# Importing

import multiprocessing

import os

import copy

import pickle

from datetime import datetime

from time import time

from matplotlib import font\_manager as fm, rc

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from numpy import array, nan, random as rnd, where as which

import pandas as pd

from pandas import DataFrame as dataframe, Series as series, isna, isnull, read\_csv

# from sklearn import datasets

from sklearn import preprocessing as prep

from sklearn.impute import KNNImputer

from sklearn.ensemble import IsolationForest

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

from sklearn.model\_selection import StratifiedKFold, KFold

from sklearn.model\_selection import GridSearchCV as GridTuner

from sklearn.feature\_selection import SelectFromModel

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.preprocessing import OneHotEncoder

from sklearn import metrics

from sklearn.pipeline import make\_pipeline

from sklearn import linear\_model as lm

from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis as qda

from sklearn import svm

import lightgbm as lgb

import xgboost as xgb

import catboost as cat

from sklearn import neighbors as knn

from sklearn import ensemble

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras import layers

from tensorflow.keras import activations

from tensorflow.keras import optimizers

from tensorflow.keras import metrics as tf\_metrics

from tensorflow.keras import callbacks as tf\_callbacks

import tensorflow\_addons as tfa

import keras\_tuner as kt

from keras\_tuner import HyperModel

from tensorflow.python.ops.inplace\_ops import inplace\_sub

from tqdm.keras import TqdmCallback

# 유용한 함수 정리

# 한글 설정 부분 지움(각자 환경마다 달라서)

# Scikit-Learn label encoder 유용하게 만든 함수

# preset 인자에 dictionary 형태로 원하는 인코딩 값 입력시 그거대로 인코딩

# 아니면 무작위 순서대로 인코딩

class MyLabelEncoder:

def \_\_init\_\_(self, preset={}):

# dic\_cat format -> {"col\_name": {"value": replace}}

self.dic\_cat = preset

def fit\_transform(self, data\_x, col\_names):

tmp\_x = copy.deepcopy(data\_x)

for i in col\_names:

# type check

if not ((tmp\_x[i].dtype.name == "object") or (tmp\_x[i].dtype.name == "category")):

print(F"WARNING : {i} is not object or category")

# if key is not in dic, update dic

if i not in self.dic\_cat.keys():

tmp\_dic = dict.fromkeys(sorted(set(tmp\_x[i]).difference([nan])))

label\_cnt = 0

for j in tmp\_dic.keys():

tmp\_dic[j] = label\_cnt

label\_cnt += 1

self.dic\_cat[i] = tmp\_dic

# transform value which is not in dic to nan

tmp\_x[i] = tmp\_x[i].astype("object")

conv = tmp\_x[i].replace(self.dic\_cat[i])

for conv\_idx, j in enumerate(conv):

if j not in self.dic\_cat[i].values():

conv[conv\_idx] = nan

# final return

tmp\_x[i] = conv.astype("float")

return tmp\_x

def transform(self, data\_x, col\_names):

tmp\_x = copy.deepcopy(data\_x)

for i in col\_names:

if not ((tmp\_x[i].dtype.name == "object") or (tmp\_x[i].dtype.name == "category")):

print(F"WARNING : {i} is not object or category")

# transform value which is not in dic to nan

tmp\_x[i] = tmp\_x[i].astype("object")

conv = tmp\_x[i].replace(self.dic\_cat[i])

for conv\_idx, j in enumerate(conv):

if j not in self.dic\_cat[i].values():

conv[conv\_idx] = nan

# final return

tmp\_x[i] = conv.astype("float")

return tmp\_x

def clear(self, dic\_cat={}):

self.dic\_cat = dic\_cat

# Scikit-Learn OneHotEncoder 유용하게 만든 함수

# label\_preset 에 MyLabelEncoder.dic\_cat 을 입력시 label 에 있는 이름대로 컬럼명이 생성

# 아니면 컬럼명은 0~n임

# fit\_transform, transfrom 의 col\_names 인자는 내가 onehotencoding 하고 싶은 열 설정

# 해당 열은 onehot 인코딩 하고 나머지는 pandas dataframe 으로 알아서 합쳐지게함

class MyOneHotEncoder:

