#시군구 생성

import pandas as pd

# 시도 시군구 비자치구 읍면동 컬럼을 생성하는 함수

def split\_legal\_dong(row):

parts = str(row).split() # 문자열로 변환하여 분할

if len(parts) == 3:

province = parts[0]

city\_district = parts[1]

subdistrict = None

neighborhood = parts[2]

elif len(parts) == 4:

province = parts[0]

city\_district = parts[1]

subdistrict = parts[2]

neighborhood = parts[3]

elif len(parts) == 2:

province = parts[0]

city\_district = None

subdistrict = None

neighborhood = parts[1]

else:

province = None

city\_district = None

subdistrict = None

neighborhood = None

return pd.Series([province, city\_district, subdistrict, neighborhood], index=['시도', '시군구', '비자치구', '읍면동'])

data = pd.read\_excel('물류창고업체\_코드작업완료\_1차 완료\_250128\_집계 전.xlsx')

data[['시도', '시군구', '비자치구', '읍면동']] = data['법정동명'].apply(split\_legal\_dong)

data.head()

data['비자치구']=data['비자치구'].fillna('None')

data.head()

data['비자치구']=data['비자치구'].fillna('None')

data.head()

data['일반창고동수']=data['일반창고동수'].fillna(0)

data['일반창고면적']=data['일반창고면적'].fillna(0)

data['냉동냉장창고동수']=data['냉동냉장창고동수'].fillna(0)

data['냉동냉장창고면적']=data['냉동냉장창고면적'].fillna(0)

data['보관장소면적']=data['보관장소면적'].fillna(0)

data['직원수']=data['직원수'].fillna(0)

data['법인여부명']=data['법인여부명'].fillna(0)

data['업태보관및창고업']=data['업태보관및창고업'].fillna(0)

data['업태운송및택배업']=data['업태운송및택배업'].fillna(0)

data['업태판매업']=data['업태판매업'].fillna(0)

data['업태제조업']=data['업태제조업'].fillna(0)

data.head()

data.info()

###

data.columns

###

data["업태운송및택배업"].unique()

data.to\_csv('물류창고업체\_전처리완료.csv', index=False, encoding='utf-8-sig')

result = data.groupby(['법정동코드','시도','시군구','비자치구','읍면동', '영업상태명']).agg(

물류창고업체수=('법정동코드', 'size')

).reset\_index()

result.head()

# 영업상태명이 영업/정상인거만

result = result[result['영업상태명']=='영업/정상']

result

result.to\_csv('물류창고업체\_집계완료.csv', index=False, encoding='utf-8-sig')

result2 = result2 = data.groupby(['법정동코드', '시도', '시군구']).agg(

물류창고업체수=('법정동코드', 'size'), # 사업장 수 계산

).reset\_index()

result2

result2.to\_csv('물류창고업체수\_시도시군구집계.csv', index=False, encoding='utf-8-sig')

#법정동코드 생성

import pandas as pd

data = pd.read\_excel(r'/Users/dd/Downloads/fulldata\_01\_01\_02\_P\_의원.xlsx')

code = pd.read\_excel(r'/Users/dd/Downloads/법정동코드\_존재.xlsx')

import pandas as pd

import re

def modity\_sido(sido):

# 특별시, 광역시

if re.findall(r'서울', sido):

sido = '서울특별시'

elif re.findall(r'부산', sido):

sido = '부산광역시'

elif re.findall(r'대구', sido):

sido = '대구광역시'

elif re.findall(r'인천', sido):

sido = '인천광역시'

elif re.findall(r'광주', sido):

sido = '광주광역시'

elif re.findall(r'대전', sido):

sido = '대전광역시'

elif re.findall(r'울산', sido):

sido = '울산광역시'

# 특별자치시

elif re.findall(r'세종', sido):

sido = '세종특별자치시'

#특별자치도

elif re.findall(r'강원', sido):

sido = '강원특별자치도'

elif re.findall(r'제주', sido):

sido = '제주특별자치도'

# 남도 북도

elif re.findall(r'충남', sido):

sido = '충청남도'

elif re.findall(r'충북', sido):

sido = '충청북도'

elif re.findall(r'경남', sido):

sido = '경상남도'

elif re.findall(r'경북', sido):

sido = '경상북도'

elif re.findall(r'전남', sido):

sido = '전라남도'

elif re.findall(r'전북|전라북도', sido):

sido = '전북특별자치도'

else:

pass

return sido

def extract\_address(row):

address\_parts = row.split()

if len(address\_parts) > 0:

address\_parts[0] = modity\_sido(address\_parts[0])

for i, part in enumerate(address\_parts):

if part.endswith(('읍', '면', '동', '가', '로')):

return ' '.join(address\_parts[:i+1])

return None

def process\_and\_merge1(data, code):

data['소재지전체주소'] = data['소재지전체주소'].fillna('')

data['주소'] = data['소재지전체주소'].apply(extract\_address)

data\_merge = pd.merge(data, code, left\_on='주소', right\_on='법정동명', how='left')

data\_merge.drop('법정동명', axis=1, inplace=True)

return data\_merge

# 도로명 전체주소에서 주소2 컬럼 생성하는 함수

def road\_process\_address(address):

words = address.split()

if len(words) > 0:

words[0] = modity\_sido(words[0])

first\_three\_words = ' '.join(words[:3])

if len(words) > 2 and words[2].endswith(('로', '길', '리')):

first\_three\_words = ' '.join(words[:2])

temp = re.findall(r'\((.\*?)\)', address)

extracted\_text = ''

if temp:

extracted\_text = temp[0].split(',')[0] if ',' in temp[0] else temp[0]

combined\_address = first\_three\_words + ' ' + extracted\_text

return combined\_address.strip()

# 데이터프레임 병합 함수

def process\_and\_merge2(data, code):

data['도로명전체주소'] = data['도로명전체주소'].fillna('')

data['주소2'] = data['도로명전체주소'].apply(road\_process\_address)

data\_merge = pd.merge(data, code, left\_on='주소2', right\_on='법정동명', how='left')

data\_merge.drop('법정동명', axis=1, inplace=True)

data\_merge['법정동코드'] = data\_merge['법정동코드\_x'].combine\_first(data\_merge['법정동코드\_y'])

data\_merge.drop(['법정동코드\_x', '법정동코드\_y'], axis=1, inplace=True)

return data\_merge

def split\_address(df):

df[['시도', '시군구']] = df['주소'].apply(lambda row: pd.Series(row.split()[:2]))

return df

data = process\_and\_merge2(process\_and\_merge1(data,code),code)

idx\_js2 = data.columns.get\_loc("주소2")

idx\_cd = data.columns.get\_loc("법정동코드")

idx\_code = code.columns.get\_loc("법정동코드")

idx\_bjd = code.columns.get\_loc("법정동명")

for i in range(len(data)):

if np.isnan(data.iat[i, idx\_cd]) and re.findall(r"면|읍", data.iat[i,idx\_js2]):

data.iat[i,idx\_js2] = re.search(r".\*?면|.\*?읍", data.iat[i,idx\_js2])[0]

for k in range(len(code)):

if code.iat[k, idx\_bjd] == data.iat[i,idx\_js2]:

data.iat[i,idx\_cd] = code.iat[k, idx\_code]

### 세종특별자치시 문제 남아 있음 / 대부분 문제 해결 됨

### 양재\*동 숫자에 \* 처리된 경우 작동 안함

data.head(30)

data.to\_csv('종합체육시설업\_코드작업완료.csv', index=False, encoding='utf-8-sig')

#데이터 전처리

#1. 데이터 확인

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import matplotlib.pyplot as plt

plt.rcParams['font.family'] ='Malgun Gothic'

