House Prices –

Advanced Regression Techniques

회귀분석 프로젝트 보고서

11 월 25 일

조아해 팀

김지헌 백지은 정혜인 안정은

kfq 한국품질재단
Korean Foundation for Quality

목차

1. 프로젝트 개요

1.1 프로젝트 목적 및 선정이유

1. 프로젝트 목적

- Kaggle Competition 상위 10% 목표
- 데이터를 분석하여 집 값을 예측할 수 있는 모델 생성

1.1.2 프로젝트 선정 이유

- Kaggle Competition 에 참가하여 교육을 통해 배운 데이터에 대한 처리과정과 머신 러닝에 대한 실전적 사용을 경험해보고 일정 수준 이상의 성적을 내어 성과를 얻는 것이 이번 프로젝트의 선정 이유
- Kaggle Competition 중 House Prices - Advanced Regression Techniques 에서 제공받은 미국의 아이오와주와 에임스주에 있는 주거용 주택의 모든 측면을 설명하는 79 개의 설명변수가 있는 데이터를 통해 집 값을 예측할 수 있는 모델을 산출
- Kaggle 에서 개최하는 대회 중, 프로젝트 기간과 겹치는 대회로 선정, 동일한 출발선에서 시작해 모델링을 하면서 객관적인 평가를 받을 수 있는 좋은 기회라고 생각해 선정하게 되었음

1.2 프로젝트 기대효과

1.2.1 Kaggle Competition 참가

- Kaggle Competition 에 참가하여 객관적인 실력을 검증받을 수 있을 것임
- 좋은 성적을 얻기 위해 팀원들과 협력하고 다른 이들과 경쟁하며, 성공적인 팀프로젝트를 위한 과정을 경험해보는 효과를 기대할 수 있을 것으로 예상됨

1.3 개발환경

개발환경	- 운영체제 → Windows 10 - IDE → Google Colab, Jupyter Notebook - 데이터 분석 → Python : 3.9.7 - 데이터 분석, 전 처리 → Numpy : 1.20.3 / Pandas : 1.3.4 - 시각화 → Matplotlib : 3.4.3 / Seaborn : 0.11.2 - 머신 러닝 → Scikit Learn : 0.24.2 , 파이프라인, 스태킹, 블랜딩....(추가 필요)

2.Kaggle

2.1 참여 대회 소개

- 참여 대회 정보 : Kaggle - House Prices - Advanced Regression Techniques
- 배경 : 주택 구매자에게 자신이 꿈꾸는 집에 대해 설명해달라고 하면 '지하실 천장의 높이가 얼마이고, 주택과 철도의 근접성은 얼마여야 한다' 라고 설명을 시작하지 않을 것입니다. 그러나 이 대회의 데이터 세트는 침실의 수나 말뚝 울타리보다 가격 협상에 훨씬 더 많은 영향을 미친다는 것을 증명합니다. 이 대회는 아이오와주와 에임스주에 있는 주거용 주택의 거의 모든 측면을 설명하는 79 개의 설명변수가 있는 데이터를 제공합니다. 대회의 참가자들은 이 데이터로 각 주택의 최종 가격을 예측해야 합니다.
- 주제 : [Algorithm] Kaggle - House Prices - Advanced Regression Techniques 에서 제공하는 데이터셋을 활용한 집 값 예측 알고리즘 개발
- 주최/주관 : Kaggle
- 참가 대상 :

  (i) Kaggle.com 에 등록된 계정 소유자

  (ii) 18 세 이상이거나 귀하의 거주 관할권에서 성년인 자

  (대회 후원사가 달리 동의하고 대회 후원사가 적절한 부모/보호자의 동의를 얻은 경우는 제외)

  (iii) 소위 도네츠크 인민 공화국(DNR) 또는 루한스크 인민 공화국(LNR)이라 불리는 크리미아, 쿠바, 이란, 시리아 또는 북한의 거주자가 아니어야 한다.

  (iv) 미국 수출 통제 또는 제재 하에 있는 법인의 개인 또는 대리인이 아니어야 한다.

  (https://www.treasury.gov/resource-center/sanctions/Programs/Pages/Programs.aspx 참조).

2.2 문제 평가내용

- 목표: 주어진 데이터셋 중 테스트 세트의 각 Id 에 대해 SalePrice 변수의 값을 예측해야 한다.
- 회귀 평가 지표 : 제출물은 예상된 값의 로그와 관찰된 판매가격(SalePrice)의 로그 사이의 RMSE(Root-Mean-Squared-Error)로 평가된다.

  (로그를 취한다는 것은 비싼집과 싼 집을 예측하는 오류가 결과에 똑같이 영향을 미친다는 것을 의미한다.)
- 제출파일 형식 : 파일은 헤더를 포함해야 하며 다음 형식이어야 한다.

```
Id,SalePrice
1461,169000.1
1462,187724.1233
1463,175221
등
```

- RMSE(Root-Mean-Squared-Error)

  = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}$

3. 프로젝트 구성

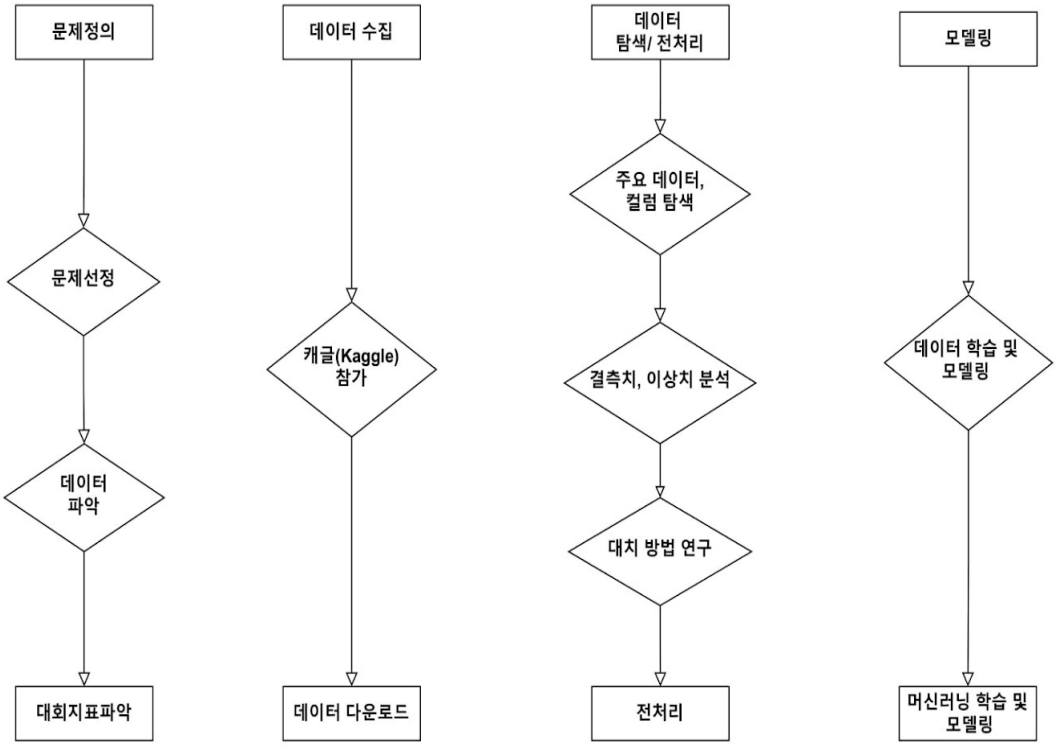
3.1. 플랫폼 아키텍처

3.2. 개발 Tool 활용

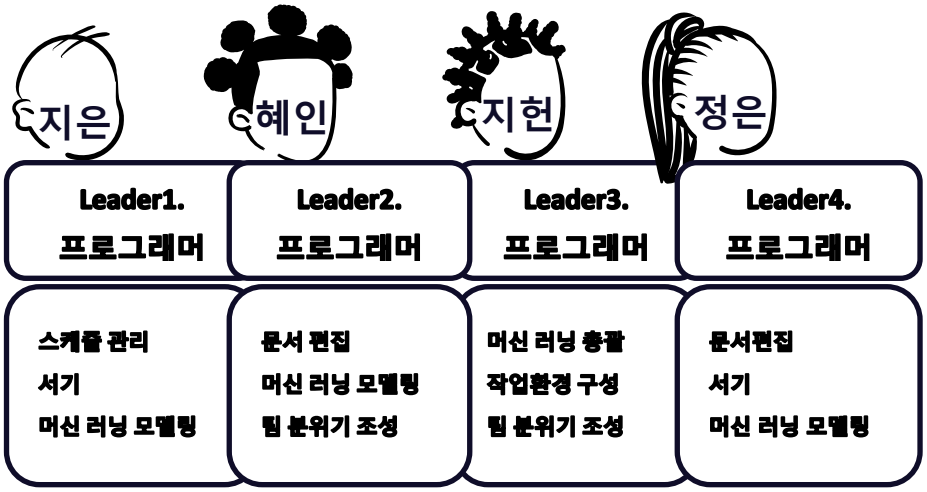
개발 Tool	업무범위
Jupyter notebook	■ 데이터 전처리 시 활용 ■ 머신 러닝 모델링
Google colab	■ 데이터 전처리 시 활용 ■ 머신 러닝 모델링

4. 프로젝트 추진체계

4.1 업무분류체계(Work Breakdown Structure, WBS)



4.2 조직 구성 및 역할



5. 프로젝트 일정 계획표

	1 주차					2 주차				
데이터 수집(주제선정)										
데이터 전처리										
데이터 분석 및 시각화										
결과 도출 및 발표자료 작성										

