Data Science

Term Project

Team 5

인공지능학과 202235013 김나현

인공지능학과 202035508 김송언

인공지능학과202235044 박석민

인공지능학과 202235152 홍지민

인공지능학과 202135838 조선민

1. End-to-End Process

1-1) Business Objective

In which season should I advertise to whom It be effective?

1-2) Data Exploration

A) Information on public bicycle use in Seoul (by time zone) - 2022 (http://data.seoul.go.kr/dataList/OA-15245/F/1/datasetView.do)

B) Seoul Metropolitan City_Hourly (second) fine dust – 2022 (https://www.data.go.kr/data/15089266/fileData.do)

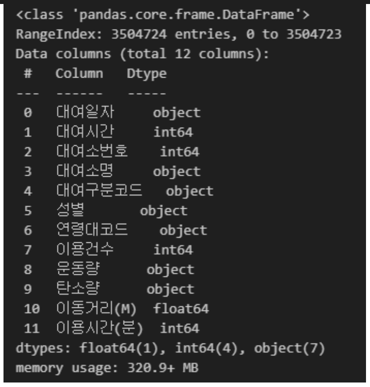
C) Weather Data - Seoul Metropolitan Government in 2022 (https://data.kma.go.kr/data/grnd/selectAsosRltmList.do?pgmNo=36)

1-3) Data Preprocessing

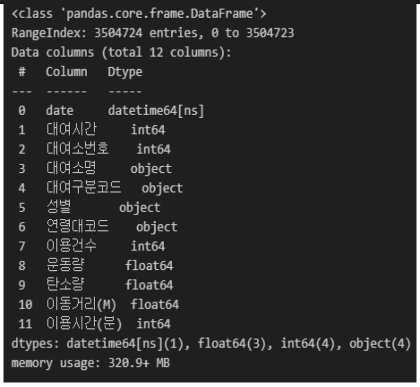
□ **Information on public bicycle use in Seoul (by time zone) - 2022**

A) Convert sequence date to data datatime of object type

```
bicycle04["대여일자"] = pd.to_datetime(bicycle04["대여일자"],
format='%Y-%m-%d')
_bicycle04 = bicycle04.rename(columns={"대여일자":"date","
이용건수":"이용건수"}) # Change column name
```



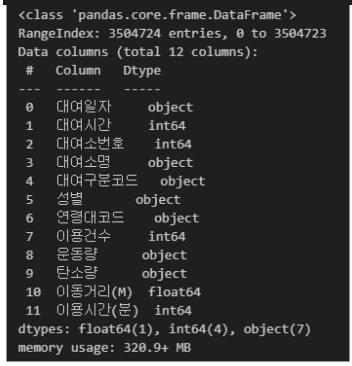
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3504724 entries, 0 to 3504723
Data columns (total 12 columns):
 #   Column    Dtype  
---  ------    -----  
 0   대여일자     object 
 1   대여시간     int64  
 2   대여소번호    int64  
 3   대여소명     object 
 4   대여구분코드   object 
 5   성별       object 
 6   연령대코드    object 
 7   이용건수     int64  
 8   운동량      object 
 9   탄소량      object 
 10  이동거리(M)  float64
 11  이용시간(분)  int64  
dtypes: float64(1), int64(4), object(7)
memory usage: 320.9+ MB
```



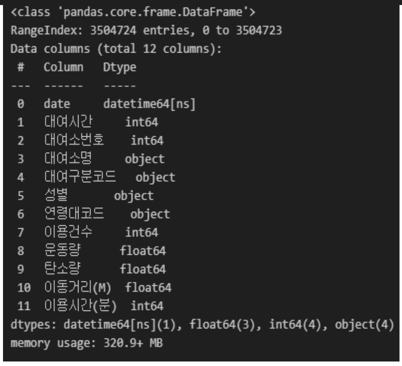
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3504724 entries, 0 to 3504723
Data columns (total 12 columns):
 #   Column    Dtype         
---  ------    -----         
 0   date      datetime64[ns]
 1   대여시간     int64         
 2   대여소번호    int64         
 3   대여소명     object        
 4   대여구분코드   object        
 5   성별       object        
 6   연령대코드    object        
 7   이용건수     int64         
 8   운동량      float64       
 9   탄소량      float64       
 10  이동거리(M)  float64       
 11  이용시간(분)  int64         
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 320.9+ MB
```

B) Modify non-numeric data in 'carbon amount' and 'momentum' data

```
_bicycle04[["탄소량","운동량"]] =
_bicycle04[["탄소량","운동량"]].apply(pd.to_numeric, errors='coerce')
```



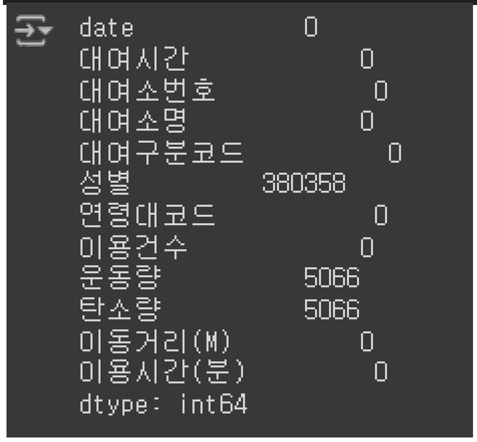
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3504724 entries, 0 to 3504723
Data columns (total 12 columns):
 #   Column    Dtype  
---  ------    -----  
 0   대여일자     object 
 1   대여시간     int64  
 2   대여소번호    int64  
 3   대여소명     object 
 4   대여구분코드   object 
 5   성별       object 
 6   연령대코드    object 
 7   이용건수     int64  
 8   운동량      object 
 9   탄소량      object 
 10  이동거리(M)  float64
 11  이용시간(분)  int64  
dtypes: float64(1), int64(4), object(7)
memory usage: 320.9+ MB
```



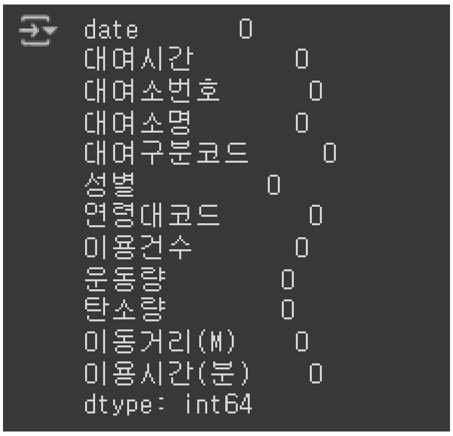
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3504724 entries, 0 to 3504723
Data columns (total 12 columns):
 #   Column    Dtype         
---  ------    -----         
 0   date      datetime64[ns]
 1   대여시간     int64         
 2   대여소번호    int64         
 3   대여소명     object        
 4   대여구분코드   object        
 5   성별       object        
 6   연령대코드    object        
 7   이용건수     int64         
 8   운동량      float64       
 9   탄소량      float64       
 10  이동거리(M)  float64       
 11  이용시간(분)  int64         
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 320.9+ MB
```

C) Drop missing data for ‘성별’ data

```
bicycle04_cleaned = _bicycle04.dropna()
# In order to use only data where gender data exists, drop because the
number of data is sufficient
bicycle04_cleaned.shape
```



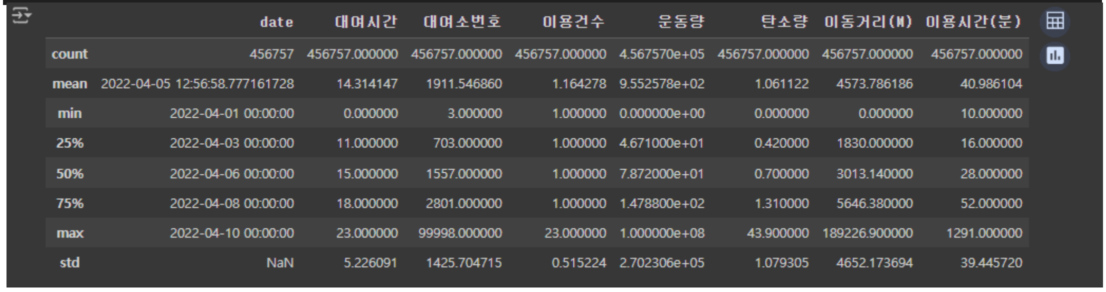
```
date            0
대여시간          0
대여소번호         0
대여소명          0
대여구분코드         0
성별       380358
연령대코드         0
이용건수          0
운동량         5066
탄소량         5066
이동거리(M)       0
이용시간(분)       0
dtype: int64
```



```
date        0
대여시간       0
대여소번호      0
대여소명       0
대여구분코드     0
성별        0
연령대코드      0
이용건수       0
운동량       0
탄소량       0
이동거리(M)    0
이용시간(분)    0
dtype: int64
```

D) Drop for less than 10 minutes in ‘이용시간(분)’ data

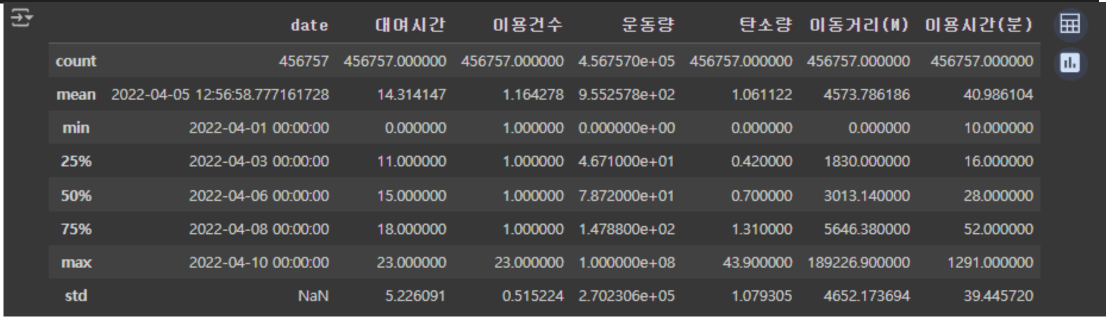
```
bicycle04_cleaned = bicycle04_cleaned[bicycle04_cleaned["이용시간(분)"]
>= 10 ]
# Only consider data that you think is meaningful: If the usage time is
less than 10 minutes, it is considered not to have been ridden and is
dropped.
```



	date	대여시간	대여소번호	이용건수	운동량	탄소량	이동거리(M)	이용시간(분)
count	456757	456757.000000	456757.000000	456757.000000	4.567570e+05	456757.000000	456757.000000	456757.000000
mean	2022-04-05 12:56:58.777161728	14.314147	1911.546860	1.164278	9.552578e+02	1.061122	4573.786186	40.986104
min	2022-04-01 00:00:00	0.000000	3.000000	1.000000	0.000000e+00	0.000000	0.000000	10.000000
25%	2022-04-03 00:00:00	11.000000	703.000000	1.000000	4.671000e+01	0.420000	1830.000000	16.000000
50%	2022-04-06 00:00:00	15.000000	1557.000000	1.000000	7.872000e+01	0.700000	3013.140000	28.000000
75%	2022-04-08 00:00:00	18.000000	2801.000000	1.000000	1.478800e+02	1.310000	5646.380000	52.000000
max	2022-04-10 00:00:00	23.000000	99998.000000	23.000000	1.000000e+08	43.900000	189226.900000	1291.000000
std	NaN	5.226091	1425.704715	0.515224	2.702306e+05	1.079305	4652.173694	39.445720

E) Drop ‘대여소번호’, ‘대여소명’, ‘대여구분코드’ data

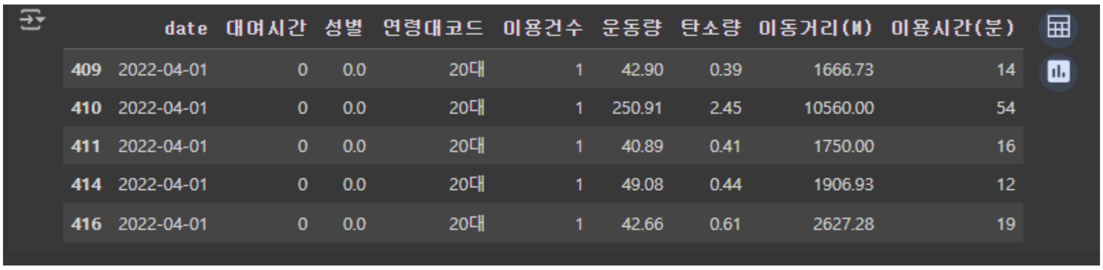
```
bicycle04_cleaned = bicycle04_cleaned.drop(columns=["대여소번호",
"대여소명", "대여구분코드"])
# Data not required for our business objective
```



	date	대여시간	이용건수	운동량	탄소량	이동거리(M)	이용시간(분)
count	456757	456757.000000	456757.000000	4.567570e+05	456757.000000	456757.000000	456757.000000
mean	2022-04-05 12:56:58.777161728	14.314147	1.164278	9.552578e+02	1.061122	4573.786186	40.986104
min	2022-04-01 00:00:00	0.000000	1.000000	0.000000e+00	0.000000	0.000000	10.000000
25%	2022-04-03 00:00:00	11.000000	1.000000	4.671000e+01	0.420000	1830.000000	16.000000
50%	2022-04-06 00:00:00	15.000000	1.000000	7.872000e+01	0.700000	3013.140000	28.000000
75%	2022-04-08 00:00:00	18.000000	1.000000	1.478800e+02	1.310000	5646.380000	52.000000
max	2022-04-10 00:00:00	23.000000	23.000000	1.000000e+08	43.900000	189226.900000	1291.000000
std	NaN	5.226091	0.515224	2.702306e+05	1.079305	4652.173694	39.445720

F) Encoding the ‘성별’ label

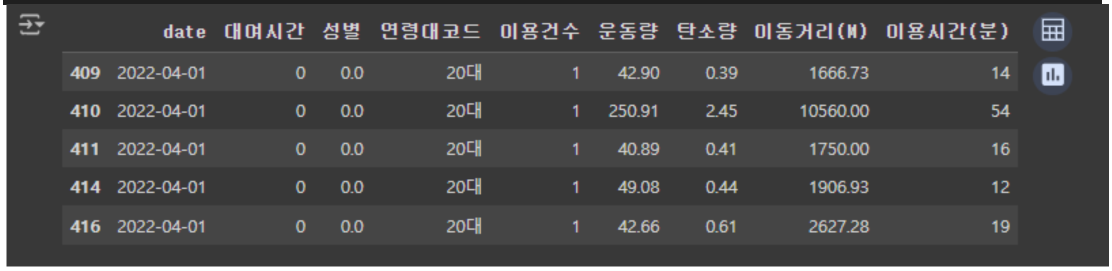
```
bicycle04_cleaned['성별'] = bicycle04_cleaned['성별'].map({'F': 0, 'M':
1})
```



	date	대여시간	성별	연령대코드	이용건수	운동량	탄소량	이동거리(M)	이용시간(분)
409	2022-04-01	0	0.0	20대	1	42.90	0.39	1666.73	14
410	2022-04-01	0	0.0	20대	1	250.91	2.45	10560.00	54
411	2022-04-01	0	0.0	20대	1	40.89	0.41	1750.00	16
414	2022-04-01	0	0.0	20대	1	49.08	0.44	1906.93	12
416	2022-04-01	0	0.0	20대	1	42.66	0.61	2627.28	19

G) ‘기타’ drops in '연령대'

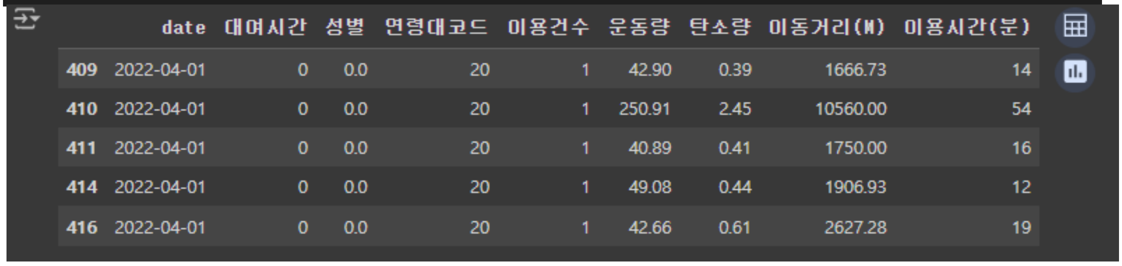
```
bicycle04_cleaned =
bicycle04_cleaned.drop(bicycle04_cleaned[bicycle04_cleaned['연령대코드']
== '기타'].index)
```



	date	대여시간	성별	연령대코드	이용건수	운동량	탄소량	이동거리(M)	이용시간(분)
409	2022-04-01	0	0.0	20대	1	42.90	0.39	1666.73	14
410	2022-04-01	0	0.0	20대	1	250.91	2.45	10560.00	54
411	2022-04-01	0	0.0	20대	1	40.89	0.41	1750.00	16
414	2022-04-01	0	0.0	20대	1	49.08	0.44	1906.93	12
416	2022-04-01	0	0.0	20대	1	42.66	0.61	2627.28	19

H) Encoding the ‘연령대’ label

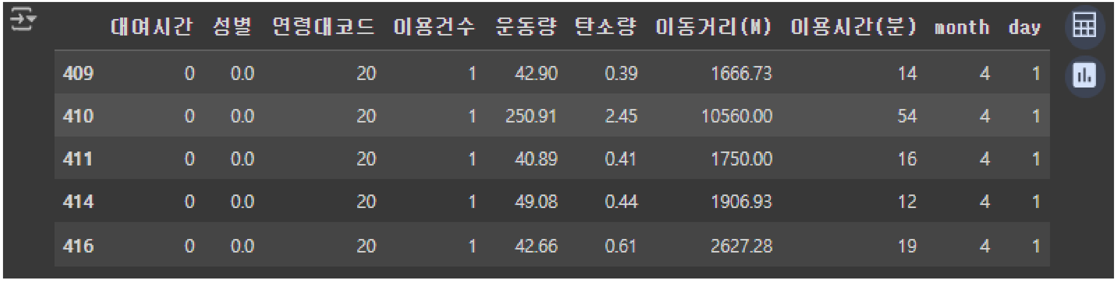
```
bicycle04_cleaned['연령대코드'] =
bicycle04_cleaned['연령대코드'].map({'~10대':10, '20대': 20, '30대': 30,
'40대': 40, '50대': 50, '60대': 60, '70대이상': 70})
#'~10대' are children under 10 years old + teenagers over 10 years old
-> Since most of them are teenagers over 10 years old who can rent
bicycles, they are classified as teenagers.
#'70대이상' refers to the elderly in their 70s and older -> Since the
number of data becomes significantly smaller from the 70s, people in
their 70s and older are classified as a group.
```



	date	대여시간	성별	연령대코드	이용건수	운동량	탄소량	이동거리(M)	이용시간(분)
409	2022-04-01	0	0.0	20	1	42.90	0.39	1666.73	14
410	2022-04-01	0	0.0	20	1	250.91	2.45	10560.00	54
411	2022-04-01	0	0.0	20	1	40.89	0.41	1750.00	16
414	2022-04-01	0	0.0	20	1	49.08	0.44	1906.93	12
416	2022-04-01	0	0.0	20	1	42.66	0.61	2627.28	19

