2조 팀 프로젝트 결과 보고서

KUBIG ML SESSION

팀원 : 김유민, 박재찬, 오화진, 제갈예빈

01. 데이터 소개

KUBIG ML SESSION 2조

사용 데이터

서울시 마포구의 날짜별, 시간별 기상상황과 따릉이 대여 수

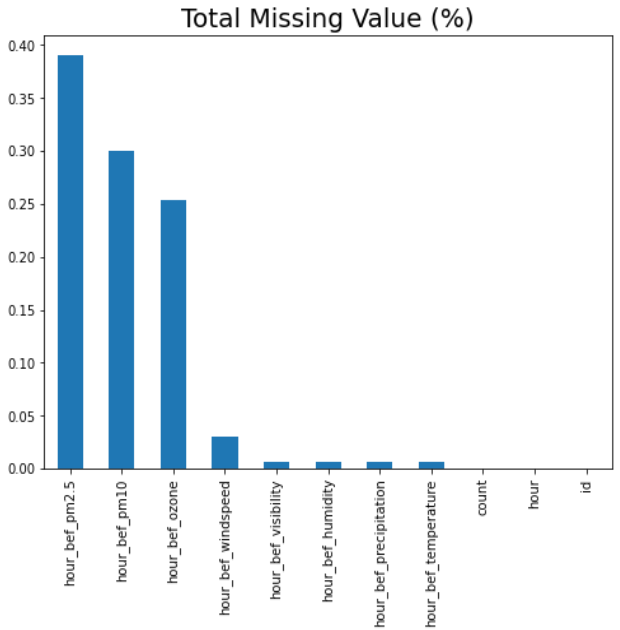
변수 설명

- id :날짜와 시간 별 id
- hour : 시간
- hour_bef_temperature : 1시간 전 기온
- hour_bef_precipation : 1시간 전 비 정보, 비가 오지 않았으면 0, 비가 오면 1 => dummy!
- hour_bef_windspeed : 1시간 전 풍속(평균)
- hour_bef_humidity : 1시간 전 습도
- hour_bef_visibility : 1시간 전 시정(視程), 시계(視界)(특정 기상 상태에 따른 가시성을 의미)
- hour_bef_ozone : 1시간 전 오존
- hour_bef_pm10 : 미세먼지(머리카락 굵기의 1/5에서 1/7 크기의 미세먼지)
- hour_bef_pm2.5 : 미세먼지(머리카락 굵기의 1/20에서 1/30 크기의 미세먼지)
- count : 시간에 따른 따릉이 대여 수

목표

2017년 4월 1일부터, 5월 31일까지 각 날짜의 1시간 전의 기상상황을 기반으로
1시간 후의 따릉이 대여 수 예측하기

결측치 처리



- 결측치 개수가 10 미만인 변수(온도,강수량,풍속, 습도,가시성)의 행은 제거
- 새벽 한 시일 때 항상 오존 및 미세먼지 변수 (pm 10, pm 2.5)가 측정X

