import os

import IPython

import multiprocessing

import copy

import pickle

import warnings

from datetime import datetime, timedelta

from time import time, sleep, mktime

from matplotlib import font\_manager as fm, rc, rcParams

import matplotlib.pyplot as plt

import seaborn as sns

from tqdm import tqdm

import re

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, read\_csv

from pandas.tseries.offsets import DateOffset

import statsmodels.api as sm

from sklearn import preprocessing as prep

from sklearn.impute import KNNImputer

from sklearn.model\_selection import train\_test\_split as tts, GridSearchCV as GridTuner, StratifiedKFold, KFold

from sklearn.feature\_selection import SelectFromModel

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

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

# # ===== tensorflow =====

# 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

# from tqdm.keras import TqdmCallback

# from scikeras.wrappers import KerasClassifier, KerasRegressor

# import tensorflow\_addons as tfa

# import keras\_tuner as kt

# from keras\_tuner import HyperModel

# ===== NLP =====

from selenium import webdriver

from konlpy.tag import Okt

from KnuSentiLex.knusl import KnuSL

# ===== task specific =====

import pykrx

# display setting

warnings.filterwarnings(action='ignore')

rcParams['axes.unicode\_minus'] = False

pd.set\_option('display.max\_columns', 100)

pd.set\_option('display.max\_rows', 100)

pd.set\_option('display.width', 1000)

# font setting

font\_path = 'myfonts/NanumSquareB.ttf'

font\_obj = fm.FontProperties(fname=font\_path, size=12).get\_name()

rc('font', family=font\_obj)

# %reset -f

# ===== utility functions =====

# label encoding for categorical column with excepting na value

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):

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

for i in list(self.dic\_cat.keys()):

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):

self.dic\_cat = {}

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):

tmp\_x = dataframe()

for i in data\_x:

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

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):

self.dic\_cat = {}

self.label\_preset = {}

class MyKNNImputer:

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

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

self.cat\_dic = {}

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

naIdx = dict.fromkeys(cat\_vars)

for i in cat\_vars:

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

naIdx[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 naIdx[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][naIdx[i]] = self.cat\_dic[i][0]

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

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

return x\_imp

def transform(self, x):

naIdx = dict.fromkeys(self.cat\_vars)

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

naIdx[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 naIdx[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][naIdx[i]] = self.cat\_dic[i][0]

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

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

return x\_imp

def clear(self):

self.imputer = None

self.cat\_dic = {}

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] = copy.deepcopy(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")

def diff(first, second):

second = set(second)

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

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

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

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

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

# concatenate by row

def ccb(df1, df2):

if type(df1) == series:

tmp\_concat = series(pd.concat([dataframe(df1), dataframe(df2)], axis=0, ignore\_index=True).iloc[:,0])

tmp\_concat.reset\_index(drop=True, inplace=True)

elif type(df1) == dataframe:

tmp\_concat = pd.concat([df1, df2], axis=0, ignore\_index=True)

tmp\_concat.reset\_index(drop=True, inplace=True)

elif type(df1) == np.ndarray:

tmp\_concat = np.concatenate([df1, df2], axis=0)

else:

print("Unknown Type: return 1st argument")

tmp\_concat = df1

return tmp\_concat

def change\_width(ax, new\_value):

for patch in ax.patches :

current\_width = patch.get\_width()

adj\_value = current\_width - new\_value

# we change the bar width

patch.set\_width(new\_value)

# we recenter the bar

patch.set\_x(patch.get\_x() + adj\_value \* .5)

def week\_of\_month(date):

month = date.month

week = 0

while date.month == month:

week += 1

date -= timedelta(days=7)

return week

def dispPerformance(result\_dic, result\_metrics):

perf\_table = dataframe(columns=result\_metrics)

for k, v in result\_dic.items():

perf\_table = pd.concat([perf\_table, v["performance"]], ignore\_index=True, axis=0)

print(perf\_table)

return perf\_table

folder\_path = "./projects/dacon\_stockprediction/"

from pykrx import stock

# Get Stock List

path = 'projects/dacon\_stockprediction/open\_week4/'

list\_name = 'Stock\_List.csv'

sample\_name = 'sample\_submission\_week4.csv'

