```python
import os, shutil
import glob
import pandas as pd
import time
import multiprocessing as mp
import gc


#tbl_temp = pd.DataFrame() #정제된 csv 컨테이너 iteration 발생 때 마다 저장 하기 위함.
tbl_temp_final = pd.DataFrame() # 마지막 일별 데이터를 concatenate함 (일별 컨테이너)
tbl_final = pd.DataFrame() # 마지막 일별 데이터를 concatenate함 (일별 컨테이너)
path = os.getcwd()
print("폴더 경로 입력(예: C:\자료): ")
folderloc = input()

path_Done= folderloc +"\\" + "Done"
#path_temp= folderloc + "\\" + "temp"
isExist = os.path.exists(path_Done)
#printing if the path exists or not
if isExist==True:
    print("이미 Done 폴더가 있습니다.")
isExist = os.path.exists(path_Done)
if not isExist:
   os.makedirs(path_Done)
   print("Done폴더 생성")

#path_final_temp= "\\\\nt1.koti.re.kr\\13 모빌리티빅데이터분석팀\\02. 개인별 폴더\\12. 이준성\\분석과정및결과" + "\\" + "final_temp"
path_final_temp= folderloc + "\\" + "final_temp"
path_final_temp_sido_time_dow_age_tf_base= path_final_temp  + "\\" + "tbl_sido_time_dow_age_tf_base"
path_final_temp_tbl_ch_sido_tf_result= path_final_temp + "\\" + "tbl_ch_sido_tf_result"

isExistfin = os.path.exists(path_final_temp)
if isExistfin==True:
    print("이미 final_temp 폴더가 있습니다.")
isExistfin = os.path.exists(path_final_temp)
if not isExistfin:
   os.makedirs(path_final_temp)
   os.makedirs(path_final_temp_sido_time_dow_age_tf_base)
   os.makedirs(path_final_temp_tbl_ch_sido_tf_result)

   print("final_temp폴더(하위포함) 생성")

#path_final= "\\\\nt1.koti.re.kr\\13 모빌리티빅데이터분석팀\\02. 개인별 폴더\\12. 이준성\\분석과정및결과" + "\\" + "final"
path_final= folderloc + "\\" + "final"

#path_final= "\\\\nt1.koti.re.kr\\13 모빌리티빅데이터분석팀\\02. 개인별 폴더\\12. 이준성\\분석과정및결과" + "\\" + "final"
isExistfin = os.path.exists(path_final)
if isExistfin==True:
    print("이미 final 폴더가 있습니다.")
isExistfin = os.path.exists(path_final)
if not isExistfin:
   os.makedirs(path_final)
```

This program is a script translated from R to Python.
[R script has only Preprocessing]
It's important to consider that there are significant differences between the workstation previously used with R and the PC (assigned to me) now using Python.

[Workstation -> PC]

CPU (i9-10900 -> i7-6700)
RAM (128GB -> 32GB)
GPU (RTX 3090 -> 960GTX)
Considerations made during the translation from R to Python include:

