11/15 과제

https://dacon.io/competitions/official/235745/codeshare/3015

목적: 유형별 임대주택 설계 시 단지 내 적정 주차 수요를 예측

각종 설치 & 한글 폰트 설정 & 패키지 불러오기 & 경로 설정

```
#패키지 불러오기 및 각종 설치 & 폰트 설정 한방에
import matplotlib
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

!apt-get update -qq
!apt-get install fonts-nanum* -qq

font_path = '/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf'
font_name = fm.FontProperties(fname=font_path, size=10).get_name()
print(font_name)
plt.rc('font', family=font_name)

fm._rebuild()
matplotlib.rcParams['axes.unicode_minus'] = False

!pip install catboost
!pip install pycaret
from tqdm.notebook import tqdm
from pycaret.regression import *

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

```
from google.colab import drive
drive.mount('/gdrive')
```

```
import os
workspace_path = '/gdrive/My Drive/dacon/주차수요 예측/'
data_path = os.path.join(workspace_path, 'parking')
```

[데이터 탐색 및 전처리]

데이터를 CSV 파일에서 읽어오고 데이터프레임으로 저장함

```
age_gender = pd.read_csv(os.path.join(data_path, 'age_gender_info.csv'))
train = pd.read_csv(os.path.join(data_path, 'train.csv'))
test = pd.read_csv(os.path.join(data_path, 'test.csv'))
```

'임대보증금' 열에서 '-' 값을 가진 행을 찾아 NaN으로 변경 ('-' 는 수치 데이터로 해석할 수 없으므로 이를 결측값으로 변환해 데이터 분석에 적합하게 만드는 과정임)

```
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)
```

'단지코드' 열에서 예측값과 실제값 사이의 오차가 가장 단지 4개의 행을 제거 (토크 게시판을 통해, 대회 참여자들이 분석 중 발견한 이상치를 게시판에서 공유했던 것으로 보임)

```
##1번 데이터 오류 (가장 큰 차이를 보이는 단지 4개에 대해 삭제 (토크 게시판에 크기순으로 나열되어있음))
train = train[train.단지코드 != 'C1804']
train = train[train.단지코드 != 'C2405']
train = train[train.단지코드 != 'C1740']
train = train[train.단지코드 != 'C1206']
# train = train[train.단지코드 != 'C2470']
# train = train[train.단지코드 != 'C1024']
# train = train[train.단지코드 != 'C1344']
# train = train[train.단지코드 != 'C2620']
# train = train[train.단지코드 != 'C2497']
# train = train[train.단지코드 != 'C1490']

# train = train[train.단지코드 != 'C1925']
# train = train[train.단지코드 != 'C1312']
```

```
# train = train[train.단지코드 != 'C2013']
# train = train[train.단지코드 != 'C1424']
# train = train[train.단지코드 != 'C2520']
# train = train[train.단지코드 != 'C2319']
# train = train[train.단지코드 != 'C1850']
# train = train[train.단지코드 != 'C1068']
# train = train[train.단지코드 != 'C2644']
# train = train[train.단지코드 != 'C2156']

# train = train[train.단지코드 != 'C2453']
# train = train[train.단지코드 != 'C1910']
# train = train[train.단지코드 != 'C2139']
# train = train[train.단지코드 != 'C2508']
# train = train[train.단지코드 != 'C1695']
# train = train[train.단지코드 != 'C2556']
# train = train[train.단지코드 != 'C2362']
# train = train[train.단지코드 != 'C2568']
# train = train[train.단지코드 != 'C2245']
# train = train[train.단지코드 != 'C2549']

# train = train[train.단지코드 != 'C1584']
# train = train[train.단지코드 != 'C2298']
# train = train[train.단지코드 != 'C2225']
# train = train[train.단지코드 != 'C1218']
# train = train[train.단지코드 != 'C1970']
# train = train[train.단지코드 != 'C1732']
# train = train[train.단지코드 != 'C2433']
# train = train[train.단지코드 != 'C1894']
# train = train[train.단지코드 != 'C1156']
# train = train[train.단지코드 != 'C2142']

