
Machine Learning
1조

PRESENTATION

# 서울시 따릉이 대여량 예측 경진대회

August 17th 2023

이은준
밤서연
박제재

1. Data explanation
2. Missing value and correlation
3. Data preprocessing
4. Regression analysis
5. Result

Machine Learning team 1

# Data Explanation

KUBIG

id : 고유 id

hour : 시간

temperature : 기온

precipitation : 비가 오지 않았으면 0, 비가 오면 1

windspeed : 풍속(평균)

humidity : 습도

visibility : 시정(視程), 시계(視界)(특정 기상 상태에 따른 가시성을 의미)

ozone : 오존

pm10 : 미세먼지(머리카락 굵기의 1/5에서 1/7 크기의 미세먼지)

pm2.5 : 미세먼지(머리카락 굵기의 1/20에서 1/30 크기의 미세먼지)

count : 시간에 따른 따릉이 대여 수

```
[ ] train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              1459 non-null   int64
 1   hour            1459 non-null   int64
 2   temperature     1457 non-null   float64
 3   precipitation   1457 non-null   float64
 4   windspeed       1450 non-null   float64
 5   humidity        1457 non-null   float64
 6   visibility      1457 non-null   float64
 7   ozone           1383 non-null   float64
 8   pm10            1369 non-null   float64
 9   pm2.5           1342 non-null   float64
 10  count           1459 non-null   float64
dtypes: float64(9), int64(2)
memory usage: 125.5 KB
```

| | id | hour | hour_bef_temperature | hour_bef_precipitation | hour_bef_windspeed | hour_bef_humidity | hour_bef_visibility | hour_bef_ozone | hour_bef_pm10 | hour_bef_pm2.5 | count |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 20 | 16.3 | 1.0 | 1.5 | 89.0 | 576.0 | 0.027 | 76.0 | 33.0 | 49.0 |
| 1 | 6 | 13 | 20.1 | 0.0 | 1.4 | 48.0 | 916.0 | 0.042 | 73.0 | 40.0 | 159.0 |
| 2 | 7 | 6 | 13.9 | 0.0 | 0.7 | 79.0 | 1382.0 | 0.033 | 32.0 | 19.0 | 26.0 |
| 3 | 8 | 23 | 8.1 | 0.0 | 2.7 | 54.0 | 946.0 | 0.040 | 75.0 | 64.0 | 57.0 |
| 4 | 9 | 18 | 29.5 | 0.0 | 4.8 | 7.0 | 2000.0 | 0.057 | 27.0 | 11.0 | 431.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1454 | 2174 | 4 | 16.8 | 0.0 | 1.6 | 53.0 | 2000.0 | 0.031 | 37.0 | 27.0 | 21.0 |
| 1455 | 2175 | 3 | 10.8 | 0.0 | 3.8 | 45.0 | 2000.0 | 0.039 | 34.0 | 19.0 | 20.0 |
| 1456 | 2176 | 5 | 18.3 | 0.0 | 1.9 | 54.0 | 2000.0 | 0.009 | 30.0 | 21.0 | 22.0 |
| 1457 | 2178 | 21 | 20.7 | 0.0 | 3.7 | 37.0 | 1395.0 | 0.082 | 71.0 | 36.0 | 216.0 |
| 1458 | 2179 | 17 | 21.1 | 0.0 | 3.1 | 47.0 | 1973.0 | 0.046 | 38.0 | 17.0 | 170.0 |

```
train.columns = ['id', 'hour', 'temperature', 'precipitation',
       'windspeed', 'humidity', 'visibility',
       'ozone', 'pm10', 'pm2.5', 'count']
```

# Missing value and Correlation

```
[ ] # 결측치 확인
    train.isna().sum()
```

```
id              0
hour            0
temperature     2
precipitation   2
windspeed       9
humidity        2
visibility      2
ozone          76
pm10           90
pm2.5         117
count           0
dtype: int64
```

```
[ ] # temperature, precipitation, windspeed, humidity, visibility의 경우 결측값인 행 제거 (결측값이 거의 없으므로)
    train = train.dropna(subset= ['temperature', 'precipitation', 'windspeed', 'humidity', 'visibility'])
```

```
[ ] train.isna().sum()
```

```
id              0
hour            0
temperature     0
precipitation   0
windspeed       0
humidity        0
visibility      0
ozone          73
pm10           87
pm2.5         114
count           0
dtype: int64
```

