AIchemist Train Session

CHAP 05 회귀(2)

실습 목차

01.
시작하기 전에

📕 Guide-Line

- Github 자료 다운 받기 & Kaggle 사이트 참고하기
- 조원과 워크북 질문 답하기 & 코드 빈칸 채우기
- 교재 참고하지 않고 채워봅시다!! 🔥
- 발표 目

데이터셋 소개

House Prices - Advanced Regression Techniques

Predict sales prices and practice feature engineering, RFs, and gradient boosting

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

주택 가격: 고급 회귀 기법
-주택의 여러 feature를 이용해 주택 가격을 예측

Feature
-SalePrice : 주택 가격 (dollar) -> 타겟값
-MSSubClass : 빌딩 클래스
-MSZoning : 일반 구역 분류
-LotFrontage : 부동산의 경계산에 연결된 거리의 선형 피트
…

Kaggle 사이트에서 피처 설명 참고

https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data?select=train.csv

02.

캐글 주택 가격 예측 워크북

모듈 임포트, 데이터세트 로드 및 확인

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

house_df_org = pd.[          ]('house_price.csv')
house_df = house_df_org.copy()
house_df.[      ]
```

Q1. 코드 빈칸의 답은?

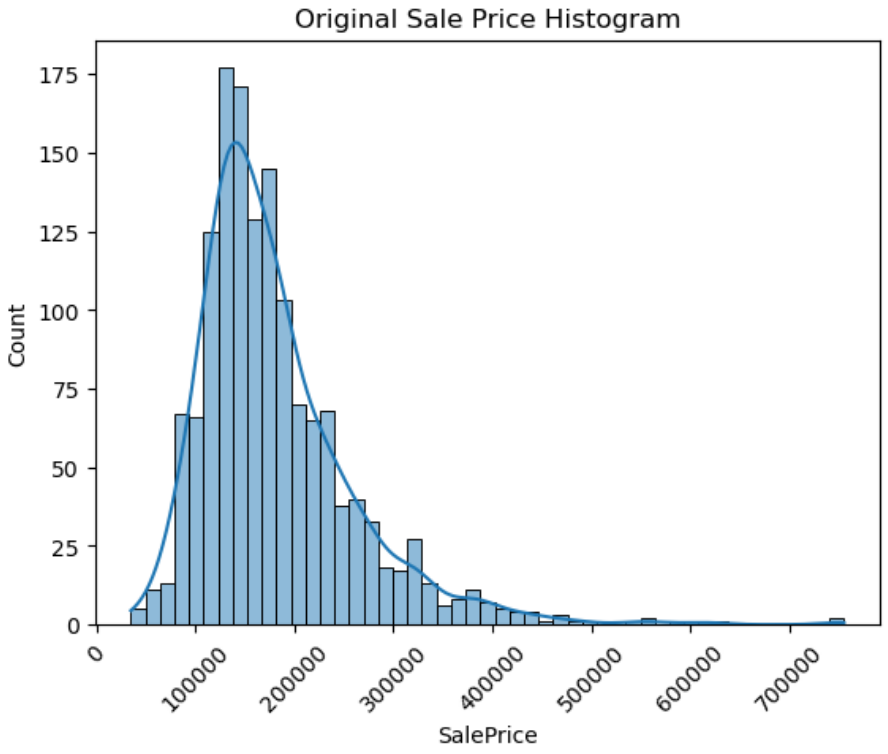
여러 데이터 전처리를 적용하므로
원본 DataFrame 복사해서 사용

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

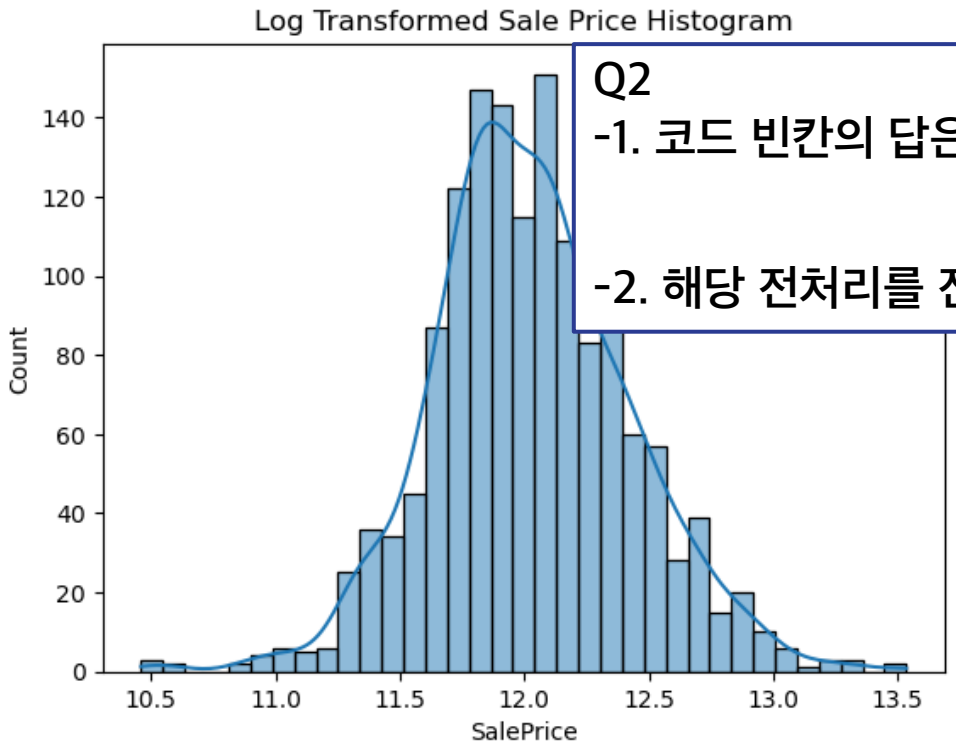
5 rows × 81 columns

전처리 - 타겟값

