KUBIG 경진대회
진행사항 보고

머신러닝 분반 1조 김태영, 주세연, 채윤병

1주차 활동(8/11-18)

1. EDA를 통해 데이터의 특성을 파악
2. Preprocessing에서의 적합한 방법을 찾기 위해 팀원 세명이 각자 다른 형식으로 전처리를 해봄
3. 전처리를 한 후 비교를 위해 선형회귀 모델에 넣어 rmse를 비교하고 이 내용을 공유

CONTENTS

01

EDA 및 데이터분석

01 EDA 및 데이터 분석

EDA

Overview

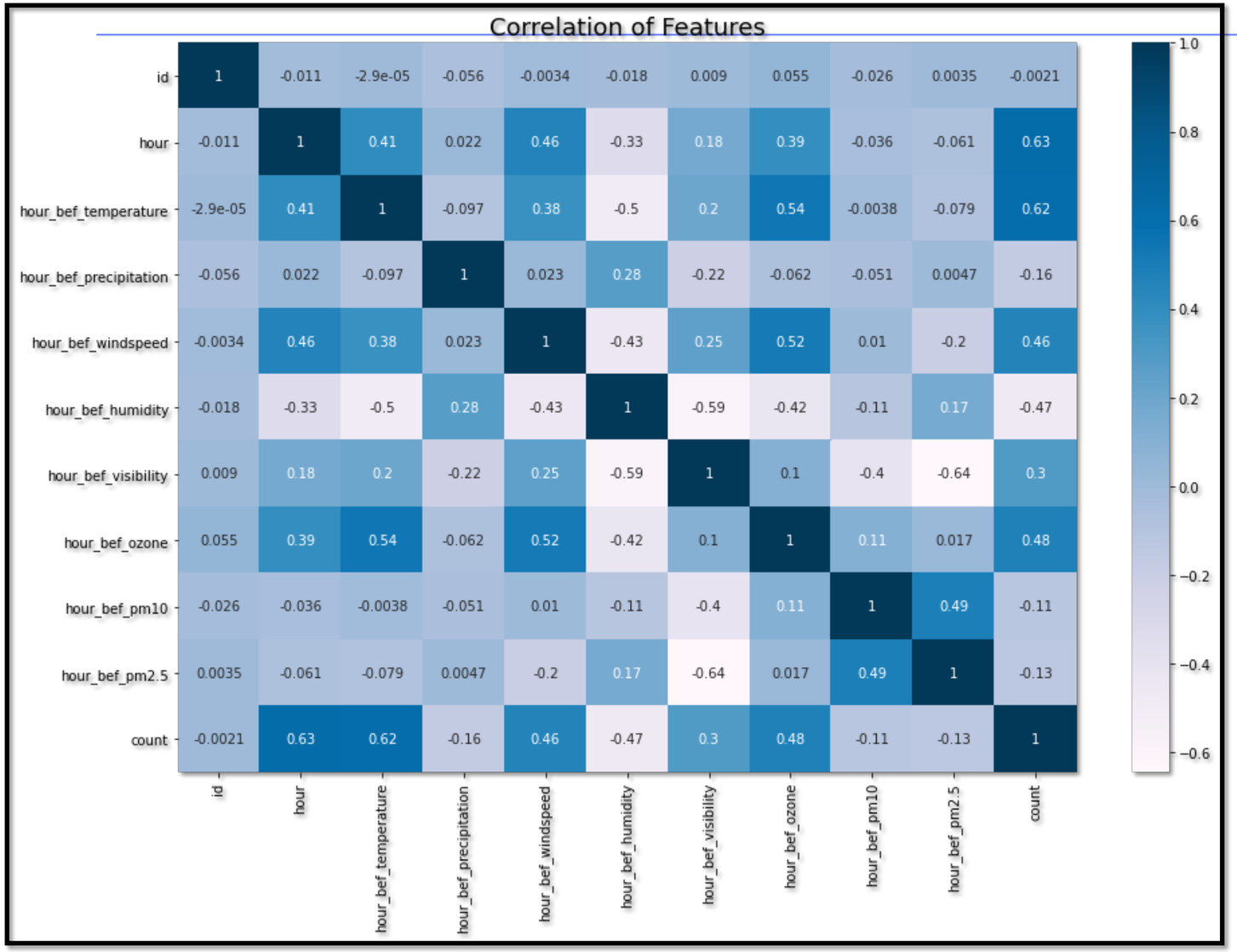
Dataset Statistics

Number of Variables	11
Number of Rows	1459
Missing Cells	300
Missing Cells (%)	1.9%
Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	125.5 KB
Average Row Size in Memory	88.1 B
Variable Types	Numerical: 10 Categorical: 1

