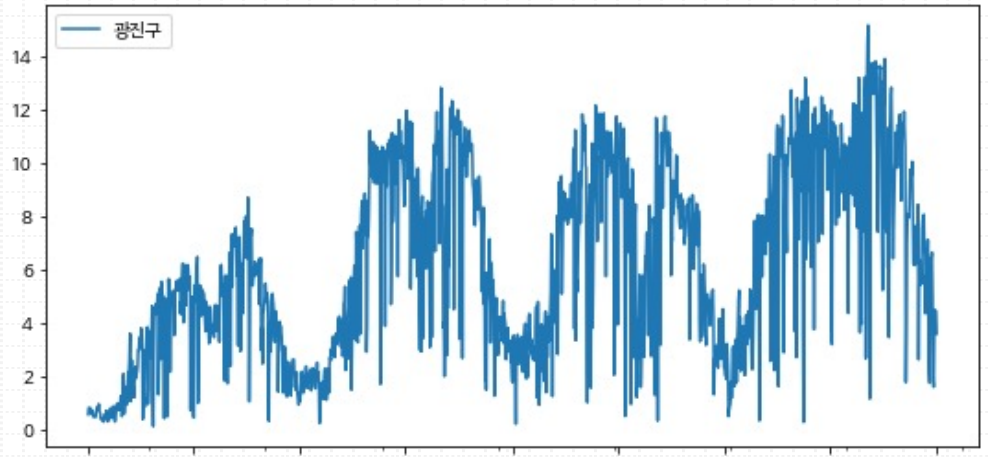
재벌집 막내조

1조

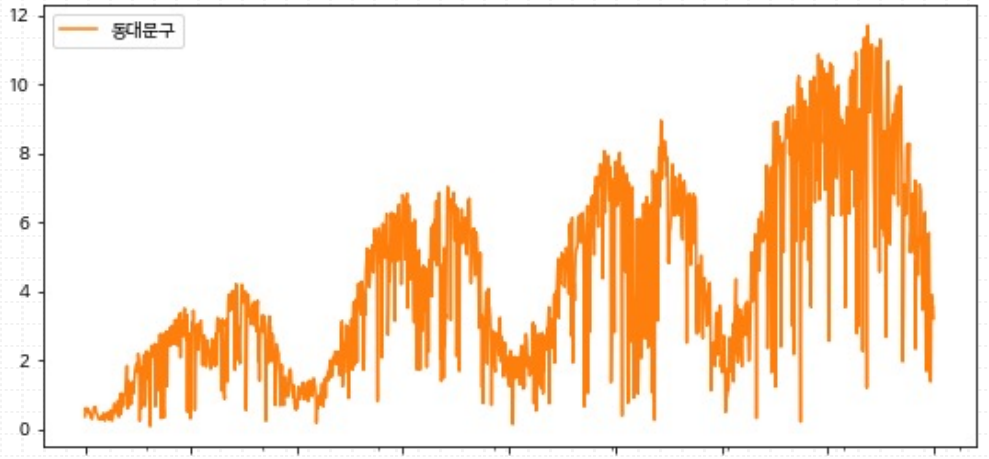
목차

시계열 성분 분석

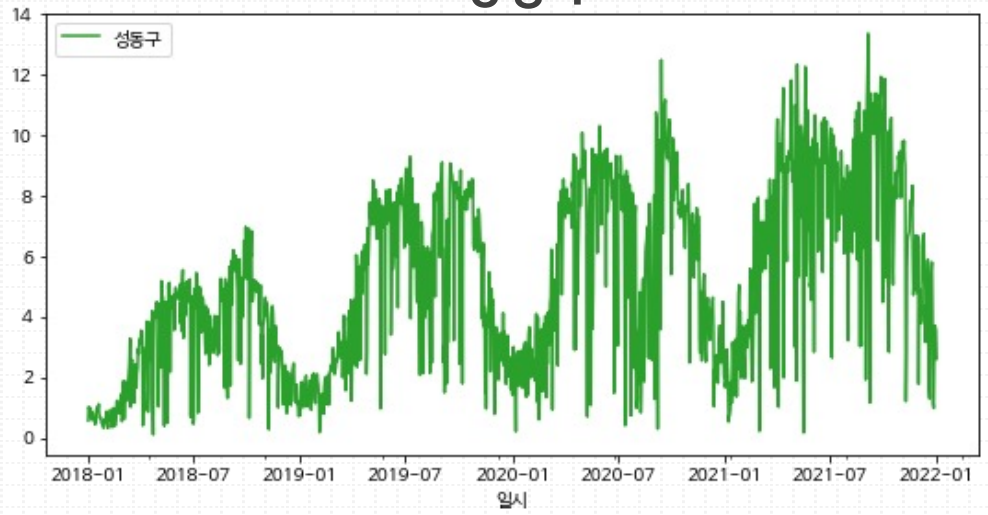
광진구



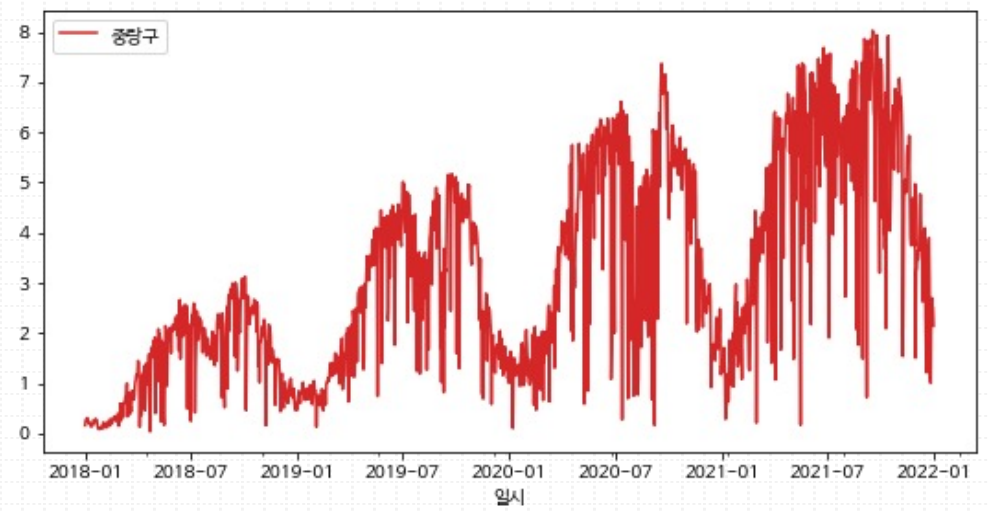
동대문구



성동구



중랑구



시계열 성분 분석

30일 기준 / 가법 분해

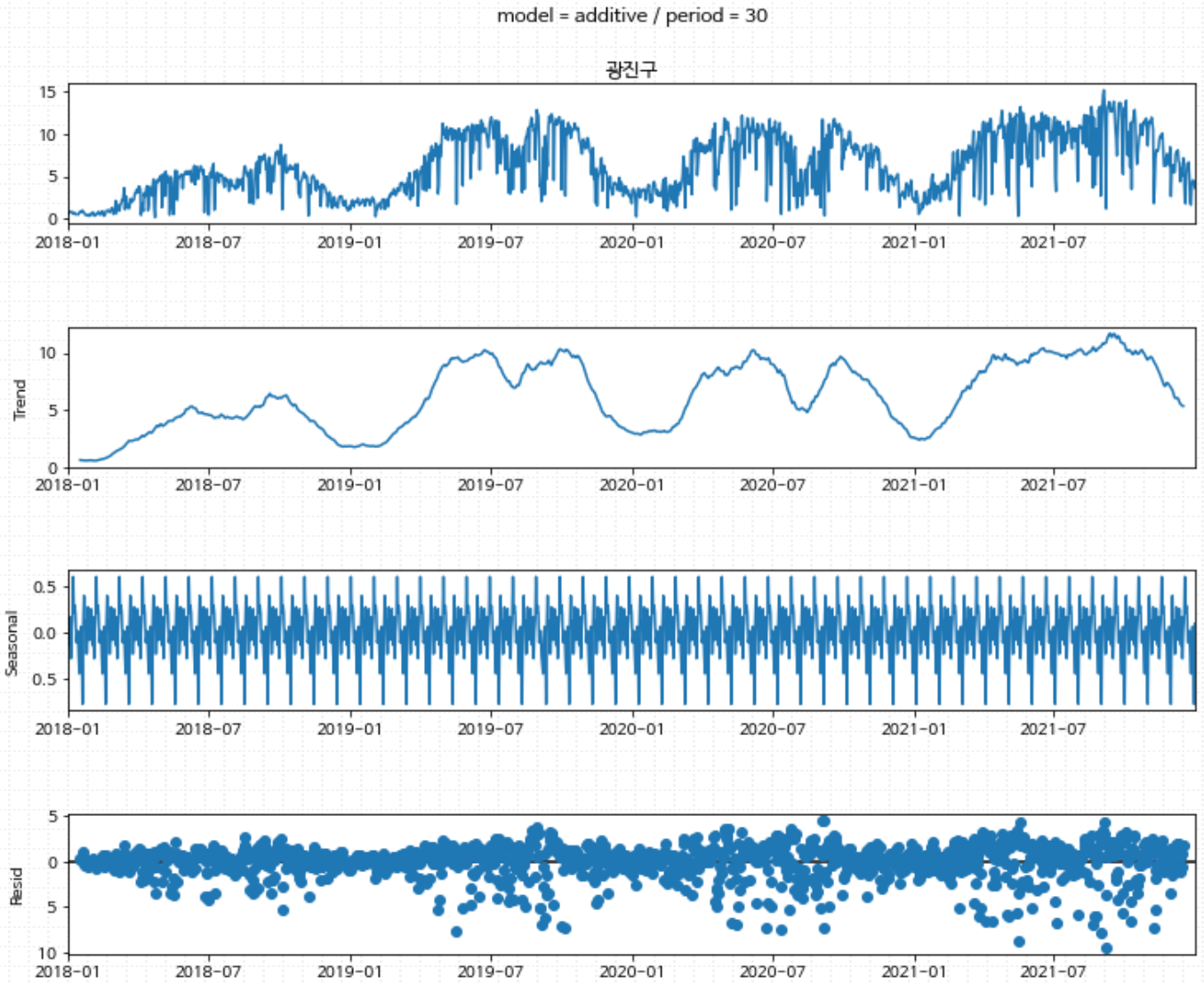
```
from statsmodels.tsa.seasonal import seasonal_decompose

gu = ['광진구', '동대문구', '성동구', '중랑구']
plt.rcParams["figure.figsize"] = [12, 6]
# 시계열 성분 분해
def time_series_decomposition(df, modeln, list, periodn):
  for i in list:
    seasonal_decompose(df[i], model= modeln, period=periodn).plot()
