**# # 패키지 불러오기**

import numpy as np

import pandas as pd

import json

import datetime as dt

import matplotlib.pyplot as plt

import matplotlib.ticker as ticker

get\_ipython().run\_line\_magic('matplotlib', 'inline')

import seaborn as sns

import re

import glob

import os

from scipy import stats

from scipy.integrate import trapz

import missingno as msno

import warnings

warnings.filterwarnings('ignore')

pd.set\_option('max\_columns', 20, 'max\_rows', 20, 'max\_colwidth', 10)

# 전처리

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.preprocessing import Binarizer

from sklearn.preprocessing import PolynomialFeatures

from sklearn.decomposition import PCA

## 알고리즘(회귀)

from sklearn import linear\_model

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.linear\_model import Ridge, Lasso, ElasticNet

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

import statsmodels.api as sm

## 데이터 셋 만들기

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, KFold

from sklearn.model\_selection import cross\_val\_score, GridSearchCV, RepeatedStratifiedKFold

from sklearn.metrics import make\_scorer

# 파이프라인

from sklearn.pipeline import Pipeline

## 데이터 평가

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score

from sklearn.metrics import roc\_auc\_score, roc\_curve, f1\_score

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from sklearn.metrics import silhouette\_score, silhouette\_samples

from sklearn.metrics import precision\_recall\_curve

**# # 데이터 입력 및 조정**

train = pd.read\_csv(r'F:\OneDrive - 퍼즐데이터\01\_sejin\03\_ADP\02\_실기\03\_연습\5장\bike-sharing-demand\train.csv')

test = pd.read\_csv(r'F:\OneDrive - 퍼즐데이터\01\_sejin\03\_ADP\02\_실기\03\_연습\5장\bike-sharing-demand\test.csv')

bike\_df = pd.concat([train, test], axis=0)

bike\_df = bike\_df.reset\_index(drop=True)

bike\_df['datetime'] = pd.to\_datetime(bike\_df['datetime'])

bike\_df['year'] = bike\_df['datetime'].dt.year

bike\_df['month'] = bike\_df['datetime'].dt.month

bike\_df['day'] = bike\_df['datetime'].dt.day

bike\_df['hour'] = bike\_df['datetime'].dt.hour

bike\_df = bike\_df.drop('datetime', axis=1)

train\_1 = bike\_df[:train.shape[0]]

test\_1 = bike\_df[:test.shape[0]]

X\_features = train\_1.drop('count', axis=1)

y\_target = train\_1[['count']]

**# # 데이터 확인**

**# ## y값 정규성 확인**

sns.distplot(y\_target)

sns.distplot(np.log1p(y\_target))

scaler = StandardScaler()

y\_target\_S = scaler.fit\_transform(y\_target)

sns.distplot(y\_target\_S)

y\_target\_S

sns.distplot(np.log1p(y\_target\_S))

**# ## x값 선형성 확인**

# 2개의 행과 4개의 열을 가진 subplots를 이용. axs는 4x2개의 ax를 가짐.

fig, axs = plt.subplots(figsize=(16,16) , ncols=4 , nrows=4)

features = ['season', 'holiday', 'workingday', 'weather', 'temp', 'atemp',

'humidity', 'windspeed', 'casual', 'registered', 'year',

'month', 'day', 'hour']

for i , feature in enumerate(features):

row = int(i/4)

col = i%4

# 시본의 regplot을 이용해 산점도와 선형 회귀 직선을 함께 표현

sns.regplot(x=feature , y='count',data=train\_1 , ax=axs[row][col])

**# # 전처리**

# \* y값을 먼저 정규화하고 로그값을 씌여 정규분표화로 만든다. 그래야 모델학습이 잘 됨

# \* 다음으로 X값을 정규분포로 만듬

**# ## 컬럼 늘리기**

# ### Polynomial Regression

# 다항식으로 변환한 단항식 생성, [[0,1],[2,3]]의 2X2 행렬 생성

X = np.arange(4).reshape(2,2)

print('일차 단항식 계수 feature:\n',X )

