# ===== EDA ======

---------------------------------- 김남이 여기야 ------------------------------

#요일별 중식계(sum) 그룹핑

df1 = train\_x[['요일']]

df2 = df\_train['중식계']

day = pd.concat([df1,df2], axis=1, join='outer')

#day

grouped = day.groupby(day['요일'])

grouped.sum().rename(index={0:'월', 1:'화', 2:'수', 3:'목', 4:'금'})

#---------------------------------피터 여기야----------------------------

train\_tmp = train\_x

train\_tmp["target"] = train\_y

groupby\_dates = train\_tmp.set\_index("일자").resample("1W").mean()["target"]

graph = sns.lineplot(groupby\_dates.index, groupby\_dates.values)

graph.set\_title("시간별 주간평균 중식계 인원")

graph = sns.boxplot(train\_x["요일"], train\_y, palette=sns.hls\_palette())

graph.set\_xticklabels(["월", "화", "수", "목", "금"])

graph.set\_title("요일별 중식계 인원")

graph = sns.boxplot(train\_x["년"], train\_y, palette=sns.hls\_palette())

graph.set\_title("년도별 중식계 인원")

graph = sns.boxplot(train\_x["월"], train\_y, palette=sns.hls\_palette())

graph.set\_title("월별 중식계 인원")

graph = sns.boxplot(train\_x["계절"], train\_y, palette=sns.hls\_palette())

graph.set\_title("신메뉴 여부별 중식계 인원")

# 주가 52주가 되어버림

graph = sns.boxplot(train\_x["주"], train\_y, palette=sns.hls\_palette())

graph.set\_title("주별 중식계 인원")

graph = sns.boxplot(train\_x["공휴일여부"], train\_y, palette=sns.hls\_palette())

graph.set\_title("전후일 공유일 여부별 중식계 인원")

# 주기성 신호로 변환된 변수는 영향 미미

graph = sns.scatterplot(train\_x["frequency\_sin\_year"], train\_y)

graph.set\_title("frequency\_sin\_year")

graph = sns.scatterplot(train\_x["frequency\_cos\_year"], train\_y)

graph.set\_title("frequency\_cos\_year")

graph = sns.regplot(train\_x["식사가용인원"], train\_y, color="green",

scatter\_kws={'s':15}, line\_kws={"color": "orange"})

graph.set\_title("식사가용인원")

graph = sns.boxplot(train\_x["특식"], train\_y, palette=sns.hls\_palette())

graph.set\_title("특식 여부별 중식계 인원")

graph = sns.boxplot(train\_x["신메뉴"], train\_y, palette=sns.hls\_palette())

graph.set\_title("신메뉴 여부별 중식계 인원")

graph = sns.regplot(train\_x["기온"], train\_y, color="green",

scatter\_kws={'s':15}, line\_kws={"color": "orange"})

graph.set\_title("기온별 중식계 인원")

graph = sns.regplot(train\_x["강수량"], train\_y, color="green",

scatter\_kws={'s':15}, line\_kws={"color": "orange"})

graph.set\_title("강수량별 중식계 인원")

graph = sns.regplot(train\_x["강수량"][train\_x["강수량"]>0], train\_y[train\_x["강수량"]>0], color="green",

scatter\_kws={'s':15}, line\_kws={"color": "orange"})

graph.set\_title("강수가 있을 경우 중식계 인원")

graph = sns.boxplot(train\_x["강수여부"], train\_y, palette=sns.hls\_palette())

graph.set\_title("강수 여부별 중식계 인원")

# ===== 모델링 =====

김영준 : RandomForest

김남이 : XGBoost

이예주 : LightGBM

이지예 : CatBoost

1. 전체적인 부분 설명 (Hyper parameter Grid search 진행 후 재학습)
2. 고정 파라미터 설명 (최종 학습시킬 트리 수를 고정 시킨 후 Grid Search 시에는 설정된 트리 수의 일부분만 사용)

