데이터 분석과 머신러닝을 활용한 다음날 미세먼지 농도 예측

김영우

- 대주제와 데이터 설명

대주제 : 다음날 미세먼지 농도를 예측하는 머신러닝 모델을 만드세요.

[데이터 설명]

* 학습 데이터
    * air_2021.csv : 2021년 미세먼지 데이터
    * weather_2021.csv : 2021년 날씨 데이터

* 테스트 데이터
    * air_2022.csv : 2022년 미세먼지 데이터
    * weather_2022.csv : 2022년 날씨 데이터

- **Index**

air_2021.csv, weather_2021.csv를 이용해 전처리 작업

air_21.csv 파일은 2021년 서울시 종로구의 SO2, CO, NO2, PM10, PM25를 보여주고 있다.

측정일시는 시간순으로 배치되어 있지 않다.

```
air_21.head(5)
```

	지역	망	측정소코드	측정소명	측정일시	SO2	CO	O3	NO2	PM10	PM25	주소
0	서울 종로구	도시대기	111123	종로구	2021100101	0.003	0.6	0.002	0.039	31.0	18.0	서울 종로구 종로35가길 19
1	서울 종로구	도시대기	111123	종로구	2021100102	0.003	0.6	0.002	0.035	27.0	16.0	서울 종로구 종로35가길 19
2	서울 종로구	도시대기	111123	종로구	2021100103	0.003	0.6	0.002	0.033	28.0	18.0	서울 종로구 종로35가길 19
3	서울 종로구	도시대기	111123	종로구	2021100104	0.003	0.6	0.002	0.030	26.0	16.0	서울 종로구 종로35가길 19
4	서울 종로구	도시대기	111123	종로구	2021100105	0.003	0.5	0.003	0.026	26.0	16.0	서울 종로구 종로35가길 19

```
air_21.tail(5)
```

	지역	망	측정소코드	측정소명	측정일시	SO2	CO	O3	NO2	PM10	PM25	주소
8755	서울 종로구	도시대기	111123	종로구	2021093020	0.003	0.7	0.020	0.036	35.0	24.0	서울 종로구 종로35가길 19
8756	서울 종로구	도시대기	111123	종로구	2021093021	0.003	0.6	0.016	0.035	34.0	21.0	서울 종로구 종로35가길 19
8757	서울 종로구	도시대기	111123	종로구	2021093022	0.003	0.6	0.012	0.036	30.0	19.0	서울 종로구 종로35가길 19
8758	서울 종로구	도시대기	111123	종로구	2021093023	0.003	0.6	0.004	0.042	33.0	19.0	서울 종로구 종로35가길 19
8759	서울 종로구	도시대기	111123	종로구	2021093024	0.003	0.6	0.003	0.042	29.0	17.0	서울 종로구 종로35가길 19

weather_21.csv 파일은 2021년 서울시의 기상 현상을 보여준다.

측정일시는 시간순으로 배치되어 있다.

```
weather_21.head(5)
```

	지점	지점명	일시	기온(°C)	기온QC플래그	강수량(mm)	강수량QC플래그	풍속(m/s)	풍속QC플래그	풍향(16방위)	풍향QC플래그	습도(%)	습도QC플래그	증기압(hPa)	이슬점온도(°C)	현지기압(hPa)	현지기압QC플래그	해면기압(hPa)	해면기압QC플래그	일조(hr)	일조QC플래그	일사(MJ/m2)	일사QC플래그	적설(cm)	3시간신적설(cm)	전운량(10분위)	중하층운량(10분위)	운형(운형약어)	최저운고(100m)	시정(10m)	지면상태(지면상태코드)	현상번호(국내식)	지면온도(°C)	지면온도QC플래그	5cm지중온도(°C)	10cm지중온도(°C)	20cm지중온도(°C)	30cm지중온도(°C)
0	108	서울	2021-01-01 01:00	-8.7	NaN	NaN	NaN	2.4	NaN	270.0	NaN	68	NaN	2.2	-13.5	1016.4	NaN	1027.7	NaN	NaN	9.0	NaN	9.0	NaN	NaN	0.0	0	NaN	NaN	2000	NaN	NaN	-6.9	NaN	-1.0	-0.8	0.3	1.6
1	108	서울	2021-01-01 02:00	-9.1	NaN	NaN	NaN	1.6	NaN	270.0	NaN	69	NaN	2.1	-13.7	1016.2	NaN	1027.5	NaN	NaN	9.0	NaN	9.0	NaN	NaN	0.0	0	NaN	NaN	2000	NaN	NaN	-7.1	NaN	-1.1	-0.8	0.3	1.6
2	108	서울	2021-01-01 03:00	-9.3	NaN	NaN	NaN	1.1	NaN	250.0	NaN	70	NaN	2.1	-13.7	1016.8	NaN	1028.1	NaN	NaN	9.0	NaN	9.0	NaN	NaN	0.0	0	NaN	NaN	2000	NaN	NaN	-7.3	NaN	-1.2	-0.9	0.3	1.6
3	108	서울	2021-01-01 04:00	-9.3	NaN	NaN	NaN	0.3	NaN	0.0	NaN	71	NaN	2.2	-13.5	1016.2	NaN	1027.5	NaN	NaN	9.0	NaN	9.0	NaN	NaN	0.0	0	NaN	NaN	2000	NaN	NaN	-7.5	NaN	-1.3	-1.0	0.2	1.5
4	108	서울	2021-01-01 05:00	-9.7	NaN	NaN	NaN	1.9	NaN	20.0	NaN	72	NaN	2.1	-13.8	1015.6	NaN	1026.9	NaN	NaN	9.0	NaN	9.0	NaN	NaN	0.0	0	NaN	NaN	2000	NaN	NaN	-7.6	NaN	-1.3	-1.0	0.2	1.5

- 시간대 논점

air_21.csv 파일은 01시 ~ 24시까지 시간대 배치
weather_21.csv 파일은 00시 ~ 23시까지 시간대 배치

사실 24시 라는 개념이 없기 때문에 24시를 00시로 배치해야 한다.
따라서 air_21.csv의 시간대와 weather_21.csv 시간대를 통일하는 것이 좋다.

```
# air_21의 '측정일시'를 활용하여 'time' 변수 생성
from datetime import timedelta
air_21['time'] = air_21['측정일시']
air_21['time'] = air_21['time'] - 1 # Hour 24때문에
air_21['time'] = pd.to_datetime(air_21['time'].astype(str), format='%Y%m%d%H')
air_21['time'] = air_21['time'] + timedelta(hours=1)
```

```
air_21.head(5)
```

	지역	망	측정소코드	측정소명	측정일시	SO2	CO	O3	NO2	PM10	PM25	주소	time
0	서울 종로구	도시대기	111123	종로구	2021100101	0.003	0.6	0.002	0.039	31.0	18.0	서울 종로구 종로35가길 19	2021-10-01 01:00:00
1	서울 종로구	도시대기	111123	종로구	2021100102	0.003	0.6	0.002	0.035	27.0	16.0	서울 종로구 종로35가길 19	2021-10-01 02:00:00
2	서울 종로구	도시대기	111123	종로구	2021100103	0.003	0.6	0.002	0.033	28.0	18.0	서울 종로구 종로35가길 19	2021-10-01 03:00:00
3	서울 종로구	도시대기	111123	종로구	2021100104	0.003	0.6	0.002	0.030	26.0	16.0	서울 종로구 종로35가길 19	2021-10-01 04:00:00
4	서울 종로구	도시대기	111123	종로구	2021100105	0.003	0.5	0.003	0.026	26.0	16.0	서울 종로구 종로35가길 19	2021-10-01 05:00:00

```
# weather_21 '일시'를 활용하여 'time' 변수 생성
weather_21['time'] = pd.to_datetime(weather_21['일시'])
```

Tablour Data 통합

‘time’ 열 기준으로 통합

[2-3] 'time' 기준으로 데이터 합치기

- 미세먼지 데이터와 날씨 데이터를 'time' 기준으로 합쳐보세요.
  - df_21에는 'time' 기준으로 21년도 미세먼지, 날씨 데이터를 합쳐보세요.
  - df_22에는 'time' 기준으로 22년도 미세먼지, 날씨 데이터를 합쳐보세요.

```
# 아래에 필요한 코드를 작성하고 결과를 확인합니다.
df_21 = pd.merge(air_21, weather_21, how='inner', on='time')
df_22 = pd.merge(air_22, weather_22, how='inner', on='time')
```