6. 데이터셋

6.1 데이터셋 구성

Data_description

Sample_submission.csv

train.csv

1460 rows × 81 columns

SalePrice(Target값)

test.csv

1460 rows × 80 columns

6.2

train data set information

column description	
	• MSSubClass: 판매와 관련된 주거 유형을 식별 • MSZoning: 판매의 일반 구역 분류를 식별 • LotFrontage: 부동산에 연결된 거리의 선형 피트 • LotArea: 부지 크기(제곱피트) • LotShape: 속성의 일반적인 모양 • LandContour: 속성의 평탄도 • LotConfig: 로트 구성 • LandSlope: 건물의 기울기 • BldgType: 주거 유형 • HouseStyle: 거주 스타일 • OverallQual: 집의 전반적인 재료와 마감을 평가 • OverallCond: 집의 전반적인 상태를 평가 • YearBuilt: 원래 건설 날짜 • YearRemodAdd: 리모델링 일자(개조나 증축이 없을 경우 공사일과 동일) • RoofStyle: 지붕 유형 • RoofMatl: 지붕 재료 • Exterior1st: 집 외부 덮음 • Exterior2nd: 집 외부 덮음(재료가 둘 이상인 경우) • MasVnrType: 벽돌 베니어 유형 • MasVnrArea: 평방 피트 단위의 석조 베니어 영역 • ExterQual: 외부 소재의 품질을 평가합니다. • ExterCond : 외부 자재의 현재 상태를 평가 • BsmtQual: 지하실의 높이를 평가 • BsmtCond: 지하실의 전반적인 상태를 평가 • BsmtExposure: 파업 또는 정원 수준 벽을 나타냅니다. • BsmtFinType1 : 지하 마감 면적 등급 • BsmtFinSF1: 유형 1 마감 평방 피트 • BsmtFinType2: 지하 마감 면적 등급(여러 유형인 경우) • BsmtFinSF2: 유형 2 마감 평방 피트 • BsmtUnfSF: 지하 공간의 미완성 평방 피트 • TotalBsmtSF: 지하 면적의 총 평방 피트

	• CentralAir: 중앙 에어컨 • 1stFlrSF: 1층 평방 피트 • 2ndFlrSF: 2층 평방 피트 • LowQualFinSF: 저품질 마감 평방 피트(모든 층) • GrLivArea: 지상 거실 면적 평방피트 • BsmtFullBath: 지하 전체 욕실 • BsmtHalfBath: 지하 반 욕실 • FullBath: 지상 위의 전체 욕실 • HalfBath: 지상 반 욕실 • KitchenQual: 주방 품질 • TotRmsAbvGrd: 지상 위의 객실 총수(욕실 제외) • FireplaceQu: 벽난로 품질 • GarageType: 차고 위치 • GarageYrBlt: 차고가 지어진 연도 • GarageFinish: 차고 내부 마감 • GarageCars: 차량 수용 가능 차고의 크기 • GarageArea: 평방 피트 단위의 차고 크기 • GarageQual: 차고 품질 • GarageCond: 차고 조건 • PavedDrive: 포장된 진입로 • WoodDeckSF: 목재 데크 면적(제곱피트) • OpenPorchSF: 평방 피트 단위의 열린 베란다 영역 • EnclosedPorch: 둘러싸인 베란다 면적(평방피트) • 3SsnPorch: 제곱피트 단위의 3계절 베란다 면적 • ScreenPorch: 스크린 베란다 면적(제곱피트) • PoolArea: 수영장 면적(제곱피트) • PoolQC: 풀 품질 • MiscFeature: 다른 범주에서 다루지 않는 기타 기능 • MiscVal: 기타 기능의 $ 가치 • MoSold: 판매 월(MM) • YrSold: 판매 연도(YYYY) • SaleType: 판매 유형 • SaleCondition: 판매 조건
test data set information	
same train columns descriptions	

7. 프로젝트 결과 및 보고

7.1 데이터 전처리

	정혜인	백지은
중점 전처리 과정	■ 시각화로 타겟값 확인해보기 - 타겟값인 SalePrice'를 산점도로 시각화해서 확인해본다. ■ 결측치 확인 및 처리 ■ 결측값이 있는 변수정리 ■ 범주형 타입 변화하기 ■ 라벨 인코딩 하기 ■ 파생 변수 생성 -각 피처들의 관련성이 따라 묶어 새로운 파생변수를 생성해주었다. ■ 수치형 변수 확인 ■ 범주형 변수 확인	■ 히트맵과 박스플롯으로 시각화하여 타겟값과 피처들간의 상관관계 확인 ■ 삭제해도 무방하다고 판단한 이상치 제거 ■ 결측값들을 피처 특성에 맞게 채워주고 라벨 인코딩 진행 ■ 타겟값과 관련성이 높다고 생각하는 변수를 묶어서 파생변수 생성 ■ 비대칭도(skew)가 높은 피처 Box Cox 변환 ■ 범주형 데이터 get dummy 변환

	안정은	김지헌
중점 전처리 과정	■ target(SalePrice)을 log 로 변환하여 정규분포로 만들어줌 ■ corr 함수로 상관관계가 (0.3 이하) 상관관계가 있는 컬럼을 추출하여 제거 ■ 다중공선성이 우려되는 컬럼을 추리고 각 컬럼들의 왜도와 첨도를 확인 ■ 수치형데이터 중 정규분포에서 치우친 컬럼들을 log 변환 처리해줌 ■ 범주형데이터 중 타겟값에 영향을 많이 끼치는 변수 추출 ■ 각 컬럼별로 결측치 확인 후 평균값으로 채워줌 ■ 범주형 데이터는 get_dummies 사용	■ 왜도(skew)가 1 이상인 컬럼만 추출 -> 중앙값으로 Null 값 채우기 ■ 범주형 데이터 결측치처리 ■ 원핫인코딩(get_dummies 사용) ■ corrwith 로 상관관계 확인 후 상관계수 낮은 컬럼 drop 시키기

7.2. 머신 러닝

	정혜인	백지은
머신 러닝	■ XGBoost 파라미터 튜닝 후 그리드서치로 최고 점수 도출 ■ LightGBMClassifier 파라미터 튜닝 그리드서치로 최고 점수 도출 ■ 앙상블, 보팅, 스태킹, 블랜딩 사용	■ rmse 함수 정의 ■ LinearRegression, Ridge, Lasso 학습, 예측, 평가 ■ Robust Scaler(이상치의 영향을 최소화) 후 Lasso (0.16), ElasticNet (0.16), Kernel Ridge (0.16), Gradient Boost Regression tree (0.16), XGBoost (0.2), LightGBM (0.16)으로 비중을 준 뒤 교차검증 수행
	안정은	김지헌
머신 러닝	■ rmse 함수 정의 ■ 각 scale 방법(log 함수, standard scaled, min-max scale)을 사용한 후 비교하여 효율적인 방법의 scaling 작업 진행 ■ 상관계수 확인과 데이터 디스크립션을 통해 변수들을 정리하여 다중공선성의 문제를 해결 ■ Linear Regression, Ridge, Lasso 학습, 예측, 평가 ■ cv 교차검증을 통해 특정 데이터셋에 대한 과적합을 방지하고 더욱 일반화된 모델을 생성하기 위해 중점을 두었다.	■ rmse 함수 정의 ■ LinearRegression, Ridge, Lasso 학습, 예측, 평가 ■ 상위 10 개 하위 10 개 coefficient(계수)추출해서 concat 으로 결합 ■ 3 개의 회귀모델의 회귀계수를 barchar 로 시각화 ■ 데이터를 분할하지 않고 교차검증 5 회 수행 - 모델별 CV RMSE 값과 평균 RMSE 값 출력 후 평균 계산 - 성능이 떨어지는 모델의 최적값을 도출하기 위해 하이퍼 파라미터 조정 - 반복작업을 위해 모델과 params 를 받아서 최적의 평균값과 alpha 를 반환하는 함수 생성 - 생성한 함수로 XGB, LGBM, LinearRegression, Ridge, Lasso 반복 실행 - 각 모델이 반환한 데이터 셋을 Stacking 형태로 결합하여 최종적으로 Lasso 모델 적용 - 도출된 예측값을 기반으로 새롭게 만들어진 학습 및 테스트용 데이터로 예측하고 RMSE 측정

7.3. 키메라(최종병기)

1. 필요 패키지 불러오기

[2]
```
import numpy as np# 넘파이, 수치해석용 파이썬 패키지
import pandas as pd #  파이썬에서 표 형태의 데이터를 탐색하고 분석하는데 사용하는 라이브러리
import io #웹정보를 제공하는 정해진 알고리즘으로 불러와 사용자에게 필요한 정보로 변환

import matplotlib as mpl # 파이썬에서 데이타를 차트나 플롯(Plot)으로 그려주는 라이브러리 패키지
import matplotlib.pyplot as plt # MATLAB과 비슷하게 명령어 스타일로 동작하는 함수의 모음
import seaborn as sns # Matplotlib을 기반으로 다양한 색상 테마와 통계용 차트 등의 기능을 추가한 시각화 패키지

%matplotlib inline


import warnings
warnings.filterwarnings('ignore') #오류창 안뜨게하기 위해서

sns.set_style('darkgrid')
plt.rc('figure', figsize=(10, 8))
```

[3]
```
%matplotlib inline
# Notebook을 실행한 브라우저에서 바로 그림을 볼 수 있게 해주는 것
%config InlineBackend.figure_format = 'retina'
#높은 해상도의 그래프 출력
```

[4]
```
mpl.rc('font', family = 'malgun_gothic') # 글자체 고딕
mpl.rc('axes', unicode_minus=False) # 유니코드 마이너스 오류 안뜨게
```

```
from sklearn.model_selection import train_test_split
# from sklearn.metrics import f1_score, confusion_matrix, precision_recall_curve, roc_curve
from scipy import stats
from scipy.stats import norm
# import statsmodels.api as sm

# 기계학습 모델 구축 및 평가 패키지
# import scipy as sp
# from scipy.stats import norm, skew

from sklearn.preprocessing import StandardScaler
#from sklearn.preprocessing import MinMaxScaler
#from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import RobustScaler
```

[6]
```
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV, StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from mlxtend.regressor import StackingRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import ElasticNet, Lasso, LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
```