→ 새벽 한 시 데이터 포함/미포함 경우의 수 고려

오존, 미세먼지 변수(pm 2.5, pm 10) 결측치 처리 알고리즘

① Simple Imputation

- 평균값, 중앙값, 최빈값 등을 활용하는 방식

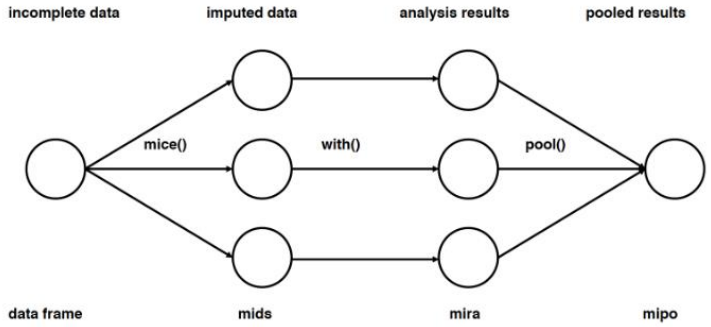
② KNN Imputation

- 주변 이웃의 값을 이용하여 거리에 따라 가중치를 두어 결측값을 메꾸는 방식

```
def optimize_k(data, target):
    errors = []
    num = []
    for k in range(1, 20, 2):
        minMaxScaler = MinMaxScaler()
        minMaxScaler.fit(data)
        data_minMaxScaled = minMaxScaler.transform(data)
        imputer = KNNImputer(n_neighbors=k)
        imputed = imputer.fit_transform(data_minMaxScaled)
        data_imputed = pd.DataFrame(imputed, columns=data.columns)

        X = data_imputed.drop(target, axis=1)
        y = data_imputed[target]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

        model = RandomForestRegressor(random_state=0)
        model.fit(X_train, y_train)
        preds = model.predict(X_test)
        error = rmse(y_test, preds)
        errors.append(error)
        num.append(k)

    df = pd.DataFrame({'K' : num, 'RMSE' : errors})
    df = df.sort_values('RMSE', ascending = True)
    return df
```

③ MICE Imputation (Multiple Iterative by Chained equations)

- 처음에는 각 컬럼에 대한 평균값으로 결측치를 메꾸고
- 이후에 불완전한 데이터셋을 회귀에 적합한 후 바로 직전의 데이터셋과의 차이를 구함. 이들의 차이가 최소가 될 때까지 이 과정을 반복하는 방식



오존, 미세먼지 변수(pm 2.5, pm 10) 결측치 처리 알고리즘

	in_KNN imp	in_MICE imp	in_Simple Imp	out_KNN imp	out_MICE imp	out_Simple Imp
R_adj	0.597796	0.598682	0.59807	0.592168	0.592574	0.591512

새벽 1시 데이터 포함 & MICE Imputation 활용

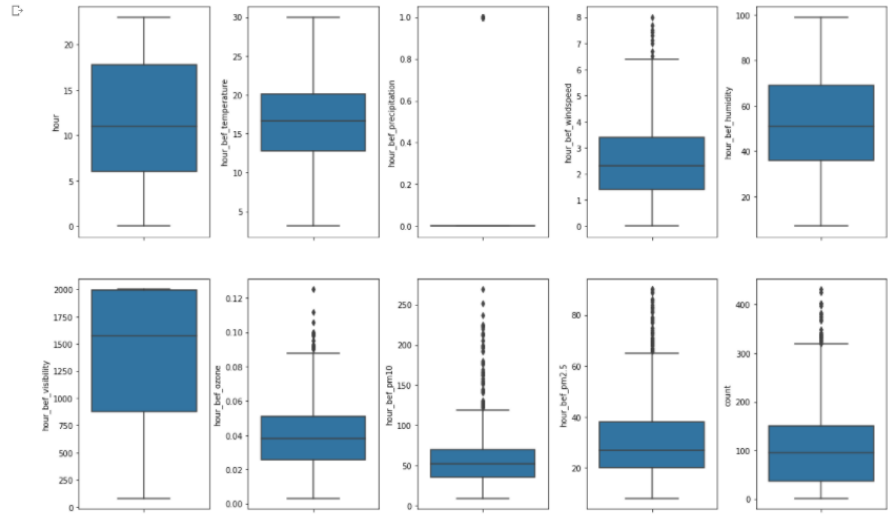
이상치 처리

01

Simplified method

InterQuartil Range(IQR)을 활용하여
Q1-1.5*IQR과 Q3+1.5*1QR를 벗어난 값을 이상치로 간주하여 제거

```
# 이상치 확인
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

fig, axs = plt.subplots(ncols=5, nrows=2, figsize=(15,9))
index = 0
axs = axs.flatten()
for k,v in df_null.items():
    sns.boxplot(y=k, data=df_null, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



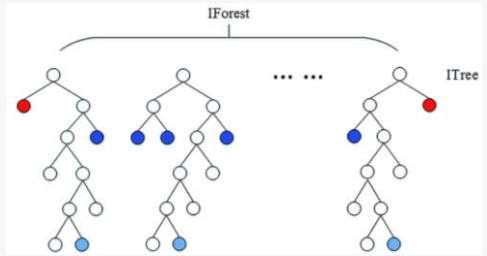

