서울시 배달건수 회귀분석

PURPOSE

효율적인
식자재 관리 활용

배달 관련
마케팅 활용

DATA STRUCTURE

DATA 구성

기상청 날씨 데이터

서울시 배달
통화건수 데이터

#강남구 #치킨

```
raw_data.head()
```

	연	월	일	시간대	요일	기온	강수량	풍속	습도	적설량	미세먼지	초미세먼지	공휴일	통화건수
0	2016	10	1	0	토	19.4	0.0	0.8	73.0	0.0	48.0	26.0	0	67.0
1	2016	10	1	1	토	18.8	0.0	1.3	77.0	0.0	48.0	26.0	0	27.0
2	2016	10	1	10	토	20.7	0.0	1.8	72.0	0.0	48.0	26.0	0	10.0
3	2016	10	1	11	토	21.6	0.0	0.6	65.0	0.0	48.0	26.0	0	28.0
4	2016	10	1	12	토	24.0	0.0	1.5	55.0	0.0	48.0	26.0	0	64.0

DATA 구성

```
raw_data.describe()
```

	연	월	일	시간대	기온	강수량	풍속	습도	적설량	미세먼지	초미세먼지	공휴일	통화건수
count	26280.000000	26280.000000	26280.000000	26280.000000	26280.000000	26280.000000	26280.000000	26280.000000	26280.00000	26280.000000	26280.000000	26280.000000	26280.000000
mean	2017.747945	6.526027	15.720548	11.500000	13.182089	0.129540	1.969376	57.405685	0.05118	40.762988	24.563379	0.014612	45.567352
std	0.924782	3.447917	8.796414	6.922318	11.339561	1.137908	1.131744	20.192473	0.38196	23.644145	16.960574	0.217432	51.339346
min	2016.000000	1.000000	1.000000	0.000000	-17.800000	0.000000	0.000000	7.000000	0.00000	3.000000	1.000000	-1.000000	0.000000
25%	2017.000000	4.000000	8.000000	5.750000	3.700000	0.000000	1.100000	42.000000	0.00000	25.000000	13.000000	0.000000	5.000000
50%	2018.000000	7.000000	16.000000	11.500000	14.300000	0.000000	1.800000	57.000000	0.00000	36.000000	21.000000	0.000000	25.000000
75%	2018.000000	10.000000	23.000000	17.250000	22.800000	0.000000	2.700000	73.000000	0.00000	52.000000	31.000000	0.000000	72.000000
max	2019.000000	12.000000	31.000000	23.000000	39.400000	50.500000	9.100000	100.000000	8.80000	200.000000	151.000000	1.000000	608.000000

REGRESSION FLOW

분석 방법 (구조)

데이터 가공
데이터 전처리

피처와 타겟
전처리

모델 학습
예측 평가

모델 검증

데이터 가공 | 데이터 전처리

```
df_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26280 entries, 0 to 26279
Data columns (total 15 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   일자       26280 non-null  object
 1   연        26280 non-null  object
 2   월        26280 non-null  object
 3   일        26280 non-null  object
 4   시간대      26280 non-null  object
 5   요일       26280 non-null  object
 6   기온       26274 non-null  float64
 7   강수량      2573 non-null   float64
 8   풍속       26231 non-null  float64
 9   습도       26275 non-null  float64
 10  적설량      812 non-null    float64
 11  통화건수     21805 non-null  float64
 12  미세먼지     24720 non-null  float64
 13  초미세먼지    24720 non-null  float64
 14  공휴일      1248 non-null   object
dtypes: float64(8), object(7)
memory usage: 3.2+ MB
```

```
df_2["통화건수"].fillna(0, inplace=True)
df_2["강수량"].fillna(0, inplace=True)
df_2["적설량"].fillna(0, inplace=True)
```

```
# 3. 미세먼지, 초미세먼지 결측치
mise_null = list(df_2[df_2['미세먼지'].isnull()].index)
```

```
for i in mise_null:
    idx = munji_mean[munji_mean['연월'] == df_2.iloc[i, 0][:6]].index
    df_2.iloc[i,12] = munji_mean.iloc[idx, 1][list(idx)[0]]
    df_2.iloc[i,13] = munji_mean.iloc[idx, 2][list(idx)[0]]
```

데이터 가공 | 데이터 전처리

```
# ['연', '월', '일', '시간대', '요일', '공휴일'] 인코딩

df = pd.get_dummies(raw_data, columns=['연', '월', '일', '시간대', '요일', '공휴일'])
df.info()
```

```
 67  시간대_12   26280 non-null  uint8
 68  시간대_13   26280 non-null  uint8
 69  시간대_14   26280 non-null  uint8
 70  시간대_15   26280 non-null  uint8
 71  시간대_16   26280 non-null  uint8
 72  시간대_17   26280 non-null  uint8
 73  시간대_18   26280 non-null  uint8
 74  시간대_19   26280 non-null  uint8
 75  시간대_20   26280 non-null  uint8
 76  시간대_21   26280 non-null  uint8
 77  시간대_22   26280 non-null  uint8
 78  시간대_23   26280 non-null  uint8
 79  요일_금     26280 non-null  uint8
 80  요일_목     26280 non-null  uint8
 81  요일_수     26280 non-null  uint8
 82  요일_월     26280 non-null  uint8
 83  요일_일     26280 non-null  uint8
 84  요일_토     26280 non-null  uint8
 85  요일_화     26280 non-null  uint8
 86  공휴일_-1   26280 non-null  uint8
 87  공휴일_0    26280 non-null  uint8
 88  공휴일_1    26280 non-null  uint8
dtypes: float64(8), uint8(81)
memory usage: 3.8 MB
```

시간형 values
원핫인코딩으로 처리

피처와 타겟 전처리

```
y_target = df['통화건수']
X_features = df.drop('통화건수',axis=1)
```

```
# 데이터 분리
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

X_train, X_test, y_train, y_test = train_test_split
(X_features,y_target, test_size= 0.3, random_state=13)

lr_reg= LinearRegression()
lr_reg.fit(X_train,y_train)
```

```
# 모델 예측
pred = lr_reg.predict(X_test)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)

mae_val = mean_absolute_error(y_test, pred)

r2 = r2_score(y_test, pred)
print(rmse, mae_val, r2)
```

