```
import os
os.chdir('/content/drive/MyDrive/Colab_Notebooks/BIGCONTEST/데이터')
```

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

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler

from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from xgboost.sklearn import XGBClassifier, XGBRegressor
from lightgbm import LGBMRegressor

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import mean_squared_error
```

```
# 데이터 불러오기
data = pd.read_csv("raw_data_final.csv", encoding="cp949")
```

```
data['INDUSTRY_CD_label'] = data['INDUSTRY_CD'].str[1:].astype('int')
```

1. train, valid, test 데이터 업종별로 15개 나누고 스케일링

```
train = data[(data['train']==1)&(data['DATA_CRTR_YM']<=202210)].reset_index()
valid = data[(data['train']==1)&(data['DATA_CRTR_YM']<=202212)&(data['DATA_CRTR_YM']>=202211)].reset_index()
test = data[data['train']==0].reset_index()
```

```
for i in range(1,16):
  exec("train"+ str(i)+ "= train[train['INDUSTRY_CD_label'] == " + str(i)+"]")
  exec("valid"+ str(i)+ "= valid[valid['INDUSTRY_CD_label'] == " + str(i)+"]")
  exec("test"+ str(i)+ "= test[test['INDUSTRY_CD_label'] == " + str(i)+"]")
```

```
columns = ['전체점포수', '프랜차이즈점포수', '일반점포수', '길단위유동인구', '주거인구', '직장인구', '개업수', '폐업수', '개업률', '폐업률',
           '전체 임대료', '1층 임대료', '1층 외 임대료', '생활물가지수', '부동산거래대비유동인구', '공실률대비매매가임대료', '젠트리피케이션',
           '지하철개수', '스타벅스개수', '65세이상', '65세이상_남', '65세이상_여', '출근시간_승차수', '출근시간_하차수', '주말_하차수']
```

```
for i in range(1,16):
  exec("X_train"+ str(i)+ "= train"+ str(i)+ "[columns]")
  exec("y_train" + str(i)+ "= train"+ str(i)+ "['SLS_GRD']")
  exec("X_valid"+ str(i)+ "= valid"+ str(i)+ "[columns]")
  exec("y_valid" + str(i)+ "= valid"+ str(i)+ "['SLS_GRD']")
  exec("X_test"+ str(i)+ "= test"+ str(i)+ "[columns]")
```

```
features_to_scale = X_train1.columns

for i in range(1,16):
  exec("scaler"+ str(i)+ "=StandardScaler()")
  exec("scaler" + str(i)+ ".fit(X_train"+ str(i)+"[features_to_scale])")
  exec("X_train"+ str(i)+ "[features_to_scale] = scaler"+ str(i)+ ".transform(X_train"+ str(i)+ "[features_to_scale])")
  exec("X_valid"+ str(i)+ "[features_to_scale] = scaler"+ str(i)+ ".transform(X_valid"+ str(i)+ "[features_to_scale])")
  exec("X_test"+ str(i)+ "[features_to_scale] = scaler"+ str(i)+ ".transform(X_test"+ str(i)+ "[features_to_scale])")
  exec("X_train" + str(i) + "['대학교여부'] = train" +str(i) + "['대학교여부']" )
  exec("X_valid" + str(i) + "['대학교여부'] = valid" +str(i) + "['대학교여부']" )
  exec("X_test" + str(i) + "['대학교여부'] = test" +str(i) + "['대학교여부']" )
```

```
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>: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/indexing.html#returnin
<string>:1: SettingWithCopyWarning:
```

변수제거

강한 상관관계(피어슨 상관분석) 갖는 변수들 제거

```
# 업종코드 A01
cols = X_train1.columns
corr1 = X_train1.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr1.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr1.iloc[i, j])
```

```
일반점포수 전체점포수 0.9926273687759758
1층 임대료 전체 임대료 0.9315276209097479
1층 외 임대료 전체 임대료 0.9167354182014473
65세이상_남 65세이상 0.9933101531022159
65세이상_여 65세이상 0.995929980722027
65세이상_여 65세이상_남 0.9788593824976499
주말_하차수 출근시간_승차수 0.9519031952075494
```

```
X_train1_1 = X_train1.drop(columns=['전체점포수', '전체 임대료', '65세이상_남', '65세이상_여', '출근시간_승차수', '출근시간_하차수', '주말_하차수'
X_valid1_1 = X_valid1.drop(columns=['전체점포수', '전체 임대료', '65세이상_남', '65세이상_여', '출근시간_승차수', '출근시간_하차수', '주말_하차수'
X_test1_1 = X_test1.drop(columns=['전체점포수', '전체 임대료', '65세이상_남', '65세이상_여', '출근시간_승차수', '출근시간_하차수', '주말_하차수'],
```

```
# A02
cols = X_train2.columns
corr2 = X_train2.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr2.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr2.iloc[i, j])
```

```
프랜차이즈점포수 전체점포수 0.9295485207794754
1층 임대료 전체 임대료 0.9229940973590847
```

```
1층 외 임대료 전체 임대료 0.9014674960912069
65세이상_남 65세이상 0.9926805992333715
65세이상_여 65세이상 0.9955496938889682
65세이상_여 65세이상_남 0.9768818037386374
출근시간_하차수 출근시간_승차수 0.9624905827458538
주말_하차수 출근시간_승차수 0.9630212852084035
주말_하차수 출근시간_하차수 0.9813962612238577
```

```
X_train2_1 = X_train2.drop(columns=['출근시간_승차수','65세이상_남', '65세이상_여', '주말_하차수'], axis = 1)
X_valid2_1 = X_valid2.drop(columns=['출근시간_승차수','65세이상_남', '65세이상_여', '주말_하차수'], axis = 1)
X_test2_1 = X_test2.drop(columns=['출근시간_승차수','65세이상_남', '65세이상_여', '주말_하차수'], axis = 1)
```

```
# A03
cols = X_train3.columns
corr3 = X_train3.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr3.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr3.iloc[i, j])
```

```
프랜차이즈점포수 전체점포수 0.943422014887277
일반점포수 전체점포수 0.9983554931174895
일반점포수 프랜차이즈점포수 0.9231759721760925
개업수 전체점포수 0.9293291507212502
개업수 일반점포수 0.9341110055074336
폐업수 전체점포수 0.9194836779362946
폐업수 일반점포수 0.9196618661503229
폐업수 개업수 0.9280880412457395
1층 임대료 전체 임대료 0.9284204836024235
1층 외 임대료 전체 임대료 0.9082848617031167
65세이상_남 65세이상 0.9936287605165895
65세이상_여 65세이상 0.996095069256475
65세이상_여 65세이상_남 0.9797985186762769
주말_하차수 출근시간_승차수 0.9400753819503475
```

```
X_train3_1 = X_train3.drop(columns=['전체점포수','65세이상_남', '65세이상_여'], axis = 1)
X_valid3_1 = X_valid3.drop(columns=['전체점포수','65세이상_남', '65세이상_여'], axis = 1)
X_test3_1 = X_test3.drop(columns=['전체점포수','65세이상_남', '65세이상_여'], axis = 1)
```

```
cols = X_train4.columns
```

```
# 업종코드 A04
corr4 = X_train4.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr4.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr4.iloc[i, j])
```

```
일반점포수 전체점포수 0.9673248735599654
1층 임대료 전체 임대료 0.923989515027985
65세이상_남 65세이상 0.9930941546490567
65세이상_여 65세이상 0.9958082206321169
65세이상_여 65세이상_남 0.9782005516530822
```

```
# A04 : 65세이상_남, 65세이상_여 제거
X_train4_1 = X_train4.drop(['65세이상_남', '65세이상_여'], axis = 1)
X_valid4_1 = X_valid4.drop(['65세이상_남', '65세이상_여'], axis = 1)
X_test4_1 = X_test4.drop(['65세이상_남', '65세이상_여'], axis = 1)
```

```
# 업종코드 A05
corr5 = X_train5.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr5.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr5.iloc[i, j])
```

