

캡스톤 기말 발표

ARIMA MODEL

- Team 1조 ASMR

2018540019 송준호

2018540000 강지호

2018540007 곽동우

Contents

중간발표 리뷰

데이터 설명

코드 설명

결론

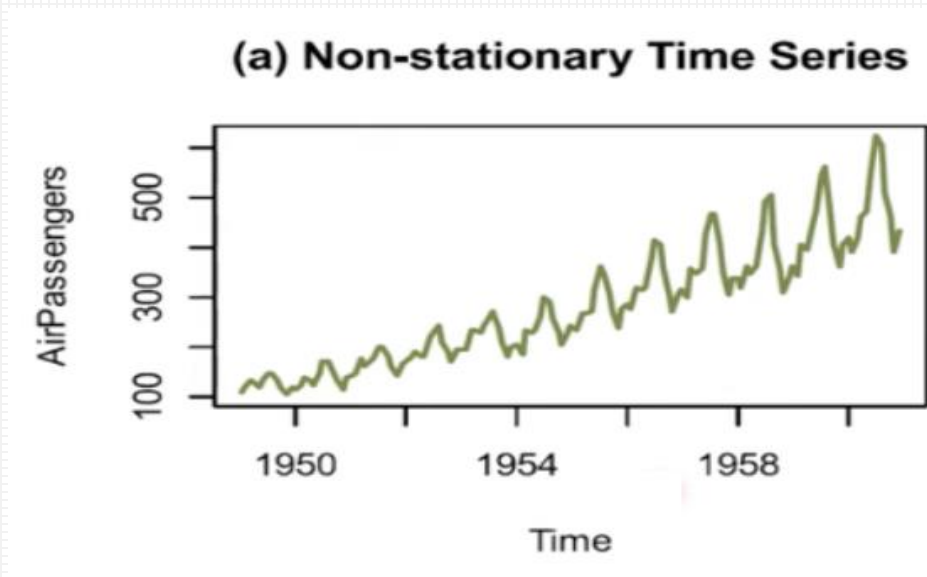


Part 1

중간발표 리뷰



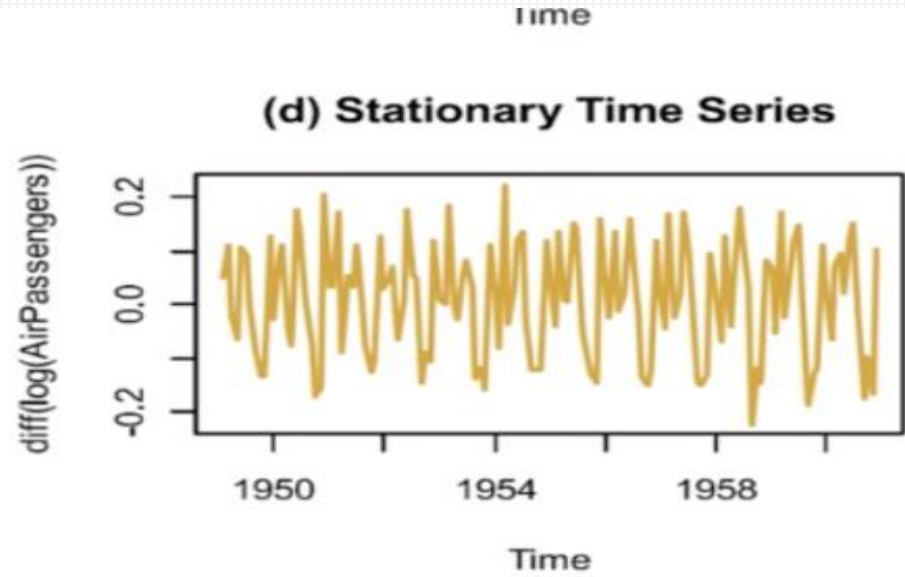
비정상 시계열



-시간에 따라 평균과 분산이
일정하지 않은 경우

-대부분의 시계열 자료

정상 시계열



-시간의 변화에 따라 평균을 중심으로
일정한 변동폭을 갖는 시계열

-대부분 비정상 시계열 데이터이므로
정상 시계열로 변환하여 분석

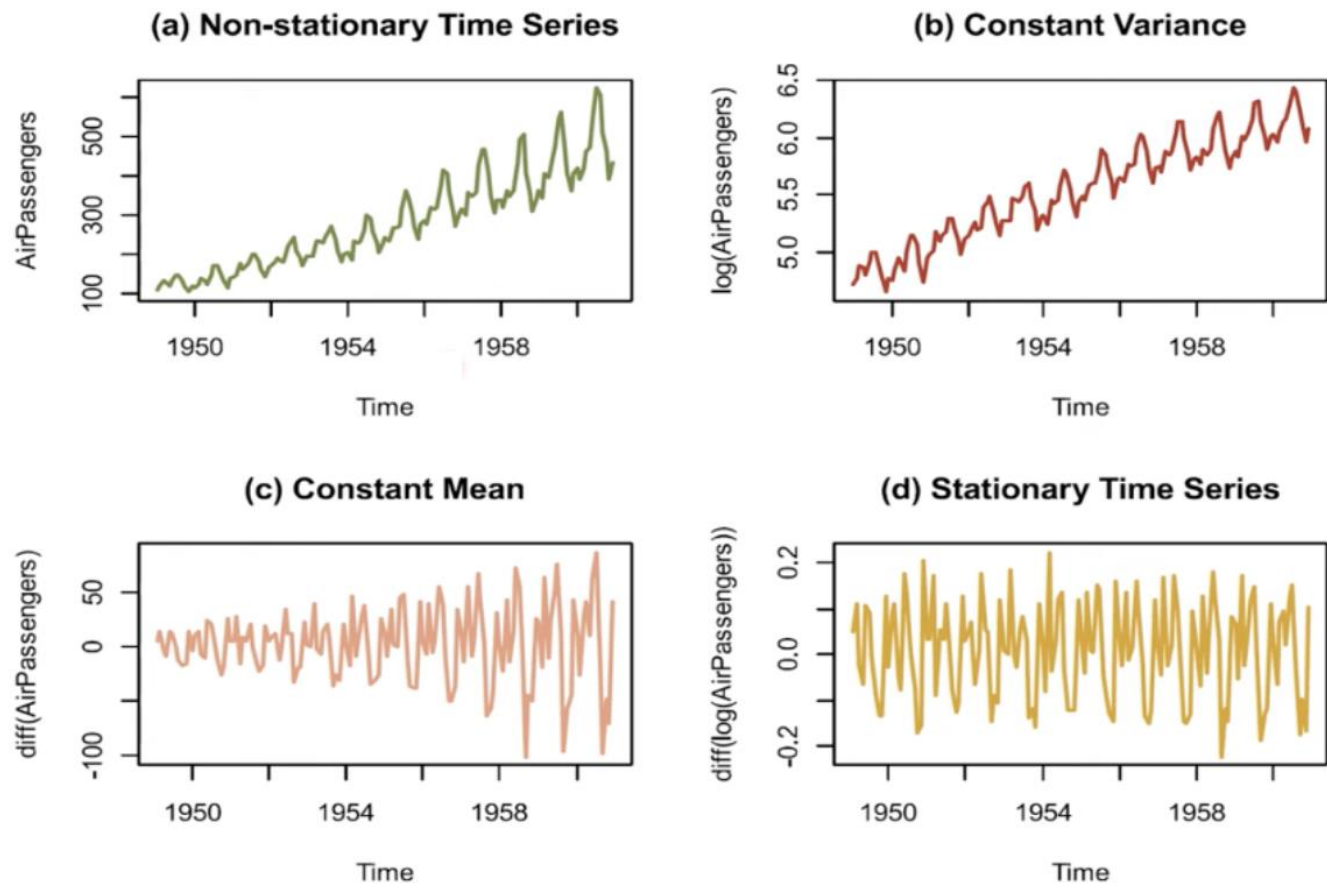


정상성

평균, 분산이 시간에 따라 일정한 성질

정상 시계열 변환

변동폭이 일정하지 않은 경우 --> 로그변환
추세, 계절성이 존재하는 경우 --> 차분(반복)



ACF(자기상관함수)

- 시차에 따른 자기상관을 의미하며, 정상시계열은 빠르게 0에 수렴하고, 비정상 시계열은 천천히 감소