# ===== 김영준 RandomForest =====

ntrees = 500

patientRate = 0.2

eta = 0.01

rnd.seed(334)

seed = 9191

tuner\_params = {"num\_leaves": [pow(2, i) - 1 for i in [2, 4, 6, 8]],

"subsample": [0.4, 0.6, 0.8],

"colsample\_bytree": [0.6, 0.8, 1],

"reg\_lambda": list(np.linspace(0.1, 10, 10).round(3))}

lgb\_model = lgb.LGBMRegressor(boosting\_type="rf", objective="regression",

n\_estimators=int(np.floor(ntrees \* patientRate)),

learning\_rate=eta, silent=True, n\_jobs=None,

subsample\_freq=1, random\_state=seed)

model\_tuner = GridTuner(lgb\_model, param\_grid=tuner\_params, cv=10, refit=False,

n\_jobs=multiprocessing.cpu\_count(),

pre\_dispatch=multiprocessing.cpu\_count(),

scoring="neg\_root\_mean\_squared\_error")

model\_tuner.fit(train\_x, train\_y, categorical\_feature=findIdx(train\_x, cat\_vars), verbose=False)

model\_rf = {}

print("Tuning Result --->", model\_tuner.best\_params\_)

model\_rf["best\_params"] = model\_tuner.best\_params\_

lgb\_model = lgb.LGBMRegressor(boosting\_type="rf", objective="regression",

num\_leaves=model\_tuner.best\_params\_["num\_leaves"],

n\_estimators=ntrees, learning\_rate=eta,

n\_jobs=multiprocessing.cpu\_count(), random\_state=seed+9,

reg\_lambda=model\_tuner.best\_params\_["reg\_lambda"],

subsample=model\_tuner.best\_params\_["subsample"],

colsample\_bytree=model\_tuner.best\_params\_["colsample\_bytree"],

subsample\_freq=1, silent=True)

model\_rf["model"] = lgb\_model.fit(train\_x, train\_y, categorical\_feature=findIdx(train\_x, cat\_vars),

eval\_set=[(val\_x, val\_y)], eval\_metric="rmse", verbose=False,

early\_stopping\_rounds=int(np.floor(ntrees \* patientRate)))

model\_rf["pred"] = model\_rf["model"].predict(val\_x)

model\_rf["performance"] = {"RMSE": np.sqrt(metrics.mean\_squared\_error(val\_y, model\_rf["pred"])),

"R2": metrics.r2\_score(val\_y, model\_rf["pred"])}

print(model\_rf["model"].best\_iteration\_)

print(model\_rf["best\_params"])

print(model\_rf["performance"])

best iteration : 79

{'colsample\_bytree': 0.6, 'num\_leaves': 63, 'reg\_lambda': 0.1, 'subsample': 0.8}

{'RMSE': 95.69424449546786, 'R2': 0.7681554859684646}

#=======================[ 남 이 XGBoost ]=======================#

# one hot encoding

oh\_encoder = MyOneHotEncoder()

train\_x\_oh = oh\_encoder.fit\_transform(train\_x, cat\_vars)

import xgboost as xgb

from xgboost import XGBRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV

from xgboost import plot\_importance

import numpy as np

#3차GridSearchCV (이제그만….)

#GridSearch 3차시도

ntrees = 5000

model\_xgb3 = XGBRegressor(booster="gbtree", n\_estimators=int(ntrees\*0.3), objective="reg:squarederror", seed=343)

xgb\_param\_grid = {

'learning\_rate': [0.01,0.05],

'max\_depth': [2,4,6],

'reg\_lambda' : [0.5, 1, 5, 10],

'subsample' : [0.5, 0.6, 0.8]

}

xgb\_grid3 = GridTuner(model\_xgb3, param\_grid=xgb\_param\_grid, scoring='neg\_root\_mean\_squared\_error',

cv=10, n\_jobs=-1, refit=False, verbose=1)

xgb\_grid3.fit(train\_x\_oh, train\_y)

#dp 씌우기

cv\_result = pd.DataFrame(xgb\_grid3.cv\_results\_)

cv\_result.sort\_values(by=['rank\_test\_score'], inplace=True)

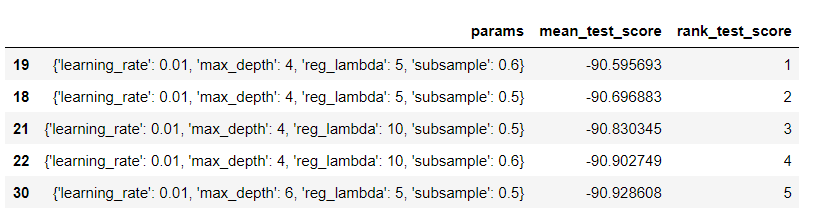
from IPython.core.display import display, HTML

pd.set\_option('display.max.colwidth', 500)

cv2 = cv\_result.loc[:,['params', 'mean\_test\_score','rank\_test\_score']]

cv2.head()