```
프랜차이즈점포수 전체점포수 0.9157110725775103
일반점포수 전체점포수 0.9809058242380593
1층 임대료 전체 임대료 0.921979793184687
65세이상_남 65세이상 0.9925635009621393
65세이상_여 65세이상 0.9954845807364625
65세이상_여 65세이상_남 0.97652683015752
주말_하차수 출근시간_승차수 0.9726047713981878
```

```
# A05 : 65세이상_남, 65세이상_여 제거
X_train5_1 = X_train5.drop(['65세이상_남', '65세이상_여'], axis = 1)
X_valid5_1 = X_valid5.drop(['65세이상_남', '65세이상_여'], axis = 1)
X_test5_1 = X_test5.drop(['65세이상_남', '65세이상_여'], axis = 1)
```

```
# 업종코드 A06
corr6 = X_train6.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr6.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr6.iloc[i, j])
```

```
일반점포수 전체점포수 0.9945492859557566
1층 임대료 전체 임대료 0.929330988313231
1층 외 임대료 전체 임대료 0.9089112962701821
65세이상_남 65세이상 0.9929555410663926
65세이상_여 65세이상 0.9957185229190008
65세이상_여 65세이상_남 0.9777515778953015
```

```
# A06 : 일반점포수, 65세이상_남, 65세이상_여 제거
X_train6_1 = X_train6.drop(['일반점포수', '65세이상_남', '65세이상_여'], axis = 1)
X_valid6_1 = X_valid6.drop(['일반점포수', '65세이상_남', '65세이상_여'], axis = 1)
X_test6_1 = X_test6.drop(['일반점포수', '65세이상_남', '65세이상_여'], axis = 1)
```

```
# 업종코드 A07
corr7 = X_train7.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr7.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr7.iloc[i, j])
```

```
프랜차이즈점포수 전체점포수 0.918155697704793
일반점포수 전체점포수 0.9803313263213509
1층 임대료 전체 임대료 0.9139678133378473
65세이상_남 65세이상 0.9924861625636805
65세이상_여 65세이상 0.995412804227946
65세이상_여 65세이상_남 0.9762271633542846
출근시간_하차수 출근시간_승차수 0.9990503806203103
주말_하차수 출근시간_승차수 0.9881925099572493
주말_하차수 출근시간_하차수 0.9870486184506275
```

```
# A07 : 65세이상_남, 65세이상_여, 출근시간_승차수 제거
X_train7_1 = X_train7.drop(['65세이상_남', '65세이상_여', '출근시간_승차수'], axis = 1)
X_valid7_1 = X_valid7.drop(['65세이상_남', '65세이상_여', '출근시간_승차수'], axis = 1)
X_test7_1 = X_test7.drop(['65세이상_남', '65세이상_여', '출근시간_승차수'], axis = 1)
```

```
# 업종코드 A08
cols = X_train8.columns
corr8 = X_train8.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr8.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr8.iloc[i, j])
```

```
일반점포수 전체점포수 0.987050144972155
개업수 일반점포수 0.9107169486928015
1층 임대료 전체 임대료 0.928159570436962
1층 외 임대료 전체 임대료 0.9072671344214377
65세이상_남 65세이상 0.9930597624820283
65세이상_여 65세이상 0.9957559145027765
65세이상_여 65세이상_남 0.978020999390672
```

```
#상관계수 & 중요도 기준으로 변수선택
X_train8_1 = X_train8.drop(columns=['일반점포수', '65세이상_남', '65세이상_여'], axis = 1)
X_valid8_1 = X_valid8.drop(columns=['일반점포수', '65세이상_남', '65세이상_여'], axis = 1)
X_test8_1 = X_test8.drop(columns=['일반점포수', '65세이상_남', '65세이상_여'], axis = 1)
```

```
# 업종코드 A09
cols = X_train9.columns
corr9 = X_train9.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr9.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr9.iloc[i, j])
```

```
프랜차이즈점포수 전체점포수 0.9119817148067038
일반점포수 전체점포수 0.9892657669743133
1층 임대료 전체 임대료 0.9228584147565753
65세이상_남 65세이상 0.9915368824277295
65세이상_여 65세이상 0.9950036739414526
65세이상_여 65세이상_남 0.9736213009910333
출근시간_하차수 출근시간_승차수 0.9993674100137823
주말_하차수 출근시간_승차수 0.990586478076013
주말_하차수 출근시간_하차수 0.9900423834007017
```

```
#상관계수 & 중요도 기준으로 변수선택
X_train9_1 = X_train9.drop(columns=['주말_하차수', '출근시간_승차수', '65세이상_남','65세이상_여', '일반점포수'], axis = 1)
X_valid9_1 = X_valid9.drop(columns=['주말_하차수', '출근시간_승차수', '65세이상_남','65세이상_여', '일반점포수'], axis = 1)
X_test9_1 = X_test9.drop(columns=['주말_하차수', '출근시간_승차수', '65세이상_남','65세이상_여', '일반점포수'], axis = 1)
```

```
# 업종코드 A10
cols = X_train10.columns
corr10 = X_train10.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr10.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr10.iloc[i, j])
```

```
프랜차이즈점포수 전체점포수 0.9419927808513194
일반점포수 전체점포수 0.9981304790086308
일반점포수 프랜차이즈점포수 0.9202376086009763
개업수 전체점포수 0.9261016801980197
개업수 일반점포수 0.9295246702037379
폐업수 전체점포수 0.9145005279580005
폐업수 일반점포수 0.9146256409419151
폐업수 개업수 0.9267723204572912
1층 임대료 전체 임대료 0.9254024187990108
65세이상_남 65세이상 0.9927896686133755
65세이상_여 65세이상 0.995501156549086
65세이상_여 65세이상_남 0.9769657101980755
출근시간_하차수 출근시간_승차수 0.9996153069959401
주말_하차수 출근시간_승차수 0.9924309613179573
주말_하차수 출근시간_하차수 0.9912234826425981
```

```
# 상관계수 & 중요도 기준으로 변수선택
X_train10_1 = X_train10.drop(columns=['출근시간_승차수', '출근시간_하차수', '65세이상_남', '65세이상_여' , '일반점포수'], axis = 1)
X_valid10_1 = X_valid10.drop(columns=['출근시간_승차수', '출근시간_하차수', '65세이상_남', '65세이상_여' , '일반점포수'], axis = 1)
X_test10_1 = X_test10.drop(columns=['출근시간_승차수', '출근시간_하차수', '65세이상_남', '65세이상_여' , '일반점포수'], axis = 1)
```

```
# 업종코드 A11
cols = X_train11.columns
corr11 = X_train11.corr()
for i in range(len(cols)) :
  for j in range(len(cols)) :
    if i > j and corr11.iloc[i, j] > 0.9 :
      print(cols[i], cols[j], corr11.iloc[i, j])
```

```
일반점포수 전체점포수 0.9988014566227834
1층 임대료 전체 임대료 0.9325053649932055
1층 외 임대료 전체 임대료 0.9100924514499427
65세이상_남 65세이상 0.9927740581459378
65세이상_여 65세이상 0.9955715154015985
65세이상_여 65세이상_남 0.9770968363899589
주말_하차수 출근시간_승차수 0.977861062227386
```

```
# 상관계수 & 중요도 기준으로 변수선택
X_train11_1 = X_train11.drop(columns=['65세이상_남','65세이상_여', '전체점포수'], axis = 1)
X_valid11_1 = X_valid11.drop(columns=['65세이상_남','65세이상_여', '전체점포수'], axis = 1)
X_test11_1 = X_test11.drop(columns=['65세이상_남','65세이상_여', '전체점포수'], axis = 1)
```

```
# 업종코드 A12
corr12 = X_train12.corr()
cols=X_train12.columns
for i in range(len(cols)) :
    for j in range(len(cols)) :
        if i > j and abs(corr12.iloc[i, j]) > 0.9 :
            print(cols[i], cols[j], corr12.iloc[i, j])
```