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

#데이터 불러오기

file\_path = r"C:\Users\NT551\_11TH\Desktop\ML\final\_data\_tap\_dance.csv"

data = pd.read\_csv(file\_path, encoding='cp949')

data.head()

data.info()

#1.1 데이터 분포 확인

import matplotlib.pyplot as plt

# 모든 수치형 데이터의 히스토그램을 확인

numeric\_cols = data.select\_dtypes(include=['int64', 'float64']).columns

# 히스토그램 그리기

fig, axes = plt.subplots(nrows=6, ncols=7, figsize=(20, 15))

axes = axes.flatten()

for i, col in enumerate(numeric\_cols):

ax = axes[i]

data[col].hist(bins=30, ax=ax)

ax.set\_title(col)

plt.tight\_layout()

plt.show()

#왼쪽으로 치우친 변수 찾기

import numpy as np

# 평균과 중앙값의 차이가 큰(=왼쪽 치우친) 변수 찾기

skewed\_cols = []

for col in numeric\_cols:

mean\_value = data[col].mean()

median\_value = data[col].median()

if mean\_value > 2 \* median\_value: # 평균이 중앙값보다 훨씬 크다면 왼쪽 치우침

skewed\_cols.append(col)

# 왼쪽 치우친 변수 목록 출력

skewed\_cols

from scipy.stats import yeojohnson

# 변환 후 저장할 데이터프레임 생성

data\_transformed = data.copy()

# 로그 변환 가능한 변수 (0보다 큰 값만 적용 가능)

log\_transform\_cols = [col for col in skewed\_cols if (data[col] > 0).all()]

# 제곱근 변환 가능한 변수 (0 이상인 경우)

sqrt\_transform\_cols = [col for col in skewed\_cols if (data[col] >= 0).all()]

# Yeo-Johnson 변환 (음수도 포함 가능)

yeo\_johnson\_transform\_cols = skewed\_cols # 모든 데이터에 적용 가능

# 로그 변환 적용

for col in log\_transform\_cols:

data\_transformed[col] = np.log1p(data[col]) # log1p는 log(x+1) 적용

# 제곱근 변환 적용

for col in sqrt\_transform\_cols:

data\_transformed[col] = np.sqrt(data[col])

# Yeo-Johnson 변환 적용 (음수 포함 가능)

for col in yeo\_johnson\_transform\_cols:

data\_transformed[col], \_ = yeojohnson(data[col]) # 들여쓰기 추가 및 변수명 수정

# 변환된 데이터 히스토그램 출력

num\_cols = len(skewed\_cols)

fig, axes = plt.subplots(nrows=(num\_cols // 3) + 1, ncols=3, figsize=(15, num\_cols \* 1.5))

axes = axes.flatten()

for i, col in enumerate(skewed\_cols):

ax = axes[i]

data\_transformed[col].hist(bins=30, ax=ax)

ax.set\_title(f"{col} (변환 후)")

# 남은 빈 그래프 숨기기

for j in range(i + 1, len(axes)):

axes[j].set\_visible(False)

plt.tight\_layout()

plt.show()

#2. 데이터 전처리

#2.1 데이터 합치기

#**식당 관련 업소 총합**

**#숙박 관련 업소 총합**

**#야영장 관련 업소 총합**

**#**2.1.1 식당 관련 업소 총합 - 끊긴 구간 많음

# 식당 관련 업소 합기기

data\_transformed['식당\_총합'] = data\_transformed['식당 수'] + data\_transformed['관광식당 수']

#0 값을 새 칼럼 형성

data\_transformed['식당\_존재여부'] = (data\_transformed['식당\_총합'] > 0).astype(int)

# 0이 최소값의 절반이 되도록 설정

min\_nonzero\_value = data\_transformed['식당\_총합'][data\_transformed['식당\_총합'] > 0].min() / 2

data\_transformed.loc[data\_transformed['식당\_총합'] == 0, '식당\_총합'] = min\_nonzero\_value

# Yeo-Johnson 변환 적용

data\_transformed['식당\_총합\_final'], \_ = yeojohnson(data\_transformed['식당\_총합'])

# 변환 전/후 히스토그램 시각화

fig, axes = plt.subplots(1, 2, figsize=(18, 5))

sns.histplot(data\_transformed['식당\_총합'], bins=30, ax=axes[0], kde=True)

axes[0].set\_title('변환 전: 식당\_총합')

sns.histplot(data\_transformed['식당\_총합\_final'], bins=30, ax=axes[1], kde=True)

axes[1].set\_title('Yeo-Johnson 변환 후: 식당\_총합\_final')

plt.show()

data\_transformed.info()

#'식당 수', '관광식당 수','숙박\_총합','숙박\_총합\_yj', '숙박\_총합 수', '식당\_총합' 변수 제거

data\_transformed.drop(['식당 수', '관광식당 수','숙박\_총합','숙박\_총합\_yj', '숙박\_총합 수', '식당\_총합' ], axis=1, inplace=True)

#2.1.2 숙박 관련 업소 총합

# 숙박 관련 업소 합기기 관광숙박업수, 관광펜션업수, 농어촌민박업수, 한옥체험업수

data\_transformed['숙박\_총합'] = data\_transformed['관광숙박업수'] + data\_transformed['관광펜션업수']

+ data\_transformed['농어촌민박업수'] + data\_transformed['한옥체험업수']

# 숙박\_총합의 분포 파악하기

plt.figure(figsize=(10,6))

sns.histplot(data\_transformed['숙박\_총합'].dropna(), bins=30, kde=True) # 히스토그램과 커널 밀도 추정(KDE) 추가

plt.title("숙박\_총합 Distribution")

plt.xlabel("숙박\_총합")

plt.ylabel("Frequency")

plt.show()

#0 값을 새 칼럼 형성

data\_transformed['숙박\_존재여부'] = (data\_transformed['숙박\_총합'] > 0).astype(int)

# 0이 최소값의 절반이 되도록 설정

min\_nonzero\_value = data\_transformed['숙박\_총합'][data\_transformed['숙박\_총합'] > 0].min() / 2

data\_transformed.loc[data\_transformed['숙박\_총합'] == 0, '숙박\_총합'] = min\_nonzero\_value

# Yeo-Johnson 변환 적용

data\_transformed['숙박\_총합\_final'], \_ = yeojohnson(data\_transformed['숙박\_총합'])

# 변환 전/후 히스토그램 시각화

fig, axes = plt.subplots(1, 2, figsize=(18, 5))

sns.histplot(data\_transformed['숙박\_총합'], bins=30, ax=axes[0], kde=True)

axes[0].set\_title('변환 전: 숙박\_총합')

sns.histplot(data\_transformed['숙박\_총합\_final'], bins=30, ax=axes[1], kde=True)

axes[1].set\_title('Yeo-Johnson 변환 후: 숙박\_총합\_final')

plt.show()

#### #2.1.3 야영장 관련 업소 총합

# 야영장 관련 업소 합기기 일반야영장업수, 자동차야영장업수

data\_transformed['야영\_총합'] = data\_transformed['일반야영장업수'] + data\_transformed['자동차야영장업수']

# 야영\_총합의 분포 파악하기

plt.figure(figsize=(10,6))

sns.histplot(data\_transformed['야영\_총합'].dropna(), bins=30, kde=True) # 히스토그램과 커널 밀도 추정(KDE) 추가

plt.title("야영\_총합 Distribution")

plt.xlabel("야영\_총합")

plt.ylabel("Frequency")

plt.show()

#0 값을 새 칼럼 형성

data\_transformed['야영\_존재여부'] = (data\_transformed['야영\_총합'] > 0).astype(int)