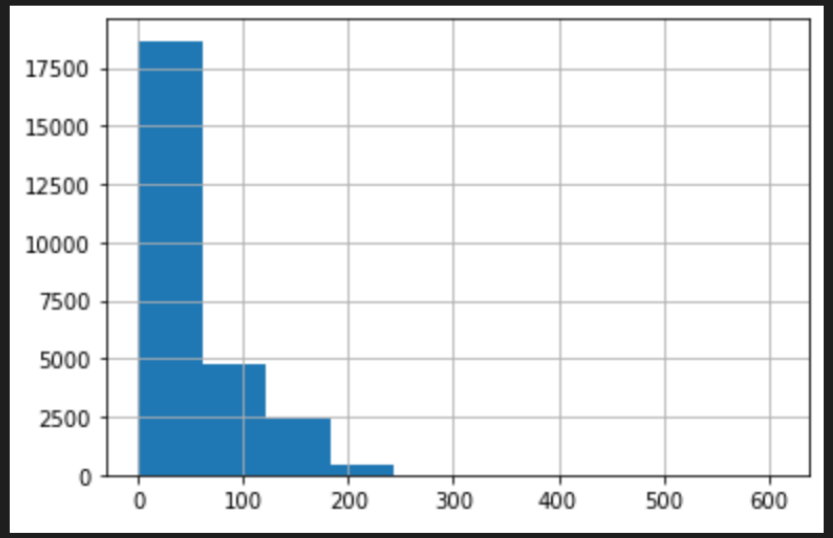
```
17.7789327231529 12.008100810301594 0.8770174841490421
```

전처리 이전 선형회귀 모델 학습|평가

피처와 타겟 전처리

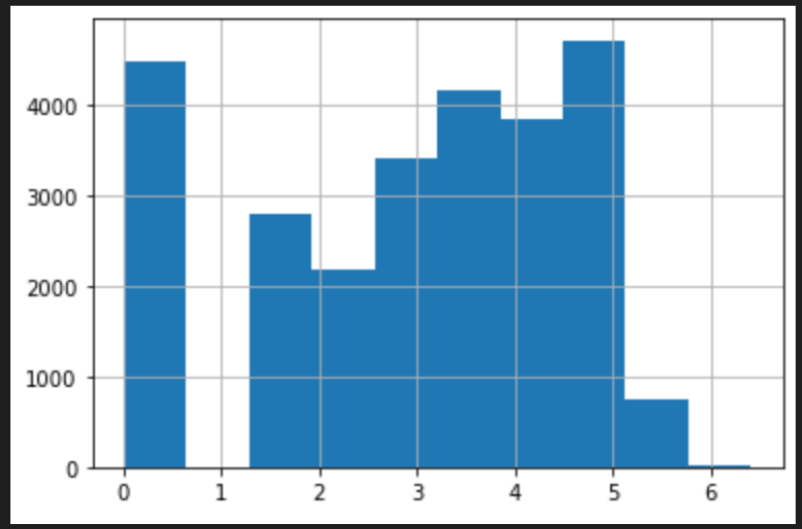
```
y_target.hist() #분포도 확인
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x264737af7c0>
```




```
y_target_log = np.log1p(y_target)
y_target_log.hist() #정규분포의 형태는 아니지만 왜곡도를 낮춤
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2647386d340>
```



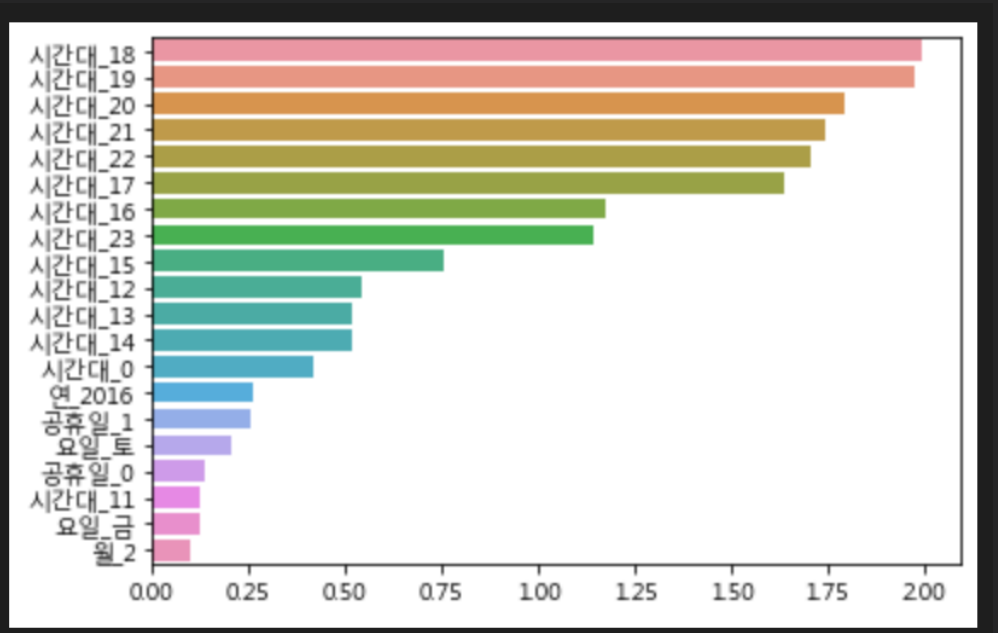
타겟(통화건수)에 대한 왜곡도 낮춤

모델 학습 | 예측 | 평가

```
print('RMSE : {} | MAE : {} | r2 : {} '.format(round
(rmse_val,2),round(mae_val,2),round(r2,3)))
```

```
RMSE : 15.01 | MAE : 8.62 | r2 : 0.912
```