def \_\_init\_\_(self, label\_preset={}):

self.dic\_cat = {}

self.label\_preset = label\_preset

def fit\_transform(self, data\_x, col\_names):

tmp\_x = dataframe()

for i in data\_x:

if i not in col\_names:

tmp\_x = pd.concat([tmp\_x, dataframe(data\_x[i])], axis=1)

else:

if not ((data\_x[i].dtype.name == "object") or (data\_x[i].dtype.name == "category")):

print(F"WARNING : {i} is not object or category")

self.dic\_cat[i] = OneHotEncoder(sparse=False, handle\_unknown="ignore")

conv = self.dic\_cat[i].fit\_transform(dataframe(data\_x[i])).astype("int")

col\_list = []

for j in self.dic\_cat[i].categories\_[0]:

if i in self.label\_preset.keys():

for k, v in self.label\_preset[i].items():

if v == j:

col\_list.append(str(i) + "\_" + str(k))

else:

col\_list.append(str(i) + "\_" + str(j))

conv = dataframe(conv, columns=col\_list)

tmp\_x = pd.concat([tmp\_x, conv], axis=1)

return tmp\_x

def transform(self, data\_x, col\_names):

tmp\_x = dataframe()

for i in data\_x:

if not i in col\_names:

tmp\_x = pd.concat([tmp\_x, dataframe(data\_x[i])], axis=1)

else:

if not ((data\_x[i].dtype.name == "object") or (data\_x[i].dtype.name == "category")):

print(F"WARNING : {i} is not object or category")

conv = self.dic\_cat[i].transform(dataframe(data\_x[i])).astype("int")

col\_list = []

for j in self.dic\_cat[i].categories\_[0]:

if i in self.label\_preset.keys():

for k, v in self.label\_preset[i].items():

if v == j: col\_list.append(str(i) + "\_" + str(k))

else:

col\_list.append(str(i) + "\_" + str(j))

conv = dataframe(conv, columns=col\_list)

tmp\_x = pd.concat([tmp\_x, conv], axis=1)

return tmp\_x

def clear(self, dic\_cat={}, label\_preset={}):

self.dic\_cat = dic\_cat

self.label\_preset = label\_preset

# Scikit-Learn KNNImputer 유용하게 만든 함수

# 카테고리 변수 처리 부분을 넣음 (카테고리는 실수가 나오면 안 돼서)

# fit\_transform, transfrom 의 cat\_vars 에 열이름을 입력하면

# 해당 열의 na 대치값은 반올림 되며, 반올림 된 값이 원래 값범위에 없으면

# 원래 값범위의 최소 or 최대 값으로 바꿔버림

# 예시) 원래 카테고리범위 0~3인데 na imputing 값이 3.6이 나옴

# 반올림 하면 4 인데 이건 원래 범위에 없음 그럼 3으로 강제로 바꿈

class MyKNNImputer:

def \_\_init\_\_(self, k=5):

self.imputer = KNNImputer(n\_neighbors=k)

self.cat\_dic = {}

self.naidx\_dix = {}

def fit\_transform(self, x, y, cat\_vars=None):

for i in cat\_vars:

self.cat\_dic[i] = diff(list(sorted(set(x[i]))), [nan])

self.naidx\_dix[i] = list(which(array(x[i].isna()))[0])

x\_imp = dataframe(self.imputer.fit\_transform(x, y), columns=x.columns)

# if imputed categorical value are not in the range, adjust the value

for i in cat\_vars:

x\_imp[i] = x\_imp[i].apply(lambda x: int(round(x, 0)))

for j in self.naidx\_dix[i]:

if x\_imp[i][j] not in self.cat\_dic[i]:

if x\_imp[i][j] < self.cat\_dic[i][0]:

x\_imp[i][self.naidx\_dix[i]] = self.cat\_dic[i][0]

elif x\_imp[i][j] > self.cat\_dic[i][0]:

x\_imp[i][self.naidx\_dix[i]] = self.cat\_dic[i][len(self.cat\_dic[i]) - 1]

return x\_imp

def transform(self, x):

for i in self.cat\_dic.keys():

self.naidx\_dix[i] = list(which(array(x[i].isna()))[0])

x\_imp = dataframe(self.imputer.transform(x), columns=x.columns)