2. 데이터 불러오기

```
house_train_org = pd.read_csv('/content/gdrive/MyDrive/kaggle/train.csv')
house_test = pd.read_csv('/content/gdrive/MyDrive/kaggle/test.csv')
house_train = house_train_org.copy() #원본을 유지하기 위해
house_train.head(3)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilit
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	All
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	All
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	All

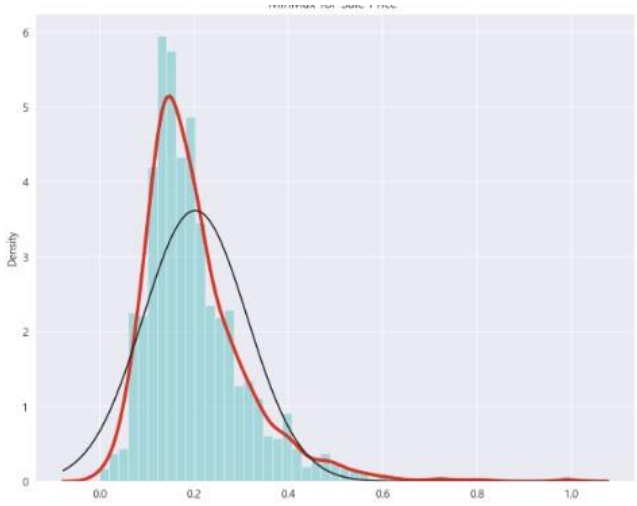
3 rows × 81 columns

3. 데이터 탐색 및 시각화

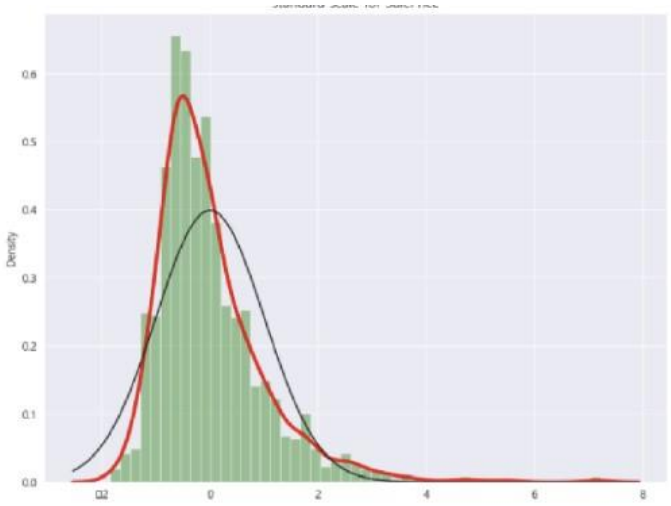
3.1 타겟값(SalePrice) 스케일링 :

왜도(skewness)와 첨도(kurtosis)를 개선하기 위해 로그 스케일링 적용

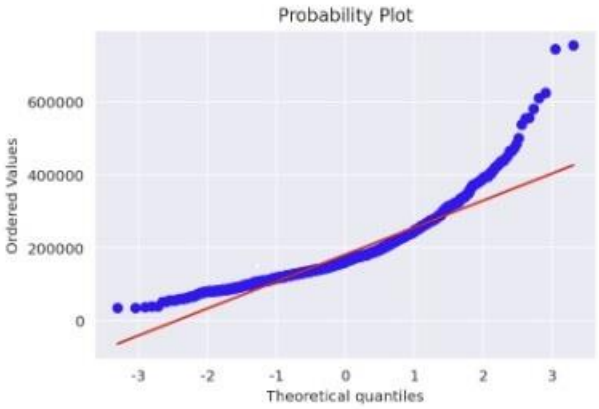
왜도: 0.121347	→	**-0.078154**
첨도: 0.809519	→	**0.957380**



<before scale>



<after scale>



<before scale>

<after scale>

- 왜도(skewness)는 분포의 비대칭도를 나타내는 통계량이다. 정규분포, T 분포와 같이 대칭인 분포의 경우 왜도가 0 이다.
- 첨도(kurtosis)는 분포의 꼬리부분의 길이와 중앙부분의 뾰족함에 대한 정보를 제공하는 통계량이다. 잘 정제된 통계량이 아니기 때문에 여전히 해석에 논란의 여지가 있다.

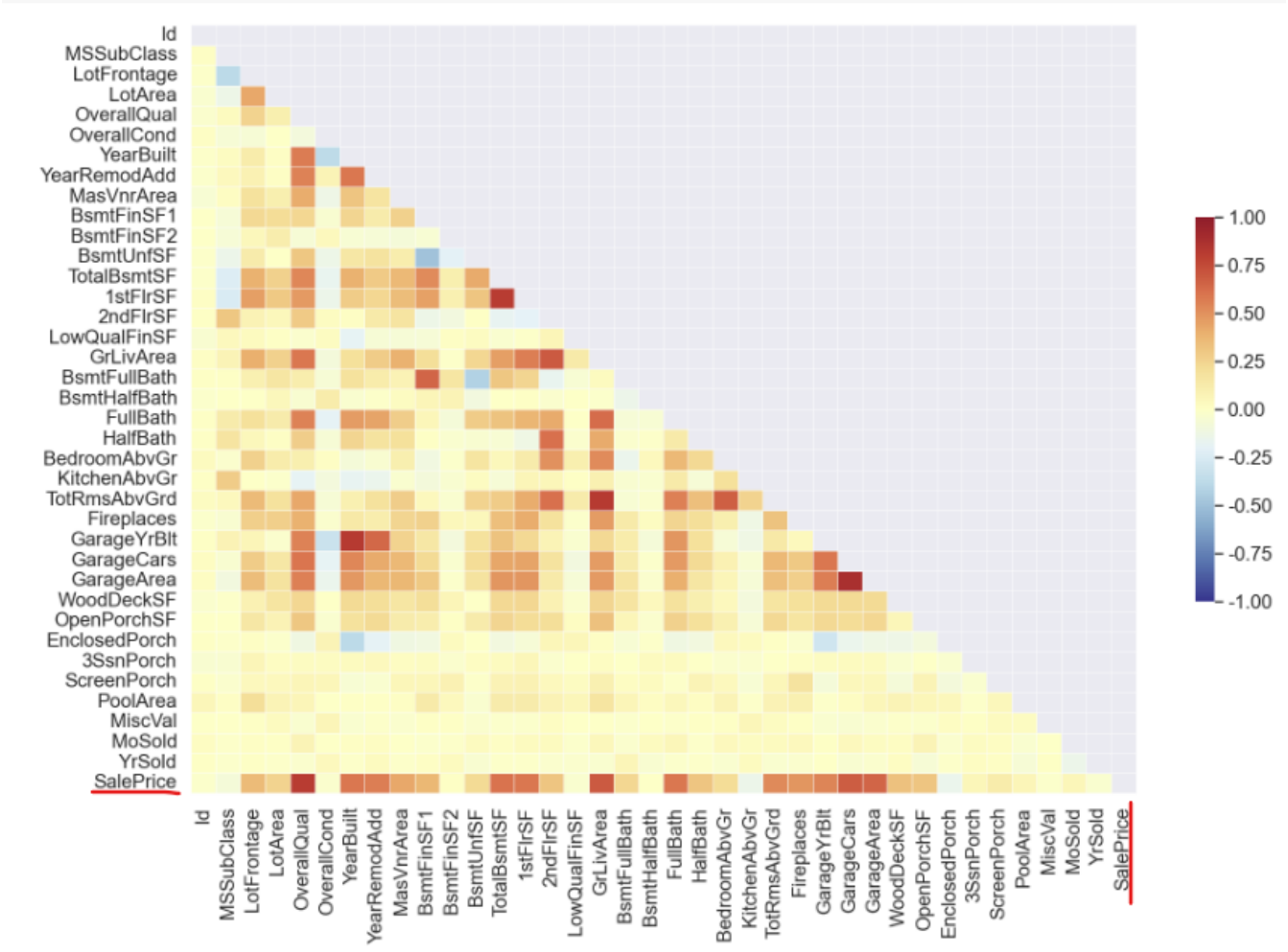
  정규분포의 첨도는 0 이다(기본적인 정의에 의하면 3 이지만, 일반적으로 정규분포의 첨도를 0 으로 만들기 위해 3 을 빼서 정의하는 경우가 많다. 정규분포가 0 이 되게 정의하는 첨도를 excess kurtosis 라고 한다). 정규분포처럼 첨도가 0 인 경우를 Mesokurtic 라고 한다.

  첨도가 0 보다 크면 정규분포보다 긴 꼬리를 갖고, 분포가 보다 중앙부분에 덜 집중되게 되므로 중앙부분이 뾰족한 모양을 가지게 된다. T 분포는 정규분포보다 더 긴 꼬리를 가지므로 첨도가 0 보다 크다. 첨도가 0 보다 큰 경우를 Leptokurtic 이라고 한다.

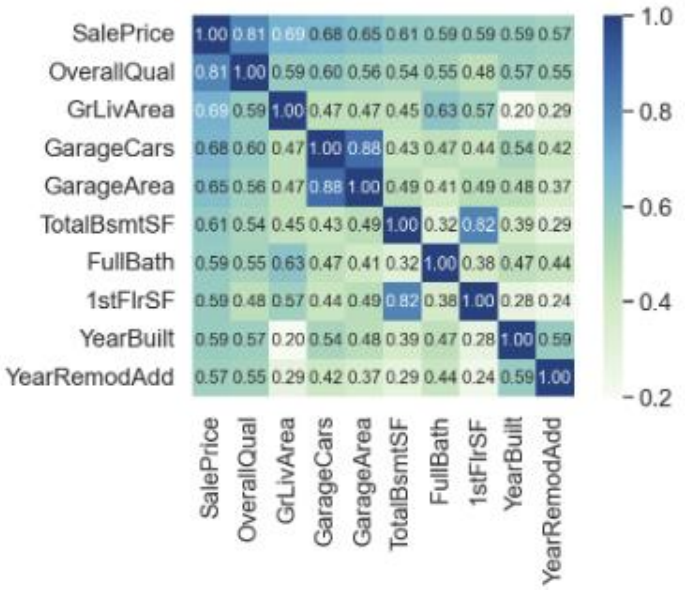
  첨도가 0 보다 작으면, 정규분포보다 짧은 꼬리를 갖고 분포가 중앙부분에 더 집중되어 중앙부분이 보다 완만한 모양을 가지게 된다

3.2 타겟값과 피처들의 상관성 확인하기

- 상관관계가 0.3 이상인 변수들만 히트맵으로 출력,
- 옅은 부분이 상관관계가 높은 것으로
- 두 변수 간에 상관관계가 너무 강하면 다중공산성 상황이 나타날 수도 있다.



모든 피처를 다 시각화하니
보기가 어려워서 SalePrice 와
상관관계가 높은 피처 10 개만
히트맵으로 표현



데이터에 따르면
'OverallQuality', 'GrLivArea',
'TotalBsmtSF', 'YearBuilt'
같은 변수가 관련이 높아 보인다.
최대한 효율적으로 탐색하기 위해
관련이 높을 것 같은 카테고리를 먼저
주의 깊게 살펴본 후,
전체적인 데이터에 집중할 계획이다.

3.3 상관관계가 0.3 이상인 것만 보기로 하여 골라내는 작업

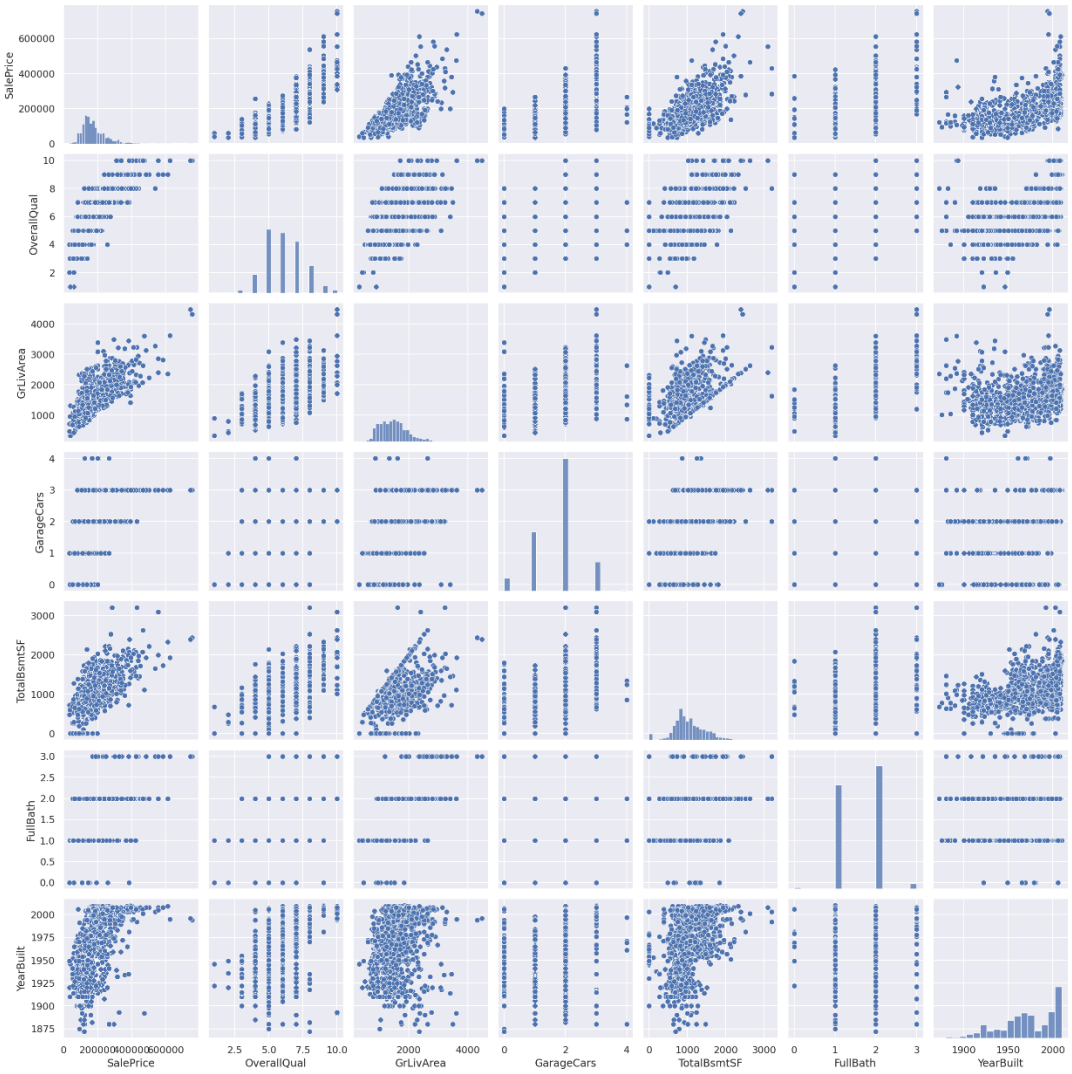
```
cor = house_train.corr()
cor_fe = cor.index[abs(cor['SalePrice']) >= 0.3]
cor_fe
```

상관관계가 어느 정도 있다는 것을 알려면, 0.3~05 어느정도의 양의 상관관계가 있다. 0.5~강한 양의 상관관계가 있다고 보기에 0.3 이상으로 관계가 있는 column 들만 들고 온다.

--------[상관계수 보는 법]--------
0.9 이상 : 상관관계가 아주 높다
0.7 ~ 0.9 : 상관관계가 높다
0.4 ~ 0.7 : 상관관계가 있다
0.2 ~ 0.4 : 상관관계가 있으나 낮다
0.2 미만 : 상관관계가 거의 없다