  return 0
```

```
time_series_decomposition(train, 'Additive', gu, 30)
#한달 기준
```

트렌드가 일정하지 않음

잔차가 큼

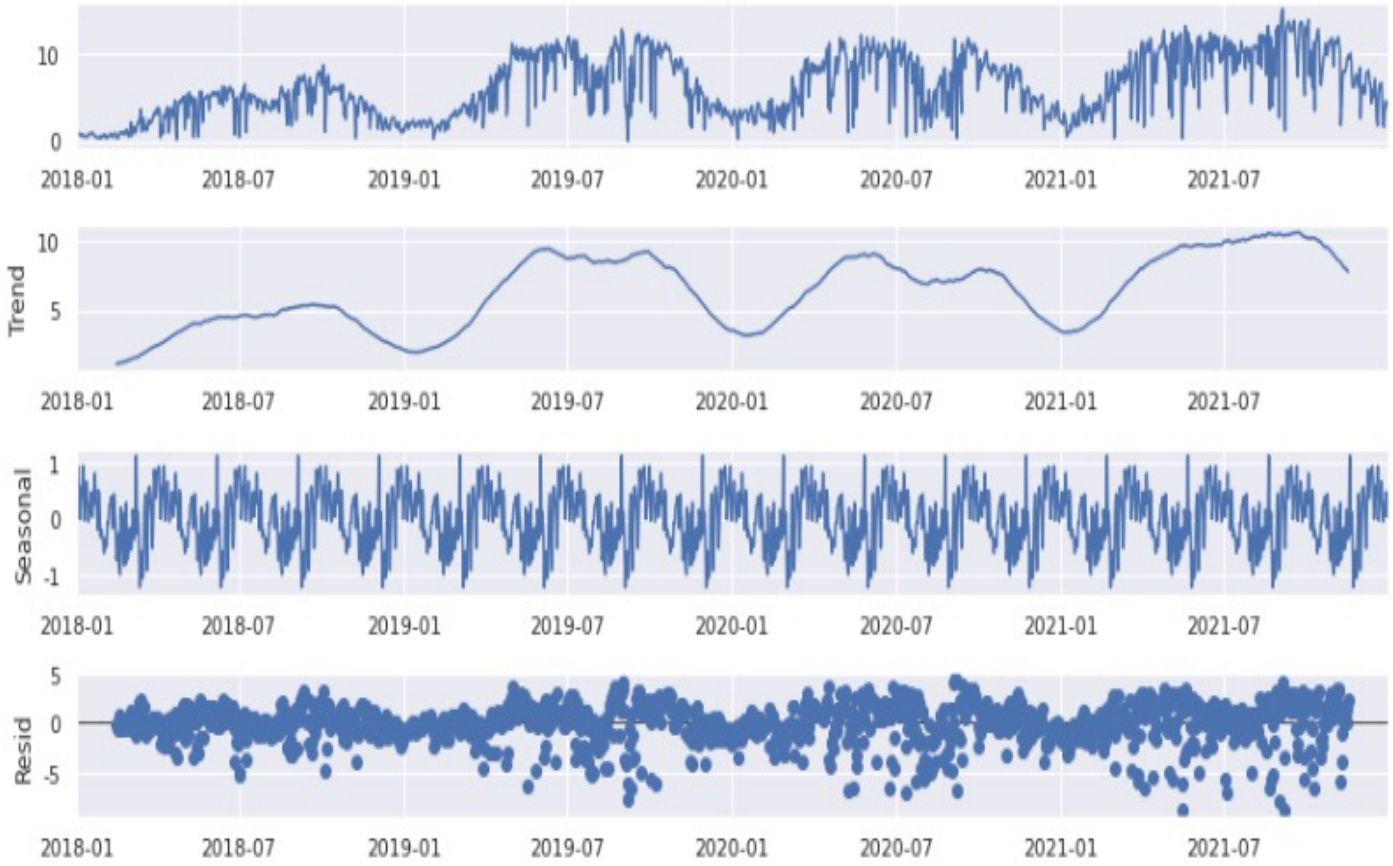


시계열 성분 분석

90일 기준 / 가법 분해

```
from statsmodels.tsa.seasonal import seasonal_decompose

gu = ['광진구', '동대문구', '성동구', '중랑구']
plt.rcParams["figure.figsize"] = [12, 6]
# 시계열 성분 분해
def time_series_decomposition(df, modeln, list, periodn):
  for i in list:
    seasonal_decompose(df[i], model= modeln, period=periodn).plot()
  return 0
time_series_decomposition(train, 'Additive', gu, 90)
#세달 기준
```