# if imputed categorical value are not in the range, adjust the value

for i in self.cat\_dic.keys():

x\_imp[i] = x\_imp[i].apply(lambda x: int(round(x, 0)))

for j in self.naidx\_dix[i]:

if x\_imp[i][j] not in self.cat\_dic[i]:

if x\_imp[i][j] < self.cat\_dic[i][0]:

x\_imp[i][self.naidx\_dix[i]] = self.cat\_dic[i][0]

elif x\_imp[i][j] > self.cat\_dic[i][0]:

x\_imp[i][self.naidx\_dix[i]] = self.cat\_dic[i][len(self.cat\_dic[i]) - 1]

return x\_imp

def clear(self, cat\_dic={}, naidx\_dix={}):

self.cat\_dic = cat\_dic

self.naidx\_dix = naidx\_dix

# Pickle 파일 쉽게 읽고 쓰는 함수

# 읽을때는 x에는 None 입력하고 Path에 pickle 파일 경로 입력

# 예시) a = easyIO(None, ./abc.pickle, op=”r”)

# 쓸때는 x에는 원하는 변수 입력하고 Path에 pickle 파일 경로 입력

# 예시) easyIO(abc, ./abc.pickle, op=”r”)

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

tmp = None

if op == "r":

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

tmp = pickle.load(f)

return tmp

elif op == "w":

tmp = {}

print(x)

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]

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")

# 구글 검색하다가 퍼온 함수 (차집합 함수)

# first인자에서 second인자에 있는 값을 제거하고 first 인자 리턴

def diff(first, second):

second = set(second)

return [item for item in first if item not in second]

# 특정 값이 어느 위치에 있는지 인덱스 리스트를 리턴하는 함수

# data\_x 을 for문 돌려 col\_names 에 있는 값이 어디에 있는지 찾음

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

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

# 리스트나 튜플 등 반복가능한 객체에 있는 원소를 ordering 하는 함수

# for\_order에 정렬하고 싶은 객체를 입력하고,

# using\_ref에 원하는 정렬 순서를 가진 객체를 입력

# using\_ref 의 순서대로 정렬된 for\_order 객체를 리턴

# 단, for\_order >(포함기호) using\_ref 관계여야 오류안날듯

def orderElems(for\_order, using\_ref):

return [i for i in using\_ref if i in for\_order]

# ===== 분석 순서 정리 =====

# 1. Preprocessing (중복제거, NA값 대치, 모든컬럼 수치화)

# 2. Feature engineering (년, 월, 요일 등 파생변수 생성 또는 로그화 등과 같은 변수 변환)

# 3. EDA (시각화 및 데이터에 대한 대략적인 정보 획득)

# 4. Anomaly Detection (이상치 탐지 및 이상치 제거 판단)

# 5. Feature Selection (분석에 영향을 주는 중요한 변수 선택, 생략할 수 있음)

및 train, validation 분리

# 6. 모델링 (머신러닝 및 딥러닝 학습 후 예측)

# 7. train 으로 학습된 모델로 validation 을 예측하여 성능 평가

# 8. 결론 도출

# ===== Preprocessing & Feature Engineering =====

# 파일 경로 유의

df\_train = read\_csv("./kdigital\_2ndproject/train.csv", parse\_dates=["일자"], encoding="euc-kr")

df\_test = read\_csv("./kdigital\_2ndproject/test.csv", parse\_dates=["일자"], encoding="euc-kr")

df\_train.drop(["조식메뉴", "석식메뉴", "석식계"], axis=1, inplace=True)

df\_test.drop(["조식메뉴", "석식메뉴"], axis=1, inplace=True)