# 0이 최소값의 절반이 되도록 설정

min\_nonzero\_value = data\_transformed['야영\_총합'][data\_transformed['야영\_총합'] > 0].min() / 2

data\_transformed.loc[data\_transformed['야영\_총합'] == 0, '야영\_총합'] = min\_nonzero\_value

# Yeo-Johnson 변환 적용

data\_transformed['야영\_총합\_final'], \_ = yeojohnson(data\_transformed['야영\_총합'])

# 변환 전/후 히스토그램 시각화

fig, axes = plt.subplots(1, 2, figsize=(18, 5))

sns.histplot(data\_transformed['야영\_총합'], bins=30, ax=axes[0], kde=True)

axes[0].set\_title('변환 전: 야영\_총합')

sns.histplot(data\_transformed['야영\_총합\_final'], bins=30, ax=axes[1], kde=True)

axes[1].set\_title('Yeo-Johnson 변환 후: 야영\_총합\_final')

plt.show()

# 로그 변환 추가 (log1p 사용하여 추가 변환)

data\_transformed['야영\_총합\_final1'] = np.log1p(data\_transformed['야영\_총합\_final'])

# 변환 전/후 히스토그램 시각화

fig, axes = plt.subplots(1, 3, figsize=(18, 5))

sns.histplot(data\_transformed['야영\_총합'], bins=30, ax=axes[0], kde=True)

axes[0].set\_title('변환 전: 야영\_총합')

sns.histplot(data\_transformed['야영\_총합\_final'], bins=30, ax=axes[1], kde=True)

axes[1].set\_title('Yeo-Johnson 변환 후: 야영\_총합\_final')

sns.histplot(data\_transformed['야영\_총합\_final1'], bins=30, ax=axes[2], kde=True)

axes[2].set\_title('최종 변환 (YJ + Log1p): 야영\_총합\_final1')

plt.show()

#Yeo-Johnson과 Yeo-Johnson+log 중 어느 것이 더 정확한지 알아보자

# QQ plot으로 정규성 검정

from scipy import stats

stat\_yeo, p\_yeo = stats.shapiro(data\_transformed['야영\_총합\_final'].dropna())

stat\_log, p\_log = stats.shapiro(data\_transformed['야영\_총합\_final1'].dropna())

print(f"Yeo-Johnson 변환: p-value = {p\_yeo}")

print(f"Log 변환: p-value = {p\_log}")

#둘다 0.05 미만이라 정규성은 띠지 않지만, Yeo-Johnson 변환 p-value가 크므로 Yeo-Johnson 채택

#'야영\_총합', '야영\_총합\_log','야영\_총합\_final1'변수 제거

data\_transformed.drop(['야영\_총합', '야영\_총합\_log','야영\_총합\_final1', '일반야영장업수', '자동차야영장업수' ], axis=1, inplace=True)

data\_transformed.info()

#2.1.5 문화시설 총합

data\_transformed['문화시설\_총합'] = df['영화관수'] + df['종합체육시설수'] + df['헬스장수']

# 문화시설\_총합의 분포 파악하기

pl.figure(figsize=(10,6))

sns.histplot(data\_transformed['문화시설\_총합'].dropna(), bins=30, kde=True) # 히스토그램과 커널 밀도 추정(KDE) 추가

plt.title("문화시설\_총합 Distribution")

plt.xlabel("문화시설\_총합")

plt.ylabel("Frequency")

plt.show()

#0 값을 새 칼럼 형성

data\_transformed['문화시설\_존재여부'] = (data\_transformed['문화시설\_총합'] > 0).astype(int)

# 0이 최소값의 절반이 되도록 설정

min\_nonzero\_value = data\_transformed['문화시설\_총합'][data\_transformed['문화시설\_총합'] > 0].min() / 2

data\_transformed.loc[data\_transformed['문화시설\_총합'] == 0, '문화시설\_총합'] = min\_nonzero\_value

# Yeo-Johnson 변환 적용

data\_transformed['문화시설\_총합\_final'], \_ = yeojohnson(data\_transformed['문화시설\_총합'])

# 변환 전/후 히스토그램 시각화

fig, axes = plt.subplots(1, 2, figsize=(18, 5))

sns.histplot(data\_transformed['문화시설\_총합'], bins=30, ax=axes[0], kde=True)

axes[0].set\_title('변환 전: 문화시설\_총합')

sns.histplot(data\_transformed['문화시설\_총합\_final'], bins=30, ax=axes[1], kde=True)

axes[1].set\_title('Yeo-Johnson 변환 후: 문화시설\_총합\_final')

plt.show()

#변수 제거

data\_transformed.drop(['문화시설\_총합','종합체육시설수', '영화관수', '헬스장수'], axis=1, inplace=True)

data\_transformed.info()

#산후조리업은 0값만 낮추기

# 0이 최소값의 절반이 되도록 설정

min\_nonzero\_value = data\_transformed['산후조리업수'][data\_transformed['산후조리업수'] > 0].min() / 2

data\_transformed.loc[data\_transformed['산후조리업수'] == 0, '산후조리업수'] = min\_nonzero\_value

# Yeo-Johnson 변환 적용

data\_transformed['산후조리업수\_final'], \_ = yeojohnson(data\_transformed['산후조리업수'])

# 산후조리업 변환 후 분포 파악하기

plt.figure(figsize=(10,6))

sns.histplot(data\_transformed['산후조리업수\_final'].dropna(), bins=30, kde=True) # 히스토그램과 커널 밀도 추정(KDE) 추가

plt.title("산후조리업수\_final Distribution")

plt.xlabel("산후조리업수")

plt.ylabel("Frequency")

plt.show()

#변환 해도 끊긴 구간이 많으니 범주형으로 변환

q1 = data\_transformed["산후조리업수\_final"].quantile(0.33)

q2 = data\_transformed["산후조리업수\_final"].quantile(0.66)

print(f"33% 분위수(q1): {q1}")

print(f"66% 분위수(q2): {q2}")

import matplotlib.pyplot as plt

# '산후조리업수\_final'을 숫자로 변환

data\_transformed["산후조리업수\_final"] = pd.to\_numeric(data\_transformed["산후조리업수\_final"], errors='coerce')

# 33%와 66% 분위수 계산

q1 = data\_transformed["산후조리업수\_final"].quantile(0.33)

q2 = data\_transformed["산후조리업수\_final"].quantile(0.66)

# 분위수 기준으로 구간 나누기

data\_transformed["산후조리업수\_구간"] = pd.cut(

data\_transformed["산후조리업수\_final"],

bins=[-float("inf"), q1, q2, float("inf")],

labels=['없음','보통','많음']

)

# 히스토그램 시각화

plt.hist(data\_transformed["산후조리업수\_final"], bins=10, alpha=0.7, edgecolor="black")

plt.axvline(q1, color="red", linestyle="dashed", label="33% 분위수")

plt.axvline(q2, color="blue", linestyle="dashed", label="66% 분위수")

plt.legend()

plt.title("산후조리업수\_final 분포 및 분위수 경계")

plt.xlabel("산후조리업수\_final 값")

plt.ylabel("빈도수")

plt.show()

data\_transformed.head()

data\_transformed.info()

data\_transformed.drop(["산후조리업수","관광숙박업수","관광펜션업수", "농어촌민박업수", "한옥체험업수",

"교육시설\_총합\_final"], axis=1, inplace=True)

sc = data[selected\_columns].copy() # 새로운 데이터프레임 생성

# 인덱스 리셋 후 병합

data\_transformed = pd.concat([data\_transformed.reset\_index(drop=True), sc.reset\_index(drop=True)], axis=1)

data\_transformed.head()

data\_transformed.info()