Analysis should not halt if Python crashes.
If a crash occurs, resume from the save point.
Use minimal RAM, preferably through CLI-based operations (avoid using libraries like PyQt5).
Graphs are outputted in Excel following the existing reporting format (Matplotlib is not used).
Continuous analysis from day 1 to day 365 without interruption.
The data spans three years, with daily data ranging from 6-8GB. For a year's worth, it's 2.5TB-3TB (many files exceed 7GB).

```python
gc.enable()

#loop_chk = 0
for f in csv_files:
    tbl_group_sido_tf_all_base = pd.DataFrame() #정제된 CSV 컨테이너 iteration 발생 때 마다 저장 하기 위함.
    tbl_group_ch_sido_tf_result = pd.DataFrame()
    tbl_group_sido_time_dow_age_tf_base = pd.DataFrame()
    tbl_tf_all_base = pd.DataFrame()

    #loop_chk = loop_chk + 1
    # print the location and filename
    print('파일 경로:', f)
    print('파일 이름:', f.split("\\")[-1])
    tbl_chunk = pd.read_csv(f, chunksize=600000)
    start_time = time.time() # 알고리즘 시작 시간
    for chunk in tbl_chunk:
        chunk['o_sido'] = chunk['o_sido'].astype("Int64") # o_sido float 문제 해결
        chunk.loc[~(chunk['traffic_time_all_mean']>=5),'traffic_all']=None
        chunk.loc[~(chunk['traffic_time_all_mean']<180), 'traffic_all']=None
        chunk.loc[~(chunk['traffic_time_30_mean']>=5),'traffic_30']=None
        chunk.loc[~(chunk['traffic_time_30_mean']<180),'traffic_30']=None
        chunk.loc[~(chunk['traffic_time_60_mean']>=5),'traffic_60']=None
        chunk.loc[~(chunk['traffic_time_60_mean']<180),'traffic_60']=None

        chunk.drop(chunk[(chunk['stay_traffic_volume'] == None) & (chunk['traffic_all'] == None) & (chunk['traffic_30'] == None) & (chunk['traffic_60'] == None)].index, inplace = True) # 같은 행에 여러 변수가 NULL일 경우 날림 -> 최종 결과값
        chunk.drop(chunk[(chunk['stay_traffic_volume'] == 0) & (chunk['traffic_all'] == 0) & (chunk['traffic_30'] == 0) & (chunk['traffic_60'] == 0)].index, inplace = True) # 같은 행에 여러 변수가 NULL일 경우 날림 -> 최종 결과값: 같음

        #tbl.loc[tbl['stay_traffic_volume'] == "*", 'stay_traffic_volume'] = 3
        chunk['stay_traffic_volume'] = chunk['stay_traffic_volume'].replace('*', 3)
        chunk['stay_traffic_volume'] = chunk.stay_traffic_volume.astype(float)
        chunk['stay_traffic_volume'] = (chunk['stay_traffic_volume'] * chunk['stay_traffic_volume_cor_index'])
        #tbl.loc[tbl['stay_traffic_volume'], 'stay_traffic_volume'] = (tbl['stay_traffic_volume'] * tbl['stay_traffic_volume_cor_index'])

        chunk['TT_V_all'] = (chunk['stay_traffic_volume'] * chunk['traffic_all'])
        chunk['TT_T_all'] = (chunk['traffic_time_all_mean'] * chunk['TT_V_all'])
        chunk['TT_V_30'] = (chunk['stay_traffic_volume'] * chunk['traffic_30'])
        chunk['TT_T_30'] = (chunk['traffic_time_30_mean'] * chunk['TT_V_30'])
        chunk['TT_V_60'] = (chunk['stay_traffic_volume'] * chunk['traffic_60'])
        chunk['TT_T_60'] = (chunk['traffic_time_60_mean'] * chunk['TT_V_60'])
        tbl_tf_all_base = chunk[['o_sido','d_sido', 'o_type', 'd_type','o_time','d_time','o_date','o_dow', 'age','stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60']]#지역별 통행량 비교 표3-9를 제외한
        tbl_group_sido_tf_all_base = pd.concat([tbl_group_sido_tf_all_base,tbl_tf_all_base])
    del tbl_chunk
    del chunk
```

When using Pandas on Windows to load a file of around 7GB into 32GB of RAM, the system, including background processes, ends up utilizing more than 32GB of RAM. Consequently, tools like Pyarrow were deemed unnecessary luxuries. Since there was no urgent need for speed (I could just set it running and leave it to finish), I opted for a method to address the issue of exceeding the 32GB RAM limit by using chunksize. I designed the logic to read and analyze the CSV file in chunks. Then, I analyzed each chunk, dropped the leftover scraps of tables to free up memory, and repeated this process until all analyzed files were concatenated together. As an extra measure, I considered the possibility of Python crashing due to this approach, so I attempted to save the analysis results in chunks to files. However, this method proved too time-consuming and was therefore excluded.

```
    tbl_group_sido_time_dow_age_tf_base = tbl_group_sido_tf_all_base.groupby(['o_sido','d_sido','o_type','d_type','o_time','d_time','o_dow','age'])[['stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60
    tbl_group_sido_time_dow_age_tf_base.to_csv(path_final_temp + "\\" + "tbl_sido_time_dow_age_tf_base\\tbl_sido_time_dow_age_tf_base_" + (f.split("\\")[-1]).replace("zip", "csv"))
    del tbl_group_sido_time_dow_age_tf_base
    del tbl_group_sido_tf_all_base
    del tbl_tf_all_base

    print("파일 로딩 및 정제에 걸린 시간: " + str(round((time.time() - start_time),2)) + "초")
    shutil.move(folderloc+"\\"+(f.split("\\")[-1]), folderloc + "\\" + "Done" + "\\" + (f.split("\\")[-1]))
    print("작업한 파일[" + f.split("\\")[-1] + "]을 작업완료 폴더(Done) 안으로 옮김")

#######

print("<마지막 연산 시작>")


tbl_final = pd.DataFrame() # (일별 컨테이너 포맷) 메모리 관리
csv_temp_final_files = glob.glob(os.path.join(path_final_temp_sido_time_dow_age_tf_base, "*.csv"))
for fin in csv_temp_final_files:
    tbl_temp_final = pd.DataFrame()
    tbl_temp_final = pd.read_csv(fin, engine="pyarrow")
    tbl_temp_final = tbl_temp_final.groupby(['o_sido','d_sido','o_type', 'd_type', 'o_time', 'd_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60']].apply(sum)
    tbl_final = pd.concat([tbl_final, tbl_temp_final], ignore_index=True)
    del tbl_temp_final
    print("시도별 OD " + fin + " 로딩 완료")
##### Epoch
```

The analyzed chunk data is systematically saved into tables and then stored back into files. By repeating this process 365 to 366 times (for leap years), an entire year's worth of data undergoes preprocessing! As a result, a daily 7GB file is reduced to 300MB to 500MB after preprocessing. Therefore, preprocessed data for 365 days amounts to approximately 15-20GB, significantly reducing the original size from 3TB to 15-20GB.