Dataset Insights

`id` is uniformly distributed	Uniform
`hour_bef_ozone` has 76 (5.21%) missing values	Missing
`hour_bef_pm10` has 90 (6.17%) missing values	Missing
`hour_bef_pm2.5` has 117 (8.02%) missing values	Missing
`hour_bef_visibility` is skewed	Skewed
`id` is normally distributed	Normal
`hour_bef_precipitation` has constant length 3	Constant Length

EDA


Correlation of Features
id 1 -0.011 -2.9e-05 -0.056 -0.0034 -0.018 0.009 0.055 -0.026 0.0035 -0.0021
hour -0.011 1 0.41 0.022 0.46 -0.33 0.18 0.39 -0.036 -0.061 0.63
hour_bef_temperature -2.9e-05 0.41 1 -0.097 0.38 -0.5 0.2 0.54 -0.0038 -0.079 0.62
hour_bef_precipitation -0.056 0.022 -0.097 1 0.023 0.28 -0.22 -0.062 -0.051 0.0047 -0.16
hour_bef_windspeed -0.0034 0.46 0.38 0.023 1 -0.43 0.25 0.52 0.01 -0.2 0.46
hour_bef_humidity -0.018 -0.33 -0.5 0.28 -0.43 1 -0.59 -0.42 -0.11 0.17 -0.47
hour_bef_visibility 0.009 0.18 0.2 -0.22 0.25 -0.59 1 0.1 -0.4 -0.64 0.3
hour_bef_ozone 0.055 0.39 0.54 -0.062 0.52 -0.42 0.1 1 0.11 0.017 0.48
hour_bef_pm10 -0.026 -0.036 -0.0038 -0.051 0.01 -0.11 -0.4 0.11 1 0.49 -0.11
hour_bef_pm2.5 0.0035 -0.061 -0.079 0.0047 -0.2 0.17 -0.64 0.017 0.49 1 -0.13
count -0.0021 0.63 0.62 -0.16 0.46 -0.47 0.3 0.48 -0.11 -0.13 1
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
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

EDA

```
[ ]  1 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   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
```

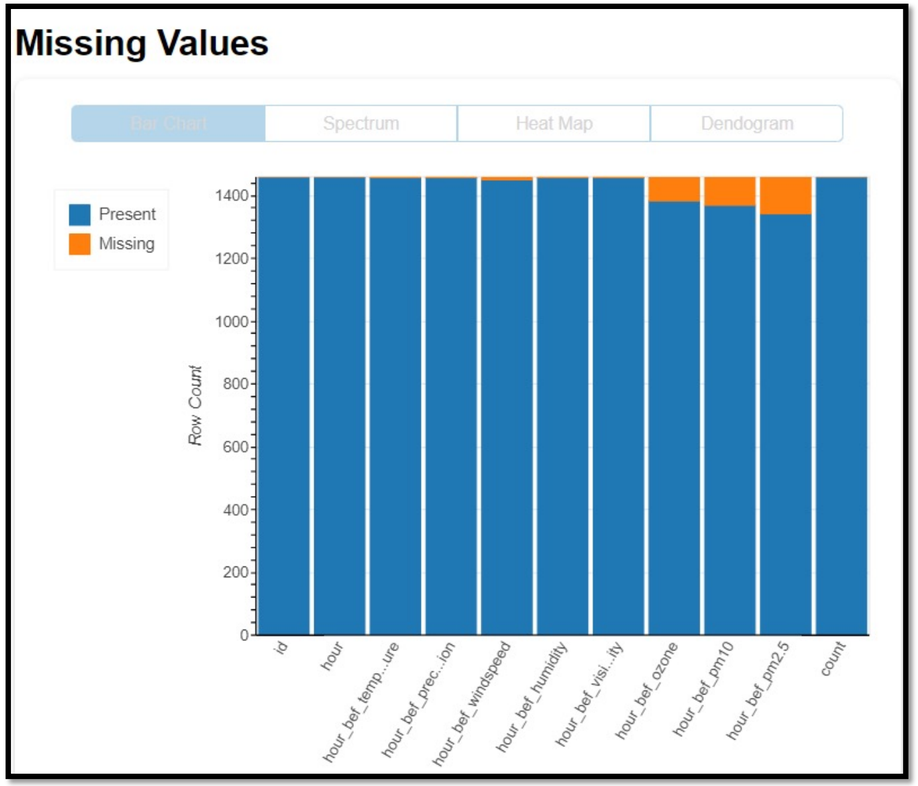
```
 1 test.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 715 entries, 0 to 714
Data columns (total 10 columns):
 #   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
```

