Machine Learning Spring 2025

Project 1 - Temperature Prediction Team: ST_ML2025_2

[80]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import math
```

[81]:
```python
# Load the given datasets(located in the input folder)
train_df = pd.read_csv("./input/train_dataset.csv")
test_df = pd.read_csv("./input/test_dataset.csv")
station_info_df = pd.read_csv("./input/station_info.csv")
sample_submission_df = pd.read_csv("./input/submission_sample.csv")
```

Dataset 분석

주어진 데이터셋의 구성은 다음과 같다.

- `train_dataset.csv`: 동두천, 서울, 강화, 인천, 이천, 양평 관측소의 2019-2024년 데이터셋
- `test_dataset.csv`: 파주, 수원 관측소 데이터셋
- `station_info.csv`: 기상 관측소별 정보

[82]:
```python
train_df.sample(10)
```

[82]:

	id	station	station_name	date	cloud_cover_0	cloud_cover_1	\
8159	12538	201	강화	05-13	0.0	0.0	
2465	4653	108	서울	10-06	1.0	1.0	
631	631	98	동두천	09-23	1.0	0.0	
175	175	98	동두천	06-25	0.0	0.0	
1017	1017	98	동두천	10-14	5.0	5.0	
12687	17066	203	이천	10-13	3.0	2.0	
2858	5046	108	서울	11-02	0.0	1.0	
3250	5438	108	서울	11-29	0.0	9.0	
1265	1265	98	동두천	06-19	7.0	7.0	
9810	14189	202	양평	11-22	2.0	5.0	

	cloud_cover_10	cloud_cover_11	cloud_cover_12	cloud_cover_13	…	\
8159	0.0	0.0	0.0	4.0	…	
2465	0.0	0.0	4.0	1.0	…	
631	6.0	1.0	2.0	7.0	…	
175	0.0	0.0	0.0	0.0	…	
1017	3.0	6.0	6.0	4.0	…	
12687	0.0	0.0	0.0	0.0	…	
2858	0.0	0.0	0.0	0.0	…	
3250	8.0	0.0	0.0	0.0	…	
1265	8.0	9.0	9.0	9.0	…	
9810	6.0	2.0	3.0	2.0	…	

	wind_speed_23	wind_speed_3	wind_speed_4	wind_speed_5	wind_speed_6	\

```
8159          0.2          0.9          1.8          1.5          0.7
2465          1.7          3.8          2.7          3.4          3.8
631           1.6          0.8          1.7          1.5          1.4
175           0.9          0.8          0.2          0.1          0.1
1017          0.4          0.0          0.5          0.0          0.0
12687         0.8          0.4          0.2          0.2          0.0
2858          3.1          1.6          1.1          1.4          1.1
3250          2.1          2.1          1.4          1.6          1.7
1265          0.1          0.3          0.0          0.0          0.5
9810          3.4          2.6          2.5          3.7          2.6

       wind_speed_7  wind_speed_8  wind_speed_9  climatology_temp    target
8159            0.0           0.4           1.1         17.689286 -3.389286
2465            3.1           4.2           4.5         15.475000 -0.675000
631             1.9           2.2           1.0         18.755357  0.144643
175             0.4           0.4           0.4         22.943636  0.456364
1017            0.1           0.0           1.5         14.939286  2.360714
12687           0.0           0.0           0.5         14.733929 -2.033929
2858            2.5           1.7           1.8         11.596429 -6.496429
3250            2.1           1.7           1.3          0.182143  5.517857
1265            0.0           0.0           0.0         23.325000  0.575000
9810            2.3           1.5           1.9          5.185714 -4.385714