- ACF는 정상성을 판단하는데 유용
- 수식

$$ACF(k) = \frac{\sum_{t=1}^{N-k} (y_t - \bar{y})(y_{t+k} - \bar{y})}{\sum_{t=1}^N (y_t - \bar{y})^2}$$

AIC(아카이케 정보 기준)

- 최소의 정보 손실을 기준으로 모델을 선택하는 방법
- AIC 값이 낮을수록 정보 손실이 적음
- 수식

$$AIC = -2 \ln(L) + 2k$$

ARIMA(p,d,q) 모형이란?

정의

ARIMA(p,d,q)모형은 d차 차분한 데이터에 위 AR(p) 모형과 MA(q) 모형을 합친 모형

수식

$$y'_t = c + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + \dots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$



Part 2

데이터 설명

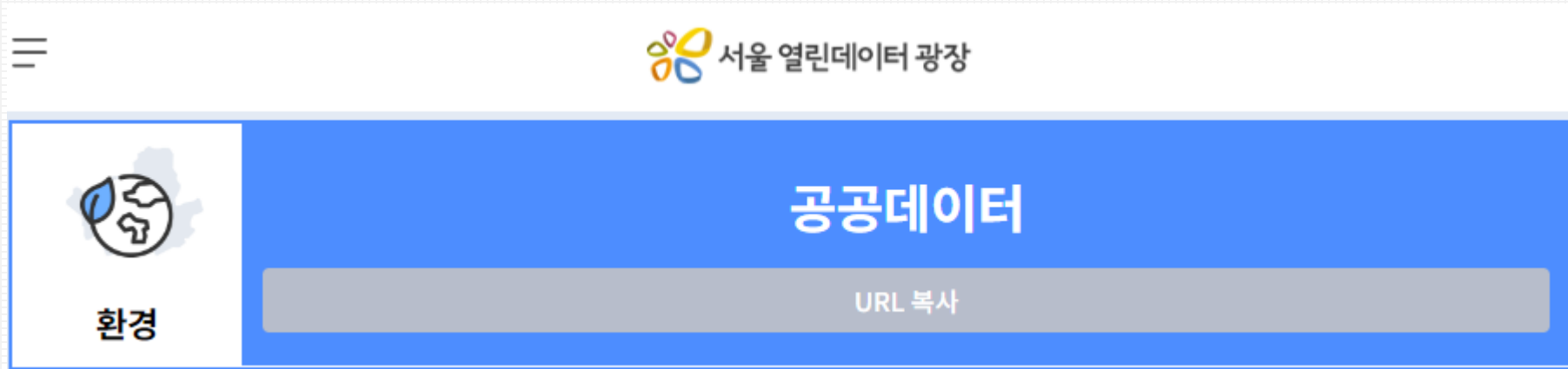
PM2.5(초미세먼지)