Machine Learning team 1

# Missing value and Correlation

KUBIG

EDA

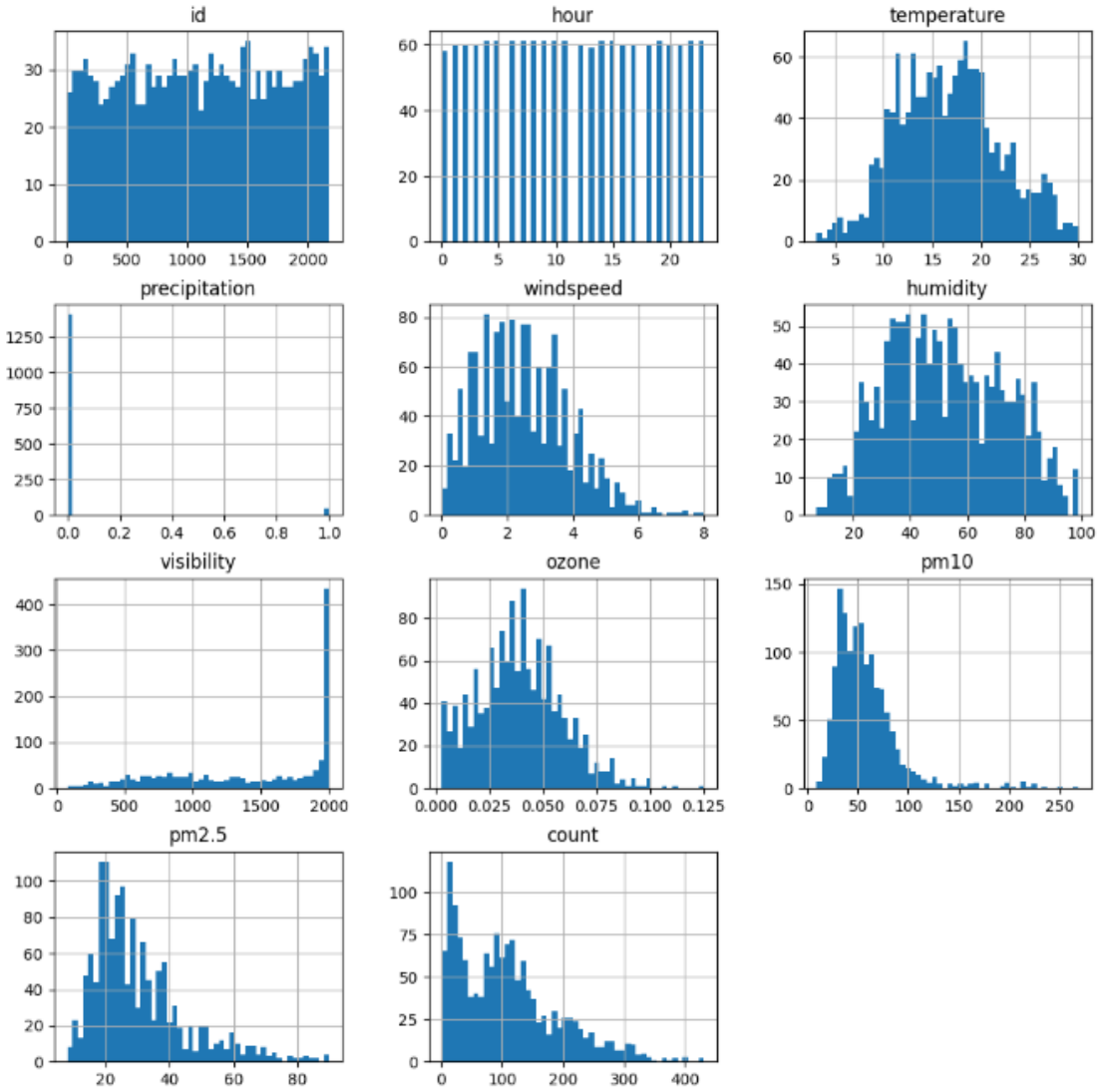
1. train set 의 히스토그램을 그려 분포 형태 확인

2. train set의 boxplot을 그려 아웃 라이어 확인

3. 상관관계 확인

4. 변수 간의 상관관계가 높으면 상호 대체 가능성 고려

5. ‘count’와 상관관계가 낮으면 변수 제거 고려



Machine Learning team 1

# Missing value and Correlation

KUBIG

EDA

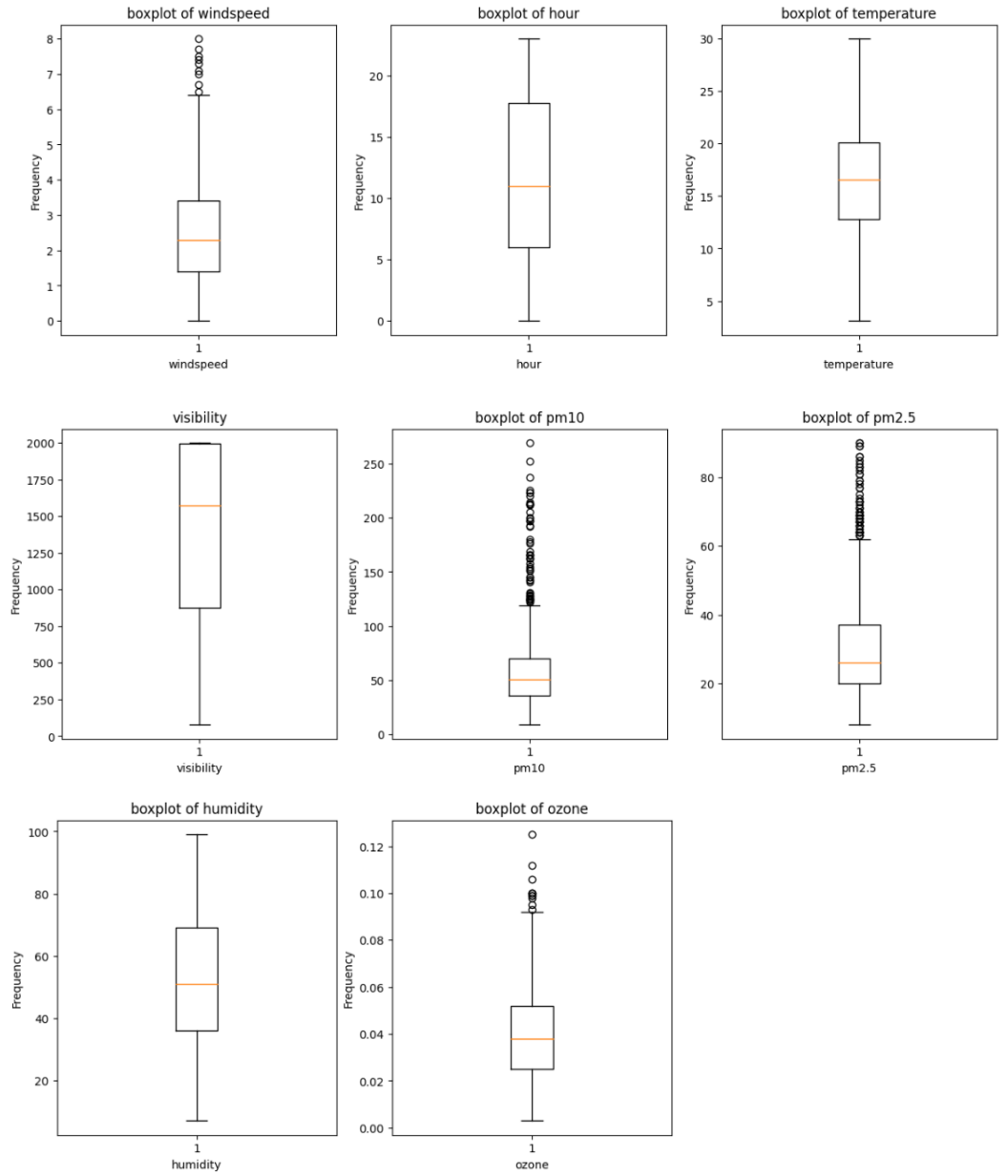
1. train set 의 히스토그램을 그려 분포 형태 확인

2. train set의 boxplot을 그려 아웃 라이어 확인

3. 상관관계 확인

4. 변수 간의 상관관계가 높으면 상호 대체 가능성 고려

5. ‘count’와 상관관계가 낮으면 변수 제거 고려



Machine Learning team 1

# Missing value and Correlation

KUBIG

상관 관계 확인 결과

precipitation, pm10, pm2.5가 상당히 낮은 상관관계를 보임

↓

Precipitation은 0, 1로 이루어진 범주형 데이터로 볼 수 있기 때문

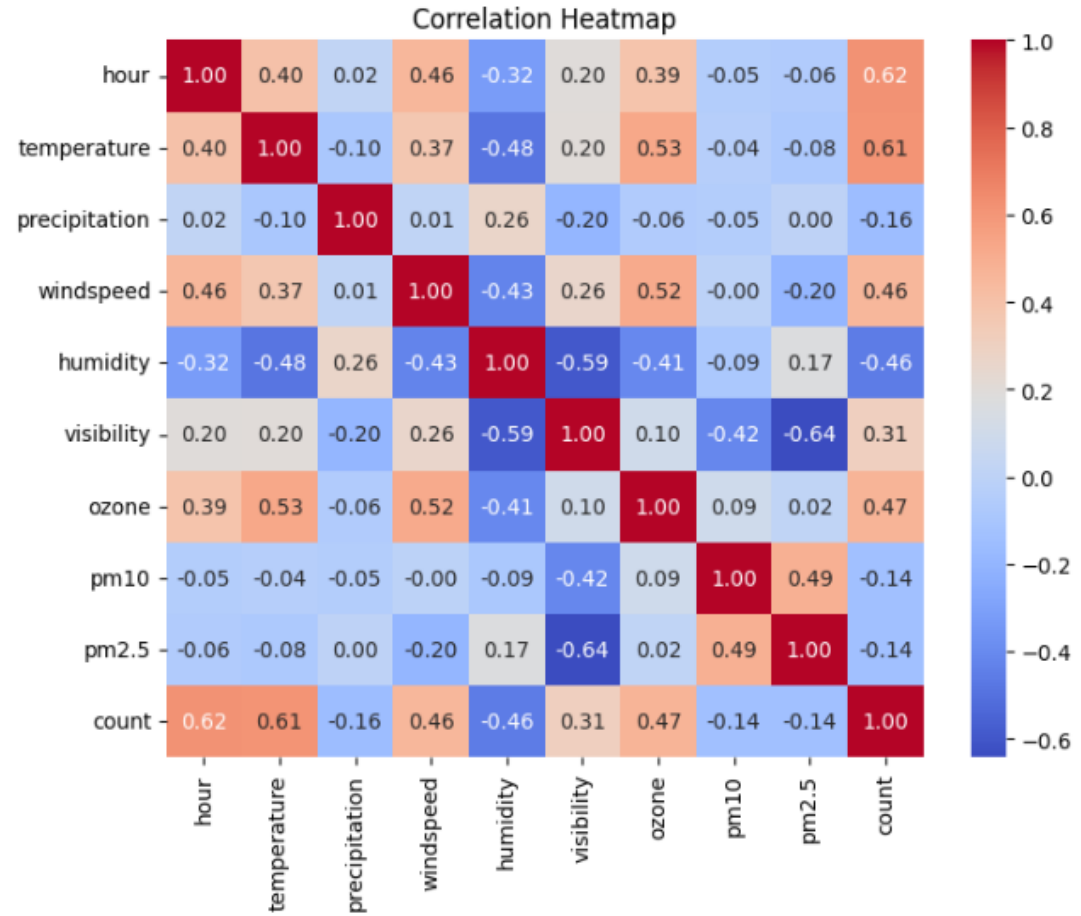
pm10, pm2.5는 data set이 코로나 19 때 측정 되었을 가능성 존재