트렌드가 일정하지 않음

아직 잔차가 큼

시계열 성분 분석

360일 기준 / 가법 분해

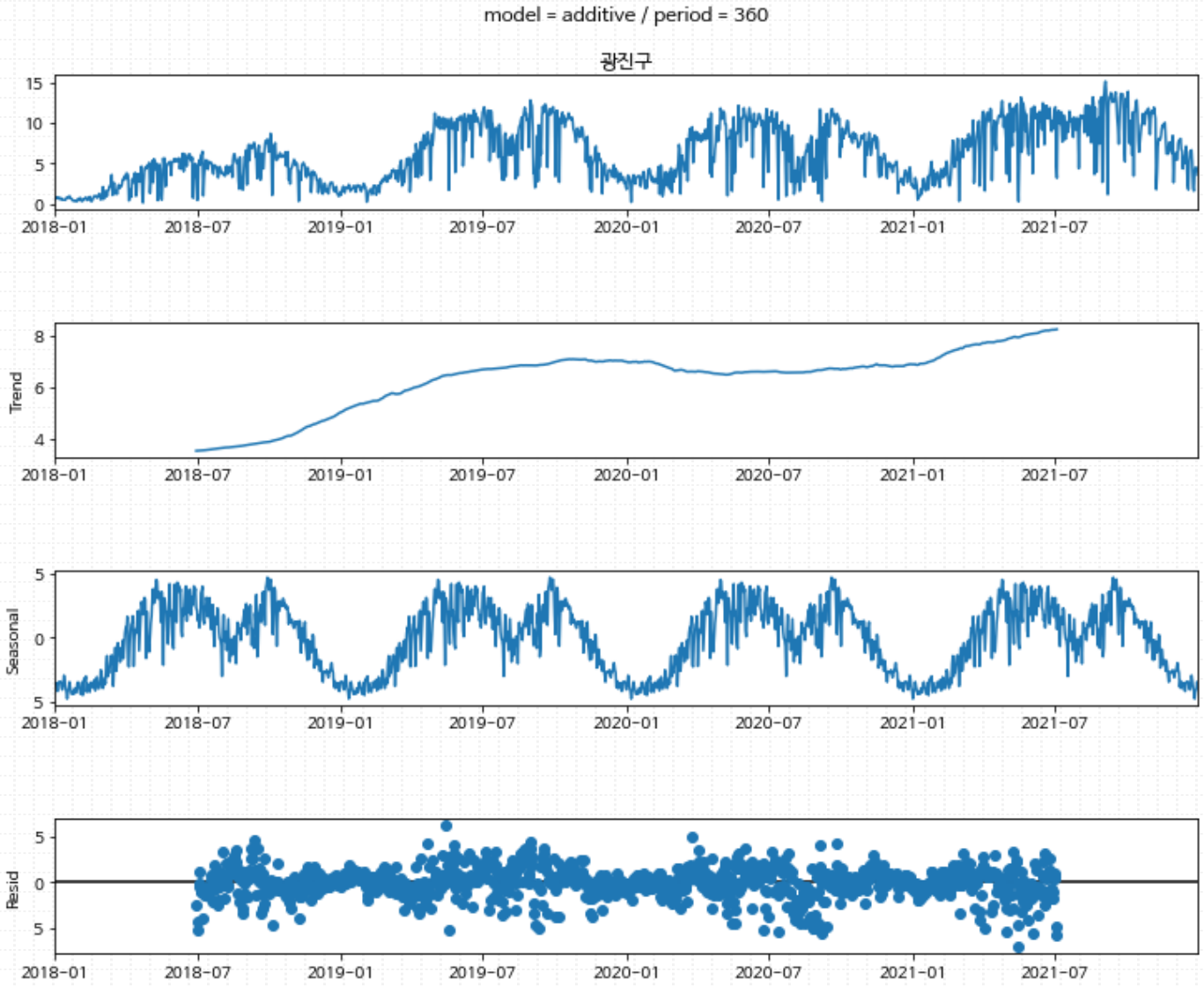
```
from statsmodels.tsa.seasonal import seasonal_decompose

gu = ['광진구', '동대문구', '성동구', '중랑구']
plt.rcParams["figure.figsize"] = [12, 6]
# 시계열 성분 분해
def time_series_decomposition(df, modeln, list, periodn):
  for i in list:
    seasonal_decompose(df[i], model= modeln, period=periodn).plot()
  return 0
time_series_decomposition(train, 'Additive', gu, 30)
#한달 기준
```

트렌드가 상승하는 개형

계절감이 원 그래프와 비슷

잔차가 밀집



시계열 성분 분석

광진구, 동대문구, 성동구, 중랑구 공통사항

가법 분해 기간에 따라 추세와 계절성 그래프간의 trade-off가 심함

현재 시계열은 비정상 시계열(정상성을 만족하지 않는 시계열)이기 때문에 정상 시계열로 변환이 필요

ACF, PACF를 확인해 어떤 부분에서 전처리가 필요한지 알아볼 필요가 있음.

시계열 성분 분석

시계열 데이터에서 정상성 (stationarity) 이란?