I) ‘date’ data split by month, day and drop ‘date’

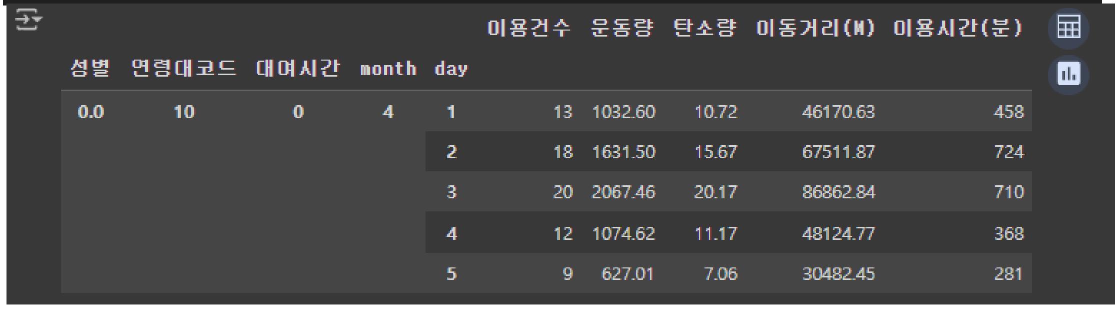
```
bicycle04_cleaned["month"] = bicycle04_cleaned["date"].dt.month
bicycle04_cleaned["day"] = bicycle04_cleaned["date"].dt.day
bicycle04_cleaned = bicycle04_cleaned.drop(columns="date")
```



	대여시간	성별	연령대코드	이용건수	운동량	탄소량	이동거리(M)	이용시간(분)	month	day
409	0	0.0	20	1	42.90	0.39	1666.73	14	4	1
410	0	0.0	20	1	250.91	2.45	10560.00	54	4	1
411	0	0.0	20	1	40.89	0.41	1750.00	16	4	1
414	0	0.0	20	1	49.08	0.44	1906.93	12	4	1
416	0	0.0	20	1	42.66	0.61	2627.28	19	4	1

J) Grouping of '운동량', '이동거리', '이용시간 according to '성별', '연령대코드', '대여시간', 'month', 'day'

```
bicycle04_grouped_sum =
bicycle04_cleaned.groupby([bicycle04_cleaned["성별"],bicycle04_cleaned[
"연령대코드"],bicycle04_cleaned["대여시간"],bicycle04_cleaned["month"],b
icycle04_cleaned["day"]]).sum()
# Reason for proceeding with sum: To compare the number of uses
bicycle04_grouped_sum.head()
```

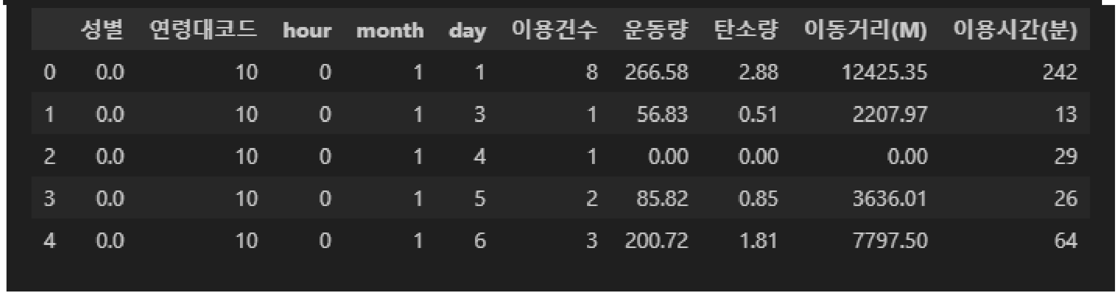


```
                                        이용건수  운동량   탄소량  이동거리(M)  이용시간(분)
성별  연령대코드  대여시간  month  day
0.0   10          0         4      1       13  1032.60  10.72   46170.63      458
                                   2       18  1631.50  15.67   67511.87      724
                                   3       20  2067.46  20.17   86862.84      710
                                   4       12  1074.62  11.17   48124.77      368
                                   5        9   627.01   7.06   30482.45      281
```

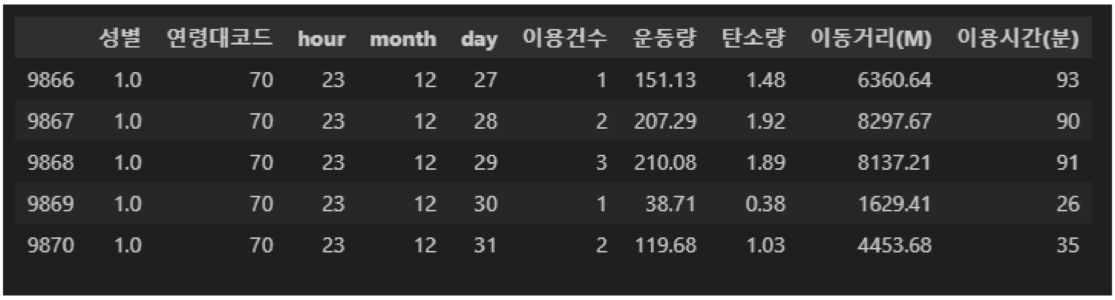
K) Apply the April data preprocessing process performed above to all month data and combine them into one data.

```
for i in bicycle_list:
    i["대여일자"] = pd.to_datetime(i["대여일자"], format='%Y-%m-%d')
    i.rename(columns={"대여일자":"date","
이용건수":"이용건수","대여시간":"hour"}, inplace=True)
    i[["탄소량","운동량"]] = i[["탄소량","운동량"]].apply(pd.to_numeric,
errors='coerce')
    i.dropna(inplace=True)
# (Convert sequence date to data datatime of object type, Modify
non-numeric data in 'carbon amount' and 'momentum' data) work is in
progress for all csv files.

for i in range(len(bicycle_list)):
    bicycle_list[i] = bicycle_list[i][bicycle_list[i]["이용시간(분)"] >=
10]
    bicycle_list[i] = bicycle_list[i].drop(columns=["대여소번호",
"대여소명", "대여구분코드"])
    bicycle_list[i]['성별'] = bicycle_list[i]['성별'].map({'F': 0, 'M':
1})
    bicycle_list[i] =
bicycle_list[i].drop(bicycle_list[i][bicycle_list[i]['연령대코드'] ==
'기타'].index)
    bicycle_list[i]['연령대코드'] =
bicycle_list[i]['연령대코드'].map({'~10대':10, '20대': 20, '30대': 30,
'40대': 40, '50대': 50, '60대': 60, '70대이상': 70})
    bicycle_list[i]["month"] = bicycle_list[i]["date"].dt.month
    bicycle_list[i]["day"] = bicycle_list[i]["date"].dt.day
    bicycle_list[i] = bicycle_list[i].drop(columns="date")
    bicycle_list[i] =
bicycle_list[i].groupby([bicycle_list[i]["성별"],bicycle_list[i]["연령대
코드"],bicycle_list[i]["hour"],bicycle_list[i]["month"],bicycle_list[i]
["day"]]).sum
```

```
bicycle_list[i].reset_index(inplace=True)
# Missing data, label encoding, and grouping are performed on all csv
files.

combined_bicycle = pd.concat(bicycle_list, ignore_index=False)
# Combine all csv files into one file
```



	성별	연령대코드	hour	month	day	이용건수	운동량	탄소량	이동거리(M)	이용시간(분)
0	0.0	10	0	1	1	8	266.58	2.88	12425.35	242
1	0.0	10	0	1	3	1	56.83	0.51	2207.97	13
2	0.0	10	0	1	4	1	0.00	0.00	0.00	29
3	0.0	10	0	1	5	2	85.82	0.85	3636.01	26
4	0.0	10	0	1	6	3	200.72	1.81	7797.50	64

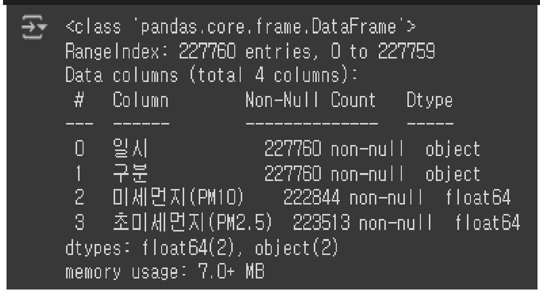


	성별	연령대코드	hour	month	day	이용건수	운동량	탄소량	이동거리(M)	이용시간(분)
9866	1.0	70	23	12	27	1	151.13	1.48	6360.64	93
9867	1.0	70	23	12	28	2	207.29	1.92	8297.67	90
9868	1.0	70	23	12	29	3	210.08	1.89	8137.21	91
9869	1.0	70	23	12	30	1	38.71	0.38	1629.41	26
9870	1.0	70	23	12	31	2	119.68	1.03	4453.68	35

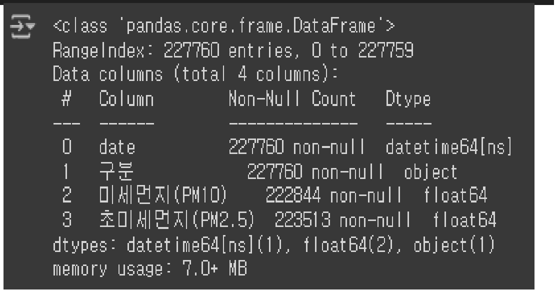
□ **Seoul Metropolitan City_Hourly (second) fine dust – 2022**

L) Convert sequence date to data datatime of object type

```
air["일시"] = air["일시"].astype("str")
air["일시"] = pd.to_datetime(air["일시"])
air.rename(columns={"일시":"date"}, inplace=True)
```



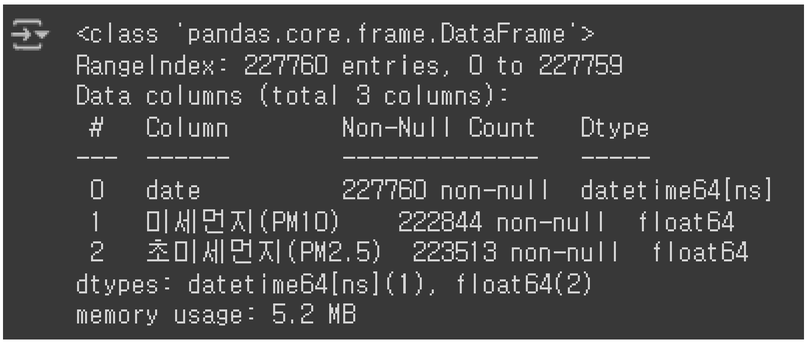
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227760 entries, 0 to 227759
Data columns (total 4 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   일시             227760 non-null  object
 1   구분             227760 non-null  object
 2   미세먼지(PM10)    222844 non-null  float64
 3   초미세먼지(PM2.5)  223513 non-null  float64
dtypes: float64(2), object(2)
memory usage: 7.0+ MB
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227760 entries, 0 to 227759
Data columns (total 4 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   date           227760 non-null  datetime64[ns]
 1   구분             227760 non-null  object
 2   미세먼지(PM10)    222844 non-null  float64
 3   초미세먼지(PM2.5)  223513 non-null  float64
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 7.0+ MB
```

M) Drop ‘구분’ data

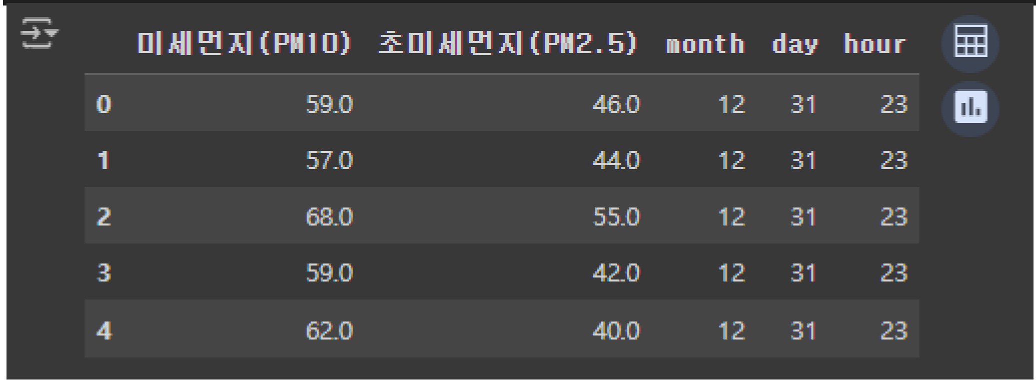
```
air = air.drop(columns="구분")
#unnecessary data
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227760 entries, 0 to 227759
Data columns (total 3 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   date          227760 non-null  datetime64[ns]
 1   미세먼지(PM10)    222844 non-null  float64
 2   초미세먼지(PM2.5)  223513 non-null  float64
dtypes: datetime64[ns](1), float64(2)
memory usage: 5.2 MB
```

N) ‘date’ data split by month and day and hour

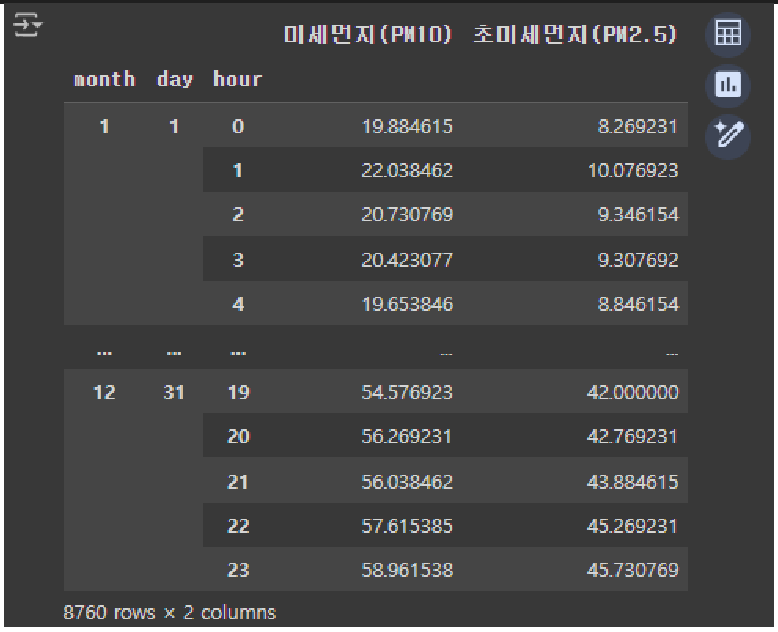
```
air["month"] = air["date"].dt.month
air["day"] = air["date"].dt.day
air["hour"] = air["date"].dt.hour

air = air.drop(columns="date")
air.head()
```



	미세먼지(PM10)	초미세먼지(PM2.5)	month	day	hour
0	59.0	46.0	12	31	23
1	57.0	44.0	12	31	23
2	68.0	55.0	12	31	23
3	59.0	42.0	12	31	23
4	62.0	40.0	12	31	23

O) Replaced with the average value of data according to the date of missing data for '미세먼지' and ‘초미세먼지’

```
air = air.groupby([air["month"],air["day"],air["hour"]]).mean()
air
```



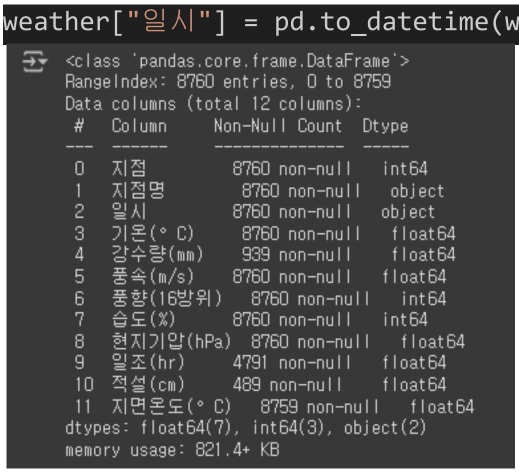
			미세먼지(PM10)	초미세먼지(PM2.5)
month	day	hour		
1	1	0	19.884615	8.269231
		1	22.038462	10.076923
		2	20.730769	9.346154
		3	20.423077	9.307692
		4	19.653846	8.846154
...	...	...	...	...
12	31	19	54.576923	42.000000
		20	56.269231	42.769231
		21	56.038462	43.884615
		22	57.615385	45.269231
		23	58.961538	45.730769

8760 rows × 2 columns

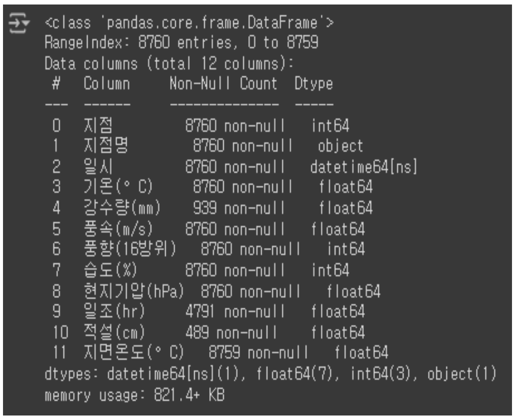
Weather Data - Seoul Metropolitan Government in 2022

P) Convert sequence date to data datatime of object type

```
weather["일시"] = pd.to_datetime(weather["일시"])
```



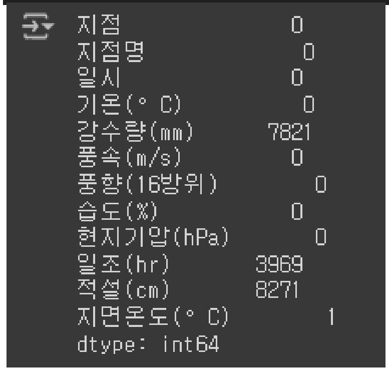
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   지점          8760 non-null   int64
 1   지점명         8760 non-null   object
 2   일시          8760 non-null   object
 3   기온(° C)      8760 non-null   float64
 4   강수량(mm)     939 non-null    float64
 5   풍속(m/s)     8760 non-null   float64
 6   풍향(16방위)   8760 non-null   int64
 7   습도(%)       8760 non-null   int64
 8   현지기압(hPa)  8760 non-null   float64
 9   일조(hr)      4791 non-null   float64
 10  적설(cm)      489 non-null    float64
 11  지면온도(° C)   8759 non-null   float64
dtypes: float64(7), int64(3), object(2)
memory usage: 821.4+ KB
```