Once preprocessing is completed, the script moves on to the post-processing stage, referencing the saved file locations.

It's noteworthy that although Pyarrow didn't support chunking at the time of script creation, rendering it unsuitable for 7GB files, it could still be used for files sized between 300-500MB. Therefore, for the sake of time efficiency, Pyarrow was utilized for preprocessing the files of this size.

```
# 시도별 OD
final= tbl_final.groupby(['o_sido','d_sido','o_type', 'd_type', 'o_time', 'd_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60']].apply(sum)
tbl_final['TT_T_all'] = (tbl_final['TT_T_all'] / tbl_final['TT_V_all'])
tbl_final['TT_T_30'] = (tbl_final['TT_T_30'] / tbl_final['TT_V_30'])
tbl_final['TT_T_60'] = (tbl_final['TT_T_60'] / tbl_final['TT_V_60'])
tbl_final_Seoul_o = tbl_final.loc[(tbl_final['o_sido'] == 11), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].apply(
tbl_final_Seoul_od = pd.concat([tbl_final_Seoul_o, tbl_final.loc[(tbl_final['d_sido'] == 11), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_6
del tbl_final_Seoul_o
tbl_final_Busan_o = tbl_final.loc[(tbl_final['o_sido'] == 21), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].apply(
tbl_final_Busan_od = pd.concat([tbl_final_Busan_o, tbl_final.loc[(tbl_final['d_sido'] == 21), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_6
del tbl_final_Busan_o
tbl_final_Daegu_o = tbl_final.loc[(tbl_final['o_sido'] == 22), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].apply(
tbl_final_Daegu_od = pd.concat([tbl_final_Daegu_o, tbl_final.loc[(tbl_final['d_sido'] == 22), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_6
del tbl_final_Daegu_o
tbl_final_Incheon_o = tbl_final.loc[(tbl_final['o_sido'] == 23), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].appl
tbl_final_Incheon_od = pd.concat([tbl_final_Incheon_o, tbl_final.loc[(tbl_final['d_sido'] == 23), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT
del tbl_final_Incheon_o
tbl_final_Gwangju_o = tbl_final.loc[(tbl_final['o_sido'] == 24), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].appl
tbl_final_Gwangju_od = pd.concat([tbl_final_Gwangju_o, tbl_final.loc[(tbl_final['d_sido'] == 24), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT
del tbl_final_Gwangju_o
tbl_final_Daejeon_o = tbl_final.loc[(tbl_final['o_sido'] == 25), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].appl
tbl_final_Daejeon_od = pd.concat([tbl_final_Daejeon_o, tbl_final.loc[(tbl_final['d_sido'] == 25), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT
del tbl_final_Daejeon_o
tbl_final_Ulsan_o = tbl_final.loc[(tbl_final['o_sido'] == 26), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].apply(
tbl_final_Ulsan_od = pd.concat([tbl_final_Ulsan_o, tbl_final.loc[(tbl_final['d_sido'] == 26), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_6
del tbl_final_Ulsan_o
tbl_final_Sejong_o = tbl_final.loc[(tbl_final['o_sido'] == 29), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].apply
tbl_final_Sejong_od = pd.concat([tbl_final_Sejong_o, tbl_final.loc[(tbl_final['d_sido'] == 29), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V
del tbl_final_Sejong_o
tbl_final_Gyeongi_o = tbl_final.loc[(tbl_final['o_sido'] == 31), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].appl
tbl_final_Gyeongi_od = pd.concat([tbl_final_Gyeongi_o, tbl_final.loc[(tbl_final['d_sido'] == 31), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT
del tbl_final_Gyeongi_o
tbl_final_Gangwon_o = tbl_final.loc[(tbl_final['o_sido'] == 32), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].appl
tbl_final_Gangwon_od = pd.concat([tbl_final_Gangwon_o, tbl_final.loc[(tbl_final['d_sido'] == 32), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT
del tbl_final_Gangwon_o
tbl_final_Chungbuk_o = tbl_final.loc[(tbl_final['o_sido'] == 33), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].app
tbl_final_Chungbuk_od = pd.concat([tbl_final_Chungbuk_o, tbl_final.loc[(tbl_final['d_sido'] == 33), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30',
del tbl_final_Chungbuk_o
tbl_final_Chungnam_o = tbl_final.loc[(tbl_final['o_sido'] == 34), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].app
tbl_final_Chungnam_od = pd.concat([tbl_final_Chungnam_o, tbl_final.loc[(tbl_final['d_sido'] == 34), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30',
del tbl_final_Chungnam_o
tbl_final_Jeonbuk_o = tbl_final.loc[(tbl_final['o_sido'] == 35), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].appl
tbl_final_Jeonbuk_od = pd.concat([tbl_final_Jeonbuk_o, tbl_final.loc[(tbl_final['d_sido'] == 35), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT
del tbl_final_Jeonbuk_o
tbl_final_Jeonnam_o = tbl_final.loc[(tbl_final['o_sido'] == 36), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].appl
tbl_final_Jeonnam_od = pd.concat([tbl_final_Jeonnam_o, tbl_final.loc[(tbl_final['d_sido'] == 36), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT
del tbl_final_Jeonnam_o
tbl_final_Gyeongbuk_o = tbl_final.loc[(tbl_final['o_sido'] == 37), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].ap
tbl_final_Gyeongbuk_od = pd.concat([tbl_final_Gyeongbuk_o, tbl_final.loc[(tbl_final['d_sido'] == 37), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30',
del tbl_final_Gyeongbuk_o
tbl_final_Gyeongnam_o = tbl_final.loc[(tbl_final['o_sido'] == 38), ['o_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].groupby(['o_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60','TT_T_60']].ap
tbl_final_Gyeongnam_od = pd.concat([tbl_final_Gyeongnam_o, tbl_final.loc[(tbl_final['d_sido'] == 38), ['d_sido', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30',
```