```
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
YearBuilt       0.522897
YearRemodAdd    0.507101
GarageYrBlt     0.486362
MasVnrArea      0.477493
Fireplaces      0.466929
BsmtFinSF1      0.386420
LotFrontage     0.351799
WoodDeckSF      0.324413
2ndFlrSF        0.319334
OpenPorchSF     0.315856
HalfBath        0.284108
LotArea         0.263843
BsmtFullBath    0.227122
BsmtUnfSF       0.214479
BedroomAbvGr    0.168213
KitchenAbvGr    0.135907
EnclosedPorch   0.128578
ScreenPorch     0.111447
PoolArea        0.092404
MSSubClass      0.084284
OverallCond     0.077856
MoSold          0.046432
3SsnPorch       0.044584
YrSold          0.028923
LowQualFinSF    0.025606
Id              0.021917
MiscVal         0.021190
BsmtHalfBath    0.016844
BsmtFinSF2      0.011378
Name: SalePrice, dtype: float64
```

3.4 관련성이 높은 변수들만 산점도 확인



'TotalBsmtSF'와 'GrLiveArea'의 상관성을 보면 점들이 거의 경계선처럼 그려져있다.

TotalBsmtSF: 지하 면적의 총 평방피트

GrLivArea: 지상(지상) 거실 면적 평방피트를 의미한다

그것을 감안하면 지상공간보다 큰 지하공간은

(벙커를 사지않는 한) 불가능하다는 점에서 해석할 수 있다.

3.5 target 값 外 변수들 확인

```
numerical_feats = house_train.dtypes[house_train.dtypes != "object"].index
print(house_train[numerical_feats].columns)
print("수치형 데이터 수: ", len(numerical_feats))
print('-'*100)
categorical_feats = house_train.dtypes[house_train.dtypes == "object"].index
print(house_train[categorical_feats].columns)
print("범주형 데이터 수: ", len(categorical_feats))
```

```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
       'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
수치형 데이터 수:  38
----------------------------------------------------------------------------------------------------
Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
범주형 데이터 수:  43
```

- 수치형 데이터와 범주형 데이터를 분리하고 개수를 추출했다
- 수치형 데이터 38 – 범주형 데이터 43

3.6 이상치 시각화

· 이상치 시각화를 위해 수치형 데이터와 범주형 데이터를 구분할 필요가 있음
· 수치형 데이터지만 범주형 데이터도 있다.

FullBath, MoSold, GarageCars, Fireplaces, TotRmsAbvGrd, KitchenAbvGr, BedroomAbvGr, HalfBath, BsmtFullBath, BsmtHalfBath, BsmtFullBath, YrSold, GarageYrBlt, YearRemodAdd, YearBuilt, OverallQual, MSSubClass, OverallCond, YrSold

· 이상의 컬럼들을 수치형 데이터로 나타난 범주형 데이터라 판단하였으나 이미 라벨 인코딩 된것과 다름 없는 점수형 데이터이거나(ex 1 은 매우 나쁨 10 은 매우 좋음) 불연속적인 연도 데이터 였기에 따로 라벨인코딩을 하지 않았음

· 최종적으로 MSSubClass 를 문자형 데이터로 변환해줌

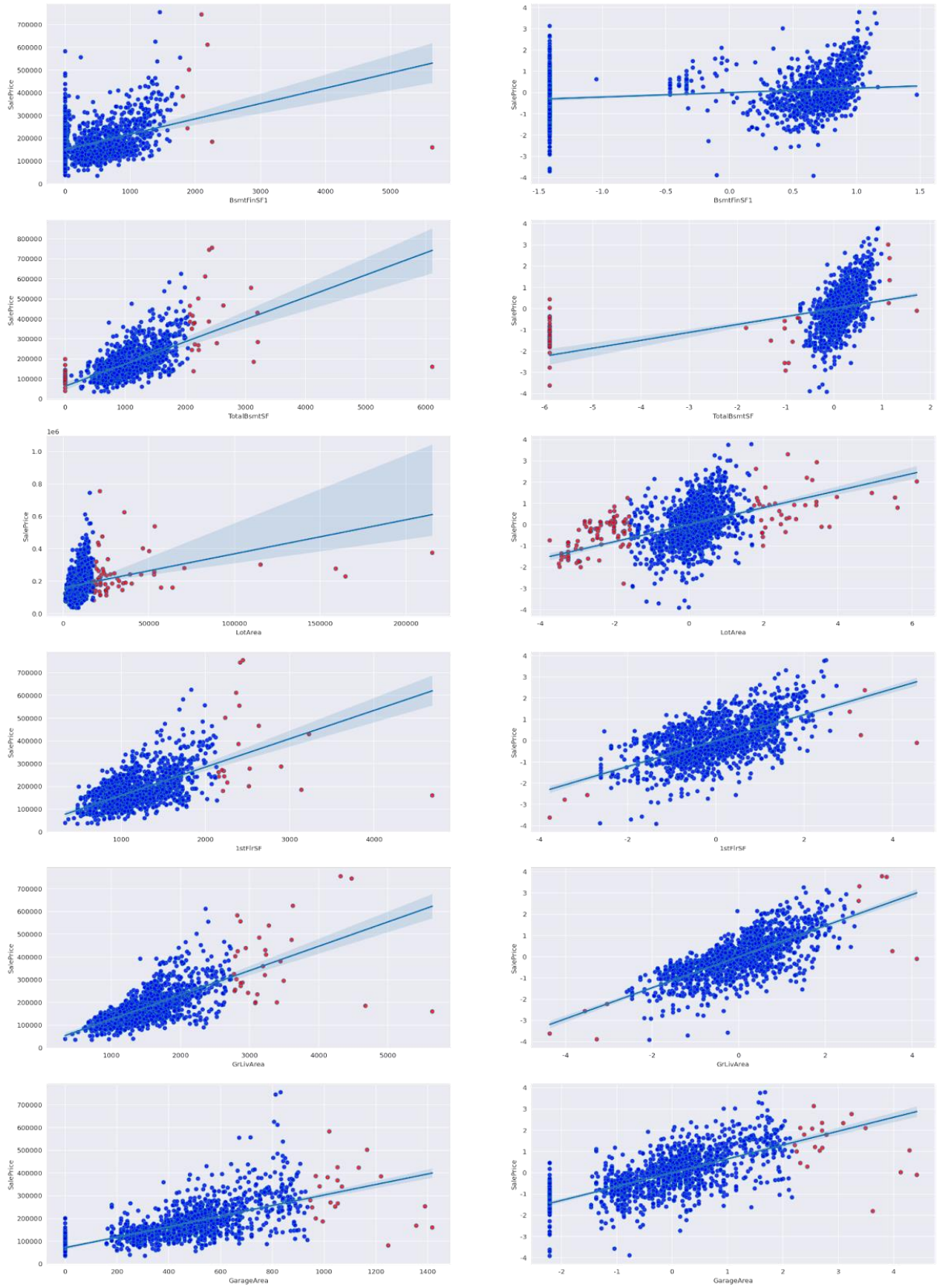
```
# 이상치를 산점도로 시각화 하기 위한 함수
# 입력 받은 데이터 프레임의 각 컬럼에서 np.percentile을 이용하여 1사분위수와 3사분위수를
# iqr을 구한다.
# 이후 max값 min값을 구한 뒤 max값을 초과하고 min값의 미만인 데이터의 인덱스를 구해 mas
# 이상치를 시각화할 컬럼의 컬럼명에 _col이 붙은 새로운 컬럼을 생성하고
# mask에서 인덱스가 담긴 배열을 가지고 for문을 이용하여 이상치가 위치한 행에 red를 담는