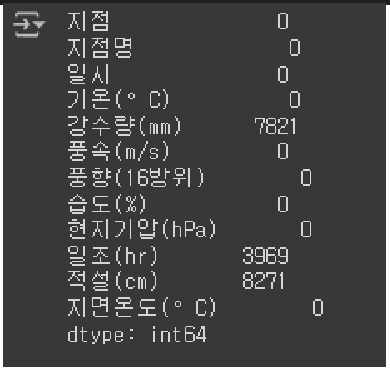
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   지점          8760 non-null   int64
 1   지점명         8760 non-null   object
 2   일시          8760 non-null   datetime64[ns]
 3   기온(° C)      8760 non-null   float64
 4   강수량(mm)     939 non-null    float64
 5   풍속(m/s)     8760 non-null   float64
 6   풍향(16방위)   8760 non-null   int64
 7   습도(%)       8760 non-null   int64
 8   현지기압(hPa)  8760 non-null   float64
 9   일조(hr)      4791 non-null   float64
 10  적설(cm)      489 non-null    float64
 11  지면온도(° C)   8759 non-null   float64
dtypes: datetime64[ns](1), float64(7), int64(3), object(1)
memory usage: 821.4+ KB
```

Q) Fill in the missing data for ‘지면온도’

```
weather["지면온도(°C)"].fillna(method='ffill', inplace=True)
# Correct missing data with the previous value
```



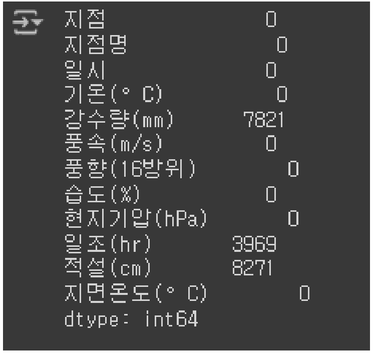
```
지점              0
지점명             0
일시              0
기온(° C)          0
강수량(mm)      7821
풍속(m/s)         0
풍향(16방위)       0
습도(%)           0
현지기압(hPa)      0
일조(hr)       3969
적설(cm)       8271
지면온도(° C)        1
dtype: int64
```



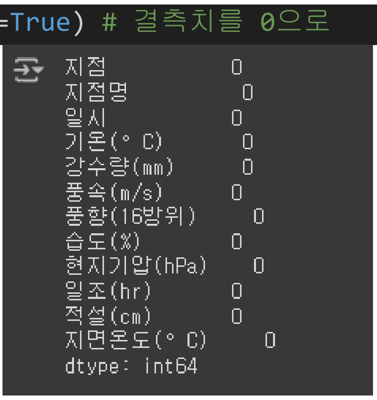
```
지점              0
지점명             0
일시              0
기온(° C)          0
강수량(mm)      7821
풍속(m/s)         0
풍향(16방위)       0
습도(%)           0
현지기압(hPa)      0
일조(hr)       3969
적설(cm)       8271
지면온도(° C)        0
dtype: int64
```

R) Fill in missing data with 0

```
weather.fillna(0,inplace=True) # 결측치를 0으로
```



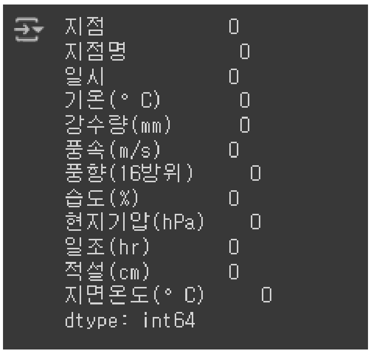
```
지점              0
지점명             0
일시              0
기온(° C)          0
강수량(mm)      7821
풍속(m/s)         0
풍향(16방위)       0
습도(%)           0
현지기압(hPa)      0
일조(hr)       3969
적설(cm)       8271
지면온도(° C)        0
dtype: int64
```



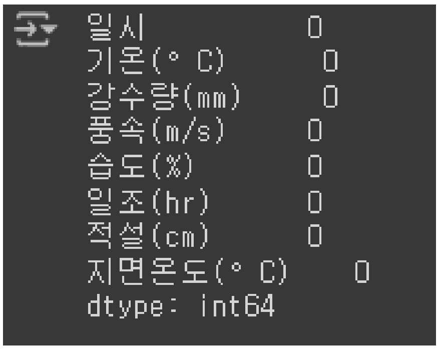
```
지점              0
지점명             0
일시              0
기온(° C)          0
강수량(mm)         0
풍속(m/s)         0
풍향(16방위)       0
습도(%)           0
현지기압(hPa)      0
일조(hr)          0
적설(cm)          0
지면온도(° C)        0
dtype: int64
```

S) Meaningless data drop

```
weather.drop(columns=["지점","지점명","풍향(16방위)","현지기압(hPa)"],
inplace=True)
```



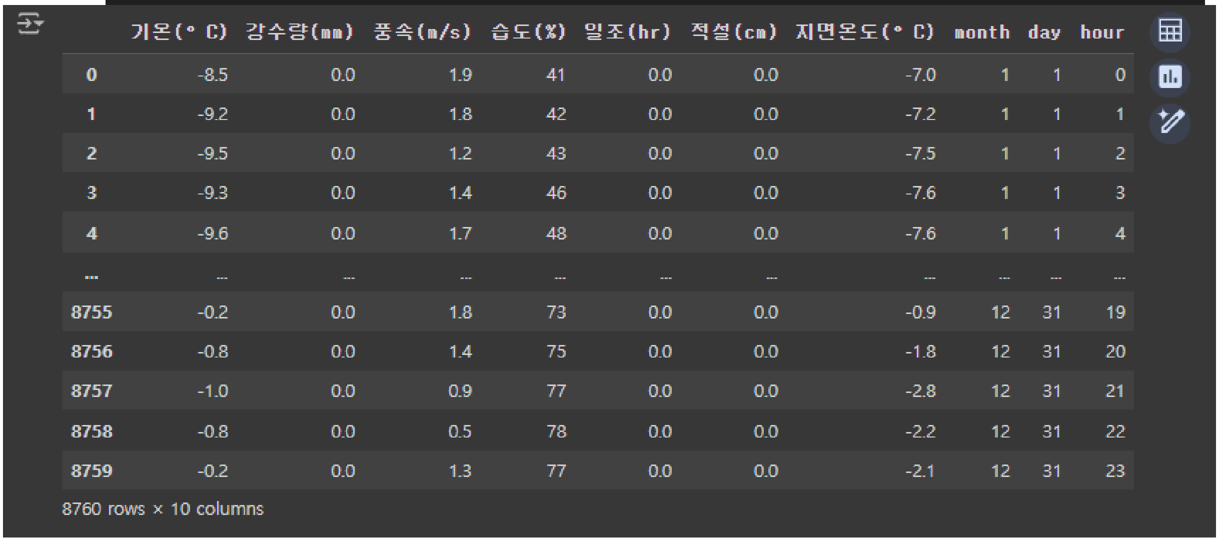
```
지점              0
지점명            0
일시              0
기온(° C)         0
강수량(mm)        0
풍속(m/s)         0
풍향(16방위)      0
습도(%)           0
현지기압(hPa)     0
일조(hr)          0
적설(cm)          0
지면온도(° C)     0
dtype: int64
```



```
일시           0
기온(° C)      0
강수량(mm)     0
풍속(m/s)      0
습도(%)        0
일조(hr)       0
적설(cm)       0
지면온도(° C)   0
dtype: int64
```

T) ‘date’ data split by month and day and hour

```
weather.rename(columns={"일시":"date"}, inplace=True)
weather["month"] = weather["date"].dt.month
weather["day"] = weather["date"].dt.day
weather["hour"] = weather["date"].dt.hour

weather = weather.drop(columns="date")
```

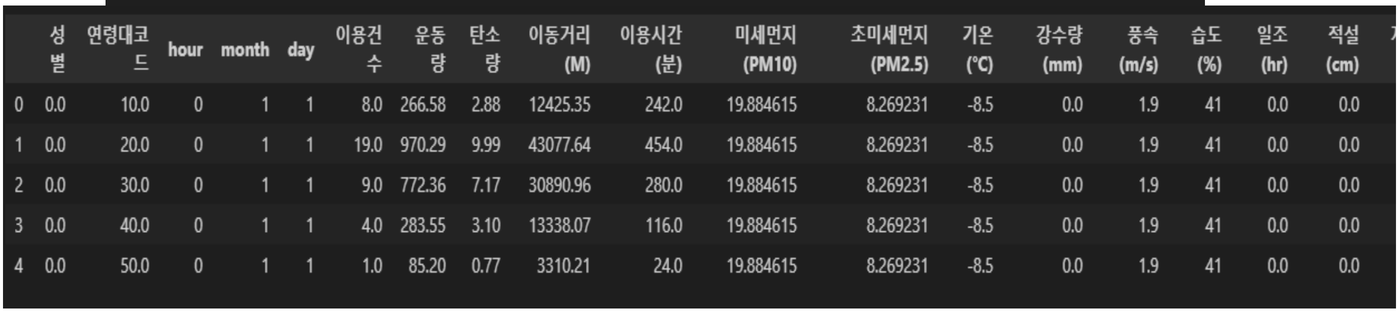


	기온(° C)	강수량(mm)	풍속(m/s)	습도(%)	일조(hr)	적설(cm)	지면온도(° C)	month	day	hour
0	-8.5	0.0	1.9	41	0.0	0.0	-7.0	1	1	0
1	-9.2	0.0	1.8	42	0.0	0.0	-7.2	1	1	1
2	-9.5	0.0	1.2	43	0.0	0.0	-7.5	1	1	2
3	-9.3	0.0	1.4	46	0.0	0.0	-7.6	1	1	3
4	-9.6	0.0	1.7	48	0.0	0.0	-7.6	1	1	4
...	...	...	...	...	...	...	...	...	...	...
8755	-0.2	0.0	1.8	73	0.0	0.0	-0.9	12	31	19
8756	-0.8	0.0	1.4	75	0.0	0.0	-1.8	12	31	20
8757	-1.0	0.0	0.9	77	0.0	0.0	-2.8	12	31	21
8758	-0.8	0.0	0.5	78	0.0	0.0	-2.2	12	31	22
8759	-0.2	0.0	1.3	77	0.0	0.0	-2.1	12	31	23

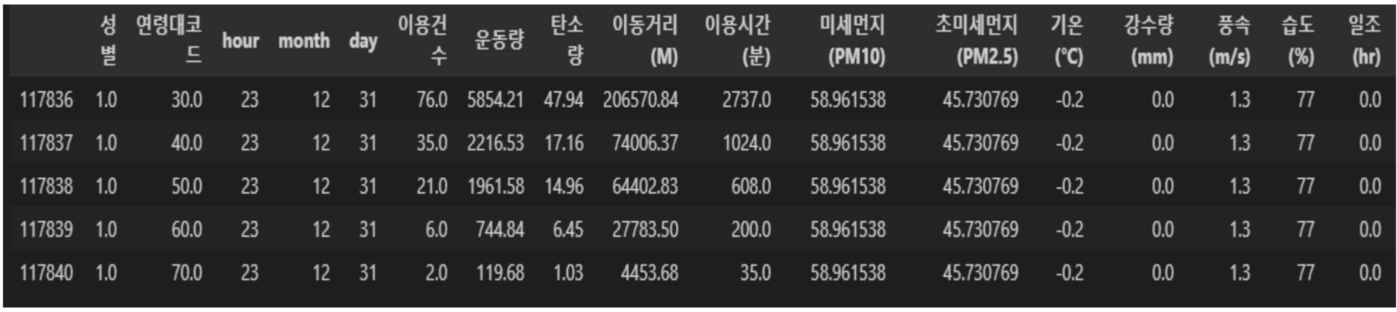
8760 rows × 10 columns

□ **Combining three datasets into one table**

```
df = pd.merge(combined_bicycle, pd.merge(air, weather,
on=['month','day','hour'], how='outer'), on=['month','day','hour'],
how='outer')
df.to_csv("data/final_data.csv", index=False)
# Combine based on 'date'
```



	성별	연령대코드	hour	month	day	이용건수	운동량	탄소량	이동거리 (M)	이용시간 (분)	미세먼지 (PM10)	초미세먼지 (PM2.5)	기온 (℃)	강수량 (mm)	풍속 (m/s)	습도 (%)	일조 (hr)	적설 (cm)
0	0.0	10.0	0	1	1	8.0	266.58	2.88	12425.35	242.0	19.884615	8.269231	-8.5	0.0	1.9	41	0.0	0.0
1	0.0	20.0	0	1	1	19.0	970.29	9.99	43077.64	454.0	19.884615	8.269231	-8.5	0.0	1.9	41	0.0	0.0
2	0.0	30.0	0	1	1	9.0	772.36	7.17	30890.96	280.0	19.884615	8.269231	-8.5	0.0	1.9	41	0.0	0.0
3	0.0	40.0	0	1	1	4.0	283.55	3.10	13338.07	116.0	19.884615	8.269231	-8.5	0.0	1.9	41	0.0	0.0
4	0.0	50.0	0	1	1	1.0	85.20	0.77	3310.21	24.0	19.884615	8.269231	-8.5	0.0	1.9	41	0.0	0.0



	성별	연령대코드	hour	month	day	이용건수	운동량	탄소량	이동거리 (M)	이용시간 (분)	미세먼지 (PM10)	초미세먼지 (PM2.5)	기온 (℃)	강수량 (mm)	풍속 (m/s)	습도 (%)	일조 (hr)
117836	1.0	30.0	23	12	31	76.0	5854.21	47.94	206570.84	2737.0	58.961538	45.730769	-0.2	0.0	1.3	77	0.0
117837	1.0	40.0	23	12	31	35.0	2216.53	17.16	74006.37	1024.0	58.961538	45.730769	-0.2	0.0	1.3	77	0.0
117838	1.0	50.0	23	12	31	21.0	1961.58	14.96	64402.83	608.0	58.961538	45.730769	-0.2	0.0	1.3	77	0.0
117839	1.0	60.0	23	12	31	6.0	744.84	6.45	27783.50	200.0	58.961538	45.730769	-0.2	0.0	1.3	77	0.0
117840	1.0	70.0	23	12	31	2.0	119.68	1.03	4453.68	35.0	58.961538	45.730769	-0.2	0.0	1.3	77	0.0

- **Reduce memory usage**

A) Calculate initial memory usage

```
 # Calculate initial memory usage
    start_mem = df.memory_usage().sum() / 1024**2  # Step: Initial
Memory Calculation
    print('Memory usage of dataframe is {:.2f}
MB'.format(start_mem))
    # Explanation: Calculate the initial memory usage of the
dataframe in megabytes and print it.
```

B) Convert data type for each column

```
# Iterate over each column in the dataframe
    for col in df.columns:
        col_type = df[col].dtype.name  # Step: Get Column Data Type
        # Explanation: Get the data type of the current column.

        # Check if column type is not datetime or category
        if ((col_type != 'datetime64[ns]') & (col_type !=
'category')):
            # If column type is not object, process for memory
reduction
            if (col_type != 'object'):
                c_min = df[col].min()  # Step: Get Minimum Value
                c_max = df[col].max()  # Step: Get Maximum Value
                # Explanation: Get the minimum and maximum values of
the current column.
```

C) Integer data type conversion

```
# Check and convert integer columns to appropriate types
                if str(col_type)[:3] == 'int':
                    if c_min > np.iinfo(np.int8).min and c_max <
np.iinfo(np.int8).max:
```

```
                        df[col] = df[col].astype(np.int8)  # Convert to int8
                    elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                        df[col] = df[col].astype(np.int16)  # Convert to int16
                    elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                        df[col] = df[col].astype(np.int32)  # Convert to int32
                    elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                        df[col] = df[col].astype(np.int64)  # Convert to int64
                # Explanation: For integer columns, convert to the smallest appropriate integer type (int8, int16, int32, int64) based on the minimum and maximum values.
```

D) Floating point data type conversion

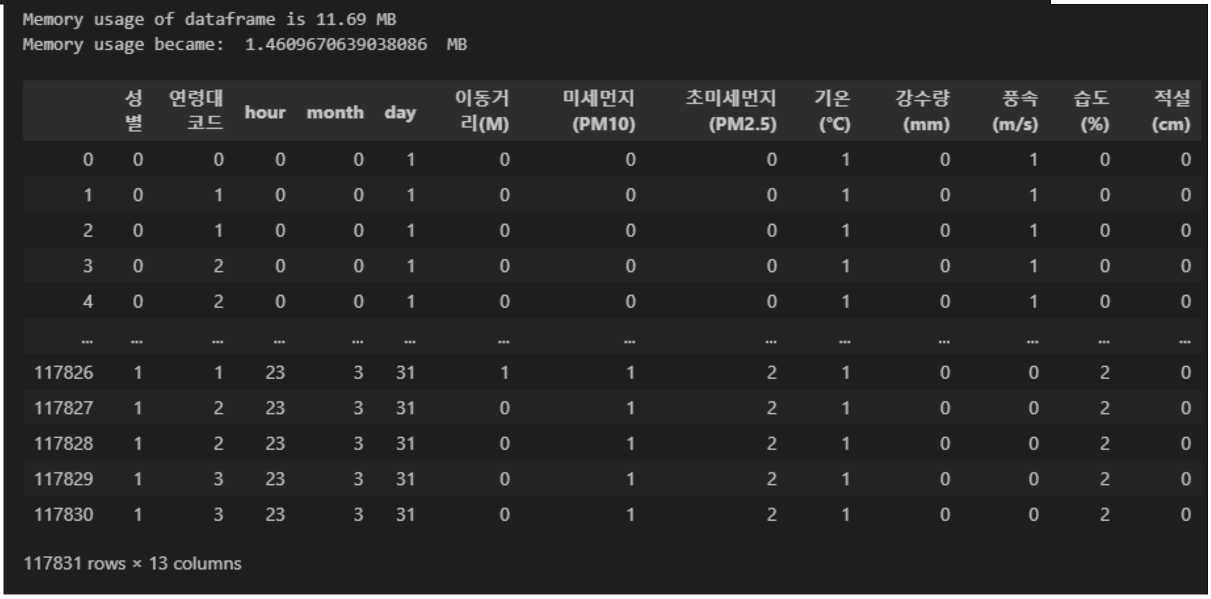
```
# Check and convert float columns to appropriate types
                else:
                    if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                        df[col] = df[col].astype(np.float16)  # Convert to float16
                    elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                        df[col] = df[col].astype(np.float32)  # Convert to float32
                    else:
                        pass  # Keep the original type if it doesn't fit into the smaller types
                # Explanation: For float columns, convert to the smallest appropriate float type (float16, float32) based on the minimum and maximum values.
```

E) String data type conversion

```
else:
                df[col] = df[col].astype('category')  # Convert object columns to category type
                # Explanation: Convert object type columns to category type to save memory.
```

F) Final memory usage calculation and output

```
    # Calculate final memory usage
    mem_usg = df.memory_usage().sum() / 1024**2  # Step: Final
Memory Calculation
    print("Memory usage became: ", mem_usg, " MB")
    # Explanation: Calculate the final memory usage of the dataframe
in megabytes and print it.

    return df  # Return the optimized dataframe

# Reduce memory usage of the dataframe
reduce_memory_usage(df)  # Step: Memory Reduction
# Explanation: Apply the reduce_memory_usage function to the
dataframe to reduce its memory footprint.