# '대학교 수' 컬럼을 float로 변환 (int -> float)

data\_transformed['대학교 수'] = data\_transformed['대학교 수'].astype(float)

# 0이 최소값의 절반이 되도록 설정

min\_nonzero\_value = data\_transformed['대학교 수'][data\_transformed['대학교 수'] > 0].min() / 2

data\_transformed.loc[data\_transformed['대학교 수'] == 0, '대학교 수'] = min\_nonzero\_value

# Yeo-Johnson 변환 적용

data\_transformed.loc[:, '대학교 수'] = yeojohnson(data\_transformed['대학교 수'])[0]

# 변환 후 히스토그램 시각화

fig, axes = plt.subplots(figsize=(10, 5)) # 단일 플롯

sns.histplot(data\_transformed['대학교 수'], bins=30, ax=axes, kde=True)

axes.set\_title('변환 후: 대학교 수')

plt.show()

data\_transformed.drop('학원수', axis=1, inplace=True)

data\_transformed.info()

final\_df\_path = r"C:\Users\NT551\_11TH\Desktop\ML\data\_transformed.csv"

a\_path = r"C:\Users\NT551\_11TH\Desktop\ML\data\_merge용.csv"

output\_path = r"C:\Users\NT551\_11TH\Desktop\ML\data\_merged.csv"

# CSV 파일 불러오기

df\_transformed = pd.read\_csv(final\_df\_path, encoding="cp949")

df\_merge = pd.read\_csv(a\_path, encoding="cp949")

# 법정동코드 기준으로 left join 수행

df\_merged = pd.merge(df\_transformed, df\_merge, on="법정동코드", how="left")

# 결과 저장

df\_merged.to\_csv(output\_path, index=False, encoding="cp949")

#EDA

#1. 데이터 확인

! gdown 1U1M2ZRfKsrZEwENCBFRC0cMfPwaoLASF

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

data = pd.read\_csv('/content/final.csv', encoding='cp949')

data.info()

data.head()

data.describe()

#전반적인 분포 확인

data.hist(figsize=(15, 10))

#X특징들 설명은 추후에 추가

#데이터의 기술통계량 파악

print(data.describe())

#모델링

#lgbm

!pip install scikit-learn==1.3.0

!pip install optuna

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score

from sklearn.preprocessing import LabelEncoder

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

from lightgbm import LGBMRegressor

import optuna

from xgboost import XGBRegressor

from optuna.samplers import TPESampler

data = pd.read\_csv('/content/fffffffffff.csv', encoding='cp949')

data.head()

data['산후조리업수\_구간'].unique()

data.set\_index('법정동코드', inplace=True)

data.columns

le = LabelEncoder()

data['산후조리업수\_구간'] = le.fit\_transform(data['산후조리업수\_구간'])

data.info()

# X, y 나누기

y\_model = data.loc[:,'y']

X\_model = data.drop('y', axis=1)

X\_model.head()

y\_model.head()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_model, y\_model, test\_size=0.2, random\_state=42)

kfold = KFold(n\_splits=3, shuffle=True, random\_state=42)

base\_lgb = LGBMRegressor(random\_state=42, n\_jobs=-1)

score = cross\_val\_score(base\_lgb, X\_train, y\_train, cv = kfold, scoring = 'neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

base\_lgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, base\_lgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, base\_lgb.predict(X\_test)))

print("\n🔥 LightGBM 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = base\_lgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

# Optuna 목적 함수 정의

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 50, 1000),

'max\_depth': trial.suggest\_int('max\_depth', 3, 30),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.5),

'subsample': trial.suggest\_float('subsample', 0.3, 1.0),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.5, 1.0),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.0, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.0, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train.values.ravel(), cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=1000)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

study\_lgb.best\_value

optuna.visualization.plot\_optimization\_history(study\_lgb)

optuna.visualization.plot\_parallel\_coordinate(study\_lgb)

optuna.visualization.plot\_param\_importances(study\_lgb)

# Optuna 목적 함수 정의

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 300, 1000),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.5),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.5, 1.0),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.0, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.0, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train.values.ravel(), cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=700)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

study\_lgb.best\_value

# 성능이 썩 맘에 들지 않음..

# L1 규제를 너무 심하게 넣었나??

# Optuna 목적 함수 정의

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 300, 1000),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.5),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.5, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.0, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train.values.ravel(), cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=100)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

study\_lgb.best\_value

optuna.visualization.plot\_optimization\_history(study\_lgb)

optuna.visualization.plot\_parallel\_coordinate(study\_lgb)

optuna.visualization.plot\_param\_importances(study\_lgb)

# 규제를 빼도 뭔가 나아지지 않는다.. 뭐가 문제일까..

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 400, 1000),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.2, 1.0),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.4, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.4, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=300)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

study\_lgb.best\_value

# 오 성능이 많이 올랐어

# 주로 만진건 colsample\_min

# 조금 더 만져보자

# 규제를 빼도 뭔가 나아지지 않는다.. 뭐가 문제일까..

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 400, 1000),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.1, 0.5),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.4, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.4, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=200)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

best\_lgb = LGBMRegressor(\*\*best\_lgb\_params)

best\_lgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_lgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_lgb.predict(X\_test)))

print("\n🔥 LightGBM 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_lgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

study\_lgb.best\_params

# 마지막으로 범위 좁혀서 튜닝

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 800, 1200),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.1, 0.5),

# 'max\_depth': trial.suggest\_int('max\_depth', 3, 25),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.4, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.4, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize", sampler=TPESampler())

study\_lgb.optimize(objective\_lgb, n\_trials=200)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

best\_lgb = LGBMRegressor(\*\*best\_lgb\_params)

best\_lgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_lgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_lgb.predict(X\_test)))

print("\n🔥 LightGBM 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_lgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

study\_lgb.best\_value

study\_lgb.best\_value

study\_lgb.best\_params

# 성능 좋으니 일단 저장

import joblib

joblib.dump(best\_lgb, 'best\_lgb.pkl')

# 성능은 굉장히 뛰어나나 과적합 위험이 높아보인다

# k-fold와 여러가지 규약들을 걸었지만

# n\_estimator은 너무 많고, colsample\_bytree가 너무 낮아서 특성 feature에만 의존할 가능성 높음..

# 조금 더 튜닝해보자

# max\_depth를 통해 조금 더 과적합 방지함과 동시에 colsample\_bytree를 높이기

# 마지막으로 범위 좁혀서 튜닝

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 800, 1200),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.3, 1),

'max\_depth': trial.suggest\_int('max\_depth', 3, 25),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.4, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.4, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=200)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

best\_lgb = LGBMRegressor(\*\*best\_lgb\_params)

best\_lgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_lgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_lgb.predict(X\_test)))

print("\n🔥 LightGBM 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_lgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

# 성능이 너무 떨어지는데... max\_depth를 빼고 colsample만 살짝 낮춰서 다시..

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 800, 1200),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.3, 0.7),

# 'max\_depth': trial.suggest\_int('max\_depth', 3, 25),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.4, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.4, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=200)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

best\_lgb = LGBMRegressor(\*\*best\_lgb\_params)

best\_lgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_lgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_lgb.predict(X\_test)))

print("\n🔥 LightGBM 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_lgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

# feature importance 높은거만 추려서 다시 돌려볼까

model = LGBMRegressor(random\_state=42, njobs=-1)

model.fit(X\_train, y\_train)

# Feature Importance 출력

feature\_importance = pd.DataFrame({

'Feature': X\_train.columns,

'Importance': model.feature\_importances\_

}).sort\_values(by="Importance", ascending=False)