# degree = 2 인 2차 다항식으로 변환하기 위해 PolynomialFeatures를 이용하여 변환

poly = PolynomialFeatures(degree=2)

poly.fit(X)

poly\_ftr = poly.transform(X)

print('변환된 2차 다항식 계수 feature:\n', poly\_ftr)

**# ## 데이터 분리**

**# ### train\_test\_split**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_features, y\_target, test\_size = 0.2, random\_state = 156, stratify=y\_target)

print('학습 세트 Shape:{0}, 테스트 세트 Shape:{1}'.format(X\_train.shape , X\_test.shape))

print(' 학습 세트 레이블 값 분포 비율')

print(y\_train.value\_counts()/y\_train.count())

print('\n 테스트 세트 레이블 값 분포 비율')

print(y\_test.value\_counts()/y\_test.count())

**# ### K-fold(전처리 후 진행해야 함)**

# - 1,2,3,4 폴드로 한번씩 나눠서 뿌리기 때문에 예측도 같이 돌려야 함

ridge\_reg = Ridge(random\_state=156)

# 5개의 폴드 세트로 분리하는 KFold 객체와 폴드 세트별 정확도를 담을 리스트 객체 생성.

kfold = KFold(n\_splits=5)

n\_iter = 0

cv\_mse = []

# KFold객체의 split( ) 호출하면 폴드 별 학습용, 검증용 테스트의 로우 인덱스를 array로 반환

for train\_index, test\_index in kfold.split(X\_features):

# kfold.split( )으로 반환된 인덱스를 이용하여 학습용, 검증용 테스트 데이터 추출

X\_train, X\_test = X\_features[train\_index], X\_features[test\_index]

y\_train, y\_test = y\_target[train\_index], y\_target[test\_index]

#학습 및 예측

ridge\_reg.fit(X\_train , y\_train)

pred = dt\_clf.predict(X\_test)

n\_iter += 1

# 반복 시 마다 정확도 측정

mse = np.round(mean\_squared\_error(y\_test, pred), 4)

train\_size = X\_train.shape[0]

test\_size = X\_test.shape[0]

print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mean\_squared\_error(y\_test, pred),

np.sqrt(mean\_squared\_error(y\_test, pred))))

print('Variance score : {0:.3f}'.format(r2\_score(y\_test, ridge\_pred)))

print('\n#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'

.format(n\_iter, accuracy, train\_size, test\_size))

print('#{0} 검증 세트 인덱스:{1}'.format(n\_iter,test\_index))

cv\_mse.append(mse)

# 개별 iteration별 정확도를 합하여 평균 정확도 계산

print('\n## 평균 검증 정확도:', np.mean(cv\_mse))

**# ### Stratified K-fold(전처리 후 진행해야 함)**

ridge\_reg = Ridge(random\_state=156)

# 5개의 폴드 세트로 분리하는 KFold 객체와 폴드 세트별 정확도를 담을 리스트 객체 생성.

skfold = StratifiedKFold(n\_splits=3)

n\_iter = 0

cv\_mse = []

# KFold객체의 split( ) 호출하면 폴드 별 학습용, 검증용 테스트의 로우 인덱스를 array로 반환

for train\_index, test\_index in skfold.split(X\_features, y\_target):

# kfold.split( )으로 반환된 인덱스를 이용하여 학습용, 검증용 테스트 데이터 추출

X\_train, X\_test = X\_features[train\_index], X\_features[test\_index]

y\_train, y\_test = y\_target[train\_index], y\_target[test\_index]

#학습 및 예측

ridge\_reg.fit(X\_train , y\_train)

pred = dt\_clf.predict(X\_test)

n\_iter += 1

# 반복 시 마다 정확도 측정

mse = np.round(mean\_squared\_error(y\_test, pred), 4)

train\_size = X\_train.shape[0]

test\_size = X\_test.shape[0]

print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mean\_squared\_error(y\_test, pred),

np.sqrt(mean\_squared\_error(y\_test, pred))))

print('Variance score : {0:.3f}'.format(r2\_score(y\_test, ridge\_pred)))

print('\n#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'

.format(n\_iter, accuracy, train\_size, test\_size))

print('#{0} 검증 세트 인덱스:{1}'.format(n\_iter,test\_index))

cv\_mse.append(mse)