```
Q1 = df_null.quantile(0.25)
Q3 = df_null.quantile(0.75)
IQR = Q3 - Q1
df_use = df_null[['hour','hour_bef_temperature','hour_bef_windspeed','hour_bef_humidity','hour_bef_visibility','hour_bef_ozone','hour_bef_pm10','hour_bef_pm2.5']]
df_out = df_null[~((df_use < (Q1 - 1.5 * IQR)) |(df_use > (Q3 + 1.5 * IQR))).any(axis=1)]
df_out.shape
```

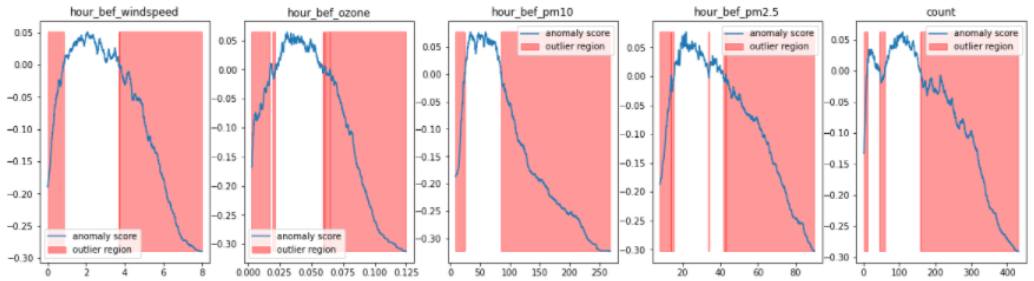
02

Automated method

① Isolation Forest
- 트리 모델 기반으로 (min-max 사이에서) 더 이상 구분되지 않는 짧은 path를 가진 값을 이상치로 간주
- 이를 기반으로 한 anomaly score가 특정 threshold 이하 인 값을 이상치로 간주하여 제거

```
[ ]  1 #Import necessary libraries
     2 from sklearn.ensemble import IsolationForest
     3 #The required columns
     4 cols = ['hour_bef_windspeed','hour_bef_ozone','hour_bef_pm10', 'hour_bef_pm2.5','count']
     5
     6
     7 #Plotting the sub plot
     8 fig, axs = plt.subplots(1, 5, figsize=(20, 5), facecolor='w', edgecolor='k')
     9 axs = axs.ravel()
    10
    11 for i, column in enumerate(cols):
    12     isolation_forest = IsolationForest(contamination='auto')
    13     isolation_forest.fit(df_null[column].values.reshape(-1,1))
    14
    15     xx = np.linspace(df_null[column].min(), df_null[column].max(), len(df_null)).reshape(-1,1)
    16     anomaly_score = isolation_forest.decision_function(xx)
    17     outlier = isolation_forest.predict(xx)
    18
    19     axs[i].plot(xx, anomaly_score, label='anomaly score')
    20     axs[i].fill_between(xx.T[0], np.min(anomaly_score), np.max(anomaly_score),
    21                         where=outlier==-1, color='r',
    22                         alpha=.4, label='outlier region')
    23     axs[i].legend()
    24     axs[i].set_title(column)
```






