A+

서울시 따릉이 대여량 예측

홍수지

목차

CONTENTS 01 경진대회 목적 및 규칙

CONTENTS 02 데이터 변수 소개

CONTENTS 03 EDA 및 데이터 전처리

CONTENTS 04 모형 학습 및 검증

CONTENTS 05 결과 해석 및 분석

경진대회 목적 및 규칙

목적 : 주어진 데이터를 바탕으로 따릉이 대여량을 예측

규칙

평가 산식 : RMSE
사용 가능 언어 : Python, R

데이터 변수 소개

* 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

[대회링크](https://dacon.io/competitions/open/235576/data)

EDA 및 데이터 전처리

train 데이터

	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

```
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      1459 non-null   int64
 1   hour                    1459 non-null   int64
 2   hour_bef_temperature    1457 non-null   float64
 3   hour_bef_precipitation  1457 non-null   float64
 4   hour_bef_windspeed      1450 non-null   float64
 5   hour_bef_humidity       1457 non-null   float64
 6   hour_bef_visibility     1457 non-null   float64
 7   hour_bef_ozone          1383 non-null   float64
 8   hour_bef_pm10           1369 non-null   float64
 9   hour_bef_pm2.5          1342 non-null   float64
 10  count                   1459 non-null   float64
dtypes: float64(9), int64(2)
memory usage: 125.5 KB
```

EDA 및 데이터 전처리

test 데이터

	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
0	0	7	20.7	0.0	1.3	62.0	954.0	0.041	44.0	27.0
1	1	17	30.0	0.0	5.4	33.0	1590.0	0.061	49.0	36.0
2	2	13	19.0	1.0	2.1	95.0	193.0	0.020	36.0	28.0
3	4	6	22.5	0.0	2.5	60.0	1185.0	0.027	52.0	38.0
4	5	22	14.6	1.0	3.4	93.0	218.0	0.041	18.0	15.0

```
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      715 non-null    int64
 1   hour                    715 non-null    int64
 2   hour_bef_temperature    714 non-null    float64
 3   hour_bef_precipitation  714 non-null    float64
 4   hour_bef_windspeed      714 non-null    float64
 5   hour_bef_humidity       714 non-null    float64
 6   hour_bef_visibility     714 non-null    float64
 7   hour_bef_ozone          680 non-null    float64
 8   hour_bef_pm10           678 non-null    float64
 9   hour_bef_pm2.5          679 non-null    float64
dtypes: float64(8), int64(2)
memory usage: 56.0 KB
```