# Meaningless in the current code, but used initially due to memory
constraints!
reduce_memory_usage(df)
```



```
Memory usage of dataframe is 11.69 MB
Memory usage became:  1.4609670639038086  MB
```

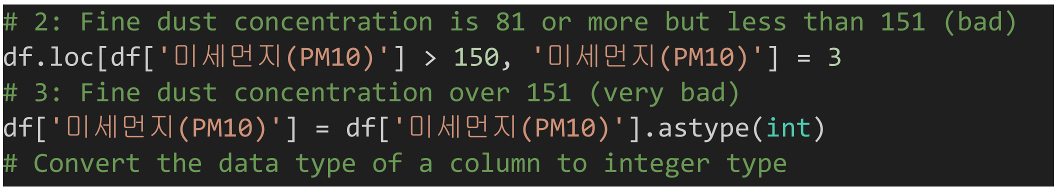
	성별	연령대코드	hour	month	day	이동거리(M)	미세먼지(PM10)	초미세먼지(PM2.5)	기온(°C)	강수량(mm)	풍속(m/s)	습도(%)	적설(cm)
0	0	0	0	0	1	0	0	0	1	0	1	0	0
1	0	1	0	0	1	0	0	0	1	0	1	0	0
2	0	1	0	0	1	0	0	0	1	0	1	0	0
3	0	2	0	0	1	0	0	0	1	0	1	0	0
4	0	2	0	0	1	0	0	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
117826	1	1	23	3	31	1	1	2	1	0	0	2	0
117827	1	2	23	3	31	0	1	2	1	0	0	2	0
117828	1	2	23	3	31	0	1	2	1	0	0	2	0
117829	1	3	23	3	31	0	1	2	1	0	0	2	0
117830	1	3	23	3	31	0	1	2	1	0	0	2	0

117831 rows × 13 columns

- **Classification – Decision Tree - Preprocessing**

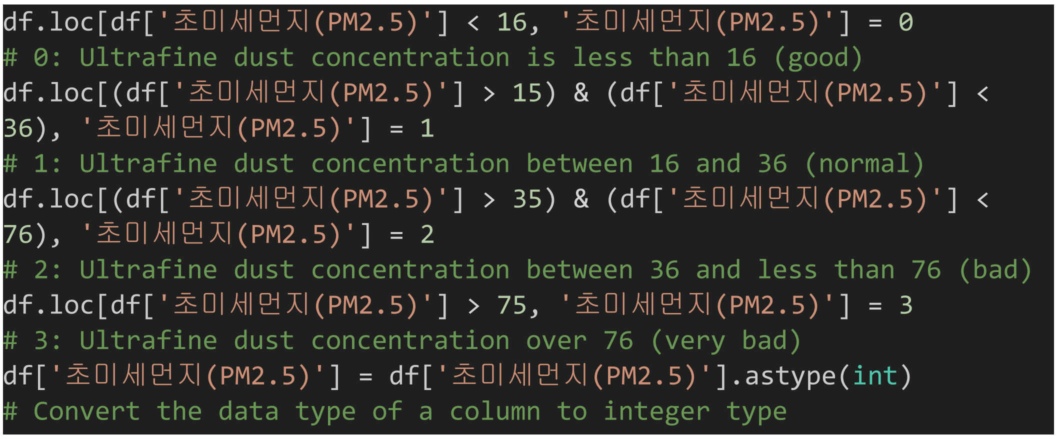
A) ‘미세먼지’ feature classification

```
df.loc[df['미세먼지(PM10)'] < 31, '미세먼지(PM10)'] = 0
# 0: Fine dust concentration is less than 31 (good)
df.loc[(df['미세먼지(PM10)'] > 30) & (df['미세먼지(PM10)'] < 81),
'미세먼지(PM10)'] = 1
# 1: Fine dust concentration between 31 and 81 (normal)
df.loc[(df['미세먼지(PM10)'] > 80) & (df['미세먼지(PM10)'] < 151),
'미세먼지(PM10)'] = 2
```



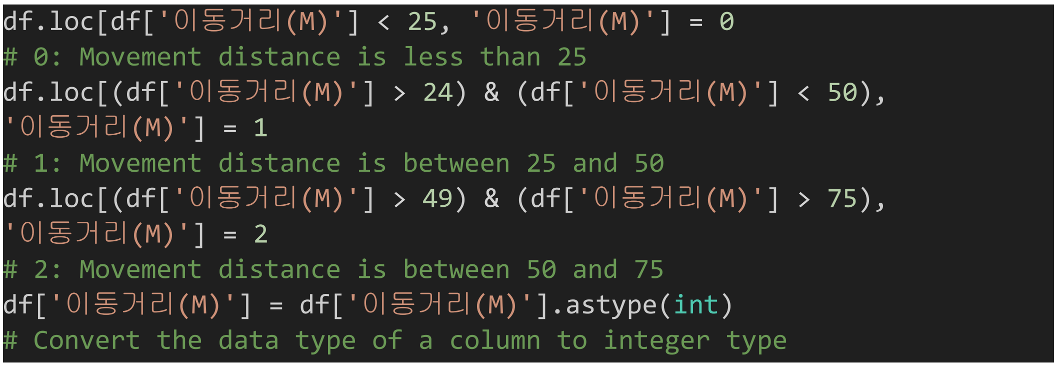
```
# 2: Fine dust concentration is 81 or more but less than 151 (bad)
df.loc[df['미세먼지(PM10)'] > 150, '미세먼지(PM10)'] = 3
# 3: Fine dust concentration over 151 (very bad)
df['미세먼지(PM10)'] = df['미세먼지(PM10)'].astype(int)
# Convert the data type of a column to integer type
```

B) ‘초미세먼지’ feature classification



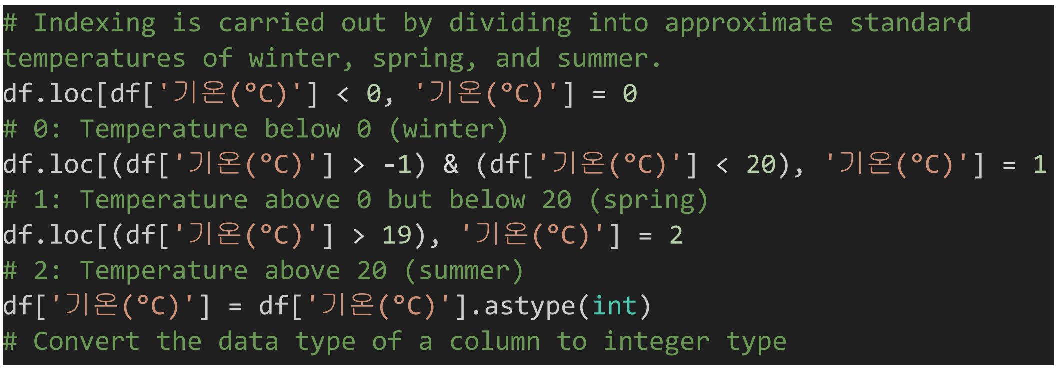
```
df.loc[df['초미세먼지(PM2.5)'] < 16, '초미세먼지(PM2.5)'] = 0
# 0: Ultrafine dust concentration is less than 16 (good)
df.loc[(df['초미세먼지(PM2.5)'] > 15) & (df['초미세먼지(PM2.5)'] <
36), '초미세먼지(PM2.5)'] = 1
# 1: Ultrafine dust concentration between 16 and 36 (normal)
df.loc[(df['초미세먼지(PM2.5)'] > 35) & (df['초미세먼지(PM2.5)'] <
76), '초미세먼지(PM2.5)'] = 2
# 2: Ultrafine dust concentration between 36 and less than 76 (bad)
df.loc[df['초미세먼지(PM2.5)'] > 75, '초미세먼지(PM2.5)'] = 3
# 3: Ultrafine dust concentration over 76 (very bad)
df['초미세먼지(PM2.5)'] = df['초미세먼지(PM2.5)'].astype(int)
# Convert the data type of a column to integer type
```

C) ‘이동거리’ feature classification



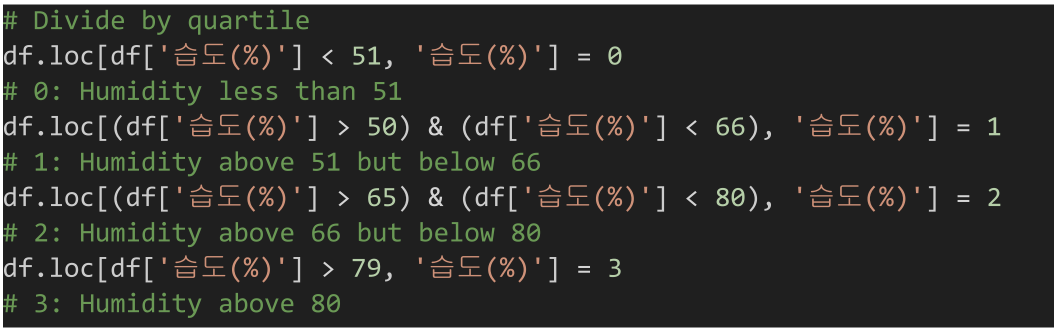
```
df.loc[df['이동거리(M)'] < 25, '이동거리(M)'] = 0
# 0: Movement distance is less than 25
df.loc[(df['이동거리(M)'] > 24) & (df['이동거리(M)'] < 50),
'이동거리(M)'] = 1
# 1: Movement distance is between 25 and 50
df.loc[(df['이동거리(M)'] > 49) & (df['이동거리(M)'] > 75),
'이동거리(M)'] = 2
# 2: Movement distance is between 50 and 75
df['이동거리(M)'] = df['이동거리(M)'].astype(int)
# Convert the data type of a column to integer type
```

D) ‘기온’ feature classification



```
# Indexing is carried out by dividing into approximate standard
temperatures of winter, spring, and summer.
df.loc[df['기온(°C)'] < 0, '기온(°C)'] = 0
# 0: Temperature below 0 (winter)
df.loc[(df['기온(°C)'] > -1) & (df['기온(°C)'] < 20), '기온(°C)'] = 1
# 1: Temperature above 0 but below 20 (spring)
df.loc[(df['기온(°C)'] > 19), '기온(°C)'] = 2
# 2: Temperature above 20 (summer)
df['기온(°C)'] = df['기온(°C)'].astype(int)
# Convert the data type of a column to integer type
```

E) ‘습도’ feature classification



```
# Divide by quartile
df.loc[df['습도(%)'] < 51, '습도(%)'] = 0
# 0: Humidity less than 51
df.loc[(df['습도(%)'] > 50) & (df['습도(%)'] < 66), '습도(%)'] = 1
# 1: Humidity above 51 but below 66
df.loc[(df['습도(%)'] > 65) & (df['습도(%)'] < 80), '습도(%)'] = 2
# 2: Humidity above 66 but below 80
df.loc[df['습도(%)'] > 79, '습도(%)'] = 3
# 3: Humidity above 80
```

```
df['습도(%)'] = df['습도(%)'].astype(int)
# Convert the data type of a column to integer type
```

F) ‘지면온도’ feature classification

```
# Divide by quartile
df.loc[df['지면온도(°C)'] < 5, '지면온도(°C)'] = 0
# 0: Ground temperature is less than 5
df.loc[(df['지면온도(°C)'] > 4) & (df['지면온도(°C)'] < 16),
'지면온도(°C)'] = 1
# 1: Ground temperature is between 5 and 16
df.loc[(df['지면온도(°C)'] > 15) & (df['지면온도(°C)'] < 25),
'지면온도(°C)'] = 2
# 2: Ground temperature is above 16 and below 25
df.loc[df['지면온도(°C)'] > 24, '지면온도(°C)'] = 3
# 3: Ground temperature is above 25
df['지면온도(°C)'] = df['지면온도(°C)'].astype(int)
# Convert the data type of a column to integer type
```

G) ‘풍속’ feature classification

```
# Divide by quartile
df.loc[df['풍속(m/s)'] <= 1.6, '풍속(m/s)'] = 0
# 0: Wind speed is 1.6 or less
df.loc[(df['풍속(m/s)'] >= 1.6) & (df['풍속(m/s)'] <= 2.2),
'풍속(m/s)'] = 1
# 1: Wind speed is 1.6 or more and 2.2 or less
df.loc[(df['풍속(m/s)'] >= 2.2) & (df['풍속(m/s)'] <= 3),
'풍속(m/s)'] = 2
# 2: Wind speed is 2.2 or more and 3 or less
df.loc[df['풍속(m/s)'] >= 3, '풍속(m/s)'] = 3
# 3: Wind speed is 3 or more
df['풍속(m/s)'] = df['풍속(m/s)'].astype(int)
# Convert the data type of a column to integer type
```

H) ‘강수량’ feature classification

```
df.loc[df['강수량(mm)'] > 0, '강수량(mm)'] = 1
# 1: Precipitation exceeds 0
df['강수량(mm)'] = df['강수량(mm)'].astype(int)
# Convert the data type of a column to integer type
```

I) ‘적설’ feature classification

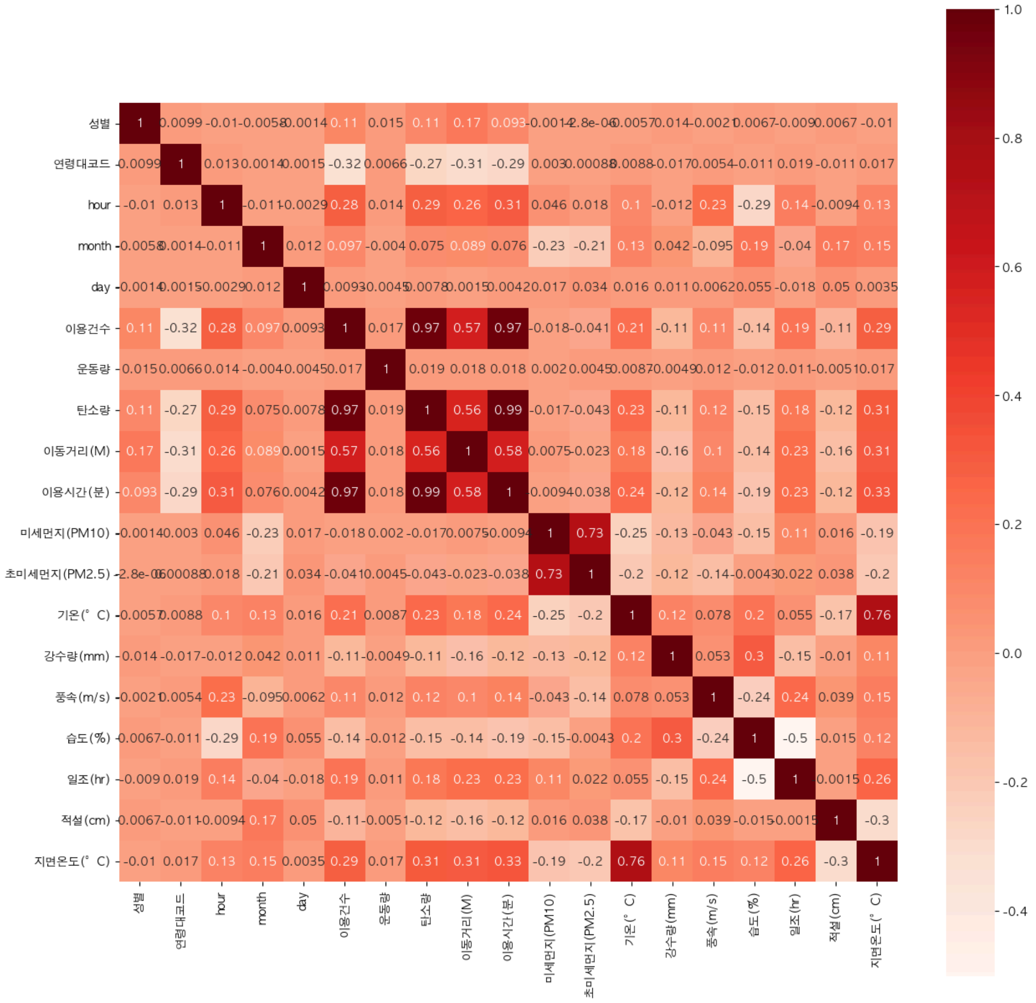
```
df.loc[df['적설(cm)'] > 0, '적설(cm)'] = 1
# 1: Snow cover exceeds 0
df['적설(cm)'] = df['적설(cm)'].astype(int)
# Convert the data type of a column to integer type
```

J) Convert the data type of a column to integer type

```
df["성별"] = df["성별"].astype(int)
df["연령대코드"] = df["연령대코드"].astype(int)
df["이동거리(M)"] = df["이동거리(M)"].astype(int)
```

K) Feature Selection

```
corrMatt = df[["성별", "연령대코드", "hour", "month", "day",
"이용건수", "운동량", "탄소량", "이동거리(M)", "이용시간(분)",
"미세먼지(PM10)", "초미세먼지(PM2.5)", "기온(°C)", "강수량(mm)",
"풍속(m/s)", "습도(%)", "일조(hr)", "적설(cm)", "지면온도(°C)"]]
corrMatt = corrMatt.corr()
fig, ax = plt.subplots()
fig.set_size_inches(15,15)
sns.heatmap(corrMatt,annot=True, square=True, cmap ='Reds')
```



```
# Strong Relationship with Travel Distance
df = df.drop(columns={"이용건수","운동량","탄소량", "이용시간(분)"})
# Significant Relationship with Temperature
df = df.drop(columns={"지면온도(°C)"})
# Dropped as it Appears to Have No Impact on the Analysis
df = df.drop(columns={"일조(hr)"})
df2 =df
# Determine the analysis was compromised due to inconsistent
particulate matter ratios, drop for now
df.drop(columns={"미세먼지(PM10)", "초미세먼지(PM2.5)"})
```

L) ‘연령대’ feature classification -For ease of judgment