↓

변수 제거 및 결측치 대체 고려 가능

↓

But, precipitation은 dummy variable이기에
'count'와의 상관관계 재확인 필요




```
count          1.000000
hour           0.620985
temperature    0.610444
ozone          0.468639
humidity       0.459149
windspeed      0.458083
visibility     0.308597
precipitation  0.159449
pm10           0.137321
pm2.5          0.136345
```

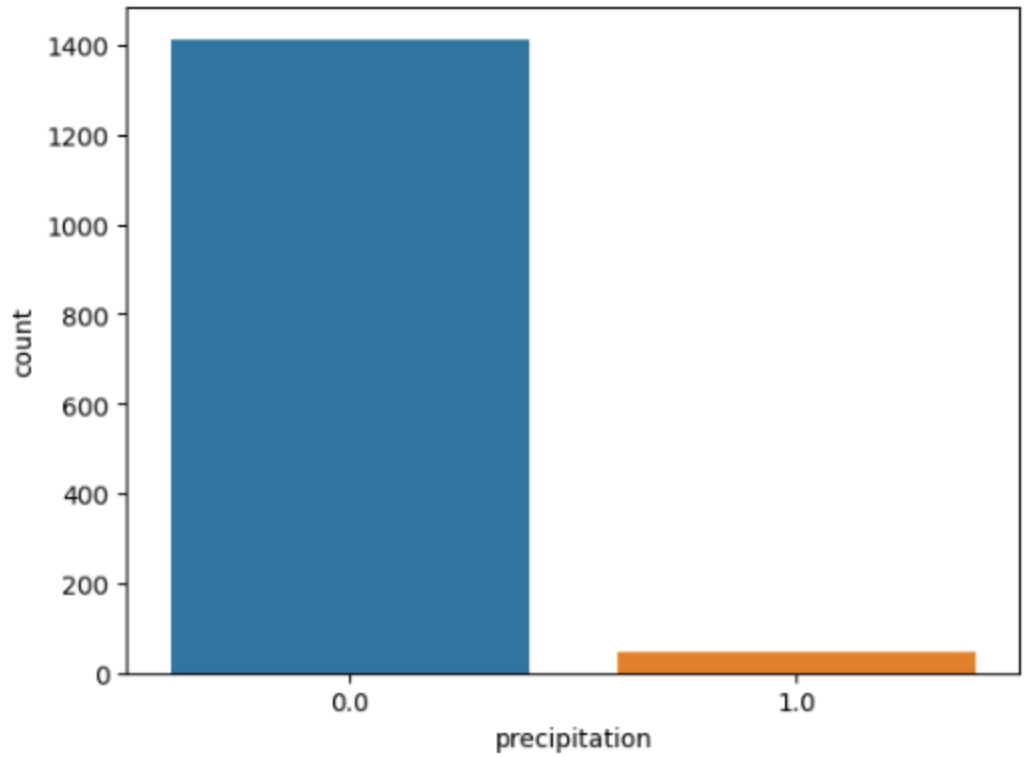
Machine Learning team 1

# Missing value and Correlation

KUBIG

```
sns.countplot(data = train, x = 'precipitation')
```

```
<Axes: xlabel='precipitation', ylabel='count'>
```




Precipitation은 0과 1로 이루어진 범주형 데이터로 볼 수 있음

↓

그러나, precipitation에 대한 'count'의 측정값이 극단적으로 쏠려 있음을 확인

↓

Ovetfitting의 가능성 존재

↓

Precipitation 역시 제거 가능성 존재

# Missing value of ozone, pm10, pm2.5

```
train.isna().sum()
```

```
id                0
hour              0
temperature       0
precipitation     0
windspeed         0
humidity          0
visibility        0
ozone            73
pm10             87
pm2.5           114
count             0
dtype: int64
```