미세먼지 : 입자의 지름이 10마이크로미터(μm) 이하인 먼지(PM-10)

초미세먼지 : 입자의 지름이 2.5마이크로미터(μm) 이하인 먼지(PM-2.5)

데이터 소개



서울 열린데이터 광장의 공공데이터

서울시 대기오염 측정정보, 서울시 대기오염 측정소 정보,

서울시 대기오염 측정항목

서울시 대기오염 측정정보 데이터

	Measurement date	Station code	Item code	Average value	Instrument status
0	2017-01-01 00:00	101	1	0.004	0
1	2017-01-01 00:00	101	3	0.059	0
2	2017-01-01 00:00	101	5	1.200	0
3	2017-01-01 00:00	101	6	0.002	0
4	2017-01-01 00:00	101	8	73.000	0
...
3885061	2019-12-31 23:00	123	9	13.000	0
3885062	2019-12-31 23:00	118	9	24.000	0
3885063	2019-12-31 23:00	105	8	19.000	0
3885064	2019-12-31 23:00	125	3	0.037	0
3885065	2019-12-31 23:00	108	3	0.030	0

Measurement date:
측정 날짜 및 시간

Station code:
측정소 위치 코드

Item code:
측정기 상태

Average value:
평균 측정값

서울시 대기오염 측정소 정보 데이터

	Station code	Station name(district)	Address	Latitude	Longitude
0	101	Jongno-gu	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005008
1	102	Jung-gu	15, Deoksugung-gil, Jung-gu, Seoul, Republic o...	37.564263	126.974676
2	103	Yongsan-gu	136, Hannam-daero, Yongsan-gu, Seoul, Republic...	37.540033	127.004850
3	104	Eunpyeong-gu	215, Jinheung-ro, Eunpyeong-gu, Seoul, Republi...	37.609823	126.934848
4	105	Seodaemun-gu	32, Segeomjeong-ro 4-gil, Seodaemun-gu, Seoul,...	37.593742	126.949679
5	106	Mapo-gu	10, Poeun-ro 6-gil, Mapo-gu, Seoul, Republic o...	37.555580	126.905597
6	107	Seongdong-gu	18, Ttukseom-ro 3-gil, Seongdong-gu, Seoul, Re...	37.541864	127.049659
7	108	Gwangjin-gu	571, Gwangnaru-ro, Gwangjin-gu, Seoul, Republi...	37.547180	127.092493
8	109	Dongdaemun-gu	43, Cheonho-daero 13-gil, Dongdaemun-gu, Seoul...	37.575743	127.028885
9	110	Junngang-gu	369, Yongmasan-ro, Junngang-gu, Seoul, Republi...	37.584848	127.094023
10	111	Seongbuk-gu	70, Samyang-ro 2-gil, Seongbuk-gu, Seoul, Repu...	37.606719	127.027279
11	112	Gangbuk-gu	49, Samyang-ro 139-gil, Gangbuk-gu, Seoul, Rep...	37.647930	127.011952
12	113	Dobong-gu	34, Sirubong-ro 2-gil, Dobong-gu, Seoul, Repub...	37.654192	127.029088
13	114	Nowon-gu	17, Sanggye-ro 23-gil, Nowon-gu, Seoul, Republ...	37.658774	127.068505

Station code:
측정소 위치 코드

Station name(district):
측정소가 위치하는 구 이름

Address:
측정소 주소

Latitude:
측정소 위도

Longitude:
측정소 경도

서울시 대기오염 측정항목 정보 데이터

	Item code	Item name	Unit of measurement	Good(Blue)	Normal(Green)	Bad(Yellow)	Very bad(Red)
0	1	SO2	ppm	0.02	0.05	0.15	1.0
1	3	NO2	ppm	0.03	0.06	0.20	2.0
2	5	CO	ppm	2.00	9.00	15.00	50.0
3	6	O3	ppm	0.03	0.09	0.15	0.5
4	8	PM10	Mircrogram/m3	30.00	80.00	150.00	600.0
5	9	PM2.5	Mircrogram/m3	15.00	35.00	75.00	500.0

Item name: 측정 항목 이름

Unit of measurement: 측정 단위


세 가지 데이터를 엑셀로 합친 데이터

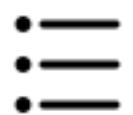
	Measurement date	Station code	Address	Latitude	Longitude	SO2	NO2	O3	CO	PM10	PM2.5
0	2017-01-01 00:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.004	0.059	0.002	1.2	73.0	57.0
1	2017-01-01 01:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.004	0.058	0.002	1.2	71.0	59.0
2	2017-01-01 02:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.004	0.056	0.002	1.2	70.0	59.0
3	2017-01-01 03:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.004	0.056	0.002	1.2	70.0	58.0
4	2017-01-01 04:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.003	0.051	0.002	1.2	69.0	61.0
...
647506	2019-12-31 19:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.028	0.013	0.5	23.0	17.0
647507	2019-12-31 20:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.025	0.015	0.4	25.0	19.0
647508	2019-12-31 21:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.023	0.015	0.4	24.0	17.0
647509	2019-12-31 22:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.040	0.004	0.5	25.0	18.0
647510	2019-12-31 23:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.037	0.005	0.5	27.0	18.0