02

Preprocessing

01. 결측치 처리
02. 이상치 처리
03. Scaler

결측치 처리




```
[6]   1 train.isna().sum()

id                          0
hour                        0
hour_bef_temperature        2
hour_bef_precipitation      2
hour_bef_windspeed          9
hour_bef_humidity           2
hour_bef_visibility         2
hour_bef_ozone             76
hour_bef_pm10              90
hour_bef_pm2.5            117
count                       0
dtype: int64
```

결측치 처리

1. 결측치가 특히 많았던 2개의 행을 미리 제거
2. Windspeed는 hour기준 평균값으로 결측치를 채움
3. pm2.5, pm10, ozone은 corr이 가장 높았던 열을 기준으로 grouping하고 평균값으로 결측치를 채움

```
1 train.iloc[[934,1035], :]
```

	id	hour	hour_bef_temperature	hour_bef_precipitation
934	1420	0	NaN	NaN
1035	1553	18	NaN	NaN

```
1 train.drop([934,1035], axis = 0, inplace =True)
2 train.reset_index(drop = True, inplace = True)
```

```
1 train['hour_bef_windspeed'].fillna(train.groupby('hour')['hour_bef_windspeed'].transform('mean'), inplace=True)
```

```
[ ]  1 np.sum(pd.isnull(train))

id                        0
hour                      0
hour_bef_temperature      0
hour_bef_precipitation    0
hour_bef_windspeed        0
hour_bef_humidity         0
```

```
[ ]  1 g1 = train.loc[(0 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 300) == True, 'hour_bef_pm2.5']
     2 g2 = train.loc[(300 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 600) == True, 'hour_bef_pm2.5']
     3 g3 = train.loc[(600 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 900) == True, 'hour_bef_pm2.5']
     4 g4 = train.loc[(900 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 1200) == True, 'hour_bef_pm2.5']
     5 g5 = train.loc[(1200 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 1500) == True, 'hour_bef_pm2.5']
     6 g6 = train.loc[(1500 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 1750) == True, 'hour_bef_pm2.5']
     7 g7 = train.loc[(1750 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] < 2000) == True, 'hour_bef_pm2.5']
     8 g8 = train.loc[(train['hour_bef_visibility'] == 2000) == True, 'hour_bef_pm2.5']

[ ]  1 g1.shape, g2.shape, g3.shape, g4.shape, g5.shape, g6.shape, g7.shape, g8.shape

((48,), (125,), (201,), (174,), (149,), (131,), (290,), (339,))

[ ]  1 g1.mean(), g2.mean(), g3.mean(), g4.mean(), g5.mean(), g6.mean(), g7.mean(), g8.mean()

(48.30769230769231,
 45.544642857142854,
 42.240437158469945,
 36.254777070063696,
 31.070921985815602,
 28.73015873015873,
 23.435424354243544,
```

결측치 처리

1. 변수별 평균값, 최빈값으로 대체
2. 결측치가 있는 행을 제거
3. KNN imputation(n_neighbors=8)
4. MICE imputation

평균값, 최빈값, 행 제거

```
[9]  1 from sklearn.impute import SimpleImputer
     2
     3 train_col = train.columns.tolist()
     4
     5 imputer1 = SimpleImputer(strategy = 'mean')
     6 imputer2 = SimpleImputer(strategy = 'most_frequent')
     7
     8 train1 = pd.DataFrame(imputer1.fit_transform(train),columns=train_col) # 열들의 평균값으로 결측치 매꿈
     9 train2 = pd.DataFrame(imputer2.fit_transform(train),columns=train_col) # 열들의 최빈값으로 결측치 매꿈
    10 train3 = pd.DataFrame(train.dropna()) # 결측치 있는 행 제거 (총 131개 행 제거)
```

```
[19] 1 # train 데이터 전체에 대해 KNN imputation
     2 train_col = train.columns.tolist()
     3
     4
     5 from sklearn.impute import KNNImputer
     6
     7 imputer = KNNImputer(n_neighbors=8) # 2~10에서 8이 RMSE 가장 작은 결과
     8 train_knn = pd.DataFrame(imputer.fit_transform(train),columns=train_col)
```

```
1 from sklearn.experimental import enable_iterative_imputer
2 from sklearn.impute import IterativeImputer
3
4 train_col = train.columns.tolist()
5
6 mice_imputer = IterativeImputer()
7 train_mice = pd.DataFrame(mice_imputer.fit_transform(train),columns=train_col)
```