```
df.loc[df["연령대코드"] < 11, "연령대코드"] = 0
# 0: If the age group is teenagers
```

```
df.loc[(df["연령대코드"] > 10) & (df["연령대코드"] < 31),
"연령대코드"] = 1
# 1: If the age group is in the 20s
df.loc[(df["연령대코드"] > 30) & (df["연령대코드"] < 51),
"연령대코드"] = 2
# 2: If the age group is in the 30s and 40s
df.loc[df["연령대코드"] > 50, "연령대코드"] = 3
# 3: If you are in your 50s or older
df["연령대코드"] = df["연령대코드"].astype(int)
# Convert the data type of a column to integer type
```

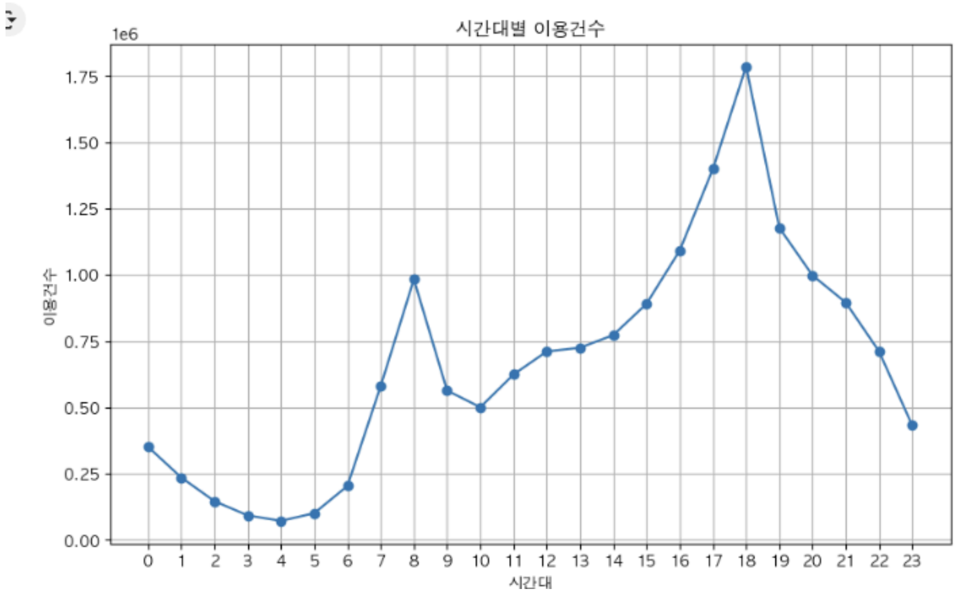
M) ‘month’ feature classification - For ease of judgment

```
df.loc[df["month"] <= 3, "month"] = 0
# 0: January ~ March
df.loc[(df["month"] >= 3) & (df["month"] <= 6), "month"] = 1
# 1: April ~ June
df.loc[(df["month"] >= 6) & (df["month"] <= 9), "month"] = 2
# 2: July ~ September
df.loc[df["month"] >= 9, "month"] = 3
# 3: October ~ December
df["month"] = df["month"].astype(int)
# Convert the data type of a column to integer type
```

□ **Dataset visualization after preprocessing**

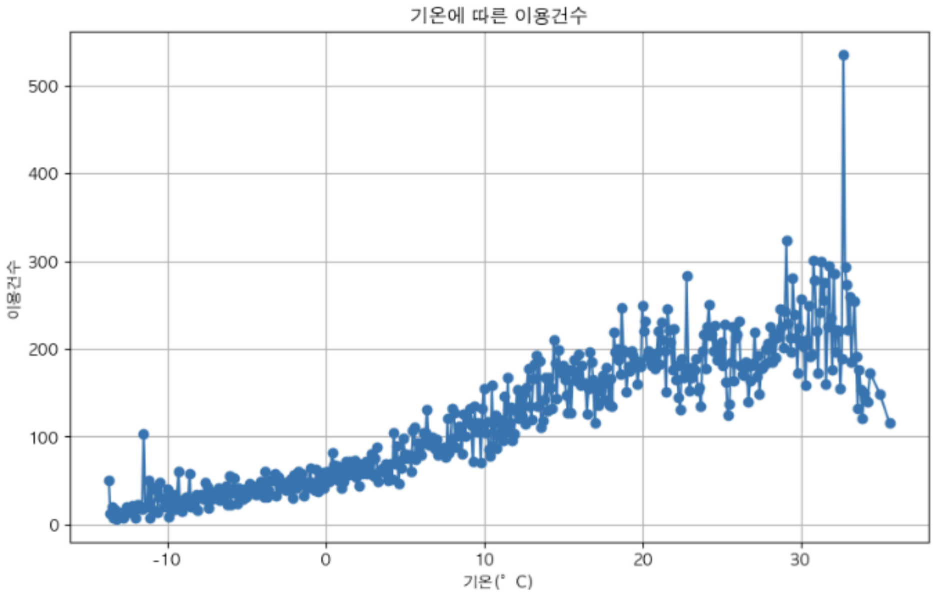
A) Number of uses by time zone

```
import pandas as pd
import matplotlib.pyplot as plt

# Importing data
data = pd.read_csv("data/final_data.csv")

# Calculation of the sum of the number of uses over time
hourly_total = data.groupby('hour')['이용건수'].sum()

#Visualization of the number of uses over time
hourly_total.plot(kind='line', marker='o', figsize=(10, 6))
plt.title('시간대별 이용건수')
plt.xlabel('시간대')
plt.ylabel('이용건수')
plt.xticks(range(24))
plt.grid(True)
plt.show()
```



B) Number of uses according to temperature

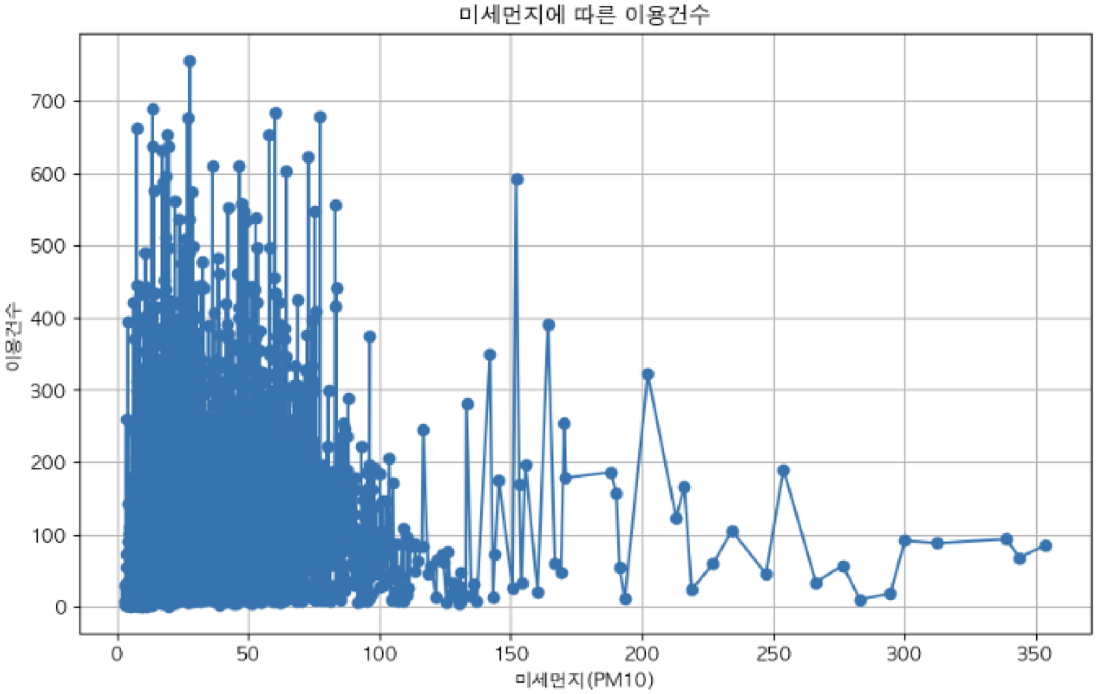
```
temperature_mean = data.groupby('기온(°C)')['이용건수'].mean()

# Visualization of the number of uses according to temperature
temperature_mean.plot(kind='line', marker='o', figsize=(10, 6))
plt.title('기온에 따른 이용건수')
plt.xlabel('기온(°C)')
plt.ylabel('이용건수')
plt.grid(True)
plt.show()
```



C) Number of uses due to fine dust

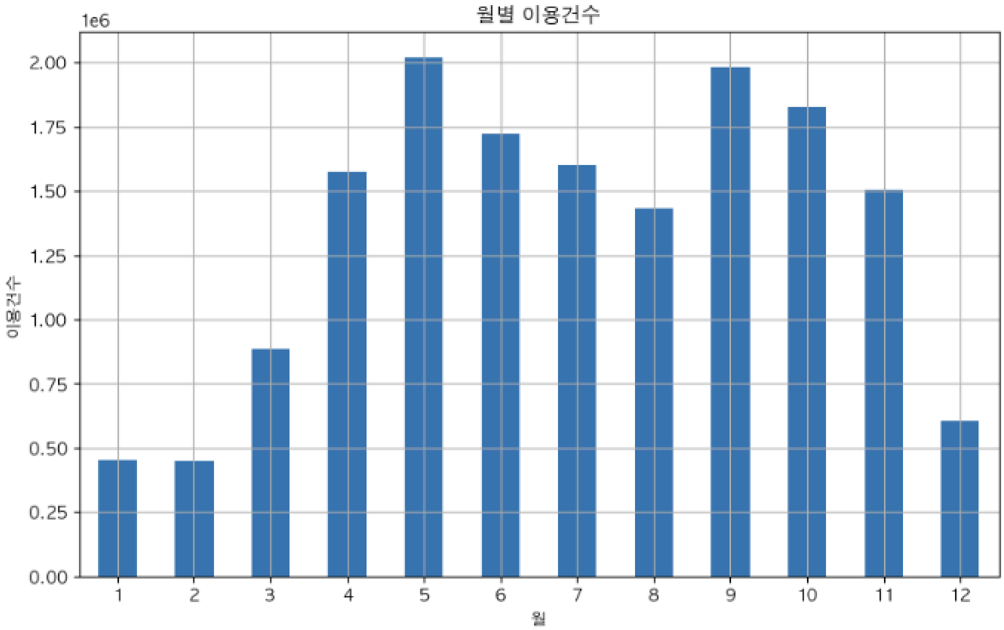
```
# Average calculation of the number of uses according to fine dust
```

```
pm10_mean = data.groupby('미세먼지(PM10)')['이용건수'].mean()

# Visualization of the number of uses according to fine dust
pm10_mean.plot(kind='line', marker='o', figsize=(10, 6))
plt.title('미세먼지에 따른 이용건수')
plt.xlabel('미세먼지(PM10)')
plt.ylabel('이용건수')
plt.grid(True)
plt.show()
```



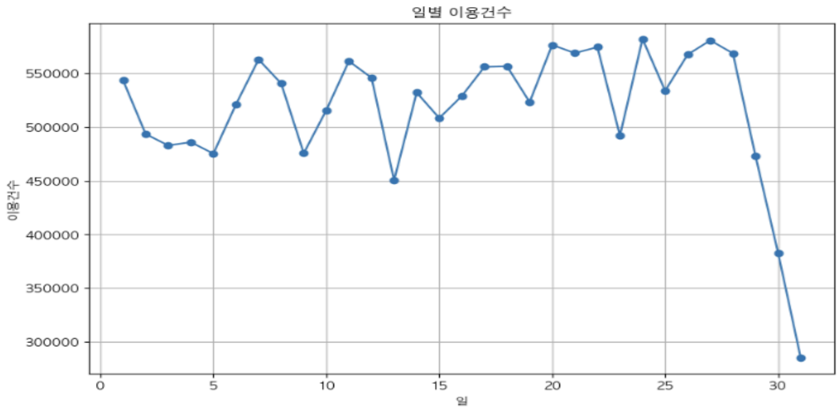
D) Number of monthly uses

```
# Calculate the sum of the number of uses in groups based on the
'month' column
monthly_total = data.groupby('month')['이용건수'].sum()

# Visualization of the number of monthly uses
monthly_total.plot(kind='bar', figsize=(10, 6))
plt.title('월별 이용건수')
plt.xlabel('월')
plt.ylabel('이용건수')
plt.xticks(rotation=0)  # No x-axis label rotation
plt.grid(True)
plt.show()
```



E) Number of uses per day

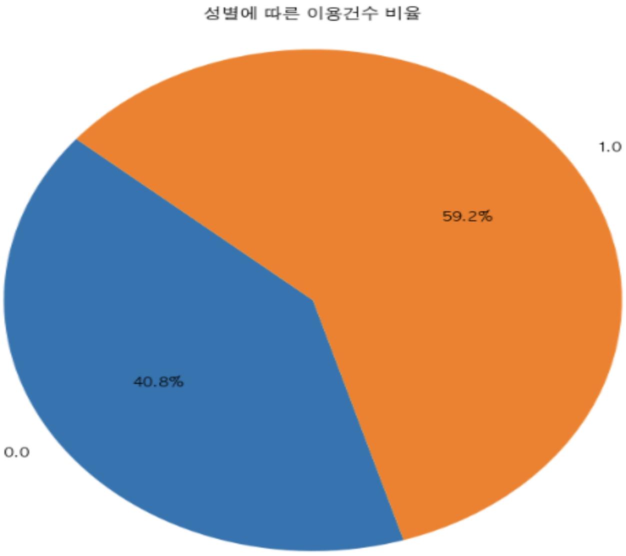
```
# Calculation of the total number of daily uses
daily_total = data.groupby('day')['이용건수'].sum()


# Visualization of the number of uses per day
daily_total.plot(kind='line', marker='o', figsize=(10, 6))
plt.title('일별 이용건수')
plt.xlabel('일')
plt.ylabel('이용건수')
plt.grid(True)
plt.show()
```



- Ratio of number of uses by gender

```
# Calculating the sum of the number of uses by gender
gender_total = data.groupby('성별')['이용건수'].sum()
# Visualize as a circle graph , ‘F’-> 0.0 , ‘M’->1.0
```

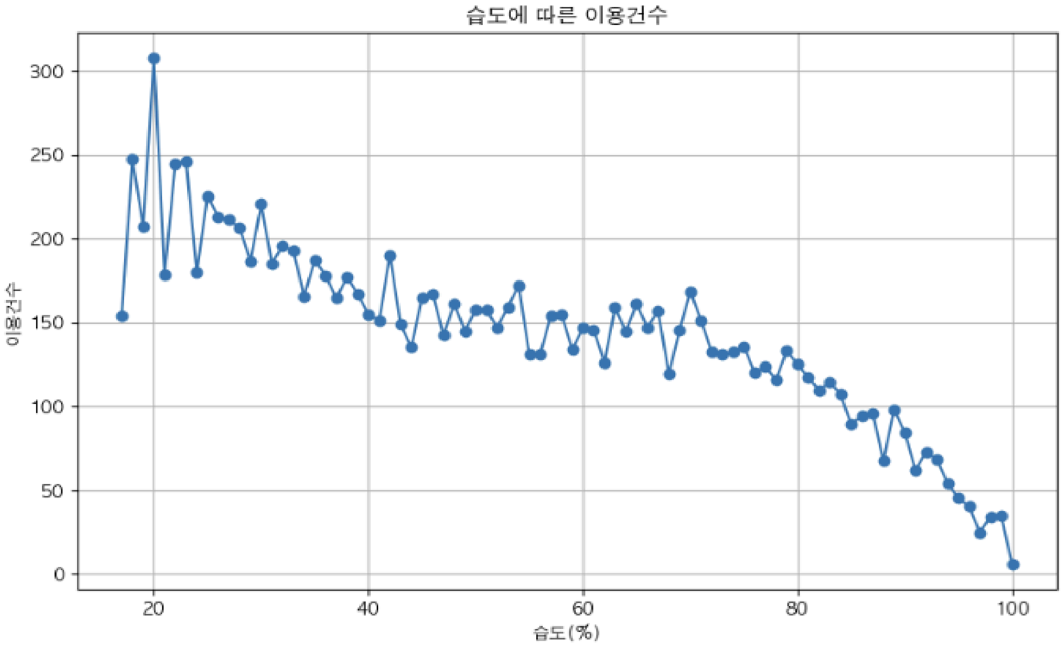
```
plt.figure(figsize=(8, 8))
plt.pie(gender_total, labels=gender_total.index, autopct='%1.1f%%',
startangle=140)
plt.title('성별에 따른 이용건수 비율')
plt.axis('equal')
plt.show()
```



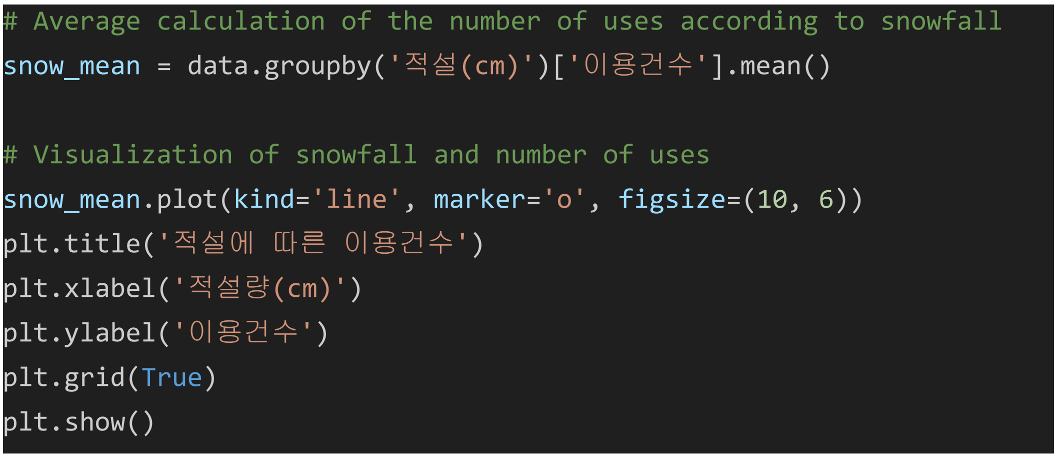
F) Number of uses according to humidity

```
# Average calculation of the number of uses according to humidity
humidity_mean = data.groupby('습도(%)')['이용건수'].mean()