# 9/12 16:00 추가 : 중복제거

df\_train.drop\_duplicates(inplace=True, ignore\_index=True)

print(df\_train)

df\_train.info()

train\_x = df\_train.drop(["중식계"], axis=1)

train\_y = df\_train["중식계"]

del df\_train

test\_x = df\_test

del df\_test

# ===== feature engineering

# 1. 일자 관련 컬럼

# 년월일주, 요일, 계절 추가

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['주'] = weekofmonth

# 2020년 전후 여부 컬럼추가 - 코로나 관련 (2020전:1 / 2020후:0)

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(train\_x)):

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

if (train\_x.iloc[n, 1]) == (train\_x.iloc[n + 1, 1] - 1):

a.append(0)

else:

a.append(1)

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

if (train\_x.iloc[n, 1] == (train\_x.iloc[n - 1, 1] + 1)):

a.append(0)

else:

a.append(1)

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

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

a.append(0)

else:

a.append(1)

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

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

a.append(0)

else:

a.append(1)

elif ((train\_x.iloc[n, 1] == (train\_x.iloc[n - 1, 1] + 1)) and (train\_x.iloc[n, 1]) == (

train\_x.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)

train\_x["공휴일여부"] = a

train\_x.head(10)

# sin, cos 주기성 변환 파생변수 컬럼추가

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()

# 1. 일자 관련 컬럼

# 년월일주, 요일, 계절 추가

test\_x['년'] = test\_x['일자'].dt.year

test\_x['월'] = test\_x['일자'].dt.month

test\_x['일'] = test\_x['일자'].dt.day

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

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

# 계절

season = []

for i in test\_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)

test\_x['계절'] = season

weekofmonth = []

for i in test\_x["주"]:

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

test\_x['주'] = weekofmonth

# 2020년 전후 여부 컬럼추가 - 코로나 관련 (2020전:1 / 2020후:0)

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

# 공휴일전후여부 컬럼추가 (공휴일전후:1 / 공휴일전후아님:0)

a = []

for n in range(len(test\_x)):

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

if (test\_x.iloc[n, 1]) == (test\_x.iloc[n + 1, 1] - 1):

a.append(0)

else:

a.append(1)

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

if (test\_x.iloc[n, 1] == (test\_x.iloc[n - 1, 1] + 1)):

a.append(0)

else:

a.append(1)

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

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

a.append(0)

else:

a.append(1)

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

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

a.append(0)

else:

a.append(1)

elif ((test\_x.iloc[n, 1] == (test\_x.iloc[n - 1, 1] + 1)) and (test\_x.iloc[n, 1]) == (

test\_x.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)

test\_x["공휴일여부"] = a

test\_x.head(10)

# sin, cos 주기성 변환 파생변수 컬럼추가

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 test\_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))

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

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

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

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

# 2. 인원수 관련 컬럼

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

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

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

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

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

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

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

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

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

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

# 3. 메뉴 관련 컬럼

# Train data

# 메뉴를 밥, 국, 메인메뉴, 반찬1, 반찬2, 김치, 사이드 변수로 분리

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

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

lunch\_train = []

for day in range(len(train\_x)):

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

tmp = ' '.join(tmp).split() # 빈 원소 삭제

search = '(' # 원산지 정보는 삭제

# (원산지는 항상 맨 뒤 space로 구분 되므로 괄호하나만 해도 처리 가능)

for menu in tmp:

if "New" in menu:

pass

elif search in menu:

tmp.remove(menu)

lunch\_train.append(tmp)

# 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

train\_x['밥'] = bob

train\_x['국'] = gook

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

train\_x['김치'] = kimchi

train\_x['사이드'] = dessert

# 특식 컬럼 추가 (쌀밥이 아닌 비빔밥 등의 특식 메뉴 제공)

jmt = []; # 특식 컬럼 변수명 지정

for i in train\_x['밥']:

if ('쌀밥' not in i) and ('흑미밥' not in i):

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

else:

jmt.append(0) # 있는 경우 0을 반환

# print(jmt)

train\_x['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\_x)): # 행 길이 만큼 반복