# 종목 코드 로드

stock\_list = read\_csv(os.path.join(path, list\_name))

stock\_list['종목코드'] = stock\_list['종목코드'].apply(lambda x: str(x).zfill(6))

stock\_list

# Get Data & Modeling

# 분석할 date 변수 지정

start\_date = '20201201'

end\_date = '20211001'

start\_weekday = pd.to\_datetime(start\_date).weekday()

max\_weeknum = pd.to\_datetime(end\_date).strftime('%V')

business\_days = pd.DataFrame(pd.date\_range(start\_date, end\_date, freq='B'), columns=['Date'])

print(f'WEEKDAY of "start\_date" : {start\_weekday}')

print(f'NUM of WEEKS to "end\_date" : {max\_weeknum}')

print(f'HOW MANY "Business\_days" : {business\_days.shape}', )

print(business\_days.head(20))

# raw features (5개)

# 주가, 거래량, 기관순매수, 외인순매수, 뉴스 기사(embedding)

# derived features (14개)

# 주가이평, 거래량이평, 기관순매수이평, 외인순매수이평, 뉴스 기사에 대한 긍부정점수, 요일, sin변환(5일), cos변환(5일)

# 산식 보조 지표

# 1. 주가 관련 지표 : Stochastic(20), RSI(20), 볼린저밴드(20)

# 2. 거래량 관련 지표 : OBV, VR(20)

# 3. 혼합지표 : MFI(주가 + 거래량)

# ===== raw data loading =====

# 한 종목코드에 대한 주가 정보를 로드

stock\_code = stock\_list.loc[0, '종목코드']

stock\_df = stock.get\_market\_ohlcv\_by\_date(start\_date, end\_date, stock\_code).reset\_index()

investor\_df = stock.get\_market\_trading\_volume\_by\_date(start\_date, end\_date, stock\_code)[["기관합계", "외국인합계"]].reset\_index()

kospi\_df = stock.get\_index\_ohlcv\_by\_date(start\_date, end\_date, "1001")[["종가"]].reset\_index()

stock\_df.columns = ["Date", "Open", "High", "Low", "Close", "Volume"]

investor\_df.columns = ["Date", "inst", "fore"]

kospi\_df.columns = ["Date", "kospi"]

# 영업일과 주가 정보를 outer 조인

train\_x = pd.merge(business\_days, stock\_df, how='left', on="Date")

train\_x = pd.merge(train\_x, investor\_df, how='left', on="Date")

train\_x = pd.merge(train\_x, kospi\_df, how='left', on="Date")

# 종가데이터에 생긴 na 값을 선형보간 및 정수로 반올림

train\_x.iloc[:,1:] = train\_x.iloc[:,1:].ffill(axis=0).round(0)

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

# ===== feature engineering =====

# 요일 및 주차 파생변수 추가

train\_x['weekday'] = train\_x["Date"].apply(lambda x: x.weekday())

train\_x['weeknum'] = train\_x["Date"].apply(lambda x: week\_of\_month(x))

cat\_vars = ["weekday", "weeknum"]

# 주기성 신호로 변환한 파생변수 추가 (이건 요일 특성을 잡아주는거랑 다를바가 없으니 다른 접근 필요)

# 차라리 해당 월에 몇번째 일 and 해당 년 몇번째 일인지

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

month\_to\_sec = 20 \* day\_to\_sec

timestamp\_s = train\_x["Date"].apply(datetime.timestamp)

timestamp\_freq = round((timestamp\_s / month\_to\_sec).diff(20)[20],1)

train\_x['monthday\_freq\_sin'] = np.sin((timestamp\_s / month\_to\_sec) \* ((2 \* np.pi) / timestamp\_freq))

train\_x['monthday\_freq\_cos'] = np.cos((timestamp\_s / month\_to\_sec) \* ((2 \* np.pi) / timestamp\_freq))

# sns.lineplot(data=train\_x['monthday\_freq\_sin'][:-1], color="g")

# ax2 = plt.twinx()

# sns.lineplot(data=train\_x["Close"][1:], color="b", ax=ax2)

# train\_x.head(30)

np.corrcoef(train\_x['monthday\_freq\_sin'][:-1], train\_x["Close"][1:])

# sns.lineplot(data=train\_x['monthday\_freq\_cos'][:-1], color="g")

# ax2 = plt.twinx()

# sns.lineplot(data=train\_x["Close"][1:], color="b", ax=ax2)

# train\_x.head(30)

np.corrcoef(train\_x['monthday\_freq\_cos'][:-1], train\_x["Close"][1:])

