

KUBIG CONTEST

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

The screenshot shows the DAICON competition interface. At the top, the DAICON logo and navigation links (대회, 커뮤니티, 교육, 랭킹, 더보기) are visible. The main banner features the title "[공공] 서울시 따릉이 자전거 이용 예측 AI모델" and a subtitle "공공 빅데이터인 서울시 따릉이 데이터를 이용하여 인공지능 모델 개발". It also displays the prize "상금: 교육", the duration "2020.01.30 ~ 2031.09.01 17:59", and the number of participants "1,030명". A "Level Up!" badge is present in the top right corner. Below the banner, there are tabs for "대회안내", "데이터", "코드 공유", "토크", "대회문의", "리더보드", "팀", and "제출". Under the "대회안내" tab, there are sub-tabs for "개요", "규칙", "일정", and "상금". The "개요" sub-tab is selected, showing the competition details: "2017년 4월 1일부터, 5월 31일까지 시간별로 서울시 따릉이 대여수와 기상상황 데이터가 주어집니다. 각 날짜의 1시간 전의 기상상황을 가지고 1시간 후의 따릉이 대여수를 예측해보세요."

DAICON 대회 커뮤니티 교육 랭킹 더보기

[공공] 서울시 따릉이 자전거 이용 예측 AI모델
공공 빅데이터인 서울시 따릉이 데이터를 이용하여 인공지능 모델 개발

상금: 교육
2020.01.30 ~ 2031.09.01 17:59 + Google Calendar
1,030명 D-3651

Level Up!

대회안내 데이터 코드 공유 토크 대회문의 리더보드 팀 제출

개요 규칙 일정 상금

2017년 4월 1일부터, 5월 31일까지 시간별로 서울시 따릉이 대여수와 기상상황 데이터가 주어집니다.
각 날짜의 1시간 전의 기상상황을 가지고 1시간 후의 따릉이 대여수를 예측해보세요.