1 & 2. 평균을 이용한 결측치 대체

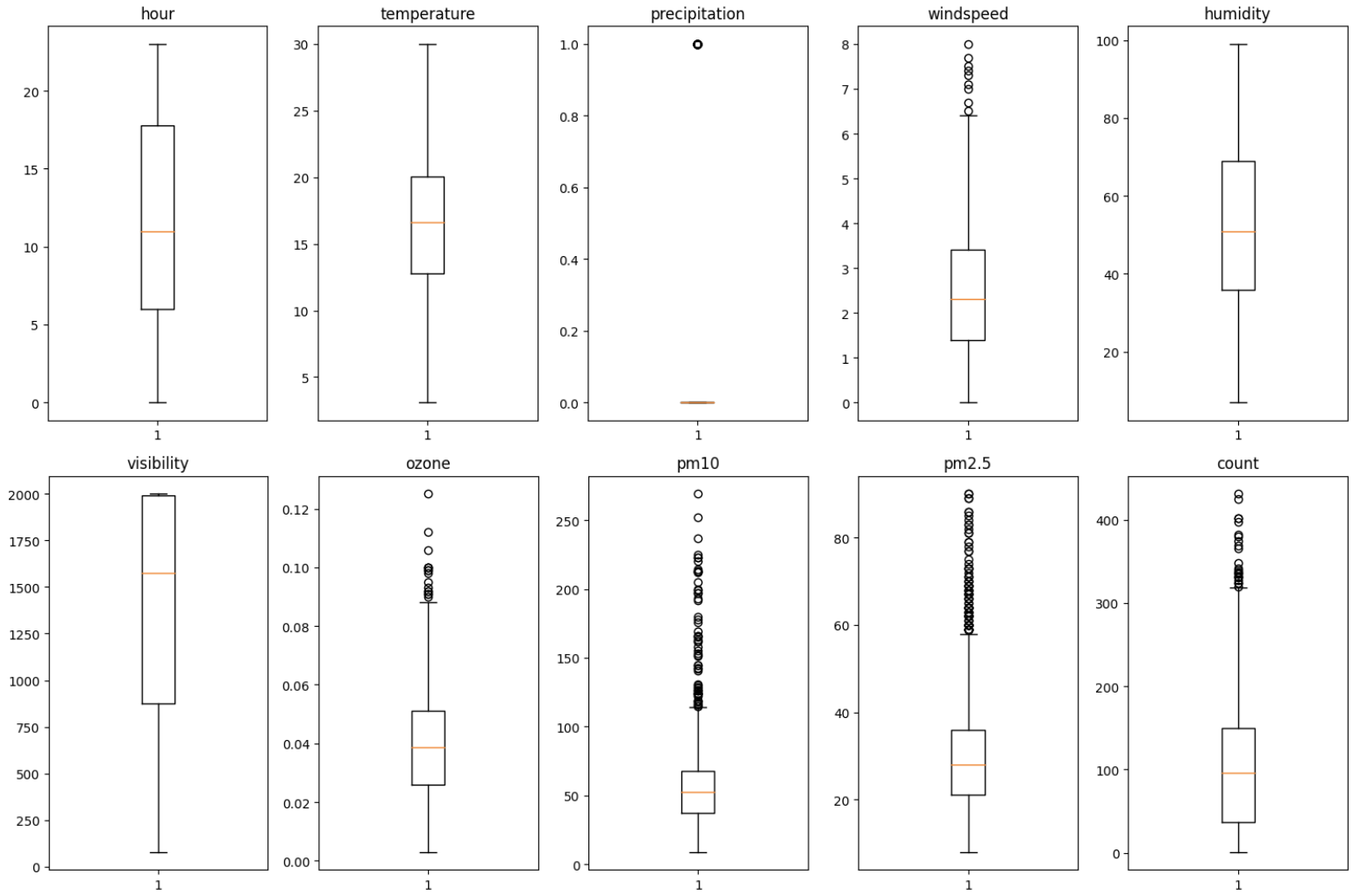
3. 회귀분석을 이용한 결측치 대체

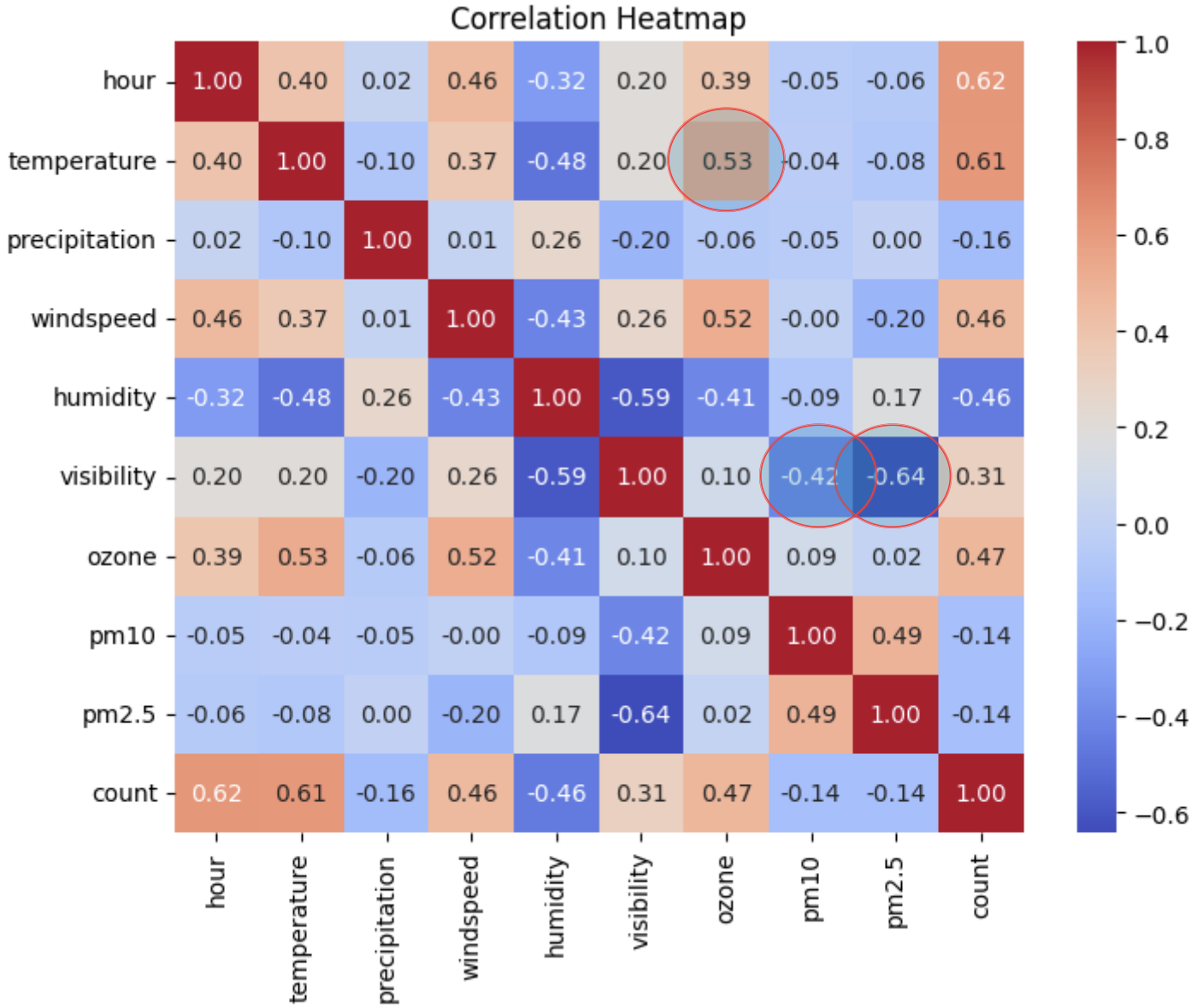
4. Knn 기법을 이용한 결측치 대체

```
[ ]  # 결측치 대체 1 = 아웃라이어 제거한 평균으로 대체
     def replace_with_mean_outlier_removed(column):
         # 결측치 제거
         column_without_na = column.dropna()
         # 아웃라이어 탐지
         q1 = column_without_na.quantile(0.25)
         q3 = column_without_na.quantile(0.75)
         iqr = q3 - q1
         lower_bound = q1 - 1.5 * iqr
         upper_bound = q3 + 1.5 * iqr
         mean_without_outliers = column_without_na[(column_without_na >= lower_bound) & (column_without_na <= upper_bound)]
         # 결측치를 평균값으로 대체
         column_filled = column.fillna(mean_without_outliers)
         return column_filled

     # 특정 열의 결측치를 아웃라이어를 제거한 뒤의 평균값으로 대체
     train1 = train.copy() # train은 ozone, pm10, pm2.5 결측치 제거하기 전의 version
     train1['ozone'] = replace_with_mean_outlier_removed(train1['ozone'])
     train1['pm10'] = replace_with_mean_outlier_removed(train1['pm10'])
     train1['pm2.5'] = replace_with_mean_outlier_removed(train1['pm2.5'])
```

```
[ ]  # 결측치 대체 2 = 아웃라이어 제거하지 않은 상태의 평균으로 대체
     def replace_with_mean(column):
         # 결측치 제거
         column_without_na = column.dropna()
         mean_value = column.mean()
         column_filled = column.fillna(mean_value)
         return column_filled

     # 특정 열의 결측치를 해당 열의 평균값으로 대체
     train2 = train.copy()
     train2['ozone'] = replace_with_mean(train2['ozone'])
     train2['pm10'] = replace_with_mean(train2['pm10'])
     train2['pm2.5'] = replace_with_mean(train2['pm2.5'])
```