결측치 처리

1. Hour과 corr이 높은 온도, 오존, 풍속, 습도 열은 hour기준 평균값으로 결측치 처리
2. 그 외의 열은 열 기준 전체 평균값으로 결측치 처리

```
# temp 결측치-시간별 평균
a_temp=train.groupby('hour').mean()['hour_bef_temperature']

temp=train[train['hour_bef_temperature'].isna()]


for index in temp.index:
  temp_mean=a_temp[train.iloc[index]['hour']]
  train['hour_bef_temperature'].fillna({index:temp_mean}, inplace=True)

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

```
## 시간별 평균으로 다루지 않을 변수들 : 시간과 상관계수가 크지 않을 경우 >> 전체평균
train.fillna({'hour_bef_precipitation':train['hour_bef_precipitation'].mean(), 'hour_bef_visibility':train['hour_bef_visibility'].mean(),
              'hour_bef_pm10':train['hour_bef_pm10'].mean(), 'hour_bef_pm2.5':train['hour_bef_pm2.5'].mean()}, inplace=True)
```

이상치 처리

1. 각 행 별로 모든 열에 대해 이상치가 있는 열을 찾고 이상치가 있는 열이 2개 이상인 행을 추출 > 해당 조건에 맞는 이상치는 없었음
2. InterQuartil Range에 들어가지 않는 모든 이상치를 제거
3. 이상치를 제거하지 않음

```
[ ]  1 def outliers(df,n,columns):
     2     outlier_indices = []
     3     for col in columns:
     4         Q1 = np.percentile(df[col],25)
     5         Q3 = np.percentile(df[col],75)
     6         IQR = (Q3 - Q1)*1.5
     7         lowest = Q1 - IQR
     8         highest = Q3 + IQR
     9
    10         outlier_index = df[col][(df[col] < lowest)|(df[col]>highest)].index
    11         outlier_indices.extend(outlier_index)
    12     outlier_indices = Counter(outlier_indices)
    13     multiple_outliers = list(k for k, v in outlier_indices.items() if v > n)
    14
    15     return multiple_outliers
```

```
[ ]  1 outliers_to_drop = outliers(train,1,['hour_bef_temperature', 'hour_bef_precipitation',
     2        'hour_bef_windspeed', 'hour_bef_humidity', 'hour_bef_visibility',
     3        'hour_bef_ozone', 'hour_bef_pm10', 'hour_bef_pm2.5'])
     4 outliers_to_drop

    []
```

```
[50]  1 def outliers_iqr(data):
      2   q1,q3=np.percentile(data, [25,75])
      3   iqr=q3-q1
      4   lower_bound=q1-(iqr*1.5)
      5   upper_bound=q3+(iqr*1.5)
      6
      7   return np.where((data>upper_bound)|(data<lower_bound))
      8
      9 temp_outlier_index=outliers_iqr(train['hour_bef_temperature'])[0]
     10 prec_outlier_index=outliers_iqr(train['hour_bef_precipitation'])[0]
     11 wind_outlier_index=outliers_iqr(train['hour_bef_windspeed'])[0]
     12 hum_outlier_index=outliers_iqr(train['hour_bef_humidity'])[0]
     13 ozone_outlier_index=outliers_iqr(train['hour_bef_ozone'])[0]
     14 bef10_outlier_index=outliers_iqr(train['hour_bef_pm10'])[0]
     15 bef25_outlier_index=outliers_iqr(train['hour_bef_pm2.5'])[0]
     16 visib_outlier_index=outliers_iqr(train['hour_bef_visibility'])[0]
     17
     18 total_outlier_index=np.concatenate((temp_outlier_index, prec_outlier_index, wind_outlier_index,
     19                                     hum_outlier_index, ozone_outlier_index, bef10_outlier_index,
     20                                    bef25_outlier_index,  visib_outlier_index))
     21
     22 total_outlier_index

array([   0,   15,   17,   19,   48,   59,  130,  154,  169,  222,  299,
        306,  309,  323,  333,  344,  363,  380,  387,  436,  495,  507,
        539,  583,  590,  605,  636,  692,  713,  718,  765,  796,  798,
        814,  842,  911,  934,  984, 1035, 1040, 1109, 1139, 1231, 1245,
       1272, 1288, 1413, 1443,  141,  314,  325,  354,  369,  380,  831,
       1118, 1293,  117,  118,  292,  296,  362,  419,  501,  641,  675,
       1008, 1107, 1215, 1222, 1293, 1427, 1430,   20,   49,   60,   62,
        120,  137,  183,  233,  234,  241,  278,  280,  287,  293,  343,
```

Scaler

1. Standard Scaler
2. Minmax Scaler
3. Robust Scaler

(Hour, Precipitation 제외)