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

weekday\_to\_sec = 5 \* day\_to\_sec

timestamp\_s = train\_x["Date"].apply(datetime.timestamp)

timestamp\_freq = round((timestamp\_s / weekday\_to\_sec).diff(5)[5],1)

train\_x['weekday\_freq\_sin'] = np.sin((timestamp\_s / weekday\_to\_sec) \* ((2 \* np.pi) / timestamp\_freq))

train\_x['weekday\_freq\_cos'] = np.cos((timestamp\_s / weekday\_to\_sec) \* ((2 \* np.pi) / timestamp\_freq))

# sns.lineplot(data=train\_x['weekday\_freq\_sin'][:-1], color="g")

# ax2 = plt.twinx()

# sns.lineplot(data=train\_x["Close"][1:], color="b", ax=ax2)

# train\_x.head(30)

np.corrcoef(train\_x['weekday\_freq\_sin'][:-1], train\_x["Close"][1:])

# sns.lineplot(data=train\_x['weekday\_freq\_cos'][:-1], color="g")

# ax2 = plt.twinx()

# sns.lineplot(data=train\_x["Close"][1:], color="b", ax=ax2)

# train\_x.head(30)

np.corrcoef(train\_x['weekday\_freq\_cos'][:-1], train\_x["Close"][1:])

train\_x.drop(['monthday\_freq\_cos', 'weekday\_freq\_sin', 'weekday\_freq\_cos'], axis=1, inplace=True)

# setting metrics days

metric\_days = 14

# obv

obv = [0]

for i in range(1, len(train\_x.Close)):

if train\_x.Close[i] > train\_x.Close[i - 1]:

obv.append(obv[-1] + train\_x.Volume[i])

elif train\_x.Close[i] < train\_x.Close[i - 1]:

obv.append(obv[-1] - train\_x.Volume[i])

else:

obv.append(obv[-1])

train\_x['obv'] = obv

train\_x['obv'][0] = nan

train\_x['obv\_ema'] = train\_x['obv'].ewm(com=metric\_days, min\_periods=metric\_days).mean()

# 매수/매도 타이밍 신호 찾는 함수

# 매수 신호: obv > obv\_ema

# 매도 신호: obv < obv\_ema

def getBreakthroughPoint(df, col1, col2, patient\_days, fill\_method="fb"):

'''

:param df: dataframe (including col1, col2)

:param col1: obj

:param col2: obj moving average

:param patient\_days: patient days detected as breakthrough point

:return: signal series

'''

sigPrice = []

flag = -1 # A flag for the trend upward/downward

for i in range(0, len(df)):

if df[col1][i] > df[col2][i] and flag != 1:

tmp = df['Close'][i:(i + patient\_days + 1)]

if len(tmp) == 1:

sigPrice.append("buy")

flag = 1

else:

if (tmp.iloc[1:] > tmp.iloc[0]).all():

sigPrice.append("buy")

flag = 1

else:

sigPrice.append(nan)

elif df[col1][i] < df[col2][i] and flag != 0:

tmp = df['Close'][i:(i + patient\_days + 1)]

if len(tmp) == 1:

sigPrice.append("sell")

flag = 0

else:

if (tmp.iloc[1:] < tmp.iloc[0]).all():

sigPrice.append("sell")

flag = 0

else:

sigPrice.append(nan)

else:

sigPrice.append(nan)

sigPrice = series(sigPrice)

for idx, value in enumerate(sigPrice):

if not isna(value):

if value == "buy":

sigPrice.iloc[1:idx] = "sell"

else:

sigPrice.iloc[1:idx] = "buy"

break

# if fill\_method == "bf":

#

# elif fill\_method == ""

sigPrice.ffill(inplace=True)

return sigPrice

# train\_x['obv\_signal'] = getBreakthroughPoint(train\_x, 'obv', 'obv\_ema', 2)

train\_x

# #OBV와 OBV\_EMA 시각화

# plt.figure(figsize=(12,8))

# plt.plot(train\_x['obv'], label='obv', color='orange')

# plt.plot(train\_x['obv\_ema'], label='obv\_ema', color='purple')

# plt.legend(loc='upper right')

# plt.xticks(rotation=45)

# #매수/매도 신호 시각화

# plt.figure(figsize=(12,8))

# plt.scatter(train\_x.index[train\_x['obv\_signal']=="buy"], train\_x["Close"][train\_x['obv\_signal']=="buy"], color = 'green',