```
plt.title('Original Sale Price Histogram')
plt.xticks(rotation=45)
sns.histplot(house_df['SalePrice'], kde=True)
plt.show()
```




```
plt.title('Log Transformed Sale Price Histogram')
log_SalePrice = np.      (house_df['SalePrice'])
sns.histplot(log_SalePrice, kde=True)
plt.show()
```



Q2

-1. 코드 빈칸의 답은? (hint: 이름에 1 들어감)

-2. 해당 전처리를 진행한 이유가 무엇일까?

전처리 - Null

```
# 로그 변환 및 Null 피처 전처리 수행

# SalePrice 로그 변환
original_SalePrice = house_df['SalePrice'] # 복사해둠
house_df['SalePrice'] = np.[redacted](house_df['SalePrice'])

# Null이 너무 많은 칼럼과 불필요한 칼럼 삭제
house_df.[redacted][[redacted]], axis=1, inplace=True)

# 드롭하지 않는 숫자형 Null 칼럼은 평균값으로 대체
house_df.[redacted](house_df.[redacted](), inplace=True)

# Null 값이 있는 피처명과 타입을 추출
null_column_count = house_df.isnull().sum()[house_df.isnull().sum() > 0]
print('## Null 피처의 Type :\n', house_df.dtypes[null_column_count.index])
```

Q3. 코드 빈칸의 답은?
(hint: 삭제 칼럼 총 6개, 다음 페이지 sweetviz 분석 참고)

```
## Null 피처의 Type :
 MasVnrType      object
BsmtQual         object
BsmtCond         object
BsmtExposure     object
BsmtFinType1     object
BsmtFinType2     object
Electrical       object
GarageType       object
GarageFinish     object
GarageQual       object
GarageCond       object
dtype: object
```

sweetviz 분석 살펴보기

sweetviz_house_price_report.html 다운받고 살펴보기

https://github.com/Ewha-AIchemist-3/Session/tree/main/Practice/Week06/docs

전처리 - object 타입

```
# 문자형 피처 원-핫 인코딩
print('수행 전 데이터 Shape:', house_df.shape)
house_df_ohe = pd.[          ](house_df)
print('수행 후 데이터 Shape:', house_df_ohe.shape)

null_column_count = house_df_ohe.isnull().sum()[house_df_ohe.isnull().sum() > 0]
print('## Null 피처의 Type :\n', house_df_ohe.dtypes[null_column_count.index])
```

```
수행 전 데이터 Shape: (1460, 76)
수행 후 데이터 Shape: (1460, 275)
## Null 피처의 Type :
 Series([], dtype: object)
```

Q4
-1. 코드 빈칸의 답은?

-2. 수행 후 데이터 Shape이 달라진 이유?

회귀 모델의 예측 평가 함수 생성

```
# 모델의 로그 변환된 RMSE(RMSLE)로 평가하는 함수
def get_rmse(model):
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test, pred)
    rmse = np.sqrt(mse)
    print(model.__class__.__name__, ' 로그 변환된 RMSE:', np.round(rmse, 3))
    return rmse

# 여러 모델의 RMSE 값 반환
def get_rmses(models):
    rmses = [ ]
    for model in models:
        rmse = get_rmse(model)
        rmses.append(rmse)
    return rmses
```

선형 회귀 모델 - 학습 / 예측 / 평가

```
# 선형 회귀 모델 학습/예측/평가
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

y_target = house_df_ohe['SalePrice']
X_features = house_df_ohe.drop(['SalePrice'], axis=1, inplace=False)
X_train, X_test, y_train, y_test =                (X_features, y_target, test_size=0.2, random_state=156)

lr_reg = LinearRegression()
lr_reg.
ridge_reg = Ridge()
ridge_reg.
lasso_reg = Lasso()
lasso_reg.