# Outlier check




Correlation Heatmap
hour 1.00 0.40 0.02 0.46 -0.32 0.20 0.39 -0.05 -0.06 0.62
temperature 0.40 1.00 -0.10 0.37 -0.48 0.20 0.53 -0.04 -0.08 0.61
precipitation 0.02 -0.10 1.00 0.01 0.26 -0.20 -0.06 -0.05 0.00 -0.16
windspeed 0.46 0.37 0.01 1.00 -0.43 0.26 0.52 -0.00 -0.20 0.46
humidity -0.32 -0.48 0.26 -0.43 1.00 -0.59 -0.41 -0.09 0.17 -0.46
visibility 0.20 0.20 -0.20 0.26 -0.59 1.00 0.10 -0.42 -0.64 0.31
ozone 0.39 0.53 -0.06 0.52 -0.41 0.10 1.00 0.09 0.02 0.47
pm10 -0.05 -0.04 -0.05 -0.00 -0.09 -0.42 0.09 1.00 0.49 -0.14
pm2.5 -0.06 -0.08 0.00 -0.20 0.17 -0.64 0.02 0.49 1.00 -0.14
count 0.62 0.61 -0.16 0.46 -0.46 0.31 0.47 -0.14 -0.14 1.00
hour
temperature
precipitation
windspeed
humidity
visibility
ozone
pm10
pm2.5
count
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

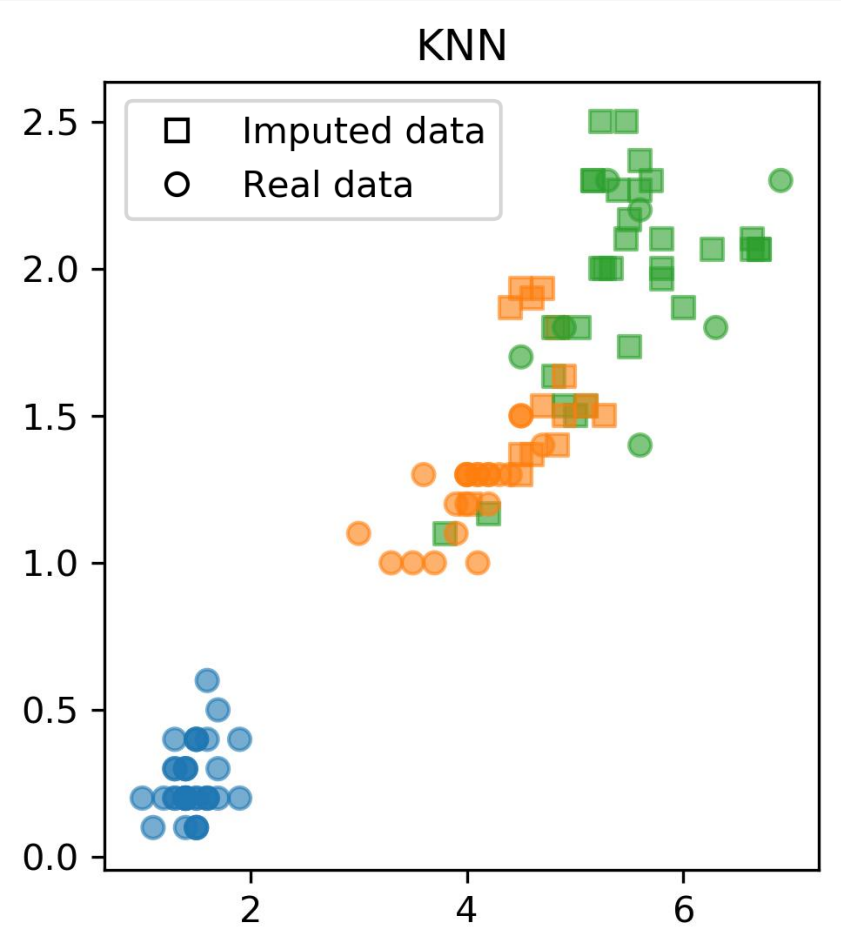
```
[ ]  # 결측치 대체 3 = 오존의 경우 temperature과의 상관도가 높으므로 고려해서 새로운 데이터 생성
     #               pm10의 경우 pm2.5와의 상관도가 높으나 결측치가 pm2.5에도 존재하므로 visibility 고려해서 새로운 데이터 생성
     #               pm2.5의 경우 visibility와의 상관도가 높으므로 고려해서 새로운 데이터 생성
     from sklearn.linear_model import LinearRegression
     tmp1 = train[['id', 'ozone', 'temperature']]
     tmp2 = train[['id', 'pm10', 'visibility']]
     tmp3 = train[['id', 'pm2.5', 'visibility']]
     train3 = train.copy()

     # 결측치가 없는 행(train)과 결측치가 있는 행(test)을 분리
     data1 = tmp1.dropna(subset=['ozone'])
     data2 = tmp1[tmp1['ozone'].isnull()]

     data3 = tmp2.dropna(subset=['pm10'])
     data4 = tmp2[tmp2['pm10'].isnull()]

     data5 = tmp3.dropna(subset=['pm2.5'])
     data6 = tmp3[tmp3['pm2.5'].isnull()]
```

```
[ ]  # 단순선형회귀모델 생성 및 학습
     X_train = data1['temperature'].values.reshape(-1,1)
     y_train = data1['ozone'].values.reshape(-1,1)
     model = LinearRegression()
     model.fit(X_train, y_train)

     # 예측값 생성
     X_test = data2['temperature'].values.reshape(-1,1)
     y_pred = model.predict(X_test)


     # 예측값으로 결측치 대체
     data2_copy = data2.copy()
     data2_copy['ozone'] = y_pred.flatten()

     new_ozone = pd.concat([data1, data2_copy]).sort_values(by='id')

     train3['ozone'] = new_ozone['ozone']
```

