```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)
```

해당 프로그램은 R로 작성 된 스크립트를 Python으로 번역 및 자동화한 스크립트이다.

[R은 전처리과정만 있음]

앞서, R을 사용 하던 워크스테이션과 Python을 사용하게 된 PC(나에게 할당 된 PC)는 차이가 많이 난다는 것을 감안해야 된다.

[워크스테이션 -> PC]

- CPU (i9- 10900 -> i7-6700)
- RAM (128GB -> 32GB)
- GPU (RTX 3090 -> 960GTX)

R -> Python 으로 번형 하면서 고려된 사항은 다음과 같다.

1. 분석 도중 Python이 중단 되면 안된다.
2. 만약, 중단 되면 세이브 포인트에서 시작한다.
3. CLI기반으로 최대한 RAM을 적게 사용한다.
   [pyqt5 등 사용 x]
4. 기존 보고서 작성방식에 따라 그래프를 엑셀에서 출력한다.(Matplotlib 사용x)
5. 무중단으로 1일 부터 365일까지 분석

데이터는 3년치로 1일치 기준 6-8GB로
1년치 기준 2.5TB-3TB (7GB이상 파일이 많음)

```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
```

윈도우 기준 Pandas를 사용하여 7GB 정도 되는 파일을 램 32GB에 올릴 경우 시스템 프로그램을 포함, 32GB 이상의 램을 사용한다. 그렇기에 Pyarrow 등과 같은 툴은 사치품..이었다. (어차피 돌려 놓고 집에 가니 딱히 빠를 필요는 없었다.)
일단, 32GB이상 사용하는 문제를 해결하기 위해서는 chunksize를 사용하여, CSV파일을 쪼개 불러 읽으면서 분석하게 로직을 구성하였다. 그리고 chunksize만큼 분석하고 분석하고 남은 찌꺼기 테이블은 drop하여 램 용량을 확보하는 방식으로 반복하여 분석된 파일들을 concat하는 방식으로 분석하였다. [번외로: 위 방식을 쓰게 될 경우 혹시 Python이 뻗을 수 있다고 판단, chunk 단위 분석된 것을 파일로 저장하는 것도 해보았지만 시간이 너무 걸려 제외 함]

```
    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
```

분석 된 chunk 데이터를 테이블에 차곡차곡 저장하고 해당 테이블을 파일 형태로 다시 저장한다.
이런 행위를 365 ~ 366(윤년)번 하면 1년치 데이터가 무려 전처리가 된 것이다!...
그리고 1일치 데이터 7GB파일이 300MB~500MB가 된다. -> 전처리 된 365일치 데이터가 15-20GB가량이 된다!
(3TB => 15~20GB)

전처리가 끝나면 저장된 파일 위치를 참조하여 후처리 과정으로 넘어가게 된다.

여기서 주목 할 것은 Pyarrow는 당시 스크립트 제작 당시 chunk를 지원하지 않아서 7GB파일에는 사용하지 못했지만,
300-500MB 크기에는 사용 할 수 있어 시간 효율성 측면에서 전처리 된 파일에는 Pyarrow를 사용 하였다.

```python
# 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',
```

근데 연산 할 것은 후처리 과정에 많았다. 일단, 아쉬운 것은 분석해야 할 요소들이 중구 난방으로 있어서 for 문을 사용하거나 regex등을 사용하기 어렵다는 것.. 그냥 ctrl + c & ctrl +v을 사용 하였다.

```
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_c
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
```

분석 된 1차 후처리 결과물인 '도시 별 출퇴근' 파일을 분할 저장해야 하는데 엑셀 파일 하나에 시트를 나눠 저장하였다.

근데 1차 후처리가 끝났는데.. 또 연산해야 될 게 있다. 전처리 된 파일에서 뽑아야 할 또 다른 여러가지 요소들이 있는데..

요일별 출퇴근, 시간별 출퇴근, 나이별 출퇴근 시간, 기타 등등 여러가지 상황에서의 데이터 연산식이 들어가게 되어
전처리 된 파일을 사용하여 연산을 시작한다.

```
tbl_final = pd.DataFrame() # (일별 컨테이너 포멧)
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
```

다시 또 연산을 시작한다... final_tbl_'이름'_base 이름 개수 만큼 전처리 된 파일을 후처리한다.

```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)
```

이 프로그램을 회사에서 할당한 PC에서 실행하였는데, PC 1대만 할당 받아서 이 작업을 수행하는 동안은 손가락 빨면서 기다려야 한다. 크롬을 켤 수도 없고 그냥 이 프로그램만 돌리고 뻗으면 문제점 파악하고 코드 개선하고 다시 시작했다. 2022년 11월에 작성한 코드로 1년 반이 넘은 지금 다시 코드를 보니 기억이 새록새록 난다. 그 컴퓨터로 연산 돌려 놓으면 3일 정도가 걸렸다. I7-6700.. 32GB 960GTX.. 그리고 번외로 CUDA버전의 스크립트도 작성 해 놓았는데 960GTX가 VRAM이 무려 2GB로 일단 안된다. 후처리 된 파일도 못 읽지 못했다. 💩컴에 최적화 하기 위해 학부에서 1년 동안 딥러닝 프로젝트를 하면서 배운 Epoch 및 iteration 개념을 사용하여 스크립트를 완성하였다. 위 로직 짜는데 이런 저런 CUDA를 써보고 등등 하면서 15일이 걸렸다. 만약, 그 때도 ChatGPT가 있었다면, 1일 정도 걸리지 않았을까 생각해본다.

~~기존 R 코드는.. 할 말 아 많지만 생략~~

실행

```
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
```