```
[ ]  1 # isolation forest
     2
     3 from sklearn.linear_model import LinearRegression
     4 from sklearn.ensemble import IsolationForest
     5 from sklearn.metrics import mean_absolute_error
     6
     7 # identify outliers in the training dataset
     8 iso = IsolationForest(contamination=0.01)
     9 yhat = iso.fit_predict(X_train)
    10 # select all rows that are not outliers
    11 mask = yhat != -1
    12 X_train, y_train = X_train[mask, :], y_train[mask]
    13 # summarize the shape of the updated training dataset
    14 print(X_train.shape, y_train.shape)
    15 # fit the model
    16 model = LinearRegression()
    17 model.fit(X_train, y_train)
    18 # evaluate the model
    19 yhat = model.predict(X_test)
    20 # evaluate predictions
    21 mae = mean_absolute_error(y_test, yhat)
    22 print('MAE: %.3f' % mae)

(865, 9) (865,)
MAE: 39.930
```

02

Automated method

```
[ ]  1 from sklearn.linear_model import LinearRegression
     2 from sklearn.neighbors import LocalOutlierFactor
     3 from sklearn.metrics import mean_absolute_error
     4 # split into train and test sets
     5 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
     6 # summarize the shape of the training dataset
     7 print(X_train.shape, y_train.shape)
     8 # identify outliers in the training dataset
     9 lof = LocalOutlierFactor(n_neighbors=20,contamination=0.01)
    10 yhat = lof.fit_predict(X_train)
    11 # select all rows that are not outliers
    12 mask = yhat != -1
    13
    14 X_train, y_train = X_train[mask, :], y_train[mask]
    15
    16 # summarize the shape of the updated training dataset
    17 print(X_train.shape, y_train.shape)
    18 # fit the model
    19 model = LinearRegression()
    20 model.fit(X_train, y_train)
    21 # evaluate the model
    22 yhat = model.predict(X_test)
    23 # evaluate predictions
    24 mae = mean_absolute_error(y_test, yhat)
    25 print('MAE: %.3f' % mae)

(971, 9) (971,)
(961, 9) (961,)
MAE: 39.786
```

② Local Outlier Factor (LOF)

- 관측치 주변의 밀도와 근접한 관측치 주변의 밀도의 상대적 비교를 통한 이상치 탐색

⇒ 1. 가장 데이터 손실이 적고, 2. LOF가 선형회귀 결과 오차(mae)가 적어서 automated 방법 중에선 LOF로 결정.

⇒ 이상치 처리 X / IQR 방법 / LOF 방법 세 가지 적용

HOUR 변수

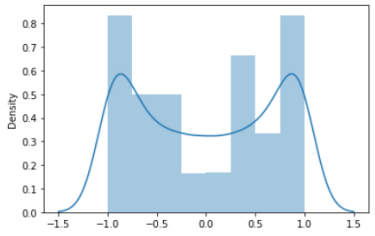
CASE 1

hour변수 그대로 사용

CASE 2

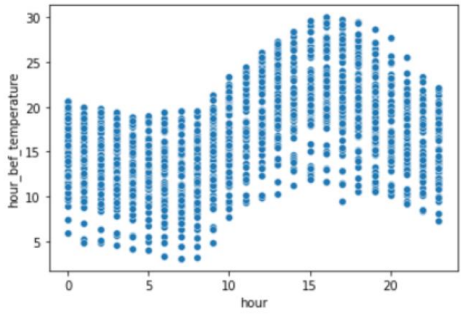
Hour(0-23시) Sin 변환

```
[4]  df['hour'].unique()

     array([20., 13.,  6., 23., 18.,  2.,  3., 21.,  9., 14.,  4., 10.,  1.,
            17.,  8., 16.,  0.,  7., 15., 19., 22., 11.,  5., 12.])

