# 중식계 컬럼 정리

년 category

월 category

주 category

요일 category

공휴일전후 binary

frequency\_sin\_year numeric < np.sin((2 \* np.pi) \* 현재날짜\_to초/하루\_to초) >

frequency\_cos\_year numeric < np.cos((2 \* np.pi) \* 현재날짜\_to초/하루\_to초) >

# 년월일주, 계절

# 1. 일자 관련 컬럼

train\_x['년'] = train\_x['일자'].dt.year

train\_x['월'] = train\_x['일자'].dt.month

train\_x['일'] = train\_x['일자'].dt.day

train\_x['주'] = train\_x['일자'].dt.isocalendar().week

train\_x['요일'] = train\_x['일자'].dt.weekday

# 계절

season = []

for i in train\_x['월']:

if i in [3,4,5]:

season.append(0)

elif i in [6,7,8]:

season.append(1)

elif i in [9,10,11]:

season.append(2)

else:

season.append(3)

train\_x['계절'] = season

weekofmonth = []

for i in train\_x["주"]:

weekofmonth.append((i-1) % 4)

train\_x["before\_2020"] = [1 if i >= 2020 else 0 for i in train\_x['년']]

**# ===== 공휴일전후여부 컬럼추가 (공휴일전후:1 / 공휴일전후아님:0) =====**

a = []

for n in range(len(df\_train)):

if n==0: #첫행은 앞의 행이 없으므로, ‘뒷행-1’값이랑만 일치하면 평일

if (df\_train.iloc[n,1]) == (df\_train.iloc[n+1,1]-1):

a.append(0)

else:

a.append(1)

elif n==len(df\_train)-1: #마지막행은 뒷 행 없음, ‘앞행+1’ 값만 일치하면 평일

if (df\_train.iloc[n,1] == (df\_train.iloc[n-1,1]+1)):

a.append(0)

else:

a.append(1)

elif df\_train.iloc[n,1] == 0: #월요일(0)의 경우, ‘뒷행-1’값은 0, ‘앞행+1’값은 5여야 평일

if (df\_train.iloc[n+1,1]-1 == 0) and (df\_train.iloc[n-1,1]+1 == 5):

a.append(0)

else:

a.append(1)

elif df\_train.iloc[n,1] == 4: #금요일(4)의 경우, ‘앞행+1’값은 4, ‘뒷행-1’ 값은 -1이어야 평일

if (df\_train.iloc[n+1,1] -1 == -1) and (df\_train.iloc[n-1,1]+1 == 4):

a.append(0)

else:

a.append(1)

elif ((df\_train.iloc[n,1] == (df\_train.iloc[n-1,1]+1)) and (df\_train.iloc[n,1]) == (df\_train.iloc[n+1,1]-1)): #월,금 아니면(1,2,3) 앞행+1값과 뒷행-1값은 모두 일치해야 평일.

a.append(0)

else:

a.append(1)

a

df\_train["공휴일여부"] = a

train\_ln = df\_train.copy()

train\_ln.head(10)

▲

**# 주기성 파생변수 생성**

time\_zero = datetime(1970, 1, 1, 0, 0, 0)

day\_to\_sec = 24\*60\*60

year\_to\_sec = (365.2425)\*day\_to\_sec

frequency\_sin\_year = []

frequency\_cos\_year = []

for i in train\_x["일자"]:

time\_to\_sec = i.to\_pydatetime()

time\_interval = (time\_to\_sec - time\_zero).total\_seconds()

frequency\_sin\_year.append(np.sin((time\_interval / year\_to\_sec) \* 2 \* np.pi))

frequency\_cos\_year.append(np.cos((time\_interval / year\_to\_sec) \* 2 \* np.pi))

train\_x["frequency\_sin\_year"] = frequency\_sin\_year

train\_x["frequency\_cos\_year"] = frequency\_cos\_year

# train\_x["frequency\_sin\_year"].plot()

# train\_x["frequency\_cos\_year"].plot()

본사정원수 numeric

본사휴가자수 numeric

본사출장자수 numeric

본사시간외근무명령서승인건수 numeric

현본사소속재택근무자수 numeric

식사가용인원 numeric (본사정원수-(본사휴가자수+본사출장자수+현본사소속재택근무자수)