EDA 및 데이터 전처리

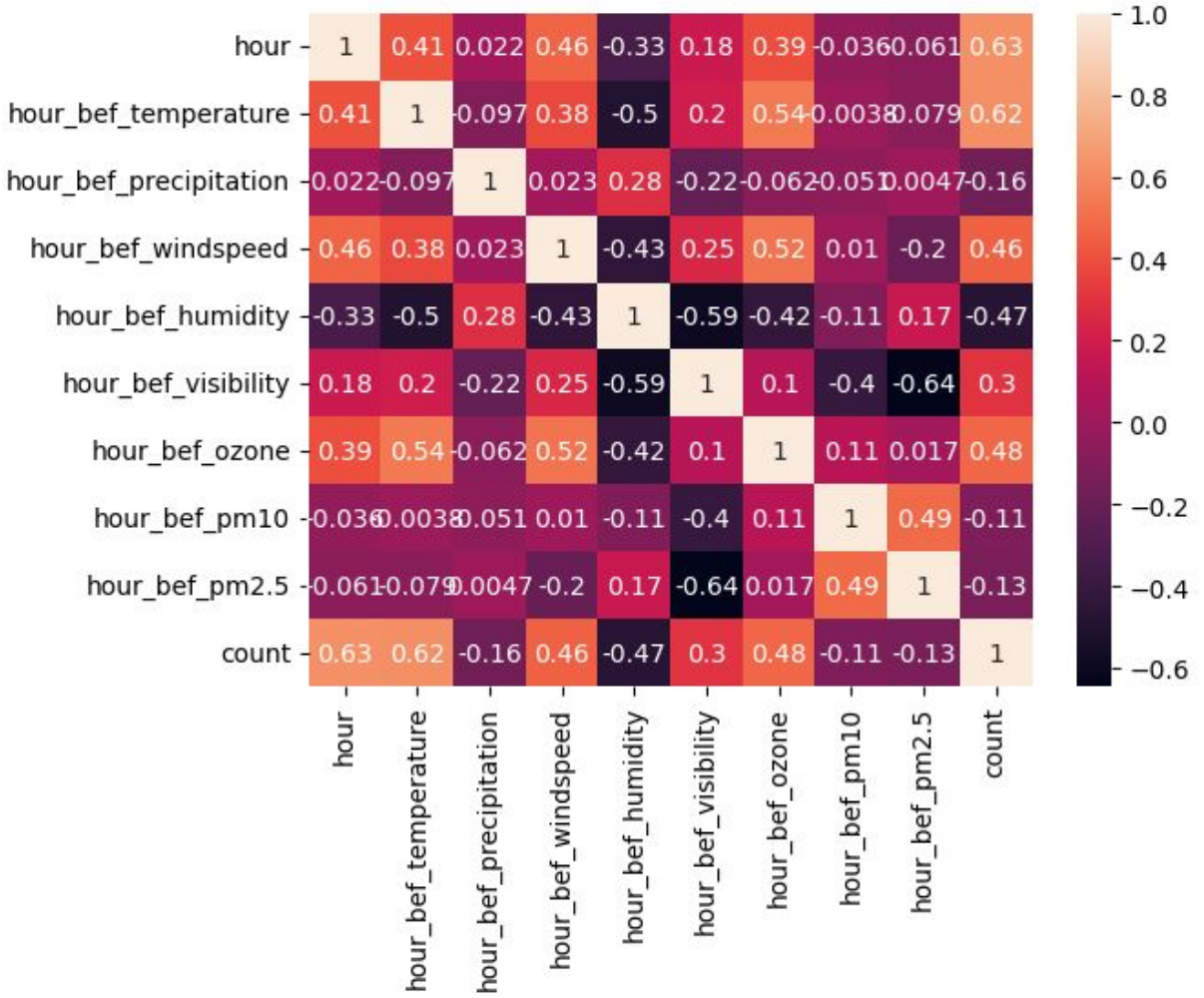
test 데이터

	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
0	0	7	20.7	0.0	1.3	62.0	954.0	0.041	44.0	27.0
1	1	17	30.0	0.0	5.4	33.0	1590.0	0.061	49.0	36.0
2	2	13	19.0	1.0	2.1	95.0	193.0	0.020	36.0	28.0
3	4	6	22.5	0.0	2.5	60.0	1185.0	0.027	52.0	38.0
4	5	22	14.6	1.0	3.4	93.0	218.0	0.041	18.0	15.0

```
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      715 non-null    int64
 1   hour                    715 non-null    int64
 2   hour_bef_temperature    714 non-null    float64
 3   hour_bef_precipitation  714 non-null    float64
 4   hour_bef_windspeed      714 non-null    float64
 5   hour_bef_humidity       714 non-null    float64
 6   hour_bef_visibility     714 non-null    float64
 7   hour_bef_ozone          680 non-null    float64
 8   hour_bef_pm10           678 non-null    float64
 9   hour_bef_pm2.5          679 non-null    float64
dtypes: float64(8), int64(2)
memory usage: 56.0 KB
```

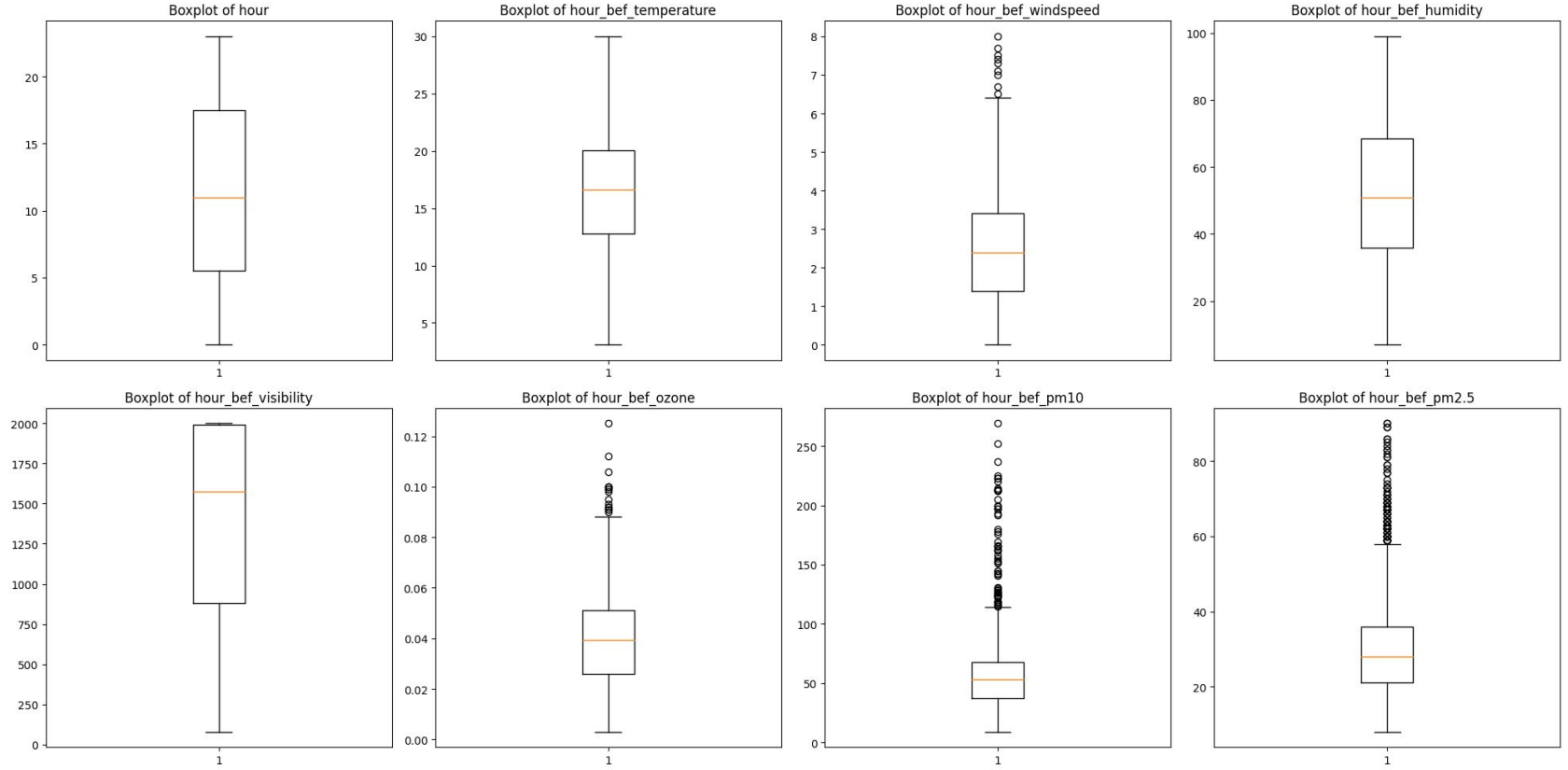
EDA 및 데이터 전처리

상관관계(heat map)