[5]  hour_transform=np.array(df['hour'])+1

[6]  hour_transform_sin=np.sin(hour_transform*2*np.pi/24)
     sns.distplot(hour_transform_sin)

     /usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: Futur
       warnings.warn(msg, FutureWarning)
     <matplotlib.axes._subplots.AxesSubplot at 0x7f0090dde610>
```

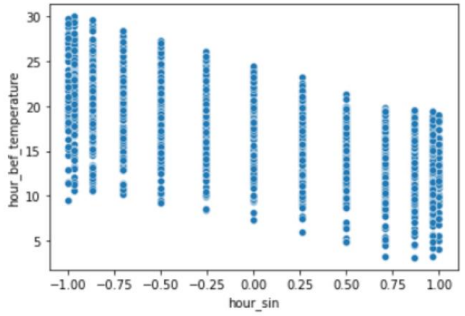


시간 sin 변환 후 linear 한 패턴 확인 가능

```
[ ]  sns.scatterplot(df['hour'],df['hour_bef_temperature'])

     /usr/local/lib/python3.7/dist-packages/seaborn/_decorators.p
       FutureWarning
     <matplotlib.axes._subplots.AxesSubplot at 0x7f84e0c65a10>
```




```
[ ]  sns.scatterplot(df['hour_sin'],df['hour_bef_temperature'])

     /usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43:
       FutureWarning
     <matplotlib.axes._subplots.AxesSubplot at 0x7f84e0c65fd0>
```

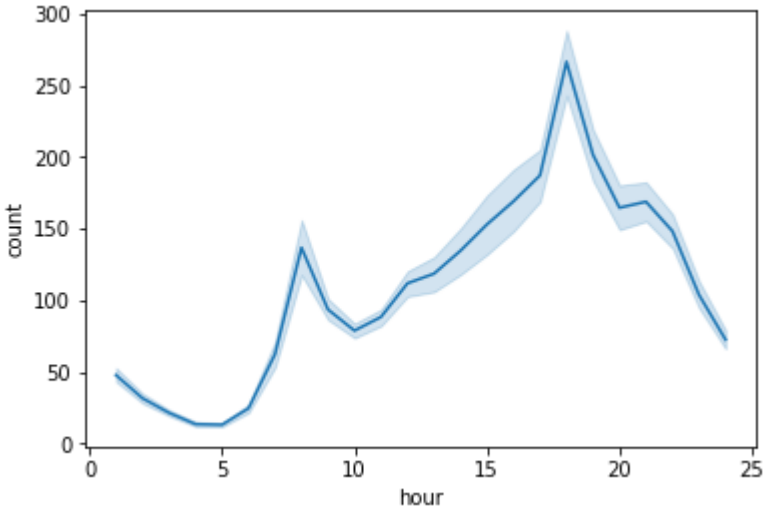


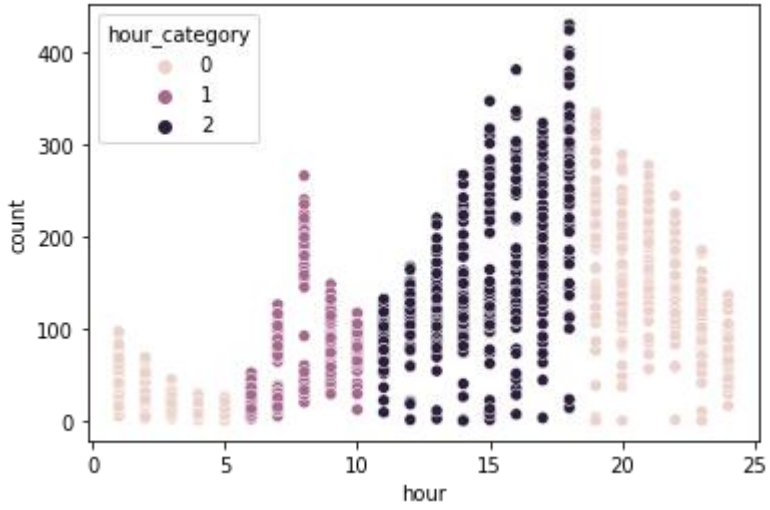
HOUR 변수

CASE 3

범주형 변수,더미 변수로 변환

Hour 별 증감/peak 를 eda한 후, 세 개의 구간으로 나누어 범주형 변수로 변환



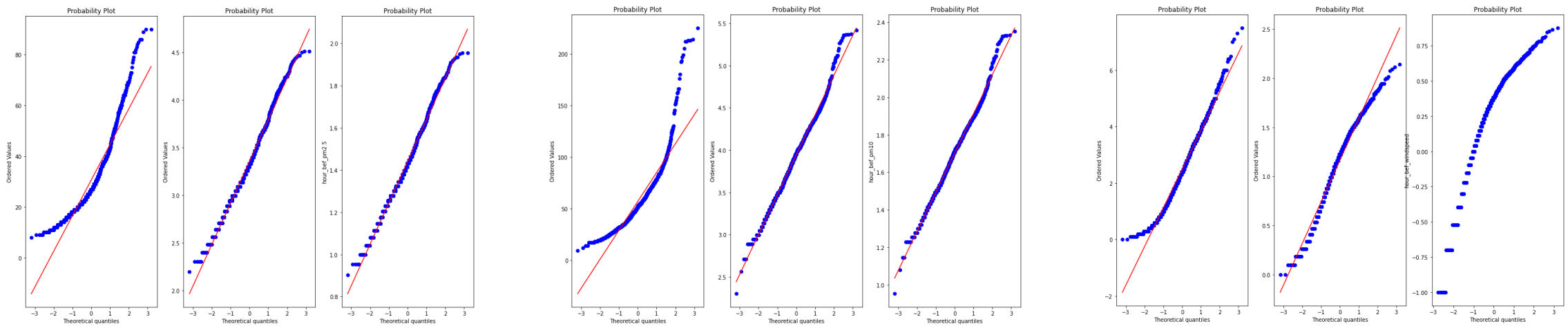


Log 스케일링

windspeed, pm10, pm2.5 정도만 로그변환의 효과가 나타남 (visibility 다른 방법 필요)

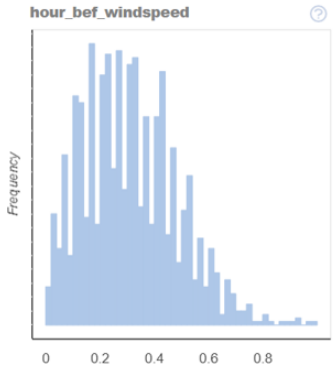
```
import seaborn as sns
for i in [0,2,4,5,6,7,8,9] :
  print('{:15}'.format(X_train.columns[i]),
        'Skewness: {:05.2f}'.format(X_train[X_train.columns[i]].skew()) ,
        '   ' ,
        'Kurtosis: {:06.2f}'.format(X_train[X_train.columns[i]].kurt())
       )
```