# Visualization of humidity and number of uses
humidity_mean.plot(kind='line', marker='o', figsize=(10, 6))
plt.title('습도에 따른 이용건수')
plt.xlabel('습도(%)')
plt.ylabel('이용건수')
plt.grid(True)
plt.show()
```

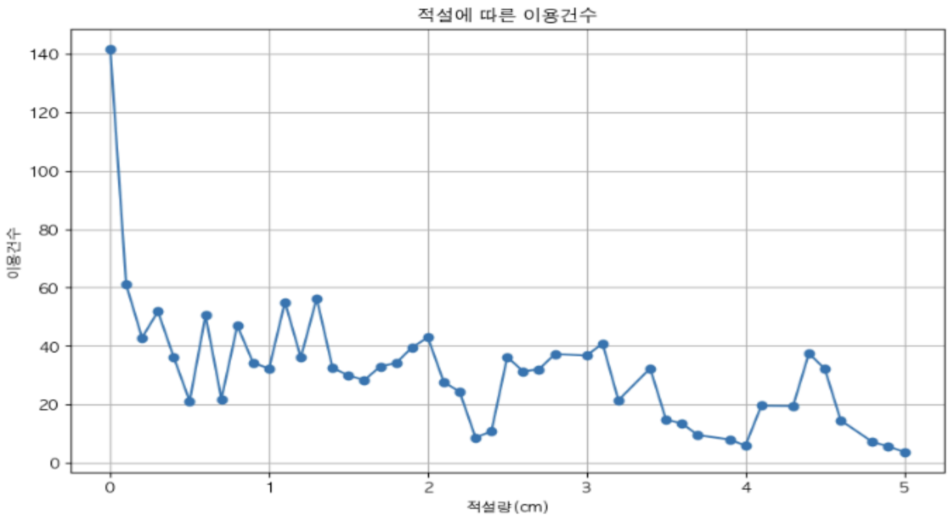
1-5



G) Number of uses according to snow cover

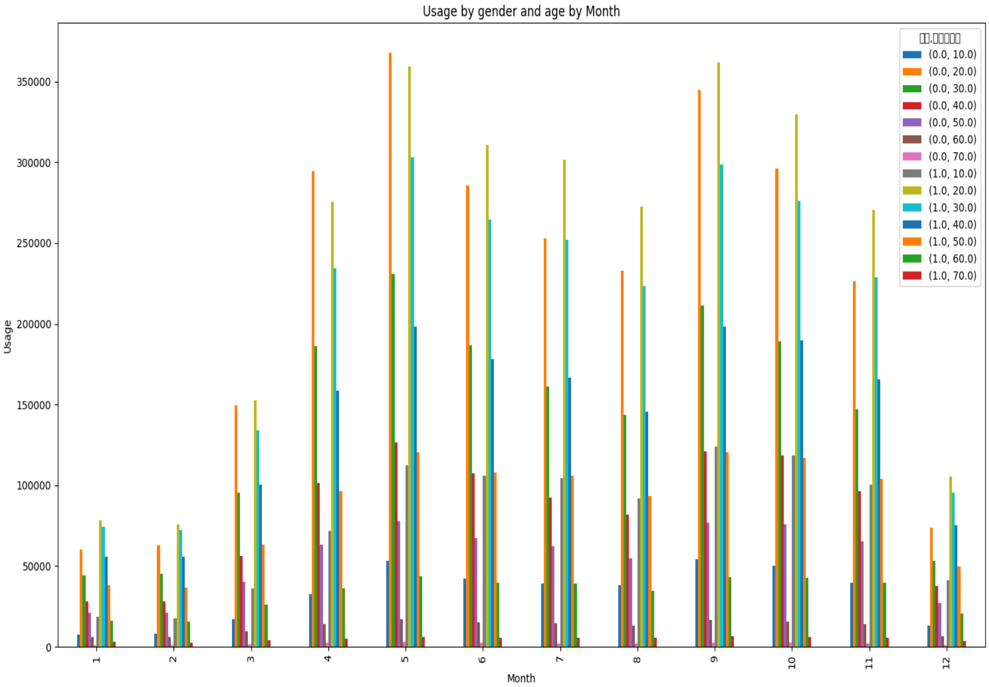


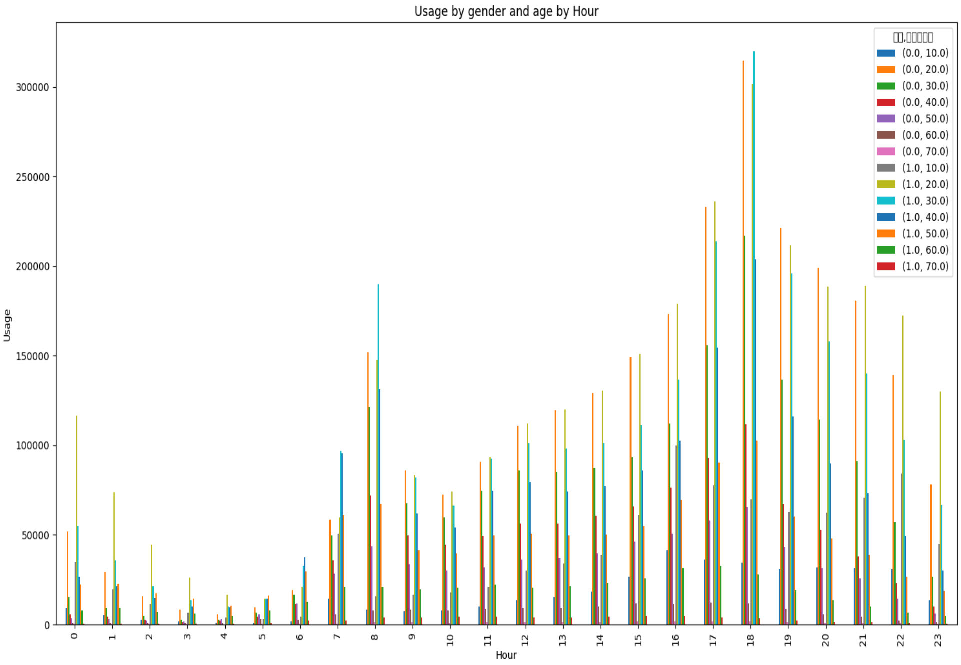
```
# Average calculation of the number of uses according to snowfall
snow_mean = data.groupby('적설(cm)')['이용건수'].mean()

# Visualization of snowfall and number of uses
snow_mean.plot(kind='line', marker='o', figsize=(10, 6))
plt.title('적설에 따른 이용건수')
plt.xlabel('적설량(cm)')
plt.ylabel('이용건수')
plt.grid(True)
plt.show()
```



H) Usage by gender and age by Month/ Usage by gender and age by Hou

```
grouped = df.groupby(['성별', '연령대코드',
'month']).sum().reset_index()
grouped2 = df.groupby(['성별', '연령대코드',
'hour']).sum().reset_index()
# Data pivoting for visualization
pivot_data = grouped.pivot_table(index='month', columns=['성별',
'연령대코드'], values='이용건수', aggfunc='sum')
pivot_data2 = grouped2.pivot_table(index='hour', columns=['성별',
'연령대코드'], values='이용건수', aggfunc='sum')


pivot_data.plot(kind='bar', figsize=(18, 10))
plt.title('Usage by gender and age by Month')
plt.ylabel('Usage')
plt.xlabel('Month')
plt.show()
pivot_data2.plot(kind='bar', figsize=(18, 10))
plt.title('Usage by gender and age by Hour')
plt.ylabel('Usage')
plt.xlabel('Hour')
plt.show()
```


Usage by gender and age by Month
Usage
350000
300000
250000
200000
150000
100000
50000
0
1
2
3
4
5
6
7
8
9
10
11
12
Month
□□,□□□□□
(0.0, 10.0)
(0.0, 20.0)
(0.0, 30.0)
(0.0, 40.0)
(0.0, 50.0)
(0.0, 60.0)
(0.0, 70.0)
(1.0, 10.0)
(1.0, 20.0)
(1.0, 30.0)
(1.0, 40.0)
(1.0, 50.0)
(1.0, 60.0)
(1.0, 70.0)


Usage by gender and age by Hour
Usage
300000
250000
200000
150000
100000
50000
0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Hour
□□,□□□□□
(0.0, 10.0)
(0.0, 20.0)
(0.0, 30.0)
(0.0, 40.0)
(0.0, 50.0)
(0.0, 60.0)
(0.0, 70.0)
(1.0, 10.0)
(1.0, 20.0)
(1.0, 30.0)
(1.0, 40.0)
(1.0, 50.0)
(1.0, 60.0)
(1.0, 70.0)

1-4) Modeling

- **Tree Model**

A) Select input features

```
# # Label data: This is the output data that the model will predict
X = df.drop(["이동거리(M)"], axis=1)
# Feature data: This is the input data that the model will learn
from
y = df[["이동거리(M)"]].astype(int)
# Label data: This is the output data that the model will predict
```

B) Train models and make predictions

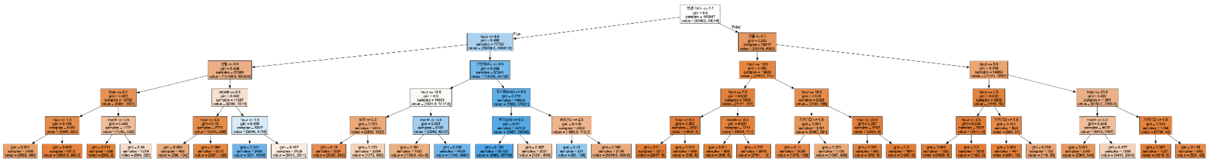
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.9, random_state=42)
# 90% of the total data is used as test data, and the remaining 10%
is used as training data
# Set a random seed to make data segmentation results reproducible
tree_model = DecisionTreeClassifier(random_state=100, max_depth=10)
# Set a random seed to make data segmentation results reproducible
tree_model.fit(X_train, y_train)
# Train a decision tree model using training data (X_train, y_train)
y_pred = tree_model.predict(X_test)
# Perform predictions on test data (X_test) using the learned model
and then save the prediction results on the test data.
```

C) Tree model

```
tree_model = DecisionTreeClassifier(random_state=100, max_depth=10)
tree_model.fit(X_train, y_train)
y_pred = tree_model.predict(X_test)
```

D) Visualization

```
# Reference:
https://velog.io/@gggggeun1/결정트리-그래프-exportgraphviz
# Generates an output file that graphviz can read and visualize as a
graph.
import graphviz
export_graphviz(tree_model, out_file='tree.dot',
feature_names=X_train.columns, impurity=True, filled=True)

# Visualize using the saved tree.dot file
with open("tree.dot")as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

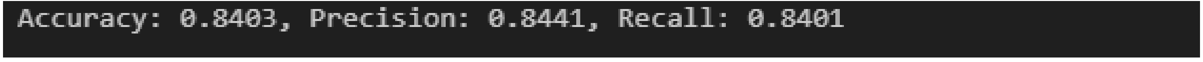


E) Evaluation

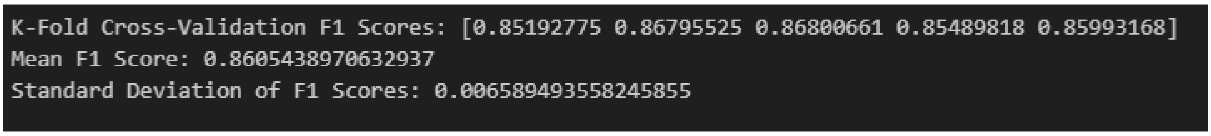
```
from sklearn.model_selection import GridSearchCV
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)  # Step: Accuracy Calculation
# Explanation: Calculate the accuracy of the model by comparing the predicted values (y_pred) with the actual values (y_test).

# Calculate the precision of the model
precision = precision_score(y_test, y_pred, average='macro')  # Step: Precision Calculation
# Explanation: Calculate the precision of the model. Precision is the ratio of true positive predictions to the sum of true positive and false positive predictions. The 'macro' average calculates the metric for each class and then takes the unweighted mean.

# Calculate the recall of the model
recall = recall_score(y_test, y_pred, average='macro')  # Step: Recall Calculation
# Explanation: Calculate the recall of the model. Recall is the ratio of true positive predictions to the sum of true positive and false negative predictions. The 'macro' average calculates the metric for each class and then takes the unweighted mean.

# Print the accuracy, precision, and recall of the model
print('Accuracy: {0:.4f}, Precision: {1:.4f}, Recall: {2:.4f}'.format(accuracy, precision, recall))  # Step: Print Metrics
# Explanation: Output the accuracy, precision, and recall scores of the model, formatted to four decimal places.

# Setting up 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)  # Step: Setup K-Fold CV
# Explanation: Initialize a K-Fold cross-validation object with 5 splits and shuffle the data.
# The random_state parameter ensures reproducibility.

# Perform cross-validation using F1 score as the scoring metric
```

```
cv_scores = cross_val_score(tree_model, X_test, y_test, cv=kf,
scoring='f1_macro')  # Step: Cross-Validation
# Explanation: Perform 5-fold cross-validation on the test data using
F1 as the scoring metric.
# The 'f1_macro' scoring parameter averages the F1 scores across all
classes to evaluate the overall model performance.

# Print cross-validation F1 scores
print(f"K-Fold Cross-Validation F1 Scores: {cv_scores}")  # Step: Print
CV Scores
# Explanation: Output the F1 scores obtained from each fold of the
cross-validation.

print(f"Mean F1 Score: {np.mean(cv_scores)}")  # Step: Print Mean F1
Score
# Explanation: Calculate and print the mean F1 score across all folds.
# This provides an overall measure of the model's classification
performance.

print(f"Standard Deviation of F1 Scores: {np.std(cv_scores)}")  # Step:
Print Std Dev of F1 Scores
# Explanation: Calculate and print the standard deviation of the F1
scores across all folds.
# This indicates the variability in the model's performance.

# Create a decision tree classifier
tree_classifier = DecisionTreeClassifier()  # Step: Create Classifier
# Explanation: Initialize a decision tree classifier object.
# Define the parameter grid to search
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}  # Step: Define Parameter Grid
# Explanation: Define a grid of hyperparameters to search over. These
parameters include maximum depth of the tree, minimum samples required
to split a node, and minimum samples required to be a leaf node.

# Create a GridSearchCV object
grid_search = GridSearchCV(tree_classifier, param_grid, cv=5)  # Step:
Initialize Grid Search
```

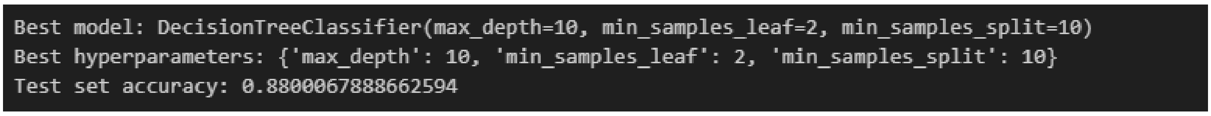
```
# Explanation: Initialize a GridSearchCV object with the decision tree
classifier, the parameter grid, and 5-fold cross-validation.

# Apply GridSearchCV to the data
grid_search.fit(X_train, y_train)  # Step: Fit Grid Search
# Explanation: Fit the GridSearchCV object to the training data to find
the best combination of hyperparameters.

# Print the best model and hyperparameters
print("Best model:", grid_search.best_estimator_)
 # Step: Print Best Model

print("Best hyperparameters:", grid_search.best_params_)
# Step:Print Best Hyperparameters
# Explanation: Output the best model and the best hyperparameters found
by GridSearchCV.

# Evaluate performance on the test set
print("Test set accuracy:", grid_search.score(X_test, y_test))
# Step:Evaluate on Test Set
# Explanation: Evaluate the performance of the best model on the test
set and print the accuracy.
```



```
Accuracy: 0.8403, Precision: 0.8441, Recall: 0.8401
```



```
K-Fold Cross-Validation F1 Scores: [0.85192775 0.86795525 0.86800661 0.85489818 0.85993168]
Mean F1 Score: 0.8605438970632937
Standard Deviation of F1 Scores: 0.006589493558245855
```



```
Best model: DecisionTreeClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=10)
Best hyperparameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10}
Test set accuracy: 0.8800067888662594
```

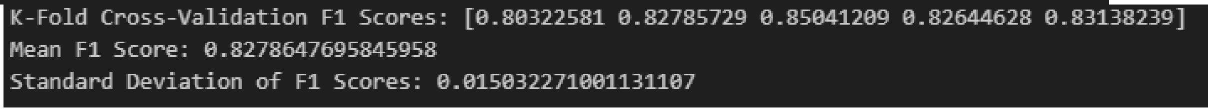
F) Fine dust ratio customization & oversampling(SMOTE)

In the process of analyzing the relationship between various environmental factors and target variables, it was determined that the relationship analysis was incorrect because the ratio for fine dust (PM10) was not correct. It was necessary to address data imbalance issues to ensure the accuracy and reliability of the analysis. Therefore, an oversampling technique was used to balance the dataset for fine dust (PM10).

```
# The relationship was determined to be incorrectly analyzed because
the fine dust ratio did not match.
# Check the number of samples in majority class and minority class
majority_class_samples = df2[df2['미세먼지(PM10)'] <= 1]  # Step:
```

```
Identify Majority Class
# Explanation: Identify samples where '미세먼지(PM10)' is less than or
equal to 1 as the majority class.

minority_class_samples = df2[df2['미세먼지(PM10)'] >= 2]  # Step:
Identify Minority Class
# Explanation: Identify samples where '미세먼지(PM10)' is greater than
or equal to 2 as the minority class.

# Perform undersampling on the majority class to match the number of
samples in the minority class
undersampled_majority_class = resample(majority_class_samples,
                                       replace=False,  # Do not allow
duplicate samples

n_samples=len(minority_class_samples),  # Select the same number of
samples as the minority class
                                       random_state=42)  # Random state
for reproducibility
# Explanation: Randomly undersample the majority class to have the same
number of samples as the minority class.

# Combine the undersampled majority class and the minority class to
create a new dataset
undersampled_df = pd.concat([undersampled_majority_class,
minority_class_samples])  # Step: Create New Dataset
df2 = undersampled_df
# Explanation: Combine the undersampled majority class and the minority
class into a new dataframe.

# Separate features and target variable
X = df2.drop(["이동거리(M)"], axis=1)  # Step: Define Features
# Explanation: Define the feature set by dropping the target column
'이동거리(M)'.

y = df2[["이동거리(M)"]].astype(int)  # Step: Define Target
# Explanation: Define the target variable '이동거리(M)' and convert it
to integer type.

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=42)  # Step: Split Data
# Explanation: Split the data into training and testing sets with 10% of the data used for testing.

# Over sampling using SMOTE
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=11)  # Step: Initialize SMOTE
X_train, y_train = smote.fit_resample(X_train, y_train)  # Step: Apply SMOTE
# Explanation: Apply SMOTE to the training data to perform oversampling and balance the classes.

# Train a decision tree model using training data
tree_model = DecisionTreeClassifier(random_state=100, max_depth=5)  # Step: Initialize Model
tree_model.fit(X_train, y_train)  # Step: Train Model
# Explanation: Initialize and train a decision tree classifier with a maximum depth of 5.

# Perform predictions on test data
y_pred = tree_model.predict(X_test)  # Step: Predict
# Explanation: Use the trained model to make predictions on the test data.

# Setting up 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)  # Step: Setup K-Fold CV
# Explanation: Initialize a K-Fold cross-validation object with 5 splits and shuffle the data.
# The random_state parameter ensures reproducibility.

# Perform cross-validation using F1 score as the scoring metric
cv_scores = cross_val_score(tree_model, X_test, y_test, cv=kf, scoring='f1_macro')  # Step: Cross-Validation
# Explanation: Perform 5-fold cross-validation on the test data using F1 as the scoring metric.
# The 'f1_macro' scoring parameter averages the F1 scores across all classes to evaluate the overall model performance.