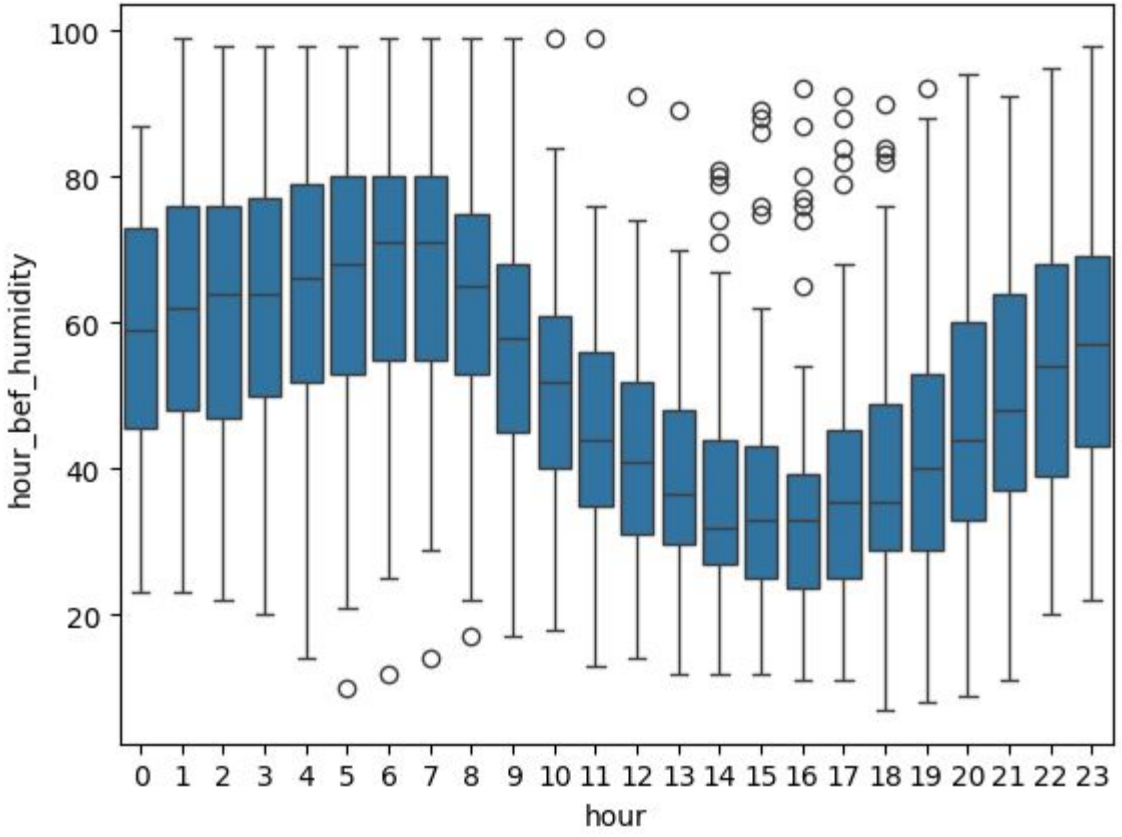
EDA 및 데이터 전처리

박스플롯(Box Plot)