시간 순으로 데이터 정렬

시계열 데이터로 만들어야 결측치 처리할 때 유용하게 할 수 있기 때문이다.

```
# df_21, df_22 'time'을 인덱스로 정의하고, sort_index를 통해 시간순 정렬

# 'time' 열을 인덱스로 설정
df_21.set_index('time', inplace=True)
df_22.set_index('time', inplace=True)

# 인덱스를 기준으로 정렬
df_21.sort_index(inplace=True)
df_22.sort_index(inplace=True)
```

```
df_21.head(1)
```

time	지역	망	측정소코드	측정소명	측정일시	SO2	CO	O3	NO2	PM10	PM25	주소	지점	지점명	일시	기온(°C)	기온QC플래그	강수량(mm)	강수량QC플래그	풍속(m/s)	풍속QC플래그	풍향(16방위)	풍향QC플래그	습도(%)	습도QC플래그	증기압(hPa)	이슬점온도(°C)	현지기압(hPa)	현지기압QC플래그	해면기압(hPa)	해면기압QC플래그	일조(hr)	일조QC플래그	일사(MJ/m2)	일사QC플래그	적설(cm)	3시간신적설(cm)	전운량(10분위)	중하층운량(10분위)	운형(운형약어)	최저운고(100m)	시정(10m)	지면상태(지면상태코드)	현상번호(국내식)	지면온도(°C)	지면온도QC플래그	5cm지중온도(°C)	10cm지중온도(°C)	20cm지중온도(°C)	30
2021-01-01 01:00:00	서울종로구	도시대기	111123	종로구	2021010101	0.002	0.5	0.022	0.016	24.0	14.0	서울종로구종로35가길19	108	서울	2021-01-01 01:00	-8.7	NaN	NaN	NaN	2.4	NaN	270.0	NaN	68	NaN	2.2	-13.5	1016.4	NaN	1027.7	NaN	NaN	9.0	NaN	9.0	NaN	NaN	0.0	0	NaN	NaN	2000	NaN	NaN	-6.9	NaN	-1.0	-0.8	0.3	

결측치 처리하는 것이 주요한 임무이다.
어떻게 처리하냐에 따라 모델의 성능에 큰 영향을 주기 때문이다.

[485]:
```
# df_21 결측치 확인
df_21.isna().sum()
```

[485]:
```
지역                    0
망                     0
측정소코드                 0
측정소명                  0
측정일시                  0
SO2                 112
CO                   80
O3                   97
NO2                  80
PM10                105
PM25                 97
주소                    0
지점                    0
지점명                   0
일시                    0
기온(°C)              0
기온 QC플래그           8759
강수량(mm)            7810
강수량 QC플래그          6996
풍속(m/s)              2
풍속 QC플래그           8757
풍향(16방위)             2
풍향 QC플래그           8757
습도(%)                0
습도 QC플래그           8759
증기압(hPa)             0
이슬점온도(°C)            0
현지기압(hPa)           0
현지기압 QC플래그         8759
해면기압(hPa)           0
해면기압 QC플래그         8759
일조(hr)             3968
일조 QC플래그           4791
일사(MJ/m2)          3968
일사 QC플래그           4791
적설(cm)             8380
3시간신적설(cm)          8730
전운량(10분위)            2
중하층운량(10분위)           0
운형(운형약어)           3812
최저운고(100m )        4212
시정(10m)              0
지면상태(지면상태코드)       8759
현상번호(국내식)          6634
지면온도(°C)            5
지면온도 QC플래그         8735
5cm 지중온도(°C)         5
10cm 지중온도(°C)        5
20cm 지중온도(°C)        5
30cm 지중온도(°C)        5
dtype: int64
```

SO2, CO, O3, NO2에 대한 결측치 처리

Ex) 0시에 해당하는 SO2에 대한 최빈값을 찾고, 결측치는 최빈값으로 대체

시간대에 따른 SO2, CO, O3, NO2가
같은 시간대에도 값이 비슷할 것이라는 생각에 위 설명처럼 결측치 처리하였다.

[487]:
```
# 'SO2'(아황산가스)의 결측치는 시(Hour)의 SO2의 최빈값으로 대체한다.
times = [time for time in range(0, 24, 1)]
for time in times:
    # ex) 0시에 해당하는 SO2의 최빈값을 찾는다.
    mode = df_21.loc[df_21.index.hour == time, 'SO2'].mode().iloc[0]
    df_21.loc[(df_21.index.hour == time) & (df_21['SO2'].isna()), 'SO2'] = mode
```

[488]:
```
# 'CO'(일산화탄소)의 결측치는 시(Hour)의 CO의 최빈값으로 대체한다.
for time in times:
    # ex) 0시에 해당하는 SO2의 최빈값을 찾는다.
    mode = df_21.loc[df_21.index.hour == time, 'CO'].mode().iloc[0]
    df_21.loc[(df_21.index.hour == time) & (df_21['CO'].isna()), 'CO'] = mode
```

[489]:
```
# 'O3'(오존)의 결측치는 시(Hour)의 O3의 최빈값으로 대체한다.
for time in times:
    # ex) 0시에 해당하는 O3의 최빈값을 찾는다.
    mode = df_21.loc[df_21.index.hour == time, 'O3'].mode().iloc[0]
    print(mode)
    df_21.loc[(df_21.index.hour == time) & (df_21['O3'].isna()), 'O3'] = mode
```

•••

[490]:
```
# 'NO2'(이산화질소)의 결측치는 시(Hour)의 NO2의 최빈값으로 대체한다.
for time in times:
    # ex) 0시에 해당하는 NO2의 최빈값을 찾는다.
    mode = df_21.loc[df_21.index.hour == time, 'NO2'].mode().iloc[0]
    print(mode)
    df_21.loc[(df_21.index.hour == time) & (df_21['NO2'].isna()), 'NO2'] = mode
```

PM10, PM25에 대한 결측치 처리

Ex) 17시에 PM10이나 PM25의 값이 NaN이면 주변 시간대(ex. 16시, 18시)의 PM10, PM25의 값을 반영하여 결측치를 처리한다. (선형 보간법)

PM10이나 PM25는 주변 시간대의 영향을 많이 받을 것 같아서 위 설명처럼 결측치 처리하였다.

[491]:
```
# 'PM10'의 결측치는 선형보간법을 이용해 주변 시간대의 'PM10'값으로 대체한다.
df_21['PM10'] = df_21['PM10'].interpolate(method='linear')
```

[492]:
```
# 'PM25'의 결측치는 선형보간법을 이용해 주변 시간대의 'PM10'값으로 대체한다.
df_21['PM25'] = df_21['PM25'].interpolate(method='linear')
```

강수량(mm)에 대한 결측치 처리

Ex) 강수량(mm)이 NaN이면 0으로 처리

해당 날짜의 강수량(mm)이 NaN일 때
해당 날짜의 강수량 정보를 인터넷에서 검색하여 실제 비가 왔는지 오지 않았는지 확인했고
얼추 들어맞기 때문에 NaN은 0으로 처리하였다.