[10 rows x 342 columns]
```

[83]:
```python
train_df.describe()
```

[83]:
```
                 id       station  cloud_cover_0  cloud_cover_1  \
count  13132.000000  13132.000000   12945.000000   12920.000000
mean    9484.110493    153.980658       2.915798       3.022291
std     5311.954253     48.183220       3.646779       3.652165
min        0.000000     98.000000       0.000000       0.000000
25%     5470.750000    108.000000       0.000000       0.000000
50%     8753.500000    112.000000       0.000000       1.000000
75%    14227.250000    202.000000       6.000000       6.000000
max    17510.000000    203.000000      10.000000      10.000000

       cloud_cover_10  cloud_cover_11  cloud_cover_12  cloud_cover_13  \
count    12916.000000    12926.000000    12931.000000    12926.000000
mean         3.126742        3.092063        3.046400        3.035046
std          3.659422        3.589739        3.512063        3.483002
min          0.000000        0.000000        0.000000        0.000000
25%          0.000000        0.000000        0.000000        0.000000
50%          1.000000        1.000000        1.000000        1.000000
75%          7.000000        7.000000        6.000000        6.000000
max         10.000000       10.000000       10.000000       10.000000

       cloud_cover_14  cloud_cover_15  …  wind_speed_23  wind_speed_3  \
count    12919.000000    12938.000000  …   13122.000000  13120.000000
mean         2.992182        2.911115  …      -0.032754     -2.491814
std          3.462070        3.440298  …     123.464434    195.197628
```

```
min          0.000000       0.000000  ...   -9999.000000  -9999.000000
25%          0.000000       0.000000  ...       0.600000      0.400000
50%          1.000000       1.000000  ...       1.200000      1.000000
75%          6.000000       6.000000  ...       2.100000      1.900000
max         10.000000      10.000000  ...      10.700000     11.300000

       wind_speed_4  wind_speed_5  wind_speed_6  wind_speed_7  wind_speed_8  \
count  13116.000000  13112.000000  13119.000000  13120.000000  13125.000000
mean      -0.234637     -1.011882     -4.068389     -7.119787    -10.073272
std      123.489868    151.258218    230.949420    289.453706    337.895538
min    -9999.000000  -9999.000000  -9999.000000  -9999.000000  -9999.000000
25%        0.400000      0.400000      0.400000      0.400000      0.500000
50%        0.900000      0.900000      0.900000      0.900000      1.000000
75%        1.900000      1.800000      1.800000      1.800000      2.000000
max        9.500000     10.600000     12.400000     11.100000     11.300000

       wind_speed_9  climatology_temp        target
count  13127.000000      13132.000000  13132.000000
mean      -9.119029         12.658557      0.221979
std      326.432094         10.023504      2.960544
min    -9999.000000         -4.487500    -12.864286
25%        0.700000          3.292857     -1.643052
50%        1.300000         12.842857      0.157143
75%        2.200000         22.271429      2.045536
max        9.400000         28.455357     11.778571

[8 rows x 340 columns]
```

[84]:
```python
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13132 entries, 0 to 13131
Columns: 342 entries, id to target
dtypes: float64(338), int64(2), object(2)
memory usage: 34.3+ MB
```

Feature 분석

Feature 분류 및 설명

[85]:
```python
def search_time_based_feature_names(df):
    "각 시간별로 구성된 feature들(e.g. 'cloud_cover_0', 'cloud_cover_1' ...)에 대해␣
↪각각의 고유한 이름(e.g. 'cloud_cover')을 찾는다."

    time_based_features_pattern = re.compile(r"^(.*)_(\d{1,2})$")
    searched_time_based_feature_names = set()

    for column in df.columns:
        match = time_based_features_pattern.match(column)
        if match:
            feature_name, hour = match.groups()
            hour = int(hour)
```