야근비율 numeric (본사시간외근무명령서승인건수 / 식사가용인원)

휴가비율 numeric (본사휴가자수 / 본사정원수)

출장비율 numeric (본사출장자수 / 본사정원수)

재택비율 numeric (현본사소속재택근무자수 / 본사정원수)

▲**# 식사 가용인원 파생변수 생성**

df\_train['식사가용인원'] =df\_train['본사정원수']-(df\_train['본사휴가자수']+df\_train['본사출장자수']+df\_train['현본사소속재택근무자수']).astype(int)

df\_train['야근비율'] = round(df\_train['본사시간외근무명령서승인건수'] / df\_train['식사가용인원'],3).astype(float)

df\_train['휴가비율'] = round(df\_train['본사휴가자수'] / df\_train['본사정원수'],3).astype(float)

df\_train['출장비율'] = round(df\_train['본사출장자수'] / df\_train['본사정원수'],3).astype(float)

df\_train['재택비율'] = round(df\_train['현본사소속재택근무자수'] / df\_train['본사정원수'],3).astype(float)

df\_train.head()

(중식계만)

밥 category

국 category

메인메뉴 category

특식 binary

신메뉴 binary

반찬1 category

반찬2 category

김치 category

사이드 category

df\_train = read\_csv("./2차프로젝트/original/train.csv", parse\_dates=["일자"])

**#**

df\_train['년'] = df\_train['일자'].dt.year

df\_train['월'] = df\_train['일자'].dt.month

df\_train['일'] = df\_train['일자'].dt.day

df\_train['주'] = df\_train['일자'].dt.week

df\_train['요일'] = df\_train['일자'].dt.weekday

**# ===== 메뉴 embedding =====**

# 일별 점심메뉴를 작은 리스트로 갖고 있는 2중 리스트 (lunch\_train) 만들기

lunch\_train = []

for day in range(len(df\_train)):

tmp = df\_train.loc[day, "중식메뉴"].split(' ') # 공백으로 문자열 구분

print(tmp)

tmp = ' '.join(tmp).split() # 빈 원소 삭제

print(tmp)

search = '(' # 원산지 정보는 삭제

for menu in tmp:

if search in menu:

tmp.remove(menu)

lunch\_train.append(tmp)

lunch\_train # 데이터 확인

# lunch train data에 메뉴명별 칼럼 만들기 (밥, 국, 반찬1-3)

menu\_len\_list = []

bob = []; gook = []; jm = []; side1 = []; side2 = []; kimchi = []; dessert = [];

for i, day\_menu in enumerate(lunch\_train):

bob\_tmp = day\_menu[0]; bob.append(bob\_tmp)

gook\_tmp = day\_menu[1]; gook.append(gook\_tmp)

jm\_tmp = day\_menu[2]; jm.append(jm\_tmp)

side1\_tmp = day\_menu[3]; side1.append(side1\_tmp)

side2\_tmp = day\_menu[4]; side2.append(side2\_tmp)

if i < 1067:

kimchi\_tmp = day\_menu[-1]; kimchi.append(kimchi\_tmp)

dessert\_tmp = day\_menu[-2]; dessert.append(dessert\_tmp)

else:

kimchi\_tmp = day\_menu[-2]; kimchi.append(kimchi\_tmp)

dessert\_tmp = day\_menu[-1]; dessert.append(dessert\_tmp)

menu\_len\_list.append([len(day\_menu),i])

menu\_len\_list # 데이터 확인

train\_ln = df\_train.copy()

train\_ln['밥'] = bob

train\_ln['국'] = gook

train\_ln['메인메뉴'] = jm; train\_ln['반찬1'] = side1; train\_ln['반찬2'] = side2

train\_ln['김치'] = kimchi

train\_ln['사이드'] = dessert

train\_ln.info()

train\_ln.head(10)