[495]:
```
# Chat GPT에 따르면 강수량(mm)이 NaN인 경우 때때로 '0mm'인 것을 의미한다고 하기도 해서
# 강수량(mm)이 NaN이면 0.0으로 대체한다.
df_21.loc[df_21['강수량(mm)'].isna(), '강수량(mm)'] = 0.0
```

풍속(m/s), 풍향(16방위)에 대한 결측치 처리

Ex) 주변 시간대의 풍속과 풍향을 반영하여 결측치 처리

풍속이나 풍향은 주변 시간대의 영향을 많이 받을 것 같아서 위 설명처럼 결측치 처리하였다.

```
# 풍속(m/s)은 선형 보간법을 이용한다. 주변의 풍속(m/s)으로 대체한다.
df_21['풍속(m/s)'] = df_21['풍속(m/s)'].interpolate()
```

```
# 풍향(16방위)은 선형 보간법을 이용한다. 주변의 풍향(16방위)으로 대체한다.
df_21['풍향(16방위)'] = df_21['풍향(16방위)'].interpolate()
```

일조(hr)에 대한 결측치 처리

일조 : 태양빛이 얼마나 오랫동안 땅에 비추었는지의 시간

밤이나 새벽에는 일조(hr)가 없을 것이라고 판단하여 0,
오전이나 오후에는 일조(hr)가 있을 것으로 생각해 1로 결측치 처리

505]:
```
# '보통 밤, 새벽에는 일조(hr)이 없고, 오전, 오후에 일조(hr)가 많으므로 그에 대한 처리를 하였다.
times = [time for time in range(0, 24, 1)]
for time in times:
    if time in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 21, 22, 23]:
        df_21.loc[(df_21.index.hour == time) & df_21['일조(hr)'].isna(), '일조(hr)'] = 0.0
    else:
        df_21.loc[(df_21.index.hour == time) & df_21['일조(hr)'].isna(), '일조(hr)'] = 1.0
```

일사(MJ/m2)에 대한 결측치 처리

일사 : 태양빛의 에너지 양

**시간대에 따른 일사(MJ/m2)의 분포도를 확인하여 최빈값이 없으면,
시간대는 밤이나 새벽으로 판단하여 0으로 처리**

**최빈값이 1개나 여러 개이면 시간대는 오전, 오후로 판단했고,
최빈값이 1개이면 그래도 결측치 처리, 여러 개면 평균을 구해서 결측치 처리**

[507]:
```
# '보통 밤, 새벽에는 일사(MJ/m2)이 없고, 오전, 오후에 일사(MJ/m2)가 많으므로 그에 대한 처리를 하였다.
# 기본적으로 시(hour)에 대한 최빈값으로 결측치를 대체했으며, 최빈값이 여러개인 경우 최빈값의 평균을 구하여 결측치를 대체했다.
times = [time for time in range(0, 24, 1)]
for time in times:
    mode = df_21.loc[(df_21.index.hour == time), '일사(MJ/m2)'].mode()
    if len(mode) == 0:
        df_21.loc[(df_21.index.hour == time ) & (df_21['일사(MJ/m2)'].isna()), '일사(MJ/m2)'] = 0.0
    elif len(mode) == 1:
        df_21.loc[(df_21.index.hour == time ) & (df_21['일사(MJ/m2)'].isna()), '일사(MJ/m2)'] = mode.iloc[0]
    else:
        df_21.loc[(df_21.index.hour == time ) & (df_21['일사(MJ/m2)'].isna()), '일사(MJ/m2)'] = mode.mean()
```

적설(cm)에 대한 결측치 처리

해당 날짜에 대한 적설량을 인터넷을 통해 검색하여 NaN이면 적설량이 일반적으로 0이라는 것을 확인
따라서 NaN이면 0으로 결측치를 처리하였다.

[509]:
```
# 일적설량(cm)이 NaN이면 0.0으로 처리한다.
df_21.loc[df_21['적설(cm)'].isna(), '적설(cm)'] = 0.0
```

결측치 처리 완료

나머지 결측치 처리 방법 설명 생략...

```
[522]: df_21.isna().sum()
```

```
[522]: 지역              0
       망               0
       측정소코드           0
       측정소명            0
       측정일시            0
       SO2             0
       CO              0
       O3              0
       NO2             0
       PM10            0
       PM25            0
       주소              0
       지점              0
       지점명             0
       일시              0
       기온(°C)          0
       강수량(mm)          0
       풍속(m/s)         0
       풍향(16방위)         0
       습도(%)           0
       증기압(hPa)         0
       이슬점온도(°C)        0
       현지기압(hPa)        0
       해면기압(hPa)        0
       일조(hr)          0
       일사(MJ/m2)        0
       적설(cm)          0
       전운량(10분위)         0
       중하층운량(10분위)      0
       시정(10m)         0
       지면온도(°C)         0
       5cm 지중온도(°C)     0
       10cm 지중온도(°C)    0
       20cm 지중온도(°C)    0
       30cm 지중온도(°C)    0
       dtype: int64
```

- **Feature Engineering**

전일 같은 시간 미세먼지 농도 변수 추가
Ex) 2021년 5월 15일 08시면, 전날인 2021년 5월 14일 08시의 미세먼지 농도를 확인하여 새로운 변수로 추가한다.

[2-6] 전일 같은 시간 미세먼지 농도 변수 추가

- 먼저 df_21, df_22에 month, day, hour 변수를 추가하세요.
  - 예) dt.month, dt.day, dt.hour 사용 또는 datetimeindex에서는 df.index.month 등 사용 가능
- 모델링에 유용한 변수로 전일 같은 시간(24시간 전) 미세먼지 농도 변수를 추가하세요.
  - 시계열 데이터 처리를 위한 shift 연산을 참고하세요.

[535]:
```
# df_21, df_22의 index(time)를 month, day, hour 로 쪼개기 (year는 필요 없음). 이후에 저장 시 index(time)은 포함하지 않음.
df_21['month'] = df_21.index.month
df_21['hour'] = df_21.index.hour
df_21['yesterday_PM10'] = df_21['PM10'].shift(24)
```

- **Feature Engineering**

1시간 이후의 미세먼지 농도 변수 추가
이 변수는 Y(target)으로 우리가 예측할 값이다.

[2-7] t+1 시점의 미세먼지 농도 데이터 생성

- t+1 시점은 1시간 후 입니다.
- t+1 시점의 미세먼지 농도 변수(PM10_1)를 생성하세요.
- t+1 시점의 미세먼지 농도는 머신러닝 모델을 통해 예측하려는 y값(target) 입니다.

[540]:
```
# df_21, df_22에 t+1 시점 변수(PM10_1) 추가
df_21['PM10_1'] = df_21['PM10'].shift(-1)
```

- **Feature Engineering**

최신 7시간대 미세먼지 농도 평균을 구한 변수 추가

이전 7시간 동안 미세먼지 농도를 확인하여 1시간 후의 미세먼지 농도를 파악할 수 있을 것으로 생각했다.

[544]:
```
# 최신 7일 미세먼지(PM10) 평균 구하는 새로운 열 변수 추가
df_21['7day_Avg_PM10'] = df_21['PM10'].rolling(window=7, min_periods=1).mean()
```

- **Feature Engineering**

해당 날짜의 시간대에 강수량 정도를 확인하는 변수 추가

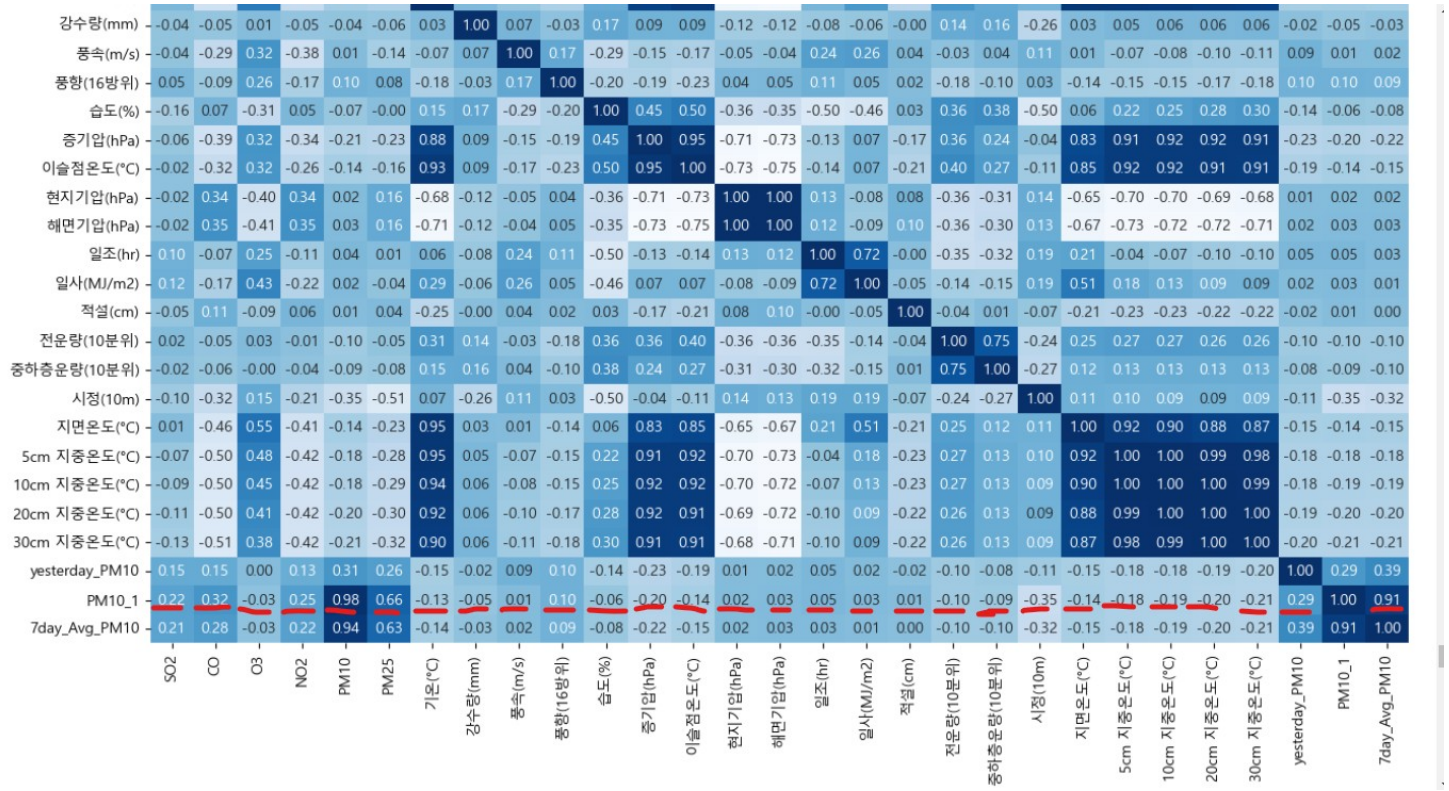
아무래도 강수량과 미세먼지가 연관관계가 있다고 생각해서
강수량 정도를 확인하면 1시간 후의 미세먼지 농도를 예측할 수 있지 않을까 생각했다.