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: 58.48643991229177
standardscaler: 58.43451860676693
robustscaler: 58.52634625753629
```

2주차 결론 및 3주차 과제 선정

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

ㄱ. 결측치가 있는 column의 corr01 가장 높은 열로 그룹핑을 하고 평균값으로 대체, 이상치는 제거하지 않음.

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

ㄷ. Hour과 corr01 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!

3주차 결론 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

CONTENTS

- 01 EDA
- 02 Preprocessing
- 03 Modeling
- 04 Conclusion

01 |

EDA

|

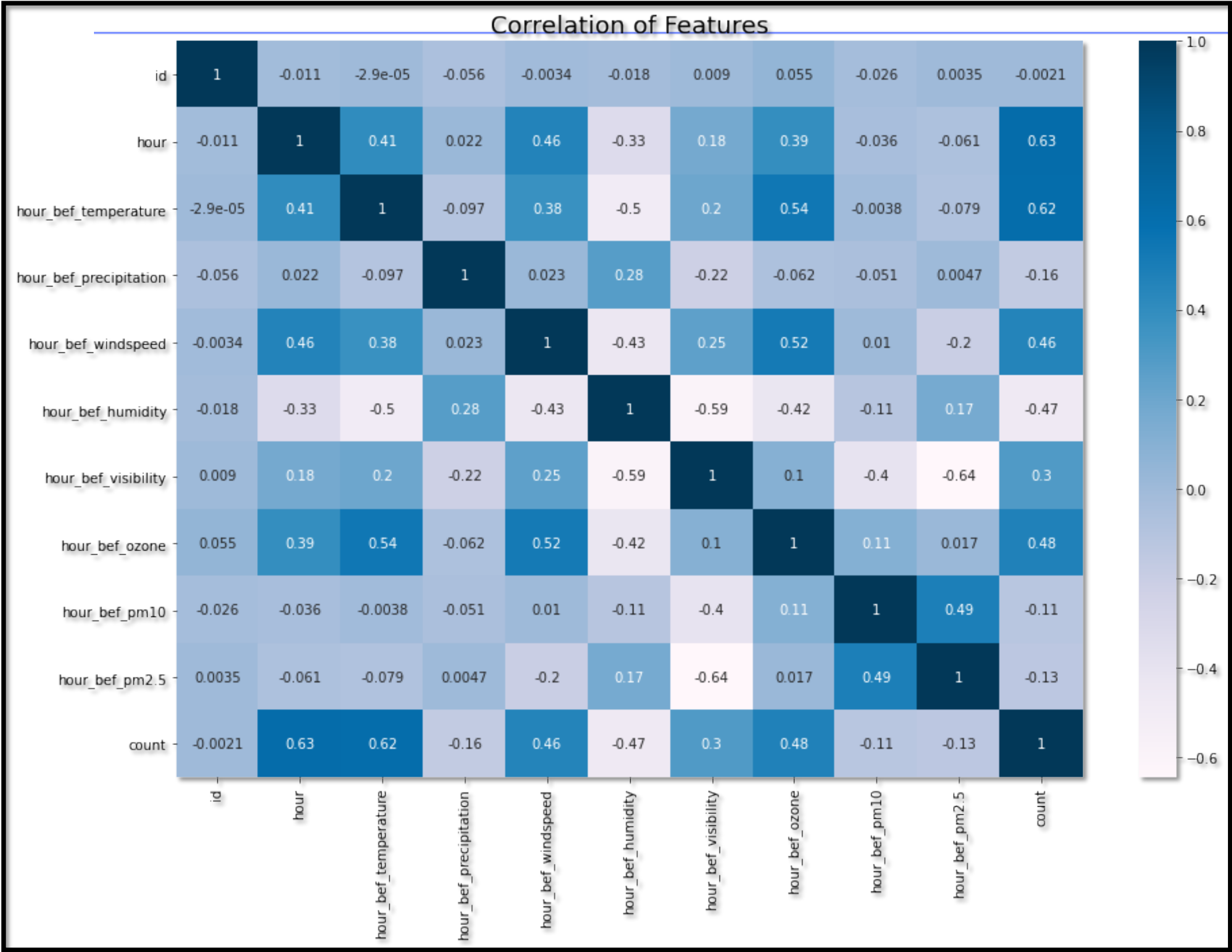
EDA

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



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
```

```
1 test.isna().sum()

id          0
hour        0
hour_bef_temperature  1
hour_bef_precipitation  1
hour_bef_windspeed    1
hour_bef_humidity     1
hour_bef_visibility   1
hour_bef_ozone        35
hour_bef_pm10         37
hour_bef_pm2.5       36
dtype: int64
```