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

new.append(1) # 1을 반환

else:

new.append(0) # 아님 0을 반환

train\_x['신메뉴'] = new

train\_x[menu\_vec].head(10)

train\_x[menu\_vec].tail(10)

train\_x.drop(["중식메뉴", "밥", "김치"], axis=1, inplace=True)

# Test data

lunch\_train = []

for day in range(len(test\_x)):

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

tmp = ' '.join(tmp).split() # 빈 원소 삭제

search = '(' # 원산지 정보는 삭제

# (원산지는 항상 맨 뒤 space로 구분 되므로 괄호하나만 해도 처리 가능)

for menu in tmp:

if "New" in menu:

pass

elif search in menu:

tmp.remove(menu)

lunch\_train.append(tmp)

# 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)

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

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

# menu\_len\_list

test\_x['밥'] = bob

test\_x['국'] = gook

test\_x['메인메뉴'] = jm; test\_x['반찬1'] = side1; test\_x['반찬2'] = side2

test\_x['김치'] = kimchi

test\_x['사이드'] = dessert

# 특식 컬럼 추가 (쌀밥이 아닌 비빔밥 등의 특식 메뉴 제공)

jmt = []; # 특식 컬럼 변수명 지정

for i in test\_x['밥']:

if ('쌀밥' not in i) and ('흑미밥' not in i):

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

else:

jmt.append(0) # 있는 경우 0을 반환

# print(jmt)

test\_x['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(test\_x)): # 행 길이 만큼 반복

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

new.append(1) # 1을 반환

else:

new.append(0) # 아님 0을 반환

test\_x['신메뉴'] = new

test\_x[menu\_vec].head(10)

test\_x[menu\_vec].tail(10)

test\_x.drop(["중식메뉴", "밥", "김치"], axis=1, inplace=True)

# data check

train\_x.info()

train\_x[menu\_vec].head(5)

train\_x[menu\_vec].tail(5)

test\_x.info()

test\_x[menu\_vec].head(5)

test\_x[menu\_vec].tail(5)

# ===== menu column embedding =====

# 메뉴관련컬럼(5개)을 tensorflow embedding layer를 통해 dense vector로 변환하기

# OneHotEncoding 시 몇 백개 이상의 컬럼이 생성됨

tf.random.set\_seed(100)

# 9/28 테스트 데이터 레이블링 start

# Label Encoding 수행

label\_encoder = MyLabelEncoder()

train\_x = label\_encoder.fit\_transform(train\_x, col\_names=["국", "메인메뉴", "반찬1", "반찬2", "사이드"])

test\_x2 = label\_encoder.transform(test\_x, col\_names=["국", "메인메뉴", "반찬1", "반찬2", "사이드"])

# series(label\_encoder.dic\_cat["국"]).to\_csv("main.csv", encoding="euc-kr")

test\_x["국"][test\_x2["국"].isna()]

test\_x2["국"][test\_x2["국"].isna()]

len(test\_x["국"][test\_x2["국"].isna()])

tmp = [51, 68, 132, 164, 173, 62, 7, 87, 94, 156, 158, 43, 75]

len(tmp)

test\_x2["국"][test\_x2["국"].isna()] = tmp

# series(label\_encoder.dic\_cat["메인메뉴"]).to\_csv("main.csv", encoding="euc-kr")

test\_x["메인메뉴"][test\_x2["메인메뉴"].isn a()]

test\_x2["메인메뉴"][test\_x2["메인메뉴"].isna()]

len(test\_x["메인메뉴"][test\_x2["메인메뉴"].isna()])

tmp = [246, 81, 45, 243, 304, 51, 81, 189, 211, 184, 79, 78, 298, 362, 165, 306,

175, 301, 7, 319, 193, 12, 117, 306, 219, 318]

len(tmp)

test\_x2["메인메뉴"][test\_x2["메인메뉴"].isna()] = tmp

# series(label\_encoder.dic\_cat["반찬1"]).to\_csv("banchan1.csv", encoding="euc-kr")

test\_x["반찬1"][test\_x2["반찬1"].isna()]

test\_x2["반찬1"][test\_x2["반찬1"].isna()]

len(test\_x["반찬1"][test\_x2["반찬1"].isna()])

tmp = [239, 7, 161, 252, 233, 101, 172, 38, 349, 217, 28, 109, 74, 251, 271,178, 421, 177,