def find_outlier(data):
    for i in data.columns:
        if data[i].dtype != object:
            q1, q3 = np.percentile(data[i], [25, 75])
            iqr = q3 - q1
            lower_bound = q1 - (iqr * 1.5)
            upper_bound = q3 + (iqr * 1.5)
            mask = np.where((data[i] > upper_bound) | (data[i]< lower_bound))
            a = i +'_col'
            data[a] = 'blue'
            for j in mask:
                data[a][j] = 'red'
```

· 각 컬럼에서 이상치를 찾고 해당 컬럼명 + '_col' 을 새로운 컬럼명으로하여 이상치가 있는 행에 red 로 수정하는 함수

```
# 산점도를 그리는데 너무 많은 그래프를 보여주는것은 좋지 않다 판단하여 몇가지를 6개의 컬
# 왼쪽은 스케일전에서 본 이상치이고 오른쪽은 로그변환후 스탠다드 스케일까지 거친 데이터
# 2개의 열, 6개의 행으로 이뤄져 있으며
# sns.regplot을 그리는데 ax =axs[행][열]로 그래프가 그려질 위치를 지정한다
# 열의 경우 위치가 고정이기에 각각 0과 1로 줬으며
# for문이 반복될 때 마다 새로운 행에 그래프를 그린다.

fig, axs =plt.subplots(figsize=(24,35), ncols = 2, nrows=6)
house_train_plt_columns = ['BsmtFinSF1', 'TotalBsmtSF', 'LotArea', '1stFlrSF',
                           'GrLivArea', 'GarageArea']

for i, features in enumerate(house_train_plt_columns):
    sns.regplot(x=house_train_non_scale[features],
                y = house_train_non_scale['SalePrice'],
                scatter_kws={'facecolors': house_train_non_scale[(features + '_col')]},
                ax = axs[i][0])

    sns.regplot(x=scaled_ss_data[features],
                y = scaled_ss_data['SalePrice'],
                scatter_kws={'facecolors': scaled_ss_data[(features + '_col')]},
                ax = axs[i][1])
```

스케일되지 않은 데이터와 스케일된 데이터로 그리는 작업업

산점도는 데이터를 바탕으로 그리고 산점도의 색깔은 find_outlier 함수로 생성된 컬럼에서 조회한 값으로 표현해줌

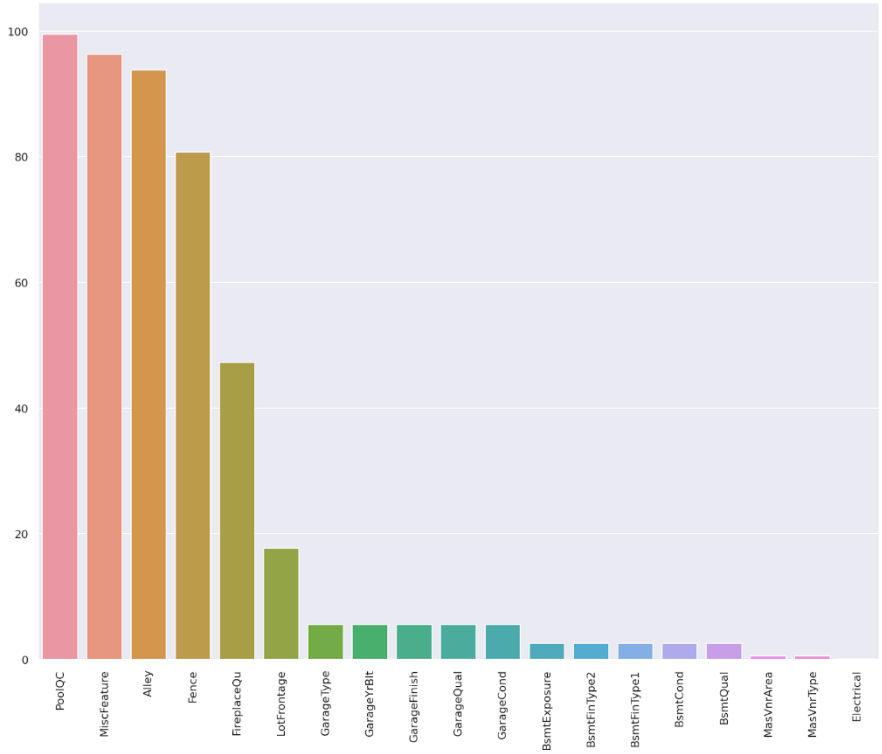


스케일 한 뒤 이상치를 잡는 범위가 달라지는 것을 확인할 수 있다.

3.7 결측치 시각화

- 결측치를 가지고 있는 변수와 결측치 비율을 확인하고, barplot 으로 시각화 함

	결측값 비율
PoolQC	99.520548
MiscFeature	96.301370
Alley	93.767123
Fence	80.753425
FireplaceQu	47.260274
LotFrontage	17.739726
GarageType	5.547945
GarageYrBlt	5.547945
GarageFinish	5.547945
GarageQual	5.547945
GarageCond	5.547945
BsmtExposure	2.602740
BsmtFinType2	2.602740
BsmtFinType1	2.534247
BsmtCond	2.534247
BsmtQual	2.534247
MasVnrArea	0.547945
MasVnrType	0.547945
Electrical	0.068493



4 트레인 데이터 피처 스케일링

4.1 이상치 제거

- 극단적인 이상치를 확인한 후 중앙값으로 대체한다

```
# LotFrontage 컬럼의 극단적 이상치 확인
house_train['LotFrontage'][197]
```

```
174.0
```

```
# LotFrontage 컬럼의 중앙값
house_train['LotFrontage'].median()
```

```
69.0
```

```
def transform_outlier(house_train_scatter_color_numeric, outliers_c, outliers):
    for i, j in zip(outliers_c, outliers):
        a = house_train_scatter_color_numeric[i].median()
        print(j)
        for k in eval(j):
            print(k)
            house_train_scatter_color_numeric[i][k] = a
```

이상치 제거 함수 정의 : 이상치를 탐색해서 중앙값으로 대체하는 함수

이상치를 제거할 데이터명과 이상치가 존재하는 컬럼명, 그 컬럼의 인덱스가 존재하는 리스트명들을 가진 리스트를 입력한다. eval 을 사용해야 outliers 에서 j 로 가져온 리스트명이 문자열 데이터가 아니라 변수명으로 작동한다.

4.2 결측치 제거

```
# PoolQC : 결측값의 비율이 99%로 매우 높고 대부분 주택에 수영장이 없음을 감안할 때 의미있는 변수입니다.
house_train['PoolQC'] = house_train['PoolQC'].fillna('None')
# MiscFeature : 기타기능 없음을 의미합니다.
house_train['MiscFeature'] = house_train['MiscFeature'].fillna('None')
# Alley : 골목 접근 금지를 의미합니다.
house_train['Alley'] = house_train['Alley'].fillna('None')
# Fence : 울타리가 없는 것을 의미합니다.
house_train['Fence'] = house_train['Fence'].fillna('None')
# FireplaceQu : 난로가 없는 것을 의미합니다.
house_train['FireplaceQu'] = house_train['FireplaceQu'].fillna('None')
```

```
# LotFrontage : 주택건물에 연결된 각 거리의 면적이 주변의 다른 주택과 비슷한 지역일 가능성이 높기 때문에
# 이웃(Neighborhood)의 LotFrontage의 중앙값으로 결측치를 채웁니다.
house_train['LotFrontage'] = house_train.groupby('Neighborhood')['LotFrontage'].transform(lambda x:x.fillna(x.median()))
```

```
# GarageType, GarageFinish, GarageQual, GarageCond : 누락된 데이터(결측치)를 None값으로 대체합니다.
# BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFin Type2
# : 지하실 기능 관련데이터의 NaN(결측값)은 None으로 대체하겠습니다.
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
            'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    house_train[col] = house_train[col].fillna('None')
```

```
# GarageYrBlt, GarageArea, GarageCars : 차고가 없는 것은 결측치를 0으로 대체합니다.
# BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath
# : 지하실이 없는 것은 결측치를 0으로 대체합니다.
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars',
            'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath'):
    house_train[col] = house_train[col].fillna(0)
```

```
# 이 집들에 대한 고정 베니어가 없다는 것을 의미합니다.
# MasVnrType: None을 대체합니다.
# MasVnrArea : 0을 대체합니다.
house_train['MasVnrType'] = house_train['MasVnrType'].fillna('None')
house_train['MasVnrArea'] = house_train['MasVnrArea'].fillna(0)
```