# label = 'Buy Signal', marker = '^', alpha = 1)

# plt.scatter(train\_x.index[train\_x['obv\_signal']=="sell"], train\_x["Close"][train\_x['obv\_signal']=="sell"], color = 'red',

# label = 'Sell Signal', marker = 'v', alpha = 1)

# # plt.plot(train\_x['obv'], label = 'OBV', alpha = 0.35)

# # plt.plot(train\_x['obv\_ema'], label = 'OBV moving average', alpha = 0.35)

# plt.plot(train\_x['Close'], label = 'Price', alpha = 0.35)

# plt.xticks(rotation=45)

# plt.title('Buy & Sell zone visualization', fontsize=15, fontweight="bold", pad=15)

# plt.xlabel('Date', fontsize = 14)

# plt.ylabel('Close Price', fontsize=14)

# plt.legend(loc='upper right')

# plt.show()

### stochastic 계산식

def stochastic(df, n=14, m=5, t=5):

#데이터 프레임으로 받아오기 때문에 불필요

#n 일중 최저가

ndays\_high = df['High'].rolling(window=n, min\_periods=n).max()

ndays\_low = df['Low'].rolling(window=n, min\_periods=n).min()

fast\_k = ((df['Close'] - ndays\_low) / (ndays\_high - ndays\_low) \* 100)

slow\_k = fast\_k.ewm(span=m, min\_periods=m).mean()

slow\_d = slow\_k.ewm(span=t, min\_periods=t).mean()

df = df.assign(fast\_k=fast\_k, fast\_d=slow\_k, slow\_k=slow\_k, slow\_d=slow\_d)

return df

# 호출 방법

train\_x[["fast\_k", "fast\_d", "slow\_k", "slow\_d"]] = stochastic(train\_x, n=metric\_days)[["fast\_k", "fast\_d", "slow\_k", "slow\_d"]]

# train\_x['stochastic\_signal'] = getBreakthroughPoint(train\_x, 'fast\_k', 'fast\_d', 2)

train\_x.head(20)

#MFI 지표 구하기

#MFI = 100 - (100/1+MFR)

#MFR = 14일간의 양의 MF/ 14일간의 음의 MF

#MF = 거래량 \* (당일고가 + 당일저가 + 당일종가) / 3

train\_x.tail(20)

#MF 컬럼 만들기

train\_x["mf"] = train\_x["Volume"] \* ((train\_x["High"]+train\_x["Low"]+train\_x["Close"]) / 3)

#양의 MF와 음의 MF 표기 컬럼 만들기

p\_n = []

for i in range(len(train\_x['mf'])):

if i == 0 :

p\_n.append(nan)

else:

if train\_x['mf'][i] >= train\_x['mf'][i-1]:

p\_n.append('p')

else:

p\_n.append('n')

train\_x['p\_n'] = p\_n

#14일간 양의 MF/ 14일간 음의 MF 계산하여 컬럼 만들기

mfr = []

for i in range(len(train\_x['mf'])):

if i < metric\_days-1:

mfr.append(nan)

else:

train\_x\_=train\_x.iloc[(i-metric\_days+1):i]

a = sum(train\_x\_['mf'][train\_x['p\_n']=='p']) / sum(train\_x\_['mf'][train\_x['p\_n'] == 'n'])

mfr.append(a)

train\_x['mfr'] = mfr

# 최종 MFI 컬럼 만들기

train\_x['mfi'] = 100 - (100/(1+train\_x['mfr']))

# train\_x["mfi\_signal"] = train\_x['mfi'].apply(lambda x: "buy" if x > 50 else "sell")

train\_x.drop(["slow\_k", "slow\_d", "mf", "p\_n", "mfr", "Open", "High", "Low"], inplace=True, axis=1)

train\_x.head(20)

train\_x["close\_mv5"] = train\_x["Close"].rolling(5, min\_periods=5).mean()

train\_x["close\_mv10"] = train\_x["Close"].rolling(10, min\_periods=10).mean()

train\_x["close\_mv20"] = train\_x["Close"].rolling(20, min\_periods=20).mean()

train\_x["volume\_mv5"] = train\_x["Volume"].rolling(5, min\_periods=5).mean()

train\_x["volume\_mv10"] = train\_x["Volume"].rolling(10, min\_periods=10).mean()