# 상위 n개 변수 선택

top\_features = feature\_importance['Feature'][:30] # 중요도 높은 30개 선택

X\_train\_selected = X\_train[top\_features]

X\_test\_selected = X\_test[top\_features]

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 600, 1200),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.3, 0.7),

'max\_depth': trial.suggest\_int('max\_depth', 3, 25),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.4, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.4, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train\_selected, y\_train, cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=300)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

best\_lgb = LGBMRegressor(\*\*best\_lgb\_params)

best\_lgb.fit(X\_train\_selected, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_lgb.predict(X\_train\_selected)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_lgb.predict(X\_test\_selected)))

print("\n🔥 LightGBM 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_lgb.predict(X\_test\_selected)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

X\_train\_selected.columns

# 법정동 코드 안 뺐네..

# 다시해..

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 800, 1200),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.3, 0.7),

# 'max\_depth': trial.suggest\_int('max\_depth', 3, 25),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.4, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.4, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize")

study\_lgb.optimize(objective\_lgb, n\_trials=200)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

best\_lgb = LGBMRegressor(\*\*best\_lgb\_params)

best\_lgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_lgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_lgb.predict(X\_test)))

print("\n🔥 LightGBM 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_lgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

# 마지막으로 범위 좁혀서 튜닝

def objective\_lgb(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 800, 1200),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.1, 0.5),

# 'max\_depth': trial.suggest\_int('max\_depth', 3, 25),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.4, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.4, 1.0)

}

model = LGBMRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold,

scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_lgb = optuna.create\_study(direction="minimize", sampler = TPESampler())

study\_lgb.optimize(objective\_lgb, n\_trials=350)

best\_lgb\_params = study\_lgb.best\_params

print("Best LightGBM Parameters:", best\_lgb\_params)

best\_lgb = LGBMRegressor(\*\*best\_lgb\_params)

best\_lgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_lgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_lgb.predict(X\_test)))

print("\n🔥 LightGBM 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_lgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

study\_lgb.best\_value

study\_lgb.best\_params

joblib.dump(best\_lgb,'best\_lgb.pkl')

#XGBOOST

base\_xgb = XGBRegressor(random\_state=42, n\_jobs=-1)

base\_xgb.fit(X\_train,y\_train)

score = cross\_val\_score(base\_xgb, X\_train, y\_train, cv = kfold, scoring = 'neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

-score

def objective\_xgb(trial):

params = {

# 트리 모델의 개수

#'n\_estimators': trial.suggest\_int('n\_estimators', 50, 300), # 'n\_estimators': 219

#'n\_estimators': trial.suggest\_int('n\_estimators', 150, 1000),

#'n\_estimators': trial.suggest\_int('n\_estimators', 100, 600),

#'n\_estimators': trial.suggest\_int('n\_estimators', 300, 500),

#'n\_estimators': trial.suggest\_int('n\_estimators', 150, 250),

'n\_estimators': trial.suggest\_int('n\_estimators', 50, 1000),

# 학습 단계 별로 이전 결과를 얼마나 반영할지 설정 / 일반적으로 0.01 ~ 0.2

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3), # 'learning\_rate': 0.057466159718714896

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.03, 0.2),

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.04, 0.12),

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.045, 0.08),

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.05, 0.065),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.5),

'min\_child\_weight': trial.suggest\_float('min\_child\_weight', 1, 20),

# 트리의 최대 깊이, 일반적으로 3~10

#'max\_depth': trial.suggest\_int('max\_depth', 3, 15), # 'max\_depth': 3

#'max\_depth': trial.suggest\_int('max\_depth', 2, 8),

#'max\_depth': trial.suggest\_int('max\_depth', 2, 5),

#'max\_depth': trial.suggest\_int('max\_depth', 2, 4),

#'max\_depth': trial.suggest\_int('max\_depth', 3, 3),

'max\_depth': trial.suggest\_int('max\_depth', 3, 30),

'gamma': trial.suggest\_int('gamma', 0, 1),

# 각 트리마다 데이터 샘플링 비율 / OF 방지, 일반적으로 0.5 ~ 1.0

#'subsample': trial.suggest\_float('subsample', 0.7, 1.0), # 'subsample': 0.7306799200046683

#'subsample': trial.suggest\_float('subsample', 0.5, 0.8),

#'subsample': trial.suggest\_float('subsample', 0.45, 0.65),

#'subsample': trial.suggest\_float('subsample', 0.4, 0.6),

#'subsample': trial.suggest\_float('subsample', 0.5, 0.75),

'subsample': trial.suggest\_float('subsample', 0.3, 1.0),

# 각 트리마다 피처 샘플링 비율, 일반적으로 0.5 ~ 1.0

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.7, 1.0), # 'colsample\_bytree': 0.9189039309883307

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.6, 0.9),

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.65, 0.75),

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.6, 0.7),

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.85, 0.95),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.3, 1.0),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.0, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.0, 1.0),

}

model = XGBRegressor(random\_state=42, \*\*params)

# RMSE를 계산하기 위해 MSE를 구한 후 제곱근을 취합니다

mse = -cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='neg\_root\_mean\_squared\_error').mean()

return mse # Optuna는 기본적으로 최소화하므로, RMSE를 그대로 반환합니다

# Optuna 실행

study\_xgb = optuna.create\_study(direction="minimize") # R2 : maximize / RMSE minimize

study\_xgb.optimize(objective\_xgb, n\_trials=300, show\_progress\_bar=True, gc\_after\_trial=True, catch=(Exception,))

best\_xgb\_params = study\_xgb.best\_params

print("Best XGBoost Parameters:", best\_xgb\_params)

study\_xgb.best\_value

optuna.visualization.plot\_optimization\_history(study\_xgb)

optuna.visualization.plot\_parallel\_coordinate(study\_xgb)

optuna.visualization.plot\_param\_importances(study\_xgb)

best\_xgb = XGBRegressor(\*\*best\_xgb\_params)

best\_xgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_xgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_xgb.predict(X\_test)))

print("\n🔥 XGB 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_xgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

def objective\_xgb(trial):

params = {

# 트리 모델의 개수

#'n\_estimators': trial.suggest\_int('n\_estimators', 50, 300), # 'n\_estimators': 219

#'n\_estimators': trial.suggest\_int('n\_estimators', 150, 1000),

#'n\_estimators': trial.suggest\_int('n\_estimators', 100, 600),

#'n\_estimators': trial.suggest\_int('n\_estimators', 300, 500),

#'n\_estimators': trial.suggest\_int('n\_estimators', 150, 250),

'n\_estimators': trial.suggest\_int('n\_estimators', 400, 1000),

# 학습 단계 별로 이전 결과를 얼마나 반영할지 설정 / 일반적으로 0.01 ~ 0.2

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3), # 'learning\_rate': 0.057466159718714896

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.03, 0.2),

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.04, 0.12),

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.045, 0.08),

#'learning\_rate': trial.suggest\_float('learning\_rate', 0.05, 0.065),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'min\_child\_weight': trial.suggest\_float('min\_child\_weight', 1, 15),

# 트리의 최대 깊이, 일반적으로 3~10

#'max\_depth': trial.suggest\_int('max\_depth', 3, 15), # 'max\_depth': 3

#'max\_depth': trial.suggest\_int('max\_depth', 2, 8),

#'max\_depth': trial.suggest\_int('max\_depth', 2, 5),

#'max\_depth': trial.suggest\_int('max\_depth', 2, 4),

#'max\_depth': trial.suggest\_int('max\_depth', 3, 3),

'max\_depth': trial.suggest\_int('max\_depth', 3, 10),

'gamma': trial.suggest\_int('gamma', 0, 1),

# 각 트리마다 데이터 샘플링 비율 / OF 방지, 일반적으로 0.5 ~ 1.0

#'subsample': trial.suggest\_float('subsample', 0.7, 1.0), # 'subsample': 0.7306799200046683