However, there was a lot of computation to be done during the post-processing stage. Unfortunately, the elements to be analyzed were scattered all over the place, making it difficult to use for loops or regular expressions. So, I just resorted to using Ctrl + C and Ctrl + V.

```
del tbl_final_Jeju_o
tbl_final_sido_concat_od = pd.concat([tbl_final_Seoul_od,tbl_final_Busan_od,tbl_final_Daegu_od,tbl_final_Incheon_od,tbl_final_Gwangju_od,tbl_final_Daejeon_od,tbl_final_Ulsan_od,tbl_final_Sejong_od,tbl_final_Gyeongi_od,tbl_final_Gangwon_o
with pd.ExcelWriter(path_final + "\\"+ 'final_tbl_sido_tf_base.xlsx') as excel: # 시도별 OD
    final.to_excel(excel, sheet_name="All_sido")
    tbl_final_sido_concat_od.to_excel(excel, sheet_name="All_sido_concat")
    tbl_final_Seoul_od.to_excel(excel, sheet_name="서울")
    tbl_final_Busan_od.to_excel(excel, sheet_name="부산")
    tbl_final_Daegu_od.to_excel(excel, sheet_name="대구")
    tbl_final_Incheon_od.to_excel(excel, sheet_name="인천")
    tbl_final_Gwangju_od.to_excel(excel, sheet_name="광주")
    tbl_final_Daejeon_od.to_excel(excel, sheet_name="대전")
    tbl_final_Ulsan_od.to_excel(excel, sheet_name="울산")
    tbl_final_Sejong_od.to_excel(excel, sheet_name="세종")
    tbl_final_Gyeongi_od.to_excel(excel, sheet_name="경기")
    tbl_final_Gangwon_od.to_excel(excel, sheet_name="강원")
    tbl_final_Chungbuk_od.to_excel(excel, sheet_name="충북")
    tbl_final_Chungnam_od.to_excel(excel, sheet_name="충남")
    tbl_final_Jeonbuk_od.to_excel(excel, sheet_name="전북")
    tbl_final_Jeonnam_od.to_excel(excel, sheet_name="전남")
    tbl_final_Gyeongbuk_od.to_excel(excel, sheet_name="경북")
    tbl_final_Gyeongnam_od.to_excel(excel, sheet_name="경남")
    tbl_final_Jeju_od.to_excel(excel, sheet_name="제주")

print("final_tbl_sido_tf_base 연산 완료")
del tbl_temp_final
del tbl_final
del csv_temp_final_files


tbl_final = pd.DataFrame() # (일별 컨테이너 포맷)
csv_temp_final_files = glob.glob(os.path.join(path_final_temp_sido_time_dow_age_tf_base, "*.csv"))
for fin in csv_temp_final_files:
    tbl_temp_final = pd.read_csv(fin, engine="pyarrow")
    tbl_final = pd.concat([tbl_final, tbl_temp_final], ignore_index=True)   #요일별 출퇴근
```

The first-stage post-processing resulted in a file named 'Commute by City' that needed to be split and saved. I divided it into sheets within one Excel file.
However, after the first-stage post-processing was completed... there was more computation to be done. There are various other elements that need to be extracted from the preprocessed file, such as commuting by day of the week, commuting by hour, commuting time by age group, and others. Different data computation formulas are involved in various scenarios, so I began the computation using the preprocessed file.