[555]:
```python
# 강수량 정도를 나타내는 열 변수 추가
def rain(mm):
    if 0.1 <= mm <= 7.5:
        return '가벼운 비'
    elif 7.5 <= mm <= 15:
        return '보통의 비'
    elif mm >= 15:
        return '심한 비'
    else:
        return '비가 오지않음'
df_21['Rain_Value'] = df_21['강수량(mm)'].apply(rain)
```

- 이변량 분석에 앞서서 변수 간에 상관계수 확인

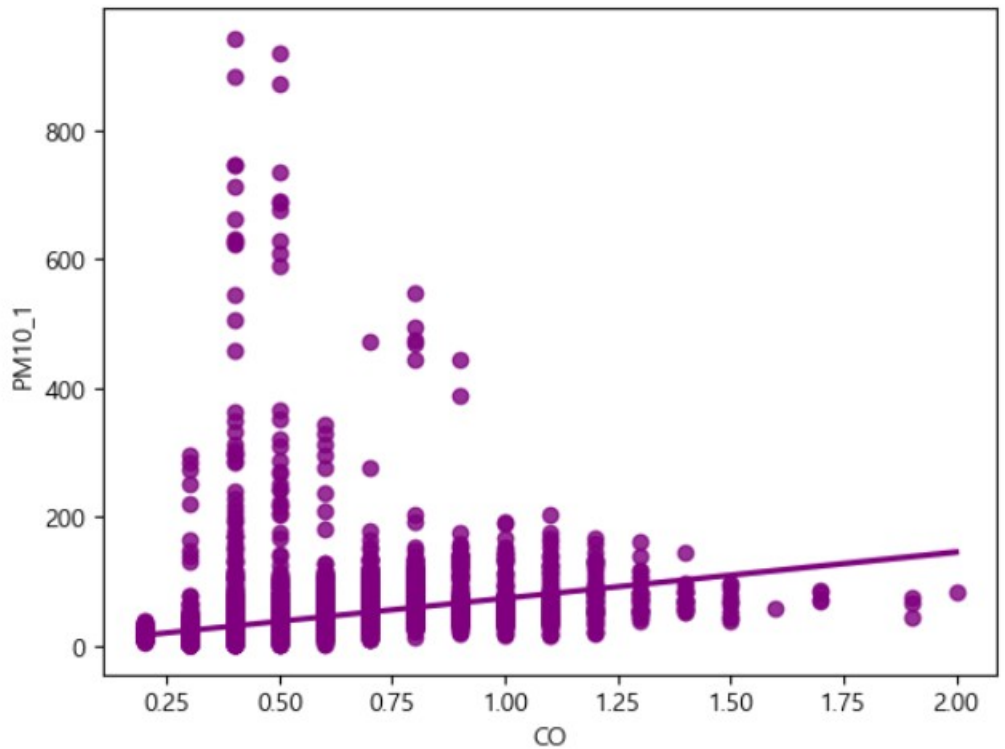
숫자형(x축) – 숫자형(y축) 간 상관계수(피어슨)를 전체적으로 확인한다.
상관계수가 높다고 판단되는 변수는 유지하고, 그렇지 않다고 판단되는 변수는 과감하게 삭제하는 방향으로 결정

범주형(x축) – 숫자형(y축)은 지원하지 않기 때문에 따로 평가지표를 봐야 한다!



- 이변량 분석 (CO, PM10_1)

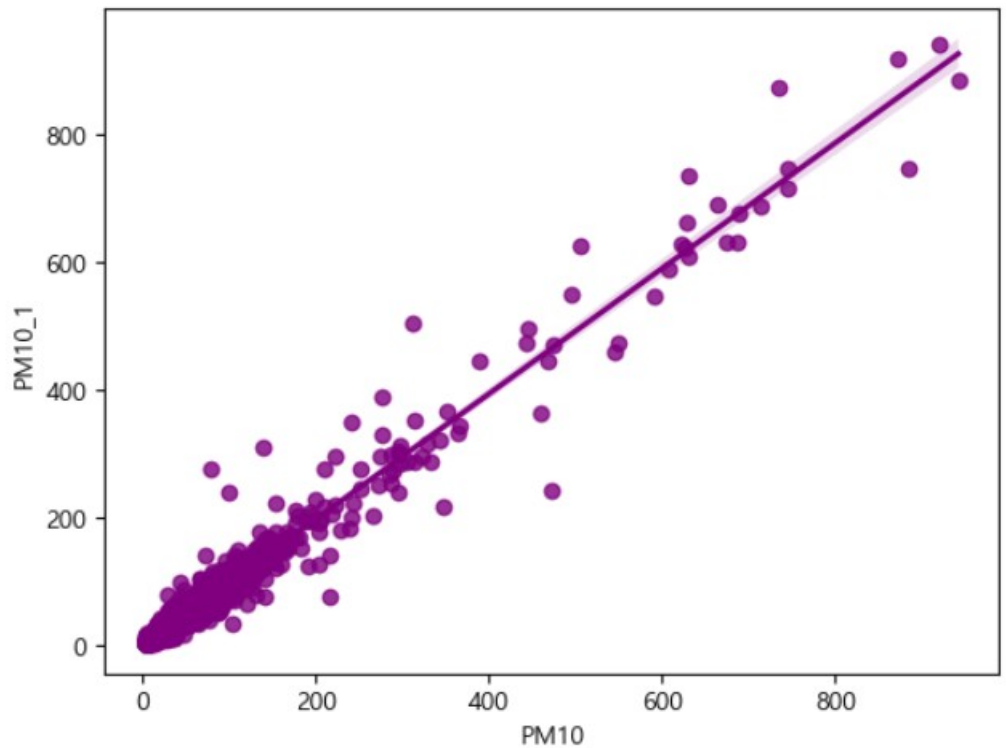
[567]:
```
# CO와 PM10_1 분석
sns.regplot(x=df_21['CO'], y=df_21['PM10_1'], color='purple')
plt.show()
stats.pearsonr(x=df_21['CO'], y=df_21['PM10_1'])
```



[567]: PearsonRResult(statistic=0.3161120680040188, pvalue=1.721318713572338e-202)

- 이변량 분석 (PM10, PM10_1)