# 개별 iteration별 정확도를 합하여 평균 정확도 계산

print('\n## 평균 검증 정확도:', np.mean(cv\_mse))

**# ## 범주 데이터 수치화**

# ### get\_dummies

# ### Label Incoding

# ### one-hot Incoding

**# ## Scaler**

# ### StandardScaler

# X\_features scaling

scaler = StandardScaler()

scaler = scaler.fit(X\_train)

X\_train\_S = pd.DataFrame(scaler.transform(X\_train),

index=X\_train.index, columns=X\_train.columns)

X\_test\_S = pd.DataFrame(scaler.transform(X\_test),

index=X\_test.index, columns=X\_test.columns)

display(X\_test\_S.describe().T)

# y\_target scaling

scaler = StandardScaler()

scaler = scaler.fit(y\_train)

y\_train\_S = pd.DataFrame(scaler.transform(y\_train),

index=y\_train.index, columns=y\_train.columns)

y\_test\_S = pd.DataFrame(scaler.transform(y\_test),

index=y\_test.index, columns=y\_test.columns)

display(y\_test\_S.describe().T)

**# ### MinMaxScaler**

# X\_features scaling

scaler = MinMaxScaler()

scaler = scaler.fit(X\_train)

X\_train\_S = pd.DataFrame(scaler.transform(X\_train),

index=X\_train.index, columns=X\_train.columns)

X\_test\_S = pd.DataFrame(scaler.transform(X\_test),

index=X\_test.index, columns=X\_test.columns)

display(X\_test\_S.describe().T)

# y\_target scaling

scaler = MinMaxScaler()

scaler = scaler.fit(y\_train)

y\_train\_S = pd.DataFrame(scaler.transform(y\_train),

index=y\_train.index, columns=y\_train.columns)

y\_test\_S = pd.DataFrame(scaler.transform(y\_test),

index=y\_test.index, columns=y\_test.columns)

display(y\_test\_S.describe().T)

**# ### Normalizer**

# X\_features scaling

scaler = Normalizer()

scaler = scaler.fit(X\_train)

X\_train\_S = pd.DataFrame(scaler.transform(X\_train),

index=X\_train.index, columns=X\_train.columns)

X\_test\_S = pd.DataFrame(scaler.transform(X\_test),

index=X\_test.index, columns=X\_test.columns)

display(X\_test\_S.describe().T)

# y\_target scaling

scaler = Normalizer()

scaler = scaler.fit(y\_train)

y\_train\_S = pd.DataFrame(scaler.transform(y\_train),

index=y\_train.index, columns=y\_train.columns)

y\_test\_S = pd.DataFrame(scaler.transform(y\_test),

index=y\_test.index, columns=y\_test.columns)

display(y\_test\_S.describe().T)

**# ### Log1p**

# X\_features scaling

X\_train\_S = pd.DataFrame(np.log1p(X\_train), index=X\_train.index, columns=X\_train.columns)

X\_test\_S = pd.DataFrame(np.log1p(X\_test), index=X\_test.index, columns=X\_test.columns)

display(X\_test\_S.describe().T)

# y\_target scaling

y\_train\_S = pd.DataFrame(np.log1p(y\_train), index=X\_train.index, columns=y\_train.columns)

y\_test\_S = pd.DataFrame(np.log1p(y\_test), index=y\_test.index, columns=y\_test.columns)

display(y\_test\_S.describe().T)

**# ## 다중공선성제거**

**# ### VIF**

**# #### 상관도 확인**

corr = X\_features.corr()

plt.figure(figsize=(14,14))

sns.heatmap(corr, annot=True, fmt='.1g')