112, 303, 256, 136, 347,41, 108, 304, 10, 182, 244, 232, 55, 38, 46, 248, 8, 396]

len(tmp)

test\_x2["반찬1"][test\_x2["반찬1"].isna()] = tmp

# series(label\_encoder.dic\_cat["반찬2"]).to\_csv("banchan2.csv", encoding="euc-kr")

test\_x["반찬2"][test\_x2["반찬2"].isna()]

test\_x2["반찬2"][test\_x2["반찬2"].isna()]

len(test\_x["반찬2"][test\_x2["반찬2"].isna()])

tmp = [132, 56, 56, 56, 96, 373, 56, 287, 279, 139, 154, 420, 155, 227, 214, 239, 228, 56, 108, 335,

19, 119, 48, 4, 88, 56, 56]

len(tmp)

test\_x2["반찬2"][test\_x2["반찬2"].isna()] = tmp

# series(label\_encoder.dic\_cat["메인메뉴"]).to\_csv("main.csv", encoding="euc-kr")

test\_x["사이드"][test\_x2["사이드"].isna()]

test\_x2["사이드"][test\_x2["사이드"].isna()]

len(test\_x["사이드"][test\_x2["사이드"].isna()])

tmp = [260, 55, 35, 260, 260, 410, 260, 260, 55, 260, 55, 35, 260, 260, 35]

len(tmp)

test\_x2["사이드"][test\_x2["사이드"].isna()] = tmp

test\_x = test\_x2

# 9/28 테스트 데이터 레이블링 end

# 레이어 만들기

def createEmbeddingModel(nCols):

# ----------- Embedding layers ----------------------

B0\_input = layers.Input(shape=(nCols), name="B0\_input")

B0\_embedding = layers.Embedding(input\_dim=1024,

output\_dim=64,

name="B0\_embedding")(B0\_input)

# ----------- Convolution layers ----------------------

B1\_conv1d = layers.Conv1D(4, 1, activation='relu', name="B1\_conv1d")(B0\_embedding)

B1\_flatten = layers.Flatten(name="extract")(B1\_conv1d)

# ----------- Regressor ----------------------

last\_regressor = layers.Dense(1)(B1\_flatten)

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

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

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

return model\_mlp

# 메뉴 5개 컬럼에 대해 한 컬럼씩 학습 후 embedding 하기

train\_ebd\_vector = dataframe()

test\_ebd\_vector = dataframe()

for i in menu\_vec:

model\_embedding = createEmbeddingModel(1)

# model\_embedding.summary()

model\_embedding.fit(train\_x[i], train\_y,

validation\_split=0.2,

epochs=200, batch\_size=8, verbose=0,

callbacks=[tf\_callbacks.EarlyStopping(patience=int(200\*0.2), restore\_best\_weights=True),

tf\_callbacks.ReduceLROnPlateau(factor=0.8, patience=int((200\*0.2)/10), min\_lr=1e-4),

TqdmCallback(verbose=0)])

# 한 메뉴에 대해 학습후 flatten layer만 가져와서 모델 만들기

model\_mlp = Model(model\_embedding.get\_layer("B0\_input").output, model\_embedding.get\_layer("extract").output)

# model\_mlp.summary()

# input과 flatten 레이어로 생성된 새로운 모델에 predict 해서 dense vector 추출하기

train\_ebd\_vector = pd.concat([train\_ebd\_vector, dataframe(model\_mlp.predict(train\_x[i], batch\_size=8))], axis=1)

test\_ebd\_vector = pd.concat([test\_ebd\_vector, dataframe(model\_mlp.predict(test\_x[i], batch\_size=8))], axis=1)