models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models)
```

같은 코드

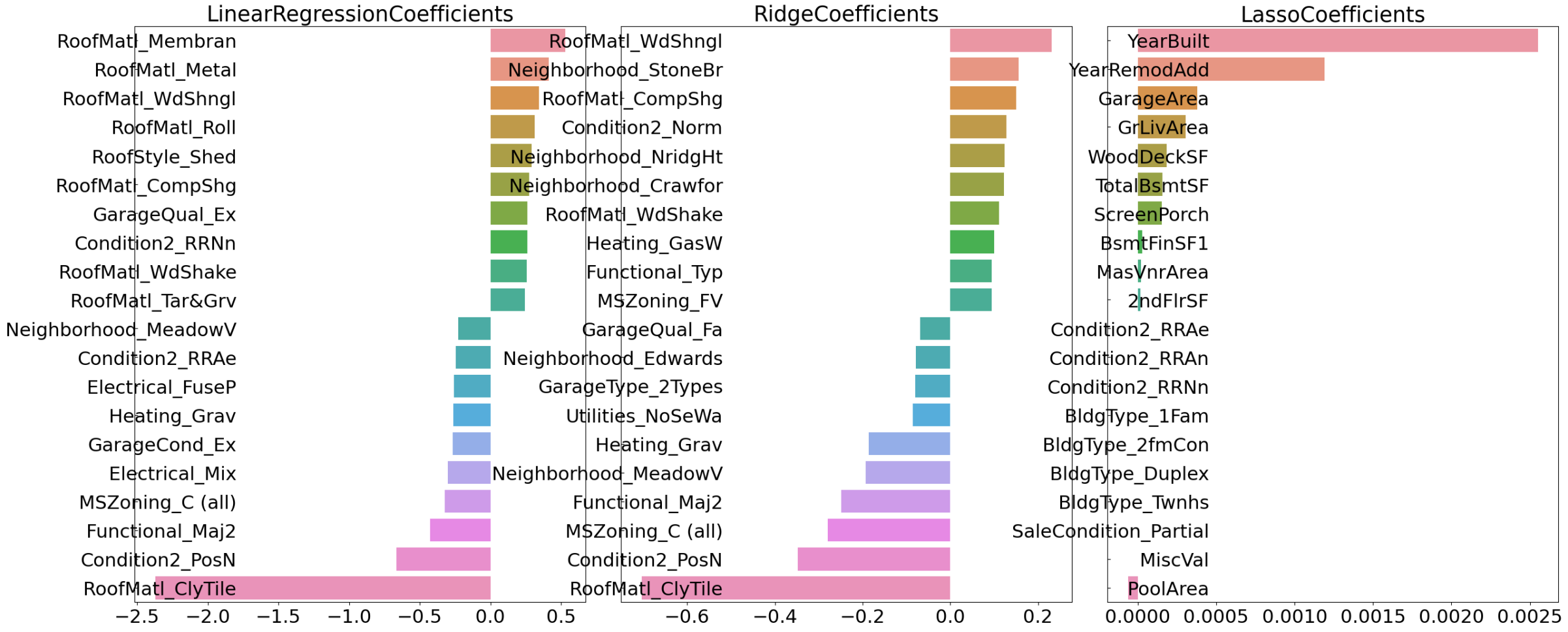
```
LinearRegression  로그 변환된 RMSE: 0.132
Ridge  로그 변환된 RMSE: 0.128
Lasso  로그 변환된 RMSE: 0.176

[0.13189576579154003, 0.12750846334053043, 0.17628250556471395]
```

Q5
-1. 코드 빈칸의 답은?

-2. 출력된 결과를 보고, 모델의 성능을 비교해보자.

선형 회귀 모델 - 모델별 회귀 계수 시각화

Q6. 다음은 모델별 상위, 하위 각 10개의 회귀 계수 시각화 결과이다. 해당 표를 보고 각 규제가 어떤 역할을 하는지 생각해보자.



선형 회귀 모델 - 교차 검증

```
# 학습데이터 데이터 분할에 문제가 있는지 전체 데이터 세트를 교차 검증 폴드 세트로 검사
from sklearn.model_selection import cross_val_score

def get_avg_rmse_cv(models):

    for model in models:
        # 분할하지 않고 전체 데이터로 cross_val_score() 수행. 모델별 CV RMSE값과 평균 RMSE 출력
        rmse_list = np.sqrt(-cross_val_score(model, X_features, y_target, scoring="neg_mean_squared_error", cv=5))
        rmse_avg = np.mean(rmse_list)
        print('\n{0} CV RMSE 값 리스트: {1}'.format(model.__class__.__name__, np.round(rmse_list, 3)))
        print('\n{0} CV 평균 RMSE 값: {1}'.format(model.__class__.__name__, np.round(rmse_avg, 3)))

# 앞 예제에서 학습한 ridge_reg, lasso_reg 모델의 CV RMSE값 출력
models = [ridge_reg, lasso_reg]
get_avg_rmse_cv(models)
```

```
Ridge CV RMSE 값 리스트: [0.117 0.154 0.142 0.117 0.189]

Ridge CV 평균 RMSE 값: 0.144

Lasso CV RMSE 값 리스트: [0.161 0.204 0.177 0.181 0.265]

Lasso CV 평균 RMSE 값: 0.198
```

Q7 해당 코드에서는 라쏘 모델의 회귀 계수가 다른 모델에 비해 너무 작아, 학습 데이터 세트의 분할에 문제가 있는지 확인하기 위해 교차 검증을 사용했다.

-1. 어떻게 문제를 확인할 수 있는지 추론해보자.
(hint: 교차 검증의 방식, 목적)

선형 회귀 모델 - 하이퍼 파라미터 튜닝

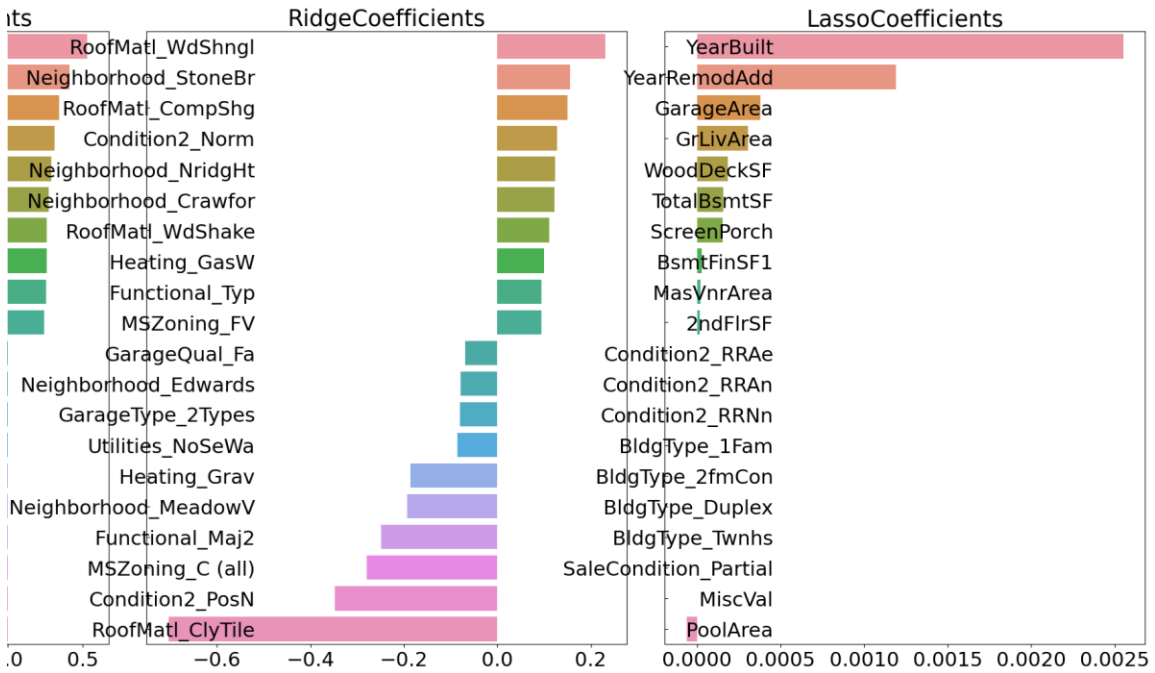
```
# 하이퍼 파라미터 튜닝 진행
from sklearn.model_selection import GridSearchCV