# correlation from features

raw\_df.corr().loc[X\_colname, X\_colname].style.background\_gradient().set\_precision(2).set\_properties(\*\*{'font-size': '11pt'})

corr = X\_features.corr().loc[:, ['count']]

corr = X\_features.corr().loc[:, ['count']].style.background\_gradient().set\_precision(2).set\_properties(\*\*{'font-size': '11pt'})

for col in raw\_df.describe().columns:

target = raw\_df[col]

figure, axes = plt.subplots(2,1,figsize=(16,10))

sm.graphics.tsa.plot\_acf(target, lags=100, use\_vlines=True, ax=axes[0], title=col)

sm.graphics.tsa.plot\_pacf(target, lags=100, use\_vlines=True, ax=axes[1], title=col)

vif = pd.DataFrame()

vif['VIF\_Factor'] = [variance\_inflation\_factor(X\_train\_S.values, i)

for i in range(X\_train\_S.shape[1])]

vif['Feature'] = X\_train\_S.columns

vif.sort\_values(by='VIF\_Factor', ascending=True)

# extract effective features using variance inflation factor

vif = pd.DataFrame()

vif['VIF\_Factor'] = [variance\_inflation\_factor(X\_train\_S.values, i)

for i in range(X\_train\_S.shape[1])]

vif['Feature'] = X\_train\_S.columns

vif.sort\_values(by='VIF\_Factor', ascending=True)['Feature'][:10].values

**# #### 독립변수 일부 반영**

### Functionalize

### extract non-multicollinearity variables by VIF

def feature\_engineering\_XbyVIF(X\_train, num\_variables):

vif = pd.DataFrame()

vif['VIF\_Factor'] = [variance\_inflation\_factor(X\_train.values, i)

for i in range(X\_train.shape[1])]

vif['Feature'] = X\_train.columns

X\_colname\_vif = vif.sort\_values(by='VIF\_Factor', ascending=True)['Feature'][:num\_variables].values

return X\_colname\_vif

# X\_colname\_vif = feature\_engineer

feature\_engineering\_XbyVIF(X\_train\_S, 10)

**# ### PCA**

pca = PCA(n\_components=2)

#fit( )과 transform( ) 을 호출하여 PCA 변환 데이터 반환

pca.fit(X\_features)

X\_features\_pca = pca.transform(X\_features)

print('PCA Component별 변동성:', pca.explained\_variance\_ratio\_)

**# ### LDA**

lda = LinearDiscriminantAnalysis(n\_components=2)

# fit()호출 시 target값 입력

lda.fit(X\_features, y\_target)

X\_features\_lda = lda.transform(X\_features)

print(iris\_lda.shape)

lda\_columns=['lda\_component\_1','lda\_component\_2']

X\_lda\_df = pd.DataFrame(X\_features\_lda,columns=lda\_columns)

X\_lda\_df['target']=y\_target

**# ### SVD(미완)**

**# ### Truncated SVD(미완)**

**# ### NMF(미완)**

**# # 통합 예측 및 평가**

# ## 로그값 취했을 경우 풀기 전환

# 로그를 취했을 경우 학습 후 로그 풀기

y\_test = np.expm1(y\_test)

ridge\_pred = np.expm1(ridge\_pred)

**# 평가**

evaluate\_regr(y\_test, ridge\_pred)

**# ## 평가 함수**

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

# log 값 변환 시 언더플로우 영향으로 log() 가 아닌 log1p() 를 이용하여 RMSLE 계산

def rmsle(y, pred):

log\_y = np.log1p(y)

log\_pred = np.log1p(pred)

squared\_error = (log\_y - log\_pred) \*\* 2

rmsle = np.sqrt(np.mean(squared\_error))

return rmsle

def msle(y, pred):

log\_y = np.log1p(y)

log\_pred = np.log1p(pred)

squared\_error = (log\_y - log\_pred) \*\* 2

msle = np.mean(squared\_error)

return msle

# 사이킷런의 mean\_square\_error() 를 이용하여 RMSE 계산

def rmse(y,pred):

rmse = np.sqrt(mean\_squared\_error(y, pred))

return rmse

def msle(y, pred):

log\_y = np.log1p(y)

log\_pred = np.log1p(pred)

squared\_error = (log\_y - log\_pred) \*\* 2

msle = np.mean(squared\_error)

return msle

def mape(y, pred):

mape = np.mean(np.abs((y\_test - pred) / y\_test)) \* 100

return mape

# MAE, RMSE, RMSLE 를 모두 계산

def evaluate\_regr(y,pred):

rmsle\_val = rmsle(y,pred)

rmse\_val = rmse(y,pred)