02 Preprocessing

01 결측치 처리

02 이상치 처리

03. Scaler

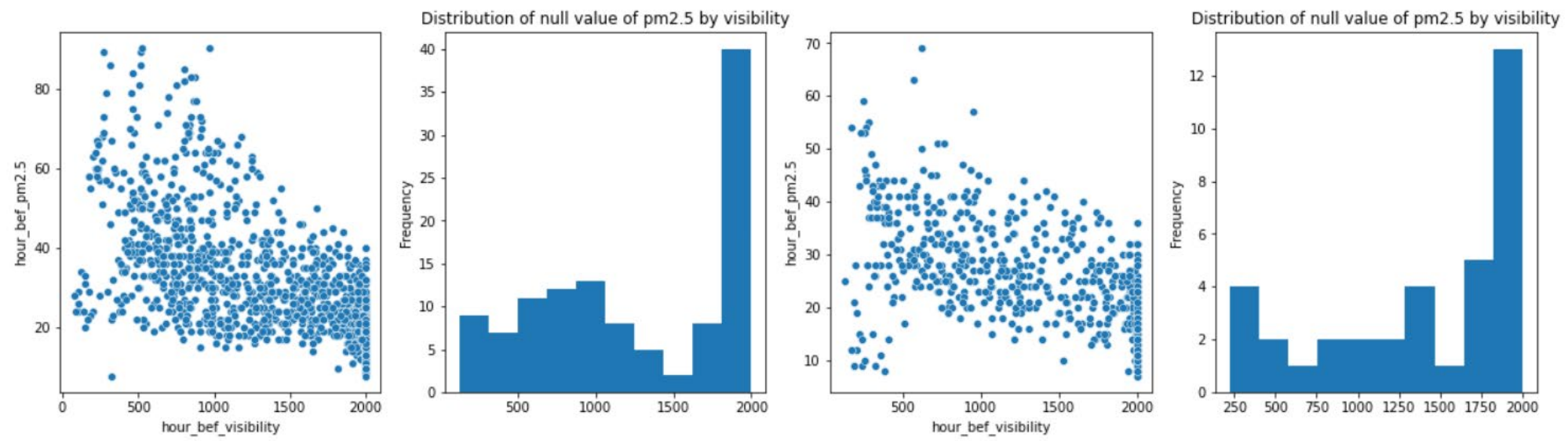
결측치 처리

1. Train set에서 934, 1035행이 모든 열에 대해 결측치를 가짐 > 제거
2. 결측치가 있는 열을 해당 열과 corr이 높은 다른 열의 그룹핑으로 평균값 대체 (pm2.5>>visibility, pm10>>pm2.5, ozone>>temperature, windspeed>>ozone)

```
1 g1 = train.loc[(0 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 500) == True, 'hour_bef_pm2.5']
2 g2 = train.loc[(500 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 1000) == True, 'hour_bef_pm2.5']
3 g3 = train.loc[(1000 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 1500) == True, 'hour_bef_pm2.5']
4 g4 = train.loc[(1500 < train['hour_bef_visibility']) & (train['hour_bef_visibility'] <= 2000) == True, 'hour_bef_pm2.5']
5
6 gt1 = test.loc[(0 < test['hour_bef_visibility']) & (test['hour_bef_visibility'] <= 500) == True, 'hour_bef_pm2.5']
7 gt2 = test.loc[(500 < test['hour_bef_visibility']) & (test['hour_bef_visibility'] <= 1000) == True, 'hour_bef_pm2.5']
8 gt3 = test.loc[(1000 < test['hour_bef_visibility']) & (test['hour_bef_visibility'] <= 1500) == True, 'hour_bef_pm2.5']
9 gt4 = test.loc[(1500 < test['hour_bef_visibility']) & (test['hour_bef_visibility'] <= 2000) == True, 'hour_bef_pm2.5']

1 g1.mean(), g2.mean(), g3.mean(), g4.mean()

(45.94059405940594, 42.31229235880399, 32.22608695652174, 22.409859154929578)
```



02 Preprocessing

01 결측치 처리

02 이상치 처리

03. Scaler

이상치 처리

1. 사분위표 interquartile 기준 ($Q1-IQR$, $Q3+IQR$) 범위에 속하지 않는 이상치를 파악
2. 해당 이상치를 제거, 1459행에서 1375행으로 축소

```
1 def outliers(df,n,colums):
2     outlier_indices = []
3     for col in colums:
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,0,[ 'hour_bef_temperature',
2     'hour_bef_windspeed', 'hour_bef_humidity', 'hour_bef_visibility',
3     'hour_bef_ozone', 'hour_bef_pm2.5'])
4 train = train.drop(outliers_to_drop,axis=0).reset_index(drop=True)
5 train.shape
```

```
(1375, 11)
```

02 Preprocessing

01 결측치 처리

02 이상치 처리

03. Scaler

Scaler

1. Standard Scaler 적용(Hour, Precipitation 제외)
2. Minmax Scaler 적용
3. Robust Scaler 적용 > 이상치를 사전에 제거할 경우 큰 효과는 없음
4. 결론적으로 Standard scaler를 적용했을 때 가장 적합했음

```
1 # 이상치 제거함, scaler 적용 x
2 from sklearn.model_selection import train_test_split
3 x=train_clean[['hour_bef_temperature', 'hour_bef_precipitation', 'hour_bef_windspeed', 'hour_bef_humidity',
4               'hour_bef_visibility', 'hour_bef_ozone', 'hour_bef_pm10', 'hour_bef_pm2.5']]
5 target=train_clean['count'].values
6
7 X1_train, X1_test, y1_train, y1_test=train_test_split(x, target, test_size=0.3, random_state=42)
8
9 # 이상치 제거함, standard scaler 적용
10 from sklearn.preprocessing import StandardScaler
11 scaler1=StandardScaler()
12
13 X2_train=scaler1.fit_transform(X1_train)
14 X2_test=scaler1.fit_transform(X1_test)
15
16 # 이상치 제거함, robustscaler 적용
17 from sklearn.preprocessing import RobustScaler
18 scaler2=RobustScaler()
19
20 X3_train=scaler2.fit_transform(X1_train)
21 X3_test=scaler2.fit_transform(X1_test)
```

```
[ ] 1 scaler1 = StandardScaler()
    2 scaler2 = RobustScaler()
    3 scaler3 = MinMaxScaler()
    4
    5
[ ] 1 X_train1 = scaler1.fit_transform(X_train)
    2 X_test1 = scaler1.transform(X_test)
    3
    4 X_train2 = scaler2.fit_transform(X_train)
    5 X_test2 = scaler2.transform(X_test)
    6
    7 X_train3 = scaler3.fit_transform(X_train)
    8 X_test3 = scaler3.transform(X_test)
```