Part 3

코드 설명





+ 모듈과 데이터 불러오기



```
[ ] import numpy as np
import pandas as pd
import os
import json
import geopandas as gpd
import seaborn as sns
import matplotlib.pyplot as plt
import folium
from folium import Choropleth, Circle, Marker
from folium.plugins import HeatMap, MarkerCluster
from IPython.display import display, Markdown
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 20, 16
import warnings
import itertools
warnings.filterwarnings("ignore")
import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint, adfuller
```

Measurement station info :
측정소 정보 데이터
Measurement info :
대기오염 측정정보 데이터
Measurement item info :
대기오염 측정항목 데이터
Measurement summary :
위 세가지 데이터를 요약한 데이터

```
[ ] stations = pd.read_csv('Measurement_station_info.csv')
measurements = pd.read_csv('Measurement_info.csv')
items = pd.read_csv('Measurement_item_info.csv')
df = pd.read_csv('Measurement_summary.csv')
```



+ Datetime으로 형식 전환 및 데이터



```
[ ] df['Measurement date'] = pd.to_datetime(df['Measurement date'])
```

해당열을 datetime 형태의 데이터로 변형하기 위해 to_datetime 함수를 사용

	Item code	Item name	Unit of measurement	Good(Blue)	Normal(Green)	Bad(Yellow)	Very bad(Red)
0	1	SO2	ppm	0.02	0.05	0.15	1.0
1	3	NO2	ppm	0.03	0.06	0.20	2.0
2	5	CO	ppm	2.00	9.00	15.00	50.0
3	6	O3	ppm	0.03	0.09	0.15	0.5
4	8	PM10	Mircrogram/m3	30.00	80.00	150.00	600.0
5	9	PM2.5	Mircrogram/m3	15.00	35.00	75.00	500.0

‘items’ 데이터

	Measurement date	Station code	Address	Latitude	Longitude	SO2	NO2	O3	CO	PM10	PM2.5
0	2017-01-01 00:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.004	0.059	0.002	1.2	73.0	57.0
1	2017-01-01 01:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.004	0.058	0.002	1.2	71.0	59.0
2	2017-01-01 02:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.004	0.056	0.002	1.2	70.0	59.0
3	2017-01-01 03:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.004	0.056	0.002	1.2	70.0	58.0
4	2017-01-01 04:00	101	19, Jong-ro 35ga-gil, Jongno-gu, Seoul, Republ...	37.572016	127.005007	0.003	0.051	0.002	1.2	69.0	61.0
...
647506	2019-12-31 19:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.028	0.013	0.5	23.0	17.0
647507	2019-12-31 20:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.025	0.015	0.4	25.0	19.0
647508	2019-12-31 21:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.023	0.015	0.4	24.0	17.0
647509	2019-12-31 22:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.040	0.004	0.5	25.0	18.0
647510	2019-12-31 23:00	125	59, Gucheonmyeon-ro 42-gil, Gangdong-gu, Seoul...	37.544962	127.136792	0.003	0.037	0.005	0.5	27.0	18.0

‘df’ 데이터



+ 데이터 전처리



```
[ ] df_109 = pd.DataFrame(df.loc[(df['Station code']==109)])  
df_109.head()  
df_109.drop("Station code", axis=1, inplace=True)
```

동대문구 측정소를 사용할 것이고,
loc 함수로 해당 열만 추출해서
df_109에 저장

```
[3] drop_all = df.loc[(df_109['SO2']<0) | (df_109['NO2']<0) | (df_109['CO']<0) | (df_109['O3']<0)]  
drop_PM = df.loc[(df_109['PM2.5']<0) | (df_109['PM10']<0) | (df_109['PM2.5']==0) | (df_109['PM10']==0)]  
  
drop_index = drop_all.index.append(drop_PM.index)  
df_new = df.drop(drop_index, axis=0)
```

결측치가 음의 값 혹은 0의 값으로 나오므로
데이터셋을 전처리하여 음의 값을 제거

```
[ ] df_new['Measurement date'] = pd.datetime(df_new['Measurement date'], format='%Y-%m-%d')  
df_new.set_index('Measurement date', drop=True, inplace=True)  
df_new.dropna(inplace = True)
```

Measurement date행을
datetime 함수로 통해
datetime으로 변환



+ 정상성 확인 함수 정의



```
[ ] def Plot(ts):
    rol_mean = ts.rolling(window = 12, center = False).mean()
    rol_std = ts.rolling(window = 12, center = False).std()

    plt.plot(ts, color = 'blue', label = 'Original Data')
    plt.plot(rol_mean, color = 'red', label = 'Rolling Mean')
    plt.plot(rol_std, color = 'black', label = 'Rolling Std')
    plt.xticks(fontsize = 25)
    plt.yticks(fontsize = 25)

    plt.xlabel('Time in Years', fontsize = 25)
    plt.ylabel('Total Emissions', fontsize = 25)
    plt.legend(loc='best', fontsize = 25)
    plt.title('Rolling Mean & Standard Deviation', fontsize = 25)
    plt.show(block= True)
```