processing the preprocessed files for each name as many times as there are names in the 'final_tbl_' name base.

```python
tbl_final = pd.DataFrame() # 일별 컨테이너 프레임
csv_temp_final_files = glob.glob(os.path.join(path_final_temp_sido_time_dow_age_tf_base, "*.csv"))
for fin in csv_temp_final_files:
    tbl_temp_final = pd.read_csv(fin, engine="pyarrow")
    tbl_final = pd.concat([tbl_final, tbl_temp_final], ignore_index=True)   #요일별 출퇴근
##### Epoch
#tbl_final = tbl_final.loc[(tbl_final['o_type'] == "H") & (tbl_final['d_type'] == "C") | (tbl_final['o_type'] == "C") & (tbl_final['d_type'] == "H"), ['o_dow', 'o_time', 'o_type', 'd_type', 'stay_traffic_volume', 'TT_V_all', 'TT_T_all',
tbl_final= tbl_final.groupby(['o_sido', 'd_sido','o_dow', 'o_time', 'd_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60']].apply(sum)
tbl_final['TT_T_all'] = (tbl_final['TT_T_all'] / tbl_final['TT_V_all'])
tbl_final['TT_T_30'] = (tbl_final['TT_T_30'] / tbl_final['TT_V_30'])
tbl_final['TT_T_60'] = (tbl_final['TT_T_60'] / tbl_final['TT_V_60'])
tbl_final = tbl_final.reset_index()
tbl_final_sun_o = tbl_final.loc[(tbl_final['o_dow'] == 1), ['o_dow','o_sido','o_time','stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['o_sido','o_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].apply
tbl_final_sun_od = pd.concat([tbl_final_sun_o, tbl_final.loc[(tbl_final['o_dow'] == 1), ['o_dow','d_sido','d_time', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido','d_time'])[['stay_traffic_volume', 'TT_V_all'
tbl_final_mon_o = tbl_final.loc[(tbl_final['o_dow'] == 2), ['o_dow','o_sido','o_time','stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['o_sido','o_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].apply
tbl_final_mon_od = pd.concat([tbl_final_mon_o, tbl_final.loc[(tbl_final['o_dow'] == 2), ['o_dow','d_sido','d_time', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido','d_time'])[['stay_traffic_volume', 'TT_V_all'
tbl_final_tue_o = tbl_final.loc[(tbl_final['o_dow'] == 3), ['o_dow','o_sido','o_time','stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['o_sido','o_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].apply
tbl_final_tue_od = pd.concat([tbl_final_tue_o, tbl_final.loc[(tbl_final['o_dow'] == 3), ['o_dow','d_sido','d_time', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido','d_time'])[['stay_traffic_volume', 'TT_V_all'
tbl_final_wed_o = tbl_final.loc[(tbl_final['o_dow'] == 4), ['o_dow','o_sido','o_time','stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['o_sido','o_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].apply
tbl_final_wed_od = pd.concat([tbl_final_wed_o, tbl_final.loc[(tbl_final['o_dow'] == 4), ['o_dow','d_sido','d_time', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido','d_time'])[['stay_traffic_volume', 'TT_V_all'
tbl_final_thu_o = tbl_final.loc[(tbl_final['o_dow'] == 5), ['o_dow','o_sido','o_time','stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['o_sido','o_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].apply
tbl_final_thu_od = pd.concat([tbl_final_thu_o, tbl_final.loc[(tbl_final['o_dow'] == 5), ['o_dow','d_sido','d_time', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido','d_time'])[['stay_traffic_volume', 'TT_V_all'
tbl_final_fri_o = tbl_final.loc[(tbl_final['o_dow'] == 6), ['o_dow','o_sido','o_time','stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['o_sido','o_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].apply
tbl_final_fri_od = pd.concat([tbl_final_fri_o, tbl_final.loc[(tbl_final['o_dow'] == 6), ['o_dow','d_sido','d_time', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido','d_time'])[['stay_traffic_volume', 'TT_V_all'
tbl_final_sat_o = tbl_final.loc[(tbl_final['o_dow'] == 7), ['o_dow','o_sido','o_time','stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['o_sido','o_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].apply
tbl_final_sat_od = pd.concat([tbl_final_sat_o, tbl_final.loc[(tbl_final['o_dow'] == 7), ['o_dow','d_sido','d_time', 'stay_traffic_volume', 'TT_V_all', 'TT_V_30', 'TT_V_60']].groupby(['d_sido','d_time'])[['stay_traffic_volume', 'TT_V_all'

tbl_final_dow_concat = pd.concat([tbl_final_sun_od,tbl_final_mon_od,tbl_final_tue_od,tbl_final_wed_od,tbl_final_thu_od,tbl_final_fri_od,tbl_final_sat_od])

with pd.ExcelWriter(path_final + "\\"+ 'final_tbl_dow_base.xlsx') as excel_dow: # 시도별 OD
    final.to_excel(excel_dow, sheet_name="All_sido")
    tbl_final_dow_concat.to_excel(excel_dow, sheet_name="요일별_통합")
    tbl_final_sun_od.to_excel(excel_dow, sheet_name="일")
    tbl_final_mon_od.to_excel(excel_dow, sheet_name="월")
    tbl_final_tue_od.to_excel(excel_dow, sheet_name="화")
    tbl_final_wed_od.to_excel(excel_dow, sheet_name="수")
    tbl_final_thu_od.to_excel(excel_dow, sheet_name="목")
    tbl_final_fri_od.to_excel(excel_dow, sheet_name="금")
    tbl_final_sat_od.to_excel(excel_dow, sheet_name="토")

print("final_tbl_dow_base 연산 완료")
del tbl_temp_final
del tbl_final
del csv_temp_final_files
```