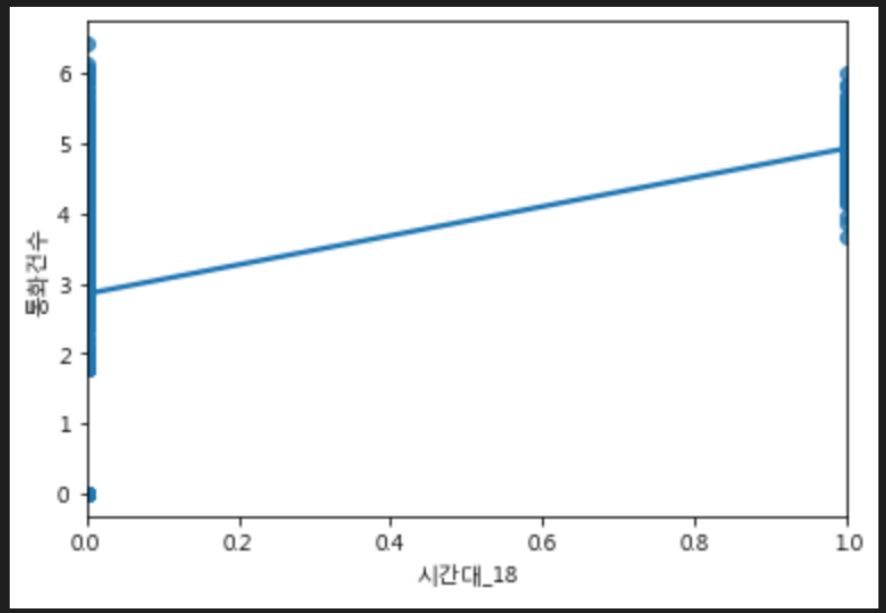
피처와 타겟 전처리



상위 회귀 계수의 이상치 없음

```
sns.regplot(x=X_features['시간대_18'], y=y_target_log, data
=df )
# 회귀계수 상위 5개 이상치 확인되지 않음
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26473c1c100>
```



모델 학습 | 예측 | 평가

```
# 여러 모델의 성능 확인 함수
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

X_train, X_test, y_train, y_test =train_test_split
(X_features, y_target_log, test_size=0.3, random_state=13)
```

```
#모델별로 평가 확인

lr_reg = LinearRegression()
ridge_reg = Ridge(alpha=0.1)
lasso_reg = Lasso(alpha=0)
tree_reg = DecisionTreeRegressor(random_state=13)
forest_reg = RandomForestRegressor(n_estimators=100,
random_state=13)

for model in [lr_reg, ridge_reg, lasso_reg,tree_reg,
forest_reg]:
    get_model_predict(model,X_train, X_test, y_train, y_test,
 is_expm1=True)
```

```
### LinearRegression ###
RMSE : 15.00659 | MAE : 8.61614 | r2 : 0.91238
### Ridge ###
RMSE : 15.0076 | MAE : 8.61591 | r2 : 0.91237
### Lasso ###
RMSE : 15.00659 | MAE : 8.61614 | r2 : 0.91238
### DecisionTreeRegressor ###
RMSE : 16.68572 | MAE : 9.71449 | r2 : 0.89168
### RandomForestRegressor ###
RMSE : 12.31447 | MAE : 7.25513 | r2 : 0.941
```

모델 학습 | 예측 | 평가

```
# Ridge 하이퍼파라미터튜닝 alpha값이 클수록(penalty 증가)
계수의 크기가 줄어듬 -> 영향력이 큰 계수의 영향력을 줄임 /
변수를 축소, 다중공선성을 방지
from sklearn.model_selection import GridSearchCV
def ridge_grid_search_cv(X_train, y_train):
    param_grid = [
        {'alpha': [0, 0.05, 0.1, 0.5, 1, 5]},
        ]

    grid_search = GridSearchCV(ridge, param_grid, cv=5,
                               scoring='r2',
                               return_train_score=True)
    grid_search.fit(X_train, y_train)

    print ('best_params_: ', grid_search.best_params_)
    cvres = grid_search.cv_results_
    for mean_test_score, params in zip(cvres
["mean_test_score"], cvres["params"]):
        print(mean_test_score, params)
```

```
ridge_grid_search_cv(X_train, y_train)
```

```
best_params_:  {'alpha': 0.1}
0.8926994523234925 {'alpha': 0}
0.89365976166098 {'alpha': 0.05}
0.8936598300530756 {'alpha': 0.1}
0.8936597497457823 {'alpha': 0.5}
0.893658091320399 {'alpha': 1}
0.8935845919935612 {'alpha': 5}
```

릿지 모델의 베스트 파라미터 확인

모델 학습 | 예측 | 평가

```
# Lasso 하이퍼파라미터튜닝 alpha 조금만 키워도 계수가 완전히
0이 되는 변수 증가 -> feaure selection, 중요한 변수만 택함,
def lasso_grid_search_cv(X_train, y_train):
    param_grid = [
        {'alpha': [0, 0.05, 0.1, 0.5, 1]},
        ]

    grid_search = GridSearchCV(lasso, param_grid, cv=5,
                               scoring='r2',
                               return_train_score=True)
    grid_search.fit(X_train, y_train)
    print ('best_params_: ', grid_search.best_params_)
    cvres = grid_search.cv_results_
    for mean_test_score, params in zip(cvres
["mean_test_score"], cvres["params"]):
        print(mean_test_score, params)
```

```
lasso_grid_search_cv(X_train, y_train)
```

```
best_params_:  {'alpha': 0}
0.8936596757775288 {'alpha': 0}
0.5070852480294363 {'alpha': 0.05}
0.14059739131969196 {'alpha': 0.1}
0.09202316846958503 {'alpha': 0.5}
0.08673239302422908 {'alpha': 1}
```

라쏘 모델의 베스트 파라미터 확인

모델 학습 | 예측 | 평가

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [30, 50, 70, 100], 'max_features':[2,4,
6,8]},
    {'bootstrap':[False], 'n_estimators': [3, 10],
'max_features':[2,3,4] }
    ]

forest_reg = RandomForestRegressor(random_state=13)
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=13),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [30, 50, 70, 100]},
                         {'bootstrap': [False], 'max_features': [2,
3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_erro
r')
```