# train = train[train.단지코드 != 'C2186']
# train = train[train.단지코드 != 'C2411']
# train = train[train.단지코드 != 'C1812']
# train = train[train.단지코드 != 'C1030']
# train = train[train.단지코드 != 'C1749']
# train = train[train.단지코드 != 'C1349']
# train = train[train.단지코드 != 'C2043']
# train = train[train.단지코드 != 'C1229']
# train = train[train.단지코드 != 'C2363']
# train = train[train.단지코드 != 'C1414']

# train = train[train.단지코드 != 'C2174']
# train = train[train.단지코드 != 'C2404']
# train = train[train.단지코드 != 'C1683']
# train = train[train.단지코드 != 'C1038']
# train = train[train.단지코드 != 'C2456']
# train = train[train.단지코드 != 'C1266']
```

```
# train = train[train.단지코드 != 'C1267']
# train = train[train.단지코드 != 'C2189']

##2번 데이터오류
train = train[train.단지코드 != 'C2085']
train = train[train.단지코드 != 'C1397']
train = train[train.단지코드 != 'C2431']
train = train[train.단지코드 != 'C1649']
train = train[train.단지코드 != 'C1036']

##3번 데이터 오류
train = train[train.단지코드 != 'C1095']
train = train[train.단지코드 != 'C2051']
train = train[train.단지코드 != 'C1218']
train = train[train.단지코드 != 'C1894']
train = train[train.단지코드 != 'C2483']
train = train[train.단지코드 != 'C1502']
train = train[train.단지코드 != 'C1988']
```

버스 정류장 및 지하철 수 처리 → test에 50이라는 너무 이상한 값이 존재 >> train의 mean으로 보정

```
train['도보 10분거리 내 버스정류장 수'].unique()
```

```
array([ 3.,  1.,  2.,  6., 10.,  5.,  4.,  7., 12., 14.,  8.,  0., 20.,
       11., 16., 15., 19.])
```

```
test['도보 10분거리 내 버스정류장 수'].unique()
```

```
array([ 2.,  3., 16.,  6.,  1.,  4.,  5.,  8., 10., 13.,  7., 11., 50.,
       12., 14., 18., 15., 19., 17.])
```

```
train['도보 10분거리 내 지하철역 수(환승노선 수 반영)'] = train['도보 10분거리 내 지하철역 수(환승노선 수 반영)'].fillna(0)
test['도보 10분거리 내 지하철역 수(환승노선 수 반영)'] = test['도보 10분거리 내 지하철역 수(환승노선 수 반영)'].fillna(0)
train['도보 10분거리 내 버스정류장 수'] = train['도보 10분거리 내 버스정류장 수'].fillna(0)
```

```
test[test['도보 10분거리 내 버스정류장 수']==50]   ##상식적으로 있을 수 없는 값이라고 판단
```

```
# train[train['지역'] == '경기도']['도보 10분거리 내 버스정류장 수'].max()
```

```
test.loc[test['도보 10분거리 내 버스정류장 수'] == 50, '도보 10분거리 내 버스정류장 수'] = train['도보 10분거리 내 버스정류장 수'].mean()
##train의 평균 값으로 처리


##train[train['지역'] == '경기도']['도보 10분거리 내 버스정류장 수'].max()
## 50정도의 큰 값이라면 해당 지역내 가장 큰 값으로 대신해도 된다고 판단, test의 통계치는 참고할 수 없으니 train의 max값으로 처리
```

```
# train = train.drop(columns=['임대료', '임대보증금']) 살리는 게 더 좋은 점수를 가져옴
# test = test.drop(columns=['임대료', '임대보증금'])
# train
```

임대료 및 임대보증금 처리

```
#임대료 처리
train.loc[train['공급유형'] == '공공분양', '임대료'] = train['임대료'].fillna(0)
train.loc[train['공급유형'] == '장기전세', '임대료'] = train['임대료'].fillna(0)
train.loc[train['공급유형'] == '국민임대', '임대료'] = train['임대료'].fillna(0)
train.loc[train['공급유형'] == '행복주택', '임대료'] = train['임대료'].fillna(0)