#'subsample': trial.suggest\_float('subsample', 0.5, 0.8),

#'subsample': trial.suggest\_float('subsample', 0.45, 0.65),

#'subsample': trial.suggest\_float('subsample', 0.4, 0.6),

#'subsample': trial.suggest\_float('subsample', 0.5, 0.75),

'subsample': trial.suggest\_float('subsample', 0.3, 0.7),

# 각 트리마다 피처 샘플링 비율, 일반적으로 0.5 ~ 1.0

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.7, 1.0), # 'colsample\_bytree': 0.9189039309883307

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.6, 0.9),

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.65, 0.75),

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.6, 0.7),

#'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.85, 0.95),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.3, 1.0),

'reg\_alpha': trial.suggest\_float('reg\_alpha', 0.0, 1.0),

'reg\_lambda': trial.suggest\_float('reg\_lambda', 0.0, 1.0),

}

model = XGBRegressor(random\_state=42, \*\*params)

# RMSE를 계산하기 위해 MSE를 구한 후 제곱근을 취합니다

mse = -cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return mse # Optuna는 기본적으로 최소화하므로, RMSE를 그대로 반환합니다

# Optuna 실행

study\_xgb = optuna.create\_study(direction="minimize") # R2 : maximize / RMSE minimize

study\_xgb.optimize(objective\_xgb, n\_trials=50, show\_progress\_bar=True, gc\_after\_trial=True, catch=(Exception,))

best\_xgb\_params = study\_xgb.best\_params

print("Best XGBoost Parameters:", best\_xgb\_params)

study\_xgb.best\_value

best\_xgb = XGBRegressor(\*\*best\_xgb\_params)

best\_xgb.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, best\_xgb.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, best\_xgb.predict(X\_test)))

print("\n🔥 XGB 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = best\_xgb.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

#RandomForest

from sklearn.ensemble import RandomForestRegressor

# 랜덤포레스트 모델 훈련

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

pd.set\_option("display.max\_rows", None)

# 변수 중요도 출력

importance = pd.DataFrame({"Feature": X\_train.columns, "Importance": rf.feature\_importances\_})

importance = importance.sort\_values(by="Importance", ascending=False)

print(importance)

# ===============================

# 🔥 Random Forest 하이퍼파라미터 튜닝 (Optuna)

# ===============================

def objective\_rf(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 50, 300),

'max\_depth': trial.suggest\_int('max\_depth', 5, 30),

'min\_samples\_split': trial.suggest\_int('min\_samples\_split', 2, 10),

'min\_samples\_leaf': trial.suggest\_int('min\_samples\_leaf', 1, 5),

}

model = RandomForestRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_rf = optuna.create\_study(direction="minimize", sampler = TPESampler())

study\_rf.optimize(objective\_rf, n\_trials=300)

best\_rf\_params = study\_rf.best\_params

print("Best Random Forest Parameters:", best\_rf\_params)

# 최적의 Random Forest 모델 생성

rf\_best = RandomForestRegressor(random\_state=42, \*\*best\_rf\_params)

# 모델 학습

rf\_best.fit(X\_train, y\_train)

# 학습 및 테스트 R² 점수

train\_score = rf\_best.score(X\_train, y\_train)

test\_score = rf\_best.score(X\_test, y\_test)

print("\n🔥 Random Forest 최종 성능:")

print(f" - 학습 데이터 R²: {train\_score:.4f}")

print(f" - 테스트 데이터 R²: {test\_score:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = rf\_best.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 MSE: {mse\_test:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

# ===============================

# 🔥 Random Forest 하이퍼파라미터 튜닝 (Optuna)

# ===============================

def objective\_rf(trial):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 50, 300),

'max\_depth': trial.suggest\_int('max\_depth', 5, 30),

'min\_samples\_split': trial.suggest\_int('min\_samples\_split', 2, 10),

'min\_samples\_leaf': trial.suggest\_int('min\_samples\_leaf', 1, 5),

}

model = RandomForestRegressor(random\_state=42, \*\*params)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1).mean()

return -score

# Optuna 실행

study\_rf = optuna.create\_study(direction="minimize", sampler = TPESampler())

study\_rf.optimize(objective\_rf, n\_trials=300)

best\_rf\_params = study\_rf.best\_params

print("Best Random Forest Parameters:", best\_rf\_params)

# # 최적의 Random Forest 모델 생성

rf\_best = RandomForestRegressor(random\_state=42, \*\*best\_rf\_params)

# # 모델 학습

rf\_best.fit(X\_train, y\_train)

# # 학습 및 테스트 R² 점수

# train\_score = rf\_best.score(X\_train, y\_train)

# test\_score = rf\_best.score(X\_test, y\_test)

# print("\n🔥 Random Forest 최종 성능:")

# print(f" - 학습 데이터 R²: {train\_score:.4f}")

# print(f" - 테스트 데이터 R²: {test\_score:.4f}")

# # 예측 및 평가 지표 계산

# y\_test\_pred = rf\_best.predict(X\_test)

# mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

# mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)

# r2\_test = r2\_score(y\_test, y\_test\_pred)

# print("\n📊 평가 지표:")

# print(f" - 테스트 MAE: {mae\_test:.4f}")

# print(f" - 테스트 MSE: {mse\_test:.4f}")

# print(f" - 테스트 R²: {r2\_test:.4f}")

rf\_best = RandomForestRegressor(random\_state=42, \*\*best\_rf\_params)

rf\_best.fit(X\_train, y\_train)

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, rf\_best.predict(X\_train)))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, rf\_best.predict(X\_test)))

print("\n🔥 RF 최종 성능:")

print(f" - 학습 데이터 RMSE: {train\_rmse:.4f}")

print(f" - 테스트 데이터 RMSE: {test\_rmse:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = rf\_best.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 RMSE: {test\_rmse:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

#KNN

from google.colab import drive

drive.mount('/content/drive')

!pip install scikit-learn==1.3.0

!pip install optuna

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import optuna

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score

from sklearn.neighbors import KNeighborsRegressor

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

# ===============================

# 데이터 로드 및 전처리

# ===============================

# 데이터 불러오기

data = pd.read\_csv("/content/drive/MyDrive/ASAC\_빅데이터\_7기/6\_ML조별프로젝트/MLproject/final.csv", encoding='cp949')

data.set\_index('법정동코드', inplace=True)

# 범주형 데이터 라벨 인코딩

categorical\_cols = data.select\_dtypes(include=['object']).columns # 범주형 컬럼 찾기

label\_encoders = {}

for col in categorical\_cols:

le = LabelEncoder()

data[col] = le.fit\_transform(data[col])

label\_encoders[col] = le # 나중에 디코딩할 수 있도록 저장

# X, y 분할

X = data.drop(columns=['y'])

y = data['y']

# 데이터 스케일링 (KNN은 거리 기반이므로 중요)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# 데이터 분할

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# ===============================

# K-Fold 설정 및 Optuna 튜닝

# ===============================

kfold = KFold(n\_splits=3, shuffle=True, random\_state=42)

def objective\_knn(trial):

params = {

'n\_neighbors': trial.suggest\_int('n\_neighbors', 1, 50),

'weights': trial.suggest\_categorical('weights', ['uniform', 'distance']),

'p': trial.suggest\_int('p', 1, 3) # Manhattan(1), Euclidean(2), Minkowski(3)