```
grid_search.best_params_
```

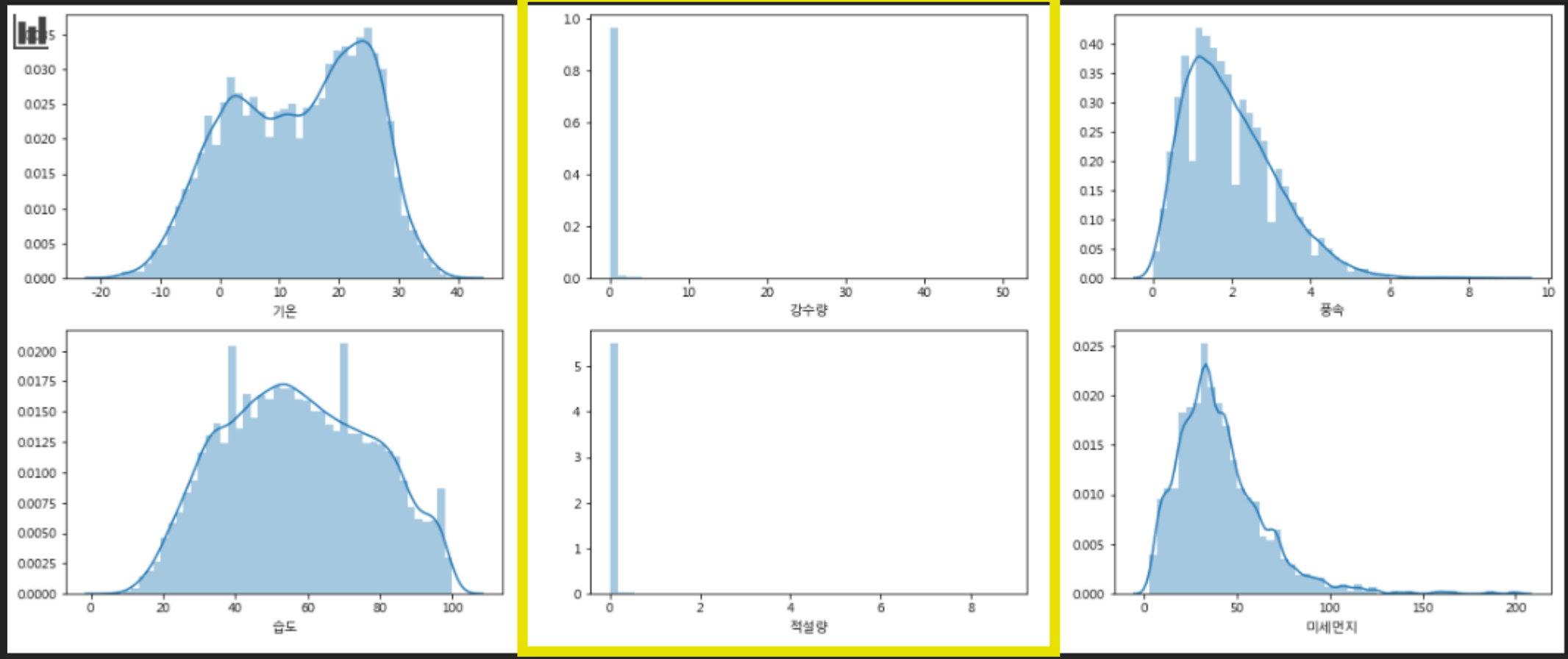
```
{'max_features': 8, 'n_estimators': 100}
```