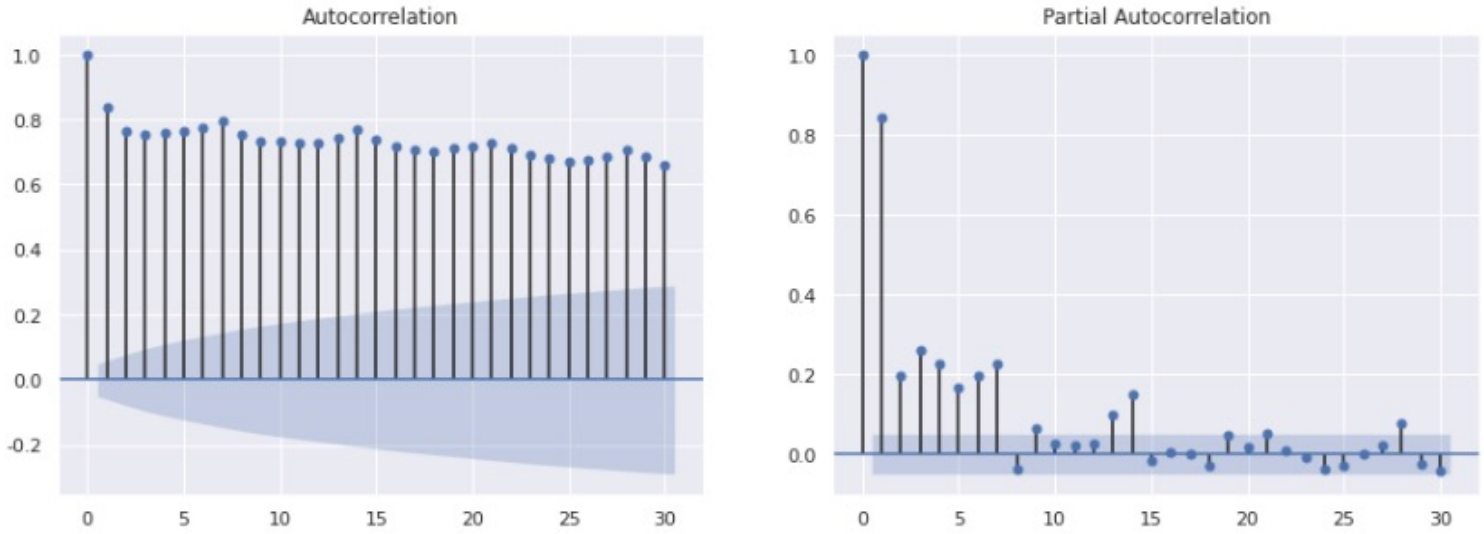
시계열 데이터가 시간에 따라 일정한 평균(mean)과 분산(variance)을 가지며, 시간의 추이에 따라 자기공분산(lag)이 변하지 않는 특성을 의미합니다.

즉, 정상성을 가진 시계열 데이터는 시간에 무관하게 일정한 분포를 유지하며, 미래의 값들을 예측할 때 현재의 값이나 과거의 값과 상관없이 일정한 경향성을 보입니다.

이러한 시계열 데이터는 예측 분석을 수행하기 위해 필요한 전제 조건 중 하나입니다.

ACF ,PACF 확인

차분을 실시 한 후 ACF와 PACF 확인



ACF와 PACF가 0으로 빠르게 수렴해야 정상 데이터

ACF : 느리게 감소
PACF : 8 이후부터 0으로 수렴
모든 '구'들이 같은 개형

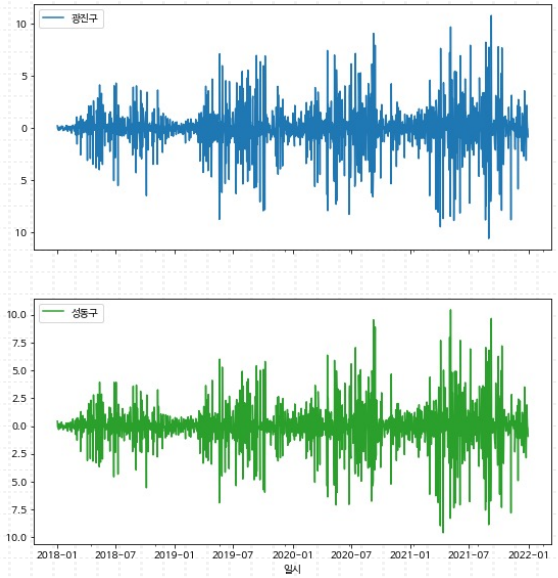
차분을 실시 했으나아직 정상성을 만족 X

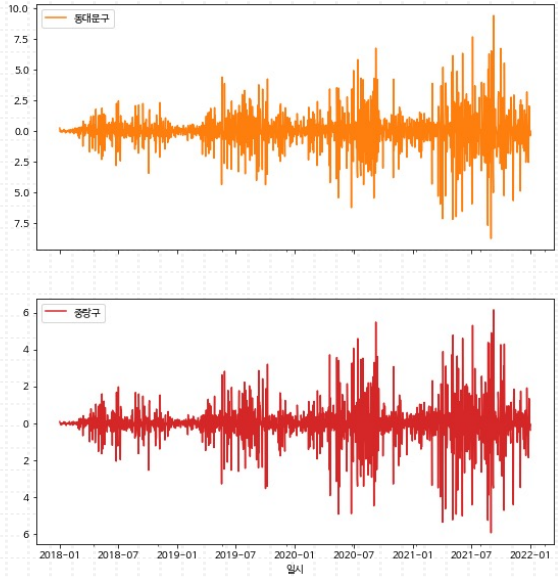
시계열 성분 분석

ACF, PACF란?

두 함수는 시계열 데이터에서 발생하는 상관관계를 파악하는데 사용되는 함수다.
ACF는 자기상관 함수로 시차가 커질 수록 ACF는 0에 가까워지는데, 정상시계열일 수록 빠르게 0으로 수렴하고 비정상 시계열 일수록 천천히 감소한다.
이때, 자기 상관이란 자신과 자신의 일정 시간 만큼 미래의 값을 비교하여 상관관계를 파악하는 지표디..
PACF는 편자기상관계수로 순수하게 시차가 다른 두 지점 간의 상관계수라 두 시차 사이의 모든 시차의 영향을 제거한 계수다. 따라서 시차가 증가함에 따라 급격히 감소하는데 이는 해당 시차에서 유의한 영향력을 가지고 있다는 것을 의미한다.