```python
            searched_time_based_feature_names.add(feature_name)

    return searched_time_based_feature_names

time_based_feature_names = search_time_based_feature_names(train_df)

print(len(time_based_feature_names))
print(time_based_feature_names)
```

```
14
{'wind_direction', 'humidity', 'vapor_pressure', 'sea_level_pressure', 'snow_depth',
'cloud_cover', 'precipitation', 'sunshine_duration', 'dew_point', 'surface_temp',
'min_cloud_height', 'wind_speed', 'visibility', 'local_pressure'}
```

train dataset에 있는 feature를 특성에 따라 다음과 같은 카테고리별로 정리하였다.

1. Data 식별
    - `id`: 순서(identical)
    - `station`: 지상관측소 번호(98: "동두천", 201: "강화" 등)
    - `station_name`: 지상관측소 이름("동두천", "서울", "강화" 등)
    - `date`: 날짜(월-일 형식, 1월 29일은 "01-29"로 표기)
2. 구름 관련 feature
    - `cloud_cover_[0-23]`: 증하층운량(10분위, 0~10)
    - `min_cloud_height_[0-23]`: 시간별 최저운고(100m 단위)
3. 온도 관련 feature
    - `dew_point_[0-23]`: 시간별 이슬점 온도(°C)
    - `surface_temp_[0-23]`: 시간별 지면온도(°C)
    - `climatology_temp`: 해당 날짜의 평균 기온(°C) (7년 동안 평균)
4. 습도 관련 feature
    - `humidity_[0-23]`: 시간별 습도(%)
    - `vapor_pressure_[0-23]`: 시간별 증기압(hPa)
    - `percipitation_[0-23]`: 시간별 강수량(mm)
    - `snow_depth_[0-23]`: 시간별 적설(cm)
5. 기압 관련 feature
    - `local_pressure_[0-23]`: 시간별 현지기압(hPa)
    - `sea_level_pressure_[0-23]`: 시간별 해면기압(hPa)
6. 가시성 관련 feature
    - `visibility_[0-23]`: 시간별 시정(10m 단위)
    - `sunshine_duration_[0-23]`: 시간별 일조(hr)
7. 바람 관련 feature
    - `wind_speed_[0-23]`: 시간별 풍속(m/s)
    - `wind_direction_[0-23]`: 시간별 풍향(도 단위)
8. target feature
    - `target`: 시간별 기온(°C, 다음날 평균 기온(°C)에서 `climatology_temp`를 뺀 값)

결측치 패턴

1. -9999: 관측소 기계에서 감지한 결측치 또는 이상치
2. `NaN`:
    - `sunshine_duration`: 야간 시간대(0, 1, 2, 3, 4, 5, 22, 23시), 즉 해가 없는 시간대
    - `snow_depth`: 눈이 오지 않은 경우
    - `precipitation`: 비가 오지 않은 경우

결측치 식별

Time-based feature에 대한 결측치

시간대별 결측률 분석

[86]:
```
def analyze_time_based_features_missing_by_hour(ax, df, base_name):
    hour_missing = {}

    cols = [col for col in df.columns if col.startswith(base_name + "_")]

    for col in cols:
        try:
            hour = int(col.split("_")[-1])
            missing_ratio = df[col].isnull().mean()
            hour_missing[hour] = missing_ratio
        except ValueError:
            continue

    hours = sorted(hour_missing.keys())
    missing_values = [hour_missing[h] for h in hours]