processing the preprocessed files for each name as many times as there are names in the 'final_tbl_' name base.

```python
tbl_final = pd.DataFrame() # (일별 컨테이너 포맷)
csv_temp_final_files = glob.glob(os.path.join(path_final_temp_sido_time_dow_age_tf_base, "*.csv"))
for fin in csv_temp_final_files:
    tbl_temp_final = pd.read_csv(fin, engine="pyarrow")
    tbl_final = pd.concat([tbl_final, tbl_temp_final], ignore_index=True)
##### Epoch
#tbl_final = tbl_final.loc[(tbl_final['o_type'] == "H") & (tbl_final['d_type'] == "C") | (tbl_final['o_type'] == "C") & (tbl_final['d_type'] == "H"), ['o_sido','o_type', 'd_type', 'stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30',
tbl_final= tbl_final.groupby(['o_sido','d_sido','o_type', 'd_type'])[['stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60']].apply(sum)
tbl_final['TT_T_all'] = (tbl_final['TT_T_all'] / tbl_final['TT_V_all'])
#tbl_group.loc['TT_T_all'] = (tbl_df['TT_T_all'] / tbl_df['TT_V_all'])
tbl_final['TT_T_30'] = (tbl_final['TT_T_30'] / tbl_final['TT_V_30'])
#tbl_group.loc['TT_T_30'] = (tbl_df['TT_T_30'] / tbl_df['TT_V_30'])
tbl_final['TT_T_60'] = (tbl_final['TT_T_60'] / tbl_final['TT_V_60'])
#tbl_group.loc['TT_T_60'] = (tbl_df['TT_T_60'] / tbl_df['TT_V_60'])
tbl_final.to_excel(path_final + "\\"+ 'final_tbl_sido_time_tf_base.xlsx')

print("final_tbl_sido_time_tf_base 연산 완료")
del tbl_temp_final
del tbl_final
del csv_temp_final_files


tbl_final = pd.DataFrame() # (일별 컨테이너 포맷)
csv_temp_final_files = glob.glob(os.path.join(path_final_temp_sido_time_dow_age_tf_base, "*.csv"))
for fin in csv_temp_final_files:
    tbl_temp_final = pd.read_csv(fin, engine="pyarrow")
    tbl_final = pd.concat([tbl_final, tbl_temp_final], ignore_index=True)   #연령별 출퇴근 통행비율
##### Epoch
#tbl_final = tbl_final.loc[(tbl_final['o_type'] == "H") & (tbl_final['d_type'] == "C") | (tbl_final['o_type'] == "C") & (tbl_final['d_type'] == "H"), ['age', 'o_time', 'o_type', 'd_type', 'stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'T
tbl_final= tbl_final.groupby(['o_type','d_type', 'age', 'o_time'])[['stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60']].apply(sum)
tbl_final['TT_T_all'] = (tbl_final['TT_T_all'] / tbl_final['TT_V_all'])
#tbl_group.loc['TT_T_all'] = (tbl_df['TT_T_all'] / tbl_df['TT_V_all'])
tbl_final['TT_T_30'] = (tbl_final['TT_T_30'] / tbl_final['TT_V_30'])
#tbl_group.loc['TT_T_30'] = (tbl_df['TT_T_30'] / tbl_df['TT_V_30'])
tbl_final['TT_T_60'] = (tbl_final['TT_T_60'] / tbl_final['TT_V_60'])
#tbl_group.loc['TT_T_60'] = (tbl_df['TT_T_60'] / tbl_df['TT_V_60'])
tbl_final.to_excel(path_final + "\\"+ 'final_tbl_age_base.xlsx')
print("final_tbl_age_base 연산 완료")
del tbl_temp_final
del tbl_final
del csv_temp_final_files


    # tbl_group_ch_sido_tf_result = pd.concat([tbl_group_ch_sido_tf_result,tbl_ch_sido_tf_result])
    # tbl_group_ch_sido_tf_result = tbl_group_ch_sido_tf_result.groupby(['o_sido','o_type', 'd_type'])[['stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60']].apply(sum)
    # tbl_group_ch_sido_tf_result.to_csv(path_final_temp + "\\" + "tbl_ch_sido_tf_result\\tbl_ch_sido_tf_result_" + str(chunkcount) + "_" + (f.split("\\")[-1]).replace("zip", "csv"))
    # del tbl_group_ch_sido_tf_result