```
id              Skewness: -0.01     Kurtosis: -01.13
hour_bef_temperature Skewness: 00.17     Kurtosis: -00.45
hour_bef_windspeed Skewness: 00.46     Kurtosis: -00.10
hour_bef_humidity Skewness: 00.17     Kurtosis: -00.80
hour_bef_visibility Skewness: -0.47     Kurtosis: -01.22
hour_bef_ozone  Skewness: 00.41     Kurtosis: 000.18
hour_bef_pm10   Skewness: 02.10     Kurtosis: 006.82
hour_bef_pm2.5  Skewness: 01.32     Kurtosis: 001.79
```

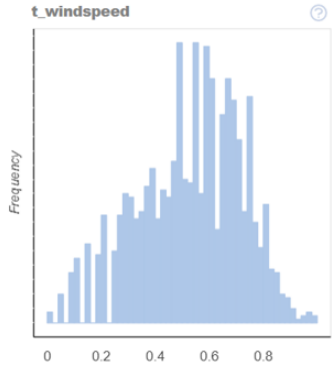


skewness features 변환

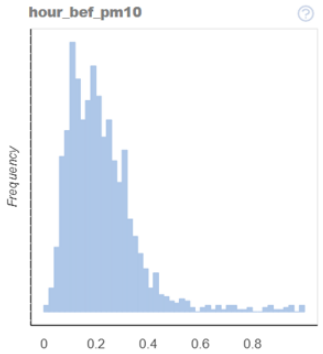
- windspeed, pm2.0, pm10, count (y)



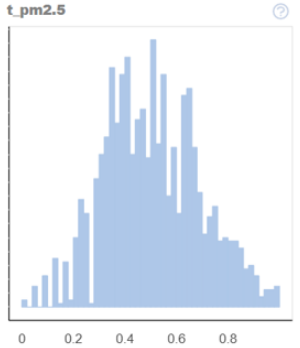
windspeed
[log 변환 전]



windspeed
[log 변환 후]



pm10
[log 변환 전]



pm10
[log 변환 후]

=> Right skewed 된 feature 3가지를 log 변환

MinMaxScaling