```
# MSZoning(The general zoning classification):'RL'이 가장 일반적인 값입니다. 따라서 'RL'로 결측값을 대체하겠습니다.
house_train['MSZoning'] = house_train['MSZoning'].fillna(house_train['MSZoning'].mode()[0])
```

```
# Utilities : 'NoSeWa'의 범주와 기능이 겹치므로 변수를 제거해주겠습니다.
house_train = house_train.drop(['Utilities'], axis = 1)
```

```
# Functional : 여기서 NA는 typical을 의미하므로 결측값을 'Typ'로 대체하겠습니다.
house_train['Functional'] = house_train['Functional'].fillna('Typ')
```

```
# Electrical : 여기서 NA는 'SBrkr'을 의미하므로 이것으로 대체하겠습니다.
house_train['Electrical'] = house_train['Electrical'].fillna(house_train['Electrical'].mode()[0])
```

```
# KitchenQual : 여기서 NA는 'TA'(가장빈번)을 의미하므로 이것으로 대체하겠습니다.
house_train['KitchenQual'] = house_train['KitchenQual'].fillna(house_train['KitchenQual'].mode()[0])
```

```
# Exterior1st, Exterior2nd : 동일한 결측값으로 대체하겠습니다.
house_train['Exterior1st'] = house_train['Exterior1st'].fillna(house_train['Exterior1st'].mode()[0])
house_train['Exterior2nd'] = house_train['Exterior2nd'].fillna(house_train['Exterior2nd'].mode()[0])
```

```
# SaleType : 가장 빈번한 데이터인 WD로 결측값 대체하겠습니다.
house_train['SaleType'] = house_train['SaleType'].fillna(house_train['SaleType'].mode()[0])
```

```
# MSSubClass : 여기서 Na는 건물 등급이 없다는 것을 의미하므로 None으로 결측값을 대체하겠습니다.
house_train['MSSubClass'] = house_train['MSSubClass'].fillna('None')
```

각 컬럼에 맞게 결측치를 제거하거나 대체해주는 작업을 진행한다

```
# 결측치 확인하기
null_df = (house_train.isna().sum() / len(house_train)) * 100
null_df = null_df.drop(null_df[null_df == 0].index).sort_values(ascending = False)
missing_data = pd.DataFrame({'결측값 비율' : null_df})
missing_data
```

결측값 비율

결측값이 모두 없어진 것을 확인할 수 있다

4.3 라벨 인코딩

라벨 인코딩 : 서열이 있는 것

ex) 연도, (초등학교, 중학교, 대학교)

Street, Alley, LotShape. LandContour, LandSlope, OverallQual, OverallCond, YearBuilt, YearRemodAdd, ExterQual

ExterCond, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, HeatingQC, CentralAir BsmtFullBath, BsmtHalfBath, FullBath, HalfBath, FireplaceQu, BedroomAbvGr, KitchenAbvGr, KitchenQual, Fireplaces, SaleCondition, YrSold, MoSold, PoolQC, GarageCond, GarageQual, GarageCars, GarageYrBlt

```
house_train = house_train.replace({'Street' :{'Grvl':1,'Pave':2},
                 "Alley" : {"None" : 0,"Grvl" : 1, "Pave" : 2},
                 'LandContour' : {'Bnk' : 0, 'Lvl': 1, 'Low': 2, 'HLS': 3},
                  "BsmtCond" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "BsmtExposure" : {"None" : 0, "Mn" : 1, "Av": 2, "Gd" : 3},
                  'SaleCondition' : {'AdjLand' : 1, 'Abnorml' : 2, 'Family' : 3, 'Alloca' : 4, 'Normal' : 5, 'Partial' : 6},
                  "BsmtFinType1" : {"None" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4,
                                    "ALQ" : 5, "GLQ" : 6},
                  "BsmtFinType2" : {"None" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4,
                                    "ALQ" : 5, "GLQ" : 6},
                  "BsmtQual" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd" : 4, "Ex" : 5},
                  "ExterCond" : {"Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" : 5},
                  "ExterQual" : {"Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" : 5},
                  'CentralAir' : {'N' : 0, 'Y' : 1},
                  "FireplaceQu" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "Functional" : {"Sal" : 1, "Sev" : 2, "Maj2" : 3, "Maj1" : 4, "Mod": 5,
                                  "Min2" : 6, "Min1" : 7, "Typ" : 8},
                  "GarageCond" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "GarageQual" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "HeatingQC" : {"Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "KitchenQual" : {"Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "LandSlope" : {"Sev" : 1, "Mod" : 2, "Gtl" : 3},
                  "LotShape" : {"IR3" : 1, "IR2" : 2, "IR1" : 3, "Reg" : 4},
                  "PavedDrive" : {"N" : 0, "P" : 1, "Y" : 2},
                  "PoolQC" : {"None" : 0, "Fa" : 1, "TA" : 2, "Gd" : 3, "Ex" : 4},
                  "Street" : {"Grvl" : 1, "Pave" : 2},
                  "Utilities" : {"ELO" : 1, "NoSeWa" : 2, "NoSewr" : 3, "AllPub" : 4}}
                )
```

라벨인코딩이 필요한 컬럼들의 값(value)들에 라벨을 지정해줌

4.4 로그 변환

```
from scipy.stats import skew
# 왜도(치우쳐진 정도)

# object가 아닌 숫자형 피처의 칼럼 index 객체 추출.
features_index = house_train.dtypes[house_train.dtypes != 'object'].index
# house_nonNa에 칼럼 index를 [ ]로 입력하면 해당하는 칼럼 데이터 세트 반환. apply lambda로 skew( ) 호출
skew_features = house_train[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))
```

```
# 왜도가 높은 컬럼명 추출 후 로그변환
house_train[skew_features_top.index] = np.log1p(house_train[skew_features_top.index])
```

숫자형 피처의 칼럼 index 를 추출함

왜도 측정 함수를 lambda 함수로 정의하여 왜도 1 이상인 칼럼만 추출해서 로그변환 작업을 진행한다.

```
# object가 아닌 숫자형 피처의 칼럼 index 객체 추출.
features_index1 = house_train.dtypes[house_train.dtypes != 'object'].index
# house_nonNa에 칼럼 index를 [ ]로 입력하면 해당하는 칼럼 데이터 세트 반환. apply lambda로 skew( ) 호출
skew_features1 = house_train[features_index1].apply(lambda x : skew(x))
# skew(왜곡) 정도가 1 이상인 칼럼만 추출.
skew_features_top1 = skew_features1[skew_features1 > 1]
print(skew_features_top1.sort_values(ascending=False))
```

```
PoolQC           15.711689
PoolArea         14.348342
3SsnPorch         7.727026
LowQualFinSF      7.452650
MiscVal           5.312458
Alley             3.949996
BsmtHalfBath      3.929022
KitchenAbvGr      3.865437
ScreenPorch       3.179407
BsmtFinSF2        2.521100
EnclosedPorch     2.110104
BsmtFinType2      1.861672
dtype: float64
```

왜도가 1 이상인 컬럼명을 저장하기 위해 skew_features_top 에 담음
이후 테스트 데이터에도 여기서 생성된 컬럼명들만 로그 변환해준다.

4.5 원핫인코딩

원핫인코딩 : 서열이 없는 것,
ex 국가명, 사람이름
MSSubClass, MSZoning, LotConfig, Neighborhood, Condition1, Condition2, BldgType, PavedDrive
HouseStyle, RoofStyle, RoofMatl, Exterior1st, Exterior2nd, MasVnrType, Foundation, GarageFinish
Heating, Electrical, TotRmsAbvGrd, Functional, GarageType SaleType, MiscFeature, Fence

```
# 문자형 컬럼 추출
house_train.dtypes[house_train.dtypes == object]
```

```
MSSubClass      object
MSZoning        object
LotConfig       object
Neighborhood    object
Condition1      object
Condition2      object
BldgType        object
HouseStyle      object
RoofStyle       object
RoofMatl        object
Exterior1st     object
Exterior2nd     object
MasVnrType      object
Foundation      object
BsmtExposure    object
Heating         object
Electrical      object
GarageType      object
GarageFinish    object
Fence           object
MiscFeature     object
SaleType        object
dtype: object
```

```
# 원핫 인코딩을 하고 house_train_ohe에 담음
house_train_ohe = pd.get_dummies(house_train)
```

```
# 원핫 인코딩 확인
house_train_ohe
```

	Id	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LandSlope	OverallQual	OverallCond
0	1	65.0	9.042040	2	0.0	4	0.693147	3	7	5
1	2	80.0	9.169623	2	0.0	4	0.693147	3	6	8
2	3	68.0	9.328212	2	0.0	3	0.693147	3	7	5
3	4	60.0	9.164401	2	0.0	3	0.693147	3	7	5
4	5	84.0	9.565284	2	0.0	3	0.693147	3	8	5
...	...	...	...	...	...	...	...	...	...	...
1455	1456	62.0	8.976894	2	0.0	4	0.693147	3	6	5
1456	1457	85.0	9.486152	2	0.0	4	0.693147	3	6	6
1457	1458	66.0	9.109746	2	0.0	4	0.693147	3	7	9
1458	1459	68.0	9.181735	2	0.0	4	0.693147	3	5	6
1459	1460	75.0	9.204121	2	0.0	4	0.693147	3	5	6

1460 rows × 239 columns

Train 과 Test 의 분포를 확인

모든 변수에 대해서 비교해봤을 때, Train 과 Test 의 분포는 거의 동일하였다. 그래서 이 부분에서는 Train 에 오버피팅(Overfitting)되는 위험은 없을 것이라 생각하고 넘어갔다.

5 테스트 데이터 피처 스케일링

5.1 결측치 제거

```
# 똑같이 수치형이지만 범주형인 컬럼을 문자형으로 변환해 준다.
house_test['MSSubClass'] = house_test['MSSubClass'].astype(str)
```

