KUBIG
머신러닝 분반 데이터 분석 대회

서울시 따릉이 자전거 이용 예측 AI모델

머신러닝 3조

컴퓨터학과 임형우   산업경영공학부 남이량
보건정책관리학부 임혜리   통계학과 김수경

CONTENTS

변수 설명

id : 날짜와 시간별 id

hour_bef_temperature : 1시간 전 기온

hour_bef_precipitation : 1시간 전 비 정보, 비가 오지 않았으면 0, 비가 오면 1

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

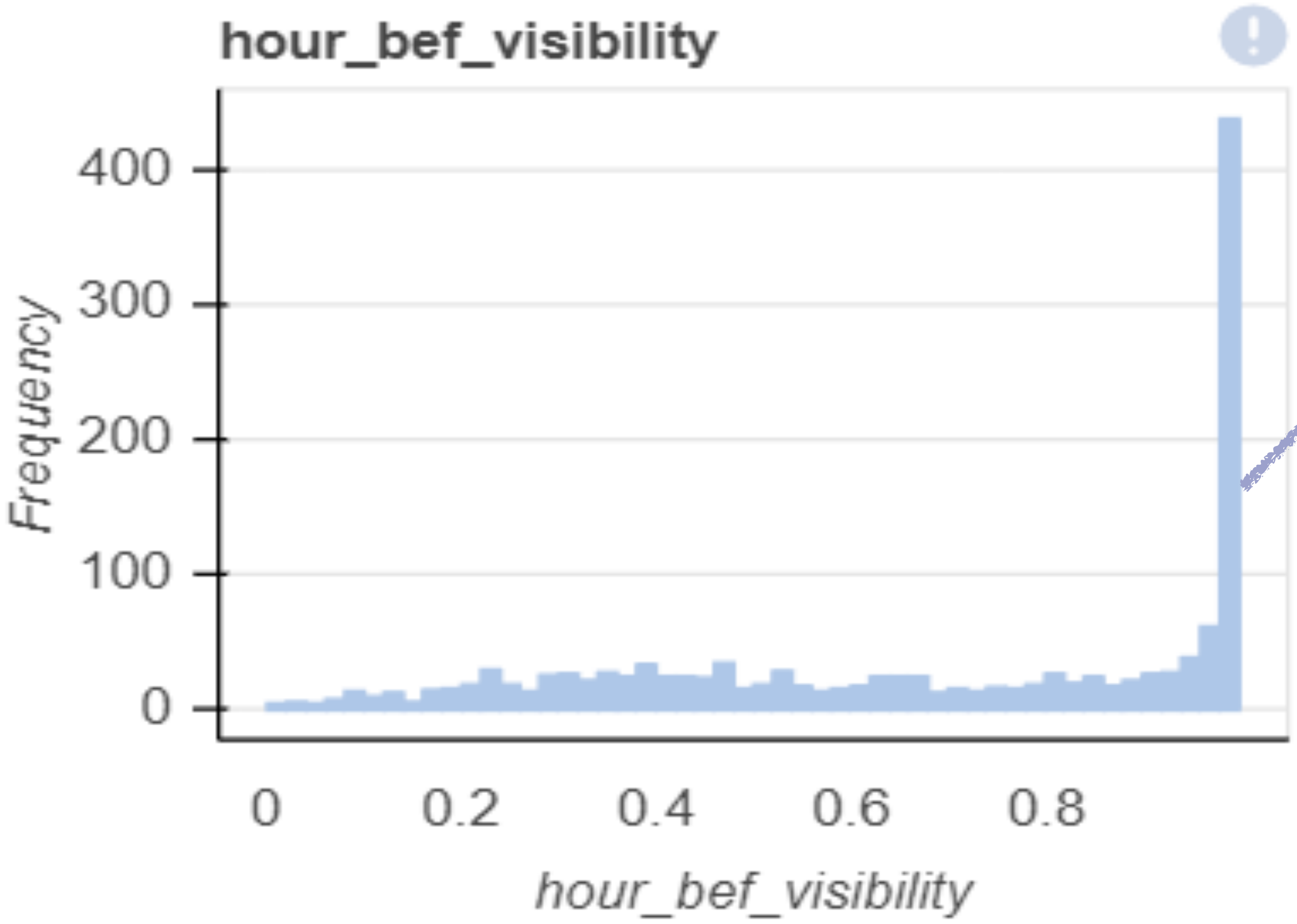
hour_bef_pm10 : 1시간 전 미세먼지(머리카락 굵기의 1/5에서 1/7 크기의 미세먼지)

hour_bef_pm2.5 : 1시간 전 미세먼지(머리카락 굵기의 1/20에서 1/30 크기의 미세먼지)

count : 시간에 따른 따릉이 대여 수 <- 예측해야하는 값

*이 중에서 hour_bef_visibility 는 왜도가 높은 관계로 최종 모델에서 제외 시킴.

변수 선택



지나치게 편향되어 있는 모습

결측치 처리 방법

```python
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

it_train = train.copy()

it_train = IterativeImputer(random_state=2021).fit_transform(it_train)

itImp = pd.DataFrame(it_train)
itImp.columns = column_names
```

결측치 삭제, 전체 평균, 시간별 평균, KNN, IterativeImputer 시도 결과 IterativeImputer가 가장 결과가 좋게 나와서 이것으로 선정

IterativeImputer의 원리
: 종속 변수 y를 선택 하고 여러 독립 변수 X 를 사용하여 y를 예측할 수 있는 함수에 맞춘다.
  이 함수는 y열의 결측값을 예측하는 데 사용된다.

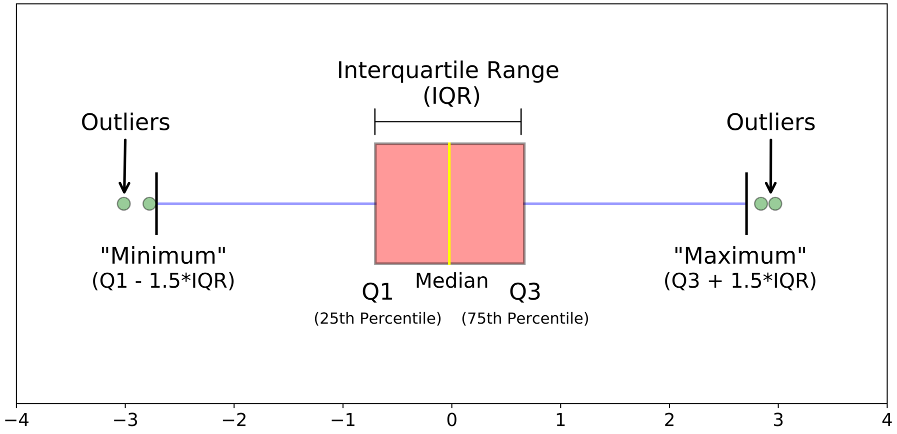
이상치 처리 방법

```
itImp_mid = itImp.copy()

for iIt in col_name:
    Q1=itImp_mid[iIt].quantile(0.25)
    Q3=itImp_mid[iIt].quantile(0.75)
    IQR=Q3-Q1
    train_delout=itImp_mid[(itImp_mid[iIt]<(Q1 - 1.5*IQR)) | (itImp_mid[iIt]>(Q3+1.5*IQR))]
    itImp_mid=itImp_mid.drop(train_delout.index, axis=0)
```

Isolation Forest, Minimum Covariance Determinant, IQR 중 IQR이 가장 결과가 좋게 나옴.