    sns.lineplot(x=hours, y=missing_values, marker="o", ax=ax, color="steelblue")
    ax.set_title(f"{base_name}")
    ax.set_xlabel("Hour")
    ax.set_ylabel("Missing Ratio")
    ax.set_xticks(hours)
    ax.set_ylim(0, max(missing_values) + 0.0025)
    ax.grid(True, linestyle="--", alpha=0.5)
```

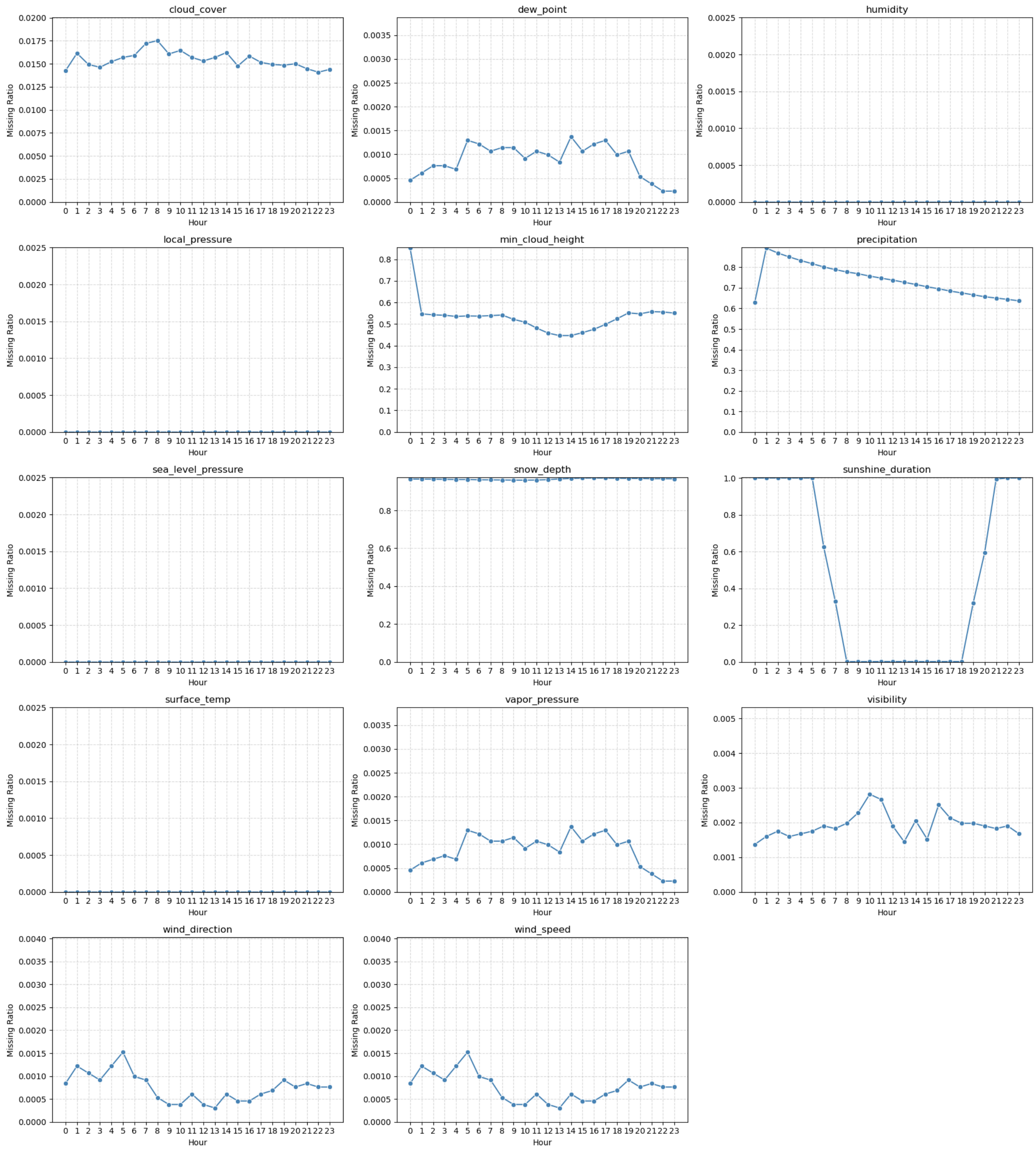
[87]:
```
num_features = len(time_based_feature_names)
n_cols = 3
n_rows = math.ceil(num_features / n_cols)

fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 6, n_rows * 4))
axes = axes.flatten()

for i, feature in enumerate(sorted(time_based_feature_names)):
    analyze_time_based_features_missing_by_hour(axes[i], train_df, feature)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