```
grid_search.best_estimator_
```

```
RandomForestRegressor(max_features=8, random_state=13)
```

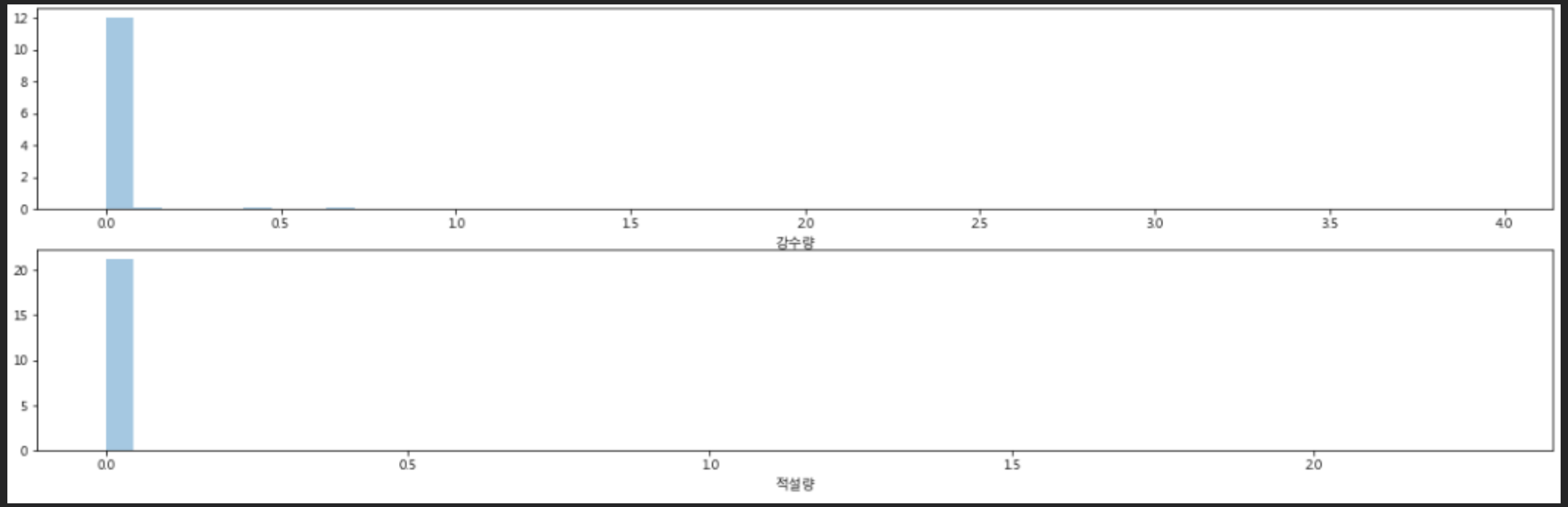
랜덤포레스트 모델의 베스트 파라미터 확인

피처와 타겟 스케일링 – 1. 피쳐 로그화



강수량 | 적설량 0에 편중되어 로그화 진행

피처와 타겟 스케일링 – 1. 피쳐 로그화



X축의 범위가 줄어듬을 확인

모델 학습 | 예측 | 평가

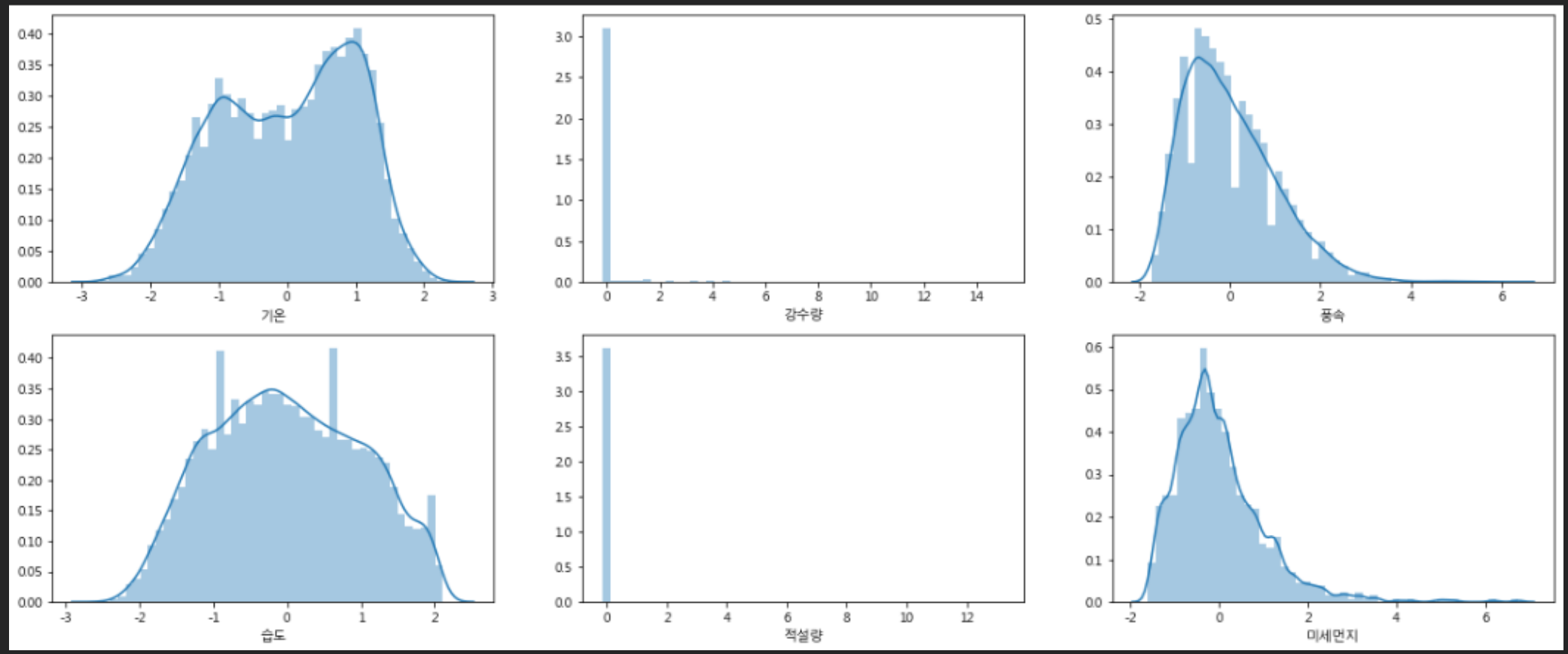
```
### LinearRegression ###
RMSE : 17.78316 | MAE : 12.01545 | r2 : 0.87696
### Ridge ###
RMSE : 17.78291 | MAE : 12.01478 | r2 : 0.87696
### Lasso ###
RMSE : 17.78316 | MAE : 12.01545 | r2 : 0.87696
### DecisionTreeRegressor ###
RMSE : 19.50873 | MAE : 10.54059 | r2 : 0.85192
### RandomForestRegressor ###
RMSE : 13.73436 | MAE : 7.83157 | r2 : 0.92661
```

피처와 타겟 스케일링– 2. Standard Scaler

```python
from sklearn.preprocessing import StandardScaler
# 데이터 분류
scaled_cols = ["기온", "강수량", "풍속", "습도", "적설량",
"미세먼지", "초미세먼지"]

scaler = StandardScaler()
scaler.fit(X_features[scaled_cols])
X_scaled = scaler.transform(X_features[scaled_cols])
X_features[scaled_cols] = X_scaled
```