IQR은 오른쪽 그림처럼 계산된 최솟값과 최댓값 사이에 있는 값 이외의 값은 이상치로 처리하여 삭제하는 방법



더미변수 추가

```python
def busyHourGen(data, col):
    lst = data[col]
    lst_ = []
    for i in lst:
        if (6 < i < 10) or (16 < i < 20):
            lst_.append(1)
        else:
            lst_.append(0)
    data['busy_hour'] = lst_
    return data
```

```python
an = busyHourGen(itImp_mid, 'hour') # an = busy_hour 추가된 데이터셋
```

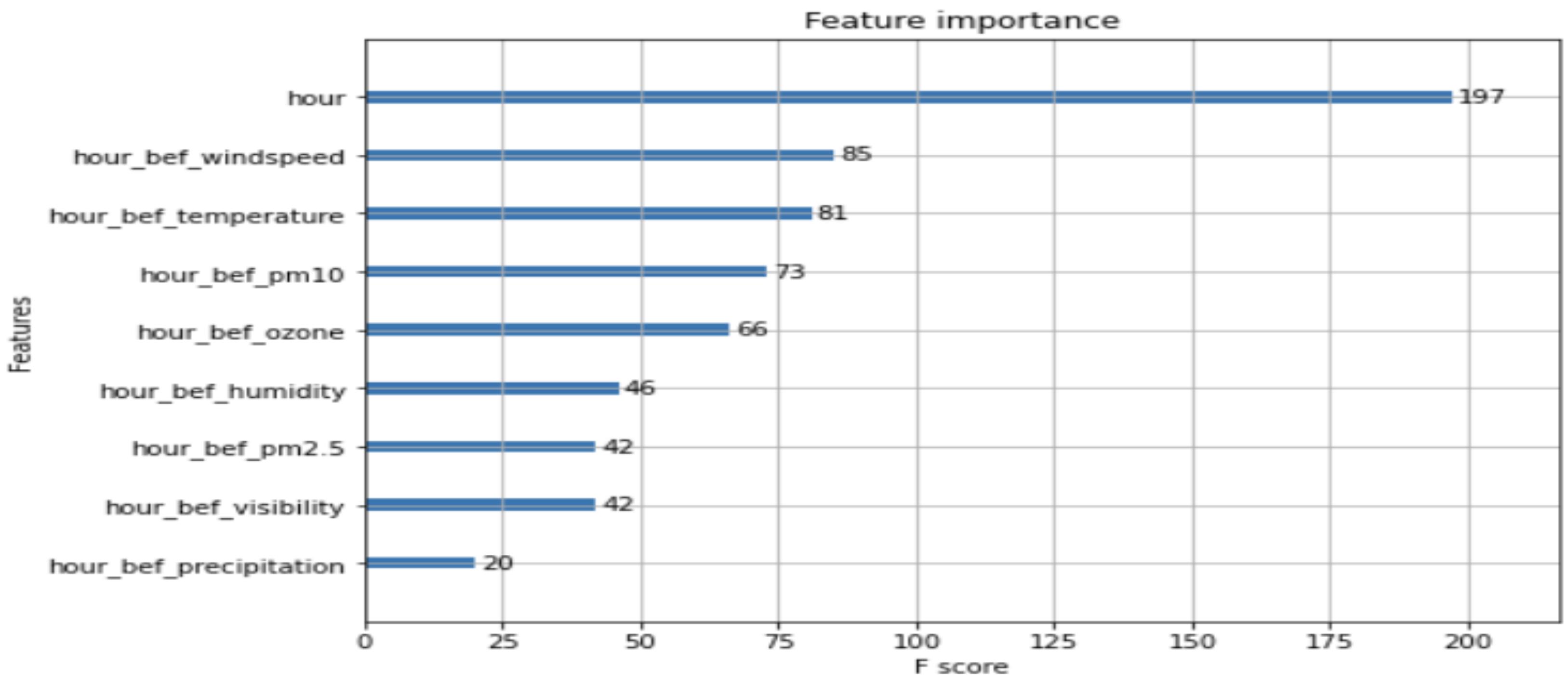
출퇴근 시간(7-9시, 17-19시)은 1, 나머지는 0으로 분류하는 더미변수를 추가함

변수 중요도 확인

```
xgbr_all = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, objective='reg:squarederror', random_state=2021)
xgbr_all.fit(X_train_all, y_train_all)
pred = xgbr_all.predict(X_val_all)

rmse = np.sqrt(mean_squared_error(y_val_all, pred))
print(rmse)

fig, ax = plt.subplots(figsize=(8, 6))
plot_importance(xgbr_all, ax=ax)
```