#임대보증금 처리
train.loc[train['공급유형'] == '공공분양', '임대보증금'] = train['임대보증금'].fillna(0)
train.loc[train['공급유형'] == '장기전세', '임대보증금'] = train['임대보증금'].fillna(0)
train.loc[train['공급유형'] == '국민임대', '임대보증금'] = train['임대보증금'].fillna(0)
train.loc[train['공급유형'] == '행복주택', '임대보증금'] = train['임대보증금'].fillna(0)
```

다른 공급 유형이면 몰라도 임대 상가에 한해서는 임대료와 임대보증금이 0원이라는 것은 말이 안 된다고 판단해 임대상가는 해당 지역의 train의 임대료와 임대보증금 평균값으로 처리

```
#임대 상가의 임대료 처리
train.loc[train['지역'] == '부산광역시', '임대료'] = train['임대료'].fillna(train[train['지역'] == '부산광역시']['임대료'].mean())
train.loc[train['지역'] == '대전광역시', '임대료'] = train['임대료'].fillna(train[train['지역'] == '대전광역시']['임대료'].mean())
train.loc[train['지역'] == '경상남도', '임대료'] = train['임대료'].fillna(train[train['지역'] == '경상남도']['임대료'].mean())
```

```
train.loc[train['지역'] == '충청남도', '임대료'] = train['임대료'].fillna(train
[train['지역'] == '충청남도']['임대료'].mean())
train.loc[train['지역'] == '강원도', '임대료'] = train['임대료'].fillna(train[tr
ain['지역'] == '강원도']['임대료'].mean())
train.loc[train['지역'] == '제주특별자치도', '임대료'] = train['임대료'].fillna(t
rain[train['지역'] == '제주특별자치도']['임대료'].mean())

#임대 상가의 임대보증금 처리
train.loc[train['지역'] == '부산광역시', '임대보증금'] = train['임대보증금'].filln
a(train[train['지역'] == '부산광역시']['임대보증금'].mean())
train.loc[train['지역'] == '대전광역시', '임대보증금'] = train['임대보증금'].filln
a(train[train['지역'] == '대전광역시']['임대보증금'].mean())
train.loc[train['지역'] == '경상남도', '임대보증금'] = train['임대보증금'].fillna
(train[train['지역'] == '경상남도']['임대보증금'].mean())
train.loc[train['지역'] == '충청남도', '임대보증금'] = train['임대보증금'].fillna
(train[train['지역'] == '충청남도']['임대보증금'].mean())
train.loc[train['지역'] == '강원도', '임대보증금'] = train['임대보증금'].fillna(t
rain[train['지역'] == '강원도']['임대보증금'].mean())
train.loc[train['지역'] == '제주특별자치도', '임대보증금'] = train['임대보증금'].fi
llna(train[train['지역'] == '제주특별자치도']['임대보증금'].mean())
```

```
# train.groupby(['공급유형'], as_index=False)['임대료'].mean()
```

```
#임대료 처리
test.loc[test['공급유형'] == '영구임대', '임대료'] = test['임대료'].fillna(0)
test.loc[test['공급유형'] == '행복주택', '임대료'] = test['임대료'].fillna(0)

#임대보증금 처리
test.loc[test['공급유형'] == '영구임대', '임대보증금'] = test['임대보증금'].fillna
(0)
test.loc[test['공급유형'] == '행복주택', '임대보증금'] = test['임대보증금'].fillna
(0)
```

```
#임대 상가의 임대료 처리  (test 특성 기준 test null 값 처리는 data leakage 여서 tra
in의 지역 기준으로 test 처리)
test.loc[test['지역'] == '부산광역시', '임대료'] = test['임대료'].fillna(train[t
rain['지역'] == '부산광역시']['임대료'].mean())
test.loc[test['지역'] == '대전광역시', '임대료'] = test['임대료'].fillna(train[t
rain['지역'] == '대전광역시']['임대료'].mean())
test.loc[test['지역'] == '울산광역시', '임대료'] = test['임대료'].fillna(train[t
rain['지역'] == '울산광역시']['임대료'].mean())
test.loc[test['지역'] == '충청남도', '임대료'] = test['임대료'].fillna(train[tra
in['지역'] == '충청남도']['임대료'].mean())
test.loc[test['지역'] == '강원도', '임대료'] = test['임대료'].fillna(train[train
['지역'] == '강원도']['임대료'].mean())
```