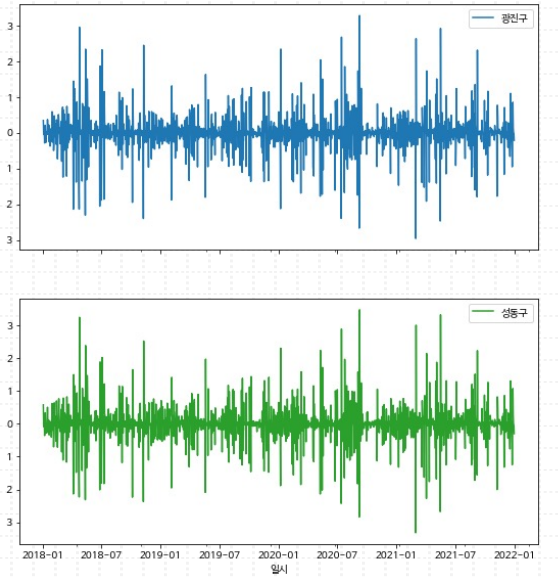
ACF ,PACF 확인

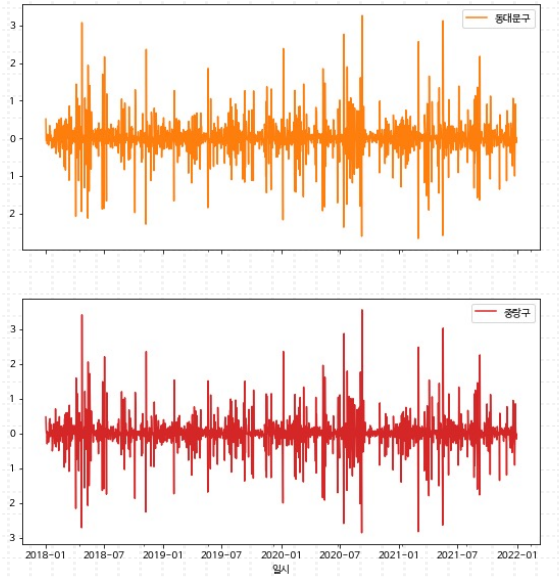




차분만 실시

중간에 주기성이 존재





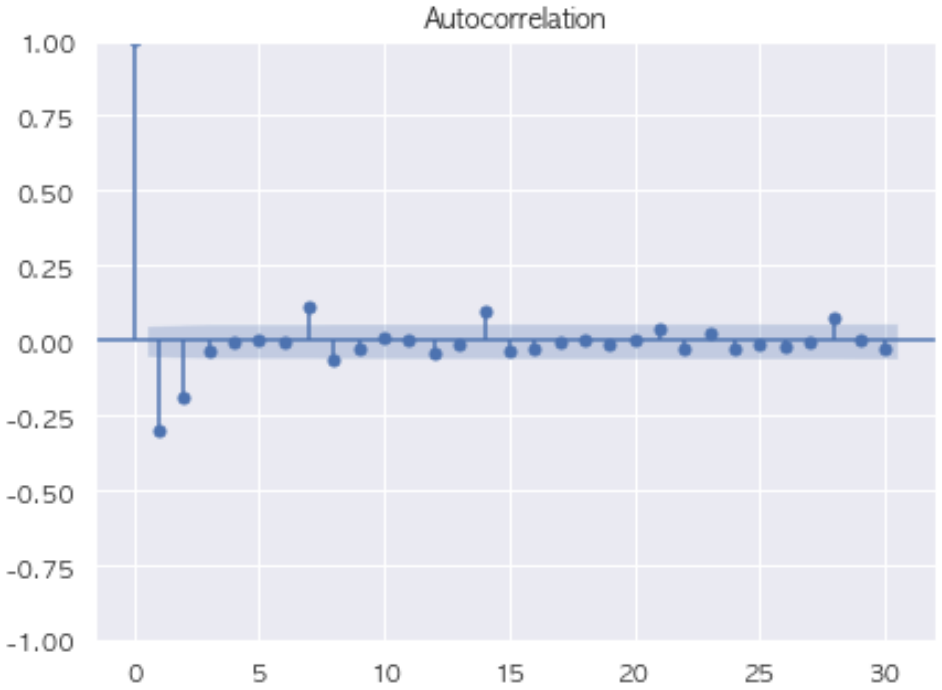
log 변환, 차분 실시

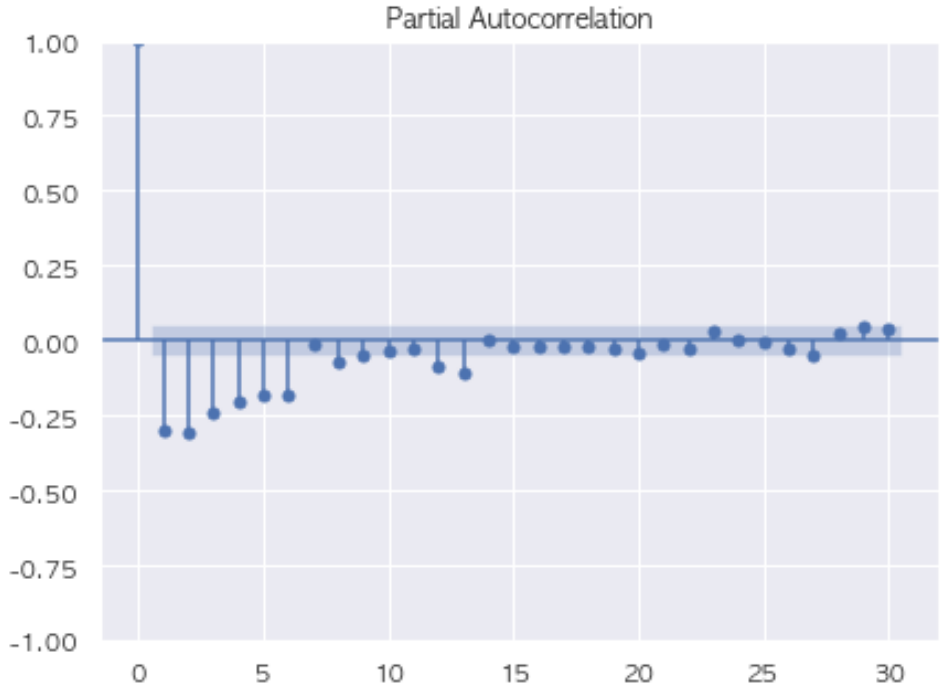
주기성이 남아 있지 않음

-> 이걸로 결정

ACF ,PACF 확인

4P





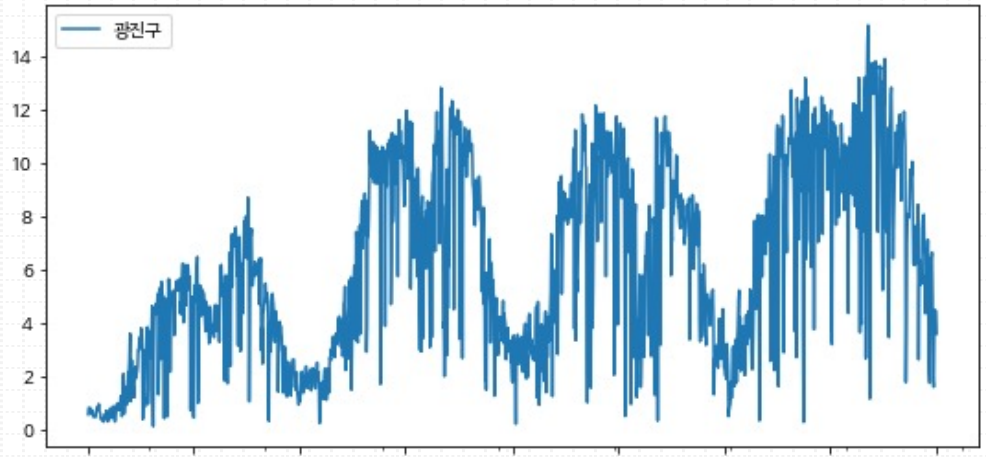
ACF : 1 이후로 0으로 수렴

PACF : 2 이후로 0으로 수렴

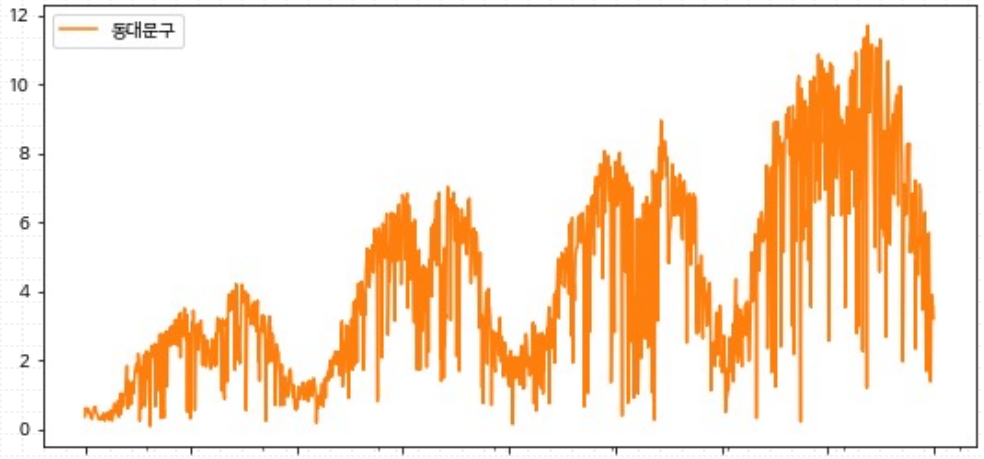
=> ARIMA 수행시 AR(1), MA(2)과 관련한 조합으로 하면 성능이 좋을 것이라 추측