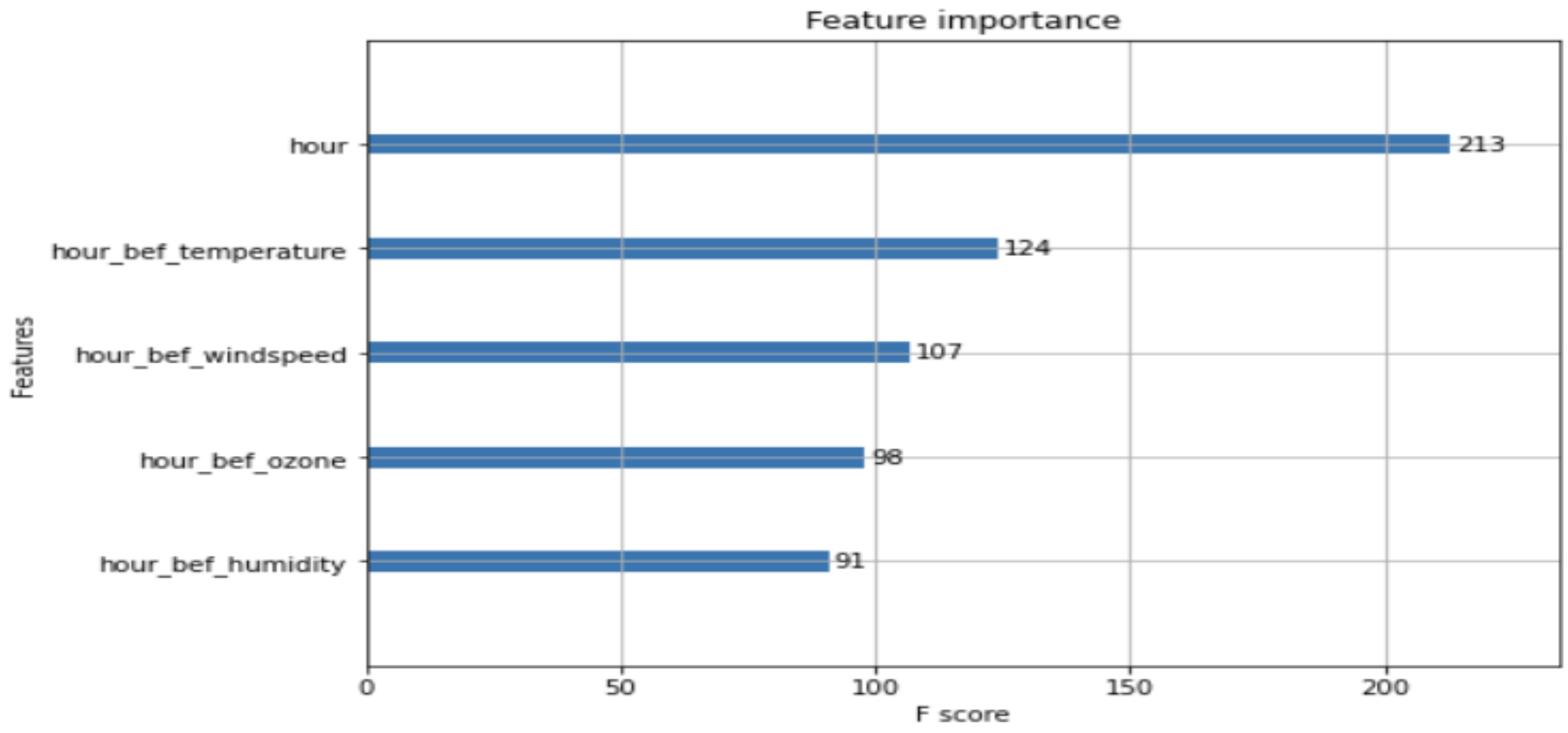
XGBRegressor feature_importance 확인 (모든 독립변수 사용)

```python
xgbr = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, objective='reg:squarederror', random_state=2021)
xgbr.fit(X_train_IF, y_train_IF)
pred = xgbr.predict(X_val_IF)

rmse = np.sqrt(mean_squared_error(y_val_IF, pred))
print(rmse)
```

42.133570222911764

```python
fig, ax = plt.subplots(figsize=(8, 6))
plot_importance(xgbr, ax=ax)
```



XGBRegressor feature_importance 확인 (종속변수와 선형관계가 절댓값 0.4 이상인 독립변수 사용)

Polynomial

```
X = an[an.columns.difference(['count', 'hour_bef_visibility'])]
colls = X.columns.tolist()
X = np.column_stack((X['hour']**5, X['hour_bef_temperature']**4, X))
y = an[['count']]
```

기존 시간과 시간 전 온도에 대한 변수 값들에 더불어 해당 변수를 한번더 학습시킬 목적으로 변수 추가

변수중요도가 높게 나온 변수 hour 과 hour_bef_temperature에 가중치를 부여함.

Modeling

1. Linear Regression
2. Ridge
3. Lasso
4. ElasticNet
5. Decision Tree
6. Randomforest
7. XGBoostRegressor
8. LightGBMRegressor
9. ExtratreeRegressor
10. AdaboostRegressor

1.Linear Regression Linear Regression

```
pipe_lr = make_pipeline(MinMaxScaler(), LinearRegression())
pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_val)
rmse = np.sqrt(mean_squared_error(y_val, y_pred))
print("rmse of Linear Regression(*hour1 제거): {:.6f}".format(rmse))

rmse of Linear Regression(*hour1 제거): 43.416379
```

※ rmse = 43.416379

2.Ridge Regression Ridge Regression

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge, Lasso
alphas = [0.01, 0.1, 1.0, 10, 100]
param_grid = {'ridge__alpha' : alphas}
pipe_Ridge = make_pipeline(MinMaxScaler(), Ridge())
grid_search = GridSearchCV(pipe_Ridge, param_grid, cv = 5, scoring = 'neg_mean_squared_error',
                          verbose = 0)
grid_search.fit(X_train, y_train)

GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('minmaxscaler',
                                        MinMaxScaler(copy=True,
                                                     feature_range=(0, 1))),
                                       ('ridge',
                                        Ridge(alpha=1.0, copy_X=True,
                                              fit_intercept=True, max_iter=None,
                                              normalize=False,
                                              random_state=None, solver='auto',
                                              tol=0.001))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'ridge__alpha': [0.01, 0.1, 1.0, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
[24] grid_search.best_params_

{'ridge__alpha': 1.0}
```

```
[25] print('Best rmse of ridge regression : {}'.format(np.sqrt(-grid_search.best_score_)))

Best rmse of ridge regression : 46.9516595211406
```

※ rmse = 46.951659

3.Lasso Regression

Lasso Regression

```
[26] param_grid = {'lasso__alpha' : alphas}
     pipe_Lasso = make_pipeline(MinMaxScaler(), Lasso())
     grid_search = GridSearchCV(pipe_Lasso, param_grid, cv = 5, scoring = 'neg_mean_squared_error',
                               verbose = 0)
     grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('minmaxscaler',
                                        MinMaxScaler(copy=True,
                                                     feature_range=(0, 1))),
                                       ('lasso',
                                        Lasso(alpha=1.0, copy_X=True,
                                              fit_intercept=True, max_iter=1000,
                                              normalize=False, positive=False,
                                              precompute=False,
                                              random_state=None,
                                              selection='cyclic', tol=0.0001,
                                              warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'lasso__alpha': [0.01, 0.1, 1.0, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
[27] grid_search.best_params_
```

```
{'lasso__alpha': 0.01}
```

```
print('Best rmse of lasso regression : {}'.format(np.sqrt(-grid_search.best_score_)))
```