**#========특식 컬럼 추가 (쌀밥이 아닌 비빔밥 등의 특식 메뉴 제공)============**

jmt = []; #특식 컬럼 변수명 지정

for i in train\_ln['밥']:

if '쌀밥' not in i:

jmt.append(1) #’쌀밥’이라는 단어가 ‘밥’ 컬럼에 있는 경우 1을 반환

else:

jmt.append(0) #있는 경우 0을 반환

#print(jmt)

train\_ln['특식'] = jmt #따라서 특식에 쌀밥이 아닌 다른 밥메뉴들이 들어갈 경우 1을 반환

**#========신메뉴 컬럼 추가 (신메뉴가 수요에 미치는 영향을 파악) (중식)===========**

new = [];

for i in range(len(train\_ln)): #행 길이 만큼 반복

if 'New' in train\_ln.loc[i, '중식메뉴']: #중식메뉴 컬럼에 ‘New’라는 문자열이 있으면

new.append(1) #1을 반환

else:

new.append(0) #아님 0을 반환

train\_ln['신메뉴'] = new

# 기상청 외부데이터 활용

# 인자 -> data\_x : iterable, col\_names: 찾을 리스트

# 리턴 -> 인덱스 위치를 담은 리스트

# col\_names 를 인자로 받아서 data\_x 에 어디 있는지 인덱스를 찾습니다

def findIdx(data\_x, col\_names):

return [int(i) for i, j in enumerate(data\_x) if j in col\_names]

# 기상청 외부데이터 활용

# 점심 11,12,13 저녁 17,18,19

forecast = read\_csv("./2차프로젝트/2016\_2021\_진주\_기온강수.csv", encoding="euc-kr")

forecast["강수량"].fillna(0, inplace=True)

forecast.isna().sum()

findIdx(forecast["기온"].isna(), [True])

forecast["기온"][40765:40768]

forecast["기온"][40766] = forecast["기온"][40765:40768].mean()

forecast["기온"][40768:40771]

forecast["기온"][40769] = forecast["기온"][40768:40771].mean()

forecast.isna().sum().sum()

train\_x.isna().sum().sum()

# forecast

tmp\_list = []

for i in forecast["일시"]:

if ("11:00" in i) or ("12:00" in i) or ("13:00" in i):

tmp\_list.append(True)

else:

tmp\_list.append(False)

forecast = forecast[tmp\_list]

forecast["일시"] = pd.to\_datetime(forecast["일시"])

forecast.set\_index("일시", inplace=True)

forecast.isna().sum().sum()

forecast.resample("1D").mean().isna().sum()

# 10:16 사이에 데이터 없음

# 8:15 사이에 데이터 없음

# round(forecast.resample("1D").mean()["기온"], 1)

# forecast.resample("1D").max()["강수량"]

forecast\_new = pd.concat([round(forecast.resample("1D").mean()["기온"], 1), forecast.resample("1D").max()["강수량"]], axis=1)

forecast\_new.isna().sum()

forecast\_new.reset\_index("일시", inplace=True)

forecast\_new.columns = ["일자", "기온", "강수량"]

# train\_x = pd.merge(train\_x, forecast\_new, left\_on="일자")

train\_x = pd.merge(train\_x, forecast\_new, how="left", on="일자")

train\_x.isna().sum().sum()

# 기온 및 강수량의 11~13 시 결측치는 전날 기온 및 강수량으로 대체하였다

**train\_x["기온"][findIdx(train\_x["기온"].isna(), [True])[0]] = train\_x["기온"][findIdx(train\_x["기온"].isna(), [True])[0]-1]**

**train\_x["강수량"][findIdx(train\_x["강수량"].isna(), [True])[0]] = 0**

train\_x.isna().sum().sum()

train\_x["강수여부"] = [1 if i>0 else 0 for i in train\_x["강수량"]]