```
#임대 상가의 임대보증금 처리
test.loc[test['지역'] == '부산광역시', '임대보증금'] = test['임대보증금'].fillna(t
rain[train['지역'] == '부산광역시']['임대보증금'].mean())
test.loc[test['지역'] == '대전광역시', '임대보증금'] = test['임대보증금'].fillna(t
rain[train['지역'] == '대전광역시']['임대보증금'].mean())
test.loc[test['지역'] == '울산광역시', '임대보증금'] = test['임대보증금'].fillna(t
rain[train['지역'] == '울산광역시']['임대보증금'].mean())
test.loc[test['지역'] == '충청남도', '임대보증금'] = test['임대보증금'].fillna(tra
in[train['지역'] == '충청남도']['임대보증금'].mean())
test.loc[test['지역'] == '강원도', '임대보증금'] = test['임대보증금'].fillna(trai
n[train['지역'] == '강원도']['임대보증금'].mean())
```

자격유형 처리 (‘자격유형’이라는 열 있음)

A로 처리한 것

```
test[test.자격유형.isnull()]
```

```
test[test.단지코드=='C2411'] #같은 단지의 자격유형이 모두 A이므로 누락된 것으로 생각하
고 A로 채워도 될 듯
```

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

C로 처리한 것

```
test[test.단지코드=='C2253'] #같은 단지의 임대상가가 아닌 영구임대 유형의 경우 모두 C이
므로 C로 채워도 될 듯
```

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

공급 + 자격?

공급유형과 자격유형이 어느정도의 관련성이 있을 것으로 예상해 이를 합쳐서 새로운 특성으로 바꿈. 실제로 고유값이 그렇게 많이 만들어지지는 않았음을 확인 가능. (EX. 국민임대_A)

```
train['공급_자격'] = train.apply(lambda r : r['공급유형'] + '_' +r['자격유형'],
axis=1)
test['공급_자격'] = test.apply(lambda r : r['공급유형'] + '_' +r['자격유형'], ax
```

```
is=1)
train
```

```
print("train - test (공급_자격 차집합) : ", set(train.공급_자격).difference(set(test.공급_자격))) ##train에만 있는 값들 찾아보기
print("test - train (공급_자격 차집합) : ", set(test.공급_자격).difference(set(train.공급_자격)))
```

```
train = train[train.공급_자격 != '공공분양_D']
train = train[train.공급_자격 != '공공임대(5년)_A']
train = train[train.공급_자격 != '국민임대_B']
train = train[train.공급_자격 != '영구임대_A']
train = train[train.공급_자격 != '영구임대_E']
train = train[train.공급_자격 != '영구임대_F']
train = train[train.공급_자격 != '장기전세_A']
train = train[train.공급_자격 != '행복주택_O']

test = test[test.공급_자격 != '영구임대_D']
```

```
train['전용면적'] = train['전용면적']//2*2  ##현재는 2*2일 때 가장 성능이 좋았었다.
test['전용면적'] = test['전용면적']//2*2
```

```
print("train - test (전용면적 차집합) : ", set(train.전용면적).difference(set(test.전용면적))) ##train에만 있는 값들 찾아보기
print("test - train (전용면적 차집합) : ", set(test.전용면적).difference(set(train.전용면적)))
```

```
train = train[train.전용면적 != 52]##
train = train[train.전용면적 != 56]##
train = train[train.전용면적 != 62]##
train = train[train.전용면적 != 64]
train = train[train.전용면적 != 66] ##
train = train[train.전용면적 != 72] ##
train = train[train.전용면적 != 78]
train = train[train.전용면적 != 108] ##
train = train[train.전용면적 != 126]##
train = train[train.전용면적 != 136]##
train = train[train.전용면적 != 316]##
train = train[train.전용면적 != 406]##

test = test[test.전용면적 != 8]
test = test[test.전용면적 != 252]
```

```
print("train - test (지역 차집합) : ", set(train.지역).difference(set(test.지역))) ##train에만 있는 값들 찾아보기
print("test - train (지역 차집합) : ", set(test.지역).difference(set(train.지역)))
```