```
일반점포수 전체점포수 0.9978319010080221
1층 임대료 전체 임대료 0.9413459571854338
1층 외 임대료 전체 임대료 0.9344866446906348
65세이상_남 65세이상 0.9939178411758572
65세이상_여 65세이상 0.9964163686913353
65세이상_여 65세이상_남 0.9810412876780098
출근시간_하차수 출근시간_승차수 0.9992791437992508
주말_하차수 출근시간_승차수 0.9912758743047618
주말_하차수 출근시간_하차수 0.9893563625782982
```

상관관계가 높은 관계(0.9이상)들 중 변수 중요도를 활용해 변수 선택

- 일반점포수, 전체점포수 중 '전체점포수' 삭제
- 65세이상_남, 65세이상_여, 65세이상 중 '65세이상_남', '65세이상_여' 삭제
- 주말_하차수, 출근시간_승차수,출근시간_하차수 중 '주말_하차수','출근시간_하차수' 삭제
- 1층 임대료, 전체 임대료, 1층 외 임대료 중 '전체 임대료' 삭제

```
X_train12_drop = X_train12.drop(columns=['전체점포수','65세이상_남', '65세이상_여','주말_하차수','출근시간_하차수','전체 임대료'])
X_valid12_drop = X_valid12.drop(columns=['전체점포수','65세이상_남', '65세이상_여','주말_하차수','출근시간_하차수','전체 임대료'])
X_test12_drop = X_test12.drop(columns=['전체점포수','65세이상_남', '65세이상_여','주말_하차수','출근시간_하차수','전체 임대료'])
```

```
# 업종코드 A13
corr13 = X_train13.corr()
cols=X_train13.columns
for i in range(len(cols)) :
    for j in range(len(cols)) :
        if i > j and abs(corr13.iloc[i, j]) > 0.9 :
            print(cols[i], cols[j], corr13.iloc[i, j])
```

```
프랜차이즈점포수 전체점포수 0.9126604792868318
일반점포수 전체점포수 0.9987574852253553
1층 임대료 전체 임대료 0.9265064518983029
1층 외 임대료 전체 임대료 0.9003770195437326
65세이상_남 65세이상 0.9928685004923449
65세이상_여 65세이상 0.995591957850369
65세이상_여 65세이상_남 0.97731069234614
주말_하차수 출근시간_승차수 0.9696871514987467
```

상관관계가 높은 관계(0.95이상)들 중 변수 중요도를 활용해 변수 선택

- 일반점포수, 전체점포수 중 '전체점포수' 삭제
- 65세이상_남, 65세이상_여, 65세이상 중 '65세이상_남', '65세이상_여' 삭제
- 주말_하차수, 출근시간_승차수 중 '주말_하차수' 삭제

```
X_train13_drop = X_train13.drop(columns=['전체점포수','65세이상_남', '65세이상_여','주말_하차수'])
X_valid13_drop = X_valid13.drop(columns=['전체점포수','65세이상_남', '65세이상_여','주말_하차수'])
X_test13_drop = X_test13.drop(columns=['전체점포수','65세이상_남', '65세이상_여','주말_하차수'])
```

```
# 업종코드 A14
corr14 = X_train14.corr()
cols=X_train14.columns
for i in range(len(cols)) :
    for j in range(len(cols)) :
        if i > j and abs(corr14.iloc[i, j]) > 0.9 :
            print(cols[i], cols[j], corr14.iloc[i, j])
```

```
일반점포수 전체점포수 0.9987647741473972
1층 임대료 전체 임대료 0.9145453433279459
65세이상_남 65세이상 0.9929011824456447
65세이상_여 65세이상 0.9954923446698459
65세이상_여 65세이상_남 0.9771448203313169
주말_하차수 출근시간_승차수 0.978629310758685
```

상관관계가 높은 관계(0.95이상)들 중 변수 중요도를 활용해 변수 선택

- 일반점포수, 전체점포수 중 '일반점포수' 삭제
- 65세이상_남, 65세이상_여, 65세이상 중 '65세이상_남', '65세이상_여' 삭제
- 주말_하차수, 출근시간_승차수 중 '주말_하차수' 삭제

```
X_train14_drop = X_train14.drop(columns=['일반점포수','65세이상_남', '65세이상_여','주말_하차수'])
X_valid14_drop = X_valid14.drop(columns=['일반점포수','65세이상_남', '65세이상_여','주말_하차수'])
X_test14_drop = X_test14.drop(columns=['일반점포수','65세이상_남', '65세이상_여','주말_하차수'])
```

```
# 업종코드 A15
corr15 = X_train15.corr()
cols=X_train15.columns
for i in range(len(cols)) :
    for j in range(len(cols)) :
        if i > j and abs(corr15.iloc[i, j]) > 0.9 :
            print(cols[i], cols[j], corr15.iloc[i, j])
```

```
프랜차이즈점포수 전체점포수 0.9260444930604556
일반점포수 전체점포수 0.9878451544809436
1층 임대료 전체 임대료 0.9347805958468048
1층 외 임대료 전체 임대료 0.9058506595155977
65세이상_남 65세이상 0.9919013783556273
65세이상_여 65세이상 0.9952579478139655
65세이상_여 65세이상_남 0.9748433159814961
```

상관관계가 높은 관계(0.95이상)들 중 변수 중요도를 활용해 변수 선택

- 프랜차이즈점포수, 일반점포수, 전체점포수 중 '전체점포수' 삭제
- 65세이상_남, 65세이상_여, 65세이상 중 '65세이상_남', '65세이상_여' 삭제

```
X_train15_drop = X_train15.drop(columns=['전체점포수','65세이상_남', '65세이상_여'])
X_valid15_drop = X_valid15.drop(columns=['전체점포수','65세이상_남', '65세이상_여'])
X_test15_drop = X_test15.drop(columns=['전체점포수','65세이상_남', '65세이상_여'])
```

A01

```
etr1_1 = ExtraTreesRegressor(random_state=0, min_samples_leaf= 1, min_samples_split= 2, n_estimators= 300)
etr1_1.fit(X_train1_1, y_train1)

pred_train1_rgr = etr1_1.predict(X_train1_1)
pred_val1_rgr = etr1_1.predict(X_valid1_1)

pred_train1_rgr_round = round(pd.DataFrame(pred_train1_rgr))
pred_val1_rgr_round = round(pd.DataFrame(pred_val1_rgr))

print('MSE of train1_1:', mean_squared_error(y_train1, pred_train1_rgr_round))
print('MSE of valid1_1:', mean_squared_error(y_valid1, pred_val1_rgr_round))


## 테스트 데이터 예측
pred_test1 = etr1_1.predict(X_test1_1)
pred_test1_round = round(pd.DataFrame(pred_test1))

## 제출 형식으로 바꾸기
pred_test_a01 = pd.concat([test1[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_te
pred_test_a01.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a01_202301 = pred_test_a01[pred_test_a01['DATA_CRTR_YM']==202301]
pred_test_a01_202302 = pred_test_a01[pred_test_a01['DATA_CRTR_YM']==202302]
pred_test_a01_202301 = pred_test_a01_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a01_202302 = pred_test_a01_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a01_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a01_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a01_202301['SLS_GRD_2302'] = pred_test_a01_202302['SLS_GRD_2302']
pred_test_a01 = pred_test_a01_202301
pred_test_a01.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