```python
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(20,8))
sns.distplot(X_features["기온"], ax=axes[0, 0], label="기온")
sns.distplot(X_features["강수량"], ax=axes[0, 1],
label="강수량")
sns.distplot(X_features["풍속"], ax=axes[0, 2], label="풍속")
sns.distplot(X_features["습도"], ax=axes[1, 0], label="습도")
sns.distplot(X_features["적설량"], ax=axes[1, 1],
label="적설량")
sns.distplot(X_features["미세먼지"], ax=axes[1, 2],
label="미세먼지")
```

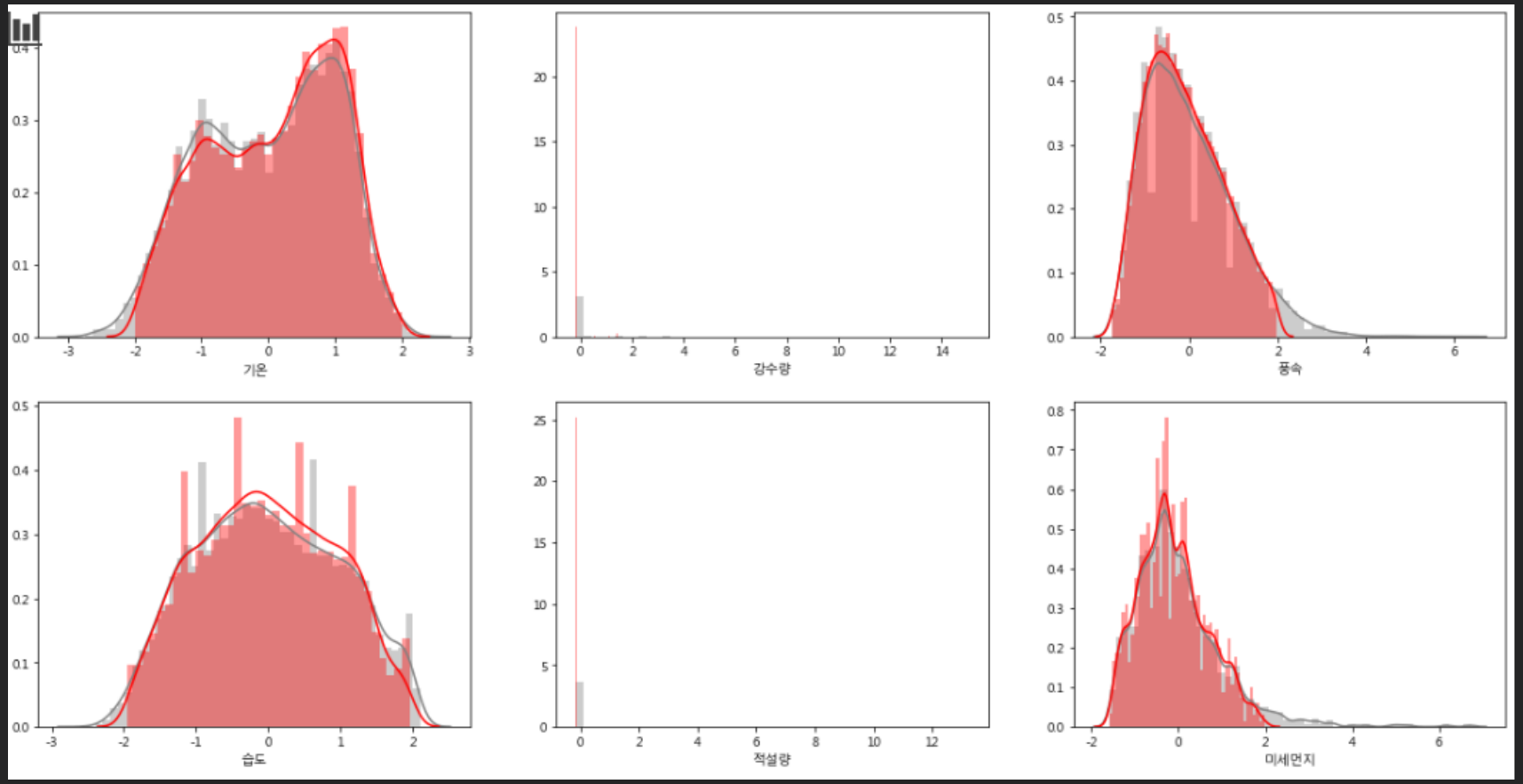
피처와 타겟 스케일링 – 2. Standard Scaler



모델 학습 | 예측 | 평가

```
### LinearRegression ###
RMSE : 17.78452 | MAE : 12.01755 | r2 : 0.87694
### Ridge ###
RMSE : 17.78291 | MAE : 12.01478 | r2 : 0.87696
### Lasso ###
RMSE : 17.78316 | MAE : 12.01545 | r2 : 0.87696
### DecisionTreeRegressor ###
RMSE : 19.52474 | MAE : 10.54148 | r2 : 0.85168
### RandomForestRegressor ###
RMSE : 13.73766 | MAE : 7.8293 | r2 : 0.92657
```

피처와 타겟 스케일링 – 3. z-score 이상치 제거



피처와 타겟 전처리 – 3. 이상치 제거

```
import scipy as sp
import scipy.stats

# check Z score
df_Zscore = pd.DataFrame()
outlier_dict = {}
outlier_idx_list = []

for one_col in df2[scaled_cols]:
    print("Check",one_col)
    df_Zscore[f'{one_col}_Zscore'] = sp.stats.zscore(df2
[one_col])
    outlier_dict[one_col] = df_Zscore[f'{one_col}_Zscore'][
(df_Zscore[f'{one_col}_Zscore']>2)|(df_Zscore[f'{one_col}
_Zscore']<-2)]
    outlier_idx_list.append(list(outlier_dict[one_col].index)
)
    if len(outlier_dict[one_col]):
        print(one_col,'Has outliers\n', outlier_dict[one_col]
)
    else:
        print(one_col,"Has Not outlier")
    print()
```

```
print("이상치 제거 전", df2.shape)
all_outlier_idx = sum(outlier_idx_list,[])
df2 = df2.drop(all_outlier_idx)
print("이상치 제거 후", df2.shape)
```