정상성 그래프를 그려줄 함수 정의

이동평균, 이동 분산을 각각 rol_mean,
rol_std에 저장

파란색이 원래 데이터, 빨강색이 이동 평균,
검은색이 이동 분산

```
[ ] def Adfuller(ts, cutoff = 0.01):
    ts_test = adfuller(ts, autolag = 'AIC')
    ts_test_output = pd.Series(ts_test[0:4], index=[ 'Test Statistic',
                                                    'p-value',
                                                    '#Lags Used',
                                                    'Number of Observations Used'])

    for key,value in ts_test[4].items():
        ts_test_output['Critical Value (%)'%key] = value
    print(ts_test_output)
```

정상성 통계검정 함수를 정의

ACF-test를 해주는 adfuller 함수 사용

귀무가설의 기각여부 확인 가능
(귀무가설 : 데이터가 비정상적이다)



+ 데이터의 정상성 확인



```
[ ] df_109 = pd.DataFrame(df_new.loc[(df_new['Station code']==109)])  
    df_109 = df_109.set_index("Measurement date")  
    df_25 = df_109.iloc[:, -1:]  
    df_25
```

```
[ ] Plot(df_25)  
    Adfuller(df_25)
```

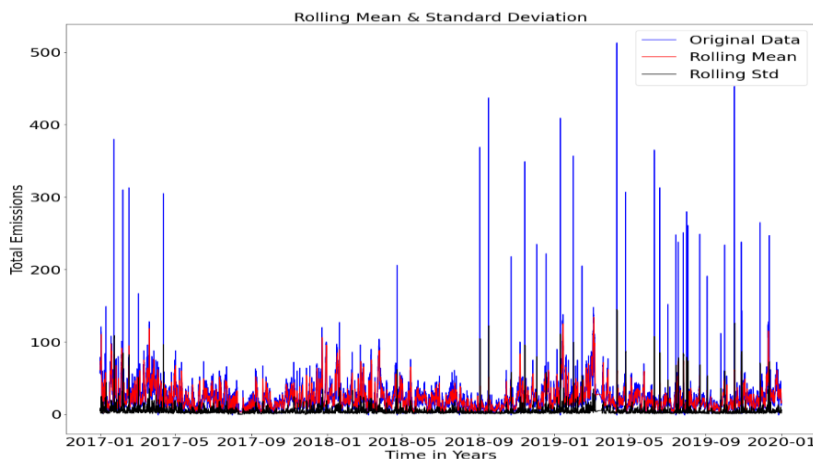
전처리한 데이터와 정의한 함수를 사용해 ARIMA분석 실시

loc함수를 사용해 PM2.5(초미세먼지)만 추출, 이를 변수 df_25에 저장

Plot 함수와 Adfuller함수를 사용하여 데이터의 정상성 확인



+ 데이터의 정상성 확인



Test Statistic	-1.713771e+01
p-value	7.079278e-30
#Lags Used	1.900000e+01
Number of Observations Used	2.577600e+04
Critical Value (1%)	-3.430604e+00
Critical Value (5%)	-2.861652e+00
Critical Value (10%)	-2.566830e+00

```
# Plot(df_25)
```

```
# 파란색이 원래 데이터, 빨강색이 이동 평균,  
검은색이 이동 분산
```

```
# Adfuller(df_25)
```

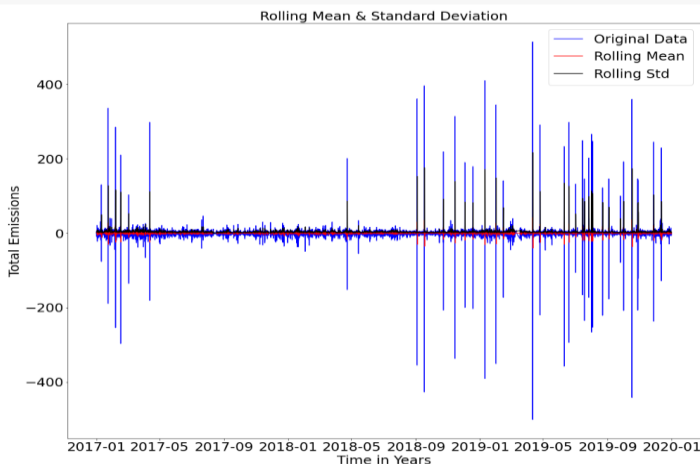
```
# 검정통계량 각 유의 수준이 1, 5, 10%보다 작기  
때문에, 정상성이 의심
```

```
# 데이터를 차분해서 정상성을 다시 확인
```

+ 1차 차분



```
[ ] first_difference = df_25 - df_25.shift(1)
Plot(first_difference.dropna(inplace=False))
Adfuller(first_difference.dropna(inplace=False))
```



Test Statistic	-31.513799
p-value	0.000000
#Lags Used	48.000000
Number of Observations Used	25746.000000
Critical Value (1%)	-3.430604
Critical Value (5%)	-2.861652
Critical Value (10%)	-2.566830

first_difference는 1차 차분식을 저장한 변수

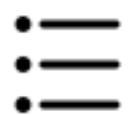
차분식은 (원래 데이터 - 원래 데이터)를 1만큼 옮긴 식

Plot(first_difference)