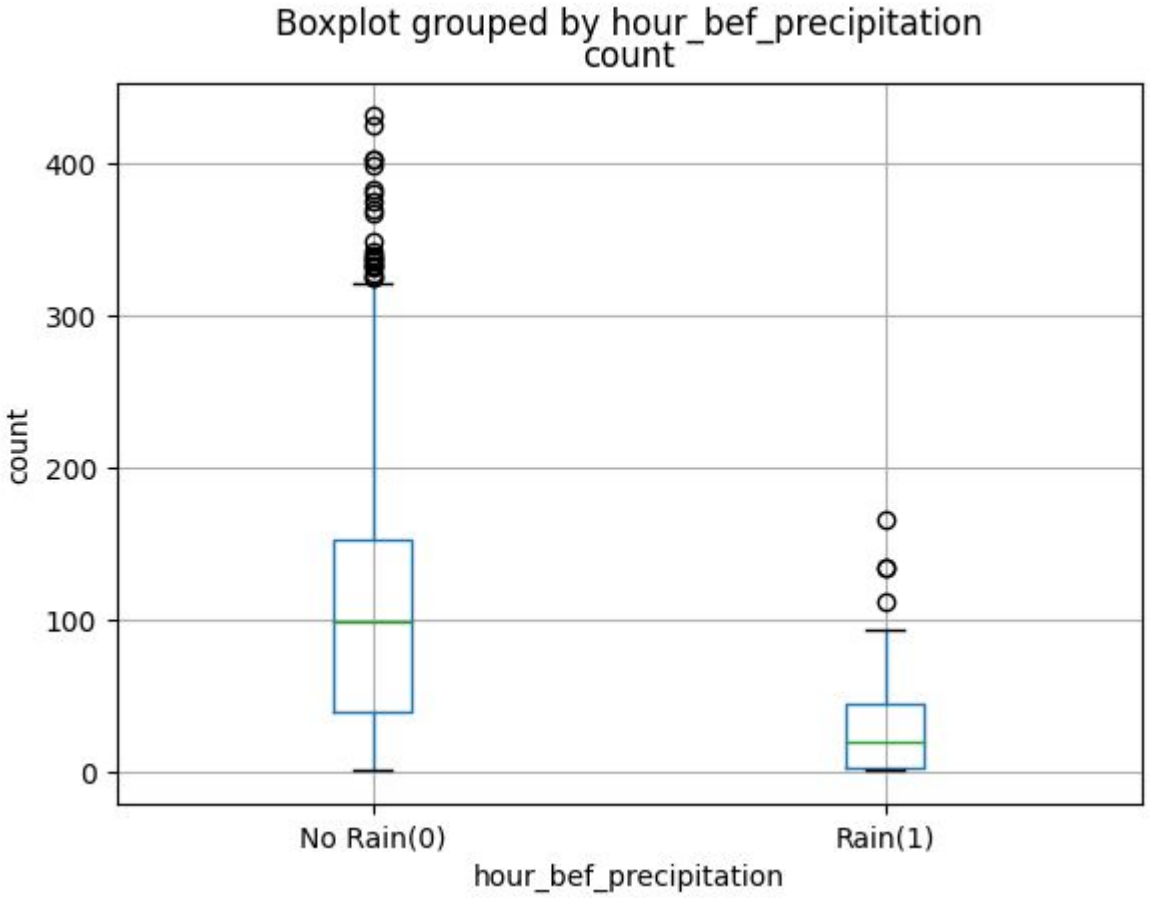
EDA 및 데이터 전처리

시간과 습도에 대한 Boxplot



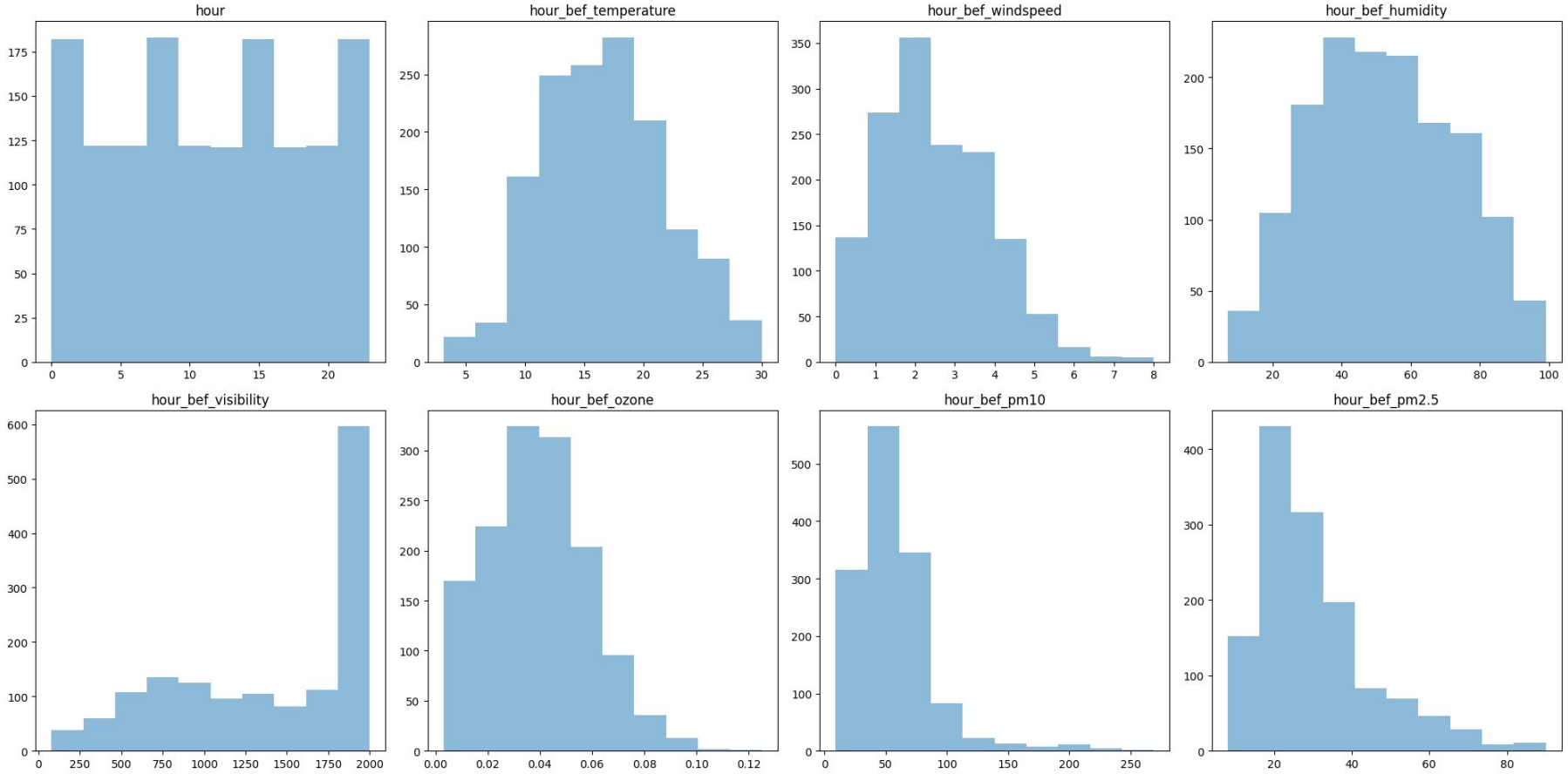
EDA 및 데이터 전처리

비의 유무와 count에 대한 Boxplot



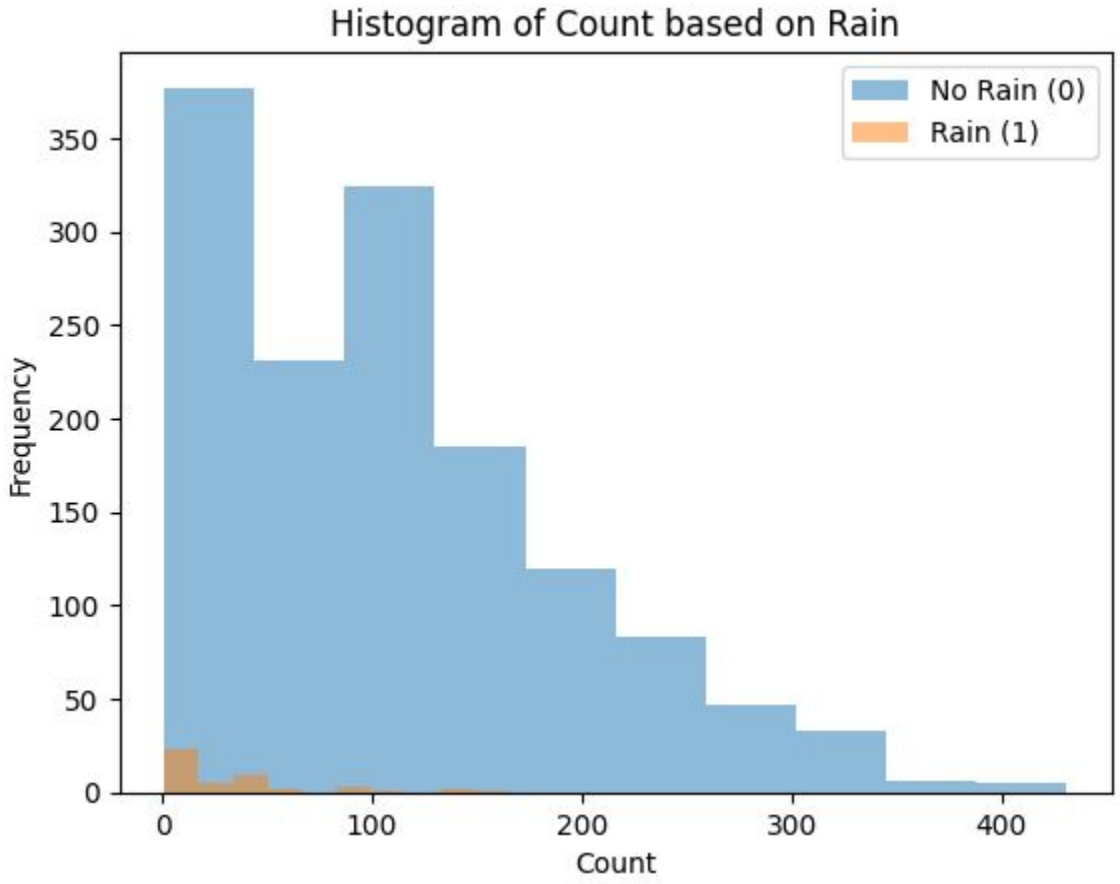
EDA 및 데이터 전처리

히스토 그램(Histogram)