```
# 결측치 확인
house_test.isna().sum()[house_test.isna().sum() >0]
```

```
# PoolQC : 결측값의 비율이 99%로 매우 높고 대부분 주택에 수영장이 없음을 감안할 때 의미있는 변수입니다.
house_test['PoolQC'] = house_test['PoolQC'].fillna('None')
# MiscFeature : 기타기능 없음을 의미합니다.
house_test['MiscFeature'] = house_test['MiscFeature'].fillna('None')
# Alley : 골목 접근 금지를 의미합니다.
house_test['Alley'] = house_test['Alley'].fillna('None')
# Fence : 울타리가 없는 것을 의미합니다.
house_test['Fence'] = house_test['Fence'].fillna('None')
# FireplaceQu : 난로가 없는 것을 의미합니다.
house_test['FireplaceQu'] = house_test['FireplaceQu'].fillna('None')
```

```
# LotFrontage : 주택건물에 연결된 각 거리의 면적이 주변의 다른 주택과 비슷한 지역일 가능성이 높기 때문에
# 이웃(Neighborhood)의 LotFrontage의 중앙값으로 결측치를 채웁니다.
house_test['LotFrontage'] = house_test.groupby('Neighborhood')['LotFrontage'].transform(lambda x:x.fillna(x.median()))
```

```
# GarageType, GarageFinish, GarageQual, GarageCond : 누락된 데이터(결측치)를 None값으로 대체합니다.
# BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFin Type2
# : 지하실 기능 관련데이터의 NaN(결측값)은 None으로 대체하겠습니다.
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
            'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    house_test[col] = house_test[col].fillna('None')
```

```
# GarageYrBlt, GarageArea, GarageCars : 차고가 없는 것은 결측치를 0으로 대체합니다.
# BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath
# : 지하실이 없는 것은 결측치를 0으로 대체합니다.
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars',
            'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath'):
    house_test[col] = house_test[col].fillna(0)
```

```
# 이 집들에 대한 고정 베니어가 없다는 것을 의미합니다.
# MasVnrType: None을 대체합니다.
# MasVnrArea : 0을 대체합니다.
house_test['MasVnrType'] = house_test['MasVnrType'].fillna('None')
house_test['MasVnrArea'] = house_test['MasVnrArea'].fillna(0)
```

```
# MSZoning(The general zoning classification):'RL'이 가장 일반적인 값입니다. 따라서 'RL'로 결측값을 대체하겠습니다.
house_test['MSZoning'] = house_test['MSZoning'].fillna(house_test['MSZoning'].mode()[0])
```

```
# Utilities : 'NoSeWa'의 범주와 기능이 겹치므로 변수를 제거해주겠습니다.
house_test = house_test.drop(['Utilities'], axis = 1)
```

```
# Functional : 여기서 NA는 typical을 의미하므로 결측값을 'Typ'로 대체하겠습니다.
house_test['Functional'] = house_test['Functional'].fillna('Typ')
```

```
# Electrical : 여기서 NA는 'SBrkr'을 의미하므로 이것으로 대체하겠습니다.
house_test['Electrical'] = house_test['Electrical'].fillna(house_test['Electrical'].mode()[0])
```

```
# KitchenQual : 여기서 NA는 'TA'(가장빈번)을 의미하므로 이것으로 대체하겠습니다.
house_test['KitchenQual'] = house_test['KitchenQual'].fillna(house_test['KitchenQual'].mode()[0])
```

```
# Exterior1st, Exterior2nd : 똑같은 결측값으로 대체하겠습니다.
house_test['Exterior1st'] = house_test['Exterior1st'].fillna(house_test['Exterior1st'].mode()[0])
house_test['Exterior2nd'] = house_test['Exterior2nd'].fillna(house_test['Exterior2nd'].mode()[0])
```

```
# SaleType : 가장 빈번한 데이터인 WD로 결측값 대체하겠습니다.
house_test['SaleType'] = house_test['SaleType'].fillna(house_test['SaleType'].mode()[0])
```

```
# MSSubClass : 여기서 Na는 건물 등급이 없다는 것을 의미하므로 None으로 결측값을 대체하겠습니다.
house_test['MSSubClass'] = house_test['MSSubClass'].fillna('None')
```

각 컬럼에 맞게 결측치를 제거하거나 대체해주는 작업을 진행한다

```
# 결측치 확인하기
null_df = (house_test.isna().sum() / len(house_test)) * 100
null_df = null_df.drop(null_df[null_df == 0].index).sort_values(ascending = False)
missing_data = pd.DataFrame({'결측값 비율' : null_df})
missing_data
```

결측값이 모두 없어진것을 확인할 수 있다

5.2 라벨 인코딩

- 라벨 인코딩 : 서열이 있는 것
- ex) 연도, (초등학교, 중학교, 대학교)
- Street, Alley, LotShape. LandContour, LandSlope, OverallQual, OverallCond, YearBuilt, YearRemodAdd, ExterQual
- ExterCond, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, HeatingQC, CentralAir
- BsmtFullBath, BsmtHalfBath, FullBath, HalfBath, FireplaceQu, BedroomAbvGr, KitchenAbvGr, KitchenQual, Fireplaces, SaleCondition, YrSold, MoSold, PoolQC, GarageCond, GarageQual, GarageCars, GarageYrBlt

```
house_test = house_test.replace({'Street' :{'Grvl':1,'Pave':2},
                 "Alley" : {"None" : 0,"Grvl" : 1, "Pave" : 2},
                 'LandContour' : {'Bnk' : 0, 'Lvl': 1, 'Low': 2, 'HLS': 3},
                  "BsmtCond" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "BsmtExposure" : {"None" : 0, "Mn" : 1, "Av": 2, "Gd" : 3},
                  'SaleCondition' : {'AdjLand' : 1, 'Abnorml' : 2, 'Family' : 3, 'Alloca' : 4, 'Normal' : 5, 'Partial' : 6},
                  "BsmtFinType1" : {"None" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4,
                                    "ALQ" : 5, "GLQ" : 6},
                  "BsmtFinType2" : {"None" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4,
                                    "ALQ" : 5, "GLQ" : 6},
                  "BsmtQual" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd" : 4, "Ex" : 5},
                  "ExterCond" : {"Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" : 5},
                  "ExterQual" : {"Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" : 5},
                  'CentralAir' : {'N' : 0, 'Y' : 1},
                  "FireplaceQu" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "Functional" : {"Sal" : 1, "Sev" : 2, "Maj2" : 3, "Maj1" : 4, "Mod": 5,
                                  "Min2" : 6, "Min1" : 7, "Typ" : 8},
                  "GarageCond" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "GarageQual" : {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "HeatingQC" : {"Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "KitchenQual" : {"Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
                  "LandSlope" : {"Sev" : 1, "Mod" : 2, "Gtl" : 3},
                  "LotShape" : {"IR3" : 1, "IR2" : 2, "IR1" : 3, "Reg" : 4},
                  "PavedDrive" : {"N" : 0, "P" : 1, "Y" : 2},
                  "PoolQC" : {"None" : 0, "Fa" : 1, "TA" : 2, "Gd" : 3, "Ex" : 4},
                  "Street" : {"Grvl" : 1, "Pave" : 2},
                  "Utilities" : {"ELO" : 1, "NoSeWa" : 2, "NoSewr" : 3, "AllPub" : 4}}
                )
```

라벨인코딩이 필요한 컬럼들의 값(value)들에 라벨을 지정해줌

5.3 로그변환

```
# 트레인 데이터에 로그변환한 컬럼과 같은 컬럼들만 로그 변환해준다.
house_test[skew_features_top.index[:-1]] = np.log1p(house_test[skew_features_top.index[:-1]])
```

트레인 데이터를 처리할 때 만들었던 skew_features_top 변수를 사용해서
왜도가 1 이상인 컬럼명들만 추출한 후 로그 변환해준다.

5.4 원핫인코딩

```
# 원핫 인코딩 확인
house_train_ohe
```

	Id	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	65.0	9.042040	2	0.0	4	0.693147
1	2	80.0	9.169623	2	0.0	4	0.693147
2	3	68.0	9.328212	2	0.0	3	0.693147
3	4	60.0	9.164401	2	0.0	3	0.693147
4	5	84.0	9.565284	2	0.0	3	0.693147
...	...	...	...	...	...	...	...
1455	1456	62.0	8.976894	2	0.0	4	0.693147
1456	1457	85.0	9.486152	2	0.0	4	0.693147
1457	1458	66.0	9.109746	2	0.0	4	0.693147
1458	1459	68.0	9.181735	2	0.0	4	0.693147
1459	1460	75.0	9.204121	2	0.0	4	0.693147

1460 rows × 239 columns

원핫인코딩 : 서열이 없는 것,
ex 국가명, 사람이름

MSSubClass, MSZoning, LotConfig,
Neighborhood, Condition1, Condition2,
BldgType, PavedDrive
HouseStyle, RoofStyle, RoofMatl,
Exterior1st, Exterior2nd, MasVnrType,
Foundation, GarageFinish
Heating, Electrical, TotRmsAbvGrd,
Functional, GarageType SaleType,
MiscFeature, Fence

5.5 트레인 데이터와 열 맞추기

```
house_test_ohe
```

	Id	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LandSlope	OverallQual
0	1461	80.0	9.360741	2	0.0	4	0.693147	3	5
1	1462	81.0	9.565775	2	0.0	3	0.693147	3	6
2	1463	74.0	9.534668	2	0.0	3	0.693147	3	5
3	1464	78.0	9.208238	2	0.0	3	0.693147	3	6
4	1465	43.0	8.518392	2	0.0	3	1.386294	3	8
...	...	...	...	...	...	...	...	...	...
1454	2915	21.0	7.568896	2	0.0	4	0.693147	3	4
1455	2916	21.0	7.546974	2	0.0	4	0.693147	3	4
1456	2917	160.0	9.903538	2	0.0	4	0.693147	3	5
1457	2918	62.0	9.253591	2	0.0	4	0.693147	3	5
1458	2919	74.0	9.172431	2	0.0	4	0.693147	2	7

1459 rows × 224 columns

```
# 타겟값을 분리하였으니 데이터 프레임에서 타겟값을 제거한다.
house_train_ohe.drop(['SalePrice'], axis = 1, inplace = True)
```

```
# for문을 이용하여 train데이터엔 있지만 test데이터엔 없는 컬럼명을 추출한다.
a = []
for i in list(house_train_ohe.columns):
    if i not in house_test_ohe.columns:
        a.append(i)
a
```