Adfuller(first_difference)

검정 통계량이 유의미하기 때문에 정상성을 만족

더 이상 차분을 진행하지 않고 완료



+ 적합한 ARIMA모형 찾기



```
[ ] p = d = q = range(0, 2) # 이 값들은 0~2사이의 값
    pdq = list(itertools.product(p, d, q))
    pdq_x_QDQs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
    print('Examples of Seasonal ARIMA parameter combinations for Seasonal ARIMA...')
    print('SARIMAX: {} x {}'.format(pdq[1], pdq_x_QDQs[1]))
    print('SARIMAX: {} x {}'.format(pdq[2], pdq_x_QDQs[2]))

[ ] start_day = '2017-01-01'
    end_day = '2019-12-31'
    con1=df_25.index>=start_day
    con2=df_25.index<=end_day
    df_25_train=df_25[con1&con2]
```

적합한 모형 ARIMA(p, d, q)을 찾는 코드

2017-01-01부터 2019-12-31까지 분석을 실시

적합한 p, d, q를 찾는다



+ 적합한 ARIMA모형 찾기



```
[ ] warnings.filterwarnings("ignore")
    for param in pdq:
        for param_seasonal in pdq_x_QDQs:
            try:
                mod = sm.tsa.statespace.SARIMAX(df_25_train,
                                                order=param,
                                                seasonal_order=param_seasonal,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False)

                results = mod.fit()

                print('ARIMA{0}x{1}12 - AIC:{2}'.format(param, param_seasonal, results.aic))
            except:
                continue
```

Grid search를 통해 최상의 파라미터 조합 선택

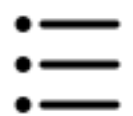
format을 사용해서 (p, d, q) 및 AIC 출력

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:249183.36421274475
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:236870.49787729746
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:228012.69458152927
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:221391.88207212024
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:224990.18637270405
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:221067.73415727346
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:223864.42146046282
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:219624.14557846077
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:231197.18523412265
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:224569.35634672883
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:222776.05964509866
(이하생략)
```

AIC값이 가장 낮은 ARIMA 모형이 적합한 모형

#ARIMA(1, 1, 1)x(1, 1, 1, 12)12 -
AIC:203266.35797617148 를 채택

#해당 모형으로 데이터 Training 진행



+ Training 및 잔차 검정



```
[ ] mod = sm.tsa.statespace.SARIMAX(df_25_train,
                                     order=(1, 1, 1),
                                     seasonal_order=(1, 1, 1, 12),
                                     enforce_stationarity=False,
                                     enforce_invertibility=False)
```

해당 계수로 데이터를 Training

```
results = mod.fit()
```

summar를 사용해 모델이 데이터에
잘 fitting 되었는지 확인

```
print(results.summary().tables[1])
```

print함수로 분석표 출력

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.2433	0.002	103.056	0.000	0.239	0.248
ma.L1	-0.7892	0.002	-360.333	0.000	-0.794	-0.785
ar.S.L12	-0.0087	0.005	-1.717	0.086	-0.019	0.001
ma.S.L12	-1.0011	0.001	-1953.838	0.000	-1.002	-1.000
sigma2	156.3456	0.173	902.871	0.000	156.006	156.685

fitting 모델의 결과의 p-value가 0.05
이상을 가짐

잔차의 검정 테스트를 잘 통과



+ 잔차 그래프

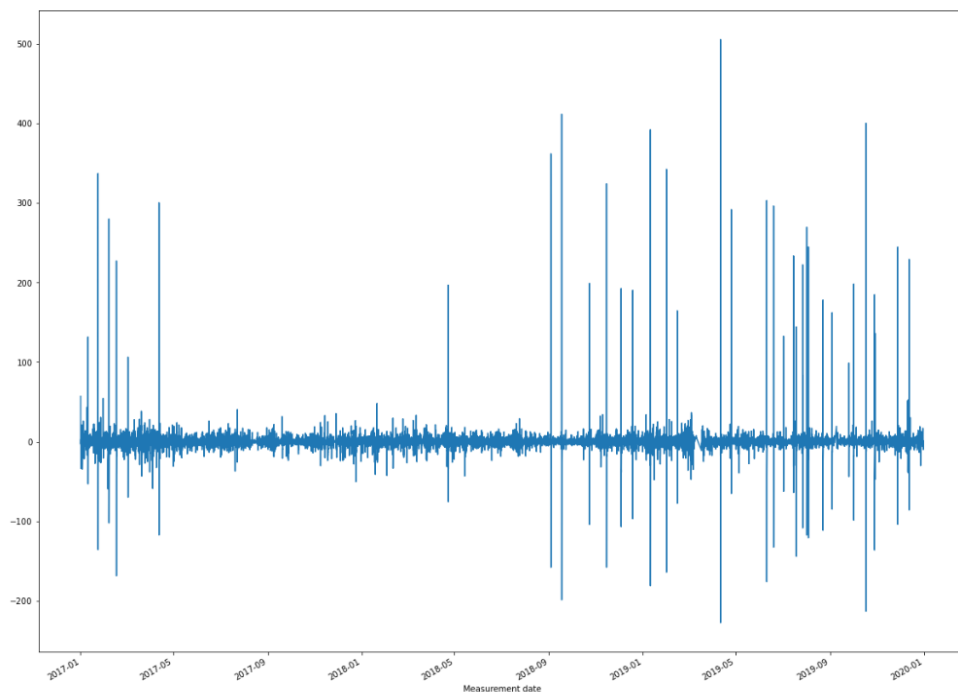