아이디어 도출 계기

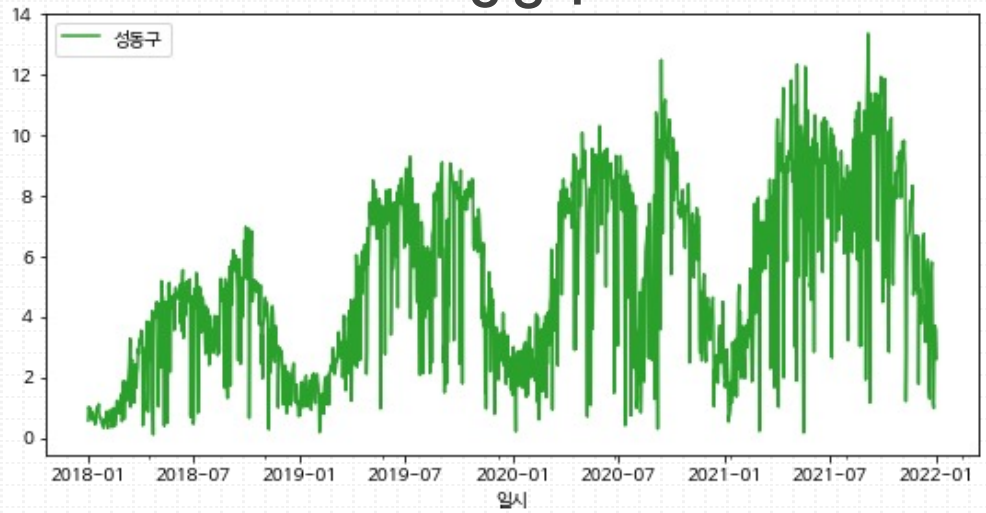
광진구



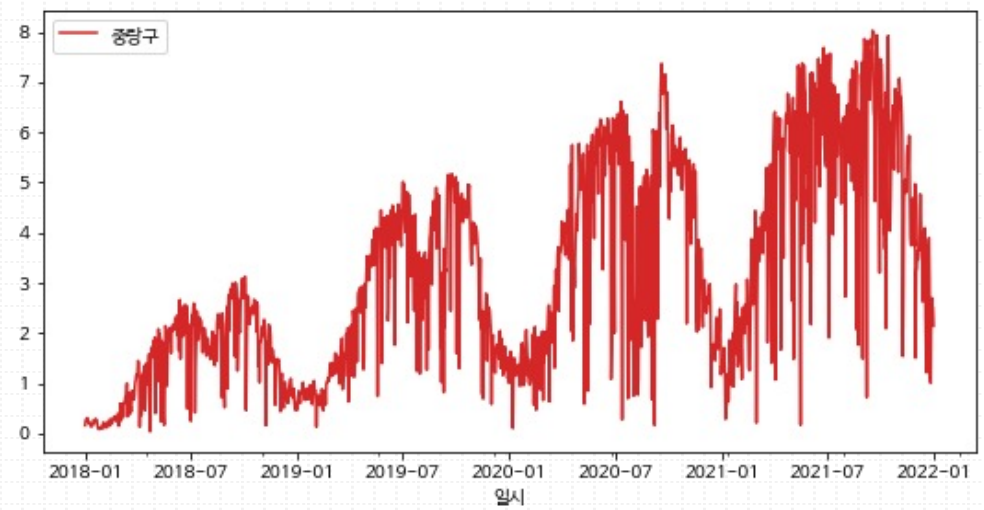
동대문구



성동구



중랑구



새로운 아이디어


```
data.set_index('일시',inplace=True)
data.index=pd.to_datetime(data.index,format='%Y%m%d')
data['week']=data.index.isocalendar().week
data['day_name']=data.index.weekday
data['weekend']=data['day_name'].apply(lambda x: 0 if x<5 else 1)
data['year']=data.index.year


sample_submission.set_index('일시',inplace=True)
sample_submission.index=pd.to_datetime(sample_submission.index,format='%Y%m%d')
sample_submission['week']=sample_submission.index.isocalendar().week
sample_submission['day_name']=sample_submission.index.weekday
sample_submission['weekend']=sample_submission['day_name'].apply(lambda x: 0 if x<5 else 1)
sample_submission['year']=sample_submission.index.year
```

```
# 년도별 공휴일 체크하기위한 library => 평일인 공휴일도 주말로 설정.

from pytimekr import pytimekr
def get_holiday(_year):
    holidays=[]
    for holiday in pytimekr.holidays(year=_year):
        if pytimekr.red_days(holiday) != None:
            ans = [i.strftime("%Y-%m-%d") for i in pytimekr.red_days(holiday)]
            holidays.extend(ans)
        else:
            ans=holiday.strftime("%Y-%m-%d")
            holidays.append(ans)
    return list(set(holidays))

data.loc[get_holiday(2018),'weekend'] = 1
data.loc[get_holiday(2019),'weekend'] = 1
data.loc[get_holiday(2020),'weekend'] = 1
data.loc[get_holiday(2021),'weekend'] = 1
sample_submission.loc[sorted(get_holiday(2022))[:-1],'주말평일'] ='주말'
```