```
from sklearn.model_selection import cross_val_score
# Ridge의 alpha값을 다르게 적용하고 다양한 데이터 변환방법에 따른 RMSE 추출.
alphas = [0.1, 1, 10, 100]
#변환 방법은 모두 6개, 원본 그대로, 표준정규분포, 표준정규분포+다항식 특성
# 최대/최소 정규화, 최대/최소 정규화+다항식 특성, 로그변환
scale_methods=[(None, None), ('Standard', None),('MinMax', None), ('Log', None)]
for scale_method in scale_methods:
    X_data_scaled = get_scaled_data(method=scale_method[0], p_degree=scale_method[1],
                                    input_data=X_train)
    print('₩n## 변환 유형:{0}, Polynomial Degree:{1}'.format(scale_method[0], scale_method[1]))
    get_linear_reg_eval('Ridge', params=alphas, X_data_n=X_data_scaled,
                        y_target_n=y_train, verbose=False)
```

```
## 변환 유형:None, Polynomial Degree:None
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 52.723
alpha 1일 때 5 폴드 세트의 평균 RMSE: 52.869
alpha 10일 때 5 폴드 세트의 평균 RMSE: 52.980
alpha 100일 때 5 폴드 세트의 평균 RMSE: 53.396

## 변환 유형:Standard, Polynomial Degree:None
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 52.728
alpha 1일 때 5 폴드 세트의 평균 RMSE: 52.727
alpha 10일 때 5 폴드 세트의 평균 RMSE: 52.716
alpha 100일 때 5 폴드 세트의 평균 RMSE: 52.826

## 변환 유형:MinMax, Polynomial Degree:None
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 52.724
alpha 1일 때 5 폴드 세트의 평균 RMSE: 52.706
alpha 10일 때 5 폴드 세트의 평균 RMSE: 53.237
alpha 100일 때 5 폴드 세트의 평균 RMSE: 61.815

## 변환 유형:Log, Polynomial Degree:None
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 56.123
alpha 1일 때 5 폴드 세트의 평균 RMSE: 56.660
alpha 10일 때 5 폴드 세트의 평균 RMSE: 57.021
alpha 100일 때 5 폴드 세트의 평균 RMSE: 59.442
```

1) None 2) Standard 3) MinMax 4)Log 스케일링 방법 비교 결과,
최종적으로 Minmaxscaler 사용 결정!

모델링

01

단일 모델 사용

XGBoost,LGBM등을 사용해서 추가된 다양한 feature들의 유효성 확인

→ Feature engineering을 통해 변형한 Hour변수들은 제외하기로

→ Log변환을 통해 추가한 변수들은 model마다 다른 결론

XGBoost

```
X_train_select=X_train[select]
X_val_select=X_val[select]
```

```
reg=XGBRegressor()
reg.fit(X_train_select,y_train_count)
y_pred=reg.predict(X_test[select])
```

LGBM