```

```python
tbl_final = pd.DataFrame() # (일별 컨테이너 포맷)
csv_temp_final_files = glob.glob(os.path.join(path_final_temp_sido_time_dow_age_tf_base, "*.csv"))
for fin in csv_temp_final_files:
    tbl_temp_final = pd.read_csv(fin, engine="pyarrow")
    tbl_final = pd.concat([tbl_final, tbl_temp_final], ignore_index=True)
##### Epoch
#tbl_final = tbl_final.loc[(tbl_final['o_type'] == "H") & (tbl_final['d_type'] == "C") | (tbl_final['o_type'] == "C") & (tbl_final['d_type'] == "H"), ['o_sido','o_type', 'd_type', 'stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30',
tbl_final = tbl_final.loc[(tbl_final['o_type'] == "H") & (tbl_final['d_type'] == "C") | (tbl_final['o_type'] == "C") & (tbl_final['d_type'] == "H"), ['o_sido', 'o_type', 'd_type', 'o_date', 'o_time', 'stay_traffic_volume', 'TT_V_all', 'T
tbl_final= tbl_final.groupby(['o_sido','o_type', 'd_type'])[['stay_traffic_volume', 'TT_V_all', 'TT_T_all', 'TT_V_30', 'TT_T_30', 'TT_V_60', 'TT_T_60']].apply(sum)
tbl_final['TT_T_all'] = (tbl_final['TT_T_all'] / tbl_final['TT_V_all'])
#tbl_group.loc['TT_T_all'] = (tbl_df['TT_T_all'] / tbl_df['TT_V_all'])
tbl_final['TT_T_30'] = (tbl_final['TT_T_30'] / tbl_final['TT_V_30'])
#tbl_group.loc['TT_T_30'] = (tbl_df['TT_T_30'] / tbl_df['TT_V_30'])
tbl_final['TT_T_60'] = (tbl_final['TT_T_60'] / tbl_final['TT_V_60'])
#tbl_group.loc['TT_T_60'] = (tbl_df['TT_T_60'] / tbl_df['TT_V_60'])
tbl_final.to_excel(path_final + "\\"+ 'final_tbl_ch_sido_tf_result.xlsx')
print("final_tbl_ch_sido_tf_result 연산 완료")
del tbl_temp_final
del tbl_final
del csv_temp_final_files


#print(tbl_group) # 엑셀에서 열기 힘들경우
#tbl_final = tbl_final.reset_index(drop=False, inplace=True) #index 추가하고 싶을 시 사용
print("전체 작업에 걸린시간: " + str(round((time.time() - start_time),2)) + "초")
os.startfile(path_final + "\\") # 폴더 위치 열기
#os.startfile(path_final + "\\"+ 'final.csv') # 변환된 파일 연결프로그램으로 열기


#print(np.where((tbl_date_base['TT_T_30']) != (tbl_date_base['TT_T_60'])))


#print(tbl_df['traffic_time_all_mean'])

#tbl_grp = tbl.groupby(['o_sido', 'o_type', 'd_type', 'o_date'])
#print(tbl_grp)
```

I executed this program on the PC allocated by the company. Since there was only one PC assigned, I had to wait while performing this task without being able to do anything else. I couldn't even open Chrome; I just ran this program and if it crashed, I would troubleshoot, improve the code, and start again. Looking back at the code I wrote in November 2022, memories flood back even though it's been over a year and a half. Running computations on that computer took about three days. It had an i7-6700 processor, 32GB of RAM, and a 960GTX graphics card. I also wrote a CUDA version of the script, but it didn't work properly because the VRAM of the 960GTX was limited to 2GB. I couldn't even read the post-processed files properly. To optimize the computer, I completed the script using the concepts of Epoch and iteration that I learned during a year-long deep learning project in college. It took me 15 days to develop the script while experimenting with various CUDA implementations and other techniques. If ChatGPT had been available back then, I wonder if it would have taken only a day to accomplish the same tasks. ~~There's a lot to say about the original R code, but I'll omit that for now.~~