# ===== pickle 로 데이터 저장 =====

# pickle 로 파이썬 객체(리스트나 딕셔너리 등 근대 tensorflow 모델은 안 됨) 쉽게 저장하기

# x: 저장하고 싶은 객체, path: 저장경로 (예: 2차프로젝트/abc.pickle <- 확장자필수)

# op: operation write or read 수행하고싶은 명령 (w or r 두가지 중 하나만 입력)

def easyIO(x=None, path=None, op="r"):

tmp = None

# op=”r” 이면 read 오퍼레이션

if op == "r":

# 파일을 binary 로 열어 저장합니다

with open(path, "rb") as f:

tmp = pickle.load(f)

return tmp

# op=”w” 면 write 오퍼레이션

elif op == "w":

tmp = {}

# 데이터 저장 전 마지막으로 확인을 위한 print

print(x)

# 여기 If 문은 딱히 신경쓸 필요 없음

if type(x) is dict:

for k in x.keys():

if "MLP" in k:

tmp[k] = {}

for model\_comps in x[k].keys():

if model\_comps != "model":

tmp[k][model\_comps] = x[k][model\_comps]

print(F"INFO : {k} model is removed (keras)")

else:

tmp[k] = x[k]

# 키보드로 y를 입력 시 저장, 이외의 값을 입력 시 저장 안 함

if input("Write [y / n]: ") == "y":

with open(path, "wb") as f:

pickle.dump(tmp, f)

print("operation success")

else:

print("operation fail")

else:

print("Unknown operation type")

# easyIO(train\_x, "./2차프로젝트/ds\_bfEDA.pickle", op="w")

#train\_x = easyIO(None, "./2차프로젝트/ds\_bfEDA.pickle", op="r")

#---------------------------------지예야 여기야----------------------------

기다려봐~웅~

여기~감사링~딩동링딩동링디기딩디기딩딩딩

##-------------시각화 [요일별 식당 이용률 (가용인원 중에 중식이용을 한 비율)] : 어느 요일에 이용률이 높은지------------------##

##

a1 = pd.concat([train\_x, train\_y],axis=1)

a2 = round(a1.groupby(['요일']).mean(),2)

a2

a2["중식이용비"] = a2["중식계"]/a2["식사가용인원"]

a2

## 요일 인덱스를 일반 컬럼으로 변경

a2.reset\_index('요일',inplace=True)

a2

## 한글폰트 불러오기

import matplotlib as mpl

font\_name = mpl.font\_manager.FontProperties(fname='C:/Windows/Fonts/malgun.ttf').get\_name()

mpl.rc('font', family=font\_name)

yoil = a2['요일']

values = a2['중식이용비']

plt.bar(yoil, values, color=['tomato', 'silver', 'silver','silver',\

'royalblue'], edgecolor='white', linewidth=5, tick\_label=['월','화','수','목','금'])

title\_font = {'fontsize' : 20,

'fontweight' : 'bold'}

title = plt.title('각 요일별 직원 중식이용비율', loc='center', pad=20, fontdict = title\_font)

plt.show()

---------------------------------- 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")

# ===== 9/14 Feature Importance Summary =====

menu\_vec = ["밥", "국", "메인메뉴", "반찬1", "반찬2", "김치", "사이드"]

menu\_cols = [i for i in train\_x if i.split("\_")[0] in menu\_vec]

menu\_idx = findIdx(train\_x, menu\_cols)

menu\_mean = dataframe()

for i in range(len(menu\_vec)):

menu\_mean = pd.concat([menu\_mean, dataframe(np.mean(array(train\_x.iloc[:,menu\_idx[i:(i+4)]]), axis=1))], axis=1)

menu\_mean.columns = menu\_vec

data\_fi = pd.concat([train\_x.drop(menu\_cols, axis=1), menu\_mean], axis=1)