def print_best_params(model, params):
    grid_model = GridSearchCV(model, param_grid=params, scoring='neg_mean_squared_error', cv=5)
    grid_model.fit(X_train, y_train)
    rmse = np.sqrt(-1*grid_model.best_score_)
    print('{0} 5 CV 시 최적 평균 RMSE 값:{1}, 최적 alpha:{2}'.format(model.__class__.__name__, np.round(rmse, 4), grid_model.best_params_))

ridge_params = {'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20]}
lasso_params = {'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5, 1, 5, 10]}
print_best_params(ridge_reg, ridge_params)
print_best_params(lasso_reg, lasso_params)
```

```
Ridge 5 CV 시 최적 평균 RMSE 값:0.1498, 최적 alpha:{'alpha': 10}
Lasso 5 CV 시 최적 평균 RMSE 값:0.1521, 최적 alpha:{'alpha': 0.001}
```

선형 회귀 모델 - 튜닝 후 회귀 계수 시각화

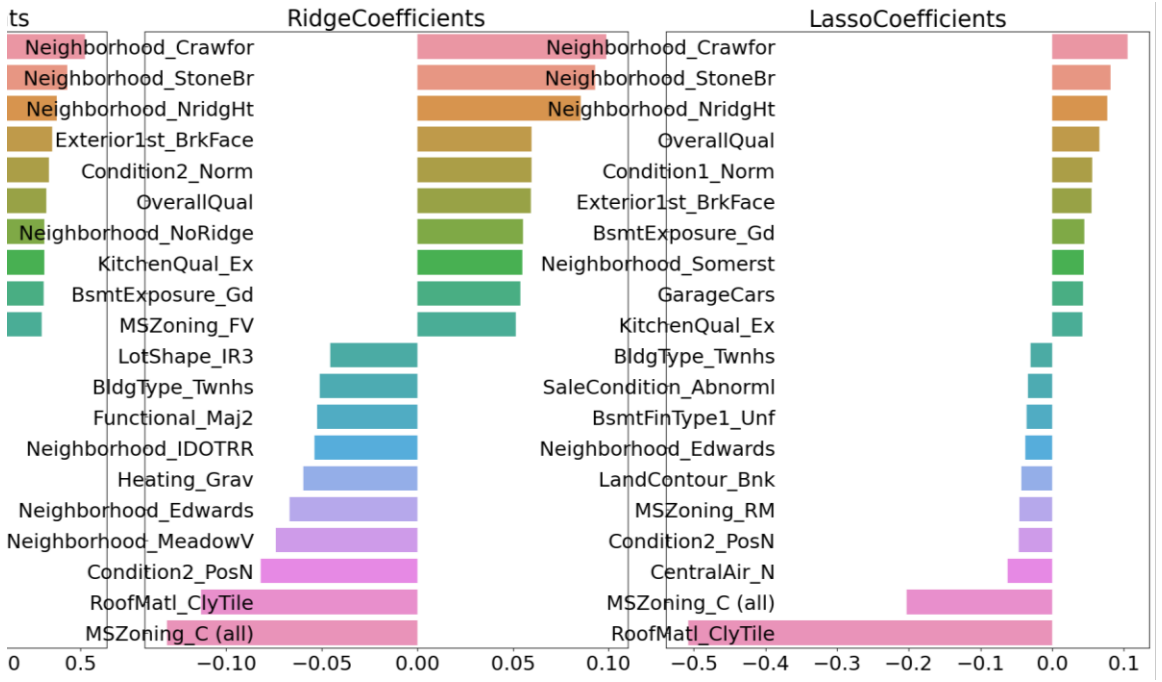
Q8. alpha 값 조정 전후의 회귀 계수 시각화 결과를 비교해보자.