train\_x["volume\_mv20"] = train\_x["Volume"].rolling(20, min\_periods=20).mean()

train\_x["inst\_mv5"] = train\_x["inst"].rolling(5, min\_periods=5).mean()

train\_x["inst\_mv10"] = train\_x["inst"].rolling(10, min\_periods=10).mean()

train\_x["inst\_mv20"] = train\_x["inst"].rolling(20, min\_periods=20).mean()

train\_x["fore\_mv5"] = train\_x["fore"].rolling(5, min\_periods=5).mean()

train\_x["fore\_mv10"] = train\_x["fore"].rolling(10, min\_periods=10).mean()

train\_x["fore\_mv20"] = train\_x["fore"].rolling(20, min\_periods=20).mean()

train\_x["kospi\_mv5"] = train\_x["kospi"].rolling(5, min\_periods=5).mean()

train\_x["kospi\_mv10"] = train\_x["kospi"].rolling(10, min\_periods=10).mean()

train\_x["kospi\_mv20"] = train\_x["kospi"].rolling(20, min\_periods=20).mean()

# 2021/1/4 이후 일자만 선택

train\_x = train\_x[train\_x["Date"] >= datetime(2021, 1, 4)]

train\_x = train\_x.dropna()

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

# create target list

target\_list = []

target\_list.append(train\_x["Close"])

target\_list.append(train\_x["Close"].shift(-1))

target\_list.append(train\_x["Close"].shift(-2))

target\_list.append(train\_x["Close"].shift(-3))

target\_list.append(train\_x["Close"].shift(-4))

target\_list.append(train\_x["Close"].shift(-5))

for idx, value in enumerate(target\_list[1:]):

value.name = "close\_shift" + str(idx+1)

train\_x.columns = train\_x.columns.str.lower()

train\_x = pd.concat([train\_x[["date"]], train\_x.iloc[:,1:].sort\_index(axis=1)], axis=1)

# bi\_data = pd.concat([train\_x, train\_x["close"].shift(-1)], axis=1, ignore\_index=True)[:-1]

# bi\_data.columns = list(train\_x.columns) + ["close\_shift1"]

# bi\_data.to\_csv("projects/dacon\_stockprediction/bi\_data.csv", encoding="euc-kr", index=False)

# ===== visualization =====

# 상관관계 시각화

# fig, ax = plt.subplots(figsize=(12, 6))

# corr\_obj = pd.concat([train\_x[:-1], target\_list[1][:-1]], axis=1).corr().round(3)

# sns.heatmap(corr\_obj, cmap="YlGnBu", linewidths=0.2, annot=True)

# # sns.heatmap(corr\_obj, cmap="YlGnBu", linewidths=0.2, annot=True)

# # plt.gcf().set\_size\_inches(16, 12)

# plt.show()

# # plt.savefig('projects/dacon\_stockprediction/graphs/corr\_heatmap.png', dpi=300)

# small\_corr = corr\_obj.index[corr\_obj["close\_shift1"].abs() < 0.1]

# small\_corr = corr\_obj["close\_shift1"].abs().sum()

# plt.title('Correlation Visualization', fontsize=15, fontweight="bold", pad=15)

# train\_x.head(20)

# # ===== scatter plot on numerical feature =====

# for i in train\_x.columns:

# if i == "Date" or i in cat\_vars:

# pass

# else:

# fig, ax = plt.subplots(figsize=(12, 6))

# graph = sns.regplot(x=train\_x[i][:-1], y=train\_x["Close"][1:], color="green",

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

# graph.set\_title(i, fontsize=15, fontweight="bold", pad=15)

# plt.show()

# # plt.savefig('projects/dacon\_stockprediction/graphs/' + i +".png", dpi=300)

# # ===== box plot on categorical feature =====

# for i in train\_x.columns:

# if i == "Date" or i not in cat\_vars:

# pass

# else:

# fig, ax = plt.subplots(figsize=(12,6))

# graph = sns.boxplot(x=train\_x[i][:-1], y=train\_x["Close"][1:], palette=sns.hls\_palette())

# graph.set\_title("Boxplot by " + i, fontsize=15, fontweight="bold", pad=15)

# change\_width(ax, 0.2)

# plt.show()

# # plt.savefig('projects/dacon\_stockprediction/graphs/' + i +".png", dpi=300)

# 분산분석

from scipy.stats import f\_oneway

tmp = pd.concat([train\_x, target\_list[1].to\_frame()], axis=1, ignore\_index=True)[1:-1]

tmp.columns = list(train\_x.columns) + ["target"]

cat\_list = tmp.groupby("weekday")["target"].apply(list)