1년을 주차별로 적용 - 총 52주차

평일, 주말로 구분

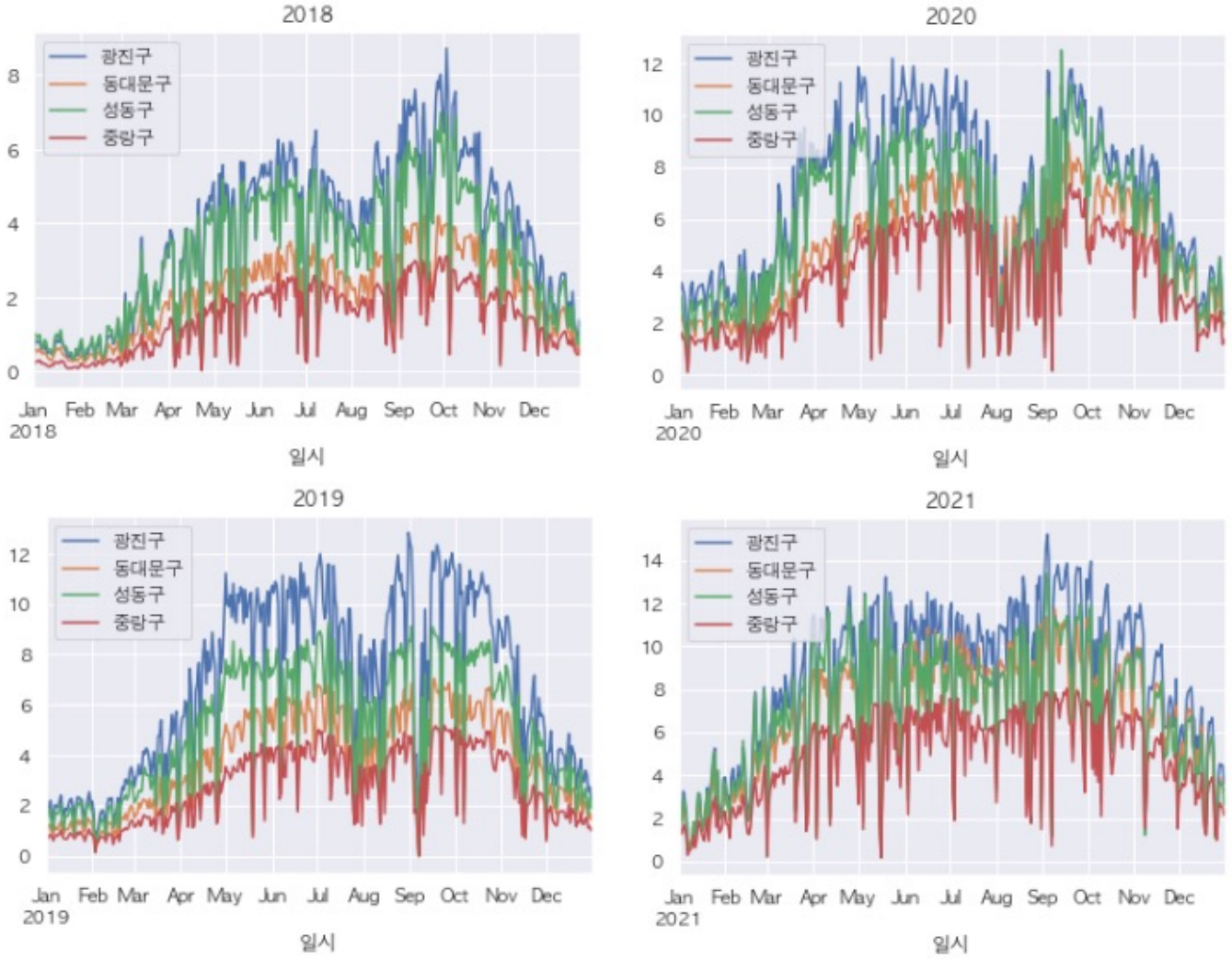
dayname = 요일

Weekday = 평일주말

Pytimekr 라이브러리를 활용하여

한국의 공휴일을 주말 처리

새로운 아이디어

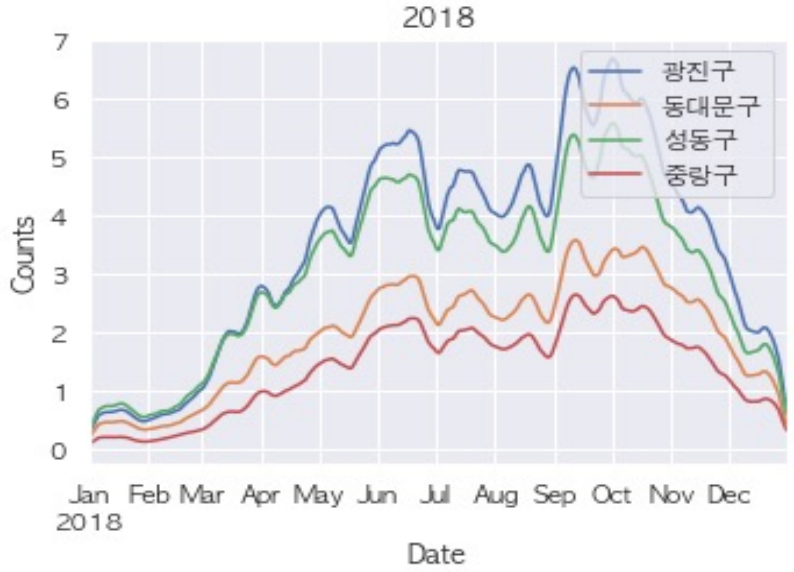


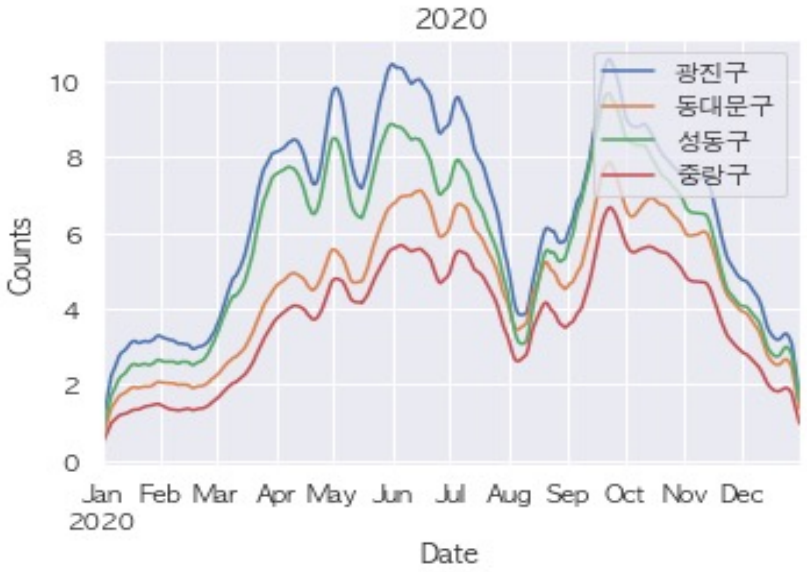

```
data[data.index.year == 2018].iloc[:,:4].plot(title='2018')
data[data.index.year == 2019].iloc[:,:4].plot(title='2019')
data[data.index.year == 2020].iloc[:,:4].plot(title='2020')
data[data.index.year == 2021].iloc[:,:4].plot(title='2021')
```

