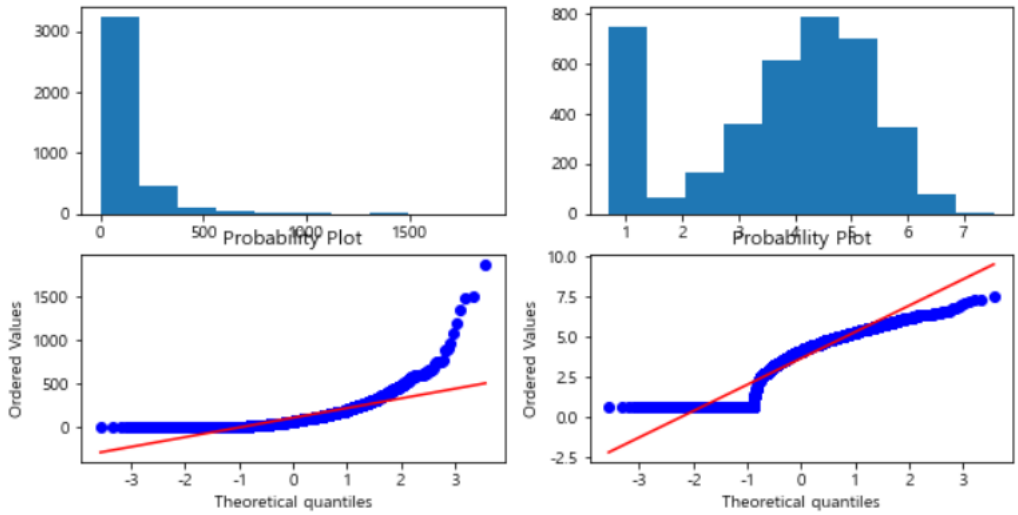
- 단지내주차면수(오른쪽이 log 씌운값) > 넣기



- 총 세대수(오른쪽이 log 씌운값) >넣기



- 전용면적별세대수(오른쪽이 log 씌운값) > 넣지 않기



- 10분내 **지하철수** 결측치데이터를 보니 **충청남도, 대전, 경상남도** 지역에서 나타났다. 해당지역은 지하철 역이 있는 도시도 있는데, 결측치처리를 어떻게 하면 좋을까? —>모델링 돌리면서 결측치를 예측해보자. @이준명

```
전체 단지 수:  414
지하철 결측치 단지 수:  20
지하철 결측치 단지:  ['C1312' 'C1874' 'C1424' 'C2100' 'C2520' 'C1616' 'C1704' 'C2258' 'C1068'
 'C1983' 'C2216' 'C2644' 'C1005' 'C1004' 'C1875' 'C2156' 'C1175' 'C2583'
 'N2431' 'C1350']
지하철 결측치 단지 지역:  ['충청남도' '대전광역시' '경상남도']
```

▼ 참조 : 변수 선택 기법(Feature Selection Method)

> 모델을 돌릴 때 쓸모 없는 변수들을 제거함으로써
> 모델의 속도 개선, 오버피팅 방지 등의 효과를 얻기 위해 사용하는 방법.

https://dyddl1993.tistory.com/18

1. Wrapper method : **모델링** 돌리면서 변수 채택

— RFE(recursive feature elimination) : 모든 변수를 우선 다 포함시킨 후 반복해서 학습을 진행하면서 중요도가 낮은 변수를 하나씩 제거하는 방식.

```
**from**

sklearn.feature_selection

**import**

RFE
RFE = RFE(model, n_features_to_select=20)
X_train_RFE = RFE.fit_transform(X_train,y_train)
X_test_RFE = RFE.transform(X_test)
```

2. Filter Method : 전처리단에서 **통계기법** 사용하여 변수 채택

— 분산(VarianceThreshold), 단일 변수 선택, 상호정보량(mutual information), Information value(IV)

3. Embedded method : **라쏘, 릿지, 엘라스틱넷** 등 내장함수 사용하여 변수 채택

— 모델 안에서 l1, l2 정규화를 통해 변수를 축소해서 사용하는 방법이다.

▼ ~~중간점검발표_결과~~

중간점검 개선방향

- **원핫인코딩, 등록차량수 정규화**를 시도하여 lgbm score는 0.01로 훨씬 내려갔지만 최종 제출 스코어는 올라갔다(116점) : test, train 데이터의 차이가 너무 심해 과적합
- '**교차검증**'을 하면 score가 낮아질까? ❓
- **1번 오류**가 전용면적별세대수와 총세대수가 오류가 난건데, 혹시 전용면적별세대수가 큰 영향을 미치지 않는다면 제거해보고 스코어가 나아지지 않을까?

  + 전용면적별세대수를 총세대수와 일일이 맞춰주면 스코어가 나아지지 않을까? (강사님은 분류로 했음)
- qcut_총세대수 (qcut_5해서0,1,2,3,4 분류) —> 10, X 하면 스코어가 다 올라갔다.
- 전용면적을 5의 배수로 맞춰준다 (소수점으로 되어있는 부분을 5의 배수 정수로 분류해주면 데이터가 더 정형화 되어 스코어가 올라가지 않을까?)
- 2번 오류 처리 —> 강사님은 test data에 있는 오류만 제거, 우리 팀은 train data와 test data 모두에 있는 오류를 제거했더니 스코어가 더 좋아졌다. 이 부분을 좀더 개선해볼 필요를 느꼈다.

최적의 모델 찾기

- 현재까지 최고 스코어 **103**점

(아름님) lgbm 사용, learning_rate = 0.05, n_estimators=2000~5000

2,3번 오류 처리 — test, train 데이터 둘다 오류를 제거해줌

이외의 파라미터를 사용해서 스코어를 낮추기는 큰 변화가 보이진 않았다.

(결국 **변수**를 정리하는 방법 밖에 없는 것 같다.)