```
Best rmse of lasso regression : 46.97090031292715
```

※ rmse = 46.9709

4.Elasticnet Regression

Elasticnet Regression

```
[30] from sklearn.linear_model import ElasticNet
     param_grid = {'elasticnet__l1_ratio' : [0.001, 0.01, 0.05, 0.1, 0.5, 1],
                   'elasticnet__alpha' : [0.001, 0.01, 0.05, 0.1, 1, 10]}
     pipe_en = make_pipeline(MinMaxScaler(), ElasticNet())
     grid_search = GridSearchCV(pipe_en, param_grid, cv = 5, scoring = 'neg_mean_squared_error',
                               verbose = 0)
     grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('minmaxscaler',
                                        MinMaxScaler(copy=True,
                                                     feature_range=(0, 1))),
                                       ('elasticnet',
                                        ElasticNet(alpha=1.0, copy_X=True,
                                                   fit_intercept=True,
                                                   l1_ratio=0.5, max_iter=1000,
                                                   normalize=False,
                                                   positive=False,
                                                   precompute=False,
                                                   random_state=None,
                                                   selection='cyclic',
                                                   tol=0.0001,
                                                   warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'elasticnet__alpha': [0.001, 0.01, 0.05, 0.1, 1, 10],
                         'elasticnet__l1_ratio': [0.001, 0.01, 0.05, 0.1, 0.5,
                                                  1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
[31] grid_search.best_params_
```

```
{'elasticnet__alpha': 0.001, 'elasticnet__l1_ratio': 0.001}
```

```
print('Best rmse of elasticnet regression: {}'.format(np.sqrt(-grid_search.best_score_)))
```

```
Best rmse of elasticnet regression: 46.952267649499
```

※ rmse = 46.95226

5. Decision Tree

Decision Tree

```
[33] from sklearn.tree import DecisionTreeRegressor
param_grid = {'min_impurity_decrease' : np.arange(0.0001, 0.001, 0.0001),
              'max_depth' : range(5, 20, 1),
              'min_samples_split' : range(2, 100, 10),
              'min_samples_leaf': [1, 3, 5]}
grid_search = GridSearchCV(DecisionTreeRegressor(random_state = 42), param_grid, cv = 3, scoring = 'neg_mean_squared_error',
                           verbose = 0, n_jobs = -1)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse',
                                             max_depth=None, max_features=None,
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             presort='deprecated',
                                             random_state=42, splitter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': range(5, 20),
                         'min_impurity_decrease': array([0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008,
       0.0009]),
                         'min_samples_leaf': [1, 3, 5],
                         'min_samples_split': range(2, 100, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
[34] grid_search.best_params_
```

```
{'max_depth': 8,
 'min_impurity_decrease': 0.0001,
 'min_samples_leaf': 3,
 'min_samples_split': 52}
```

```
print('Best rmse of decision tree: {}'.format(np.sqrt(-grid_search.best_score_)))
```

```
Best rmse of decision tree: 44.80211363615863
```

※ rmse = 44.8021136

6.Randomforest

Randomforest

```
from sklearn.ensemble import RandomForestRegressor
param_grid ={
    'n_estimators':[100,200],
    'max_depth':[6,8,10,12],
    'max_features': [5, 6, 8],
    'min_samples_leaf':[1,3,5],
    'min_samples_split':[8,16,20]}
grid_search = GridSearchCV(RandomForestRegressor(random_state = 42), param_grid, cv=2, n_jobs=-1, scoring = 'neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=2, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=None,
                                             oob_score=False, random_state=42,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [6, 8, 10, 12], 'max_features': [5, 6, 8],
                         'min_samples_leaf': [1, 3, 5],
                         'min_samples_split': [8, 16, 20],
                         'n_estimators': [100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
[37] grid_search.best_params_
```

```
{'max_depth': 10,
 'max_features': 8,
 'min_samples_leaf': 1,
 'min_samples_split': 8,
 'n_estimators': 200}
```

```
[38] print('Best rmse of random forest: {}'.format(np.sqrt(-grid_search.best_score_)))
```

```
Best rmse of random forest: 40.48403013008874
```

※ rmse = 40.4840301

7. XGBoostRegressor XGBoostRegressor

```
import xgboost as xgb
from xgboost import XGBRegressor
grid_search = GridSearchCV(XGBRegressor(random_state = 42), param_grid, cv=2, n_jobs=-1, scoring = 'neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

```
[40] grid_search.best_params_
```

```
{'max_depth': 6,
 'max_features': 5,
 'min_samples_leaf': 1,
 'min_samples_split': 8,
 'n_estimators': 100}
```

```
[41] print('Best rmse of random forest: {}'.format(np.sqrt(-grid_search.best_score_)))
```

```
Best rmse of random forest: 40.498049359641925
```

※ rmse = 40.498049

8. LightGBMRegressor LightGBMRegressor

```
from lightgbm import LGBMRegressor
grid_search = GridSearchCV(LGBMRegressor(random_state = 42), param_grid, cv=2, n_jobs=-1, scoring = 'neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

```
[43] grid_search.best_params_
```

```
{'max_depth': 6,
 'max_features': 5,
 'min_samples_leaf': 1,
 'min_samples_split': 8,
 'n_estimators': 100}
```

```
[44] print('Best rmse of random forest: {}'.format(np.sqrt(-grid_search.best_score_)))
```

```
Best rmse of random forest: 41.12409774869825
```

※ rmse = 41.1240977

9. ExtratreeRegressor ExtratreeRegressor

```
[45] from sklearn.ensemble import ExtraTreesRegressor
     grid_search = GridSearchCV(LGBMRegressor(random_state = 42), param_grid, cv=2, n_jobs=-1, scoring = 'neg_mean_squared_error')
     grid_search.fit(X_train, y_train)
```

```
[47] grid_search.best_params_

     {'max_depth': 6,
      'max_features': 5,
      'min_samples_leaf': 1,
      'min_samples_split': 8,
      'n_estimators': 100}
```

```
print('Best rmse of random forest: {}'.format(np.sqrt(-grid_search.best_score_)))

Best rmse of random forest: 40.498049359641925
```

※ rmse = 40.498049

10.AdaboostRegressor AdaboostRegressor

```
[49] from sklearn.ensemble import AdaBoostRegressor
     params = {
      'n_estimators': [50, 100],
      'learning_rate' : [0.01, 0.05, 0.1, 0.5],
      'loss' : ['linear', 'square', 'exponential']
      }
     grid_search = GridSearchCV(AdaBoostRegressor(random_state = 42), params, cv=2, n_jobs=-1, scoring = 'neg_mean_squared_error')
     grid_search.fit(X_train, y_train)