```
# 이상치 제거함, scaler 적용 x
from sklearn.model_selection import train_test_split
x=train_clean[['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']]
target=train_clean['count'].values

X1_train, X1_test, y1_train, y1_test=train_test_split(x, target, test_size=0.3, random_state=42)

# 이상치 제거함, standard scaler 적용
from sklearn.preprocessing import StandardScaler
scaler1=StandardScaler()

X2_train=scaler1.fit_transform(X1_train)
X2_test=scaler1.fit_transform(X1_test)

# 이상치 제거함, robustsclaer 적용
from sklearn.preprocessing import RobustScaler
scaler2=RobustScaler()

X3_train=scaler2.fit_transform(X1_train)
X3_test=scaler2.fit_transform(X1_test)
```

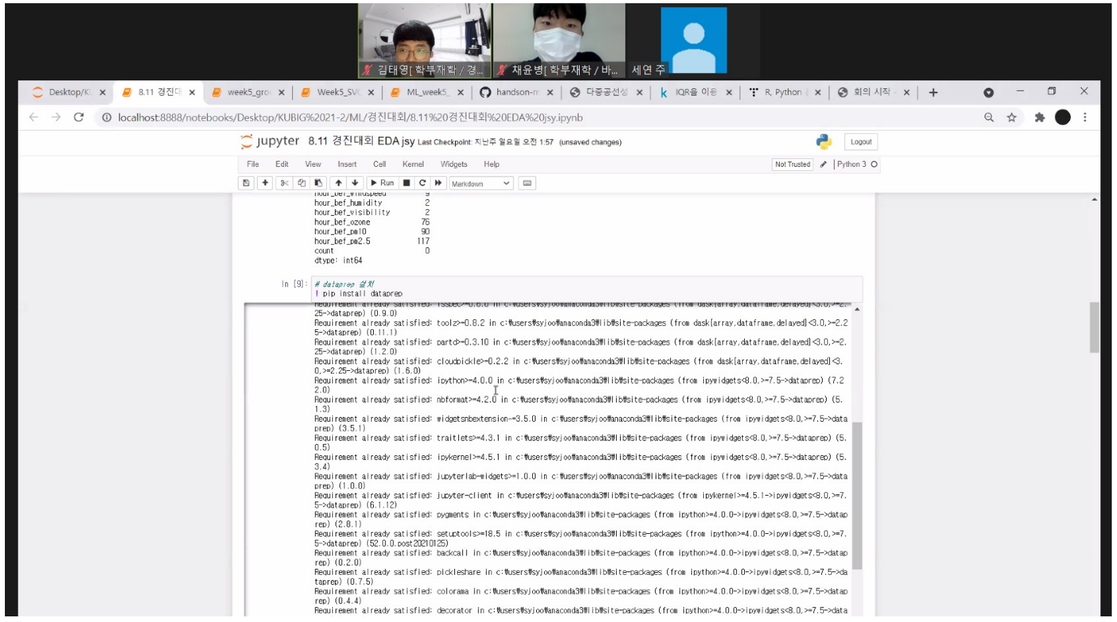
```
[ ]  scaler1 = StandardScaler()
     scaler2 = RobustScaler()
     scaler3 = MinMaxScaler()

[ ]  X_train1 = scaler1.fit_transform(X_train)
     X_test1 = scaler1.transform(X_test)

     X_train2 = scaler2.fit_transform(X_train)
     X_test2 = scaler2.transform(X_test)