```
train = train[train.지역 != '서울특별시'] ##컬럼의 특정 값이면 제거하는 방법
```

---

age_gender에서 미성년자(20대 미만) 비율만 가져와서 join

→ 이 값이 높다면 등록차량 수가 낮을 것이라는 판단

```
minors = ['10대미만(여자)', '10대미만(남자)', '10대(여자)', '10대(남자)']
age_gender['미성년자'] = age_gender[minors].sum(axis=1)
```

```
train = train.merge(age_gender, left_on= ["지역"], right_on= ["지역"], how='left')
test = test.merge(age_gender, left_on= ["지역"], right_on= ["지역"], how='left')
```

[데이터 정리 및 병합]

```
train.shape, train.drop_duplicates().shape
```

```
test.shape, test.drop_duplicates().shape
```

```
train = train.drop_duplicates()
test = test.drop_duplicates()
```

```
unique_cols = ['총세대수', '지역', '공가수', '미성년자',
               '도보 10분거리 내 지하철역 수(환승노선 수 반영)',
               '도보 10분거리 내 버스정류장 수',
               '단지내주차면수', '등록차량수']
##,'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)'
train_agg = train.set_index('단지코드')[unique_cols].drop_duplicates()
test_agg = test.set_index('단지코드')[[col for col in unique_cols if col!='등록차량수']].drop_duplicates()
```

```
train_agg.head()
```

임대 건물 구분, 전용면적 별 세대수, 임대보증금

→ 등과 같이 처리가 애매한 값은 각 단지 별 mean값이나 고유값 개수로 처리

```
tr = train.groupby(['단지코드']).mean() ##같은 단지코드 안에서의 평균 값, max 값 등은 data leakage가 아닌 것으로 보임
ts = test.groupby(['단지코드']).mean()

train_agg['전용면적별세대수 평균'] = tr['전용면적별세대수']
test_agg['전용면적별세대수 평균'] = ts['전용면적별세대수']

train_agg['임대보증금 평균'] = tr['임대보증금']
test_agg['임대보증금 평균'] = ts['임대보증금']

train_agg['임대료 평균'] = tr['임대료']
test_agg['임대료 평균'] = ts['임대료']
```

```
tr = train.groupby(['단지코드']).nunique(dropna=False)
ts = test.groupby(['단지코드']).nunique(dropna=False)

train_agg['임대건물구분'] = tr['임대건물구분']
test_agg['임대건물구분'] = ts['임대건물구분']
```

---

categorical 값들 펼치기 -'파베르'님 코드 공유 참고 (데이터를 테이블 형태로 깔끔하게 정리)

```
def reshape_cat_features(data, cast_col, value_col):
    res = data.drop_duplicates(['단지코드', cast_col]).assign(counter=1).pivot(index='단지코드', columns=cast_col, values=value_col).fillna(0)
    res.columns.name = None
    res = res.rename(columns={col:col for col in res.columns})
    return res
```

```
def reshape_cat_features_plus_underbar(data, cast_col, value_col):
    res = data.drop_duplicates(['단지코드', cast_col]).assign(counter=1).pivot(index='단지코드', columns=cast_col, values=value_col).fillna(0)
    res.columns.name = None
    res = res.rename(columns={col:cast_col+'_'+ str(col) for col in res.columns})
    return res
```

```
def transportation_level(x): ##대중교통 값 조정 (원래 있던 값을 drop 하지 않았을 때 더 성능이 잘 나옴)
  result = 0
  if x <= 0 :
    result = 5
  elif 1 <= x <= 2:
    result = 4
  elif 3 <= x <= 4 :
    result = 3
  elif 5 <= x <= 6 :
    result = 2
  elif 7 <= x:
    result = 1
  return result
```

예시:

```
단지코드    범주    값
A1        cat1    1
A1        cat2    2
A2        cat1    3
A2        cat3    4
A3        cat2    5
A3        cat3    6
```

```
      cat1  cat2  cat3
단지코드
A1        1     2     0
A2        3     0     4
A3        0     5     6
```

다 합친 데이터 생성 및 추가 처리 (log)