```
이상치 제거 전 (26280, 89)
이상치 제거 후 (22059, 89)
```

모델 학습 | 예측 | 평가

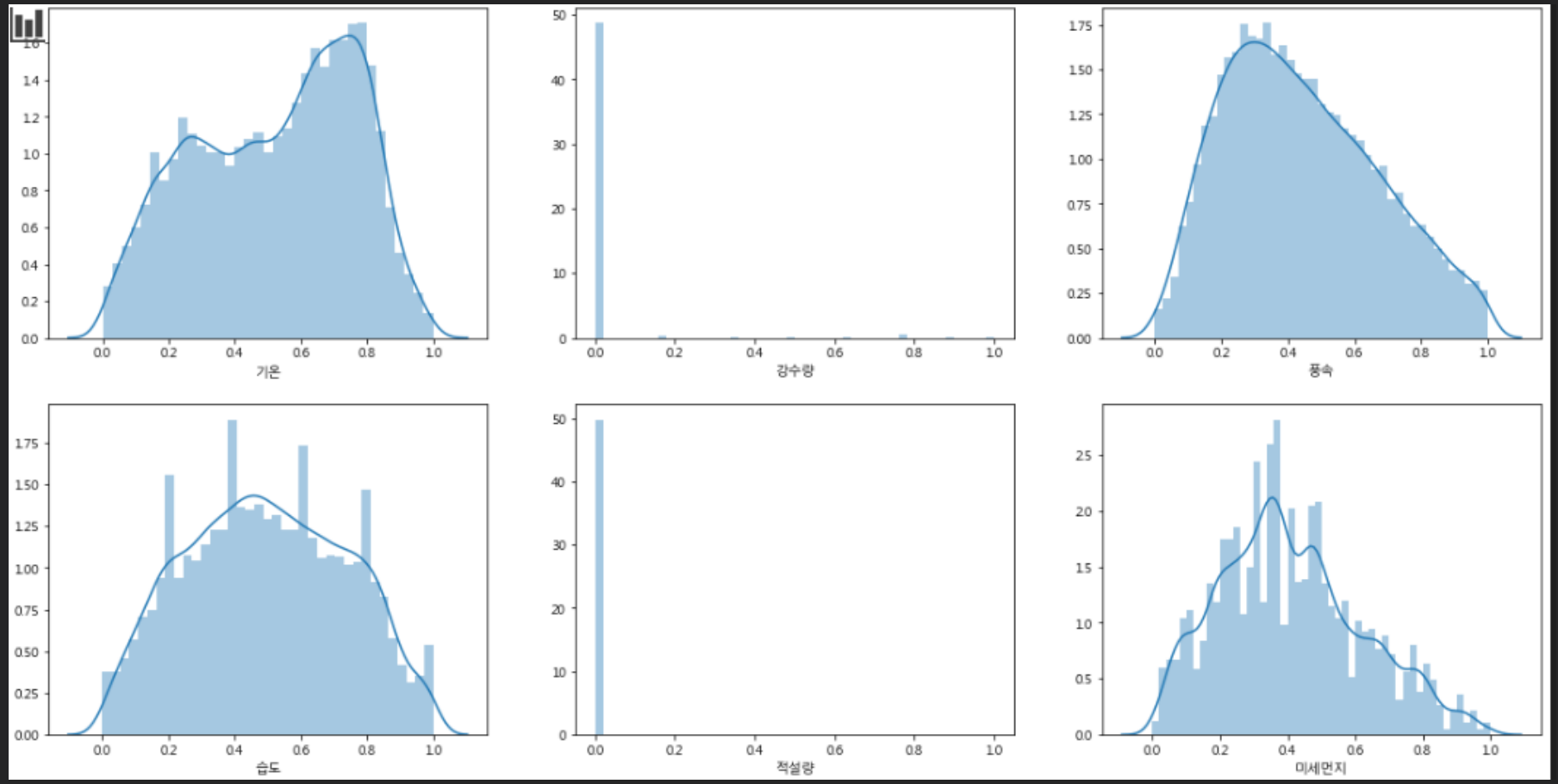
```
### LinearRegression ###
RMSE : 18.52616 | MAE : 12.10483 | r2 : 0.86883
### Ridge ###
RMSE : 18.52565 | MAE : 12.10339 | r2 : 0.86883
### Lasso ###
RMSE : 18.52562 | MAE : 12.1041 | r2 : 0.86883
### DecisionTreeRegressor ###
RMSE : 18.90954 | MAE : 10.41659 | r2 : 0.86334
### RandomForestRegressor ###
RMSE : 14.89225 | MAE : 8.03347 | r2 : 0.91524
```

피처와 타겟 스케일링 – 4. MinmaxScaler

```
from sklearn.preprocessing import MinMaxScaler

scaled_cols = ["기온", "강수량", "풍속", "습도", "적설량",
"미세먼지", "초미세먼지"]

scaler = MinMaxScaler()
scaler.fit(X_features[scaled_cols])
X_scaled = scaler.transform(X_features[scaled_cols])
X_features[scaled_cols] = X_scaled

print('feature들의 최소 값')
print(X_features[scaled_cols].min())
print('\nfeature들의 최대 값')
print(X_features[scaled_cols].max())
```

```
feature들의 최소 값
기온        0.0
강수량       0.0
풍속        0.0
습도        0.0
적설량       0.0
미세먼지      0.0
초미세먼지     0.0
dtype: float64

feature들의 최대 값
기온        1.0
강수량       1.0
풍속        1.0
습도        1.0
적설량       1.0
미세먼지      1.0
초미세먼지     1.0
dtype: float64
```

피처와 타겟 스케일링 – 4. Minmax Scaler



모델 학습 | 예측 | 평가

```
### LinearRegression ###
RMSE : 18.54645 | MAE : 12.14396 | r2 : 0.86854
### Ridge ###
RMSE : 18.52565 | MAE : 12.10338 | r2 : 0.86883
### Lasso ###
RMSE : 18.52562 | MAE : 12.1041 | r2 : 0.86883
### DecisionTreeRegressor ###
RMSE : 18.90604 | MAE : 10.42823 | r2 : 0.86339
### RandomForestRegressor ###
RMSE : 14.89023 | MAE : 8.03255 | r2 : 0.91526
```

모델 검증 K-Fold

```
# k-fold 교차검증준비
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, X_test, y_test, scoring =
"neg_mean_squared_error", cv=5)
tree_rmse_scores = np.sqrt(-scores)
```

```
def display_socres(model):
    scores = cross_val_score(model, X_test, y_test, scoring =
"neg_mean_squared_error", cv=5)
    model_rmse_scores =np.sqrt(-scores)
    print('###', model.__class__.__name__,'###')
    print("점수:", model_rmse_scores)
    print("평균:", model_rmse_scores.mean())
    print("표준편차:", model_rmse_scores.std())

for model in [lr_reg, ridge_reg, lasso_reg,tree_reg,forest_reg]:
    display_socres(model)
```

모델 검증 K-Fold