# MAE 는 scikit learn의 mean\_absolute\_error() 로 계산

msle\_val = msle(y, pred)

mae\_val = mean\_absolute\_error(y,pred)

mse\_val = mean\_squared\_error(y, pred)

mape\_val = mape(y, pred)

print('RMSLE: {0:.3f}, RMSE: {1:.3F}'.format(rmsle\_val, rmse\_val))

print('MSLE: {0:.3f}, MSE: {1:.3F}, MAE: {2:.3F}, MAPE: {2:.4F}'.format(msle\_val, mse\_val,

mae\_val, mape\_val))

print('-------------------------------------------------------------------------------------------------\n')

# 모델과 학습/테스트 데이터 셋을 입력하면 성능 평가 수치를 반환

def get\_model\_predict(model, X\_train, X\_test, y\_train, y\_test, is\_expm1=False):

model.fit(X\_train, y\_train)

pred = model.predict(X\_test)

if is\_expm1 :

y\_test = np.expm1(y\_test)

pred = np.expm1(pred)

print('###',model.\_\_class\_\_.\_\_name\_\_,'###')

evaluate\_regr(y\_test, pred)

def get\_model\_cv\_prediction(model, X\_features, y\_target):

neg\_mse\_scores = cross\_val\_score(model, X\_features, y\_target, scoring="neg\_mean\_squared\_error",

cv = 5, n\_jobs=-1)

# MAE

mae\_scores = np.sqrt(abs(neg\_mse\_scores))

avg\_mae = np.mean(mae\_scores)

# MSE

mse\_scores = (-1 \* neg\_mse\_scores)

avg\_mse = np.mean(mse\_scores)

# MAPE

mape\_scores = np.sqrt(abs(neg\_mse\_scores))\*100

avg\_mape = np.mean(mape\_scores)

# RMSE

rmse\_scores = np.sqrt(-1 \* neg\_mse\_scores)

avg\_rmse = np.mean(rmse\_scores)

print('##### 5 교차 검증 ',model.\_\_class\_\_.\_\_name\_\_ , ' #####\n')

print(' - MAE :\n {0} '.format(mae\_scores))

print(' - 평균 MAE :\n {0:.3f} '.format(avg\_mae))

print(' - MSE :\n {0} '.format(mse\_scores))

print(' - 평균 MSE :\n {0:.3f} '.format(avg\_mse))

print(' - MAPE :\n {0} '.format(mape\_scores))

print(' - 평균 MAPE :\n {0:.3f} '.format(avg\_mape))

print(' - RMSE :\n {0} '.format(rmse\_scores))

print(' - 평균 RMSE :\n {0:.3f} '.format(avg\_rmse))

print('-------------------------------------------------------------------------------------------------\n')

**# ## 교차 검증 없이 예측 및 평가**

# model 별로 평가 수행

lr\_reg = LinearRegression()

ridge\_reg = Ridge(alpha=0.1)

lasso\_reg = Lasso(alpha=0.1)

ela\_reg = ElasticNet(alpha=0.01, l1\_ratio=0.7)

dt\_reg = DecisionTreeRegressor(max\_depth=3, min\_samples\_split=3)

rf\_reg = RandomForestRegressor(n\_estimators=500)

gbm\_reg = GradientBoostingRegressor(n\_estimators=500)

xgb\_reg = XGBRegressor(n\_estimators=500)

lgbm\_reg = LGBMRegressor(n\_estimators=500)

model\_list = [lr\_reg, ridge\_reg, lasso\_reg, ela\_reg, dt\_reg, rf\_reg, gbm\_reg, xgb\_reg, lgbm\_reg]

for model in model\_list:

get\_model\_predict(model, X\_train, X\_test, y\_train, y\_test,is\_expm1=False)