```
train_data = pd.concat([train_agg.drop(columns=['지역']), reshape_cat_features(data=train, cast_col='공급_자격', value_col='counter'),
                        reshape_cat_features(data=train, cast_col='지역', value_col='counter'),
                       reshape_cat_features_plus_underbar(data=train, cast_col='전용면적', value_col='counter')
                       ], axis=1)
test_data = pd.concat([test_agg.drop(columns=['지역']), reshape_cat_features(data=test, cast_col='공급_자격', value_col='counter'),
                       reshape_cat_features(data=test, cast_col='지역', value_col='counter'),
                       reshape_cat_features_plus_underbar(data=test, cast_c
```

```
ol='전용면적', value_col='counter')
                      ], axis=1)

train_data['실거주율'] = (train_agg['총세대수'] - train_agg['공가수'])/train_agg['총세대수']
test_data['실거주율'] = (test_agg['총세대수'] - test_agg['공가수'])/test_agg['총세대수']

transport = ['도보 10분거리 내 지하철역 수(환승노선 수 반영)', '도보 10분거리 내 버스정류장 수']
train_data['대중교통현황'] = train_agg[transport].sum(axis=1)
test_data['대중교통현황'] = test_agg[transport].sum(axis=1)

# train_data['대중교통현황'] = train_data['대중교통현황'].apply(transportation_level)
# test_data['대중교통현황'] = test_data['대중교통현황'].apply(transportation_level)
```

```
train_data.shape, test_data.shape
```

```
train_data.corr()['등록차량수'].sort_values() ##단지내주차면수가 너무 큰 상관관계성을 띄고 있음
```

```
train_data['주차면수/총세대'] = train_data['단지내주차면수'] / train_data['총세대수']
test_data['주차면수/총세대'] = test_data['단지내주차면수'] / test_data['총세대수']

## log취하기
train_data['총세대수log'] = np.log1p(train_data['총세대수'])
test_data['총세대수log'] = np.log1p(test_data['총세대수'])

train_data['단지내주차면수log'] = np.log1p(train_data['단지내주차면수'])
test_data['단지내주차면수log'] = np.log1p(test_data['단지내주차면수'])

train_data['단지내주차면수/100'] = train_data['단지내주차면수']/100
test_data['단지내주차면수/100'] = test_data['단지내주차면수']/100

train_data['버스지하철log'] = np.log1p(train_data['대중교통현황'])
test_data['버스지하철log'] = np.log1p(test_data['대중교통현황'])

train_data['전용면적별세대수 평균 log'] = np.log1p(train_data['전용면적별세대수 평균'])
test_data['전용면적별세대수 평균 log'] = np.log1p(test_data['전용면적별세대수 평균'])

train_data['임대보증금 평균 log'] = np.log1p(train_data['임대보증금 평균'])
```

```
test_data['임대보증금 평균 log'] = np.log1p(test_data['임대보증금 평균'])

train_data['임대료 평균 log'] = np.log1p(train_data['임대료 평균'])
test_data['임대료 평균 log'] = np.log1p(test_data['임대료 평균'])

##target 값 비율로 바꾸고 필요 없는 특성 제거 (이를 target encoding 으로 칭하는 듯
함..?)
train_data['주차면수대비등록확률'] = train_data['등록차량수'] / train_data['단지내
주차면수']
train_data = train_data.drop(columns=['등록차량수'])

# train_data = train_data.drop(columns=['실거주가구'])
# test_data = test_data.drop(columns=['실거주가구'])

# train_data = train_data.drop(columns=['단지내주차면수'])
# test_data = test_data.drop(columns=['단지내주차면수'])
```

```
train_data.head()
```

```
test_data.head()
```

[Pycaret으로 모델 생성]

(`compare_models` : **PyCaret** 라이브러리에서 제공하는 함수로, 여러 머신러닝 모델을 비교하고 가장 성능이 좋은 모델들을 선택)

- Pycaret에 집어넣고 가장 성능이 잘 나오는 5개로 블랜딩해서 최종 모델을 생성
- 실제로 모델 튜닝을 더 진행하고 파라미터 값 등을 더 좋게 변경해줬을 때 public score가 더 떨어지거나 cv 값이 더 떨어짐...