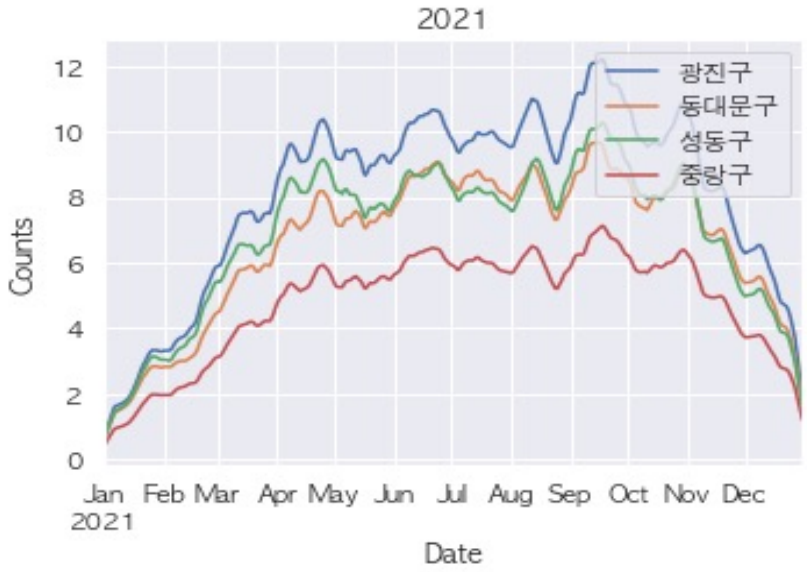
연도별로 묶으면

그래프 노이즈가 심함

새로운 아이디어







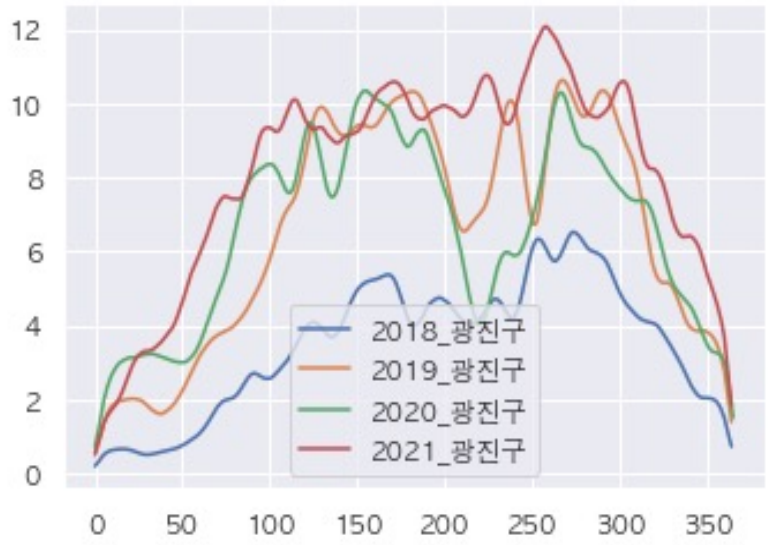

```python
def smooth(y, box_pts):
    box = np.ones(box_pts)/box_pts
    y_smooth = np.convolve(y, box, mode='same')
    return y_smooth

temp_2018=data[data.index.year == 2018].iloc[:,:4]
for col in temp_2018:
    temp_2018[col] = smooth(temp_2018[col],10)
temp_2018.plot(title='2018')
temp_2019=data[data.index.year == 2019].iloc[:,:4]
for col in temp_2019:
    temp_2019[col] = smooth(temp_2019[col],10)
temp_2019.plot(title='2019')
temp_2020=data[data.index.year == 2020].iloc[:,:4]
for col in temp_2020:
    temp_2020[col] = smooth(temp_2020[col],10)
temp_2020.plot(title='2020')
temp_2021=data[data.index.year == 2021].iloc[:,:4]
for col in temp_2021:
    temp_2021[col] = smooth(temp_2021[col],10)
temp_2021.plot(title='2021')
```

Smooth 이용해서

보기 편하게 바꿈

새로운 아이디어




```python
from pytimekr import pytimekr
def get_holiday(_year):
    holidays=[]
    for holiday in pytimekr.holidays(year=_year):
        if pytimekr.red_days(holiday) != None:
            ans = [i.strftime("%Y-%m-%d") for i in pytimekr.red_days(holiday)]
            holidays.extend(ans)
        else:
            ans=holiday.strftime("%Y-%m-%d")
            holidays.append(ans)
    return list(set(holidays))

data.loc[get_holiday(2018),'weekend'] = 1
data.loc[get_holiday(2019),'weekend'] = 1
data.loc[get_holiday(2020),'weekend'] = 1
data.loc[get_holiday(2021),'weekend'] = 1
sample_submission.loc[sorted(get_holiday(2022))[:-1],'주말평일'] ='주말'
```

연도 별 구별 그래프

연도별 공휴일 체크하기위한 library

=> 평일인 공휴일도 주말로 설정

ex) 설날, 추석

새로운 아이디어


```
# 년도별 주별 (평일,주말) 별 평균값 구하기
data_week_groupby=data.groupby(['year','week','weekend']).mean().reset_index()
```

	year	week	weekend	광진구	동대문구	성동구	중랑구	day_name
0	2018	1	0	0.934400	0.632400	1.020000	0.350400	2.000000
1	2018	1	1	0.602667	0.456000	0.627333	0.209333	3.666667
2	2018	2	0	0.585600	0.417200	0.702800	0.168400	2.000000
3	2018	2	1	0.494000	0.355000	0.504000	0.179000	5.500000
4	2018	3	0	0.831600	0.602400	0.993200	0.256000	2.000000
...	...	...	...	...	...	...	...	...
412	2021	50	1	2.043000	1.736000	1.473000	1.386000	5.500000
413	2021	51	0	5.991200	5.163200	5.117600	3.500000	2.000000
414	2021	51	1	1.783000	1.426000	1.070000	1.125000	5.500000
415	2021	52	0	4.178000	3.522800	3.286400	2.445200	2.000000
416	2021	53	1	2.016667	1.803333	1.769333	1.313333	5.000000

417 rows × 8 columns

연도별 주별 (평일, 주말)

별 평균값 구하기

새로운 아이디어

< 평일과 공휴일의 평균대여량 추세 >

마찬가지로 노이즈가 심해서 이동평균법을 통한 데이터 평활과 스케일링을 같이 진행

새로운 아이디어


```
# 각 연도별 minmaxscaling 진행
from sklearn.preprocessing import MinMaxScaler
mms_2018 = MinMaxScaler()
mms_2019 = MinMaxScaler()
mms_2020 = MinMaxScaler()
mms_2021 = MinMaxScaler()
```

```
year_df_list=[df_2018,df_2019,df_2020,df_2021]
scaler_list=[mms_2018,mms_2019,mms_2020,mms_2021]
for df,mms in zip(year_df_list,scaler_list):
    df.iloc[:,2:] = mms.fit_transform(df.iloc[:,2:])
```

```
# 연도별 대여량을 스케일링 한후 주별,(평일,주말별) 대여량 평균구하기
week_mean = pd.concat(year_df_list).groupby(['week','weekend']).mean().reset_index()
```

새로운 아이디어

4P

```
# 각 연도별 minmaxscaling 진행
from sklearn.preprocessing imp
mms_2018 = MinMaxScaler()
mms_2019 = MinMaxScaler()
mms_2020 = MinMaxScaler()
mms_2021 = MinMaxScaler()
```

```
year_df_list=[df_2018,df_2019,d
scaler_list=[mms_2018,mms_2019,
for df,mms in zip(year_df_list,
    df.iloc[:,2:] = mms.fit_tra
```

```
# 연도별 대여량을 스케일링 한후 주별,(평
week_mean = pd.concat(year_df_l
```

week_mean

	week	weekend	광진구	동대문구	성동구	중랑구
0	1	0	0.000000	0.000000	0.000000	0.000000
1	1	1	0.034379	0.036138	0.037351	0.027567
2	2	0	0.069503	0.071094	0.073133	0.059770
3	2	1	0.105487	0.109272	0.112585	0.090830
4	3	0	0.099792	0.101856	0.104050	0.088229
...	...	...	...	...	...	...
101	51	1	0.197307	0.235297	0.191927	0.212765
102	52	0	0.131153	0.158553	0.124878	0.148527
103	52	1	0.091892	0.118654	0.092212	0.110412
104	53	0	0.028854	0.074243	0.051411	0.062857
105	53	1	0.088068	0.091297	0.072824	0.093768

106 rows × 6 columns

새로운 아이디어