# 여기부분만 다른 모델을 사용하면

# 그 모델로 산출한 변수의 중요도를 파악할 수 있음

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

n\_estimators=1000, num\_leaves=2\*\*6-1, learning\_rate=0.01,

n\_jobs=multiprocessing.cpu\_count(), random\_state=6767,

subsample=0.8, silent=True)

model\_fi.fit(data\_fi, train\_y, categorical\_feature=cat\_vars)

lgb.plot\_importance(model\_fi, max\_num\_features=10)

# ===== 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 number of trees : 79

best hyper parameters :

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

performance :

{'RMSE': 95.02388741903363, 'R2': 0.7713923419027696}

#=======================[ 남 이 Z O N E ]=======================#

#-----------------------XGBoost 최적 파라미터값 찾기(GridSearchCV 사용)------

#3차GridSearchCV (이제그만….)

#GridSearch 3차시도

ntrees = 5000 #트리개수 조정 짱많이

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]

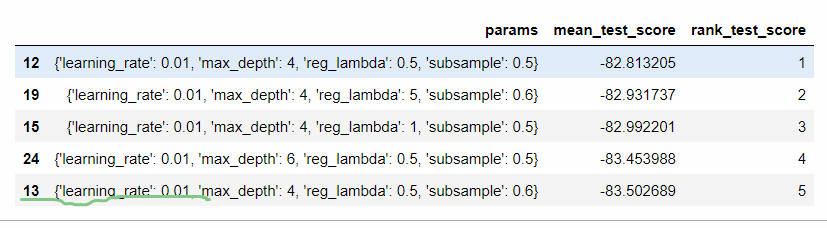
}

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)

>>최적의 하이퍼파라미터 (결과)



#파라미터에 colsample\_bytree=0.8 추가, 빨간 글씨는 최적 파라미터 값

xgb4 = XGBRegressor(n\_estimators=5000, **learning\_rate=0.01, max\_depth=4, reg\_lambda= 0.5, subsample=0.5**, objective="reg:squarederror",colsample\_bytree=0.8,seed=444)

evals=[(X\_test, y\_test)]

xgb4.fit(X\_train, y\_train, 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(X\_test)

xgb4\_rmse = mean\_squared\_error(y\_test, xgb4\_pred)

xgb4\_r2 = r2\_score(y\_test, xgb4\_pred)

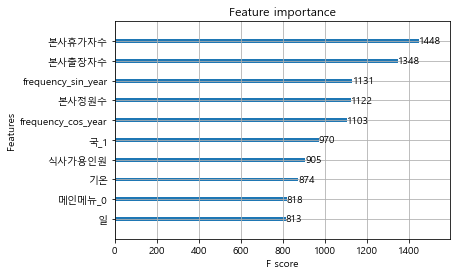
print('Mean squared error: ', np.sqrt(xgb4\_rmse))

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

#결과

* [1144] validation\_0-rmse:82.42786
* RMSE: 82.42786061464622
* R2 score: **0.8277243056124098**

#째금 높아짐



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

######## CatBoost

### 1. 연습용

import time

import numpy as np

import catboost as cb

from catboost import CatBoostRegressor

start = time.time()

cb\_dtrain = cb.Pool(data = train\_x\_oh, label = train\_y) #학습 데이터를 Catboost 모델에 맞게 변환

cb\_param = {'max\_depth':5, #트리 깊이

'learning\_rate':0.01, # Step Size

'n\_estimators':100, # Number of trees, 트리 생성 개수

'eval\_metric':'RMSE', # 평가 척도

'loss\_function':'RMSE'} #손실함수

cb\_model = cb.train(pool = cb\_dtrain, params = cb\_param) #학습 진행

cb\_model\_predict = cb\_model.predict(val\_x\_oh) #평가 데이터 예측

print("Time:%.2f" % (time.time() - start), "seconds") # 코드 실행 시간 계산

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

from math import sqrt

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

### 2. 본격) CatBoost 최적 하이퍼 파라미터 찾기

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

#학습셋과 테스트 셋 구분, shape 확인

X\_train, X\_test, y\_train, y\_test = tts(train\_x\_oh, train\_y, test\_size=0.2, random\_state=11)

print(X\_train.shape, X\_test.shape)

ntrees = 3000

#예측값확인

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(X\_train, y\_train)