```
reg = setup(data=train_data,
            target='주차면수대비등록확률',
            session_id = 201,
            numeric_imputation = 'mean',
            fold_shuffle = True,
            numeric_features=list(train_data.drop(columns = ['주차면수대비등록
확률']).columns),
            ignore_low_variance = True,
            combine_rare_levels = True, rare_level_threshold = 0.05,
            remove_multicollinearity = True, multicollinearity_threshold =
0.90,
            normalize = True,
            silent= True)
```

```
best_5_l = compare_models(sort='MAE', n_select=5)
```

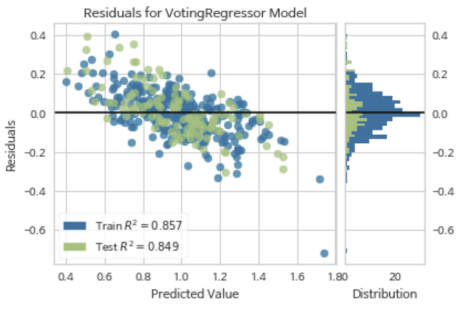
	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
rf	Random Forest Regressor	0.1985	0.0753	0.2706	0.2295	0.1391	0.2628	0.575
catboost	CatBoost Regressor	0.2004	0.0767	0.2736	0.2085	0.1404	0.2624	5.027
br	Bayesian Ridge	0.2052	0.0765	0.2719	0.2132	0.1399	0.2699	0.016
lightgbm	Light Gradient Boosting Machine	0.2078	0.0810	0.2809	0.1605	0.1431	0.2664	0.042
gbr	Gradient Boosting Regressor	0.2120	0.0845	0.2854	0.1368	0.1462	0.2755	0.117

```
best_5_l # 각 모델 튜닝하려고 추출했던 것이지만 과대적합 관련해서 default 값으로 진행
```

```
blended_l = blend_models(estimator_list= best_5_l, fold=5, optimize='MAE')
# 상위 5개의 모델을 결합(Blending)하여 하나의 앙상블 모델을 생성하고, 5-fold 교차 검증을 통해 성능을 평가함

pred_holdout = predict_model(blended_l)
# 검증 데이터를 사용하여 예측 결과를 반환하고 성능 평가

final_model_l = finalize_model(blended_l)
# 앙상블 모델을 최종화하여, 전체 데이터(학습 + 검증)를 사용해 다시 학습한 최종 모델을 생성함

pred_esb_l = predict_model(final_model_l, test_data)
# 최종화된 모델을 사용해 테스트 데이터에 대한 예측을 수행함
```

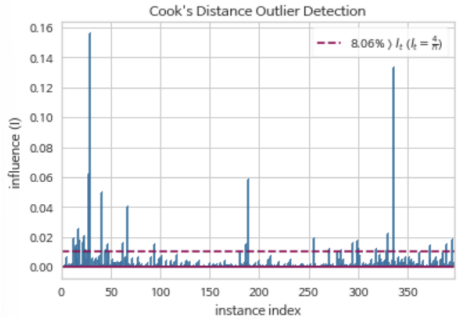
[모델 시각화]

```
plot_model(final_model_l, plot='residuals')
```



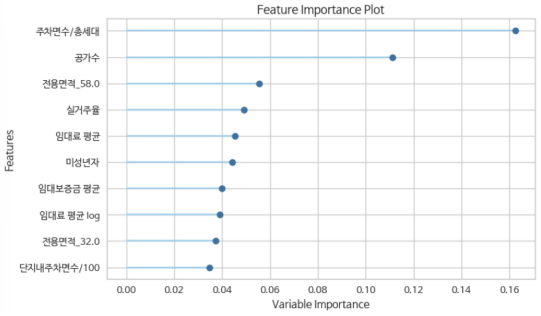
```
# plot_model(final_model_l, plot='learning')
```

```
plot_model(final_model_l, plot='cooks') ## 심한 아웃 라이어들이 존재하는 것을 볼 수 있습니다..
```




```
rf_model = create_model('rf')
```

```
plot_model(rf_model, plot='feature')
```




```
# plot_model(huber_model, plot='rfe')
```

```
sample_submission = pd.read_csv(os.path.join(data_path, 'sample_submission.csv'))
sample_submission.head()
```