03

Modeling

- 01. Best Model
- 02. Grid Search
- 03. Voting & Stacking

03 Modeling

01 Best Model

02 Grid search

Best Model – Model List

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

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

Best Model – AutoML Pycaret

	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

03 Modeling

01 Best Model

02 Grid search

03 Voting & Stacking

Best Model – Model List

1. RandomForest, LGBM, GradientBoosting, ExtraTree, XGBoost를 선정
2. 이 때 전처리나 하이퍼파라미터의 설정에 따라 rmse가 많이 변동했음
3. Hyperparameter Example – Random Forest

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

03 Modeling

01 Best Model

02 Grid search

03 Voting & Stacking

Grid Search

```
1 RFR = RandomForestRegressor(random_state = 2021)
2 params = {'n_estimators' : [1000],
3           'max_depth' : [10,11],
4           'min_samples_split' : [2,3],
5           'min_samples_leaf' : [1,2]
6           }
7 clf = GridSearchCV(RFR, param_grid=params ,cv = kfold, n_jobs=-1)
8 clf.fit(x_train_sc, y_train)
```

RandomForest : 36.3454

```
1 GBR = GradientBoostingRegressor(random_state = 2021)
2 params3 = {'n_estimators' : [200,400,600,800,1000],
3            'learning_rate' : [0.001,0.005,0.01,0.05,0.1],
4            'max_depth' : [3,5,7,9],
5            }
6
7 clf3 = GridSearchCV(GBR, param_grid=params3 ,cv = kfold, n_jobs = -1)
8 clf3.fit(x_train_sc, y_train)
```

GradientBoost : 36.8457

```
1 LGB = LGBMRegressor(random_state=2021)
2 params2 = {'n_estimators' : [1000,1200,1400,1600,1800,2000],
3            'num_leaves' : [8,16,32,64,128,256,512],
4            'learning_rate' : [0.001,0.01,0.1,0.5],
5            'max_depth' : [3,6,9,12]
6            }
7 clf2 = GridSearchCV(LGB, param_grid=params2 ,cv = kfold)
8 clf2.fit(x_train_sc, y_train)
```

LGBM : 36.7599

```
1 ETR = ExtraTreesRegressor(random_state = 2021)
2 params = {'n_estimators' : [1400,1600,1800],
3            'max_depth' : [10,11,12,13,14,15,16,17],
4            }
5 clf5 = GridSearchCV(ETR, param_grid=params ,cv = kfold, n_jobs=-1)
6 clf5.fit(x_train_sc, y_train)
```

ExtraTree : 36.2187

```
1 XGB = XGBRegressor(random_state=2021)
2 params = {'learning_rate' : [0.01],
3            'n_estimators' : [2000]
4            , 'max_depth' : [9]
5            }
6 clf6 = GridSearchCV(XGB, param_grid = params, cv = kfold, n_jobs = -1 )
7 clf6.fit(x_train_sc, y_train)
```