Time-based feature 종류별 평균 결측치 비율

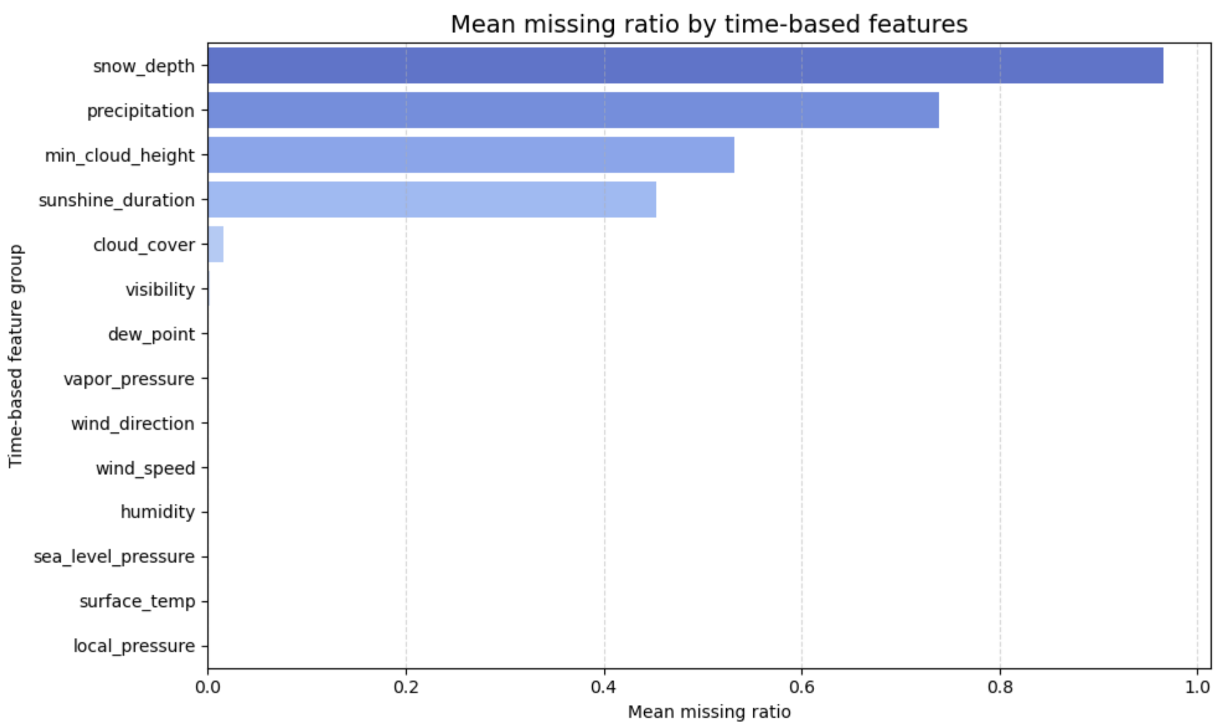
[88]:
```python
def analyze_missing_values_by_time_groups(df, time_based_feature_names):
    group_missing_summary = []

    for base_name in time_based_feature_names:
        cols = [col for col in df.columns if col.startswith(base_name + "_")]
        if not cols:
            continue
        missing_ratio = df[cols].isnull().mean().mean()  # 각 시간대의 평균 결측률
        group_missing_summary.append((base_name, missing_ratio))
```

```
    summary_df = pd.DataFrame(group_missing_summary, columns=["feature_group",␣
↪"mean_missing_ratio"])
    summary_df = summary_df.sort_values(by="mean_missing_ratio", ascending=False).
↪reset_index(drop=True)
    return summary_df
```

[89]:
```
group_missing_ratio = analyze_missing_values_by_time_groups(train_df,␣
↪time_based_feature_names)

plt.figure(figsize=(10, 6))
sns.barplot(
    data=group_missing_ratio,
    x="mean_missing_ratio",
    y="feature_group",
    hue="feature_group",
    palette="coolwarm",
)
plt.title("Mean missing ratio by time-based features", fontsize=14)
plt.xlabel("Mean missing ratio")
plt.ylabel("Time-based feature group")
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



Feature Engineering

결측치 처리

결측치 처리 방법

1. -9999는 NaN으로 처리
2. snow_depth, precipitation, min_cloud_height, sunshine_duration과 같이 결측치 비율이 높은 feature 처리 논의…
3. sunshine_duration이 야간에 결측치 -> 0으로 처리

[90]:
```python
def handle_missing_values(df):
    # 1. -9999를 NaN으로 변환
    df = df.replace(-9999, np.nan)