     GridSearchCV(cv=2, error_score=nan,
                  estimator=AdaBoostRegressor(base_estimator=None, learning_rate=1.0,
                                              loss='linear', n_estimators=50,
                                              random_state=42),
                  iid='deprecated', n_jobs=-1,
                  param_grid={'learning_rate': [0.01, 0.05, 0.1, 0.5],
                              'loss': ['linear', 'square', 'exponential'],
                              'n_estimators': [50, 100]},
                  pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                  scoring='neg_mean_squared_error', verbose=0)
```

```
[50] grid_search.best_params_

     {'learning_rate': 0.1, 'loss': 'exponential', 'n_estimators': 100}
```

```
print('Best rmse of random forest: {}'.format(np.sqrt(-grid_search.best_score_)))

Best rmse of random forest: 45.94408293360038
```

※ rmse = 45.94408

확실히 앙상블 기법 모델에서 rmse 값이 작음을 확인 할 수 있음!!

Pycaret(auto ML 사용)

```
df_train = pd.DataFrame(itImp_mid, columns = col_name)
from pycaret.regression import *
reg = setup(df_train, target = 'count', train_size=0.8)
```

```
best = compare_models(sort = 'RMSE')
```

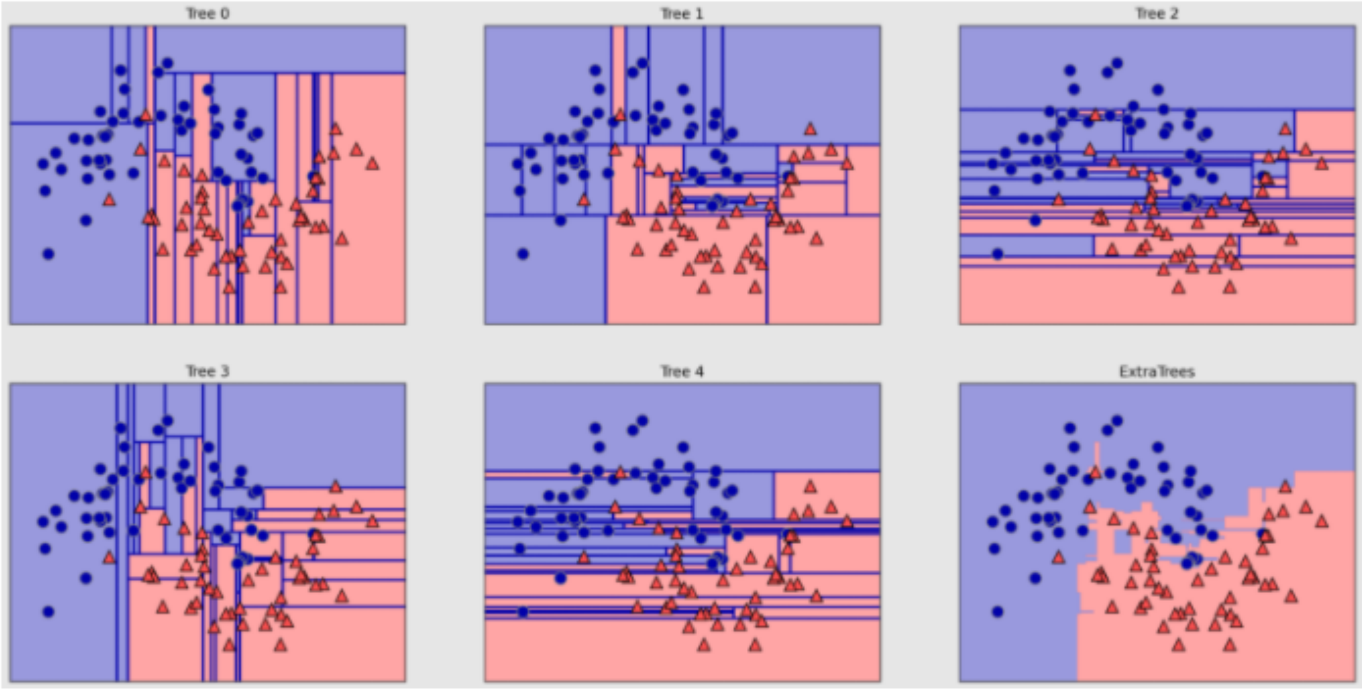
	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	26.6149	1585.3635	39.5798	0.7532	0.5150	0.7911	0.483
rf	Random Forest Regressor	26.9390	1624.4292	40.0401	0.7473	0.5149	0.7701	0.595
lightgbm	Light Gradient Boosting Machine	28.5280	1717.9340	41.1941	0.7326	0.5567	0.8189	0.104
gbr	Gradient Boosting Regressor	28.3165	1725.2821	41.3538	0.7330	0.5244	0.7307	0.112
knn	K Neighbors Regressor	34.0773	2324.6122	47.9688	0.6449	0.6006	0.8970	0.060
dt	Decision Tree Regressor	34.4026	2857.4482	52.9623	0.5457	0.6310	0.6668	0.020
ridge	Ridge Regression	40.9188	2975.7489	54.3707	0.5472	0.7766	1.3281	0.012
br	Bayesian Ridge	40.9274	2976.2587	54.3741	0.5473	0.7795	1.3245	0.015
lasso	Lasso Regression	40.9291	2976.7872	54.3779	0.5473	0.7805	1.3241	0.015
lr	Linear Regression	40.9124	2977.8184	54.3846	0.5469	0.7794	1.3283	0.280
en	Elastic Net	40.9454	2979.2839	54.3950	0.5474	0.7685	1.3153	0.015
huber	Huber Regressor	39.8534	3056.7320	55.0113	0.5389	0.7691	1.2334	0.032
ada	AdaBoost Regressor	46.0833	3107.4500	55.6446	0.5162	0.8936	1.9182	0.107
omp	Orthogonal Matching Pursuit	47.9573	4192.4937	64.5197	0.3645	0.8554	1.8988	0.013
par	Passive Aggressive Regressor	57.7696	5642.5954	72.7618	0.1243	0.9414	1.9716	0.017

Pycaret 결과 ‘extra trees regressor’모델에서 가장 성능이 좋음을 확인 > 이 모델을 사용하기로 결정!!

Extra Trees Regressor

앙상블 학습의 일종이며 Bagging 방식을 통해 각각의 예측 모델(결정 트리)이 데이터 샘플링을 다르게 가져가 최종적으로 모든 결정 트리의 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법

Random Forest 알고리즘과 비슷하지만, Extra Trees 알고리즘은 보다 더 랜덤한 방식(splitter='random')으로 결정 트리 생성



하이퍼 파라미터 튜닝