```
pred_esb_l['예측'] = pred_esb_l['Label'] * pred_esb_l['단지내주차면수']
```

```
pred = pred_esb_l.loc['C1072':'C2189', '예측'].values
```

```
pred
```

```
array([ 715.47628571, 1213.09895727,  503.10909876,  541.21391526,
       1152.39240666, 1871.87129142, 1075.32150798,  531.56642893,
        371.9539117 ,  272.04405972,  458.09530239,  296.57328549,
        436.7749657 ,  263.34412069,  284.72162963,  223.90102075,
        469.22772347,  312.58555349,  156.44783054,  628.3146743 ,
        227.6224592 ,  396.05323855,  458.63354413,  399.5458289 ,
        363.53435037,  147.58768897,  144.88278099,  529.40125893,
        475.03933592,  498.81650837,  920.39354799,  149.501102  ,
        478.65805667,  251.51153403,  104.33955036,  352.88263642,
        376.77731254,  575.3960213 ,  783.51450824,  315.44684633,
        481.43039984,  513.92808366,  453.31298815,  592.98091434,
        865.2392907 , 1210.04051422,  482.1868088 ,  584.32543049,
        390.29541931,  355.93944414,  815.24266941,  277.83812918,
        960.64198344,  581.57530468,  629.7777787 ,  247.61705507,
        601.55339033,  264.14515564,  481.91449019,  184.83837316,
        315.91618455,  554.32831378,  993.07021128,  443.02088754,
        175.08944726,  269.84656158,  566.24147815,  856.53669993,
        528.14027585,  386.96940743,  687.61873508,  277.6987613 ,
        624.6655853 ,  915.38994079,  977.84697348,  440.51964171,
        658.89976005, 1051.29169519,  795.99522334,  915.00020949,
        860.9677109 , 1327.22661884,  337.3300379 ,  229.41860915,
        283.63725847,  229.61727356,  204.11659419,  378.57588068,
        322.12016453,  886.6214887 ,  913.85942038,  688.17434972,
        251.73466495,  674.62700338,  987.70165363,  838.76195282,
        604.7374485 , 1206.12743188,  752.41769713,  786.7581302 ,
        688.14884079,  402.31792443,  887.04856707,  744.25212767,
       1075.32558928,  576.57315792, 1146.65545914,  735.73594598,
        957.2228101 ,  289.37837758,  576.43194113, 1053.28694777,
        793.251024  ,  958.90320793,  759.90894762,  139.78741474,
        127.65108062,  563.20539688,  905.02148733, 1607.9385686 ,
        562.9352507 ,  781.09294345,  766.38325692,  358.51628296,
        771.93384407,  275.54826269,  373.3205473 ,  628.09901605,
         88.01496369,   29.50601069,   85.83927958,  591.99039856,
        524.22509488,  345.46523075,  268.21709106,  273.5585408 ,
        453.96301403,  489.60592905,  437.662872  ,  113.61269492,
        516.02096144,   62.41074869,  123.85152877,  232.77747605,
        313.56200712,  247.34278064,  475.88326636,   31.57776451,
        374.37734983,  183.45318001])
```

```
sample_submission['num'] = pred
sample_submission
```

```
data_path2 = os.path.join(workspace_path, 'parking2')
```

```
sample_submission.to_csv(os.path.join(data_path2, '최고점 재현_f.csv'), index
=False)
# 파일 경로를 동적으로 생성하여 재사용성을 높임
```

[배울 점]

1. 결측치 처리, 데이터 타입 변환, 그리고 데이터 전처리의 중요성
2. PyCaret를 활용하여 데이터 분석과 모델링 과정을 간소화할 수 있음.
3. PyCaret에서는 setup( )단계에서 train, holdout, test 데이터로 자동으로 나눔. train 데이터는 모델의 학습에, holdout 데이터는 학습에 사용되지 않은 데이터로 모델의 성능을 평가하는 데 이용된다는 걸 알게 됐다.
4. train = train[train.단지코드 != 'C1804'] 처럼 특정 값을 포함하는 행을 한번에 지우는 방법