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

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

**##ㄴ----------> [결과값]**

#최적 하이퍼 파라미터:

# {'l2\_leaf\_reg': 0, 'learning\_rate': 0.06, 'max\_depth': 5}

#최고 예측 정확도(RMSE의 -값): -79.7502

### 3) 최적 하이퍼파라미터에 적용시키기

import time

import numpy as np

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

from sklearn.metrics import accuracy\_score

from math import sqrt

import catboost as cb

from catboost import CatBoostRegressor

cb1 = cb.CatBoostRegressor(l2\_leaf\_reg = 0,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(X\_train, y\_train, eval\_set=[(X\_test, y\_test)])

# GridSearchCV를 이용해 최적으로 학습된 estimators로 예측 수행

cb1\_model\_predict = cb1\_model.predict(X\_test)

start = time.time()

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

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

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

**##ㄴ------->[결과값]**

#bestTest = 84.85323095

#bestIteration = 1185

#Shrink model to first 1186 iterations.

#Time:0.0 seconds

#RMSE: 84.853

#R2: 0.829

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

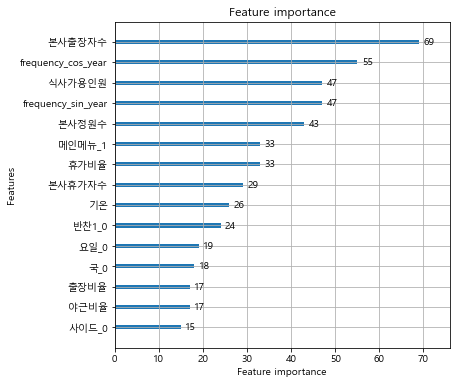
#========================이예주 LightGBM==============================#

| #최적파라미터찾기 (2000)  ntrees = 2000  model\_lgb = LGBMRegressor(boosting="goss", n\_estimators=int(ntrees\*0.2), objective="regression", seed=525)  lgb\_param\_grid = {  'learning\_rate': [0.01,0.05],  '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\_) |
| --- |
| #결과  Fitting 10 folds for each of 72 candidates, totalling 720 fits  최적 하이퍼 파라미터: {'learning\_rate': 0.05, 'num\_leaves': 7, 'reg\_lambda': 0.1, 'subsample': 0.5} |

| pd.set\_option('display.max.colwidth', 300) #셀크기 조정  cv\_result2 = pd.DataFrame(lgb\_grid.cv\_results\_)  cv\_result2.sort\_values(by=['rank\_test\_score'], inplace=True)  cv2 = cv\_result2.loc[:,['params', 'mean\_test\_score','rank\_test\_score']]  cv2.head() |
| --- |
|  |

| #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': 15, 'reg\_lambda': 0.1, 'subsample': 0.5} |

| #학습  start = time.time()  lgb = LGBMRegressor(boosting="goss", n\_estimators = ntrees,  objective="regression", seed=525,  learning\_rate = 0.01, num\_leaves = 15,  reg\_lambda= 0.1,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:  [329] valid\_0's rmse: 90.5191 valid\_0's l2: 8193.71  Mean squared error: 90.51911534354515  R2 score: 0.8052377660686656 |



| B1= layers.Dense(2\*\*11,activation="relu")(B0)  B2= layers.Dense(2\*\*10, activation="relu")(B1)  B3= layers.Dense(2\*\*9, activation="relu")(B2)  B4= layers.Dense(2\*\*8, activation="relu")(B3)  B5= layers.Dense(2\*\*7, activation="relu")(B4)  B6= layers.Dense(2\*\*6, activation="relu")(B5)  B7= layers.Dense(2\*\*5, activation="relu")(B6) |
| --- |
| Mean squared error: 83.50075871854453  R2 score: 0.8342685275680901 |