}

model = KNeighborsRegressor(\*\*params)

# 교차 검증 점수 계산

score = cross\_val\_score(model, X\_train, y\_train.values.ravel(), cv=kfold, scoring='neg\_mean\_squared\_error').mean()

return -score # MSE는 음수로 반환되므로 부호 반전

# Optuna 실행

study\_knn = optuna.create\_study(direction="maximize")

study\_knn.optimize(objective\_knn, n\_trials=100) # n\_trials를 적절히 조정 가능

best\_knn\_params = study\_knn.best\_params

print("Best KNN Parameters:", best\_knn\_params)

# ===============================

# 최적 모델 학습 및 평가

# ===============================

knn\_best = KNeighborsRegressor(\*\*best\_knn\_params)

knn\_best.fit(X\_train, y\_train)

# 학습 및 테스트 R² 점수

train\_score = knn\_best.score(X\_train, y\_train)

test\_score = knn\_best.score(X\_test, y\_test)

print("\n🔥 KNN 최종 성능:")

print(f" - 학습 데이터 R²: {train\_score:.4f}")

print(f" - 테스트 데이터 R²: {test\_score:.4f}")

# 예측 및 평가 지표 계산

y\_test\_pred = knn\_best.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 MSE: {mse\_test:.4f}")

print(f" - 테스트 R²: {r2\_test:.4f}")

# ===============================

# 결과 시각화 (예측 vs 실제값)

# ===============================

plt.figure(figsize=(8, 6))

plt.scatter(y\_test, y\_test\_pred, alpha=0.6)

plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='--')

plt.xlabel("Actual")

plt.ylabel("Predicted")

plt.title("Actual vs Predicted")

plt.show()

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import optuna

from optuna.visualization.matplotlib import plot\_optimization\_history, plot\_parallel\_coordinate, plot\_param\_importances

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score

from sklearn.neighbors import KNeighborsRegressor

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from sklearn.preprocessing import OneHotEncoder

from sklearn.pipeline import Pipeline

# ===============================

# 데이터 로드 및 전처리

# ===============================

# 데이터 불러오기

data = pd.read\_csv("/content/drive/MyDrive/ASAC\_빅데이터\_7기/6\_ML조별프로젝트/MLproject/final.csv", encoding='cp949')

data.set\_index('법정동코드', inplace=True)

# OneHotEncoder

encoder = OneHotEncoder(sparse=False)

encoded\_categorical\_data = encoder.fit\_transform(data[categorical\_cols])

encoded\_df = pd.DataFrame(encoded\_categorical\_data, columns=encoder.get\_feature\_names\_out(categorical\_cols))

# 기존 데이터를 업데이트

data = data.drop(columns=categorical\_cols).reset\_index(drop=True)

data = pd.concat([data, encoded\_df], axis=1)

# 데이터 스케일링 (KNN은 거리 기반이므로 중요)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# X, y 분할

X = data.drop(columns=['y'])

y = data['y']

pipeline = Pipeline([

('scaler', StandardScaler()),

('knn', KNeighborsRegressor(\*\*best\_knn\_params))

])

pipeline.fit(X\_train, y\_train)

y\_pred = pipeline.predict(X\_test)

rmse\_pipeline = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(f"Pipeline 테스트 RMSE: {rmse\_pipeline:.4f}")

# 데이터 분할

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# ===============================

# K-Fold 설정 및 Optuna 튜닝

# ===============================

kfold = KFold(n\_splits=3, shuffle=True, random\_state=42)

def objective\_knn(trial):

params = {

'n\_neighbors': trial.suggest\_int('n\_neighbors', 7, 121),

'weights': trial.suggest\_categorical('weights', ['uniform']),

'p': trial.suggest\_int('p', 1, 2)

}

model = KNeighborsRegressor(\*\*params)

# 교차 검증 점수 계산 (R² 사용)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='r2').mean()

return score

# Optuna 실행

study\_knn = optuna.create\_study(direction="maximize")

study\_knn.optimize(objective\_knn, n\_trials=500) # n\_trials를 적절히 조정 가능

best\_knn\_params = study\_knn.best\_params

print("Best KNN Parameters:", best\_knn\_params)

# ===============================

# 최적 모델 학습 및 평가

# ===============================

knn\_best = KNeighborsRegressor(\*\*best\_knn\_params)

knn\_best.fit(X\_train, y\_train)

# 학습 데이터 예측 및 평가 지표 계산

y\_train\_pred = knn\_best.predict(X\_train)

rmse\_train = np.sqrt(mean\_squared\_error(y\_train, y\_train\_pred)) # 학습 데이터 RMSE 계산

# 기존 테스트 데이터 RMSE 계산

rmse\_test = np.sqrt(mean\_squared\_error(y\_test, y\_test\_pred))

# 결과 출력

print("\n📊 평가 지표:")

print(f" - 학습 RMSE: {rmse\_train:.4f}") # 학습 RMSE 출력

print(f" - 테스트 RMSE: {rmse\_test:.4f}") # 테스트 RMSE 출력

# 예측 및 평가 지표 계산

y\_test\_pred = knn\_best.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)

rmse\_test = np.sqrt(mse\_test) # RMSE 계산

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 MSE: {mse\_test:.4f}")

print(f" - 테스트 RMSE: {rmse\_test:.4f}") # RMSE 출력

print(f" - 테스트 R²: {r2\_test:.4f}")

# ===============================

# 결과 시각화 (예측 vs 실제값)

# ===============================

plt.figure(figsize=(8, 6))

plt.scatter(y\_test, y\_test\_pred, alpha=0.6)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linestyle='--')

plt.xlabel("Actual")

plt.ylabel("Predicted")

plt.title("Actual vs Predicted")

plt.show()

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import optuna

from optuna.visualization.matplotlib import plot\_optimization\_history, plot\_parallel\_coordinate, plot\_param\_importances

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score

from sklearn.neighbors import KNeighborsRegressor

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from sklearn.preprocessing import OneHotEncoder

from sklearn.pipeline import Pipeline

# ===============================

# 데이터 로드 및 전처리

# ===============================

# 데이터 불러오기

data = pd.read\_csv("/content/drive/MyDrive/ASAC\_빅데이터\_7기/6\_ML조별프로젝트/MLproject/final.csv", encoding='cp949')

data.set\_index('법정동코드', inplace=True)

# 범주형 데이터 라벨 인코딩

categorical\_cols = data.select\_dtypes(include=['object']).columns # 범주형 컬럼 찾기

label\_encoders = {}

for col in categorical\_cols:

le = LabelEncoder()

data[col] = le.fit\_transform(data[col])

label\_encoders[col] = le # 나중에 디코딩할 수 있도록 저장

# OneHotEncoder

encoder = OneHotEncoder(sparse=False)

encoded\_categorical\_data = encoder.fit\_transform(data[categorical\_cols])

encoded\_df = pd.DataFrame(encoded\_categorical\_data, columns=encoder.get\_feature\_names\_out(categorical\_cols))

# 기존 데이터를 업데이트

data = data.drop(columns=categorical\_cols).reset\_index(drop=True)

data = pd.concat([data, encoded\_df], axis=1)

# X, y 분할

X = data.drop(columns=['y'])

y = data['y']

pipeline = Pipeline([

('scaler', StandardScaler()),

('knn', KNeighborsRegressor(\*\*best\_knn\_params))

])

pipeline.fit(X\_train, y\_train)

y\_pred = pipeline.predict(X\_test)

rmse\_pipeline = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(f"Pipeline 테스트 RMSE: {rmse\_pipeline:.4f}")