```
Ridge  로그 변환된 RMSE: 0.128
Lasso  로그 변환된 RMSE: 0.176
```



전

```
Ridge  로그 변환된 RMSE: 0.124
Lasso  로그 변환된 RMSE: 0.12
```



후

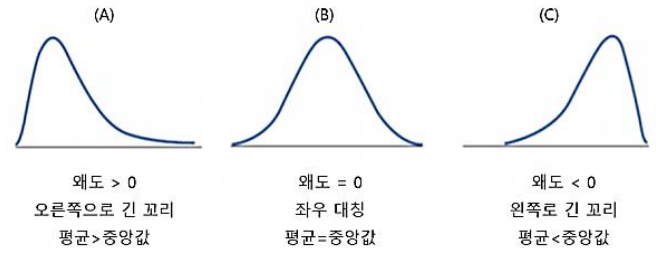
데이터세트 추가 전처리 - (1) 피처 데이터 세트 분포

```python
# 숫자형 피처의 왜곡 확인 (사이파이 stats 모듈의 skew() 이용. 원-핫 인코딩 적용 X DF 사용)
from scipy.stats import skew

# object가 아닌 숫자형 칼럼 index 객체 추출
features_index = house_df.dtypes[house_df.dtypes != 'object'].index
# house_df에 칼럼 index를 []로 입력하면 해당하는 칼럼 데이터 세트 반환. apply lambda로 skew() 호출
skew_features = house_df[features_index].apply(lambda x : skew(x))
# skew(왜곡) 정도가 1 이상인 칼럼만 추출
skew_features_top = skew_features[skew_features > 1]
print(skew_features_top.sort_values(ascending = False))
```

```
MiscVal         24.451640
PoolArea        14.813135
LotArea         12.195142
3SsnPorch       10.293752
LowQualFinSF     9.002080
KitchenAbvGr     4.483784
BsmtFinSF2       4.250888
ScreenPorch      4.117977
BsmtHalfBath     4.099186
EnclosedPorch    3.086696
MasVnrArea       2.673661
LotFrontage      2.382499
OpenPorchSF      2.361912
BsmtFinSF1       1.683771
WoodDeckSF       1.539792
TotalBsmtSF      1.522688
MSSubClass       1.406210
1stFlrSF         1.375342
GrLivArea        1.365156
dtype: float64
```

scipy.stats 의 skew 함수 => 데이터의 왜도(비대칭)를 계산해주는 함수!

- 0 에 가까운 값: 데이터가 좌우대칭에 가까운 경우
- 양수 값: 그래프의 오른쪽 꼬리가 긴 경우
- 음수 값: 그래프의 왼쪽 꼬리가 긴 경우



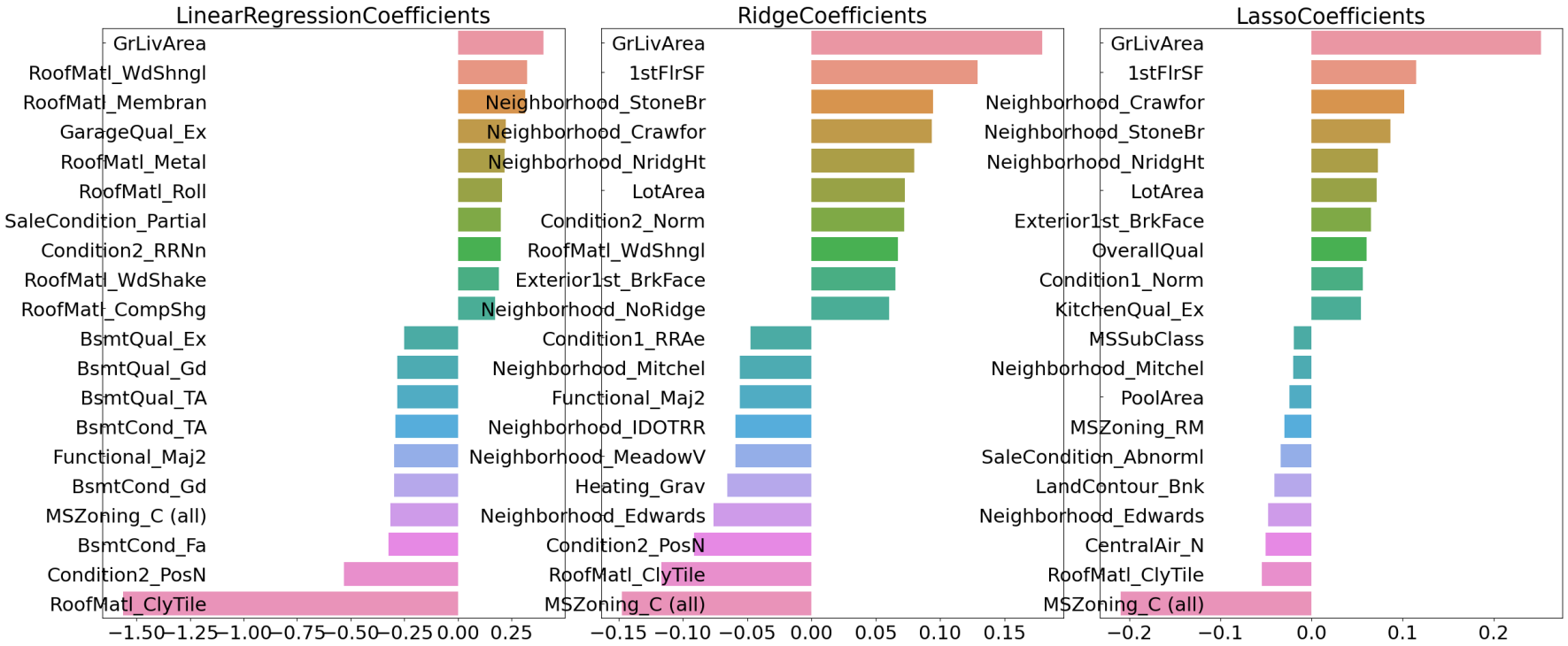
- 보통 1 이상의 값이 왜곡 정도가 높다고 판단 (상황별 편차 존재)
- 로그 변환은 큰 값을 더 많이 줄이기 때문에 그래프의 오른쪽 꼬리가 긴(= 왜도가 양수인) 경우에 사용

데이터세트 추가 전처리 - (1) 피처 데이터 세트 분포

- 해당 피처들 로그 변환 후 다시 GridSearchCV를 이용한 최적 하이퍼 파라미터 도출