     X_train3 = scaler3.fit_transform(X_train)
     X_test3 = scaler3.transform(X_test)
```

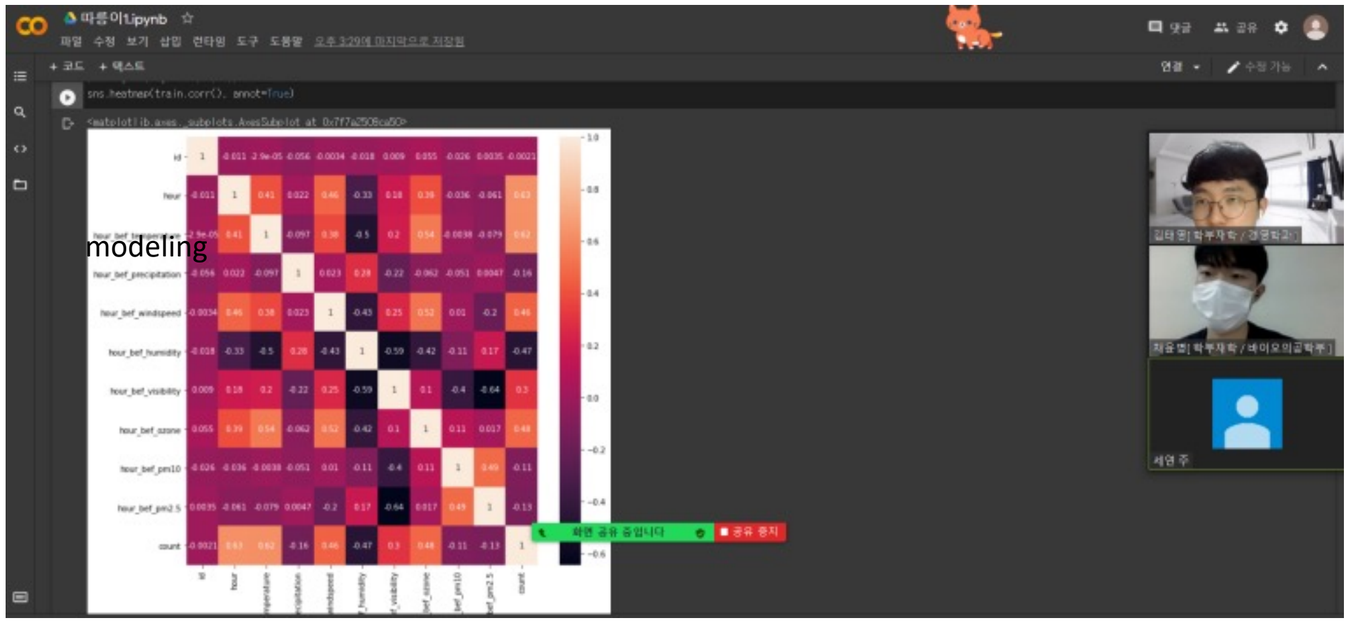
1주차 결론 및 2주차 과제 선정

1. 전처리에 있어서 결측치, 이상치 처리 및 scaler을 최대한 다양하게 적용해보고 그 결과를 비교해보고자 했음
2. 하지만 다양하게 적용한 model의 rmse를 기준으로 비교했을 때 0.5~0.6 사이로 유의미한 차이를 보이지 않음
3. 이렇게 전처리 과정에서 다양하게 적용했으나 큰 차이를 보이지 않은 것이 1주차 목표인 linear regression에 적용이라는 한정적인 모델에만 해당하는 것 때문인지 파악하기로 결정함
4. 이에 2주차에는 다양하게 전처리 해본 것을 기반으로 다양한 모델에 적용해보고 가장 높은 score를 갖는 모델에 대해 서로 공유하기로 함

```
[   1 model1.score(X_test1,y_test), model2.score(X_test2,y_test), model3.score(X_test3,y_test)

(0.5682331399366554, 0.5682331399366551, 0.5682331399366551)
```

```
 6 model1=lin.fit(X1_train, y1_train)
 7 y1_pred=model1.predict(X1_test)
 8 rmse1=np.sqrt(mean_squared_error(y1_test, y1_pred))
 9
10 model2=lin.fit(X2_train, y1_train)
11 y2_pred=model2.predict(X2_test)
12 rmse2=np.sqrt(mean_squared_error(y1_test, y2_pred))
13
14 model3=lin.fit(X3_train, y1_train)
15 y3_pred=model3.predict(X3_test)
16 rmse3=np.sqrt(mean_squared_error(y1_test, y3_pred))
17
18 print('scaler적용 안함:', rmse1)
19 print('standardscaler:', rmse2)
20 print('robustscaler:', rmse3)