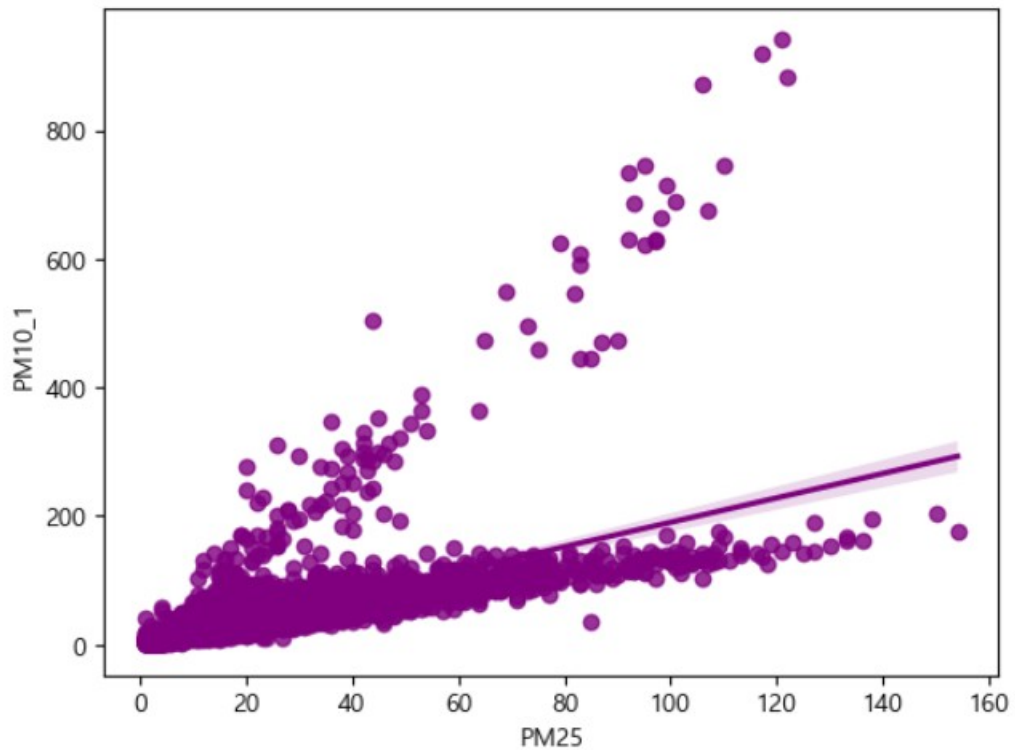
[568]:
```
# PM10와 PM10_1 분석
sns.regplot(x=df_21['PM10'], y=df_21['PM10_1'], color='purple')
plt.show()
stats.pearsonr(x=df_21['PM10'], y=df_21['PM10_1'])
```



[568]: PearsonRResult(statistic=0.9820851725143437, pvalue=0.0)

- 이변량 분석 (PM25, PM10_1)

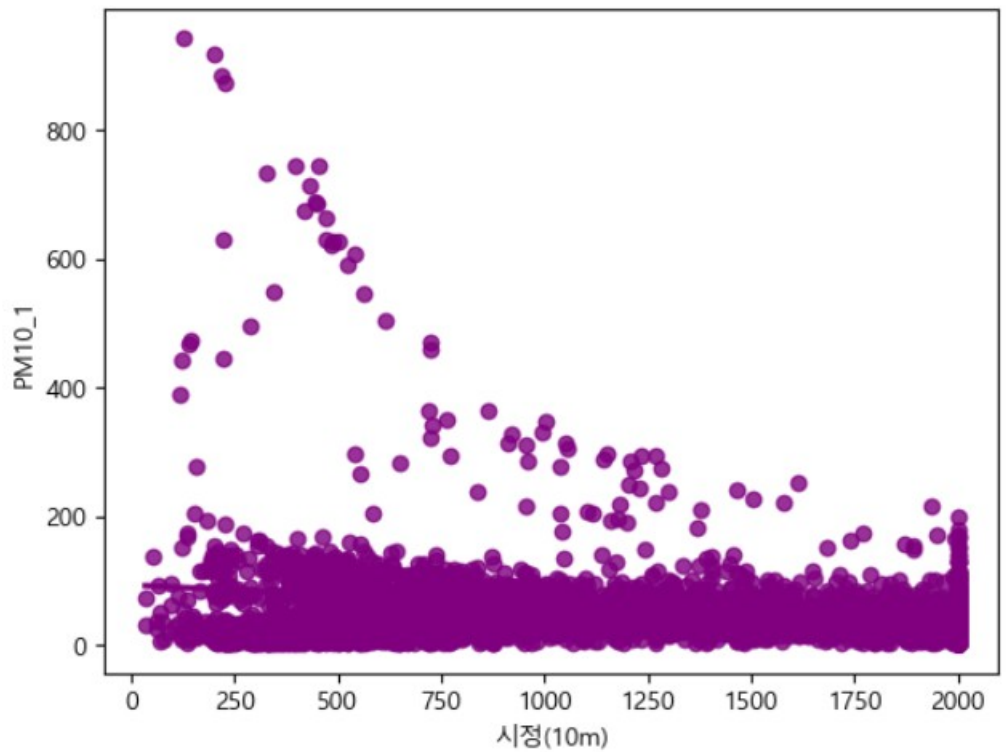
[569]:
```
# PM25와 PM10_1 분석
sns.regplot(x=df_21['PM25'], y=df_21['PM10_1'], color='purple')
plt.show()
stats.pearsonr(x=df_21['PM25'], y=df_21['PM10_1'])
```



[569]: PearsonRResult(statistic=0.6640497523401349, pvalue=0.0)

- **이변량 분석 (시정 (10m), PM10_1)**

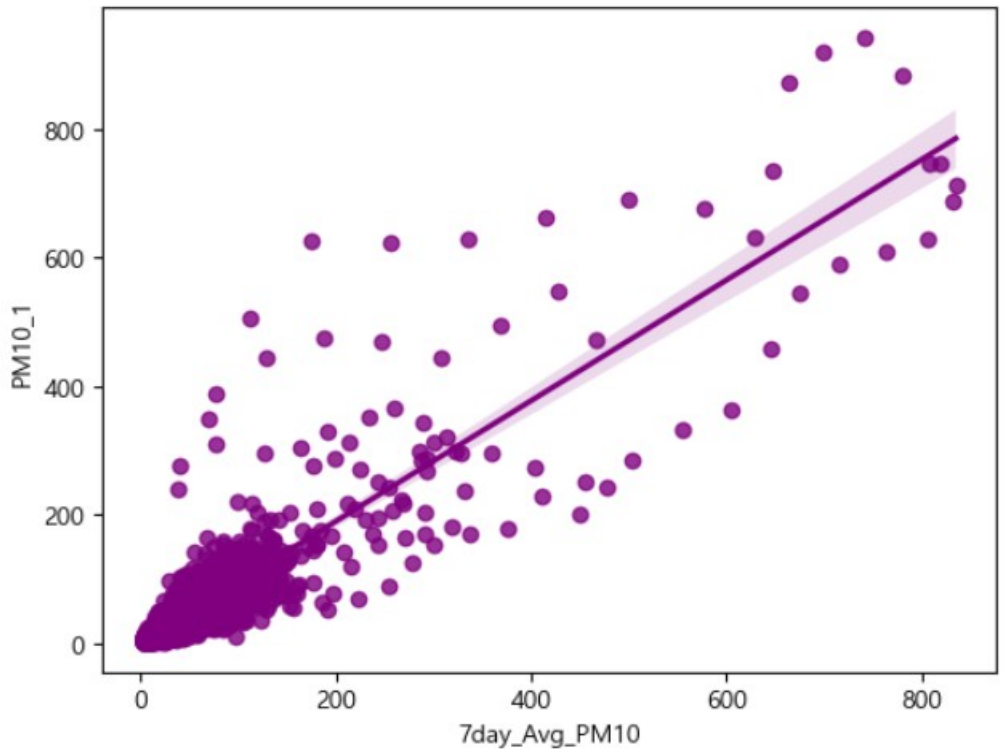
[570]:
```
# 시정(10m)와 PM10_1 분석
sns.regplot(x=df_21['시정(10m)'], y=df_21['PM10_1'], color='purple')
plt.show()
stats.pearsonr(x=df_21['시정(10m)'], y=df_21['PM10_1'])
```



[570]:
```
PearsonRResult(statistic=-0.3547547502525966, pvalue=4.098104739020815e-258)
```

- 이변량 분석 (7day_Avg_PM10(최신 7시간 미세먼지 농도 평균) , PM10_1)