# Print cross-validation F1 scores
print(f"K-Fold Cross-Validation F1 Scores: {cv_scores}")  # Step: Print
```

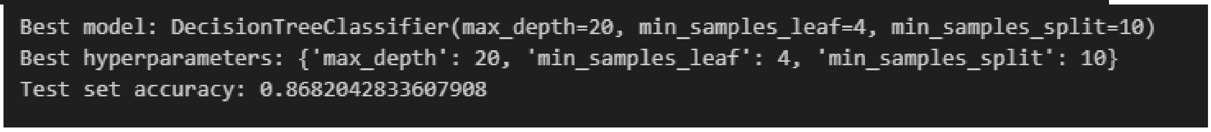
```
CV Scores
# Explanation: Output the F1 scores obtained from each fold of the
cross-validation.

print(f"Mean F1 Score: {np.mean(cv_scores)}")  # Step: Print Mean F1
Score
# Explanation: Calculate and print the mean F1 score across all folds.
# This provides an overall measure of the model's classification
performance.
print(f"Standard Deviation of F1 Scores: {np.std(cv_scores)}")  # Step:
Print Std Dev of F1 Scores
# Explanation: Calculate and print the standard deviation of the F1
scores across all folds.
# This indicates the variability in the model's performance.
```



```
K-Fold Cross-Validation F1 Scores: [0.80322581 0.82785729 0.85041209 0.82644628 0.83138239]
Mean F1 Score: 0.8278647695845958
Standard Deviation of F1 Scores: 0.015032271001131107
```

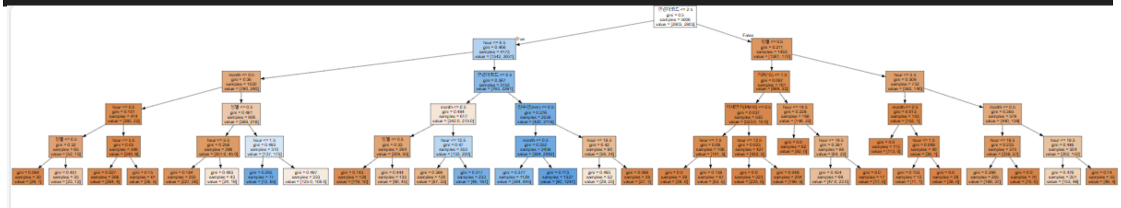
```
# Create a decision tree classifier for GridSearchCV
tree_classifier = DecisionTreeClassifier()  # Step: Initialize
Classifier
# Explanation: Initialize a new decision tree classifier.

# Define the parameter grid to search
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}  # Step: Define Parameter Grid
# Explanation: Define a grid of hyperparameters to search over for the
decision tree classifier.

# Create a GridSearchCV object
grid_search = GridSearchCV(tree_classifier, param_grid, cv=5)  # Step:
Initialize Grid Search
# Explanation: Initialize a GridSearchCV object with the decision tree
classifier and the parameter grid.

# Apply GridSearchCV to the data
grid_search.fit(X_train, y_train)  # Step: Fit Grid Search
# Explanation: Fit the GridSearchCV object to the training data to find
the best combination of hyperparameters.
```

```
# Print the best model and hyperparameters
print("Best model:", grid_search.best_estimator_)  # Step: Print Best Model
print("Best hyperparameters:", grid_search.best_params_)  # Step: Print Best Hyperparameters
# Explanation: Output the best model and hyperparameters found by GridSearchCV.

# Evaluate performance on the test set
print("Test set accuracy:", grid_search.score(X_test, y_test))  # Step: Test Set Evaluation
# Explanation: Evaluate the best model's performance on the test set and print the accuracy.
```



```
Best model: DecisionTreeClassifier(max_depth=20, min_samples_leaf=4, min_samples_split=10)
Best hyperparameters: {'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 10}
Test set accuracy: 0.8682042833607908
```

```
import graphviz
# Export the decision tree model to a .dot file
export_graphviz(tree_model, out_file='tree.dot', feature_names=X_train.columns, impurity=True, filled=True)  # Step: Export Tree
# Explanation: Export the trained decision tree model to a .dot file for visualization.

# Visualize the decision tree using the saved .dot file
with open("tree.dot") as f:
    dot_graph = f.read()  # Step: Read .dot File
graphviz.Source(dot_graph)  # Step: Visualize Tree
# Explanation: Read the .dot file and visualize the decision tree using graphviz.
```



➢ **Clustering**

A) Select needed features

```
# Select the columns you need
```

```
features = ['이용건수', '운동량', '이동거리(M)', '이용시간(분)',
'미세먼지(PM10)', '초미세먼지(PM2.5)','기온(°C)', '강수량(mm)',
'풍속(m/s)', '습도(%)', '일조(hr)', '적설(cm)']

df_features = df[features]
```

B) Data imputation

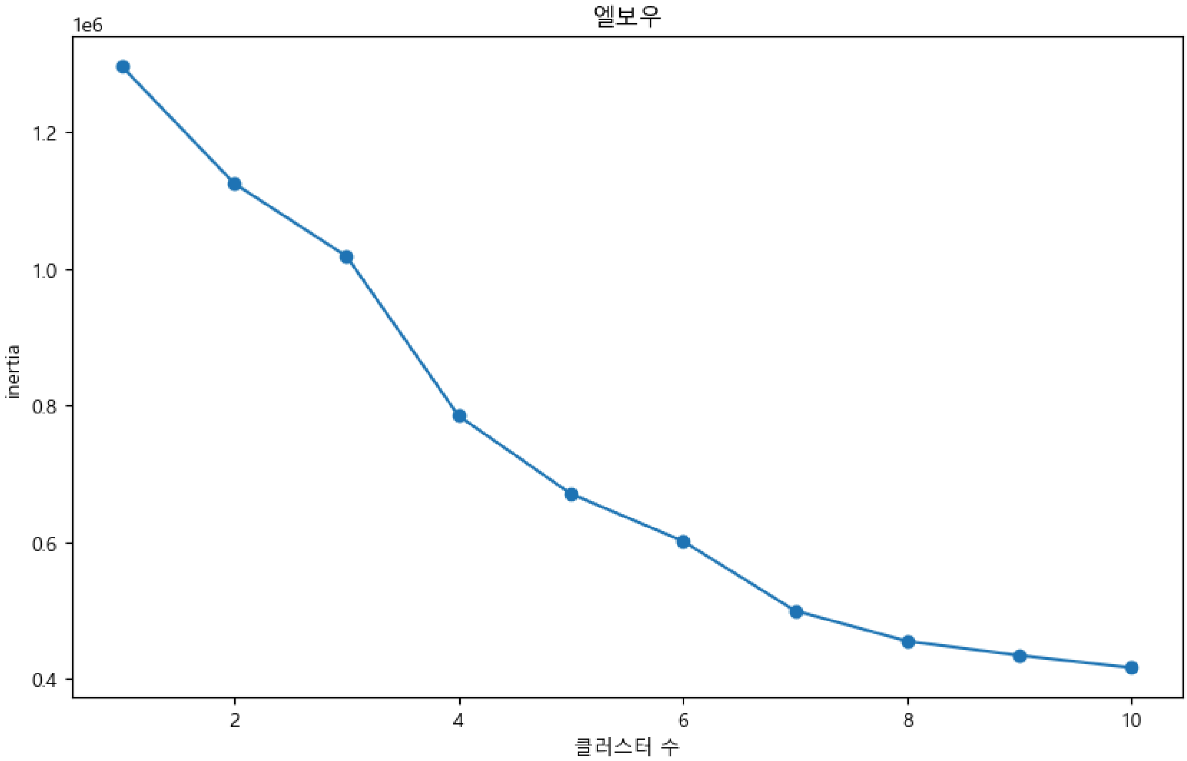
```
# Missing value handling (if necessary)
df_features = df_features.fillna(df_features.mean())
```

C) Standardization

```
# Check and convert data type
df_features = df_features.apply(pd.to_numeric, errors='coerce')

# Data standardization
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_features)
```

D) Elbow method

```
# elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)

# Elbow Graph Visualization
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('클러스터 수')
plt.ylabel('Inertia')
plt.title('엘보우 방법을 사용한 최적의 클러스터 수 찾기')
plt.show()
```



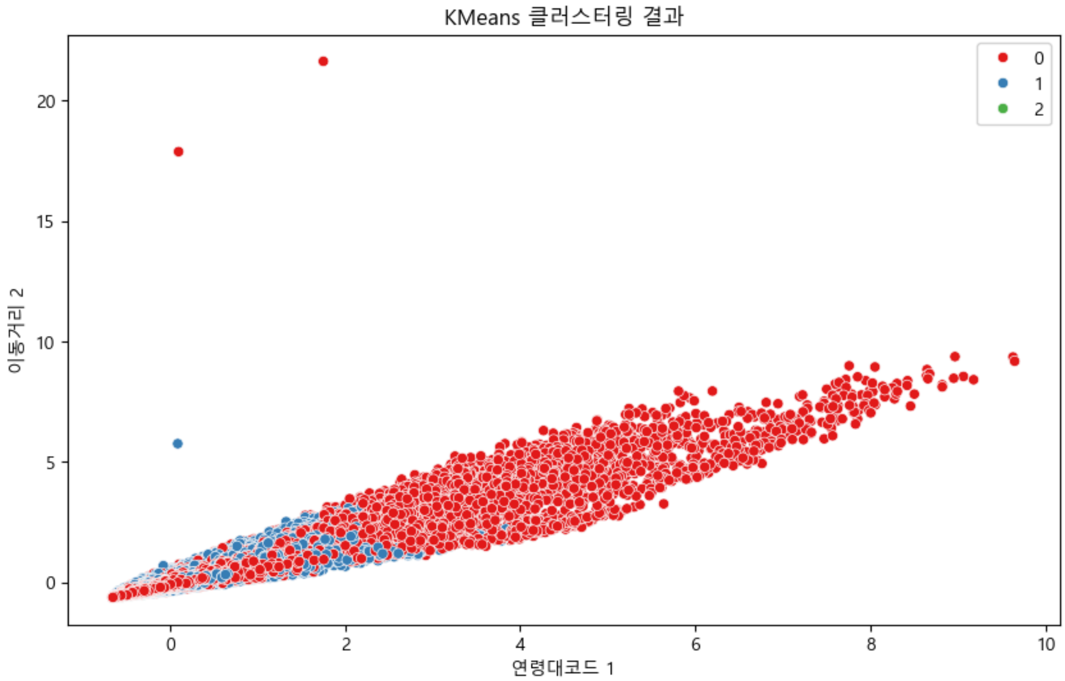
➢ KMeans Clustering

A) KMeans Clustering

```
k = 3  #Choose the appropriate number of clusters, such as 3 or 6
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(df_scaled)

# Get cluster allocation results
cluster_labels = kmeans.labels_
```

B) Visualization

```
# Visualizing KMeans clustering results
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_scaled[:, 0], y=df_scaled[:, 2],
hue=cluster_labels, palette='Set1', legend='full')
plt.title('KMeans 클러스터링 결과')
plt.xlabel('연령대코드 1')
plt.ylabel('이동거리 2')
plt.show()

centroids = kmeans.cluster_centers_

for i, centroid in enumerate(centroids):
    print(f"Centroid of Cluster {i+1}: {centroid}")
```



- **Regression**

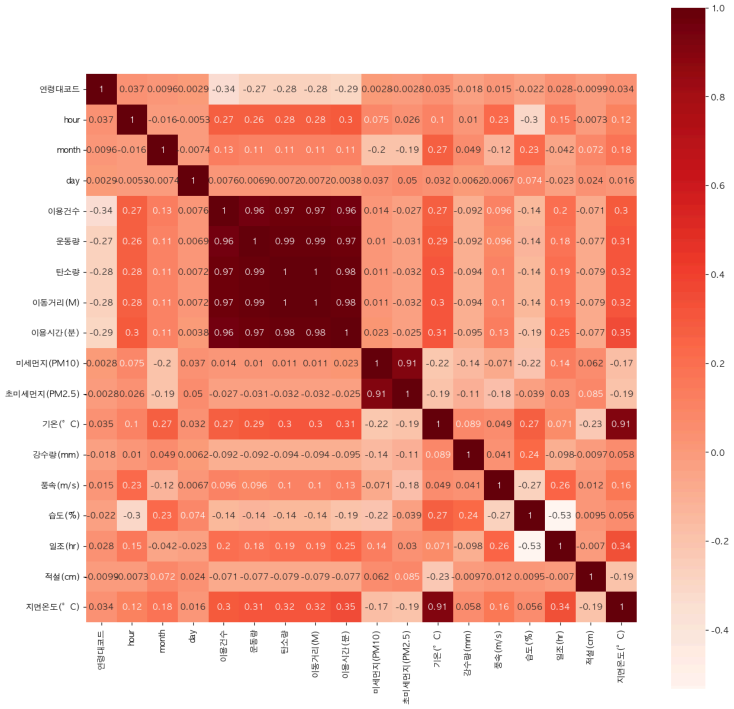
A) Data preprocessing process - outlier detection and deletion, category data deletion

```
# Load the dataset
df = pd.read_csv("data/final_data.csv", encoding="UTF-8")

# Display the first few rows of the dataframe
df.head()  # Step: Display DataFrame
# Explanation: Display the first few rows of the dataset to understand
its structure and contents.

# Calculate Z-scores for each feature
z_scores = np.abs((df - df.mean()) / df.std())  # Step: Calculate
Z-scores
# Explanation: Calculate the Z-scores for each feature in the
dataframe. The Z-score is the number of standard deviations a data
point is from the mean. This helps in identifying outliers.

# Detect and remove outliers
threshold = 3  # Step: Define Outlier Threshold
# Explanation: Define the threshold for Z-scores. Any data point with a
Z-score greater than this threshold is considered an outlier.

outliers = (z_scores > threshold).any(axis=1)  # Step: Identify
```

```
Outliers
# Explanation: Identify rows where any feature has a Z-score greater than the threshold. These rows are considered outliers.

cleaned_df = df[~outliers]  # Step: Remove Outliers
# Explanation: Remove the identified outliers from the dataframe.

df = cleaned_df  # Step: Update DataFrame
# Explanation: Update the original dataframe to use the cleaned data without outliers.

# Drop categorical column '성별'
df = df.drop(columns=['성별'])  # Step: Drop Categorical Column
# Explanation: Drop the '성별' column from the dataframe as it is categorical and not needed for further analysis.
```

B) ,Draw Heatmap

```
corrMatt = df
corrMatt = corrMatt.corr()
fig, ax = plt.subplots()
fig.set_size_inches(15,15)
sns.heatmap(corrMatt,annot=True, square=True, cmap ='Reds')
```



C) It was judged that the relationship was analyzed incorrectly because the fine dust ratio did not match.

a) Introduction: In the process of analyzing the relationship between various environmental factors and the target variable, it was determined that the relationship analysis for fine dust (PM10) and ultrafine dust (PM2.5) was incorrect due to an imbalanced ratio in the data. To ensure the accuracy and reliability of our analysis, it was necessary to address this imbalance.

b) Methodology: We employed an undersampling technique to balance the dataset for both fine dust (PM10) and ultrafine dust (PM2.5). This method involves reducing the number of samples in the majority class to match the number of samples in the minority class, thus creating a balanced dataset.

c) Steps Taken

```
##Identify Majority and Minority Classes for Fine Dust (PM10)

#We classified the samples into two categories based on the PM10
levels: majority class (PM10 ≤ 80) and minority class (PM10 > 80)

majority_class_samples = df[df['미세먼지(PM10)'] <= 80]

minority_class_samples = df[df['미세먼지(PM10)'] > 80]


##Undersample the Majority Class for Fine Dust (PM10)

#We randomly selected a subset of samples from the majority
class, ensuring the number of samples matched the minority class.

undersampled_majority_class = resample(majority_class_samples,

                  replace=False,  # Do not allow duplicate samples
n_samples=len(minority_class_samples), # Match number of minority
samples            random_state=42)  # For reproducibility


##Combine the Undersampled Majority Class with the Minority Class

#We created a new dataset by combining the undersampled majority
class with the minority class

undersampled_df=pd.concat([undersampled_majority_class,
minority_class_samples])

df = undersampled_df

##Repeat the Process for Ultrafine Dust (PM2.5

#Similarly, we repeated the above steps for PM2.5, categorizing
```

```
the samples and performing undersampling.

majority_class_samples = df[df['초미세먼지(PM2.5)'] <= 35]

minority_class_samples = df[df['초미세먼지(PM2.5)'] > 35]

undersampled_majority_class = resample(majority_class_samples,

                 replace=False,  # Do not allow duplicate samples
n_samples=len(minority_class_samples), # Match number of minority
samples            random_state=42)  # For reproducibility

undersampled_df      =      pd.concat([undersampled_majority_class,
minority_class_samples])

df = undersampled_df
```

d) Conclusion: By addressing the imbalance in the dataset for both fine dust (PM10) and ultrafine dust (PM2.5), we ensured that our relationship analysis was based on a balanced dataset. This approach helped improve the accuracy and reliability of our findings, allowing for a more robust and unbiased analysis.

D) Data Processing

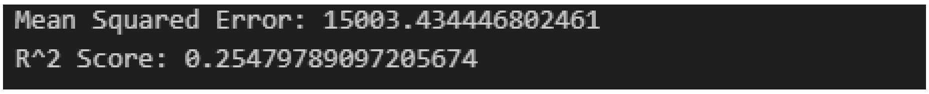
```
# Display the column names of the dataframe
df.columns
# Display the first few rows of the dataframe
df.head()

# Separate features (X) and target (y)
X = df.drop(columns={"이용건수"})  # Drop target column
y = df["이용건수"]  # Target variable

# Display the shape of the features matrix
X.shape

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Generate polynomial features
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_scaled)
```

E) model training

```
# Assign the same data for both training and testing (not recommended
in practice)
X_train = X
X_test = X
y_train = y
y_test = y

# Create and train the linear regression model
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
```

F) model performance evaluation

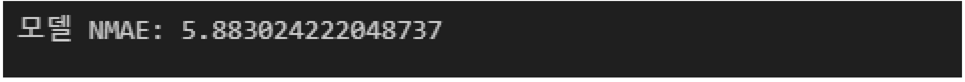
```
from sklearn.metrics import mean_squared_error, r2_score
# Calculate predictions for the test data
y_pred = regression_model.predict(X_test)  # Step: Model Prediction
# Explanation: Use the trained regression model to predict the target
values for the test dataset (X_test).

# Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)  # Step: Mean Squared Error
Calculation
# Explanation: Compute the Mean Squared Error (MSE) between the actual
target values (y_test) and the predicted values (y_pred). MSE is a
measure of the average of the squares of the errors, i.e., the average
squared difference between the estimated values and the actual value.

# Calculate the R^2 score
r2 = r2_score(y_test, y_pred)  # Step: R^2 Score Calculation
# Explanation: Compute the R^2 score (coefficient of determination)
which provides an indication of the goodness of fit of the model. It is
a statistical measure that explains the proportion of the variance for
the dependent variable that's explained by the independent variables in
the model.