# 데이터 스케일링 (KNN은 거리 기반이므로 중요)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# 데이터 분할

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# ===============================

# K-Fold 설정 및 Optuna 튜닝

# ===============================

kfold = KFold(n\_splits=3, shuffle=True, random\_state=42)

def objective\_knn(trial):

params = {

'n\_neighbors': trial.suggest\_int('n\_neighbors', 7, 121),

'weights': trial.suggest\_categorical('weights', ['uniform']),

'p': trial.suggest\_int('p', 1, 2)

}

model = KNeighborsRegressor(\*\*params)

# 교차 검증 점수 계산 (R² 사용)

score = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='r2').mean()

return score

# Optuna 실행

study\_knn = optuna.create\_study(direction="maximize")

study\_knn.optimize(objective\_knn, n\_trials=500) # n\_trials를 적절히 조정 가능

best\_knn\_params = study\_knn.best\_params

print("Best KNN Parameters:", best\_knn\_params)

# ===============================

# 최적 모델 학습 및 평가

# ===============================

knn\_best = KNeighborsRegressor(\*\*best\_knn\_params)

knn\_best.fit(X\_train, y\_train)

# 학습 데이터 예측 및 평가 지표 계산

y\_train\_pred = knn\_best.predict(X\_train)

rmse\_train = np.sqrt(mean\_squared\_error(y\_train, y\_train\_pred)) # 학습 데이터 RMSE 계산

# 기존 테스트 데이터 RMSE 계산

rmse\_test = np.sqrt(mean\_squared\_error(y\_test, y\_test\_pred))

# 결과 출력

print("\n📊 평가 지표:")

print(f" - 학습 RMSE: {rmse\_train:.4f}") # 학습 RMSE 출력

print(f" - 테스트 RMSE: {rmse\_test:.4f}") # 테스트 RMSE 출력

# 예측 및 평가 지표 계산

y\_test\_pred = knn\_best.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)

rmse\_test = np.sqrt(mse\_test) # RMSE 계산

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 MSE: {mse\_test:.4f}")

print(f" - 테스트 RMSE: {rmse\_test:.4f}") # RMSE 출력

print(f" - 테스트 R²: {r2\_test:.4f}")

# ===============================

# 결과 시각화 (예측 vs 실제값)

# ===============================

plt.figure(figsize=(8, 6))

plt.scatter(y\_test, y\_test\_pred, alpha=0.6)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linestyle='--')

plt.xlabel("Actual")

plt.ylabel("Predicted")

plt.title("Actual vs Predicted")

plt.show()

#SVR

!pip install scikit-learn==1.3.0

!pip install optuna

import pandas as pd

import numpy as np

import optuna

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

from sklearn.svm import SVR

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

# 데이터 불러오기

data = pd.read\_csv('/content/final.csv', encoding='cp949')

data.set\_index('법정동코드', inplace=True)

# 범주형 데이터 라벨 인코딩

categorical\_cols = data.select\_dtypes(include=['object']).columns

label\_encoders = {}

for col in categorical\_cols:

le = LabelEncoder()

data[col] = le.fit\_transform(data[col])

label\_encoders[col] = le

# X, y 분할

X = data.drop(columns=['y'])

y = data[['y']].astype(float)

# 🔹 데이터 정규화 (SVR에 필요)

scaler = StandardScaler()

X = pd.DataFrame(scaler.fit\_transform(X), columns=X.columns)

# 데이터 분할

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# K-Fold 설정

kfold = KFold(n\_splits=3, shuffle=True, random\_state=42)

# ===============================

# 🔥 SVR 하이퍼파라미터 튜닝 (Optuna, RMSE 최소화)

# ===============================

def objective\_svr(trial):

params = {

'C': trial.suggest\_float('C', 0.01, 50.0, log=True), # 더 넓은 범위

'epsilon': trial.suggest\_float('epsilon', 0.5, 1.0),

'kernel': trial.suggest\_categorical('kernel', ['linear', 'rbf', 'poly', 'sigmoid']),

}

if params['kernel'] == 'poly':

params['degree'] = trial.suggest\_int('degree', 2, 7) # 다항식 차수 추가

model = SVR(\*\*params)

# K-Fold 교차 검증 수행

rmse\_scores = []

for train\_idx, val\_idx in kfold.split(X\_train):

X\_tr, X\_val = X\_train.iloc[train\_idx], X\_train.iloc[val\_idx]

y\_tr, y\_val = y\_train.iloc[train\_idx], y\_train.iloc[val\_idx]

model.fit(X\_tr, y\_tr.values.ravel()) # 모델 학습

y\_val\_pred = model.predict(X\_val) # 검증 데이터 예측

# RMSE 계산 (MSE의 제곱근)

rmse = np.sqrt(mean\_squared\_error(y\_val, y\_val\_pred))

rmse\_scores.append(rmse)

return np.mean(rmse\_scores) # 평균 RMSE 반환 (최소화 목표)

# Optuna 실행 (RMSE 최소화)

study\_svr = optuna.create\_study(direction="minimize") # ⬅️ 방향 변경

try:

study\_svr.optimize(objective\_svr, n\_trials=350, show\_progress\_bar=True, gc\_after\_trial=True, catch=(Exception,))

except Exception as e:

print("⚠ Optuna 실행 중 오류 발생:", e)

# 최적의 하이퍼파라미터 확인

if study\_svr.best\_trial:

best\_svr\_params = study\_svr.best\_params

print("Best SVR Parameters (최소 RMSE 기준):", best\_svr\_params)

# 최적의 SVR 모델 생성

svr\_best = SVR(\*\*best\_svr\_params)

# 모델 학습

svr\_best.fit(X\_train, y\_train.values.ravel())

# 학습 데이터에 대한 예측

y\_train\_pred = svr\_best.predict(X\_train)

# 학습 데이터 RMSE 계산

rmse\_train = np.sqrt(mean\_squared\_error(y\_train, y\_train\_pred))

# 학습 및 테스트 R² 점수

train\_score = svr\_best.score(X\_train, y\_train)

test\_score = svr\_best.score(X\_test, y\_test)

print("\n🔥 SVR 최종 성능 (최소 RMSE 기준):")

print(f" - 학습 데이터 R²: {train\_score:.4f}")

print(f" - 테스트 데이터 R²: {test\_score:.4f}")

print(f" - 학습 데이터 RMSE: {rmse\_train:.4f}") # 학습 데이터 RMSE 출력

# 예측 및 평가 지표 계산

y\_test\_pred = svr\_best.predict(X\_test)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)

rmse\_test = np.sqrt(mse\_test) # RMSE 추가

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

print("\n📊 평가 지표:")

print(f" - 테스트 MAE: {mae\_test:.4f}")

print(f" - 테스트 MSE: {mse\_test:.4f}")

print(f" - 테스트 RMSE: {rmse\_test:.4f}") # 테스트 RMSE 출력

print(f" - 테스트 R²: {r2\_test:.4f}")

else:

print("⚠ 최적의 하이퍼파라미터를 찾지 못했습니다.")

import optuna

from optuna.visualization import plot\_optimization\_history

# 기존 코드 실행

fig = plot\_optimization\_history(study\_svr)

fig.update\_xaxes(range=[0, 360])

fig.update\_yaxes(range=[1.4495, 1.4497])

fig.add\_vline(

x=study\_svr.best\_trial.number,

line\_dash="dash",

line\_color="red",

annotation\_text=f"Trial {study\_svr.best\_trial.number}",

annotation\_position="top"

)

fig.show()

optuna.visualization.plot\_parallel\_coordinate(study\_svr)

optuna.visualization.plot\_param\_importances(study\_svr)