# ===== 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"])

# ----- 9/14 딥러닝 모델 만들기 예시 -----

# 인풋 레이어를 만듭니다

B0 = layers.Input(feature 갯수 입력)

# 학습시킬 Dense layer를 만듭니다. (자유는 구조)

# 32 개의 뉴런들이 다른 가중치를 가지고 선형 결합됩니다.

B1 = layers.Dense(32, activation="relu")(B0)

# 회귀 or 분류 레이어를 만듭니다.

layer\_regressor = layers.Dense(1)(B1)

# input layer + output layer 를 인자로 넣어 모델 생성.

model\_mlp = keras.Model(B0, layer\_regressor)

# 생성한 모델을 Compile 합니다.

model\_mlp.compile(loss=”mse”, optimizer=”adam”)

# ===== 딥러닝 모델 ======

# ===== 김영준 =====

# ===== 김영준 MLP =====

def createMLP():

hiddenLayers = 1024

dropoutRate = 0.2

B0\_input = layers.Input(shape=train\_x\_oh.shape[1], name="B0\_input")

B0\_embedding = layers.Dense(hiddenLayers \* 2, activation="relu",

kernel\_regularizer="l2", name="B0\_embedding")(B0\_input)

B1\_dense = layers.Dense(hiddenLayers, activation="relu", name="B1\_dense")(B0\_embedding)

B1\_dropout = layers.Dropout(rate=dropoutRate, name="B1\_dropout")(B1\_dense)

B1\_concat1 = layers.concatenate([B0\_embedding, B1\_dropout], name="B1\_concat1")

B2\_dense = layers.Dense(hiddenLayers, activation="relu", name="B2\_dense")(B1\_concat1)

B2\_dropout = layers.Dropout(dropoutRate, name="B2\_dropout")(B2\_dense)

B2\_concat2 = layers.concatenate([B0\_embedding, B1\_dropout, B2\_dropout], name="B2\_concat2")

B3\_dense = layers.Dense(hiddenLayers, activation="relu", name="B3\_dense")(B2\_concat2)

B3\_dropout = layers.Dropout(dropoutRate, name="B3\_dropout")(B3\_dense)

layer\_final = layers.Dense(int(hiddenLayers/2), activation="relu", name="layer\_final")(B3\_dropout)

layer\_regressor = layers.Dense(1, name="Regressor")(layer\_final)

model\_mlp = Model(B0\_input, layer\_regressor)

model\_mlp.compile(loss="mse", optimizer=optimizers.Adam(3e-3),

metrics=tf\_metrics.RootMeanSquaredError(name="rmse"))

# model\_mlp.summary()

return model\_mlp

tf.random.set\_seed(54321)

epochs = 200

batch\_size = 8

patientRate = 0.2

cb\_earlystopping = tf\_callbacks.EarlyStopping(patience=int(np.floor(epochs \* patientRate)), restore\_best\_weights=True)

cb\_reduceLR = tf\_callbacks.ReduceLROnPlateau(patience=int(np.floor(epochs \* (patientRate \*\* 2))), factor=0.8, min\_lr=1e-4)

cb\_lists = [cb\_earlystopping, cb\_reduceLR, TqdmCallback(verbose=0)]

model\_mlp = {}

model\_mlp['model'] = createMLP()

model\_mlp['model'].summary()

model\_mlp['model'].fit(x=train\_x\_oh, y=train\_y,

epochs=epochs, batch\_size=batch\_size,

validation\_data=(val\_x\_oh, val\_y),

verbose=0, shuffle=False,

callbacks=cb\_lists)

model\_mlp["pred"] = model\_mlp["model"].predict(val\_x\_oh, batch\_size=batch\_size).flatten()

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

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

print(model\_mlp["performance"])

performance :

{'RMSE': 95.02388741903363, 'R2': 0.7713923419027696}