```
Ridge 5 CV 시 최적 평균 RMSE 값:0.1321, 최적 alpha:{'alpha': 8}
Lasso 5 CV 시 최적 평균 RMSE 값:0.1308, 최적 alpha:{'alpha': 0.001}
```

- 최적 하이퍼 파라미터 적용한 모델별 회귀 계수 시각화 결과

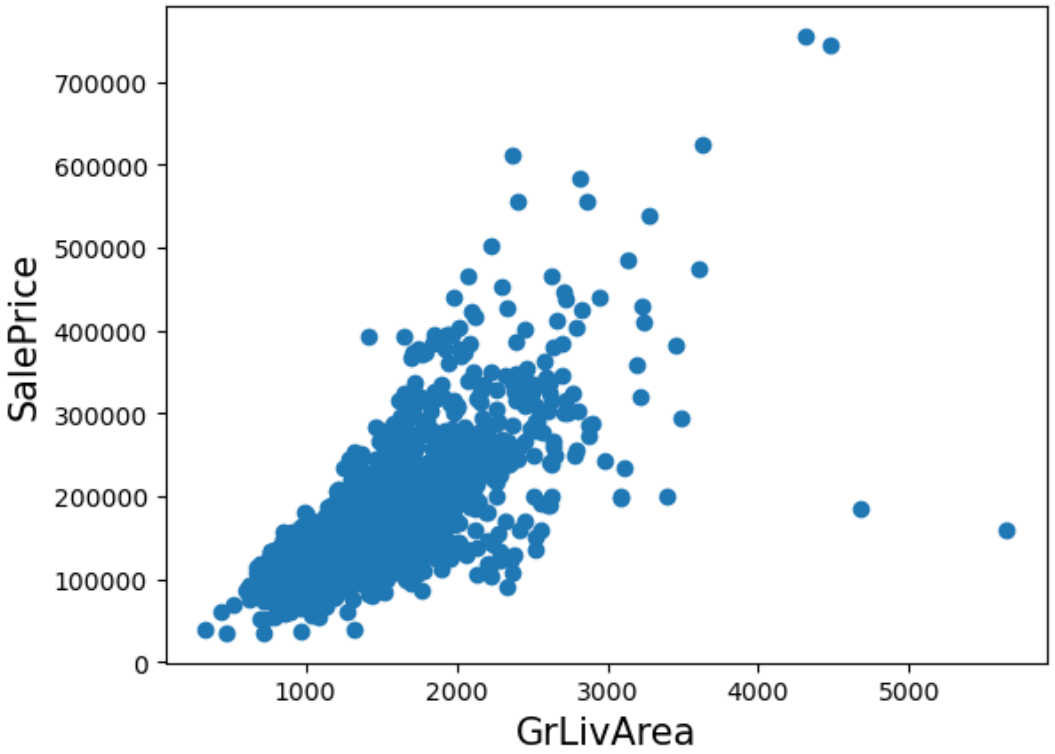


데이터세트 추가 전처리 - (2) 이상치 데이터

```
plt.scatter(x=house_df_org['GrLivArea'], y=house_df_org['SalePrice'])
plt.ylabel('SalePrice', fontsize=15)
plt.xlabel('GrLivArea', fontsize=15)
plt.show()
```

```
# GrLivAre와 SalePrice 모두 로그 변환됐으므로 이를 반영한 조건 생성
cond1 = house_df_ohe['GrLivArea'] > np.log1p(4000)
cond2 = house_df_ohe['SalePrice'] < np.log1p(500000)
outlier_index = house_df_ohe[cond1 & cond2].index

print('이상치 레코드 index :', outlier_index.values )
print('이상치 삭제 전 house_df_ohe shape:', house_df_ohe.shape)

# DataFrame의 인덱스를 이용해 이상치 레코드 삭제
house_df_ohe.drop(outlier_index, axis=0, inplace=True)
print('이상치 삭제 gn house_df_ohe shape:', house_df_ohe.shape)
```