```python
def objective_extratree(trial):
    params_et = {
        "n_estimators": trial.suggest_int("n_estimators", 100, 1000, step=1),
        "max_depth": trial.suggest_int("max_depth", 2, 20),
        "min_samples_split": trial.suggest_int("min_samples_split", 2, 4),
        "min_samples_leaf": trial.suggest_int("min_samples_leaf", 1, 3),
        "max_features": trial.suggest_categorical("max_features", ["auto", "sqrt", "log2", 4, 5, 6]),
        "warm_start": trial.suggest_categorical("warm_start", [True, False]),
        "random_state": 2021
    }

    X = an[an.columns.difference(['count', 'hour_bef_visibility'])]
    y = an[['count']]
    X_train_ori, X_val_ori, y_train_ori, y_val_ori = train_test_split(X, y, test_size=0.33, random_state=2021)

    model = ExtraTreesRegressor(**params_et)
    model.fit(X_train_ori, y_train_ori)

    pred = model.predict(X_val_ori)
    rmse = np.sqrt(mean_squared_error(y_val_ori, pred))

    return rmse
```

```python
sampler = TPESampler(seed=2021)
study = optuna.create_study(
    study_name="et_optimizer",
    direction="minimize",
    sampler=sampler,
)
study.optimize(objective_extratree, n_trials=30)
```

대표적인 방법으로 GridSearchCV, RandomizedSearchCV, Optuna 등이 있지만, Optuna가 월등히 빠른 속도와 준수한 정확도를 보여 사용

Optuna 작동 방식
:각 파라미터의 범위와 모델의 정확도 측정 방식이 정의된 Objective 함수를 입력하면 Trial을 거듭하며 실행 History를 바탕으로 더 정확한 파라미터를 선정 (n_trial까지 trial 반복)

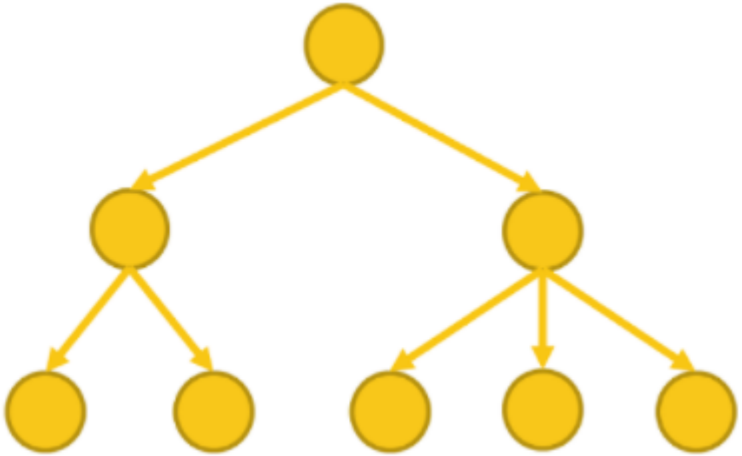
주요 파라미터 설명

n_estimator
:Extra Trees 알고리즘에서 총 몇 개의 결정 트리를 사용하는지

max_depth
:Extra Trees 알고리즘에서 사용될 트리의 최대 깊이

min_samples_split
:트리 생성시 내부 노드를 분할하는 데 필요한 최소 샘플 수

max_features
:랜덤하게 뽑을 독립변수들(column)의 가짓수의 max 값



모델 예측 정확도에 해당 파라미터들의 영향이 큰 듯하여 중점적으로 튜닝

하이퍼 파라미터 튜닝

```
[I 2021-08-29 15:47:29,858] A new study created in memory with name: et_optimizer
[I 2021-08-29 15:47:31,222] Trial 0 finished with value: 36.30854722532004 and parameters: {'n_estimators': 700, 'max_depth': 15, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', '
[I 2021-08-29 15:47:32,252] Trial 1 finished with value: 37.58139185260117 and parameters: {'n_estimators': 1000, 'max_depth': 13, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt',
[I 2021-08-29 15:47:32,903] Trial 2 finished with value: 37.26820547849423 and parameters: {'n_estimators': 600, 'max_depth': 11, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 5, 'warm_
[I 2021-08-29 15:47:33,357] Trial 3 finished with value: 36.81363252749193 and parameters: {'n_estimators': 400, 'max_depth': 18, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 5, 'warm_
[I 2021-08-29 15:47:34,012] Trial 4 finished with value: 38.74131749169292 and parameters: {'n_estimators': 700, 'max_depth': 8, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 4, 'warm_s
[I 2021-08-29 15:47:34,115] Trial 5 finished with value: 38.344489656756 and parameters: {'n_estimators': 100, 'max_depth': 9, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_features': 4, 'warm_sta
[I 2021-08-29 15:47:34,776] Trial 6 finished with value: 41.85845483168387 and parameters: {'n_estimators': 800, 'max_depth': 7, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 'sqrt', 'w
[I 2021-08-29 15:47:35,820] Trial 7 finished with value: 37.42930616021127 and parameters: {'n_estimators': 1000, 'max_depth': 9, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_features': 6, 'warm_
[I 2021-08-29 15:47:36,426] Trial 8 finished with value: 40.23539903680105 and parameters: {'n_estimators': 700, 'max_depth': 8, 'min_samples_split': 3, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'w
[I 2021-08-29 15:47:36,986] Trial 9 finished with value: 47.91344730562039 and parameters: {'n_estimators': 800, 'max_depth': 4, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 'log2', 'w
[I 2021-08-29 15:47:37,677] Trial 10 finished with value: 36.57784820943486 and parameters: {'n_estimators': 400, 'max_depth': 20, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'auto',
[I 2021-08-29 15:47:38,194] Trial 11 finished with value: 36.58384153346146 and parameters: {'n_estimators': 300, 'max_depth': 20, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'auto',
[I 2021-08-29 15:47:38,882] Trial 12 finished with value: 36.26400097284186 and parameters: {'n_estimators': 400, 'max_depth': 16, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'auto',
[I 2021-08-29 15:47:39,286] Trial 13 finished with value: 36.23819944941056 and parameters: {'n_estimators': 200, 'max_depth': 15, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto',
[I 2021-08-29 15:47:39,456] Trial 14 finished with value: 37.19649270756004 and parameters: {'n_estimators': 100, 'max_depth': 16, 'min_samples_split': 4, 'min_samples_leaf': 2, 'max_features': 'auto',
[I 2021-08-29 15:47:40,037] Trial 15 finished with value: 36.4662674946946 and parameters: {'n_estimators': 300, 'max_depth': 14, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', '
[I 2021-08-29 15:47:40,289] Trial 16 finished with value: 36.015987978537325 and parameters: {'n_estimators': 200, 'max_depth': 17, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:40,504] Trial 17 finished with value: 37.89995545224416 and parameters: {'n_estimators': 200, 'max_depth': 12, 'min_samples_split': 3, 'min_samples_leaf': 2, 'max_features': 'log2',
[I 2021-08-29 15:47:40,803] Trial 18 finished with value: 35.231076820623805 and parameters: {'n_estimators': 200, 'max_depth': 18, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:41,341] Trial 19 finished with value: 37.23343530550354 and parameters: {'n_estimators': 500, 'max_depth': 18, 'min_samples_split': 4, 'min_samples_leaf': 2, 'max_features': 'log2',
[I 2021-08-29 15:47:41,485] Trial 20 finished with value: 57.53637068361058 and parameters: {'n_estimators': 200, 'max_depth': 2, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_features': 'log2', '
[I 2021-08-29 15:47:41,778] Trial 21 finished with value: 35.231076820623805 and parameters: {'n_estimators': 200, 'max_depth': 18, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:41,933] Trial 22 finished with value: 35.70077965473891 and parameters: {'n_estimators': 100, 'max_depth': 18, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:42,098] Trial 23 finished with value: 35.265361922541146 and parameters: {'n_estimators': 100, 'max_depth': 19, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:42,534] Trial 24 finished with value: 35.40626252319288 and parameters: {'n_estimators': 300, 'max_depth': 20, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:42,683] Trial 25 finished with value: 36.15258044518943 and parameters: {'n_estimators': 100, 'max_depth': 19, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 6, 'warm
[I 2021-08-29 15:47:43,126] Trial 26 finished with value: 35.767349331435966 and parameters: {'n_estimators': 300, 'max_depth': 17, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:43,760] Trial 27 finished with value: 35.65014664654578 and parameters: {'n_estimators': 500, 'max_depth': 14, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:43,995] Trial 28 finished with value: 37.046981489831126 and parameters: {'n_estimators': 200, 'max_depth': 19, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'log2',
[I 2021-08-29 15:47:44,155] Trial 29 finished with value: 35.60512186685815 and parameters: {'n_estimators': 100, 'max_depth': 17, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
Best Score: 35.231076820623805
```

제일 낮은 rmse값의 파라미터 조합을 찾기 위해 반복적으로 시도

하이퍼 파라미터 튜닝 결과