```
### LinearRegression ###
점수: [17.81665894 17.08861769 18.85129292 16.31935925 18.94199735]
평균: 17.80358522983982
표준편차: 1.0107373381159623
### Ridge ###
점수: [17.81541732 17.08799691 18.85066075 16.31967703 18.94384639]
평균: 17.803519679958615
표준편차: 1.0110146210269217
### Lasso ###
점수: [17.81665894 17.08862526 18.85129564 16.31934381 18.94199735]
평균: 17.80358420111108
표준편차: 1.0107413637181468
### DecisionTreeRegressor ###
점수: [20.1471914  19.0110771  18.6654116  18.14714744 18.06279585]
평균: 18.80672467956527
표준편차: 0.7545929477603188
### RandomForestRegressor ###
점수: [14.23753136 14.31239192 15.16196213 12.90057537 14.53227398]
평균: 14.228946953231006
표준편차: 0.7394771683661917
```

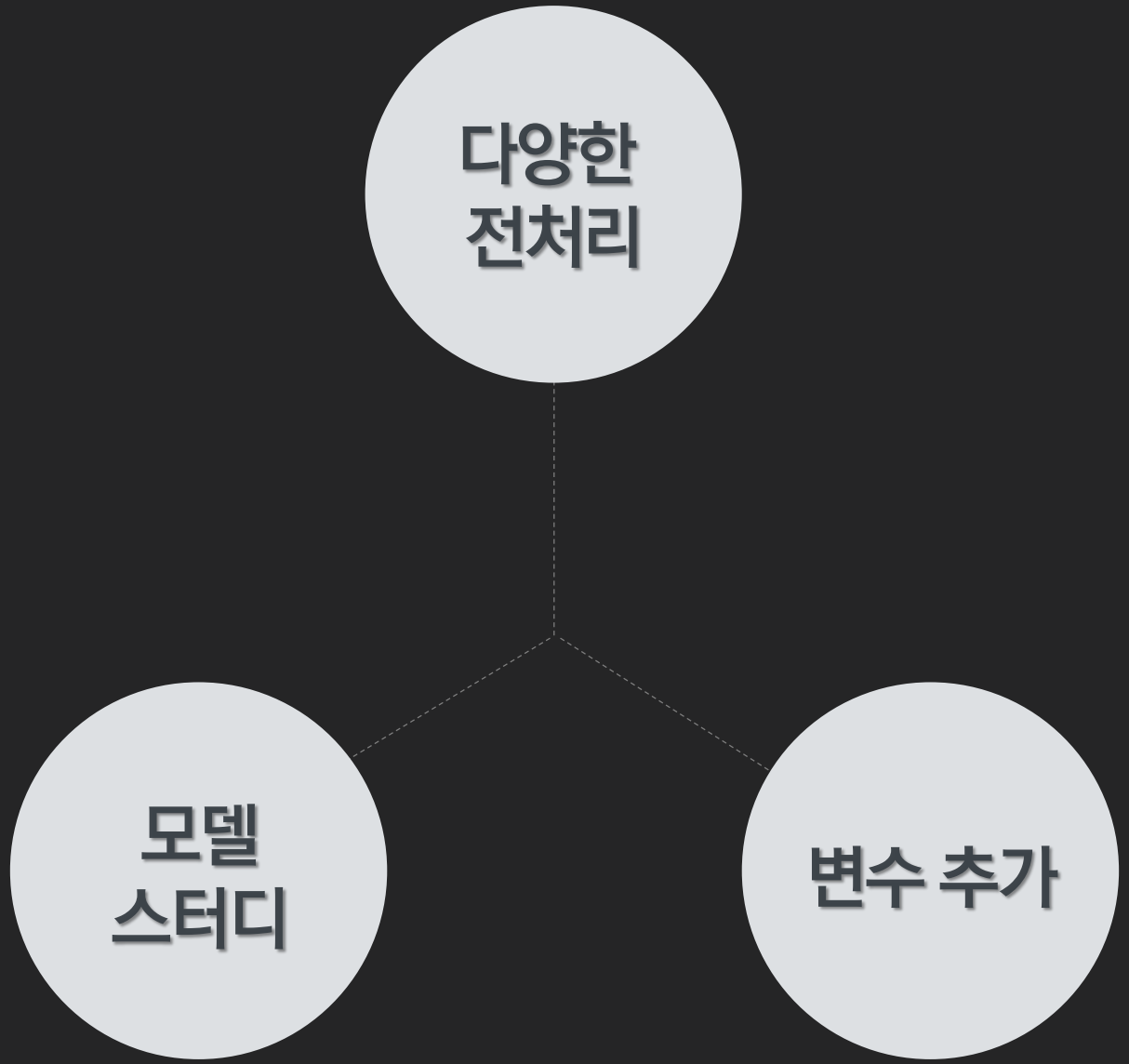
모델 검증 K-Fold

```
def display_socres(model):
    scores = cross_val_score(model, X_test, y_test, scoring =
"r2", cv=5)
    print('###', model.__class__.__name__,'###')
    print("점수:", scores)
    print("평균:", scores.mean())
    print("표준편차:", scores.std())

for model in [lr_reg, ridge_reg, lasso_reg,tree_reg,forest_reg]:
    display_socres(model)
```

모델 검증 K-Fold

```
### LinearRegression ###
점수: [0.87519632 0.88097532 0.85790961 0.89568484 0.87160951]
평균: 0.8762751206322978
표준편차: 0.012321145020713 59
### Ridge ###
점수: [0.87521371 0.88098397 0.85791914 0.89568078 0.87158444]
평균: 0.8762764089684838
표준편차: 0.012319286475601 37
### Lasso ###
점수: [0.87519632 0.88097522 0.85790957 0.89568504 0.87160951]
평균: 0.8762751308026158
표준편차: 0.012321211361901637
### DecisionTreeRegressor ###
점수: [0.84041066 0.85268852 0.86069793 0.87100944 0.88325151]
평균: 0.8616116110900766
표준편차: 0.014741379919141052
### RandomForestRegressor ###
점수: [0.92030258 0.91650748 0.90808355 0.93481324 0.92443017]
평균: 0.9208274038134686
표준편차: 0.008830930500885665
```

GOAL



END OF DOCUMENT