```
[ ] results.resid.plot()
```

잔차 그래프를 그리는 코드

```
[ ] print(results.resid.describe())
```

그래프에 대한 정보를 프린트



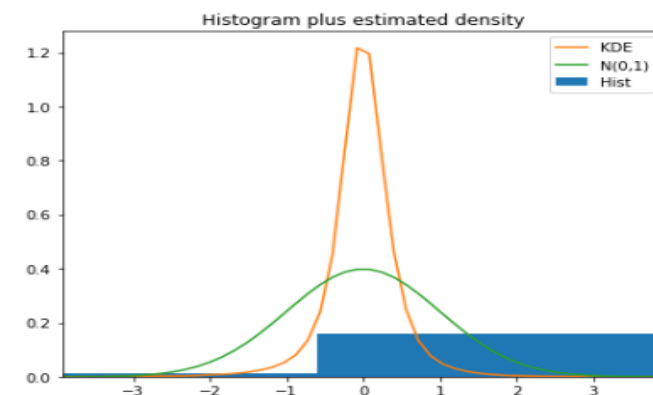
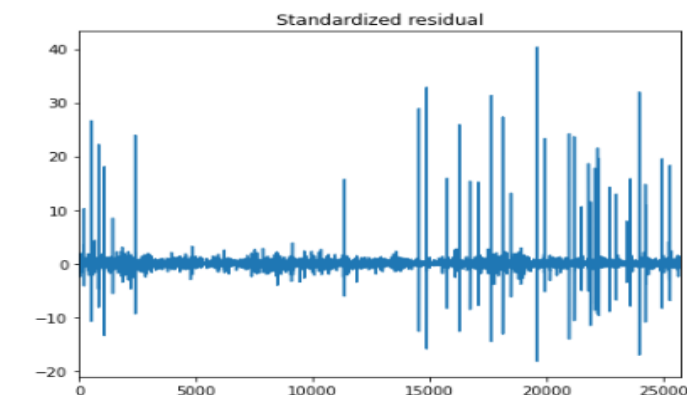
count	25773.000000
mean	0.026306
std	12.536952
min	-227.231885
25%	-2.437688
50%	-0.065331
75%	2.370695
max	505.209292
dtype:	float64

+ 잔차 검정 그래프



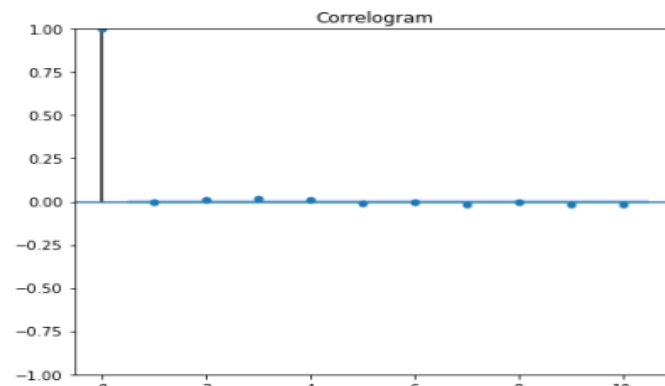
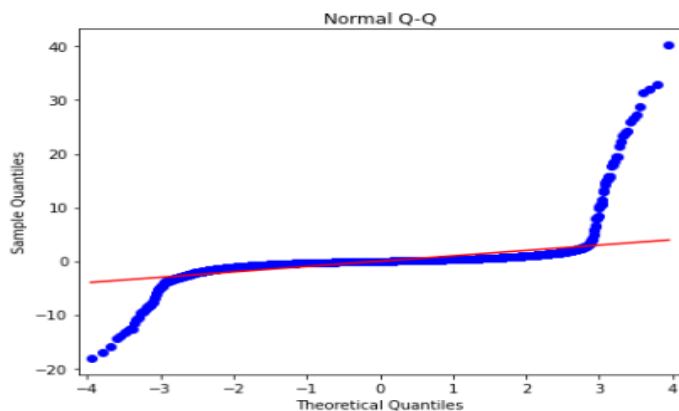
```
[ ] results.plot_diagnostics(figsize=(15, 12))
plt.show()
```

잔차 검정 그래프를 출력



Normal Q-Q 그래프로
잔차의 정규성을 확인

Correlogram은 ACF의
시각화의 표준적인 방법



Correlogram으로 잔차
가 정상성을 만족함을 보임



+ 예측값 그래프



```
[ ] pred = results.get_prediction(start=pd.to_datetime('2019-12-01'), dynamic=False)
    pred_ci = pred.conf_int()

    ax = df_25['2017-01:'].plot(label='observed')
    pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7)

    ax.fill_between(pred_ci.index,
                    pred_ci.iloc[:, 0],
                    pred_ci.iloc[:, 1], color='k', alpha=.2)

    ax.set_xlabel('Date')
    ax.set_ylabel('PM2.5 Levels')
    plt.legend()