XgBoost : 35.9662

03 Modeling

01 Best Model

02 Grid search

03 Voting & Stacking

Grid Search

```
1 RandomForest_clf = RandomForestRegressor(n_estimators = 1000, max_depth = 10, min_samples_split = 3, random_state = 2021)
2 LGB_clf = LGBMRegressor(learning_rate = 0.01, n_estimators = 1000, num_leaves = 8, max_depth = 3, random_state=2021)
3 Gradient_clf = GradientBoostingRegressor(learning_rate = 0.01, n_estimators = 1000, max_depth = 3, random_state = 2021)
4 ExtraTree_clf = ExtraTreesRegressor(max_depth = 11, n_estimators = 1500, random_state = 2021)
5 XGBoost_clf = XGBRegressor(learning_rate = 0.01, n_estimators = 2000, max_depth = 9, random_state = 2021)
```

```
] 1 estimators = [RandomForest_clf, LGB_clf, Gradient_clf, ExtraTree_clf, XGBoost_clf]
   2 for estimator in estimators :
   3     estimator.fit(x_train_sc, y_train)
```

```
[12:58:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
] 1 for estimator in estimators :
   2     y_pred = estimator.predict(x_val_sc)
   3     mse = mean_squared_error(y_pred, y_val)
   4     rmse = np.sqrt(mse)
   5     print(rmse)
```

```
36.34154209734633
36.75993597408493
36.8457160269155
34.842187131709395
35.966223476032326
```

03 Modeling

01 Best Model

02 Grid search

03 Voting & Stacking

Voting & Stacking

```
1 Estimators_voting = [  
2     ("RandomForest_clf", RandomForest_clf),  
3     ("LGB_clf", LGB_clf),  
4     ("Gradient_clf", Gradient_clf),  
5     ("ExtraTree_clf", ExtraTree_clf),  
6     ("XGBoost_clf", XGBoost_clf)]  
7 Estimators_stacking = [  
8     ("RandomForest_clf", RandomForest_clf),  
9     ("LGB_clf", LGB_clf),  
0     ("Gradient_clf", Gradient_clf),  
1     ("ExtraTree_clf", ExtraTree_clf),  
2     ("XGBoost_clf", XGBoost_clf)]
```

```
1 Voting_clf = VotingRegressor(Estimators_voting)  
2 Voting_clf.fit(x_train_sc, y_train)  
3 Stacking_clf = StackingRegressor(Estimators_stacking, cv = kfold, n_jobs = -1)  
4 Stacking_clf.fit(x_train_sc, y_train)
```

```
1 y_pred = Voting_clf.predict(x_val_sc)  
2 mse = mean_squared_error(y_pred, y_val)  
3 rmse = np.sqrt(mse)  
4 print(rmse)  
5 y_pred = Stacking_clf.predict(x_val_sc)  
6 mse = mean_squared_error(y_pred, y_val)  
7 rmse = np.sqrt(mse)  
8 print(rmse)
```



```
35.07586395859205  
33.73938554671867
```

04

Conclusion

Conclusion

```
1 Estimators_final = [  
2  
3  
4     ("Gradient_final", Gradient_final),  
5     ("ExtraTree_final", ExtraTree_final),  
6     ("XGBoost_final", XGBoost_final)  
7 ]  
  
1 Voting_final = VotingRegressor(Estimators_final)  
2 Voting_final.fit(X_train_sc, train['count'])
```

12	needmorecaffeine		31.00069	4	4분 전
13	운뱅뱅		31.00069	29	2시간 전

Conclusion

1. EDA의 중요성 – 컬럼별로 상호관련성이 높거나 다중공선성을 가지는 등 관련 과제에서 가장 먼저 이루어져야 하는 과정이며 그 내용을 자세히 살펴봐야 함
2. 전처리, 즉 결측치 처리, 이상치 처리, scaler의 접근법은 매우 다양하며 이에 따라 모델의 성능의 변동이 큼
3. 전처리에서 다양한 방법을 시도하는 것이 중요하고 이 때 original data를 최대한 유지하거나 아니면 지나치게 변형하는 방법은 오히려 성능을 떨어뜨릴 수 있음
4. 예측문제 내에서도 통상적으로 알려져있는 모델들의 성능 순위와 달리 data set과 전처리 과정에 따라 그 순위는 바뀔 수 있음
5. 모델 선정에 있어서 다양한 방법을 시도하고 특히 단일 모델만이 아닌 voting과 stacking과 같은 시도도 반드시 해볼 필요성이 있음