# 귀무가설(H0) : 두 변수는 상관관계가 없다

# 대립가설(H1) : 두 변수는 상관관계가 있다

anova = f\_oneway(\*cat\_list)

print(anova)

cat\_list = tmp.groupby("weeknum")["target"].apply(list)

# 귀무가설(H0) : 두 변수는 상관관계가 없다

# 대립가설(H1) : 두 변수는 상관관계가 있다

anova = f\_oneway(\*cat\_list)

print(anova)

# 영향력 적은 변수 제거 및 재시각화

# train\_x.drop(["close\_mv10", "close\_mv20", "volume\_mv5", "volume\_mv10", "inst\_mv5", "inst\_mv20", "fore\_mv5", "fore\_mv10", "kospi\_mv5", "kospi\_mv10",

# "monthday\_freq\_sin", "obv\_ema", "fast\_k", "mfi", "weekday", "weeknum"], axis=1, inplace=True)

# fig, ax = plt.subplots(figsize=(12, 6))

# corr\_obj = pd.concat([train\_x[:-1], target\_list[1][1:]], axis=1).corr().round(3)

# sns.heatmap(corr\_obj, cmap="YlGnBu", linewidths=0.2, annot=True)

# # sns.heatmap(corr\_obj, cmap="YlGnBu", linewidths=0.2, annot=True)

# # plt.gcf().set\_size\_inches(16, 12)

# plt.show()

# # plt.savefig('projects/dacon\_stockprediction/graphs/corr\_heatmap.png', dpi=300)

# small\_corr = corr\_obj.index[corr\_obj["close\_shift1"].abs() < 0.1]

# small\_corr = corr\_obj["close\_shift1"].abs().sum()

# plt.title('Correlation Visualization', fontsize=15, fontweight="bold", pad=15)

onehot\_encoder = MyOneHotEncoder()

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

print(train\_x.info())

print(train\_x\_oh.info())

# dimension check

train\_x.info()

train\_x.head(10)

train\_x\_oh.head(10)

# remove date

full\_x = train\_x.copy()[:-1]

full\_x\_oh = train\_x\_oh.copy()[:-1]

full\_y = target\_list[1][:-1]

del train\_x, train\_x\_oh

# # train test split

# train 2021/1/6 ~ 2021/9/5

# validation 2021/9/6 ~ 2021/9/17

# test 2021/9/27 ~ 2021/10/1

train\_x = full\_x[full\_x["date"] < datetime(2021, 9, 6)]

train\_x\_oh = full\_x\_oh[full\_x["date"] < datetime(2021, 9, 6)]

train\_y = full\_y[full\_x["date"] < datetime(2021, 9, 6)]

val\_x = full\_x[(full\_x["date"] >= datetime(2021, 9, 6)) & (full\_x["date"] < datetime(2021, 9, 18))]

val\_x\_oh = full\_x\_oh[(full\_x["date"] >= datetime(2021, 9, 6)) & (full\_x["date"] < datetime(2021, 9, 18))]

val\_y = full\_y[(full\_x["date"] >= datetime(2021, 9, 6)) & (full\_x["date"] < datetime(2021, 9, 18))]

test\_x = full\_x[full\_x["date"] >= datetime(2021, 9, 27)]

test\_x\_oh = full\_x\_oh[full\_x["date"] >= datetime(2021, 9, 27)]

test\_y = full\_y[full\_x["date"] >= datetime(2021, 9, 27)]

full\_x.shape[0] == train\_x.shape[0] + val\_x.shape[0] + test\_x.shape[0] + 5

full\_y.shape[0] == train\_y.shape[0] + val\_y.shape[0] + test\_y.shape[0] + 5

full\_x.drop("date", axis=1, inplace=True)

full\_x\_oh.drop("date", axis=1, inplace=True)

train\_x.drop("date", axis=1, inplace=True)

train\_x\_oh.drop("date", axis=1, inplace=True)

val\_x.drop("date", axis=1, inplace=True)

val\_x\_oh.drop("date", axis=1, inplace=True)

test\_x.drop("date", axis=1, inplace=True)

test\_x\_oh.drop("date", axis=1, inplace=True)