```
이상치 레코드 index : [ 523 1298]
이상치 삭제 전 house_df_ohe shape: (1460, 271)
이상치 삭제 gn house_df_ohe shape: (1458, 271)
```



Q9

-1. 코드를 보고, 어떤 데이터를 이상치로 설정했는지 그림에서 찾아보자.

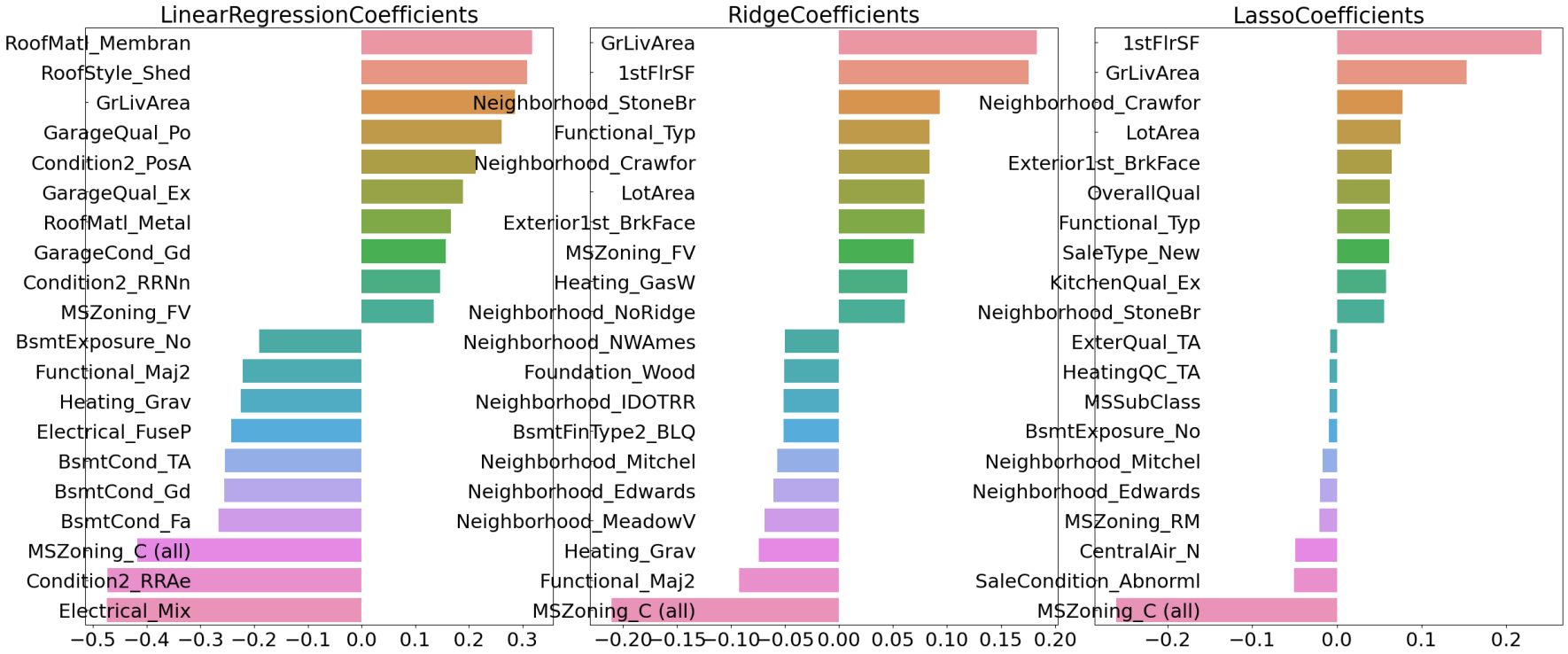
-2. 'GrLivArea'를 보고 이상치 데이터를 판별한 이유가 무엇일까?
(hint: 앞 페이지의 회귀 계수 시각화 표)

데이터세트 추가 전처리 - (2) 이상치 데이터

- 해당 이상치 데이터 제거 후 다시 GridSearchCV를 이용한 최적 하이퍼 파라미터 도출

```
Ridge 5 CV 시 최적 평균 RMSE 값:0.1158, 최적 alpha:{'alpha': 5}
Lasso 5 CV 시 최적 평균 RMSE 값:0.1153, 최적 alpha:{'alpha': 0.001}
```

- 최적 하이퍼 파라미터 적용한 모델별 회귀 계수 시각화 결과



선형 회귀 모델 - 앙상블

```
# 예측 결과 혼합
def get_rmse_pred(preds):
    for key in preds.keys():
        pred_value = preds[key]
        mse = mean_squared_error(y_test , pred_value)
        rmse = np.sqrt(mse)
        print('{0} 모델의 RMSE: {1}'.format(key, rmse))

# Ridge, Lasso 혼합
# 개별 모델의 학습
ridge_reg = Ridge(alpha=5)
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso(alpha=0.001)
lasso_reg.fit(X_train, y_train)
```

```
# 개별 모델 예측
ridge_pred = ridge_reg.predict(X_test)
lasso_pred = lasso_reg.predict(X_test)

# 개별 모델 예측값 혼합으로 최종 예측값 도출
pred = 0.4 * ridge_pred + 0.6 * lasso_pred
preds = {'최종 혼합': pred,
         'Ridge': ridge_pred,
         'Lasso': lasso_pred}

#최종 혼합 모델, 개별모델의 RMSE 값 출력
get_rmse_pred(preds)
```

```
최종 혼합 모델의 RMSE: 0.10007930884470503
Ridge 모델의 RMSE: 0.10345177546603253
Lasso 모델의 RMSE: 0.10024170460890021
```

회귀 트리 모델 - 앙상블

```python
# 회귀 트리 모델
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

# 객체 생성(XGBRegressor, LGBMRegressor)
xgb_reg = XGBRegressor(n_estimators=1000, learning_rate=0.05, colsample_bytree=0.5, subsample=0.8)
lgbm_reg = LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4,
                         subsample=0.6, colsample_bytree=0.5, reg_lambda=10, n_jobs=-1)

# 각 모델 학습/예측
xgb_reg.fit(X_train, y_train)
xgb_pred = xgb_reg.predict(X_test)
lgbm_reg.fit(X_train, y_train)
lgbm_pred = lgbm_reg.predict(X_test)

# 단순 앙상블
pred =
preds = {'최종 혼합': pred,
         'XGBM': xgb_pred,
         'LGBM': lgbm_pred}

get_rmse_pred(preds)
```

Q10. 코드 빈칸의 답은?
(hint: 앞 페이지 코드 참고, 비율: 0.5, 0.5)

```
최종 혼합 모델의 RMSE: 0.10186397087165838
XGBM 모델의 RMSE: 0.10738295638346222
LGBM 모델의 RMSE: 0.10296653528637886
```

스태킹 앙상블 모델