**# ## 교차 검증을 포함한 예측 및 평가**

# model 별로 평가 수행

lr\_reg = LinearRegression()

ridge\_reg = Ridge(alpha=0.01)

lasso\_reg = Lasso(alpha=0.01)

ela\_reg = ElasticNet(alpha=0.01, l1\_ratio=0.7)

dt\_reg = DecisionTreeRegressor(max\_depth=3, min\_samples\_split=3)

rf\_reg = RandomForestRegressor(n\_estimators=500)

gbm\_reg = GradientBoostingRegressor(n\_estimators=500)

xgb\_reg = XGBRegressor(n\_estimators=500)

lgbm\_reg = LGBMRegressor(n\_estimators=500)

model\_list = [lr\_reg, ridge\_reg, lasso\_reg, ela\_reg, dt\_reg, rf\_reg, gbm\_reg, xgb\_reg, lgbm\_reg]

for model in model\_list:

get\_model\_cv\_prediction(model, X\_features, y\_target)

**# # 개별 예측 및 평가**

**# ## 평가 함수**

# 모델과 학습/테스트 데이터 셋을 입력하면 성능 평가 수치를 반환

def get\_model\_no\_machin\_predict(model, X\_train, X\_test, y\_train, y\_test, is\_expm1=False):

model.fit(X\_train, y\_train)

pred = model.predict(X\_test)

if is\_expm1 :

y\_test = np.expm1(y\_test)

pred = np.expm1(pred)

print('###',model.\_\_class\_\_.\_\_name\_\_,'###')

evaluate\_regr(y\_test, pred)

# 회귀계수 추출

coeff = pd.Series(data=model.coef\_ , index=X\_features.columns)

return coeff

**# ## LinearRegression**

# Linear Regression OLS로 학습/예측/평가 수행.

lr\_reg = LinearRegression()

lr\_reg.fit(X\_train ,y\_train)

lr\_pred = lr\_reg.predict(X\_test)

mse = mean\_squared\_error(y\_test, lr\_pred)

rmse = np.sqrt(mse)

**# # 평가**

# evaluate\_regr(y\_test, lr\_pred)

print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))

print('Variance score : {0:.3f}'.format(r2\_score(y\_test, lr\_pred)))

print('절편 값:',lr\_reg.intercept\_)

print('회귀 계수값:', np.round(lr\_reg.coef\_, 1))

coeff\_df = pd.DataFrame()

params = [0.1, 0.3, 0.5]

for param in params:

lasso\_reg = Lasso(alpha=param)

neg\_mse\_scores = cross\_val\_score(lasso\_reg, X\_features, y\_target, scoring="neg\_mean\_squared\_error",

cv = 5)

# 평가

avg\_mse = np.mean(-1 \* neg\_mse\_scores)

avg\_rmse = np.mean(np.sqrt(-1 \* neg\_mse\_scores))

print('alpha {0}일 때 5 폴드 세트의 평균 MSE: {1:.3f}, 평균 RMSE: {2:.3f}' .format(param, avg\_mse, avg\_rmse))

# cross\_val\_score는 evaluation metric만 반환하므로 모델을 다시 학습하여 회귀 계수 추출

lasso\_reg.fit(X\_features , y\_target)

# alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가.

coeff = pd.Series(data=lasso\_reg.coef\_ , index=X\_features.columns )

colname='alpha:'+str(param)

coeff\_df[colname] = coeff

coeff\_df

**# ## Ridge**

X\_features

pd.Series(data=ridge\_reg.coef\_ , index=X\_features.columns)

ridge\_reg.coef\_.flatten()

coeff\_df = pd.DataFrame()

# 학습

ridge\_reg = Ridge(alpha=0.1)

ridge\_reg.fit(X\_train, y\_train)

ridge\_pred = ridge\_reg.predict(X\_test)

coeff = pd.Series(data=ridge\_reg.coef\_.flatten() , index=X\_features.columns)

coeff\_df['coefficient'] = coeff

coeff\_df = coeff\_df.sort\_values('coefficient', ascending=False)

coeff\_df

print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))

print('Variance score : {0:.3f}'.format(r2\_score(y\_test, ridge\_pred)))

print('절편 값:',ridge\_reg.intercept\_)

print('회귀 계수값:', np.round(ridge\_reg.coef\_, 1))