```
[I 2021-08-29 15:47:29,858] A new study created in memory with name: et_optimizer
[I 2021-08-29 15:47:31,222] Trial 0 finished with value: 36.30854722532004 and parameters: {'n_estimators': 700, 'max_depth': 15, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', '
[I 2021-08-29 15:47:32,252] Trial 1 finished with value: 37.5813918526017 [illegible] 1000 [illegible]: 13, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt',
[I 2021-08-29 15:47:32,903] Trial 2 finished with value: 37.26820547849423 [illegible] 600 [illegible]: 11, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 5, 'warm_
[I 2021-08-29 15:47:33,357] Trial 3 finished with value: 36.81363252749193 [illegible] 400 [illegible]: 18, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 5, 'warm_
[I 2021-08-29 15:47:34,012] Trial 4 finished with value: 38.74131749169292 and parameters: {'n_estimators': 700, 'max_depth': 8, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 4, 'warm_s
[I 2021-08-29 15:47:34,115] Trial 5 finished with value: 38.344489656756 and parameters: {'n_estimators': 100, 'max_depth': 9, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_features': 4, 'warm_sta
[I 2021-08-29 15:47:34,776] Trial 6 finished with value: 41.85845483168387 and parameters: {'n_estimators': 800, 'max_depth': 7, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 'sqrt', 'w
[I 2021-08-29 15:47:35,820] Trial 7 finished with value: 37.42930616021127 and parameters: [illegible] 'max_depth': 9, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_features': 6, 'warm_
[I 2021-08-29 15:47:36,426] Trial 8 finished with value: 40.23539903680105 and [illegible] 700 [illegible] 'max_depth': 8, 'min_samples_split': 3, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'w
[I 2021-08-29 15:47:36,986] Trial 9 finished with value: 47.91344730562039 and parameters: {'n_estimators': 800, 'max_depth': 4, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 'log2', 'w
[I 2021-08-29 15:47:37,677] Trial 10 finished with value: 36.57784820943486 and parameters: {'n_estimators': 400, 'max_depth': 20, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'auto',
[I 2021-08-29 15:47:38,194] Trial 11 finished with value: 36.58384153346146 and parameters: {'n_estimators': 300, 'max_depth': 20, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'auto',
[I 2021-08-29 15:47:38,882] Trial 12 finished with value: 36.26400097284186 and par [illegible] 400 [illegible] depth': 16, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'auto',
[I 2021-08-29 15:47:39,286] Trial 13 finished with value: 36.23819944941056 and parameters: {'n_estimators': 200, 'max_depth': 15, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto',
[I 2021-08-29 15:47:39,456] Trial 14 finished with value: 37.19649270756004 and parameters: {'n_estimators': 100, 'max_depth': 16, 'min_samples_split': 4, 'min_samples_leaf': 2, 'max_features': 'auto',
[I 2021-08-29 15:47:40,037] Trial 15 finished with value: 36.4662674946946 and parameters: {'n_estimators': 300, 'max_depth': 14, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', '
[I 2021-08-29 15:47:40,289] Trial 16 finished with value: 36.015987978537325 [illegible]: 17, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:40,504] Trial 17 finished with value: 37.89995545224416 and parameters: {'n_estimators': 200, 'max_depth': 12, 'min_samples_split': 3, 'min_samples_leaf': 2, 'max_features': 'log2',
[I 2021-08-29 15:47:40,803] Trial 18 finished with value: 35.231076820623805 and parameters: {'n_estimators': 200, 'max_depth': 18, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:41,341] Trial 19 finished with value: 37.23343530550354 and parameters: {'n_estimators': 500, 'max_depth': 18, 'min_samples_split': 4, 'min_samples_leaf': 2, 'max_features': 'log2',
[I 2021-08-29 15:47:41,485] Trial 20 finished with value: 57.53637068361058 [illegible]: 2, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_features': 'log2', '
[I 2021-08-29 15:47:41,778] Trial 21 finished with value: 35.231076820623805 and parameters: {'n_estimators': 200, 'max_depth': 18, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:41,933] Trial 22 finished with value: 35.70077965473891 and parameters: {'n_estimators': 100, 'max_depth': 18, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:42,098] Trial 23 finished with value: 35.265361922541146 and parameters: {'n_estimators': 100, 'max_depth': 19, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:42,534] Trial 24 finished with value: 35.4062625231928 [illegible]: 20, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:42,683] Trial 25 finished with value: 36.15258044518943 and parameters: {'n_estimators': 100, 'max_depth': 19, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 6, 'warm
[I 2021-08-29 15:47:43,126] Trial 26 finished with value: 35.767349331435966 and parameters: {'n_estimators': 300, 'max_depth': 17, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:43,760] Trial 27 finished with value: 35.6501466465 4578 and parameters: {'n_estimators': 500, 'max_depth': 14, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
[I 2021-08-29 15:47:43,995] Trial 28 finished with value: 37.046981489831126 and parameters: {'n_estimators': 200, 'max_depth': 19, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'log2',
[I 2021-08-29 15:47:44,155] Trial 29 finished with value: 35.6051218668 5815 and parameters: {'n_estimators': 100, 'max_depth': 17, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2',
Best Score: 35.231076820623805
```

튜닝 최종 결과

n_estimator : 507

Max_depth : 14

Min_samples_split : 2

Min_samples_leaf : 1

Max_features : auto

Test Data 적용