A02

```
# 파라미터 튜닝
'''
## ExtraTreesClassifier 모델 생성
clf = ExtraTreesClassifier(random_state=0)

## 탐색할 하이퍼파라미터 그리드 정의
param_grid = {
    'n_estimators': [100, 200, 300],  # 트리 개수
    'max_depth': [None, 10, 20],  # 트리의 최대 깊이
    'min_samples_split': [2, 5, 10],  # 노드 분할 최소 샘플 수
    'min_samples_leaf': [1, 2, 4],  # 리프 노드 최소 샘플 수
}

## Grid Search 객체 생성
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)

## Grid Search 수행
grid_search.fit(X_train2_1, y_train2)

## 최적의 하이퍼파라미터 출력
print("최적 하이퍼파라미터:", grid_search.best_params_)

## 최적 모델의 성능 출력
print("최적 모델의 MSE:", -grid_search.best_score_)  # 음수 MSE 값을 양수로 변환하여 출력
'''


et2 = ExtraTreesClassifier(random_state=0, min_samples_leaf=1, min_samples_split=2, n_estimators=300)
et2.fit(X_train2_1, y_train2)

pred_train2 = et2.predict(X_train2_1)
```

```
pred_val2 = et2.predict(X_valid2_1)

print('Accuracy of train2:', precision_score(y_train2, pred_train2, average='macro'))
print('Accuracy of valid2:', precision_score(y_valid2, pred_val2, average='macro'))
print('MSE of train2:', mean_squared_error(y_train2, pred_train2))
print('MSE of valid2:', mean_squared_error(y_valid2, pred_val2))


## 테스트 데이터 예측
pred_test2 = et2.predict(X_test2_1)
pred_test2_round = round(pd.DataFrame(pred_test2))

## 제출 형식으로 바꾸기
pred_test_a02 = pd.concat([test2[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_te
pred_test_a02.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a02_202301 = pred_test_a02[pred_test_a02['DATA_CRTR_YM']==202301]
pred_test_a02_202302 = pred_test_a02[pred_test_a02['DATA_CRTR_YM']==202302]
pred_test_a02_202301 = pred_test_a02_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a02_202302 = pred_test_a02_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a02_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a02_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a02_202301['SLS_GRD_2302'] = pred_test_a02_202302['SLS_GRD_2302']
pred_test_a02 = pred_test_a02_202301
pred_test_a02.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

A03

```
etr3_1 = ExtraTreesRegressor(random_state=0, n_estimators=300)
etr3_1.fit(X_train3_1, y_train3)

pred_train3_1 = etr3_1.predict(X_train3_1)
pred_val3_1 = etr3_1.predict(X_valid3_1)

pred_train3_1 = round(pd.DataFrame(pred_train3_1))
pred_val3_1 = round(pd.DataFrame(pred_val3_1))

print('Accuracy of train3_1:', precision_score(y_train3, pred_train3_1, average='macro'))
print('Accuracy of valid3_1:', precision_score(y_valid3, pred_val3_1, average='macro'))
print('MSE of train3_1:', mean_squared_error(y_train3, pred_train3_1))
print('MSE of valid3_1:', mean_squared_error(y_valid3, pred_val3_1))
```

```
Accuracy of train3_1: 0.9914882918519086
Accuracy of valid3_1: 0.6504189452182882
MSE of train3_1: 0.015529033314117755
MSE of valid3_1: 0.5556496875992799
```

```
## 테스트 데이터 예측
pred_test3 = etr3_1.predict(X_test3_1)
pred_test3_round = round(pd.DataFrame(pred_test3))

## 제출 형식으로 바꾸기
pred_test_a03 = pd.concat([test3[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_te
pred_test_a03.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a03_202301 = pred_test_a03[pred_test_a03['DATA_CRTR_YM']==202301]
pred_test_a03_202302 = pred_test_a03[pred_test_a03['DATA_CRTR_YM']==202302]
pred_test_a03_202301 = pred_test_a03_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a03_202302 = pred_test_a03_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a03_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a03_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a03_202301['SLS_GRD_2302'] = pred_test_a03_202302['SLS_GRD_2302']
pred_test_a03 = pred_test_a03_202301
pred_test_a03.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

A04

분류, 회귀 모델의 결과값을 평균내서 사용

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 8, 16],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


# 분류 모델 그리드서치
'''
model = ExtraTreesClassifier(random_state = 0)
grid_search4 = GridSearchCV(model, param_grid, cv = 5, scoring = 'neg_mean_squared_error', n_jobs = -1)
grid_search4.fit(X_train4_1, y_train4)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search4.best_params_)

# 최적 모델 출력
best_model4 = grid_search4.best_estimator_
print(best_model4)
'''
```

```
    '\nmodel = ExtraTreesClassifier(random_state = 0)\ngrid_search4 = GridSearchCV(model, param_grid, cv = 5, scoring = \'neg
    id_search4.best_params_)\n\n# 최적 모델 출력\nbest_model4 = grid_search4.best_estimator_\nprint(best_model4)\n'
```

```
# 회귀 모델 그리드서치
'''
model = ExtraTreesRegressor(random_state = 0)
grid_search4r = GridSearchCV(model, param_grid, cv = 5, scoring = 'neg_mean_squared_error', n_jobs = -1)
grid_search4r.fit(X_train4_1, y_train4)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search4r.best_params_)

# 최적 모델 출력
best_model4r = grid_search4r.best_estimator_
print(best_model4r)
'''
```

```
    '\nmodel = ExtraTreesRegressor(random_state = 0)\ngrid_search4r = GridSearchCV(model, param_grid, cv = 5, scoring = \'neg
    rid_search4r.best_params_)\n\n# 최적 모델 출력\nbest_model4r = grid_search4r.best_estimator_\nprint(best_model4r)\n'
```

```
# 그리드서치 결과 도출된 최적의 파라미터 값 사용
best_model4 = ExtraTreesClassifier(n_estimators=200, random_state=0)
best_model4.fit(X_train4_1, y_train4)
pred4 = best_model4.predict(X_valid4_1)


best_model4r = ExtraTreesRegressor(n_estimators=300, random_state=0)
best_model4r.fit(X_train4_1, y_train4)
pred4r = best_model4r.predict(X_valid4_1)


# 그리드서치 완료한 2개 모델 앙상블(평균내기)
pred4_e = np.round((pred4 + pred4r) / 2)
print('A04 ensemble MSE : {:.4f}'.format(mean_squared_error(y_valid4, pred4_e)))
```

```
    A04 ensemble MSE : 0.3878
```

```
## 테스트 데이터 예측
test_pred4 = best_model4.predict(X_test4_1)
test_pred4r = best_model4r.predict(X_test4_1)

pred_test4_round = round(pd.DataFrame((test_pred4 + test_pred4r)/ 2))

## 제출 형식으로 바꾸기
pred_test_a04 = pd.concat([test4[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_te
pred_test_a04.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a04_202301 = pred_test_a04[pred_test_a04['DATA_CRTR_YM']==202301]
pred_test_a04_202302 = pred_test_a04[pred_test_a04['DATA_CRTR_YM']==202302]
pred_test_a04_202301 = pred_test_a04_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a04_202302 = pred_test_a04_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a04_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a04_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a04_202301['SLS_GRD_2302'] = pred_test_a04_202302['SLS_GRD_2302']
pred_test_a04 = pred_test_a04_202301
pred_test_a04.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

A05

회귀 단일 모델 사용

```
# 회귀 모델 그리드서치
'''
model = ExtraTreesRegressor(random_state = 0)
grid_search5r = GridSearchCV(model, param_grid, cv = 5, scoring = 'neg_mean_squared_error', n_jobs = -1)
grid_search5r.fit(X_train5_1, y_train5)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search5r.best_params_)

# 최적 모델 출력
best_model5r = grid_search5r.best_estimator_
print(best_model5r)
'''
```

```
'\nmodel = ExtraTreesRegressor(random_state = 0)\ngrid_search5r = GridSearchCV(model, param_grid, cv = 5, scoring = \'neg
rid_search5r.best_params_)\n\n# 최적 모델 출력\nbest_model5r = grid_search5r.best_estimator_\nprint(best_model5r)\n'
```

```
best_model5r = ExtraTreesRegressor(n_estimators=300, random_state=0)
best_model5r.fit(X_train5_1, y_train5)
pred5r = best_model5r.predict(X_valid5_1)
pred5_round = np.round(pred5r)
print('A05 Regression MSE : {:.4f}'.format(mean_squared_error(y_valid5, pred5_round)))
```

```
A05 Regression MSE : 0.3903
```

```
## 테스트 데이터 예측
pred_test5 = best_model5r.predict(X_test5_1)
pred_test5_round = round(pd.DataFrame(pred_test5))

## 제출 형식으로 바꾸기
pred_test_a05 = pd.concat([test5[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_te
pred_test_a05.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a05_202301 = pred_test_a05[pred_test_a05['DATA_CRTR_YM']==202301]
pred_test_a05_202302 = pred_test_a05[pred_test_a05['DATA_CRTR_YM']==202302]
pred_test_a05_202301 = pred_test_a05_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a05_202302 = pred_test_a05_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a05_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a05_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a05_202301['SLS_GRD_2302'] = pred_test_a05_202302['SLS_GRD_2302']
pred_test_a05 = pred_test_a05_202301
pred_test_a05.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

A06

분류, 회귀 모델의 결과값을 평균내서 사용