**# ### make\_scorer**

coeff\_df = pd.DataFrame()

params = [0.1, 0.3, 0.5]

for param in params:

ridge\_reg = Ridge(alpha=param)

neg\_mse\_scores = cross\_val\_score(ridge\_reg, X\_features, y\_target, scoring=make\_scorer(mape),

cv = 5)

avg\_rmse = np.mean(np.sqrt(-1 \* neg\_mse\_scores))

print('alpha {0}일 때 개별 MSE scores: ', np.round(-1 \* neg\_mse\_scores, 2))

print('alpha {0}일 때 세트의 평균 MSE: {1:.3f} '.format(param, avg\_mse))

print('alpha {0}일 때 개별 RMSE scores: ', np.round(-1 \* neg\_mse\_scores, 2))

print('alpha {0}일 때 세트의 평균 RMSE: {1:.3f} '.format(param, avg\_rmse))

# cross\_val\_score는 evaluation metric만 반환하므로 모델을 다시 학습하여 회귀 계수 추출

ridge\_reg.fit(X\_features , y\_target)

# alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가.

coeff = pd.Series(data=ridge\_reg.coef\_.flatten() , index=X\_features.columns )

colname='alpha:'+str(param)

coeff\_df[colname] = coeff

coeff\_df

**# ### neg\_mean\_squared\_error**

coeff\_df = pd.DataFrame()

params = [0.1, 0.3, 0.5]

for param in params:

ridge\_reg = Ridge(alpha=param)

neg\_mse\_scores = cross\_val\_score(ridge\_reg, X\_features, y\_target, scoring="neg\_mean\_squared\_error",

cv = 5)

avg\_rmse = np.mean(np.sqrt(-1 \* neg\_mse\_scores))

print('alpha {0}일 때 개별 MSE scores: ', np.round(-1 \* neg\_mse\_scores, 2))

print('alpha {0}일 때 세트의 평균 MSE: {1:.3f} '.format(param, avg\_mse))

print('alpha {0}일 때 개별 RMSE scores: ', np.round(-1 \* neg\_mse\_scores, 2))

print('alpha {0}일 때 세트의 평균 RMSE: {1:.3f} '.format(param, avg\_rmse))

# cross\_val\_score는 evaluation metric만 반환하므로 모델을 다시 학습하여 회귀 계수 추출

ridge\_reg.fit(X\_features , y\_target)

# alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가.

coeff = pd.Series(data=ridge\_reg.coef\_ , index=X\_features.columns )

colname='alpha:'+str(param)

coeff\_df[colname] = coeff

coeff\_df

# cross\_val\_score( )로 5 Fold 셋으로 MSE 를 구한 뒤 이를 기반으로 다시 RMSE 구함.

neg\_mse\_scores = cross\_val\_score(lr, X\_features, y\_target, scoring="neg\_mean\_squared\_error", cv = 5)

rmse\_scores = np.sqrt(-1 \* neg\_mse\_scores)

avg\_rmse = np.mean(rmse\_scores)

# cross\_val\_score(scoring="neg\_mean\_squared\_error")로 반환된 값은 모두 음수

print(' 5 folds 의 개별 Negative MSE scores: ', np.round(neg\_mse\_scores, 2))

print(' 5 folds 의 개별 RMSE scores : ', np.round(rmse\_scores, 2))

print(' 5 folds 의 평균 RMSE : {0:.3f} '.format(avg\_rmse))

**# ## Lasso(수정)**

coeff\_df = pd.DataFrame()

params = [0.1, 0.3, 0.5]

for param in params:

lasso\_reg = Lasso(alpha=param)

neg\_mse\_scores = cross\_val\_score(lasso\_reg, X\_features, y\_target, scoring="neg\_mean\_squared\_error",

cv = 5)

# 평가

avg\_mse = np.mean(-1 \* neg\_mse\_scores)

avg\_rmse = np.mean(np.sqrt(-1 \* neg\_mse\_scores))

print('alpha {0}일 때 5 폴드 세트의 평균 MSE: {1:.3f}, 평균 RMSE: {2:.3f}' .format(param, avg\_mse, avg\_rmse))

# cross\_val\_score는 evaluation metric만 반환하므로 모델을 다시 학습하여 회귀 계수 추출

lasso\_reg.fit(X\_features , y\_target)

# alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가.

coeff = pd.Series(data=lasso\_reg.coef\_ , index=X\_features.columns )

colname='alpha:'+str(param)

coeff\_df[colname] = coeff

coeff\_df

**# ## ElasticNet**

coeff\_df = pd.DataFrame()

params = [0.1, 0.3, 0.5]

for param in params:

en\_reg = ElasticNet(alpha=param, l1\_ratio=0.7)

neg\_mse\_scores = cross\_val\_score(en\_reg, X\_features, y\_target, scoring="neg\_mean\_squared\_error",

cv = 5)

avg\_rmse = np.mean(np.sqrt(-1 \* neg\_mse\_scores))

print('alpha {0}일 때 5 폴드 세트의 평균 RMSE: {1:.3f} '.format(param, avg\_rmse))

# cross\_val\_score는 evaluation metric만 반환하므로 모델을 다시 학습하여 회귀 계수 추출

en\_reg.fit(X\_features , y\_target)

# alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가.

coeff = pd.Series(data=en\_reg.coef\_ , index=X\_features.columns )

colname='alpha:'+str(param)

coeff\_df[colname] = coeff

coeff\_df

**# ## DecisionTreeRegressor**

**# ### GridSearchCV**

max\_depth = [3, None]

max\_features = [0.1, 0.2, 0.5, 0.8]

min\_samples\_split = [2, 10]

min\_samples\_leaf = [2, 10]

hyperparams = {'max\_depth': max\_depth, 'max\_features': max\_features,

'min\_samples\_split': min\_samples\_split, 'min\_samples\_leaf': min\_samples\_leaf}

gd=GridSearchCV(estimator = DecisionTreeRegressor(), param\_grid = hyperparams,

verbose=True, cv=5, scoring = "neg\_mean\_squared\_error", n\_jobs=-1)

gd.fit(X\_train, y\_train)

print(gd.best\_score\_)

print(gd.best\_params\_)

coeff\_df = pd.DataFrame()

params = [3,2]

for param in params:

ridge\_reg = DecisionTreeRegressor(max\_depth = param)

neg\_mse\_scores = cross\_val\_score(ridge\_reg, X\_features, y\_target, scoring="neg\_mean\_squared\_error",

cv = 5)

avg\_rmse = np.mean(np.sqrt(-1 \* neg\_mse\_scores))

print('alpha {0}일 때 세트의 평균 MSE: {1:.3f} '.format(param, avg\_rmse))

print('개별 MSE scores:',np.round(-1 \* neg\_mse\_scores, 2))

print('alpha {0}일 때 개별 RMSE scores: '.format(param, np.round(-1 \* neg\_mse\_scores, 2))

print('alpha {0}일 때 세트의 평균 RMSE: {1}'.format(param, avg\_rmse))

# cross\_val\_score는 evaluation metric만 반환하므로 모델을 다시 학습하여 회귀 계수 추출

ridge\_reg.fit(X\_features , y\_target)

# alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가.

coeff = pd.Series(data=ridge\_reg.coef\_ , index=X\_features.columns )

colname='alpha:'+str(param)

coeff\_df[colname] = coeff

coeff\_df

**# ## RandomForestRegressor**

**# ## GradientBoostingRegressor**

**# ## XGBRegressor**

**# ## LGBMRegressor**

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import GradientBoostingRegressor

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

dt\_reg = DecisionTreeRegressor(random\_state=0, max\_depth=4)

rf\_reg = RandomForestRegressor(random\_state=0, n\_estimators=1000)

gb\_reg = GradientBoostingRegressor(random\_state=0, n\_estimators=1000)

xgb\_reg = XGBRegressor(n\_estimators=1000)

lgbm\_reg = LGBMRegressor(n\_estimators=1000)

# 트리 기반의 회귀 모델을 반복하면서 평가 수행

models = [dt\_reg, rf\_reg, gb\_reg, xgb\_reg, lgbm\_reg]

for model in models:

get\_model\_predict(model,X\_train, X\_test, y\_train, y\_test,is\_expm1=True)

get\_model\_cv\_prediction(model, X\_data, y\_target)