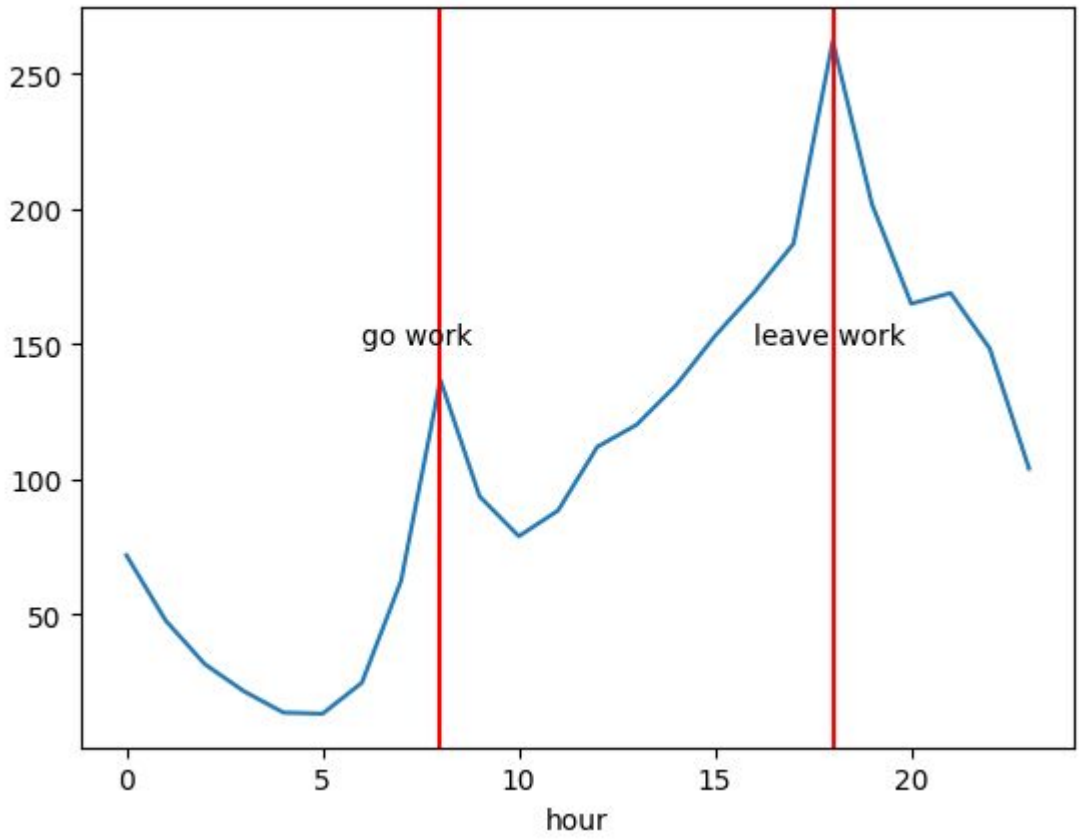
EDA 및 데이터 전처리

비의 유무와 count에 대한 히스토그램



EDA 및 데이터 전처리

시간대별 사용자 수에 대한 line plot



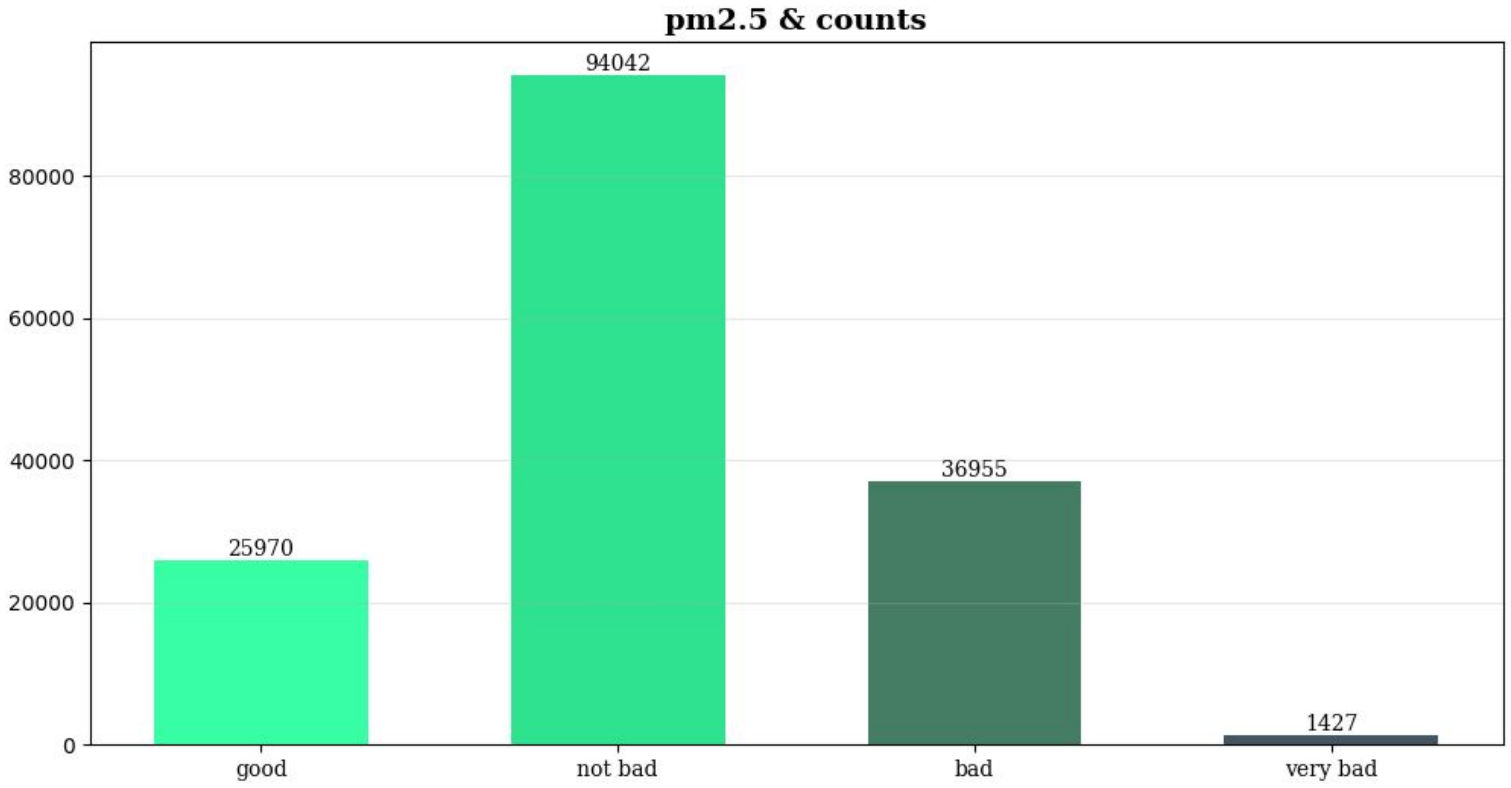
6시부터 10시, 16시부터 20시 사용자수가 급격히 많아진것으로 분류를 해볼 수 있다.

```
#출퇴근 시간 분류
def busyhour(hour):
    if (6<hour<10) or (16<hour<20):
        return 1
    else:
        return 0

train['busyhour'] = train['hour'].apply(busyhour)
test['busyhour'] = test['hour'].apply(busyhour)
```