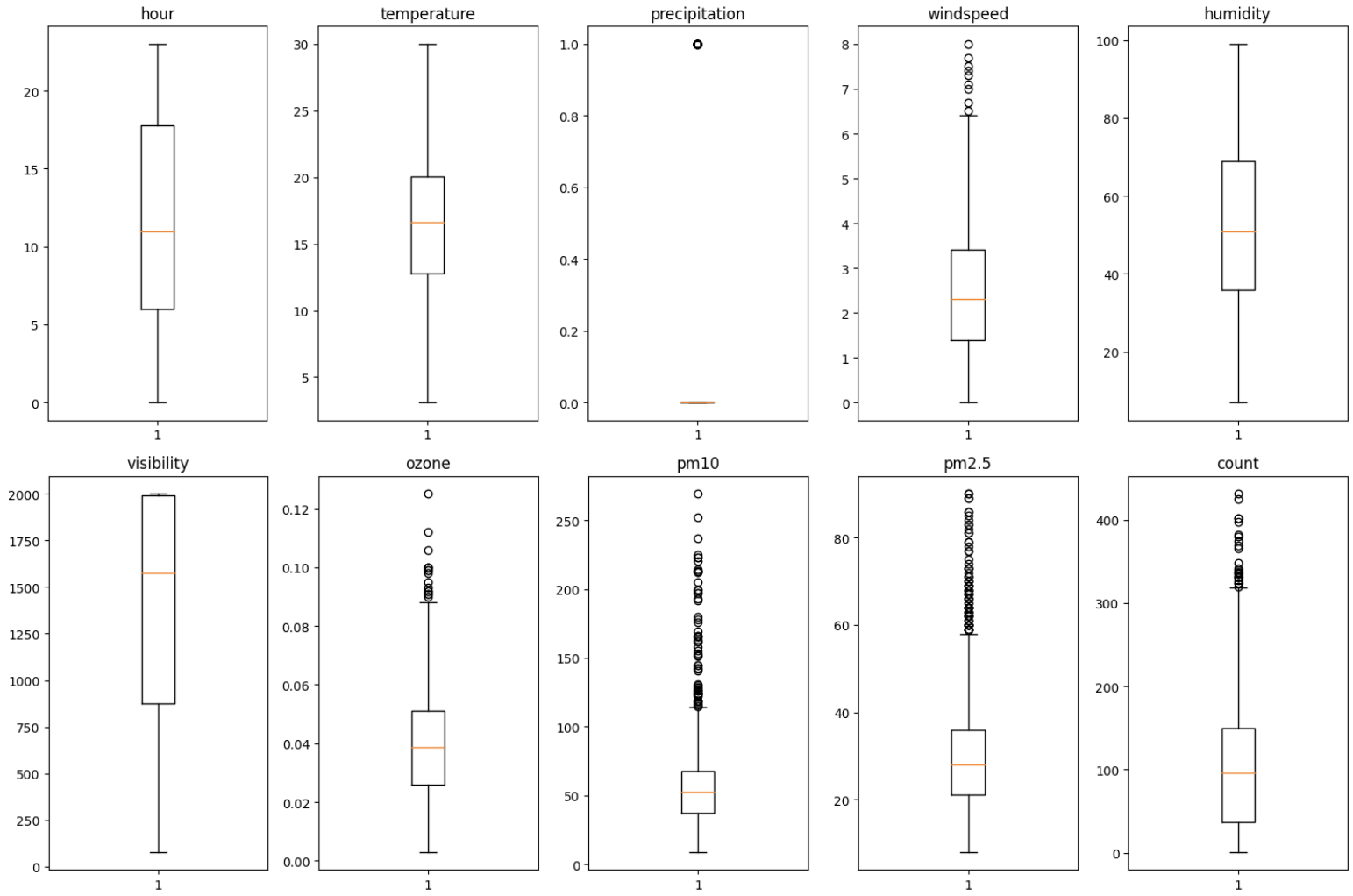
```
# 단순선형회귀모델 생성 및 학습
X_train = data3['visibility'].values.reshape(-1,1)
y_train = data3['pm10'].values.reshape(-1,1)
model = LinearRegression()
model.fit(X_train, y_train)

# 예측값 생성
X_test = data4['visibility'].values.reshape(-1,1)
y_pred = model.predict(X_test)

# 예측값으로 결측치 대체
data4_copy = data4.copy()
data4_copy['pm10'] = y_pred.flatten()

new_pm10 = pd.concat([data3, data4_copy]).sort_values(by='id')

train3['pm10'] = new_pm10['pm10']
```

```
# 단순선형회귀모델 생성 및 학습
X_train = data5['visibility'].values.reshape(-1,1)
y_train = data5['pm2.5'].values.reshape(-1,1)
model = LinearRegression()
model.fit(X_train, y_train)

# 예측값 생성
X_test = data6['visibility'].values.reshape(-1,1)
y_pred = model.predict(X_test)

# 예측값으로 결측치 대체
data6_copy = data6.copy()
data6_copy['pm2.5'] = y_pred.flatten()

new_pm25 = pd.concat([data5, data6_copy]).sort_values(by='id')

train3['pm2.5'] = new_pm25['pm2.5']
```

```
[ ]  from sklearn.impute import KNNImputer

     # 결측치 대체 4 = knn 기법 이용해서 대체
     def replace_with_knn(column, k=3):
         knn_imputer = KNNImputer(n_neighbors=k)
         t_column = column.values.reshape(-1,1)
         imputed_column = knn_imputer.fit_transform(t_column)
         return imputed_column

     # 특정 열의 결측치를 해당 열의 평균값으로 대체
     train4 = train.copy()
     train4['ozone'] = replace_with_knn(train4['ozone'])
     train4['pm10'] = replace_with_knn(train4['pm10'])
     train4['pm2.5'] = replace_with_knn(train4['pm2.5'])
```



# Outlier check & removal



# Scaling

```
[ ]  from sklearn.preprocessing import MinMaxScaler, StandardScaler
     def scaling(df):
       selected_columns = df.columns[~df.columns.isin(['id', 'precipitation', 'count'])]

       # 정규화를 위한 MinMaxScaler 생성 및 적용
       scaler = MinMaxScaler()
       df[selected_columns] = scaler.fit_transform(df[selected_columns])
       return df
```

| | id | hour | temperature | precipitation | windspeed | humidity | visibility | ozone | pm10 | pm2.5 | count |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 1.227917 | -0.078534 | 1.0 | -0.715749 | 1.767924 | -1.582245 | -0.659299 | 1.361162 | 0.584922 | 49.0 |
| 1 | 6 | 0.225821 | 0.661081 | 0.0 | -0.791417 | -0.251203 | -0.981965 | 0.205031 | 1.202264 | 1.306705 | 159.0 |
| 2 | 7 | -0.776275 | -0.545660 | 0.0 | -1.321092 | 1.275454 | -0.159229 | -0.313567 | -0.969345 | -0.858644 | 26.0 |
| 5 | 13 | -1.348901 | -0.604051 | 0.0 | -0.564414 | 1.324701 | -0.704777 | -0.659299 | -0.863413 | -1.271092 | 39.0 |
| 6 | 14 | -1.205744 | -1.187958 | 0.0 | -0.715749 | 0.241267 | 0.133849 | -0.025457 | 0.619637 | 0.584922 | 23.0 |

# Dimension reduction (PCA)

```
[ ]  # 차원축소 1 = PCA (precipitation, pm10, pm2.5 제거하지 않을 경우 실행)
     from sklearn.decomposition import PCA

     def pca(df):
       X = df.drop(columns=['id', 'count'])
       pca = PCA(n_components=3)
       X_pca = pca.fit_transform(X)
       c_variance = pca.explained_variance_ratio_
       print(c_variance)
       pca_result = pd.DataFrame(X_pca, columns=['PCA1', 'PCA2', 'PCA3'])
       return (pca_result)
```

```
[ ]  # 차원축소 2 = PCA (precipitation, pm10, pm2.5 제거할 경우 실행)
     from sklearn.decomposition import PCA

     def pca(df):
       # precipitation, pm10, pm2.5 제거하기
       X = df.drop(columns=['id', 'count', 'precipitation', 'pm10', 'pm2.5'])
       pca = PCA(n_components=3)
       X_pca = pca.fit_transform(X)
       c_variance = pca.explained_variance_ratio_
       print(c_variance)
       pca_result = pd.DataFrame(X_pca, columns=['PCA1', 'PCA2', 'PCA3'])
       return (pca_result)
```

```
[0.48703327 0.18368499 0.1158931 ]
```

| | PCA1 | PCA2 | PCA3 |
|---|---|---|---|
| **0** | -1.389380 | 1.728005 | 0.993965 |
| **1** | -0.068202 | 0.642999 | -0.664529 |
| **2** | -1.843660 | -0.084826 | -0.384806 |
| **3** | -0.256678 | 0.789678 | 1.886454 |
| **4** | 3.863397 | -0.335353 | -0.600187 |
| **...** | ... | ... | ... |
| **1445** | -0.575567 | -1.297349 | -0.695567 |
| **1446** | -0.078505 | -1.272155 | 0.157186 |
| **1447** | -0.812868 | -1.592214 | -0.247226 |
| **1448** | 2.529259 | 1.205930 | -0.069649 |
| **1449** | 1.426142 | -0.268987 | 0.214263 |