Execution

```
파일 정제에 걸린 시간: 6285.04초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221021_masking.csv
파일 이름: koti_final_all_20221021_masking.csv
파일 로딩에 걸린 시간: 6553.42초
koti_final_all_20221021_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 6658.57초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221022_masking.csv
파일 이름: koti_final_all_20221022_masking.csv
파일 로딩에 걸린 시간: 6909.51초
koti_final_all_20221022_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 6980.29초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221023_masking.csv
파일 이름: koti_final_all_20221023_masking.csv
파일 로딩에 걸린 시간: 7202.49초
koti_final_all_20221023_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 7252.06초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221024_masking.csv
파일 이름: koti_final_all_20221024_masking.csv
파일 로딩에 걸린 시간: 7518.24초
koti_final_all_20221024_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 7618.84초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221025_masking.csv
파일 이름: koti_final_all_20221025_masking.csv
파일 로딩에 걸린 시간: 7894.78초
koti_final_all_20221025_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 7997.4초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221026_masking.csv
파일 이름: koti_final_all_20221026_masking.csv
파일 로딩에 걸린 시간: 8282.7초
koti_final_all_20221026_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 8389.1초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221027_masking.csv
파일 이름: koti_final_all_20221027_masking.csv
파일 로딩에 걸린 시간: 8684.63초
koti_final_all_20221027_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 8786.88초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221028_masking.csv
파일 이름: koti_final_all_20221028_masking.csv
파일 로딩에 걸린 시간: 9098.19초
koti_final_all_20221028_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 9199.26초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221029_masking.csv
파일 이름: koti_final_all_20221029_masking.csv
파일 로딩에 걸린 시간: 9505.43초
koti_final_all_20221029_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 9568.56초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221030_masking.csv
파일 이름: koti_final_all_20221030_masking.csv
파일 로딩에 걸린 시간: 9820.06초
koti_final_all_20221030_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 9861.44초
파일 경로: C:\Users\Jun\Desktop\KOTI\자료분석\1달자료\koti_final_all_20221031_masking.csv
파일 이름: koti_final_all_20221031_masking.csv
파일 로딩에 걸린 시간: 10175.54초
koti_final_all_20221031_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 10249.16초
<마지막 연산 시작>
전체 작업에 걸린시간: 10249.45초
```

```
builder@ASUS-G14-2021: /mn
파일 이름: temp_koti_final_all_20221023_masking.csv
파일 로딩에 걸린 시간: 519.65초
temp_koti_final_all_20221023_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 519.75초
파일 경로: /mnt/c/Users/Jun/Desktop/KOTI/자료분석/1달자료/temp/temp_koti_final_all_20221024_masking.csv
파일 이름: temp_koti_final_all_20221024_masking.csv
파일 로딩에 걸린 시간: 549.42초
temp_koti_final_all_20221024_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 549.57초
파일 경로: /mnt/c/Users/Jun/Desktop/KOTI/자료분석/1달자료/temp/temp_koti_final_all_20221025_masking.csv
파일 이름: temp_koti_final_all_20221025_masking.csv
파일 로딩에 걸린 시간: 579.99초
temp_koti_final_all_20221025_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 580.15초
파일 경로: /mnt/c/Users/Jun/Desktop/KOTI/자료분석/1달자료/temp/temp_koti_final_all_20221026_masking.csv
파일 이름: temp_koti_final_all_20221026_masking.csv
파일 로딩에 걸린 시간: 610.75초
temp_koti_final_all_20221026_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 610.91초
파일 경로: /mnt/c/Users/Jun/Desktop/KOTI/자료분석/1달자료/temp/temp_koti_final_all_20221027_masking.csv
파일 이름: temp_koti_final_all_20221027_masking.csv
파일 로딩에 걸린 시간: 641.39초
temp_koti_final_all_20221027_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 641.56초
파일 경로: /mnt/c/Users/Jun/Desktop/KOTI/자료분석/1달자료/temp/temp_koti_final_all_20221028_masking.csv
파일 이름: temp_koti_final_all_20221028_masking.csv
파일 로딩에 걸린 시간: 669.91초
temp_koti_final_all_20221028_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 670.05초
파일 경로: /mnt/c/Users/Jun/Desktop/KOTI/자료분석/1달자료/temp/temp_koti_final_all_20221029_masking.csv
파일 이름: temp_koti_final_all_20221029_masking.csv
파일 로딩에 걸린 시간: 682.82초
temp_koti_final_all_20221029_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 682.92초
파일 경로: /mnt/c/Users/Jun/Desktop/KOTI/자료분석/1달자료/temp/temp_koti_final_all_20221030_masking.csv
파일 이름: temp_koti_final_all_20221030_masking.csv
파일 로딩에 걸린 시간: 692.17초
temp_koti_final_all_20221030_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 692.25초
파일 경로: /mnt/c/Users/Jun/Desktop/KOTI/자료분석/1달자료/temp/temp_koti_final_all_20221031_masking.csv
파일 이름: temp_koti_final_all_20221031_masking.csv
파일 로딩에 걸린 시간: 704.0초
temp_koti_final_all_20221031_masking.csv에 대한 데이터 정제 작업 완료
파일 정제에 걸린 시간: 704.08초
<마지막 연산 시작>
전체 작업에 걸린시간: 705.43초
```