EDA 및 데이터 전처리

미세먼지에 대한 사용자수 1

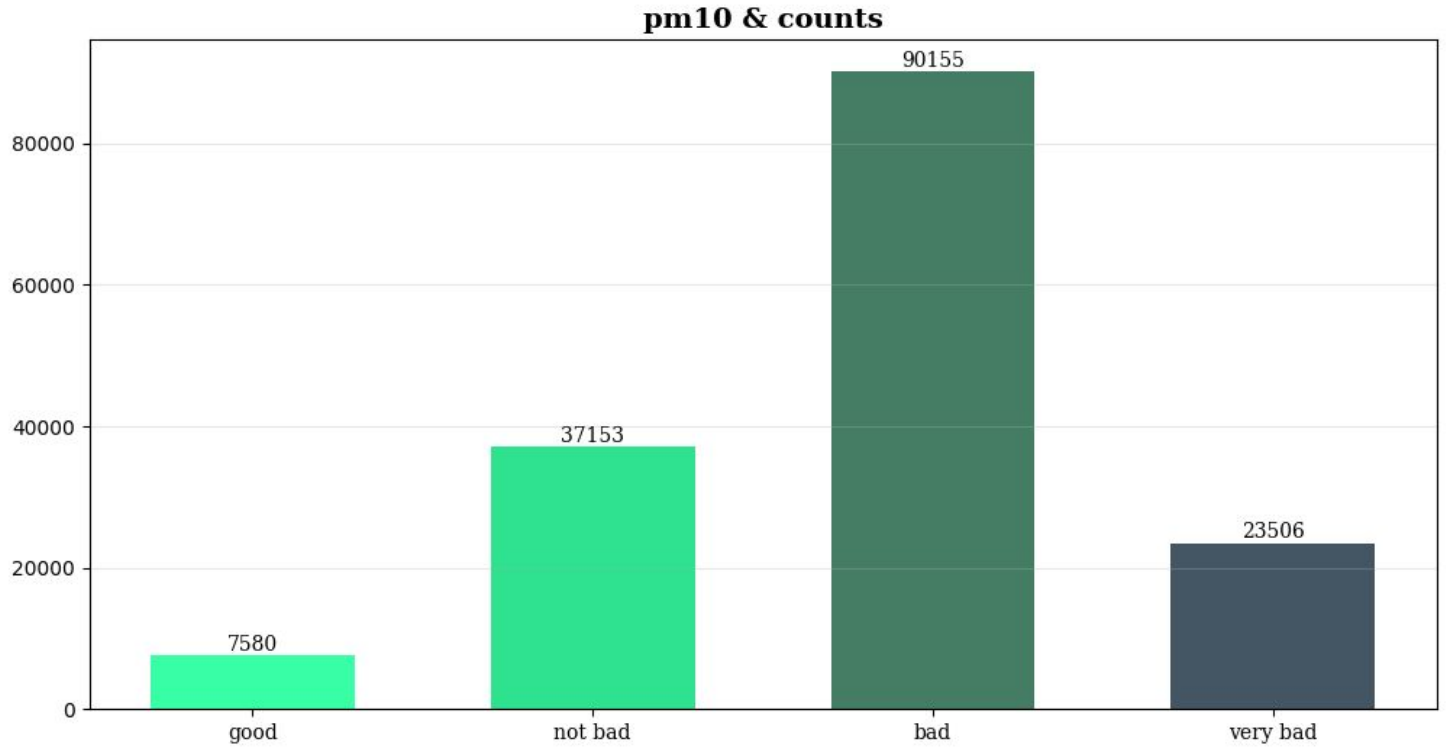


pm2.5 안좋은날 사용자수가 많아 변수를 변경하거나 추가 할 수 있다.

'very bad' :76이상
'bad' :36이상 76미만
'not bad':16이상 36미만
'good' :16미만

EDA 및 데이터 전처리

미세먼지에 대한 사용자수 2



pm2.5 안좋은날 사용자수가 많아 변수를 변경하거나 추가 할 수 있다.