```
# 분류 모델 그리드서치
'''
model = ExtraTreesClassifier(random_state = 0)
grid_search6 = GridSearchCV(model, param_grid, cv = 5, scoring = 'neg_mean_squared_error', n_jobs = -1)
grid_search6.fit(X_train6_1, y_train6)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search6.best_params_)

# 최적 모델 출력
best_model6 = grid_search6.best_estimator_
print(best_model6)
'''
```

```
'\nmodel = ExtraTreesClassifier(random_state = 0)\ngrid_search6 = GridSearchCV(model, param_grid, cv = 5, scoring = \'neg
id_search6.best_params_)\n\n# 최적 모델 출력\nbest_model6 = grid_search6.best_estimator_\nprint(best_model6)\n'
```

```
# 회귀 모델 그리드서치
'''
model = ExtraTreesRegressor(random_state = 0)
```

```
grid_search6r = GridSearchCV(model, param_grid, cv = 5, scoring = 'neg_mean_squared_error', n_jobs = -1)
grid_search6r.fit(X_train6_1, y_train6)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search6r.best_params_)

# 최적 모델 출력
best_model6r = grid_search6r.best_estimator_
print(best_model6r)
'''
```

```
    '\nmodel = ExtraTreesRegressor(random_state = 0)\ngrid_search6r = GridSearchCV(model, param_grid, cv = 5, scoring = \'ne
    rid_search6r.best_params_)\n\n# 최적 모델 출력\nbest_model6r = grid_search6r.best_estimator_\nprint(best_model6r)\n'
```

```
best_model6 = ExtraTreesClassifier(random_state=0)
best_model6.fit(X_train6_1, y_train6)
pred6 = best_model6.predict(X_valid6_1)
```

```
best_model6r = ExtraTreesRegressor(n_estimators=300, random_state=0)
best_model6r.fit(X_train6_1, y_train6)
pred6r = best_model6r.predict(X_valid6_1)
```

```
# 그리드서치 완료한 2개 모델 앙상블(평균내기)
pred6_e = np.round((pred6 + pred6r) / 2)
print('A06 ensemble MSE : {:.4f}'.format(mean_squared_error(y_valid6, pred6_e)))
```

```
    A06 ensemble MSE : 0.3811
```

```
## 테스트 데이터 예측
test_pred6 = best_model6.predict(X_test6_1)
test_pred6r = best_model6r.predict(X_test6_1)

pred_test6_round = round(pd.DataFrame((test_pred6 + test_pred6r)/ 2))

## 제출 형식으로 바꾸기
pred_test_a06 = pd.concat([test6[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_te
pred_test_a06.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a06_202301 = pred_test_a06[pred_test_a06['DATA_CRTR_YM']==202301]
pred_test_a06_202302 = pred_test_a06[pred_test_a06['DATA_CRTR_YM']==202302]
pred_test_a06_202301 = pred_test_a06_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a06_202302 = pred_test_a06_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a06_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a06_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a06_202301['SLS_GRD_2302'] = pred_test_a06_202302['SLS_GRD_2302']
pred_test_a06 = pred_test_a06_202301
pred_test_a06.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

A07

분류, 회귀 모델의 결과값을 평균내서 사용

```
# 분류 모델 그리드서치
'''
model = ExtraTreesClassifier(random_state = 0)
grid_search7 = GridSearchCV(model, param_grid, cv = 5, scoring = 'neg_mean_squared_error', n_jobs = -1)
grid_search7.fit(X_train7_1, y_train7)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search7.best_params_)

# 최적 모델 출력
best_model7 = grid_search7.best_estimator_
print(best_model7)
'''
```

```
    '\nmodel = ExtraTreesClassifier(random_state = 0)\ngrid_search7 = GridSearchCV(model, param_grid, cv = 5, scoring = \'ne
    id_search7.best_params_)\n\n# 최적 모델 출력\nbest_model7 = grid_search7.best_estimator_\nprint(best_model7)\n'
```

```
# 회귀 모델 그리드서치
'''
model = ExtraTreesRegressor(random_state = 0)
grid_search7r = GridSearchCV(model, param_grid, cv = 5, scoring = 'neg_mean_squared_error', n_jobs = -1)
```

```
grid_search7r.fit(X_train7_1, y_train7)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search7r.best_params_)

# 최적 모델 출력
best_model7r = grid_search7r.best_estimator_
print(best_model7r)
'''
```

```
'\nmodel = ExtraTreesRegressor(random_state = 0)\ngrid_search7r = GridSearchCV(model, param_grid, cv = 5, scoring = \'ne
rid_search7r.best_params_)\n\n# 최적 모델 출력\nbest_model7r = grid_search7r.best_estimator_\nprint(best_model7r)\n'
```

```
best_model7 = ExtraTreesClassifier(n_estimators=200, random_state=0)
best_model7.fit(X_train7_1, y_train7)
pred7 = best_model7.predict(X_valid7_1)
```

```
best_model7r = ExtraTreesRegressor(min_samples_split=5, n_estimators=300, random_state=0)
best_model7r.fit(X_train7_1, y_train7)
pred7r = best_model7r.predict(X_valid7_1)
```

```
# 그리드서치 완료한 2개 모델 앙상블(평균내기)
pred7_e = np.round((pred7 + pred7r) / 2)
print('A07 ensemble MSE : {:.4f}'.format(mean_squared_error(y_valid7, pred7_e)))
```

```
A07 ensemble MSE : 0.4889
```

```
## 테스트 데이터 예측
test_pred7 = best_model7.predict(X_test7_1)
test_pred7r = best_model7r.predict(X_test7_1)

pred_test7_round = round(pd.DataFrame((test_pred7 + test_pred7r)/ 2))

## 제출 형식으로 바꾸기
pred_test_a07 = pd.concat([test7[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_te
pred_test_a07.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a07_202301 = pred_test_a07[pred_test_a07['DATA_CRTR_YM']==202301]
pred_test_a07_202302 = pred_test_a07[pred_test_a07['DATA_CRTR_YM']==202302]
pred_test_a07_202301 = pred_test_a07_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a07_202302 = pred_test_a07_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a07_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a07_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a07_202301['SLS_GRD_2302'] = pred_test_a07_202302['SLS_GRD_2302']
pred_test_a07 = pred_test_a07_202301
pred_test_a07.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

▸ A08

[ ] ↳ 숨겨진 셀 4개

▾ A09

```
# 파라미터 튜닝
'''
# ExtraTreesRegressor 모델 생성
reg = ExtraTreesRegressor(random_state=0)

# 탐색할 하이퍼파라미터 그리드 정의
param_grid = {
    'n_estimators': [100, 200, 300],  # 트리 개수
    'max_depth': [None, 10, 20, 30],  # 트리의 최대 깊이
    'min_samples_split': [2, 5, 10],  # 노드 분할 최소 샘플 수
    'min_samples_leaf': [1, 2, 4],  # 리프 노드 최소 샘플 수
    'max_features': ['auto', 'sqrt', 'log2'],  # 최대 특성 개수 설정
}

# Grid Search 객체 생성
grid_search = GridSearchCV(estimator=reg, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)

# Grid Search 수행
grid_search.fit(X_train9_1, y_train9)
```

```
# 최적의 하이퍼파라미터 출력
print("최적 하이퍼파라미터:", grid_search.best_params_)

# 최적 모델의 성능 출력
print("최적 모델의 MSE:", -grid_search.best_score_)  # 음수 MSE 값을 양수로 변환하여 출력
'''
```

```
'\n# ExtraTreesRegressor 모델 생성\nreg = ExtraTreesRegressor(random_state=0)\n\n# 탐색할 하이퍼파라미터 그리드 정의\nparam_grid =
t\': [2, 5, 10],  # 노드 분할 최소 샘플 수\n    \'min_samples_leaf\': [1, 2, 4],  # 리프 노드 최소 샘플 수\n    \'max_features\': [
param_grid, scoring=\'neg_mean_squared_error\', cv=5)\n\n# Grid Search 수행\ngrid_search.fit(X_train9_1, y_train9)\n\n# 최
st_score_)  # 음수 MSE 값을 양수로 변환하여 출력\n'
```