#결과



# train의 20% 를 validation 으로 분리

train\_x, val\_x, train\_x\_oh, val\_x\_oh, train\_y, val\_y = tts(train\_x, train\_x\_oh, train\_y, test\_size=0.2, random\_state=777)

xgb4 = XGBRegressor(n\_estimators=5000, learning\_rate=0.01, max\_depth=4, reg\_lambda= 5, subsample=0.5, objective="reg:squarederror",colsample\_bytree=0.8,seed=343)

evals=[(val\_x\_oh, val\_y)]

xgb4.fit(train\_x\_oh, train\_y, early\_stopping\_rounds=500,eval\_metric='rmse', eval\_set=evals, verbose=1)

from sklearn.preprocessing import OneHotEncoder

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import r2\_score

xgb4\_pred = xgb4.predict(val\_x\_oh)

xgb4\_rmse = mean\_squared\_error(val\_y, xgb4\_pred)

xgb4\_r2 = r2\_score(val\_y, xgb4\_pred)

print('Mean squared error: ', np.sqrt(xgb4\_rmse))

print('R2 score: ', xgb4\_r2)

{'learning\_rate': 0.01, 'max\_depth': 4, 'reg\_lambda': 5, 'subsample': 0.6}

**#best iteration**

[2478] validation\_0-rmse:80.79243

**#결과**

Mean squared error: 80.1128598973427

R2 score: 0.8394807725314319

**####------------------------[ 이지예의 Catboost ] -------------------------####**

**### 1. CatBoost 최적 하이퍼 파라미터 찾기**

import time

import numpy as np

import catboost as cb

from catboost import CatBoostRegressor

from sklearn.model\_selection import train\_test\_split as tts

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_squared\_error, r2\_score

from math import sqrt

ntrees = 3000

cb = cb.CatBoostRegressor(random\_state=11, n\_estimators=int(ntrees\*0.2), loss\_function = 'RMSE' )

from sklearn.model\_selection import GridSearchCV

param = {

'learning\_rate' : [0.05, 0.06, 0.1],

'max\_depth' : [2,5,8],

'l2\_leaf\_reg' : [0,3,5,10]

}

grid\_cv = GridSearchCV(cb, param\_grid=param, scoring='neg\_root\_mean\_squared\_error', cv=10, verbose=1, n\_jobs=-1)

grid\_cv.fit(train\_x\_oh, train\_y)

print('최적 하이퍼 파라미터: \n', grid\_cv.best\_params\_)

print('최고 예측 정확도(RMSE의 -값): {0:.4f}'.format(grid\_cv.best\_score\_))

**##================> [결과값]**

**최적 하이퍼 파라미터:**

**{'l2\_leaf\_reg': 3, 'learning\_rate': 0.06, 'max\_depth': 5}**

**최고 예측 정확도(RMSE의 -값): -79.7086**

**### 2. 최적 하이퍼파라미터에 적용시키기**

ntrees = 3000

cb1 = cb.CatBoostRegressor(l2\_leaf\_reg = 3,learning\_rate = 0.06, n\_estimators = ntrees, max\_depth=5, boosting\_type='Plain', early\_stopping\_rounds=500, use\_best\_model=True, loss\_function = 'RMSE') # 최적 하이퍼파라미터에 적용한 후 다시 학습시키기

cb1\_model = cb1.fit(train\_x\_oh, train\_y, eval\_set=[(val\_x\_oh, val\_y)])

# GridSearchCV를 이용해 최적으로 학습된 estimators로 예측 수행

cb1\_model\_predict = cb1\_model.predict(val\_x\_oh)

start = time.time()

print("Time:%.1f" % (time.time() - start), "seconds") # 코드 실행 시간 계산

print("RMSE: {:.3f}".format(sqrt(mean\_squared\_error(val\_y,cb1\_model\_predict))))

print("R2: {:.3f}".format(r2\_score(val\_y,cb1\_model\_predict)))

**##================> [결과값]**

**bestTest = 78.74562259**

**bestIteration = 535**

**Shrink model to first 536 iterations.**

**Time:0.0 seconds**

**RMSE: 78.746**

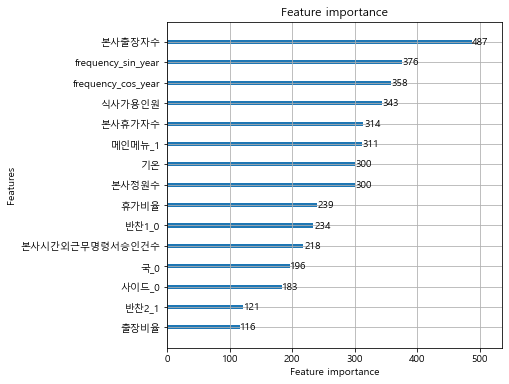
**R2: 0.843**

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<예주zone>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