'very bad' :76이상
'bad' :36이상 76미만
'not bad':16이상 36미만
'good' :16미만

EDA 및 데이터 전처리

train 데이터 전처리 방법

1. 결측값처리
2. 이상치 처리
3. 데이터 정규화

EDA 및 데이터 전처리

1. 결측값 처리 방법

(1) fillna 함수 사용

```
[ ]  1 train = train.fillna(0)
```

```
1 train = train.fillna(train.mean())
```

결측값에 0 또는 평균, 중앙값등 대입

1. 결측값 처리 방법

(2) Iterative Imputer

피처 간의 상관관계를 활용하여 결측값을 예측하므로, 특히 피처 간 상관관계가 높은 데이터셋에서 유용합니다.

```python
# 결측값 처리
from sklearn.experimental import enable_iterative_imputer  # IterativeImputer를 사용하기 위해 필요
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge

imputer = IterativeImputer(estimator=BayesianRidge(), max_iter=10, random_state=0)
target = train.pop('count')
for col in train.columns:
    train[col] = imputer.fit_transform(train[[col]])
    test[col] = imputer.transform(test[[col]])
```

EDA 및 데이터 전처리

1. 결측값 처리 방법

(2) Iterative Imputer

Iterative Imputer의 장점

* **더 나은 예측 성능** : 피처들 간의 상관관계를 고려하여 결측값을 예측하기 때문에 단순한 평균이나 중앙값 대체보다 더 정확한 값을 제공할 수 있습니다.
* **다양한 모델 사용 가능** : BayesianRidge 외에도 결정 트리, 랜덤 포레스트 등 다양한 회귀 모델을 사용할 수 있어 데이터 특성에 맞춘 유연한 결측값 처리 가능.
* **다중 대체 가능성** : 반복적 과정에서 여러 번의 예측값을 생성하므로, 다중 대체 방식으로 불확실성을 고려한 분석이 가능합니다.

EDA 및 데이터 전처리

1. 결측값 처리 방법

(2) Iterative Imputer

Iterative Imputer의 단점

* **계산 비용:** 반복적으로 모델을 학습하고 예측해야 하므로, 대규모 데이터셋에서는 시간이 많이 소요될 수 있습니다.
* **복잡성:** 알고리즘이 단순한 대체법보다 복잡하기 때문에 이해하고 설정하는 데 더 많은 노력이 필요합니다.
* **모델 편향:** 잘못된 모델을 선택하면 결측치 대체가 잘못될 수 있습니다. 따라서 데이터에 적합한 모델을 선택하는 것이 중요합니다.

2. 이상치 처리 방법

(1) Winsorize

극단적인 값들을 제거하지 않고, 일정한 범위 내의 값으로 대체하여 데이터의 분포를 조정합니다.

```
[ ]   1 from scipy.stats.mstats import winsorize
      2 train = train.apply(lambda x: winsorize(x, limits=[0.05, 0.05]))
```

2. 이상치 처리 방법

(1) Winsorize

Winsorize의 장점:

* **이상치의 영향 감소** : 평균, 표준편차와 같은 통계적 측정치가 이상치의 영향을 덜 받게 되어 더 안정적인 결과를 얻을 수 있습니다.
* **데이터 손실 방지** : Winsorize는 데이터를 잘라내는 대신 극단적인 값을 대체하기 때문에, 데이터 손실을 최소화할 수 있습니다.
* **분포 왜곡 감소** : 데이터의 분포를 극단적인 값 없이 더 명확하게 파악할 수 있습니다.|

2. 이상치 처리 방법

(1) Winsorize

Winsorize의 단점:

* **데이터의 진짜 특성 왜곡 가능성** : 극단적인 값을 대체함으로써, 원래 데이터의 특성과 분포가 왜곡될 수 있습니다.
* **주관적인 한계 설정** : 어느 비율을 Winsorize할지 결정하는 것은 주관적이며, 적절한 한계 설정이 필요합니다.
* **결과 해석의 어려움** : 통계 결과 해석 시 Winsorize가 적용된 데이터를 사용했다면, 이를 고려하여 해석해야 합니다.

2. 이상치 처리 방법

(2) IQR(Interquartile Range, 사분위 범위)

IQR은 데이터의 1사분위수(Q1, 25번째 백분위수)와 3사분위수(Q3, 75번째 백분위수)의 차이를 말합니다.

```
# IQR
train['count'] = target
for col in train.columns:
    Q1 = train[col].quantile(0.25)
    Q3 = train[col].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1-1.5*IQR
    upper = Q3+1.5*IQR
    train_iqr = train[(train[col]<lower)|(train[col]>upper)]
    train = train.drop(train_iqr.index,axis=0)
```

2. 이상치 처리 방법

(2) IQR(Interquartile Range, 사분위 범위)

IQR 이상치 처리의 장점

* **간단함**: IQR을 사용하는 방법은 계산이 간단하고 직관적입니다.
* **비모수적 방법** : 데이터가 특정 분포를 따르지 않아도 적용할 수 있어 유연합니다.
* **이상치의 영향 감소** : 이상치를 처리함으로써 평균이나 분산 같은 통계량이 이상치에 의해 왜곡되는 것을 방지합니다.

2. 이상치 처리 방법

(2) IQR(Interquartile Range, 사분위 범위)

IQR 이상치 처리의 단점

***정보 손실:** 이상치를 제거하거나 대체함으로써 데이터의 중요한 정보가 손실될 수 있습니다.

***임의성:** IQR과 1.5 배수를 사용한 이상치 기준은 임의적일 수 있으며, 특정 분석에 적합하지 않을 수 있습니다.

.

3. 데이터 정규화

(1) Standard Scaler

Standard Scaler는 데이터의 평균을 0, 표준편차를 1로 맞추어 정규화합니다.

```
# StandatdScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train = pd.DataFrame(scaler.fit_transform(train), columns=train.columns)
test = pd.DataFrame(scaler.transform(test), columns=test.columns)
```

3. 데이터 정규화

(1) Min-MaxScaler

MinMax Scaler는 데이터를 주어진 최소값과 최대값 사이로 스케일링합니다.