```
etr9_1_reg = ExtraTreesRegressor(max_depth=30,
                                 max_features= 'sqrt',
                                 min_samples_leaf= 1,
                                 min_samples_split= 2,
                                 n_estimators=200,
                                 random_state=0)

etr9_1_reg.fit(X_train9_1, y_train9)
```

```
▾                          ExtraTreesRegressor
ExtraTreesRegressor(max_depth=30, max_features='sqrt', n_estimators=200,
                    random_state=0)
```

```
pred_val9 = etr9_1_reg.predict(X_valid9_1)
pred_val9_round = round(pd.DataFrame(pred_val9))
```

```
pred_test9 = etr9_1_reg.predict(X_test9_1)
pred_test9_round = round(pd.DataFrame(pred_test9))

# 제출 형식으로 바꾸기
pred_test_a09 = pd.concat([test9[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_te
pred_test_a09.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a09_202301 = pred_test_a09[pred_test_a09['DATA_CRTR_YM']==202301]
pred_test_a09_202302 = pred_test_a09[pred_test_a09['DATA_CRTR_YM']==202302]
pred_test_a09_202301 = pred_test_a09_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a09_202302 = pred_test_a09_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a09_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a09_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a09_202301['SLS_GRD_2302'] = pred_test_a09_202302['SLS_GRD_2302']
pred_test_a09 = pred_test_a09_202301
pred_test_a09.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

▾ A10

```
# 파라미터 튜닝
'''
# RandomForestRegressor 모델 생성
reg = RandomForestRegressor(random_state=0)

# 탐색할 하이퍼파라미터 그리드 정의
param_grid = {
    'n_estimators': [100, 200, 300],  # 트리 개수
    'max_depth': [None, 10, 20, 30],  # 트리의 최대 깊이
    'min_samples_split': [2, 5, 10],  # 노드 분할 최소 샘플 수
    'min_samples_leaf': [1, 2, 4],  # 리프 노드 최소 샘플 수
    'max_features': ['auto', 'sqrt', 'log2'],  # 최대 특성 개수 설정
}

# Grid Search 객체 생성
grid_search = GridSearchCV(estimator=reg, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)

# Grid Search 수행
grid_search.fit(X_train10_1, y_train10)

# 최적의 하이퍼파라미터 출력
print("최적 하이퍼파라미터:", grid_search.best_params_)

# 최적 모델의 성능 출력
```

```
print("최적 모델의 MSE:", -grid_search.best_score_)  # 음수 MSE 값을 양수로 변환하여 출력
'''
```

```
'\n# RandomForestRegressor 모델 생성\nreg = RandomForestRegressor(random_state=0)\n\n# 탐색할 하이퍼파라미터 그리드 정의\nparam_gri
split\': [2, 5, 10],  # 노드 분할 최소 샘플 수\n     \'min_samples_leaf\': [1, 2, 4],  # 리프 노드 최소 샘플 수\n     \'max_features\
rid=param_grid, scoring=\'neg_mean_squared_error\', cv=5)\n\n# Grid Search 수행\ngrid_search.fit(X_train10_1, y_train10)\r
rch.best_score_)  # 음수 MSE 값을 양수로 변환하여 출력\n'
```

```
rf10_1_reg = RandomForestRegressor(max_depth=None,
                                   max_features= 'sqrt',
                                   min_samples_leaf= 1,
                                   min_samples_split= 2,
                                   n_estimators=300,
                                   random_state=0)

rf10_1_reg.fit(X_train10_1, y_train10)
```

```
▾                         RandomForestRegressor
RandomForestRegressor(max_features='sqrt', n_estimators=300, random_state=0)
```

```
pred_val10 = rf10_1_reg.predict(X_valid10_1)
pred_val10_round = round(pd.DataFrame(pred_val10))
```

```
pred_test10 = rf10_1_reg.predict(X_test10_1)
pred_test10_round = round(pd.DataFrame(pred_test10))

# 제출 형식으로 바꾸기
pred_test_a10 = pd.concat([test10[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_t
pred_test_a10.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a10_202301 = pred_test_a10[pred_test_a10['DATA_CRTR_YM']==202301]
pred_test_a10_202302 = pred_test_a10[pred_test_a10['DATA_CRTR_YM']==202302]
pred_test_a10_202301 = pred_test_a10_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a10_202302 = pred_test_a10_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a10_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a10_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a10_202301['SLS_GRD_2302'] = pred_test_a10_202302['SLS_GRD_2302']
pred_test_a10 = pred_test_a10_202301
pred_test_a10.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

▾ A11

```
# 파라미터 튜닝
'''
# 탐색할 하이퍼파라미터 그리드 정의
param_grid = {
    'n_estimators': [100, 200, 300],  # 트리 개수
    'max_depth': [None, 10, 20, 30],  # 트리의 최대 깊이
    'min_samples_split': [2, 5, 10],  # 노드 분할 최소 샘플 수
    'min_samples_leaf': [1, 2, 4],  # 리프 노드 최소 샘플 수
    'max_features': ['auto', 'sqrt', 'log2'],  # 최대 특성 개수 설정
}

# Grid Search 객체 생성
reg = ExtraTreesRegressor(random_state=0)
grid_search = GridSearchCV(estimator=reg, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)

# Grid Search 수행
grid_search.fit(X_train11_1, y_train11)

# 최적의 하이퍼파라미터 출력
print("최적 하이퍼파라미터:", grid_search.best_params_)

# 최적 모델의 성능 출력
print("최적 모델의 MSE:", -grid_search.best_score_)  # 음수 MSE 값을 양수로 변환하여 출력
'''
```

```
'\n# 탐색할 하이퍼파라미터 그리드 정의\nparam_grid = {\n     \'n_estimators\': [100, 200, 300],  # 트리 개수\n     \'max_depth\': [No
# 리프 노드 최소 샘플 수\n     \'max_features\': [\'auto\', \'sqrt\', \'log2\'],  # 최대 특성 개수 설정\n}\n\n# Grid Search 객체 생성
ared_error\', cv=5)\n\n# Grid Search 수행\ngrid_search.fit(X_train11_1, y_train11)\n\n# 최적의 하이퍼파라미터 출력\nprint("최적 하
환하여 출력\n'
```

```
etr11_1_reg = ExtraTreesRegressor(max_depth=None,
                                  max_features= 'sqrt',
                                  min_samples_leaf= 1,
                                  min_samples_split= 2,
                                  n_estimators=300,
                                  random_state=0)

etr11_1_reg.fit(X_train11_1, y_train11)
```

```
▾                           ExtraTreesRegressor
ExtraTreesRegressor(max_features='sqrt', n_estimators=300, random_state=0)
```

```
pred_val11 = etr11_1_reg.predict(X_valid11_1)
pred_val11_round = round(pd.DataFrame(pred_val11))


pred_test11 = etr11_1_reg.predict(X_test11_1)
pred_test11_round = round(pd.DataFrame(pred_test11))

# 제출 형식으로 바꾸기
pred_test_a11 = pd.concat([test11[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_t
pred_test_a11.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a11_202301 = pred_test_a11[pred_test_a11['DATA_CRTR_YM']==202301]
pred_test_a11_202302 = pred_test_a11[pred_test_a11['DATA_CRTR_YM']==202302]
pred_test_a11_202301 = pred_test_a11_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a11_202302 = pred_test_a11_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a11_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a11_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a11_202301['SLS_GRD_2302'] = pred_test_a11_202302['SLS_GRD_2302']
pred_test_a11 = pred_test_a11_202301
pred_test_a11.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