[571]:
```
# 7day_Avg_PM10와 PM10_1 분석
sns.regplot(x=df_21['7day_Avg_PM10'], y=df_21['PM10_1'], color='purple')
plt.show()
stats.pearsonr(x=df_21['7day_Avg_PM10'], y=df_21['PM10_1'])
```



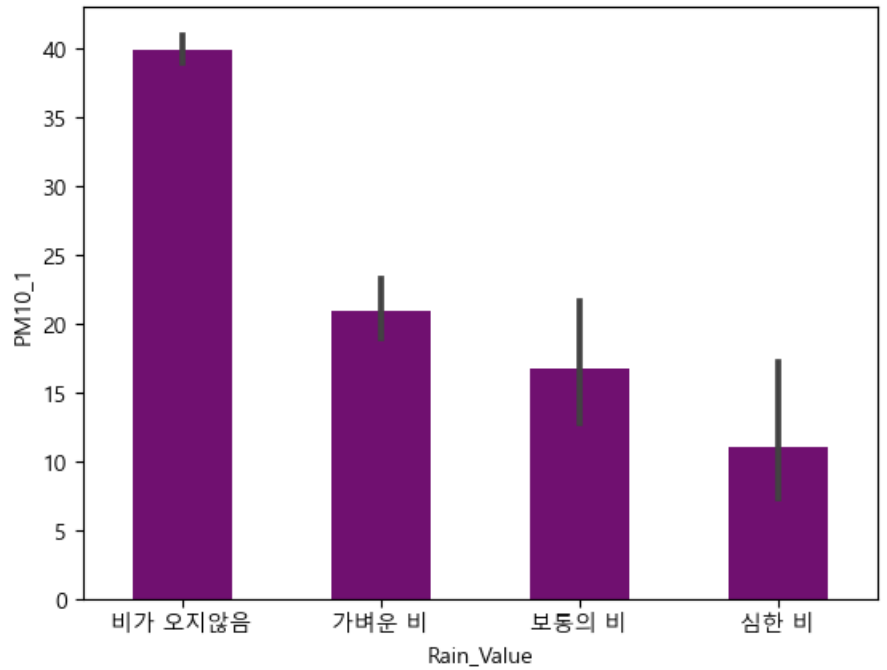
[571]:
```
PearsonRResult(statistic=0.9067226229735692, pvalue=0.0)
```

- 이변량 분석 (Rain_Value , PM10_1)

[573]:
```
# Rain_Value와 PM10_1 분석
sns.barplot(x=df_21['Rain_Value'], y=df_21['PM10_1'], width=0.5, color='purple')
plt.show()

r1 = df_21.loc[df_21['Rain_Value'] == '비가 오지않음', 'PM10_1']
r2 = df_21.loc[df_21['Rain_Value'] == '가벼운 비', 'PM10_1']
r3 = df_21.loc[df_21['Rain_Value'] ==  '보통의 비', 'PM10_1']
r4 = df_21.loc[df_21['Rain_Value'] == '심한 비', 'PM10_1']

display(stats.f_oneway(r1, r2, r3, r4))
```




```
F_onewayResult(statistic=27.445474485481633, pvalue=1.1677309377153117e-17)
```

- 최종 전처리 1 – 필요없는 변수 제거, 가변수화 or 원핫인코딩

최종적으로 변수는
X(features) : CO, PM10, 시정(10m), 7day_Avg_PM10(최신 7시간 미세먼지 농도 평균), Rain_Value
Y(Target) : PM10_1

모델링을 하려면 값은 숫자형태여야 하므로, 범주형 변수는 가변수화나 원핫인코딩을 해야 한다.

그래서 대략적인 Tablour Data는 아래와 같다.

	CO	PM10	PM25	시정(10m)	7day_Avg_PM10	PM10_1	Rain_Value_가벼운 비	Rain_Value_보통의 비	Rain_Value_비가 오지않음	Rain_Value_심한 비
0	0.5	24.0	14.0	2000	24.0	25.0	0	0	1	0

- 최종 전처리2 – x축 값 스케일링

X(features) 값의 범위를 0 ~ 1로 통일하는 것이 좋다.
그러면 모델 학습, 평가할 때 보다 제대로 결과를 도출할 수 있고, 데이터를 처리하기 쉬워진다.

스케일링 방법 중 하나인 MinMaxScaler를 통해 작업
Tablour Data는 그냥 아래 그림과 같이 작업을 하면 된다.

다만 이미지 정보 같은 경우, 아래와 같이 MinMaxScaler 라이브러리를 그대로 사용하면 안된다.
왜냐하면 저 라이브러리는 2차원 표 데이터에 한해서 처리가 가능하기 때문에, 이 부분을 잘 기억해야 한다.

[397]:
```
# 스케일링 작업
# x(features) 값의 범위를 0~ 1로 통일한다.
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df_21_X)
df_21_X = scaler.transform(df_21_X)
df_22_X = scaler.transform(df_22_X)

df_21_X = pd.DataFrame(df_21_X, columns = ['CO', 'PM10', 'PM25', '시정(10m)', '7day_Avg_PM10', 'Rain_Value_가벼운 비', 'Rain_Value_보통의 비', '
df_22_X = pd.DataFrame(df_22_X, columns = ['CO', 'PM10', 'PM25', '시정(10m)', '7day_Avg_PM10', 'Rain_Value_가벼운 비', 'Rain_Value_보통의 비', '
```

[398]:
```
df_21_X.head()
```

[398]:

	CO	PM10	PM25	시정(10m)	7day_Avg_PM10	Rain_Value_가벼운 비	Rain_Value_보통의 비	Rain_Value_비가 오지않음	Rain_Value_심한 비
0	0.166667	0.022364	0.084967	1.0	0.025249	0.0	0.0	1.0	0.0
1	0.222222	0.023429	0.084967	1.0	0.025850	0.0	0.0	1.0	0.0
2	0.222222	0.025559	0.098039	1.0	0.026852	0.0	0.0	1.0	0.0
3	0.222222	0.021299	0.078431	1.0	0.026151	0.0	0.0	1.0	0.0
4	0.222222	0.022364	0.084967	1.0	0.025970	0.0	0.0	1.0	0.0

통합한 Tablour 데이터를 이용해 머신러닝 모델 작업

- 모델링 절차

1 : 회귀 모델인 것을 염두

2 : LinearRegression, KNN, Decision Tree, Random Forest, XGBoost, LightGBM 모델을 이용해 MSE, R2 Score를 확인한다.

3. 한 모델을 이용해 파라미터 튜닝을 하여 MSE, R2 Score를 확인한다.

- **Linear Regression**

맨 앞10개 예측값과 실제값
[23.3, 20.4, 20.8, 19.7, 25.2, 20.0, 25.3, 28.4, 27.5, 22.7]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[401]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