보통 0과 1 사이로 정규화합니다.

```
# MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
train = pd.DataFrame(scaler.fit_transform(train), columns=train.columns)
test = pd.DataFrame(scaler.transform(test), columns=test.columns)
```

모형 학습 및 검증

1. Linear Regression :43.514

```
1 from sklearn.linear_model import LinearRegression
2 lr = LinearRegression()
3 lr.fit(x_tr,y_tr)
4 pred = lr.predict(x_val)
5
6 print(rmse(y_val, pred))
7
```

```
43.51417799033894
```

모형 학습 및 검증

2. Logistic Regression :63.434

```
#LogisticRegression
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
log.fit(x_tr,y_tr)
pred = log.predict(x_val)


# 평가
print(rmse(y_val, pred))
```

```
63.433921116214 61
```

모형 학습 및 검증

3. RandomForest Regression: 36.735

```
1 from sklearn.ensemble import RandomForestRegressor
2 rf = RandomForestRegressor(n_estimators=200, max_depth=15, random_state=2023)
3 rf.fit(x_tr,y_tr)
4 pred = rf.predict(x_val)
5
6 print(rmse(y_val, pred))
```

```
36.73478424655903
```

모형 학습 및 검증

4. LightGBM:37.270

```python
from lightgbm import LGBMRegressor
lgbm = LGBMRegressor()
lgbm.fit(x_tr,y_tr)
pred = lgbm.predict(x_val)

print(rmse(y_val, pred))
```

```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000193 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 550
[LightGBM] [Info] Number of data points in the train set: 986, number of used features: 8
[LightGBM] [Info] Start training from score 109.134888
37.26970935899319
```

모형 학습 및 검증

5. XGBoost : 39.537

```
1 from xgboost import XGBRegressor
2 xgb = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, objective='reg:squarederror',random_state=110)
3 xgb.fit(x_tr,y_tr)
4 pred = xgb.predict(x_val)
```

```
39.53703830239874
```

모형 학습 및 검증

6. Gradient Boosting : 40.253

```
1 # GradientBoosting
2 from sklearn.ensemble import GradientBoostingRegressor
3 gb = GradientBoostingRegressor(n_estimators=100)
4 gb.fit(x_tr,y_tr)
5
6 pred = gb.predict(x_val)
7 print(rmse(y_val, pred))
```

```
40.25310393630883
```

모형 학습 및 검증

7. AdaBoosting : 42.910

```
1 from sklearn.ensemble import AdaBoostRegressor
2 ada = AdaBoostRegressor(n_estimators=50, random_state=110)
3 ada.fit(x_tr,y_tr)
4
5 pred = ada.predict(x_val)
6 print(rmse(y_val, pred))
```

```
42.910450714795125
```

모형 학습 및 검증

8. Decision Tree ：44.420

```
#DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
dtc = DecisionTreeRegressor(max_depth=5, min_samples_split=2, min_impurity_decrease=0.0)
dtc.fit(x_tr, y_tr)
pred = dtc.predict(x_val)
print(rmse(y_val, pred))
```

```
44.42039759734831
```

모형 학습 및 검증

9. Bagging : 42.910

```
# Bagging
from sklearn.ensemble import BaggingRegressor
base_model = RandomForestRegressor()
bag = BaggingRegressor(estimator=base_model,
                       n_estimators=300,
                       bootstrap=True,
                       n_jobs=-1,
                       random_state=110)
bag.fit(x_tr,y_tr)
pred = bag.predict(x_val)
print(rmse(y_val, pred))
```

```
36.892133196803 65
```

모형 학습 및 검증

9. Bagging : 42.910

```
# Bagging
from sklearn.ensemble import BaggingRegressor
base_model = RandomForestRegressor()
bag = BaggingRegressor(estimator=base_model,
                       n_estimators=300,
                       bootstrap=True,
                       n_jobs=-1,
                       random_state=110)
bag.fit(x_tr,y_tr)
pred = bag.predict(x_val)
print(rmse(y_val, pred))
```

```
36.892133196803 65
```

결과 해석 및 분석

현재 점수는 44.615로 다양한 방법을 조합하여 다른 점수를 낼 수 있다

2024-08-28 09:46:37 44.6147126697

pycaret의 다양한 기능을 이용하여 여러 가지 회귀 모델을 쉽게 생성하고 비교할 수 있습니다.

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	24.8809	1311.9799	36.0890	0.7854	0.4199	0.4279	0.2980
lightgbm	Light Gradient Boosting Machine	25.0643	1352.7758	36.4861	0.7790	0.4352	0.4280	0.6140
xgboost	Extreme Gradient Boosting	25.0877	1368.9240	36.8784	0.7765	0.4298	0.3977	0.1380
rf	Random Forest Regressor	25.7269	1397.5426	37.2067	0.7719	0.4219	0.4304	0.6810
gbr	Gradient Boosting Regressor	26.1621	1457.4649	37.9752	0.7610	0.4435	0.4595	0.1620
lr	Linear Regression	34.8424	2084.7624	45.5594	0.6580	0.6589	0.7345	1.2760
lar	Least Angle Regression	34.8424	2084.7616	45.5594	0.6580	0.6589	0.7345	0.0380
ada	AdaBoost Regressor	36.5918	2089.0494	45.6302	0.6574	0.6355	0.8761	0.1370
ridge	Ridge Regression	35.0168	2097.7645	45.7067	0.6561	0.6602	0.7274	0.0470
br	Bayesian Ridge	35.0672	2103.3408	45.7665	0.6553	0.6631	0.7275	0.0290
llar	Lasso Least Angle Regression	35.0729	2108.8075	45.8231	0.6547	0.6625	0.7270	0.0310
lasso	Lasso Regression	35.0728	2108.8048	45.8231	0.6547	0.6625	0.7270	0.0410
huber	Huber Regressor	34.6232	2132.7930	46.0997	0.6510	0.6654	0.6812	0.0880
en	Elastic Net	37.0998	2396.4305	48.7958	0.6096	0.6744	0.7683	0.0450
dt	Decision Tree Regressor	32.6643	2523.9388	49.9720	0.5863	0.5231	0.4601	0.0510
knn	K Neighbors Regressor	38.5640	2738.8530	52.0866	0.5543	0.6760	0.8168	0.0530
par	Passive Aggressive Regressor	48.1270	3351.9645	57.5208	0.4524	0.8380	1.3542	0.0270
omp	Orthogonal Matching Pursuit	56.0585	4975.1479	70.4038	0.1885	0.8858	1.4249	0.0250
dummy	Dummy Regressor	63.6039	6252.5797	79.0027	-0.0204	1.0117	1.9440	0.0220

Thank you