```
# 스태킹 앙상블
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error

# 개별 기반 모델에서 최종 메타 모델이 사용할 학습 및 테스트용 데이터를 생성하기 위한 함수.
def get_stacking_base_datasets(model, X_train_n, y_train_n, X_test_n, n_folds ):
    # 지정된 n_folds값으로 KFold 생성.
    kf = KFold(n_splits=n_folds, shuffle=False)
    # 추후에 메타 모델이 사용할 학습 데이터 반환을 위한 넘파이 배열 초기화
    train_fold_pred = np.zeros((X_train_n.shape[0] ,1 ))
    test_pred = np.zeros((X_test_n.shape[0],n_folds))
    print(model.__class__.__name__ , ' model 시작 ')

    for folder_counter , (train_index, valid_index) in enumerate(kf.split(X_train_n)):
        # 입력된 학습 데이터에서 기반 모델이 학습/예측할 폴드 데이터 셋 추출
        print('\t 폴드 세트: ',folder_counter,' 시작 ')
        X_tr = X_train_n[train_index]
        y_tr = y_train_n[train_index]
        X_te = X_train_n[valid_index]

        # 폴드 세트 내부에서 다시 만들어진 학습 데이터로 기반 모델의 학습 수행.
        model.fit(X_tr, y_tr)
        # 폴드 세트 내부에서 다시 만들어진 검증 데이터로 기반 모델 예측 후 데이터 저장.
        train_fold_pred[valid_index, :] = model.predict(X_te).reshape(-1,1)
        # 입력된 원본 테스트 데이터를 폴드 세트내 학습된 기반 모델에서 예측 후 데이터 저장.
        test_pred[:, folder_counter] = model.predict(X_test_n)

    # 폴드 세트 내에서 원본 테스트 데이터를 예측한 데이터를 평균하여 테스트 데이터로 생성
    test_pred_mean = np.mean(test_pred, axis=1).reshape(-1,1)

    # train_fold_pred는 최종 메타 모델이 사용하는 학습 데이터, test_pred_mean은 테스트 데이터
    return train_fold_pred, test_pred_mean
```

```
# 메타 모델이 사용할 학습 피처/ 테스트 피처 데이터 세트 추출
# get_stacking_base_datasets( )은 넘파이 ndarray를 인자로 사용하므로 DataFrame을 넘파이로 변환.
X_train_n = X_train.values
X_test_n = X_test.values
y_train_n = y_train.values

# 각 개별 기반(Base)모델이 생성한 학습용/테스트용 데이터 반환.
ridge_train, ridge_test = get_stacking_base_datasets(ridge_reg, X_train_n, y_train_n, X_test_n, 5)
lasso_train, lasso_test = get_stacking_base_datasets(lasso_reg, X_train_n, y_train_n, X_test_n, 5)
xgb_train, xgb_test = get_stacking_base_datasets(xgb_reg, X_train_n, y_train_n, X_test_n, 5)
lgbm_train, lgbm_test = get_stacking_base_datasets(lgbm_reg, X_train_n, y_train_n, X_test_n, 5)
```

```
# 개별 모델이 반환한 학습 및 테스트용 데이터 세트를 스태킹 형태로 결합.
Stack_final_X_train = np.concatenate((ridge_train, lasso_train,
                                      xgb_train, lgbm_train), axis=1)
Stack_final_X_test = np.concatenate((ridge_test, lasso_test,
                                     xgb_test, lgbm_test), axis=1)

# 최종 메타 모델은 라쏘 모델을 적용.
meta_model_lasso = Lasso(alpha=0.0005)

#기반 모델의 예측값을 기반으로 새롭게 만들어진 학습 및 테스트용 데이터로 예측하고 RMSE 측정.
meta_model_lasso.fit(Stack_final_X_train, y_train)
final = meta_model_lasso.predict(Stack_final_X_test)
mse = mean_squared_error(y_test , final)
rmse = np.sqrt(mse)
print('스태킹 회귀 모델의 최종 RMSE 값은:', rmse)
```

```
스태킹 회귀 모델의 최종 RMSE 값은: 0.09784801717114296
```

스태킹 앙상블 모델

Q11. 앞 페이지의 코드를 참고하여 스태킹 앙상블의 단계를 올바른 순서로 나열해보자.

① 개별 모델의 K-Fold 교차 검증 설정 및 학습: 각 개별 모델에 대해 K-Fold 교차 검증을 설정하고, 교차 검증을 통해 학습을 수행한다.

② 메타 모델 학습 및 최종 예측: 결합된 학습 데이터를 사용해 메타 모델을 학습하고, 테스트 데이터로 최종 예측을 수행하여 평가한다.

③ 스태킹 데이터 결합: 개별 모델의 예측값을 결합하여 메타 모델의 학습용 및 테스트용 데이터를 구성한다.

④ 테스트 세트 예측값 평균 계산: 모든 개별 모델이 생성한 테스트 세트 예측값을 폴드별 평균하여 최종 테스트 데이터를 생성한다.

⑤ 검증 및 테스트 세트 예측값 생성: 각 개별 모델이 교차 검증 내 검증 세트와 테스트 세트에 대해 예측한다.

마무리

Q12. 이번 실습의 전체적인 흐름을 요약해보자. (간단하게!)

1) 이번 실습에서 진행한 전처리 과정은 무엇이었는가?
ex) 데이터세트에 ~한 문제가 있을 때 ~전처리를 했다.

2) 함수 기반 회귀는 어떻게 성능을 최적화했는가?

3) 회귀의 평가는 어떤 지표를 사용했는가?

수고하셨습니다