1450 rows × 3 columns

# Validation

성능 평가 (MSE, R squared)

```
[31] # s_no_train1 = 아웃라이어 제거한 평균으로 결측치 대체
     # 다중선형회귀 (precipitation, pm10, pm2.5 제거 x)
     reg(s_no_train1)
```

[2277.2417733703437, 0.5698185503060063]

```
[46] # s_no_train1
     # 다중선형회귀 (precipitation, pm10, pm2.5 제거 O)
     reg(s_no_train1)
```

[2262.8626950211815, 0.5725348243713425]

```
[57] # s_no_train1
     # ExtraTreesRegressor
     reg(s_no_train1)
```

[1307.3252071428572, 0.7530402527273704]

```
[33] # s_no_train3 = 상관도 가장 높은 다른 설명변수로 회귀분석 후 결측치 대체
     # 다중선형회귀 (precipitation, pm10, pm2.5 제거 x)
     reg(s_no_train3)
```

[2801.4543312161272, 0.5874904806014543]

```
[48] # s_no_train3
     # 다중선형회귀 (precipitation, pm10, pm2.5 제거 O)
     reg(s_no_train3)
```

[2882.247591084277, 0.5755938066391184]

```
[59] # s_no_train3
     # ExtraTreesRegressor
     reg(s_no_train3)
```

[1280.8942671814673, 0.81139043650731]

```
[32] # s_no_train2 = 아웃라이어 제거하지 않은 평균으로 결측치 대체
     # 다중선형회귀 (precipitation, pm10, pm2.5 제거 x)
     reg(s_no_train2)
```

[2277.2616064057056, 0.5698148037543649]

```
[47] # s_no_train2
     # 다중선형회귀 (precipitation, pm10, pm2.5 제거 O)
     reg(s_no_train2)
```

[2262.7004511808805, 0.5725654729793388]

```
[58] # s_no_train2
     # ExtraTreesRegressor
     reg(s_no_train2)
```

[1338.8019515873016, 0.7470941508619027]

```
[35] # s_no_train4 = knn으로 결측치 대체
     # 다중선형회귀 (precipitation, pm10, pm2.5 제거 x)
     reg(s_no_train4)
```

[2277.2616064057056, 0.5698148037543649]

```
[49] # s_no_train4
     # 다중선형회귀 (precipitation, pm10, pm2.5 제거 O)
     reg(s_no_train4)
```

[2262.7004511808805, 0.5725654729793388]

```
[60] # s_no_train4
     # ExtraTreesRegressor
     reg(s_no_train4)
```

[1332.518175793651, 0.7482811850240523]

# Test Dataset preprocessing

```
filtered = test[test['hour']==19]
filtered # hour = 19에 대해 특정 row가 모든 값이 없음. id = 1943
```

| | id | hour | temperature | precipitation | windspeed | humidity | visibility | ozone | pm10 | pm2.5 |
|---|---|---|---|---|---|---|---|---|---|---|
| **653** | 1943 | 19 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

```
[14] hour_19_mean = filtered.mean(skipna=True)
     hour_19_mean
```

```
id              1038.533333
hour              19.000000
temperature       26.110345
precipitation      0.068966
windspeed          3.541379
humidity          47.689655
visibility      1561.758621
ozone              0.054207
pm10              39.214286
pm2.5             26.071429
dtype: float64
```

```
id_to_find = 1943
values_to_fill = [26.110345, 0, 3.541379, 47.689655, 1561.758621]

if id_to_find in test['id'].values:
    index_to_fill = test[test['id'] == id_to_find].index[0]
    test.iloc[index_to_fill, 2:7] = values_to_fill
    print("값이 채워진 결과:")
    print(test)
```

```
[16] test.isnull().sum()
```

```
id              0
hour            0
temperature     0
precipitation   0
windspeed       0
humidity        0
visibility      0
ozone          35
pm10           37
pm2.5          36
dtype: int64
```

# Test Dataset preprocessing

결측치 전처리 진행 후 scaling까지 완료한 test 데이터셋 s_test1 과 train 데이터셋 s_no_train3.

```
from sklearn.linear_model import LinearRegression
tmp1 = test[['id', 'ozone', 'temperature']]
tmp2 = test[['id', 'pm10', 'visibility']]
tmp3 = test[['id', 'pm2.5', 'visibility']]
test1 = test.copy()

# 결측치가 없는 행(train)과 결측치가 있는 행(test)을 분리
data1 = tmp1.dropna(subset=['ozone'])
data2 = tmp1[tmp1['ozone'].isnull()]

data3 = tmp2.dropna(subset=['pm10'])
data4 = tmp2[tmp2['pm10'].isnull()]

data5 = tmp3.dropna(subset=['pm2.5'])
data6 = tmp3[tmp3['pm2.5'].isnull()]
```

```
# 단순선형회귀모델 생성 및 학습
X_train = data1['temperature'].values.reshape(-1,1)
y_train = data1['ozone'].values.reshape(-1,1)
model = LinearRegression()
model.fit(X_train, y_train)

# 예측값 생성
X_test = data2['temperature'].values.reshape(-1,1)
y_pred = model.predict(X_test)


# 예측값으로 결측치 대체
data2_copy = data2.copy()
data2_copy['ozone'] = y_pred.flatten()

new_ozone = pd.concat([data1, data2_copy]).sort_values(by='id')

test1['ozone'] = new_ozone['ozone']
```

```
# 단순선형회귀모델 생성 및 학습
X_train = data3['visibility'].values.reshape(-1,1)
y_train = data3['pm10'].values.reshape(-1,1)
model = LinearRegression()
model.fit(X_train, y_train)

# 예측값 생성
X_test = data4['visibility'].values.reshape(-1,1)
y_pred = model.predict(X_test)

# 예측값으로 결측치 대체
data4_copy = data4.copy()
data4_copy['pm10'] = y_pred.flatten()

new_pm10 = pd.concat([data3, data4_copy]).sort_values(by='id')

test1['pm10'] = new_pm10['pm10']
```

```
# 단순선형회귀모델 생성 및 학습
X_train = data5['visibility'].values.reshape(-1,1)
y_train = data5['pm2.5'].values.reshape(-1,1)
model = LinearRegression()
model.fit(X_train, y_train)

# 예측값 생성
X_test = data6['visibility'].values.reshape(-1,1)
y_pred = model.predict(X_test)

# 예측값으로 결측치 대체
data6_copy = data6.copy()
data6_copy['pm2.5'] = y_pred.flatten()

new_pm25 = pd.concat([data5, data6_copy]).sort_values(by='id')

test1['pm2.5'] = new_pm25['pm2.5']
```

# Random Forest Regressor & Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV
```

```
params = { 'n_estimators' : [10, 100],
           'max_depth' : [6, 8, 10, 12],
           'min_samples_leaf' : [6, 8, 12, 18],
           'min_samples_split' : [4, 8, 12, 16, 20]
           }

rf_clf = RandomForestRegressor(random_state = 42, n_jobs = -1)
grid_cv = GridSearchCV(rf_clf, param_grid = params, cv = 3, n_jobs= -1)
grid_cv.fit(X_train,y_train)

print('최적 초모수 : ', grid_cv.best_params_)
```

```
최적 초모수 :  {'max_depth': 12, 'min_samples_leaf': 6, 'min_samples_split': 4, 'n_estimators': 100}
```