    # 2. 시간대별 특징에 맞게 처리
    night_hours = [0, 1, 2, 3, 4, 5, 22, 23]
    day_hours = [h for h in range(6, 22) if h not in night_hours]

    # 2.1. sunshine_duration
    # 2.1.1. 밤 시간대
    for hour in night_hours:
        col = f'sunshine_duration_{hour}'
        if col in df.columns:
            df[col] = df[col].fillna(0)

    # 2.1.2. 낮 시간대
    for hour in day_hours:
        col = f'sunshine_duration_{hour}'
        if col in df.columns:
            df[col] = df.groupby('station')[col].transform(lambda x: x.fillna(x.
↪mean()))
            # 낮시간대에 측정소 문제로 NaN이 있을 경우, global mean으로 대체함
            df[col] = df[col].fillna(df[col].mean())

    # 2.2. snow_depth
    # 눈이 오지 않는 계절에는 0으로 대체하고, 눈이 오는 계절에는 그룹별 평균으로␣
↪대체
    month_series = df['date'].str.split('-').str[0].astype(int)
    winter_mask = month_series.isin([12, 1, 2])

    for hour in range(24):
        col = f'snow_depth_{hour}'
        if col in df.columns:
            # 겨울철
            df.loc[winter_mask, col] = df.loc[winter_mask].groupby('station')[col].
↪transform(
                lambda x: x.fillna(x.mean()))

            # 겨울 X -> 0으로 대체
            df.loc[~winter_mask, col] = df.loc[~winter_mask][col].fillna(0)

            # 겨울철에도 NaN이 남아있을 경우, global mean으로 대체
```

```python
            df[col] = df[col].fillna(df[col].mean())

            if 'month' in df.columns:
                df = df.drop('month', axis=1)

    # 2.3. percipitation
    # 지역별(측정소별) persistency가 존재할 것이므로, forward / back fill 사용(임시)
    for hour in range(24):
        col = f'precipitation_{hour}'
        if col in df.columns:
            df[col] = df.groupby('station')[col].transform(lambda x: x.ffill())
            df[col] = df.groupby('station')[col].transform(lambda x: x.bfill())
            df[col] = df[col].fillna(0) # 비 안오는 날은 0으로

    # 2.4. min_cloud_height
    # outlier 영향을 줄이기 위해 median 사용함
    for hour in range(24):
        col = f'min_cloud_height_{hour}'
        if col in df.columns:
            df[col] = df.groupby('station')[col].transform(lambda x: x.fillna(x.
↪median()))
            df[col] = df[col].fillna(df[col].median())

    # 3. 나머지 feature 처리
    remaining_time_based_features = [
        'cloud_cover', 'dew_point', 'humidity', 'local_pressure',
        'sea_level_pressure', 'surface_temp', 'vapor_pressure',
        'visibility', 'wind_direction', 'wind_speed'
    ]

    for feature in remaining_time_based_features:
        for hour in range(24):
            col = f'{feature}_{hour}'
            if col in df.columns and df[col].isna().any():
                # station별 평균으로 대체
                df[col] = df.groupby('station')[col].transform(lambda x: x.fillna(x.
↪mean()))
                # 남은 결측치는 global mean으로
                df[col] = df[col].fillna(df[col].mean())

    return df
```

[91]:
```python
print("결측치 처리 전 train_df 결측치 개수")
print(train_df.isnull().sum().sum())

train_df = handle_missing_values(train_df)

print("결측치 처리 후 train_df 결측치 개수")
print(train_df.isnull().sum().sum())
```

```
결측치 처리 전 train_df 결측치 개수
853979
```

결측치 처리 후 train_df 결측치 개수
0

```
[ ]:
```