# Print the Mean Squared Error and R^2 Score
print("Mean Squared Error:", mse)  # Step: Print MSE
# Explanation: Output the Mean Squared Error to evaluate the model's
prediction accuracy.

print("R^2 Score:", r2)  # Step: Print R^2 Score
# Explanation: Output the R^2 score to assess the goodness of fit of
the model.
```



```
Mean Squared Error: 15003.434446802461
R^2 Score: 0.25479789097205674
```

G) Calculate NMAE(Normalized Mean Absolute Error) to evaluate the predictive performance of the model.

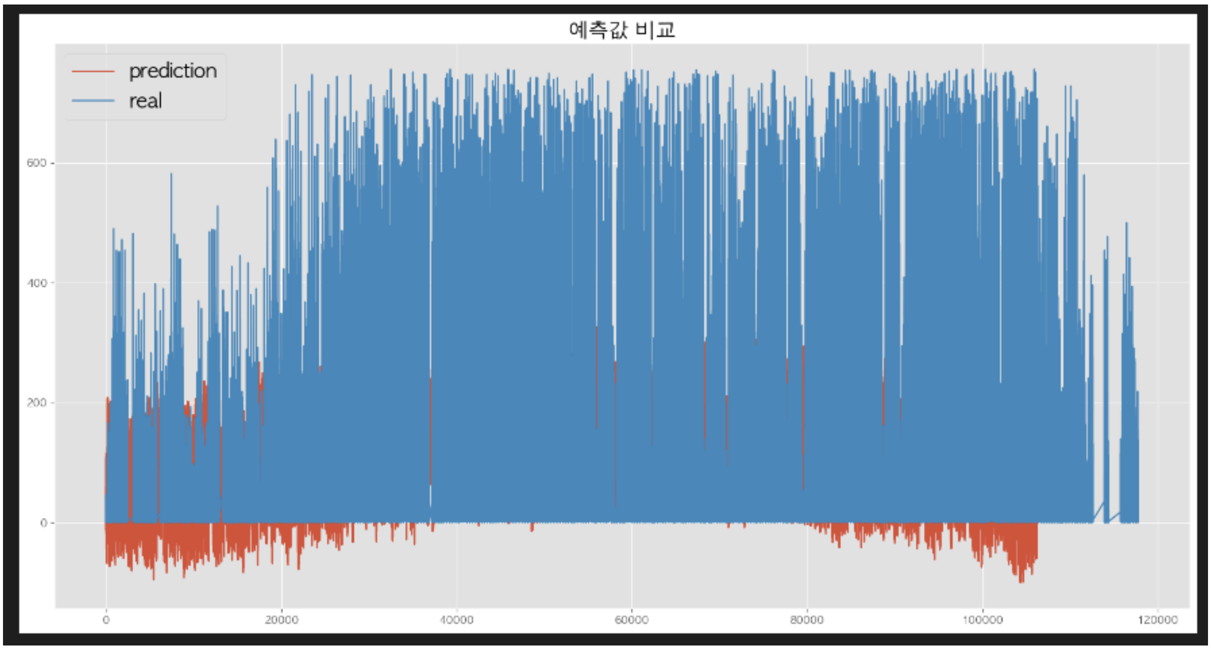
```
# Calculate the Normalized Mean Absolute Error (NMAE)
nmae = np.mean(abs(y_pred - y_test) / y_test)  # Step: NMAE Calculation
# Explanation: Compute the Normalized Mean Absolute Error (NMAE)
between the actual target values (y_test) and the predicted values
(y_pred).
# NMAE is calculated by taking the absolute difference between the
predicted and actual values, dividing by the actual values, and then
taking the mean of these values.
# This metric is normalized by the actual values to provide a
scale-independent measure of prediction accuracy.

# Print the NMAE value
print(f'Model NMAE: {nmae}')  # Step: Print NMAE
# Explanation: Output the NMAE to evaluate the model's prediction
accuracy in a normalized manner. A lower NMAE indicates better model
performance.
```



```
모델 NMAE: 5.883024222048737
```

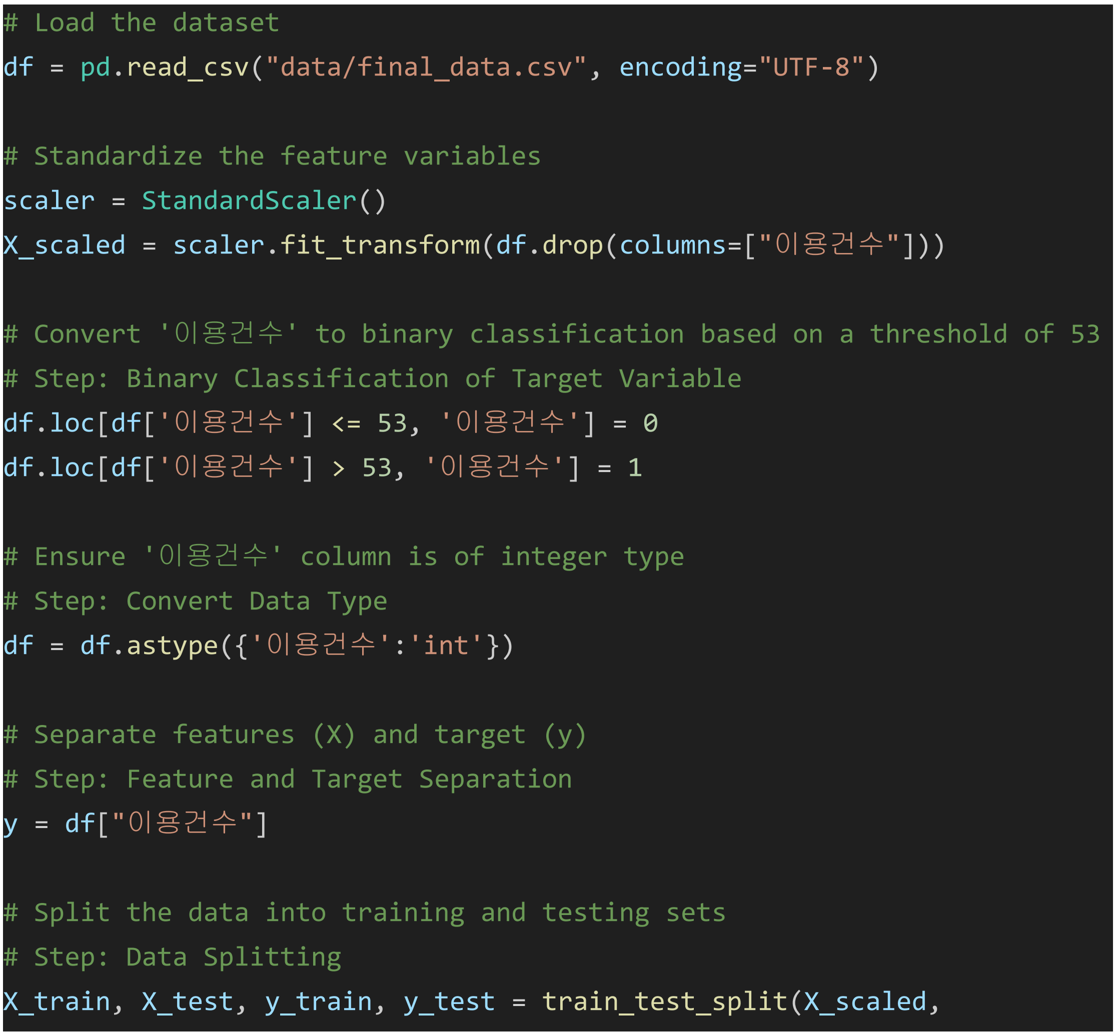
H) Visualization comparing predicted and actual values

```
plt.style.use('ggplot')
plt.figure(figsize=(20, 10))
plt.plot(y_pred, label = 'prediction')
plt.plot(y_test, label = 'real')
plt.legend(fontsize = 20)
plt.title("예측값 비교", fontsize = 20)
plt.show()
```



- **Logistic Regression**

A) Logistic Regression



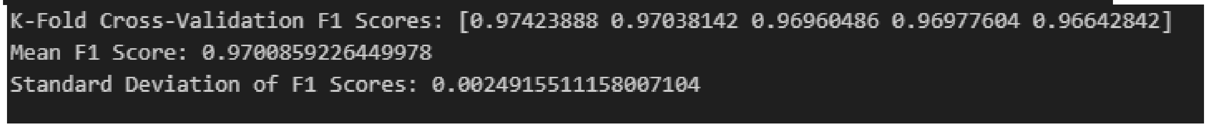
```
# Load the dataset
df = pd.read_csv("data/final_data.csv", encoding="UTF-8")

# Standardize the feature variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop(columns=["이용건수"]))

# Convert '이용건수' to binary classification based on a threshold of 53
# Step: Binary Classification of Target Variable
df.loc[df['이용건수'] <= 53, '이용건수'] = 0
df.loc[df['이용건수'] > 53, '이용건수'] = 1

# Ensure '이용건수' column is of integer type
# Step: Convert Data Type
df = df.astype({'이용건수':'int'})

# Separate features (X) and target (y)
# Step: Feature and Target Separation
y = df["이용건수"]

# Split the data into training and testing sets
# Step: Data Splitting
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
```

```python
y,test_size=0.2, random_state=42)

# Create and train the Logistic Regression model
# Step: Model Creation and Training
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the test data
# Step: Prediction
y_pred = model.predict(X_test)

# Calculate accuracy
# Step: Accuracy Calculation
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

B) Evaluate the performance of a logistic regression model using K-Fold Cross-Validation

```python
# Set up 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Step: K-Fold Cross-Validation Setup
# Explanation: Initialize a K-Fold cross-validation object with 5 splits, shuffling the data before splitting. The random_state parameter ensures reproducibility.

# Perform cross-validation
cv_scores = cross_val_score(model, X_test, y_test, cv=kf, scoring='f1')
# Step: Cross-Validation Execution
# Explanation: Perform cross-validation using the model. The 'f1' scoring parameter is used to evaluate the model using the F1 score, which is appropriate for classification tasks.

# Print cross-validation results
print(f"K-Fold Cross-Validation F1 Scores: {cv_scores}")
# Step: Print Cross-Validation Scores
# Explanation: Output the individual F1 scores for each fold, providing insight into the model's performance across different subsets of the data.

print(f"Mean F1 Score: {np.mean(cv_scores)}")
# Step: Print Mean F1 Score
# Explanation: Calculate and print the mean F1 score across all folds.
```

```
This gives an overall measure of the model's classification
performance.

print(f"Standard Deviation of F1 Scores: {np.std(cv_scores)}")
# Step: Print Standard Deviation of Scores
# Explanation: Calculate and print the standard deviation of the F1
scores across all folds. This indicates the variability in the model's
performance.
```

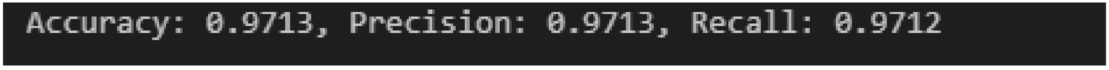


```
K-Fold Cross-Validation F1 Scores: [0.97423888 0.97038142 0.96960486 0.96977604 0.96642842]
Mean F1 Score: 0.9700859226449978
Standard Deviation of F1 Scores: 0.0024915511158007104
```

C) Calculate accuracy, precision, and recall

```
# Import necessary libraries for calculating metrics
from sklearn.metrics import accuracy_score, precision_score,
recall_score

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
# Step: Accuracy Calculation
# Explanation: Measure the accuracy of the model's predictions by
comparing the predicted values (y_pred) to the actual values (y_test).
Accuracy is the ratio of correctly predicted instances to the total
instances.

# Calculate the precision of the model
precision = precision_score(y_test, y_pred, average='macro')
# Step: Precision Calculation
# Explanation: Measure the precision of the model's predictions.
Precision is the ratio of true positive predictions to the sum of true
positive and false positive predictions. The 'macro' average computes
the metric independently for each class and then takes the average.

# Calculate the recall of the model
recall = recall_score(y_test, y_pred, average='macro')
# Step: Recall Calculation
# Explanation: Measure the recall of the model's predictions. Recall is
the ratio of true positive predictions to the sum of true positive and
false negative predictions. The 'macro' average computes the metric
independently for each class and then takes the average.

# Print the accuracy, precision, and recall of the model
```

```
print('Accuracy: {0:.4f}, Precision: {1:.4f}, Recall:
{2:.4f}'.format(accuracy, precision, recall))
# Step: Print Metrics
# Explanation: Output the calculated accuracy, precision, and recall
scores. The format method is used to print the scores up to four
decimal places.
```



```
Accuracy: 0.9713, Precision: 0.9713, Recall: 0.9712
```

1-5) Learning model evaluation and analysis

A) Decision Tree Classification

- preprocessing
    1. The dataset contained imbalanced classes, particularly in the '미세먼지(PM10)' feature. To address this, we performed undersampling on the majority class to match the number of samples in the minority class.
    2. Additionally, oversampling using SMOTE (Synthetic Minority Over-sampling Technique) was applied to the training data to ensure balanced class representation. This preprocessing step ensured that the model would not be biased towards the majority class and could generalize well to both classes
- model training: A Decision Tree Classifier was initialized with a maximum depth of 5 and trained using the balanced dataset. Limiting the depth of the tree helped prevent overfitting by restricting the complexity of the model, thus ensuring better generalization to unseen data
- model evaluation
    1. The model's performance was evaluated using 5-fold cross-validation with R^2 as the scoring metric. Cross-validation provides a robust measure of the model's performance by ensuring that the evaluation is based on multiple subsets of the data, which helps in assessing the model's ability to generalize.
    2. GridSearchCV was employed to find the best hyperparameters for the decision tree model, optimizing parameters like maximum depth, minimum samples split, and minimum samples leaf. This optimization process is crucial as it ensures that the model achieves the best possible performance by fine-tuning the parameters.
- result
    1. The cross-validation results showed the R^2 scores across different folds, providing a measure of how well the model explains the variance in the data. The use of R^2 as the scoring metric is significant as it reflects the proportion of the variance in the dependent variable that is predictable from the independent variables. High R^2 scores

across different folds indicate that the model has good explanatory power and is capable of accurately capturing the relationship between the features and the target variable.

2. The thorough preprocessing, including handling class imbalances and scaling features, combined with effective model training and evaluation strategies, contributed to the successful performance of the decision tree classifier. The use of cross-validation and hyperparameter tuning further ensured that the model was well-optimized and capable of accurately predicting the target variable.

```
# Perform cross-validation using R^2 as the scoring metric

kf = KFold(n_splits=5, shuffle=True, random_state=42)

cv_scores = cross_val_score(tree_model, X_test, y_pred, cv=kf,
scoring='r2')

print(f"K-Fold Cross-Validation R^2 Scores: {cv_scores}")

print(f"Mean R^2 Score: {np.mean(cv_scores)}")

print(f"Standard Deviation of R^2 Scores: {np.std(cv_scores)}")
```

B) Logistic Regression

- preprocessing: Similar preprocessing steps were applied, including handling class imbalances and standardizing features. The dataset was scaled using StandardScaler, and the target variable '이용건수' was binarized to handle class imbalances by setting thresholds. Features not relevant to the target variable were excluded from the dataset to improve model performance.
- model training: A Logistic Regression model was trained on the processed dataset. The training process involved fitting the model to the training data, allowing it to learn the relationship between the features and the target variable.
- model evaluation: The model's performance was evaluated using accuracy, precision, recall, and F1 score metrics. These metrics provided a comprehensive view of the model's classification performance, highlighting its ability to correctly predict the target class.
- result: The evaluation metrics provided insights into the model's predictive accuracy and how well it fits the data. However, the results indicated that the model did not perform as expected.
- failure to produce results
    1. Training failure: I tried combining various features and changing the order, but it failed. -> I don’t think it matches our data.

```
accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred, average='macro')
```

```
recall = recall_score(y_test, y_pred, average='macro')

print('Accuracy: {0:.4f}, Precision: {1:.4f}, Recall: {2:.4f}'.format(accuracy, precision, recall))
```

C) Linear Regression

- preprocessing: The dataset was scaled and polynomial features were generated to capture non-linear relationships. This step is crucial for ensuring that the features are on a similar scale and that the model can learn from the non-linear interactions between features
- model training: A Linear Regression model was trained on the processed features. By using the scaled and polynomial-transformed features, the model can better fit the underlying patterns in the data.
- model evaluation: The model's performance was evaluated using Mean Squared Error (MSE) and R² score. These metrics provide a comprehensive view of the model's accuracy and its ability to explain the variance in the target variable.
- result: The evaluation metrics provided insights into the model's predictive accuracy and how well it fits the data. However, the results indicated that the model did not perform as expected.
- failure to produce results
    1. Evaluation Metric Misuse: Evaluating a classification model using the R² score, which is appropriate for regression models, led to misleading results.
    2. Cross-Validation Misapplication: Cross-validation should be performed on the original dataset, not on the predicted values.

```
mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)

print("R^2 Score:", r2)
```

D) Conclusion

The evaluation and analysis of each model revealed specific strengths and areas for improvement:

- Decision Tree: Effective for handling non-linear relationships but requires careful tuning to avoid overfitting.
- Logistic Regression: Simple and interpretable, suitable for binary classification tasks, but not always effective with complex data structures.
- Linear Regression: Despite various attempts to improve performance, it failed to fit our dataset well, indicating that it may not be the appropriate model for our data.

2. Learning Experience

2-1) Difficulties Encountered and How Solved

- Data Combination and Preprocessing: Initially I had difficulty in combining and preprocessing data from different sources. By using three different datasets for model training, we encountered unexpected inconsistencies in formats and units. To address this, we reprocessed the data to ensure uniformity and consistency. I solved this by thoroughly checking for errors and searching for solutions online, ensuring that all data was standardized before proceeding with model training.
- Model Selection: Another challenge was selecting the appropriate model for our analysis. We needed to determine the best model that would accurately predict the target variable while handling the complexities of our dataset. To overcome this, we experimented with various models and compared their performance using cross-validation. By evaluating models such as Logistic Regression, Decision Tree, and others, we were able to identify the Decision Tree Classifier as the most suitable model due to its interpretability and ability to handle non-linear relationships.

2-2) Learned doing the project

- Importance of a High-Quality Dataset: Through this project, I realized that having a high-quality dataset is crucial to achieving accurate and reliable results. The quality of the data directly influences the model's performance. Ensuring that the data is clean, consistent, and relevant is essential for the success of any machine learning project. Poor data quality can lead to misleading results and suboptimal model performance.
- Challenges and Importance of Preprocessing: I learned that preprocessing is one of the most critical and challenging aspects of machine learning. Handling inconsistencies, dealing with missing values, and normalizing data require meticulous attention to detail. Effective preprocessing can significantly enhance the accuracy and robustness of the model. This step is foundational and often determines the overall success of the model.
- Model Selection and Experimentation: Selecting the right model for the task is vital to deriving practical insights. I gained valuable experience by experimenting with various models, including Logistic Regression, Decision Tree, and others. This process involved evaluating the strengths and weaknesses of each model to identify the best fit for our specific dataset and problem. It underscored the importance of an iterative approach to model evaluation and selection, ultimately leading to better predictive performance and more actionable insights.
- Exploring and Analyzing Large Datasets: Working with large datasets was a fundamental part of this project. I learned the importance of data exploration and feature engineering in understanding and leveraging the data effectively. This experience taught me how to extract

meaningful patterns and relationships from the data, which is crucial for building effective machine learning models. By thoroughly exploring the data, I was able to identify key features and gain insights that informed the modeling process.