```
# 독립변수, 종속변수 설정
X = an[an.columns.difference(['count', 'hour_bef_visibility'])]
colls = X.columns.tolist()
X = np.column_stack((X['hour']**5, X['hour_bef_temperature']**4, X))
y = an[['count']]

# 테스트 셋 전처리 (결측치, 더미 변수 추가)
it_test = test[test.columns.difference(['id', 'hour_bef_visibility'])].copy()
it_test = busyHourGen(it_test, 'hour')
it_test = it_test[colls]
X_test = np.column_stack((it_test['hour']**5, it_test['hour_bef_temperature']**4, it_test))
X_test = IterativeImputer(random_state=2021).fit_transform(X_test)

# ExtraTreesRegressor
best_reg = ExtraTreesRegressor(n_estimators= 507, max_depth=14, warm_start= True, random_state=2021)
best_reg.fit(X, y)
predicts = best_reg.predict(X_test)

submission['count'] = predicts
submission.to_csv('final.csv', index=False)
```

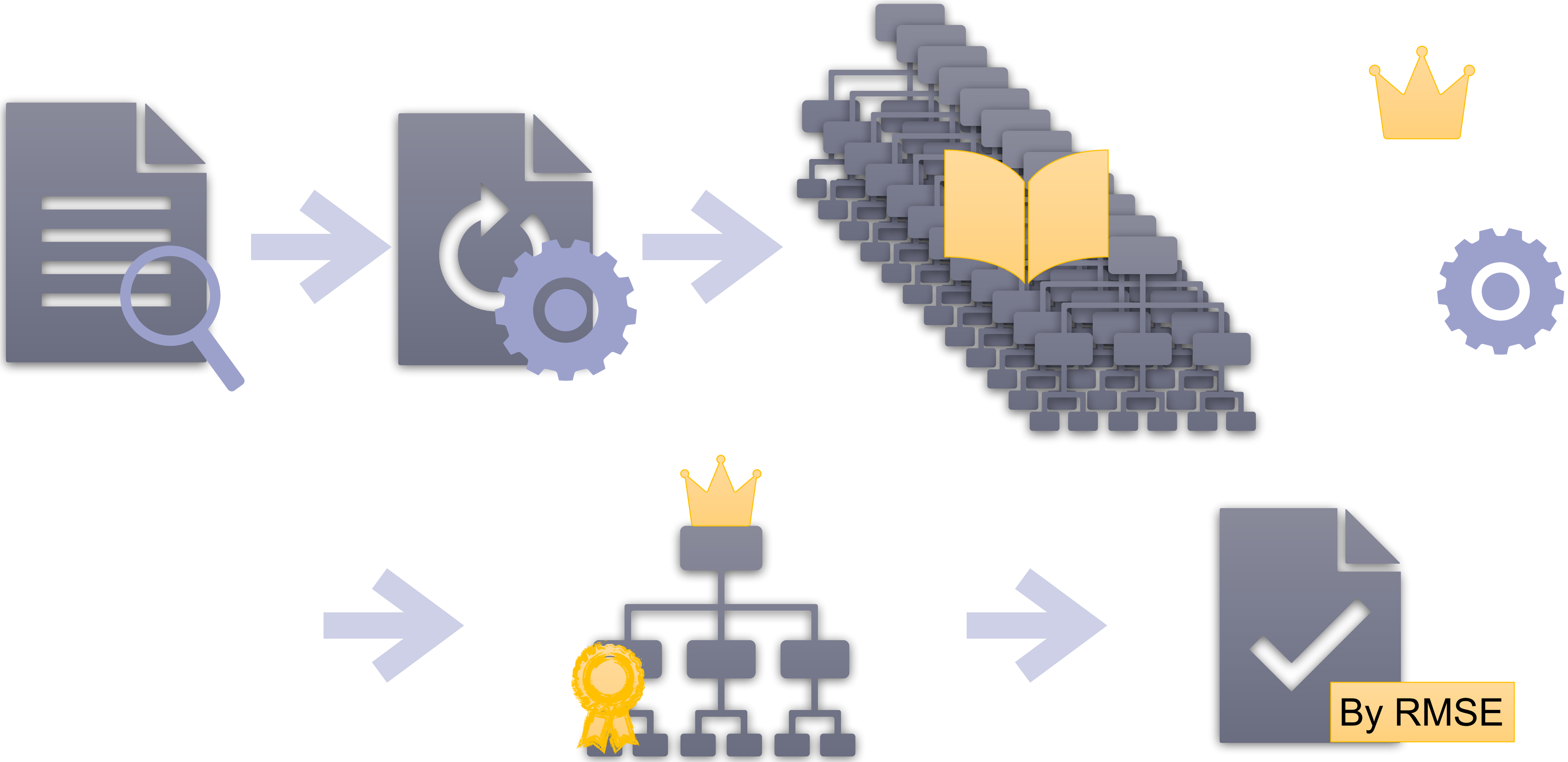
Train 데이터와 동일하게 Test 데이터에 이상치 제거 제외한 전처리를 한 후 최종 모델 훈련 및 예측

최종 결과

1	수갱		29.31283	41	2일 전

최종 제출 모델 rmse 대략 29.3으로 해당 대회 전체 1등!!!!

최종 정리



Q/A

Thank You ☺