scaler적용 안함: 58.48643931229177
standardscaler: 58.43451860676693
robustscaler: 58.52634625753629
```


김태영[학부재학 / 경...
채윤병[학부재학 / 바...
세연 주
localhost:8888/notebooks/Desktop/KUBIG%2021-2/ML/경진대회/8.11%20경진대회%20EDA%20jsy.ipynb
jupyter 8.11 경진대회 EDA jsy
File Edit View Insert Cell Kernel Widgets Help
hour_bef_humidity 2
hour_bef_visibility 2
hour_bef_ozone 76
hour_bef_pm10 90
hour_bef_pm2.5 117
count 0
dtype: int64
In [9]:
dataprep 설치
! pip install dataprep


따릉이1.ipynb
파일 수정 보기 삽입 런타임 도구 도움말 오후 3:29에 마지막으로 저장됨
+ 코드 + 텍스트
sns.heatmap(train.corr(), annot=True)
modeling
김태영[학부재학 / 경영학과]
채윤병[학부재학 / 바이오의공학부]
세연 주
화면 공유 중입니다
공유 중지

03

Modeling

01. Model 찾기
02. Grid Search

1주차~2주차 요약

1. 각자 생각한 전처리 방법으로 전처리 진행(결측치 처리, 이상치 처리)

ㄱ. 결측치가 있는 column의 corr이 가장 높은 열로 그룹핑을 하고 평균값으로 대체, 이상치는 제거하지 않음.

ㄴ. MICE imputation을 통해서 결측치 처리, 이상치는 제거하지 않고 visibility열에 log변환

ㄷ. Hour과 corr이 높은 온도, 오존, 풍속, 습도 열은 hour기준 평균값으로 결측치 처리 그 외의 열은 열 기준 전체 평균값으로 결측치 처리, IQR에 벗어나는 값 모두 제거

2. 3주차 과제 : 각자의 전처리 방법으로 모델의 성능 최대한 내보기!

model 찾기 – Bagging과 Boosting

Bagging (Bootstrap aggregating) : 훈련 세트의 서브셋을 무작위로 구성하여 분류기를 각기 다르게 학습시키는 것
> 중복을 허용한 샘플링 – Bagging, 허용 x - pasting
➢ 랜덤 포레스트는 일반적으로 배깅 방법을 사용

* 랜덤 포레스트와 엑스트라 트리의 차이점
1. 엑스트라 트리는 비복원추출(중복이 불가)
2. 랜덤 포레스트는 특성에 대한 정보 이득(information gain)을 계산하고 가장 높은 변수의 partition을 split node로 선택, but 엑스트라 트리는 무작위로 feature를 선정 > 계산이 빠르고 분산이 낮아짐

model 찾기 – Bagging과 Boosting

Boosting : 앞의 모델의 잘못된 예측에 대한 가중치를 두어 강한 학습기를 만드는 앙상블 방법, 오답 노트와 비슷한 개념

1. Adaboost –잘못 예측한 샘플의 가중치 상대적으로 높이면서 가중치를 업데이트
2. Gradient boosting - Adaboost처럼 오차를 보정하도록 예측기를 순차적으로 추가, 샘플의 가중치를 수정하는 대신 이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습

Bagging과 Boosting의 차이?

- Bagging은 새로운 샘플링을 통해 병렬적으로 학습을 진행, Boosting은 오차를 보정하며 순차적으로 학습을 진행

model 찾기

ensemble.AdaBoostClassifier([...])	An AdaBoost classifier.
ensemble.AdaBoostRegressor([base_estimator, ...])	An AdaBoost regressor.
ensemble.BaggingClassifier([base_estimator, ...])	A Bagging classifier.
ensemble.BaggingRegressor([base_estimator, ...])	A Bagging regressor.
ensemble.ExtraTreesClassifier([...])	An extra-trees classifier.
ensemble.ExtraTreesRegressor([n_estimators, ...])	An extra-trees regressor.
ensemble.GradientBoostingClassifier(*[, ...])	Gradient Boosting for classification.
ensemble.GradientBoostingRegressor(*[, ...])	Gradient Boosting for regression.
ensemble.IsolationForest(*[, n_estimators, ...])	Isolation Forest Algorithm.
ensemble.RandomForestClassifier([...])	A random forest classifier.
ensemble.RandomForestRegressor([...])	A random forest regressor.
ensemble.RandomTreesEmbedding([...])	An ensemble of totally random trees.
ensemble.StackingClassifier(estimators[, ...])	Stack of estimators with a final classifier.
ensemble.StackingRegressor(estimators[, ...])	Stack of estimators with a final regressor.
ensemble.VotingClassifier(estimators, *[, ...])	Soft Voting/Majority Rule classifier for unfitted estimators.
ensemble.VotingRegressor(estimators, *[, ...])	Prediction voting regressor for unfitted estimators.

Sklearn.ensemble의 부스팅 모델, 트리 모델 사용 > 성능이 좋은 모델 찾기

model 찾기

```
ETR = ExtraTreesRegressor(random_state = 2021)
params = {'n_estimators' : [100,300,500,1000],
          'max_depth' : [3,5,7,10],
          'min_samples_split' : [2,3,4],
          'min_samples_leaf' : [2,3]
         }
clf5 = GridSearchCV(ETR, param_grid=params ,cv = kfold, n_jobs=-1)
clf5.fit(x_train_sc, y_train)
```

개별 모델에서 성능이 높았던 모델 >
RandomForestRegressor, GradientBoostingRegressor, ExtraTreeRegressor
각 모델에서 그리드 서치 진행 > 종합하여 Voting