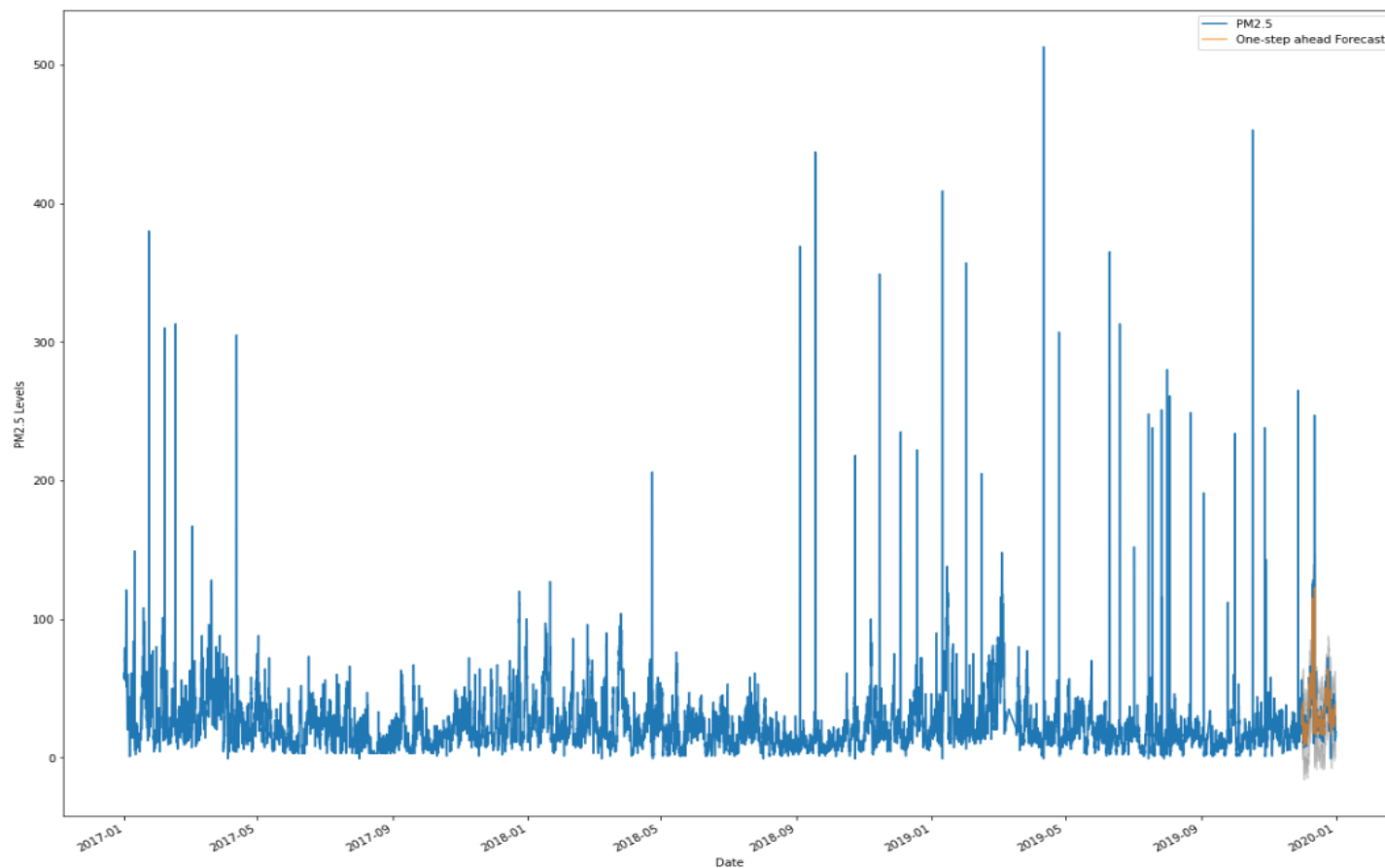
    plt.show()
```

시작은 2019-12-01 부터 예측값 그래프를 그리도록 설정

실제 데이터의 그래프를 2017-01부터 그리도록 설정 후 그래프의 색깔을 파랑으로 지정



+ 예측값 그래프



파란색:
PM2.5 그래프

주황색:
예측 값 그래프

잔차 검정으로 모델이 데이
터의 잘 fitting 됨

예측값 그래프 역시 기존 그
래프와 유사하게 나옴



Part 4

결론

분석의 타당성 설명

- ① ACF의 P-value가 0.05보다 작음을 통해 정상성을 확인
- ① AIC값이 가장 낮게 나오는 모형을 채택하여 ARIMA분석 진행
- ① 잔차분석을 해본 결과 모형의 타당성 검증

분석의 한계점 및 기대효과

⊗ 단점

- 시계열 데이터 한정으로만 효과적임
- 많은 양의 데이터를 필요하므로 계산비용이 큼
- 장기적인 예측 성능이 떨어짐

○ 장점

- 시계열 데이터에 대해 효과적임
- ARIMA 모형은 여러 프로그래밍 언어에 내장함수로 있어 접근성이 좋음

✦ 기대효과

- 일반적인 데이터를 넣을 수 있는 함수를 정의하였기 때문에, 형식에 맞는 데이터를 가져온다면 전국 미세먼지 데이터를 분석할 수 있을 것을 기대