#========================이예주 LightGBM==============================#

| #최적 파라미터 찾기(5000)  ntrees = 5000  model\_lgb = LGBMRegressor(boosting="goss", n\_estimators=int(ntrees\*0.2), objective="regression", seed=525)  lgb\_param\_grid = {  'learning\_rate': [0.01,0.05,0.1],  'num\_leaves': [3,7,15,31],  'reg\_lambda' : [0.1, 1, 10],  'subsample' : [0.5,0.6,0.7]  }  lgb\_grid = GridSearchCV(model\_lgb, param\_grid=lgb\_param\_grid,  scoring='neg\_root\_mean\_squared\_error',  cv=10, n\_jobs=-1, refit=False, verbose=1)  lgb\_grid.fit(train\_x\_oh, train\_y)  print("최적 하이퍼 파라미터:" , lgb\_grid.best\_params\_) |
| --- |
| Fitting 10 folds for each of 108 candidates, totalling 1080 fits  최적 하이퍼 파라미터: {'learning\_rate': 0.01, 'num\_leaves': 7, 'reg\_lambda': 1, 'subsample': 0.5} |

| #학습  start = time.time()  lgb = LGBMRegressor(boosting="goss", n\_estimators = ntrees,  objective="regression", seed=525,  learning\_rate = 0.01, num\_leaves = 7,  reg\_lambda= 1,subsample= 0.5)  evals = [(X\_test, y\_test)]  lgb.fit(X\_train, y\_train, early\_stopping\_rounds = 100, eval\_metric='rmse', eval\_set=evals, verbose=True)  lgb\_pred = lgb.predict(X\_test)  from math import sqrt  lgb\_rmse = sqrt(mean\_squared\_error(y\_test, lgb\_pred))  lgb\_r2 = r2\_score(y\_test, lgb\_pred)  print('Mean squared error: ', lgb\_rmse)  print('R2 score: ', lgb\_r2) |
| --- |
| Early stopping, best iteration is:  [912] valid\_0's rmse: 79.3972 valid\_0's l2: 6303.91  Mean squared error: 79.39717382358009  R2 score: 0.8373379845788484 |



#---------------- Stacked Ensemble----------------

# RandomForest 제외 (성능 문제)

rnd.seed(1234)

stacking\_base\_models = [

("XGBoost", xgb.XGBRegressor(booster="gbtree", objective="reg:squarederror",

n\_estimators=2478, max\_depth=4,

subsample=0.6, colsample\_bytree=0.8,

reg\_lambda=5, learning\_rate=0.01,

verbosity=0, random\_state=777)),

("LightGBM", lgb.LGBMRegressor(boosting\_type="goss", objective="regression",

n\_estimators=912, num\_leaves=2\*\*3-1,

subsample=0.5, colsample\_bytree=0.8,

reg\_lambda=11, learning\_rate=0.01,

silent=True, random\_state=777)),

("CatBoost", cat.CatBoostRegressor(boosting\_type='Plain', loss\_function='RMSE',

n\_estimators=535, max\_depth=5,

rsm=0.8, # rsm = colsample\_bytree

l2\_leaf\_reg=3, learning\_rate=0.06,

silent=True, random\_seed=777))

]

meta\_learner\_model = lm.ElasticNetCV(l1\_ratio=[.1, .3, .5, .6, .7, .75, .8, .85, .9, .95, .99, 1], n\_alphas=1000, random\_state=777)

{'RMSE': 77.21209404644407, 'R2': 0.8712949169257178}

meta\_learner\_model = lm.LinearRegression()

{'RMSE': 76.8885220689633, 'R2': 0.8723713829212315}

result\_se = {}

result\_se["model"] = ensemble.StackingRegressor(estimators=stacking\_base\_models,

final\_estimator=meta\_learner\_model,

cv=10,

n\_jobs=multiprocessing.cpu\_count())

result\_se["model"].fit(train\_x\_oh, train\_y)

result\_se["pred"] = result\_se["model"].predict(val\_x\_oh)

result\_se["performance"] = {"RMSE": np.sqrt(metrics.mean\_squared\_error(val\_y, result\_se["pred"])),

"R2": metrics.r2\_score(val\_y, result\_se["pred"])}

print(result\_se["performance"])