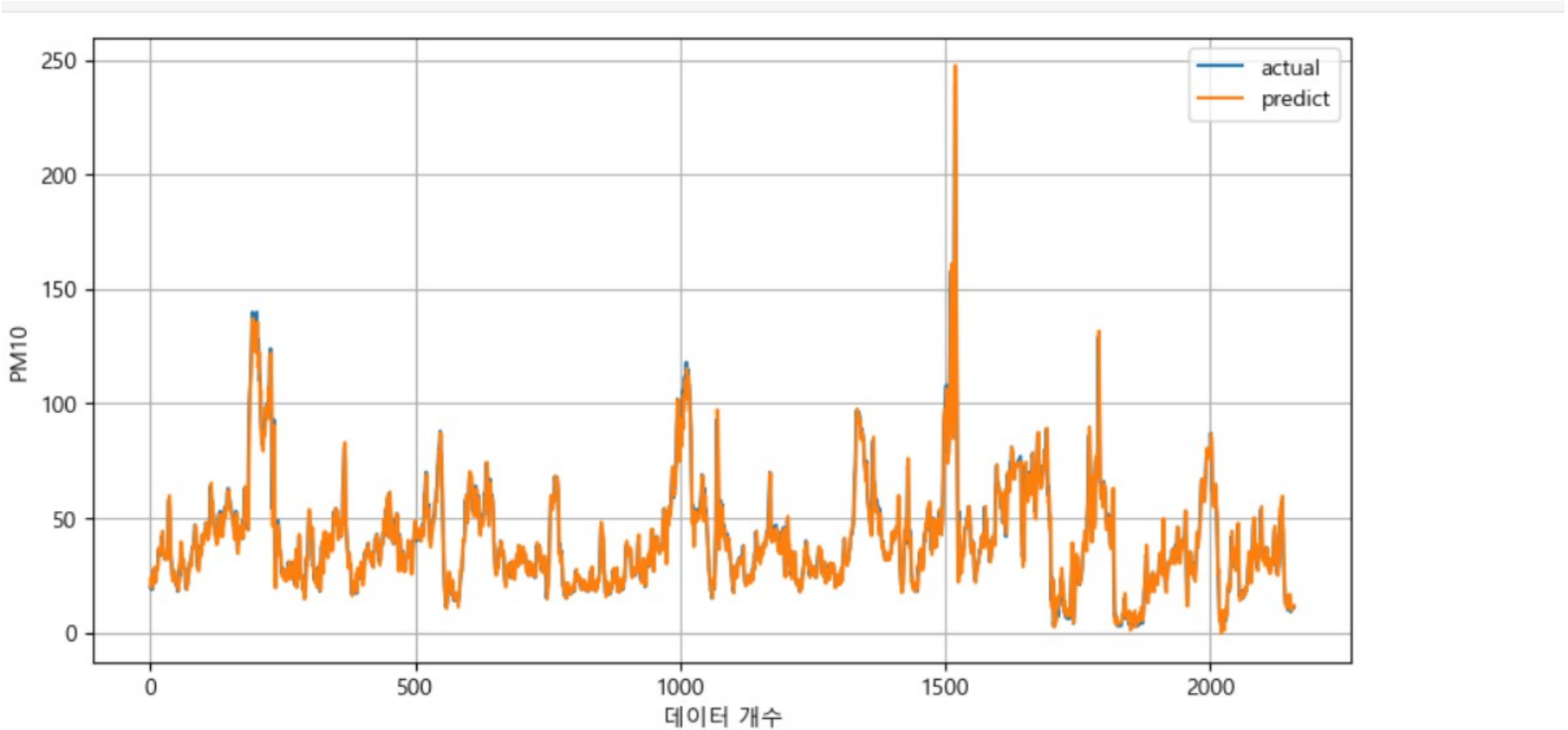
```
예측값 10개 :  [23.3049260888712, 20.405370049025993, 20.849706522019446, 19.72003038367461, 25.154100746888354, 19.95539829104444, 25.264164175828164, 28.387247619745608, 27.45800167949904, 22.711454807403975]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

[402]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  36.615024251185154
mae :  3.9034997737955197
r2 :  0.9337486810879552
```

- **Linear Regression**

예측값과 실제값 시각화



- **KNN**

맨 앞10개 예측값과 실제값
[23.5, 19.7, 19.7, 20.5, 21.5, 20.7, 27.3, 26.0, 25.1, 21.4]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[414]:
```
# KNN
KNN = KNeighborsRegressor(n_neighbors=10)

KNN.fit(df_21_X, df_21_Y)

y_pred = KNN.predict(df_22_X)
```

[415]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [23.5, 19.7, 19.7, 20.5, 21.5, 20.65, 27.3, 26.0, 25.05, 21.4]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

[416]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  81.33150105963531
rmse :  9.018397920896778
mae :  5.981443622792587
r2 :  0.8528385731132541
```

- Decision Tree

맨 앞10개 예측값과 실제값
[22.8, 20.7, 20.7, 18.9, 24.4, 18.9, 24.4, 27.7, 25.9, 21.7]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[417]:
```
# Decision Tree
DT = DTR(max_depth=8)

DT.fit(df_21_X, df_21_Y)

y_pred = DT.predict(df_22_X)
```

[418]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [22.77777777777778, 20.695575221238936, 20.695575221238936, 18.92156862745098, 24.41062801932367, 18.92156862745098, 24.4106280
1932367, 27.687637969094926, 25.883582089552238, 21.741622574955905]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

[419]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  56.40727122052505
rmse :  7.510477429599602
mae :  4.2140793582068055
r2 :  0.897936538592672
```

- **Random Forest**

맨 앞10개 예측값과 실제값
[23.8, 20.5, 21.3, 18.9, 26.4, 17.6, 26.6, 28.1, 27.7, 25.0]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[420]:
```
# RandomForest
```

[421]:
```
RF = RFR(n_estimators=50, max_depth=None)

RF.fit(df_21_X, df_21_Y)

y_pred = RF.predict(df_22_X)
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_14848\2519420311.py:3: DataConversionWarning: A column-vector y was p
ected. Please change the shape of y to (n_samples,), for example using ravel().
  RF.fit(df_21_X, df_21_Y)
```

[422]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [23.78, 20.54, 21.28, 18.94, 26.38, 17.6, 26.56, 28.14, 27.72, 25.04]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

[423]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  40.447196380687465
rmse :  6.359811033410306
mae :  4.199424076047289
r2 :  0.9268147389953375
```

- **Gradient Boosting**

맨 앞10개 예측값과 실제값
[23.0, 20.4, 20.8, 19.7, 25.3, 20.0, 25.5, 27.4, 26.4, 22.2]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

424]:
```
# Gradient Boosting
GB = GBR(learning_rate=0.1, n_estimators=101, max_depth=3)

GB.fit(df_21_X, df_21_Y)

y_pred = GB.predict(df_22_X)
```

```
C:\Users\user\anaconda3\Lib\site-packages\sklearn\ensemble\_gb.py:437: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

425]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [23.019422482603233, 20.356005077035686, 20.818022892735, 19.746998455610584, 25.250290405576422, 20.024212326069797, 25.527504276035636, 27.423013524010646, 26.435989450649075, 22.247600824617656]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

426]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  33.616353645110195
rmse :  5.797961162780 43
mae :  3.800400570192615
r2 :  0.9391744833835474
```

- **XGBoost**

맨 앞10개 예측값과 실제값
[23.3, 20.4, 21.0, 19.6, 24.9, 19.6, 25.4, 28.3, 27.0, 22.4]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[427]:
```
# XGBoost
import xgboost

XGB = xgboost.XGBRegressor()

XGB.fit(df_21_X, df_21_Y)

y_pred = XGB.predict(df_22_X)
```

[428]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [23.334785, 20.48951, 21.003105, 19.630617, 24.943495, 19.635365, 25.431904, 28.337498, 27.028109, 22.466545]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

[429]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  35.75506793943041
rmse :  5.979554158917738
mae :  3.9863039125018753
r2 :  0.9353046882469193
```

- **LightGBM**

맨 앞10개 예측값과 실제값
[23.1, 20.6, 21.0, 19.5, 25.0, 19.6, 25.2, 28.1, 26.5, 22.5]
[20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]

[430]:
```
# LightGBM
import lightgbm
lgbm = lightgbm.LGBMRegressor()

lgbm.fit(df_21_X, df_21_Y)

y_pred = lgbm.predict(df_22_X)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000938 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 884
[LightGBM] [Info] Number of data points in the train set: 8759, number of used features: 8
[LightGBM] [Info] Start training from score 38.786049
```