```
Estimators = [
    ("RandomForest_clf", RandomForest_clf),
    ("Gradient_clf", Gradient_clf),
    ("ExtraTree_clf", ExtraTree_clf),
]
```

```
Voting_clf = VotingRegressor(Estimators)
Voting_clf.fit(x_train_sc, y_train)
```

```
y_pred = Voting_clf.predict(x_val_sc)
mse = mean_squared_error(y_pred, y_val)
rmse = np.sqrt(mse)
rmse
```

```
35.7328643217972
```

model 찾기

```
models = [
    ('ridge', lm.Ridge()),
    ('lasso', lm.Lasso()),
    ('elastic', lm.ElasticNet()),
    ('LassoLars', lm.LassoLars()),
    ('LogisticRegression', lm.LogisticRegression()),
    ('SGDRegressor', lm.SGDRegressor()),
    ('Perceptron', lm.Perceptron(n_jobs=-1)),
    ('xgboost', xgb.XGBRegressor()),
    ('adaboost', AdaBoostRegressor())
]
```

지금까지 배웠던 모델들을 사용 > 파라미터를 조정하여 그리드 서치를 진행

부스팅 방법을 사용하는 Xgboost의 성능이 가장 좋았음.

```
params = {
    'ridge': {
        'alpha': [0.01, 0.1, 1.0, 10, 100],
        'fit_intercept': [True, False],
        'normalize': [True, False],
    },
    'lasso': {
        'alpha': [0.1, 1.0, 10],
        'fit_intercept': [True, False],
        'normalize': [True, False],
    },
    'elastic': {
        'alpha': [0.1, 1.0, 10],
        'normalize': [True, False],
        'fit_intercept': [True, False],
    },
```

```
ridge 51.90232528433858
lasso 51.93554083191449
elastic 52.85413365229021
LassoLars 51.93580205129492
LogisticRegression 62.30525824903908
SGDRegressor 55.8819353179203
Perceptron 80.23058948172121
xgboost 39.284652467005245
adaboost 58.286694907742415
```

```
ridge 51.79826243258097
lasso 51.831636689182744
elastic 52.63182897793959
LassoLars 51.83205299732882
LogisticRegression 59.75037494389532
SGDRegressor 53.713550461418244
Perceptron 194.90523982544832
xgboost 38.635730908444536
adaboost 59.98681515705783
```

model 찾기

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	25.1921	1373.3447	36.8115	0.7915	0.5163	0.7901	0.1330
lightgbm	Light Gradient Boosting Machine	25.8581	1448.5111	37.7513	0.7809	0.5558	0.8364	0.0270
rf	Random Forest Regressor	26.5597	1531.6863	38.8407	0.7686	0.5321	0.8494	0.1660
gbr	Gradient Boosting Regressor	27.6909	1601.5385	39.7069	0.7581	0.5658	0.8825	0.0530
knn	K Neighbors Regressor	28.1211	1673.5837	40.6325	0.7466	0.5577	1.0724	0.0110
ada	AdaBoost Regressor	41.8867	2632.4846	51.1874	0.5972	0.8327	1.7634	0.0440
lr	Linear Regression	39.2973	2694.6444	51.4630	0.5966	0.7955	1.2475	0.0110
ridge	Ridge Regression	39.2851	2694.6770	51.4613	0.5966	0.7952	1.2518	0.0090
lar	Least Angle Regression	39.2973	2694.6443	51.4630	0.5966	0.7955	1.2475	0.0100
br	Bayesian Ridge	39.2549	2701.4905	51.5148	0.5958	0.8006	1.2857	0.0060
lasso	Lasso Regression	39.4729	2737.3074	51.8566	0.5905	0.7906	1.3628	0.0080

Test data에도 MICE imputation을 진행
AutoML 패키지인 pycaret을 사용해서 가장 좋은 성능을 내는 모델을 파악

Grid Search 및 4주차 과제 선정

1. 개별적으로 그리드 서치를 진행 > 하이퍼 파라미터 설정의 한계
2. 랜덤 포레스트의 경우

n_estimators - 결정트리의 개수를 지정 , 트리 개수를 늘리면 성능이 좋아질 수 있지만 시간이 오래 걸림
min_samples_split – 노드를 분할하기 위한 최소한의 샘플 데이터 수, 과적합을 제어하는데 사용
min_samples_leaf – 리프노드가 되기 위해 필요한 최소한의 샘플 데이터 수 , 과적합을 제어하는데 사용
max_depth – 트리의 최대 깊이, 깊이가 깊어지면 과적합

3. 각 모델에서 하이퍼 파라미터 설정을 좀 더 구체화하기

4. Voting을 비롯해 Stacking 방법도 고려