▾ A12

```
from lightgbm import LGBMRegressor
'''
model12 = LGBMRegressor(random_state =0)
# GridSearchCV를 사용하여 하이퍼파라미터 튜닝 설정
param_grid12 = {
      'learning_rate': [0.1, 0.01],
      'n_estimators': [150,200, 300],
      'max_depth': [8,16,20],
      'colsample_bytree': [0.7,0.8,1.0,1.1],
      'subsample': [0.7,0.8, 0.9],
      'min_child_samples': [1, 5, 10]
}


grid_search12 = GridSearchCV(model12, param_grid12, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search12.fit(X_train12_drop, y_train12)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search12.best_params_)

# 최적 모델 출력
best_model = grid_search12.best_estimator_
print(best_model)
'''
```

```
'\nmodel12 = LGBMRegressor(random_state =0)\n# GridSearchCV를 사용하여 하이퍼파라미터 튜닝 설정\nparam_grid12 = {\n      \'learni
0.8,1.0,1.1],\n      \'subsample\': [0.7,0.8, 0.9],\n      \'min_child_samples\': [1, 5, 10]\n}\n\n\ngrid_search12 = Gri
\n\n# 최적의 하이퍼파라미터 출력\nprint("Best Parameters: ", grid_search12.best_params_)\n\n# 최적 모델 출력\nbest_model = grid_sea
```

```
lgb12 = LGBMRegressor(colsample_bytree = 0.8,learning_rate = 0.1, max_depth = 20,min_child_samples = 10,
                              n_estimators = 300, subsample = 0.7, random_state = 0, n_jobs = -1)
lgb12.fit(X_train12_drop, y_train12)
y_pred12 = lgb12.predict(X_valid12_drop).round(0)
print('MSE of valid12: {:.4f}'.format(mean_squared_error(y_valid12, y_pred12 )))

# 테스트 데이터에 모델 적용 및 평가
y_pred12 =lgb12.predict(X_valid12_drop)
y_pred12 = round(pd.DataFrame(y_pred12))
```

```
mse = mean_squared_error(y_valid12, y_pred12)
print("Mean Squared Error on Test Data: ", mse)
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_le
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_le
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testing was 0.007076 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2555
[LightGBM] [Info] Number of data points in the train set: 22243, number of used features: 20
[LightGBM] [Info] Start training from score 2.999865
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_le
MSE of valid12: 1.0406
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_le
Mean Squared Error on Test Data:  1.0405597241938755
```

```
# 테스트 데이터 예측
pred_test12 = lgb12.predict(X_test12_drop)
pred_test12_round = round(pd.DataFrame(pred_test12))

# 제출 형식으로 바꾸기
pred_test_a12 = pd.concat([test12[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_t
pred_test_a12.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a12_202301 = pred_test_a12[pred_test_a12['DATA_CRTR_YM']==202301]
pred_test_a12_202302 = pred_test_a12[pred_test_a12['DATA_CRTR_YM']==202302]
pred_test_a12_202301 = pred_test_a12_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a12_202302 = pred_test_a12_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a12_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a12_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a12_202301['SLS_GRD_2302'] = pred_test_a12_202302['SLS_GRD_2302']
pred_test_a12 = pred_test_a12_202301
pred_test_a12.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_le
```

A13

```
# 엑스트라 트리 회귀
from sklearn.ensemble import ExtraTreesRegressor
'''
ext_reg = ExtraTreesRegressor(random_state=0)
param_grid13_1 = {
    'n_estimators': [200, 300, 400],
    'max_depth': [None, 30,40,50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search13_1 = GridSearchCV(ext_reg , param_grid13_1, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search13_1.fit(X_train13_drop, y_train13)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search13_1.best_params_)

# 최적 모델 출력
best_model_1 = grid_search13_1.best_estimator_
print(best_model_1)
'''

ext_reg13 = ExtraTreesRegressor(max_depth=30, min_samples_leaf =1, min_samples_split = 2, n_estimators = 400,
                            random_state = 0,n_jobs = -1)
ext_reg13.fit(X_train13_drop, y_train13)
pred_val13_drop1 = ext_reg13.predict(X_valid13_drop).round(0)
print('MSE of valid13: {:.4f}'.format(mean_squared_error(y_valid13, pred_val13_drop1)))
```

```
MSE of valid13: 0.4849
```

```
# 엑스트라트리 분류
from sklearn.ensemble import ExtraTreesClassifier
'''
ext_cls = ExtraTreesClassifier(random_state=0)
param_grid13_2 = {
    'n_estimators': [200, 300, 400],
    'max_depth': [None, 30,40,50],
    'max_depth': [None, 30,40,50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search13_2 = GridSearchCV(ext_cls , param_grid13_2, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search13_2.fit(X_train13_drop, y_train13)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search13_1.best_params_)

# 최적 모델 출력
best_model = grid_search13_1.best_estimator_
print(best_model)
'''

ext_cls13 = ExtraTreesClassifier(max_depth=40, min_samples_leaf =1, min_samples_split = 2,
                               n_estimators = 300, random_state = 0, n_jobs = -1)
ext_cls13.fit(X_train13_drop, y_train13)
pred_val13_drop2 = ext_cls13.predict(X_valid13_drop)
print('MSE of valid13: {:.4f}'.format(mean_squared_error(y_valid13, pred_val13_drop2)))
```

```
    MSE of valid13: 0.4736
```

```
# 테스트 데이터 예측
pred_test13_1 = ext_reg13.predict(X_test13_drop)
pred_test13_2 = ext_cls13.predict(X_test13_drop)
pred_test13_round = round(pd.DataFrame((pred_test13_1+pred_test13_2)/2)) # 결과 앙상블

# 제출 형식으로 바꾸기
pred_test_a13 = pd.concat([test13[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_t
pred_test_a13.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a13_202301 = pred_test_a13[pred_test_a13['DATA_CRTR_YM']==202301]
pred_test_a13_202302 = pred_test_a13[pred_test_a13['DATA_CRTR_YM']==202302]
pred_test_a13_202301 = pred_test_a13_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a13_202302 = pred_test_a13_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a13_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a13_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a13_202301['SLS_GRD_2302'] = pred_test_a13_202302['SLS_GRD_2302']
pred_test_a13 = pred_test_a13_202301
pred_test_a13.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

A14

```
from sklearn.ensemble import ExtraTreesRegressor
'''
model14 = ExtraTreesRegressor(random_state=0)
param_grid14 = {
    'n_estimators': [200, 300, 400],
    'max_depth': [None, 30,40,50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search14 = GridSearchCV(model14 , param_grid14, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search14.fit(X_train14_drop, y_train14)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search14.best_params_)

# 최적 모델 출력
best_model = grid_search14.best_estimator_
print(best_model)
'''
model14 = ExtraTreesRegressor(max_depth=30, min_samples_leaf =1, min_samples_split = 2,
                              n_estimators = 400, random_state = 0, n_jobs = -1)
model14.fit(X_train14_drop, y_train14)
pred_val14_drop = model14.predict(X_valid14_drop).round(0)
print('MSE of valid14: {:.4f}'.format(mean_squared_error(y_valid14, pred_val14_drop)))
```

```
    MSE of valid14: 0.5712
```

```
# 테스트 데이터 예측
pred_test14 = model14.predict(X_test14_drop)
pred_test14_round = round(pd.DataFrame(pred_test14))

# 제출 형식으로 바꾸기
pred_test_a14 = pd.concat([test14[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_t
pred_test_a14.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a14_202301 = pred_test_a14[pred_test_a14['DATA_CRTR_YM']==202301]
pred_test_a14_202302 = pred_test_a14[pred_test_a14['DATA_CRTR_YM']==202302]
pred_test_a14_202301 = pred_test_a14_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a14_202302 = pred_test_a14_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a14_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a14_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a14_202301['SLS_GRD_2302'] = pred_test_a14_202302['SLS_GRD_2302']
pred_test_a14 = pred_test_a14_202301
pred_test_a14.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

	LT_UNQ_NO	INDUSTRY_CD	STDG_EMD_CD	SLS_GRD_2301	SLS_GRD_2302
0	1111010100100720000	A14	11110101	3.0	3.0
1	1111010600100940000	A14	11110106	2.0	2.0
2	1111011400100970000	A14	11110114	3.0	3.0
3	1111011400101050000	A14	11110114	3.0	3.0
4	1111011700100440004	A14	11110117	4.0	4.0