```
RFR_tuned = RandomForestRegressor(n_estimators = 100, max_depth = 12, min_samples_leaf = 6,
                                  min_samples_split = 4, random_state = 42, n_jobs = -1)
RFR_tuned.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=12, min_samples_leaf=6, min_samples_split=4,
                      n_jobs=-1, random_state=42)
```

```
y_pred_RFR_tuned = RFR_tuned.predict(X_test)
```

```
[35]  from sklearn.ensemble import RandomForestRegressor

      RFR = RandomForestRegressor(random_state = 42)
```

```
[36]  features = ['hour', 'temperature', 'precipitation',
                  'windspeed', 'humidity', 'visibility',
                  'ozone', 'pm10', 'pm2.5']
      X_train = s_no_train3[features]
      y_train = s_no_train3['count']
      X_test = s_test1[features]
```

Score : 62.5565

# Extra Trees Regressor

- 앙상블 학습으로 여러 트리를 생성하여 정확도를 높임.
- 부트스트랩 샘플을 사용하지 않아 전체 훈련 세트를 사용할 때 랜덤으로 분할하여 무작위성을 증가시킴.
- 랜덤 포레스트는 feature에 대해 정보이득을 계산해 높은 순으로 노드를 split하지만, 엑스트라 트리는 split할 때 무작위로 feature를 선택.
- 준수한 성능과 오버피팅 방지 효과가 있으며, 랜덤 포레스트와 다르게 최적의 분할을 계산하지 않기에 연산량이 상대적으로 적어 속도가 빠름.
- 모든 데이터를 사용하기에 낮은 bias.

# Pycaret & Extra Trees Regressor

```
def compare_regression_models_RMSE(df):
    exp = setup(df, target='count', session_id=20)
    best_model = compare_models(fold=5, sort='RMSE')

    return best_model
```

```
X_train_pycaret = s_no_train3[['hour', 'temperature', 'precipitation',
       'windspeed', 'humidity', 'visibility',
       'ozone', 'pm10', 'pm2.5','count']]

best_RMSE = compare_regression_models_RMSE(X_train_pycaret)
```

| | Model | MAE | MSE | RMSE | R2 | RMSLE | MAPE | TT (Sec) |
|---|---|---|---|---|---|---|---|---|
| **et** | Extra Trees Regressor | 24.1156 | 1222.1696 | 34.8521 | 0.7897 | 0.4825 | 0.6122 | 0.4000 |
| **catboost** | CatBoost Regressor | 24.6908 | 1242.5665 | 35.1295 | 0.7859 | 0.5125 | 0.6415 | 2.2080 |
| **rf** | Random Forest Regressor | 25.2410 | 1311.9723 | 36.1002 | 0.7745 | 0.5055 | 0.6851 | 0.5800 |
| **xgboost** | Extreme Gradient Boosting | 24.7548 | 1341.4079 | 36.4844 | 0.7685 | 0.5025 | 0.5940 | 0.1940 |
| **lightgbm** | Light Gradient Boosting Machine | 25.9528 | 1380.1133 | 37.0406 | 0.7615 | 0.5965 | 0.8631 | 0.3800 |
| **gbr** | Gradient Boosting Regressor | 25.8633 | 1402.8475 | 37.3048 | 0.7587 | 0.5259 | 0.7291 | 0.2680 |
| **ada** | AdaBoost Regressor | 35.2093 | 1873.0806 | 43.2572 | 0.6759 | 0.7024 | 1.3082 | 0.2080 |
| **knn** | K Neighbors Regressor | 34.5326 | 2168.0439 | 46.4868 | 0.6230 | 0.7075 | 1.2392 | 0.0860 |
| **lr** | Linear Regression | 35.3795 | 2200.0131 | 46.8802 | 0.6198 | 0.7576 | 1.2020 | 0.3300 |
| **ridge** | Ridge Regression | 35.3725 | 2200.0567 | 46.8815 | 0.6197 | 0.7548 | 1.2121 | 0.0540 |

# Pycaret & Extra Trees Regressor

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import ExtraTreesRegressor

param_grid = {'n_estimators': [50, 100, 150],
    'min_samples_leaf': [3, 5, 7, 9],
    'min_samples_split': [2, 4, 6],
    'max_depth': [30, 50, 100]}

et_reg = ExtraTreesRegressor(random_state = 20)

grid_search = GridSearchCV(et_reg, param_grid, cv=5)
grid_search.fit(X_train,y_train)

grid_search.best_estimator_
```

```
submission_ETR_tuned = submission.copy()
submission_ETR_tuned['count'] = y_pred_ETR_tuned
submission_ETR_tuned
```

| | id | count |
|---|---|---|
| 0 | 0 | 53.837690 |
| 1 | 1 | 220.917698 |
| 2 | 2 | 39.042000 |
| 3 | 4 | 30.161556 |
| 4 | 5 | 37.492333 |
| ... | ... | ... |
| 710 | 2148 | 63.939956 |
| 711 | 2149 | 39.279905 |
| 712 | 2165 | 100.121889 |
| 713 | 2166 | 154.125508 |
| 714 | 2177 | 141.470524 |

Score: 61.9959

```
ETR_tuned = ExtraTreesRegressor(n_estimators = 150, max_depth = 30, min_samples_leaf = 3, random_state = 20, n_jobs = -1)
ETR_tuned.fit(X_train, y_train)
```

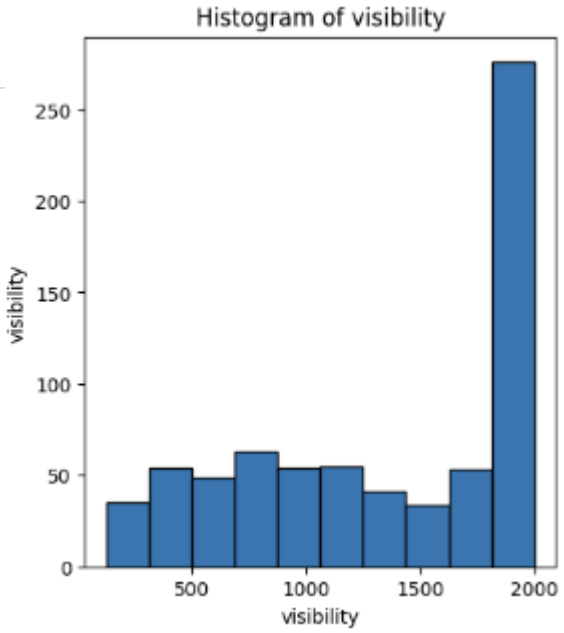
```
ExtraTreesRegressor
ExtraTreesRegressor(max_depth=30, min_samples_leaf=3, n_estimators=150,
                    n_jobs=-1, random_state=20)
```

```
y_pred_ETR_tuned = ETR_tuned.predict(X_test)
```

# Result

다음 회귀분석 프로젝트 때 보완할 점

1. 변수의 중요도 확인
2. 결측치 대체 방식에 대한 재고
-> random forest imputer, iterative imputer, 다중회귀분석을 이용한 결측치 대체 등의 방식을 채택했다면 visibility 제거가 가능했으리라는 한계



# 감사합니다