# 메뉴 이름 만들기

name\_list = []

for i in train\_x.loc[:,menu\_vec].columns:

for j in range(model\_mlp.get\_layer("B1\_conv1d").output.shape[2]):

name\_list.append(i + "\_" + str(j))

train\_ebd\_vector.columns = name\_list

test\_ebd\_vector.columns = name\_list

train\_x = pd.concat([train\_x.drop(menu\_vec, axis=1), train\_ebd\_vector], axis=1)

test\_x = pd.concat([test\_x.drop(menu\_vec, axis=1), test\_ebd\_vector], axis=1)

train\_ebd\_vector.iloc[4, 0:8]

test\_ebd\_vector.iloc[4, 0:8]

# 데이터 확인

train\_x.info()

test\_x.info()

train\_x.head(10)

test\_x.head(10)

# 기상청 외부데이터 추가 (날씨)

# 점심 11,12,13 저녁 17,18,19

forecast = read\_csv("./kdigital\_2ndProject/진주\_강수기온/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"]]

# train\_x.drop(["일자"], axis=1, inplace=True)

train\_x.tail(50)

# 9/28 테스트 강수기온 업데이트

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

test\_x.isna().sum().sum()

# 테스트는 결측치 없음

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

# test\_x.drop(["일자"], axis=1, inplace=True)

test\_x.tail(50)

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

# # hue 옵션은 2차원 데이터에 추가할 3차원 자료들 ex -> (x1, x2, 클래스)

# train\_tmp = train\_x.copy()

# train\_tmp["target"] = train\_y.copy()

# 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")

# 9/28 테스트 데이터에도 일자와 일 drop

# 일자와 일 컬럼 Drop 후 categorical 컬럼들 int 로 변환

train\_x.drop(["일자", "일"], axis=1, inplace=True)

train\_x.info()

test\_x.drop(["일자", "일"], axis=1, inplace=True)

test\_x.info()

# 년도와 월을 0~n 까지 Label Encoding (LightGBM 오류 방지)

train\_x = label\_encoder.fit\_transform(train\_x, ["년", "월"])

test\_x = label\_encoder.transform(test\_x, ["년", "월"])

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

test\_x.isna().sum().sum()

# train\_x["년"].value\_counts()

# train\_x["월"].value\_counts()

cat\_vars = ["요일", "년", "월", "주", "계절"]

for i in cat\_vars:

train\_x[i] = train\_x[i].astype("int")

test\_x[i] = test\_x[i].astype("int")

# feature engineering 후 최종 데이터 확인

train\_x.info()

train\_x.head(10)

test\_x.info()

test\_x.head(10)

# # ===== Anomaly Detection =====

# # Isolation Forest

# # 무작위 n 개의 feature 를 뽑아 무작위 split 을 합니다. (max\_depth 까지)

# # 각 객체마다 루트노드와 리프노드의 평균 거리를 계산해 anomaly 여부를 판단합니다.

# # 데이터에 이상치가 적다면 contamination을 작게 지정, 데이터에 이상치가 많다면 contamination 값을 크게 설정 할 수 있다.

# ad\_eif = IsolationForest(n\_estimators=100, max\_samples=train\_x.shape[0], max\_features=2, contamination=0.01,

# bootstrap=True, n\_jobs=multiprocessing.cpu\_count(), random\_state=898)

# anomalyVec = ad\_eif.fit\_predict(train\_x)

#

#

# # 이상치인 값을 제거

# train\_x = train\_x[anomalyVec == 1]

# train\_x.reset\_index(drop=True, inplace=True)

# train\_y = train\_y[anomalyVec == 1]

# train\_y.reset\_index(drop=True, inplace=True)

# ===== Feature Importance Summary (LightGBM feature importance) =====

menu\_vec = ["국", "메인메뉴", "반찬1", "반찬2", "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)

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=0.6, colsample\_bytree=0.8, silent=True)

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

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

# one hot encoding

oh\_encoder = MyOneHotEncoder()

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

test\_x\_oh = oh\_encoder.transform(test\_x, cat\_vars)

# 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)

# ===== Modeling (각자 맡은 부분) =====