A15

```
# 엑스트라트리 회귀
'''
ext_reg = ExtraTreesRegressor(random_state=0)
param_grid15_1 = {
    'n_estimators': [200, 300, 400],
    'max_depth': [None, 30,40,50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search15_1 = GridSearchCV(ext_reg , param_grid15_1, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search15_1.fit(X_train15_drop, y_train15)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search15_1.best_params_)

# 최적 모델 출력
best_model_1 = grid_search15_1.best_estimator_
print(best_model_1)
'''
ext_reg15 = ExtraTreesRegressor(max_depth=30, min_samples_leaf =1, min_samples_split = 5,
                              n_estimators = 400, random_state = 0, n_jobs = -1)
ext_reg15.fit(X_train15_drop, y_train15)
pred_val15_drop1 = ext_reg15.predict(X_valid15_drop).round(0)

print('MSE of valid15 {:.4f}'.format(mean_squared_error(y_valid15, pred_val15_drop1)))
```

```
MSE of valid15 0.3496
```

```
# 분류 랜덤포레스트 튜닝
from sklearn.ensemble import RandomForestClassifier
'''
rf_cls = RandomForestClassifier(random_state=0)
param_grid15_2 = {
    'n_estimators': [300, 400,500],
    'max_depth': [None, 30,40,50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search15_2 = GridSearchCV(ext_reg , param_grid15_2, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search15_2.fit(X_train15_drop, y_train15)

# 최적의 하이퍼파라미터 출력
print("Best Parameters: ", grid_search15_2.best_params_)

# 최적 모델 출력
best_model_2 = grid_search15_2.best_estimator_
print(best_model_2)
'''
rf_cls15= RandomForestClassifier(max_depth= None, min_samples_leaf =1, min_samples_split = 2,
                               n_estimators = 400, random_state = 0, n_jobs = -1)
rf_cls15.fit(X_train15_drop, y_train15)
```

```
pred_val15_drop2 = rf_cls15.predict(X_valid15_drop)
print('MSE of valid15 {:.4f}'.format(mean_squared_error(y_valid15, pred_val15_drop2)))
```

```
    MSE of valid15 0.3612
```

```
# 테스트 데이터 예측
pred_test15_1 = ext_reg15.predict(X_test15_drop)
pred_test15_2 = rf_cls15.predict(X_test15_drop)
pred_test15_round = round(pd.DataFrame((pred_test15_1+pred_test15_2)/2)) # 결과 앙상블

# 제출 형식으로 바꾸기
pred_test_a15 = pd.concat([test15[['DATA_CRTR_YM', 'LT_UNQ_NO', 'INDUSTRY_CD', 'STDG_EMD_CD']].reset_index(drop=True), pred_t
pred_test_a15.rename(columns={0:'SLS_GRD'}, inplace=True)

pred_test_a15_202301 = pred_test_a15[pred_test_a15['DATA_CRTR_YM']==202301]
pred_test_a15_202302 = pred_test_a15[pred_test_a15['DATA_CRTR_YM']==202302]
pred_test_a15_202301 = pred_test_a15_202301.sort_values(by='LT_UNQ_NO').reset_index(drop=True)
pred_test_a15_202302 = pred_test_a15_202302.sort_values(by='LT_UNQ_NO').reset_index(drop=True)

pred_test_a15_202301.rename(columns={'SLS_GRD':'SLS_GRD_2301'}, inplace=True)
pred_test_a15_202302.rename(columns={'SLS_GRD':'SLS_GRD_2302'}, inplace=True)

pred_test_a15_202301['SLS_GRD_2302'] = pred_test_a15_202302['SLS_GRD_2302']
pred_test_a15 = pred_test_a15_202301
pred_test_a15.drop(columns='DATA_CRTR_YM', axis=1, inplace=True)
```

데이터 합치기

```
'''
pred1 = pred_val1_rgr_round
pred2 = pd.DataFrame(pred_val2)
pred3 = pred_val3_1
pred4 = pd.DataFrame(pred4_e)
pred5 = pd.DataFrame(pred5_round)
pred6 = pd.DataFrame(pred6_e)
pred7 = pd.DataFrame(pred7_e)
pred8 = pred_val8_round
pred9 = pred_val9_round
pred10 = pred_val10_round
pred11 = pred_val11_round
pred12 = y_pred12
pred13 = round(pd.DataFrame((ext_reg13.predict(X_valid13_drop) +ext_cls13.predict(X_valid13_drop) )/2))
pred14 = pd.DataFrame(pred_val14_drop)
pred15 = round(pd.DataFrame((ext_reg15.predict(X_valid15_drop) + rf_cls15.predict(X_valid15_drop))/2))
'''
```

```
    '\npred10 = pred_val10_round\npred11 =pred_val11_round \npred12 = y_pred12\npred13 = round(pd.DataFrame((ext_reg13.predic
    t_reg15.predict(X_valid15_drop) + rf_cls15.predict(X_valid15_drop))/2))\n'
```

```
# validation 예측값 끊어서 저장
'''
pred1_2 = pd.concat([pred1, pred2])
pred3_9 = pd.concat([pred3, pred4, pred5, pred6, pred7, pred8, pred9])
pred10_15 = pd.concat([pred10,pred11,pred12,pred13,pred14,pred15])
'''
```

```
# test 예측값 끊어서 저장
'''
test1_2 = pd.concat([pred_test_a01,pred_test_a02 ])
test3_9 = pd.concat([pred_test_a03, pred_test_a04,pred_test_a05, pred_test_a06, pred_test_a07,pred_test_a08, pred_test_a09])
test10_15 = pd.concat([pred_test_a10,pred_test_a11, pred_test_a12,pred_test_a13,pred_test_a14,pred_test_a15])
'''
```

```
'''
pred1_2.to_csv("업종1_2_validation결과.csv", index=False, encoding="cp949")
pred3_9.to_csv("업종3_9_validation결과.csv", index=False, encoding="cp949")
pred10_15.to_csv("업종10_15_validation결과.csv", index=False, encoding="cp949")

test1_2.to_csv("업종1_2_test결과.csv", index=False, encoding="cp949")
test3_9.to_csv("업종3_9_test결과.csv", index=False, encoding="cp949")
test10_15.to_csv("업종10_15_test결과.csv", index=False, encoding="cp949")
'''
```

Validation MSE

```
pred1_2 = pd.read_csv("업종1_2_validation결과.csv", encoding="cp949")
pred3_9 = pd.read_csv("업종3_9_validation결과.csv", encoding="cp949")
pred10_15 = pd.read_csv("업종10_15_validation결과.csv", encoding="cp949")
```

```
val_pred = pd.concat([pred1_2,pred3_9,pred10_15]).reset_index()
```

```
val_pred.drop(columns='index', inplace=True)
```

```
y_valid = pd.concat([pd.DataFrame(y_valid1),pd.DataFrame(y_valid2), pd.DataFrame(y_valid3), pd.DataFrame(y_valid4), pd.DataFr
           pd.DataFrame(y_valid7), pd.DataFrame(y_valid8), pd.DataFrame(y_valid9),
           pd.DataFrame(y_valid10), pd.DataFrame(y_valid11), pd.DataFrame(y_valid12), pd.DataFrame(y_valid13), pd.DataFrame(y
```

```
y_valid.reset_index(inplace=True)
```

```
y_valid.drop(columns='index', inplace=True)
```

```
mean_squared_error(val_pred['0'], y_valid['SLS_GRD'])

    0.5665165380827315
```

Test Data 생성

```
test1_2 = pd.read_csv("업종1_2_test결과.csv", encoding="cp949")
test3_9 = pd.read_csv("업종3_9_test결과.csv", encoding="cp949")
test10_15 = pd.read_csv("업종10_15_test결과.csv", encoding="cp949")
```

```
test_pred = pd.concat([test1_2,test3_9, test10_15])
test_pred.to_csv("test_pred_최종제출파일.csv", encoding="cp949", index=False)
```