```
[ ]  import lightgbm as lgb
     select=list(set(cols)-set(['hour_sin','hour_category','hour_category_1','hour_category_2','t_pm2.5','t_pm10','t_windspeed','hour_bef_visibility']))

     X_train_select=X_train[select]
     X_val_select=X_val[select]

     reg=lgb.LGBMRegressor()
     reg.fit(X_train_select,y_train_count)
     print(reg.score(X_val_select,y_val_count))
```

```
0.8010530337526112
```

02

AutoML 사용

auto-sklearn/H2O/TPOT 사용

→ TPOT이 가장 좋은 성능을 보임

```
aml= H2OAutoML(sort_metric='rmse')
aml.train(x=x,y=y,training_frame=train_select)
lb=aml.leaderboard
print(lb)
print('generate predictions')
test_y=aml.leader.predict(test)
test_y=test_y.as_data_frame()
```

H2O

```
AutoML progress: |████████████████████████████████████████████████| 100%
```

model_id	rmse	mean_residual_deviance	mse	mae	rmsle
StackedEnsemble_AllModels_AutoML_20210901_132802	33.8915	1148.63	1148.63	23.5407	nan
StackedEnsemble_BestOfFamily_AutoML_20210901_132802	34.6915	1203.5	1203.5	24.3945	nan
GBM_3_AutoML_20210901_132802	37.1589	1380.79	1380.79	25.9824	nan
GBM_4_AutoML_20210901_132802	37.2723	1389.23	1389.23	25.6857	nan
GBM_2_AutoML_20210901_132802	37.317	1392.56	1392.56	25.8049	nan

```
from tpot import TPOTRegressor
tpot = TPOTRegressor(n_jobs=-1, verbosity=3,periodic_checkpoint_folder="tpot_results_no.txt")
tpot.fit(X_train, y_train)
print(tpot.score(X_val, y_val))
```

TPOT

```
1 import autosklearn.regression
2 reg = autosklearn.regression.AutoSklearnRegressor()
3 select=list(set(cols)-set(['hour_sin','hour_category','hour_category_1','hour_category_2','t_pm2.5','t_pm10','t_windspeed','hour_bef_visibility']))
4
5
6 X_train_select=X_train[select]
7 X_val_select=X_val[select]
8
9 reg.fit(X_train_select,y_train)
```

autosklearn

```
AutoSklearnRegressor(per_run_time_limit=360)
```

02

AutoML 사용

auto-sklearn/H2O/TPOT 사용

→ TPOT이 가장 좋은 성능을 보임

```
from tpot import TPOTRegressor
tpot = TPOTRegressor(n_jobs=-1, verbosity=3,periodic_checkpoint_folder="tpot_results_no.txt")
tpot.fit(X_train, y_train)
print(tpot.score(X_val, y_val))
```

1. Feature 조절

→ 변환한 시간변수 및 로그 변환

변수 모두 입력

2. Oulier처리 방법조절
(LOF,IQR,처리X)

→ IQR

02

AutoML 사용

feature importance 확인 결과,

시간 > 온도 > 강수량 > 미세먼지 > 가시성 > 풍속 > 오존 > 습도 순

로그 변환 값들은 중요도가 매우 낮음

hour	0.290271
hour_category	0.097498
hour_bef_temperature	0.045220
hour_bef_precipitation	0.031316
hour_category_1	0.006296
hour_bef_pm10	0.003886
hour_bef_visibility	0.002771
hour_bef_windspeed	0.002762
hour_category_2	0.001273
t_pm10	0.000829
hour_bef_ozone	0.000712
t_pm2.5	0.000626
hour_bef_pm2.5	0.000476
hour_bef_humidity	0.000416
hour_sin	0.000377
t_windspeed	0.000371

성능 확인 : RMSE 기준

31.2871

감사합니다.