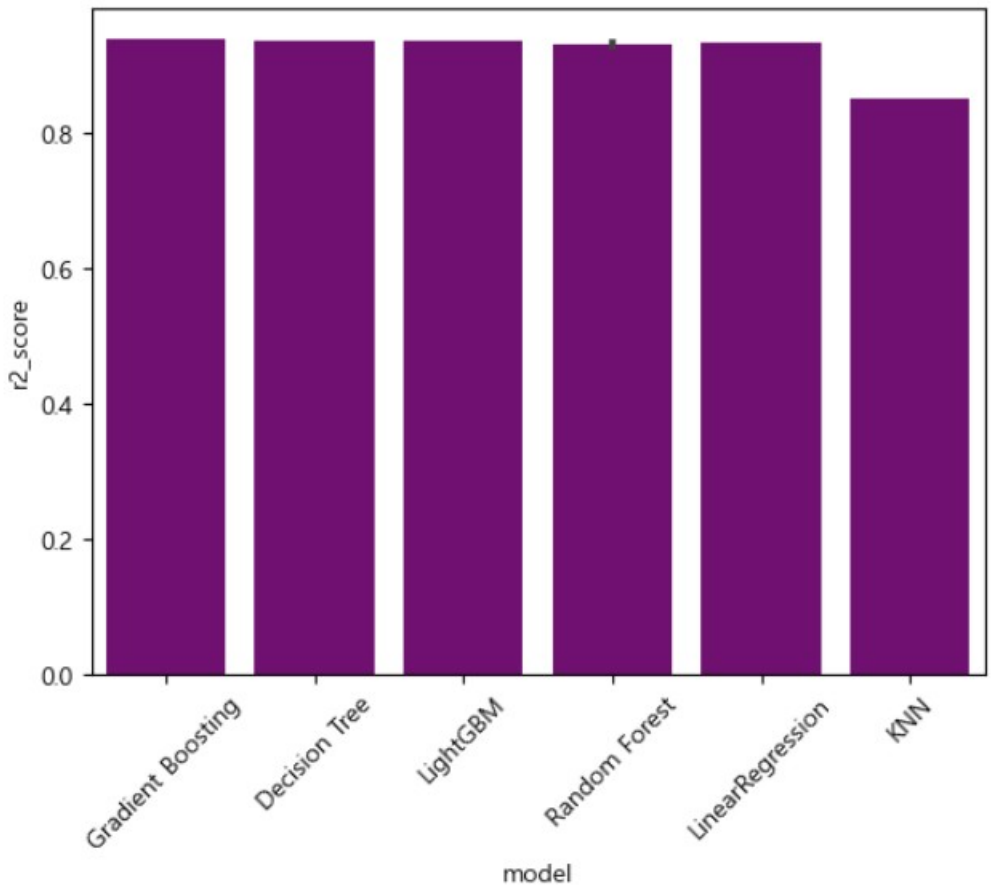
[431]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [23.13211023131903, 20.603573926791512, 21.054374213170444, 19.48381886950244, 25.0710642652207, 19.60744579024335, 25.19469118596161, 28.120141741366
695, 26.548438053429837, 22.52585730302512]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

[432]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', r2_score(df_22_Y, y_pred))
```

```
mse :  34.31363770680151
rmse :  5.857784368411107
mae :  3.8534995255957245
r2 :  0.9379128158116107
```

- 모델 R2 score 비교



- Decision Tree로 파라미터 튜닝하여 모델 성능 평가

max_depth를 1부터 99까지 돌려서(Grid Search)
가장 모델 성능이 좋은 max_depth를 확인!

(선택 수행)[4-4] 머신러닝 모델에 대해 성능 최적화 진행

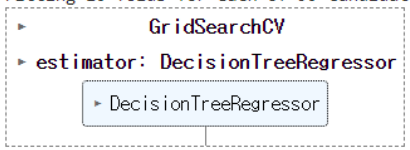
- 위 머신러닝 모델들에 대해 성능 최적화를 진행해보세요.
- Decision Tree에 대해서 파라미터 최적화를 해본다.

441]:
```
# 아래에 필요한 코드를 작성하고 결과를 확인합니다.
model = DTR()
model = GridSearchCV(model, {'max_depth' : range(1, 100, 1)}, scoring='r2', cv=10, verbose=1,)
```

442]:
```
model.fit(df_21_X, df_21_Y)
```

```
Fitting 10 folds for each of 99 candidates, totalling 990 fits
```

442]:



443]:
```
y_pred = model.predict(df_22_X)
```

444]:
```
# 예측값과 실제값 비교
predict = list(y_pred.flatten()[0:10:1])
real = list(df_22_Y['PM10_1'][0:10:1])
print('예측값 10개 : ', predict)
print('실제값 10개 : ', real)
```

```
예측값 10개 :  [23.90142021720969, 21.156593977154728, 21.156593977154728, 18.374287974683543, 23.90142021720969, 18.374287974683543, 23.90142021720969, 26.332669322
709158, 26.332669322709158, 21.156593977154728]
실제값 10개 :  [20.0, 20.0, 19.0, 24.0, 19.0, 24.0, 27.0, 26.0, 22.0, 22.0]
```

- **Decision Tree로 파라미터 튜닝하여 모델 성능 평가**

max_depth를 1부터 99까지 돌려서(Grid Search)
가장 모델 성능이 좋은 max_depth를 확인!

[445]:
```
# 최적의 파라미터와 성능 값
print('최적의 파라미터 값 : ', model.best_params_)
print('최적의 모델 성능 : ', model.best_score_) # GridSearch할 때 scoring='r2'로 줘서 model.best_score_도 r2 score이다.
```

```
최적의 파라미터 값 :  {'max_depth': 6}
최적의 모델 성능 :  0.8561050375070215
```

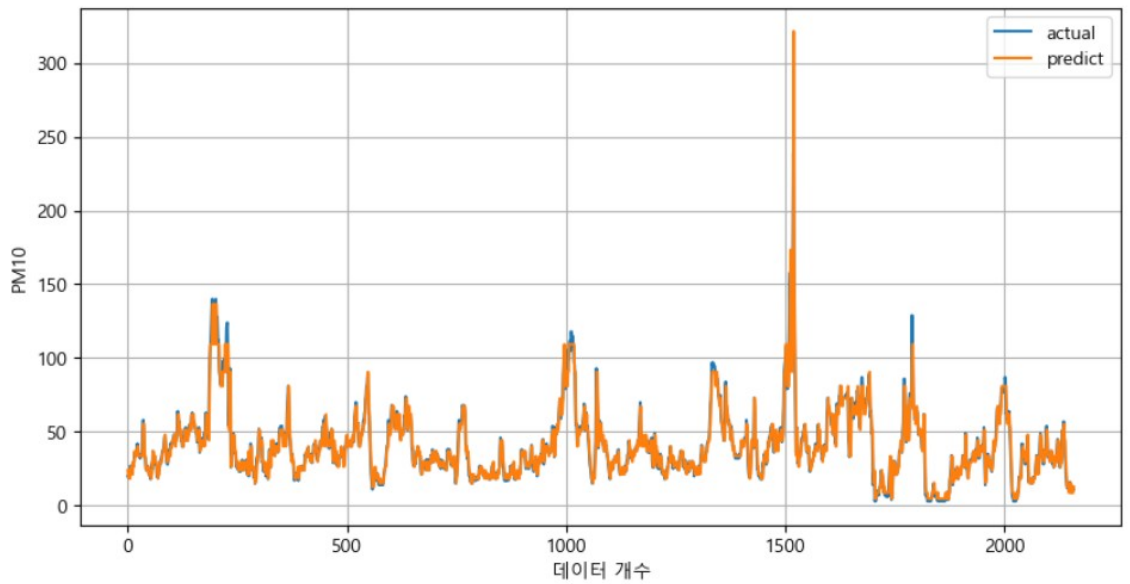
[446]:
```
# test_y 데이터와 y_pred_LR 데이터로 성능을 평가하여 출력해보세요.
# 성능지표는 mse와 r2를 이용하세요.
print('mse : ', mse(df_22_Y, y_pred))
print('rmse : ', mse(df_22_Y, y_pred) ** 0.5)
print('mae : ', mae(df_22_Y, y_pred))
print('r2 : ', model.best_score_)
```

```
mse :  47.48323598028081
rmse :  6.890808078903432
mae :  4.101369692373876
r2 :  0.8561050375070215
```

- **Decision Tree로 파라미터 튜닝하여 모델 성능 평가**

max_depth를 1부터 99까지 돌려서(Grid Search)
가장 모델 성능이 좋은 max_depth를 확인!

[447]:
```
# 예측값과 실제값 시각화
plt.figure(figsize=(10, 5))
plt.plot(df_22_Y, label='actual')
plt.plot(y_pred, label='predict')
plt.xlabel('데이터 개수')
plt.ylabel('PM10')
plt.legend()
plt.grid()
plt.show()
```



- 미니 프로젝트를 진행하면서 느낀점

1. 대학교 빅데이터 수업때 전체 중에 전처리 하는 작업이 70퍼라고 했는데 그 말이 맞다는 생각을 한다. 전처리를 어떻게 하느냐에 따라 모델의 성능이 좌우되기 때문에 전처리 작업이 정말 중요하다고 생각한다. (ex. 결측치 작업, Feature Enginering)

2 : Feature Enginering을 하는 것이 중요한데 주제에 대한 도메인 지식을 그래도 아는 것이 Feature Enginering을 하는데 도움이 될 것 같다는 생각을 한다.

3. 데이터 분석 ~ 머신러닝 모델로 이어지는 파이프라인을 이해할 수 있었다.