```
['Condition2_RRAe',
 'Condition2_RRAn',
 'Condition2_RRNn',
 'HouseStyle_2.5Fin',
 'RoofMatl_ClyTile',
 'RoofMatl_Membran',
 'RoofMatl_Metal',
 'RoofMatl_Roll',
 'Exterior1st_ImStucc',
 'Exterior1st_Stone',
 'Exterior2nd_Other',
 'Heating_Floor',
 'Heating_OthW',
 'Electrical_Mix',
 'MiscFeature_TenC']
```

```
# 위에서 추출된 컬럼들을 train데이터에서 제거해 준다.
house_train_ohe = house_train_ohe.drop(['Condition2_RRAe',
 'Condition2_RRAn',
 'Condition2_RRNn',
 'HouseStyle_2.5Fin',
 'RoofMatl_ClyTile',
 'RoofMatl_Membran',
 'RoofMatl_Metal',
 'RoofMatl_Roll',
 'Exterior1st_ImStucc',
 'Exterior1st_Stone',
 'Exterior2nd_Other',
 'Heating_Floor',
 'Heating_OthW',
 'Electrical_Mix',
 'MiscFeature_TenC'], axis = 1)
```

```
# 역으로 test데이터엔 존재하지만 train데이터에 존재하지 않는 컬럼명을 추출한다.
a = []
for i in list(house_test_ohe.columns):
    if i not in house_train_ohe.columns:
        a.append(i)
a
```

```
['MSSubClass_150']
```

```
# 위에서 추출된 컬럼을 test데이터에서 제거해 준다.
house_test_ohe.drop(['MSSubClass_150'], axis = 1, inplace = True)
```

test 데이터에 존재하지만 train 데이터에 존재하지 않는 컬럼명도 추출하여 제거해준다

```
house_test_ohe
```

1459 rows × 223 columns

```
# 열 갯수가 동일한것을 확인할 수 있다.
house_train_ohe
```

1460 rows × 223 columns

train 데이터와 test 데이터의 열 갯수가 같아진 것을 확인할 수 있다

6. 선형 회귀 모델 학습/예측/평가

```
from sklearn.linear_model import ElasticNetCV, LassoCV, RidgeCV
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
```

```
def get_rmse(model):
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test , pred)
    rmse = np.sqrt(mse)
    print('{0} 로그 변환된 RMSE: {1}'.format(model.__class__.__name__,np.round(rmse, 3)))
    return rmse

def get_rmses(models):
    rmses = [ ]
    for model in models:
        rmse = get_rmse(model)
        rmses.append(rmse)
    return rmses
```

- get_rmse(model)은 단일 모델의 RMSE 값을, get_rmse(models)는 get_rmse()를 이용해 여러 모델의 RMSE 값을 반환
- 이것을 토대로 선형 회귀 모델을 학습하고 예측, 평가 해보겠다.

```
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_train_ohe_target
X_features = house_train_ohe

X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.2, random_state=2022)

# LinearRegression, Ridge, Lasso 학습, 예측, 평가
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)

ridge_reg = Ridge()
ridge_reg.fit(X_train, y_train)

lasso_reg = Lasso()
lasso_reg.fit(X_train, y_train)

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

```
LinearRegression 로그 변환된 RMSE: 0.156
Ridge 로그 변환된 RMSE: 0.129
Lasso 로그 변환된 RMSE: 0.183

[0.15609547243171934, 0.12928361936540356, 0.18273117704485786]
```

```
house_train_org.iloc[:,:-1]  # 타겟땐거
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	

1460 rows × 80 columns

- 다중공선성의 문제로 robust 트리모델 사용하기로 결정

```
# 주어진 데이터로 인해 표본의 크기를 증가 할 수 없기 때문에 robust 한 트리모델을 사용하기로 결정
# 데이터에 RoubustScaler화 시키기
from sklearn.preprocessing import RobustScaler
rbst_scaler = RobustScaler()
X_rbst = rbst_scaler.fit_transform(house_train_ohe)
test_rbst = rbst_scaler.transform(house_test_ohe)
```

6.1 교차검증(K-folder)

```
kfold = KFold(n_splits = 4)
```

```
random_state = 1
reg = []

reg.append(Lasso(random_state = random_state))
reg.append(ElasticNet(random_state = random_state))
reg.append(RandomForestRegressor(random_state = random_state))
reg.append(GradientBoostingRegressor(random_state = random_state))
reg.append(XGBRegressor(silent = True, random_state = random_state))
reg.append(LGBMRegressor(verbose_eval = False, random_state = random_state))
```

```
reg_results = []

for regre in reg:
    reg_results.append(np.mean(np.sqrt(-cross_val_score(
        regre, X_rbst, y = house_train_ohe_target, scoring = 'neg_mean_squared_error', cv = kfold, n_jobs = -4))
```

```
reg_means = []
reg_std = []
for reg_result in reg_results:
    reg_means.append(reg_result.mean())
    reg_std.append(reg_result.std())
```

```
reg_re = pd.DataFrame({'CrossValMeans' : reg_means, 'CrossValerrors' : reg_std})
reg_re
```

	CrossValMeans	CrossValerrors
0	0.387013	0.0
1	0.378090	0.0
2	0.144581	0.0
3	0.126058	0.0
4	0.140093	0.0
5	0.131120	0.0

6.2 파라미터 튜닝 & 그리드 서치

- CrossValMeans 확인 결과 Gradient boosting, xgboost, lightgbm 모델에 대해 파라미터 튜닝

6.3 Gradient boosting 파라미터 튜닝

```
GBC = GradientBoostingRegressor()
gb_param_grid = {'n_estimators' : [100, 200, 300],
                 'learning_rate' : [0.1, 0.05, 0.01],
                 'max_depth' : [4, 8],
                 'min_samples_leaf' : [100, 150],
                 'max_features' : [0.3, 0.1]}
gsGBC = GridSearchCV(GBC, param_grid = gb_param_grid, cv = kfold, scoring = 'neg_mean_squared_error',
                     n_jobs = 4, verbose = 1)
gsGBC.fit(X_rbst, house_train_ohe_target)
GBC_best = gsGBC.best_estimator_

# 최고 점수
gsGBC.best_score_
```

```
Fitting 4 folds for each of 72 candidates, totalling 288 fits
```

```
-0.019166843602173293
```

6.4 XGBoost 파라미터 튜닝

```
XGB = XGBRegressor()
xgb_param_grid = {'learning_rate' : [1, 0.1, 0.01, 0.001],
                  'n_estimators' : [50, 100, 200, 500, 1000],
                  'max_depth' : [1, 3, 5, 10 ,50]}
gsXGB = GridSearchCV(XGB, param_grid = xgb_param_grid,
                     cv = kfold, scoring = 'neg_mean_squared_error', n_jobs = 4, verbose = 1)
gsXGB.fit(X_rbst, house_train_ohe_target)
XGB_best = gsXGB.best_estimator_

# 최고 점수
gsXGB.best_score_
```

```
Fitting 4 folds for each of 100 candidates, totalling 400 fits
```

```
-0.01595032267750892
```

6.5 LGBMClassifier 파라미터 튜닝

```
LGB = LGBMRegressor()
lgb_param_grid = {
    'num_leaves' : [1, 5, 10],
    'learning_rate' : [1, 0.1, 0.01, 0.001],
    'n_estimators' : [50, 100, 200, 500, 1000, 5000],
    'max_depth' : [15, 20, 25],
    'num_leaves' : [50, 100, 200],
    'min_split_gain' : [0.3, 0.4]
}
gsLGB = GridSearchCV(LGB, param_grid = lgb_param_grid, cv = kfold, scoring = 'neg_mean_squared_error',
                     n_jobs = 4, verbose = 1)
gsLGB.fit(X_rbst, house_train_ohe_target)
LGB_best = gsLGB.best_estimator_

# 최고 점수
gsLGB.best_score_

```

```
Fitting 4 folds for each of 432 candidates, totalling 1728 fits
```

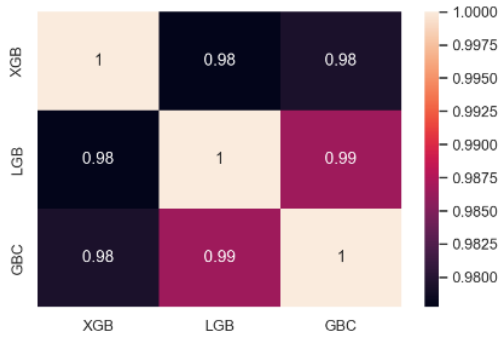
```
-0.020552351982487723
```

- 파라미터 튜닝을 한 것을 토대로 모델링 예측을 시작

7. 예측

7.1 앙상블

```
test_Survived_GBC = pd.Series(GBC_best.predict(test_rbst), name = 'GBC')
test_Survived_XGB = pd.Series(XGB_best.predict(test_rbst), name = 'XGB')
test_Survived_LGB = pd.Series(LGB_best.predict(test_rbst), name = 'LGB')

ensemble_results = pd.concat([test_Survived_XGB, test_Survived_LGB, test_Survived_GBC], axis = 1)
g = sns.heatmap(ensemble_results.corr(), annot = True)
```




```
ensemble = np.expm1(0.1 * test_Survived_GBC + 0.8 * test_Survived_XGB + 0.1 * test_Survived_LGB)
submission = pd.DataFrame({'Id' :house_test['Id'],
                           'SalePrice' : ensemble})
submission.head()
```

	Id	SalePrice
0	1461	122076.949816
1	1462	160211.679165
2	1463	188604.400512
3	1464	194841.483822
4	1465	192673.852878

7.2 보팅 ¶

```
from sklearn.ensemble import VotingRegressor
votingC = VotingRegressor(estimators=[('XGB', XGB_best), ('LGB', LGB_best), ('GBC',GBC_best)], n_jobs=4)
votingC = votingC.fit(X_rbst, house_train_ohe_target)
```

```
test_SalePrice = pd.Series(votingC.predict(test_rbst), name="SalePrice")
submission = pd.DataFrame({
    "Id" :house_test['Id'],
    "SalePrice": np.expm1(test_SalePrice)
})
submission.head()
```

	Id	SalePrice
0	1461	122910.971932
1	1462	160173.239941
2	1463	184185.482185
3	1464	192294.542394
4	1465	192231.826436

7.3 스태킹

```
params = {'meta_regressor__C': [0.1, 1.0, 10.0, 100.0],
          'use_features_in_secondary' : [True, False]}
```

```
clf1 = XGB_best
clf2 = LGB_best
clf3 = GBC_best

lr = LogisticRegression()
st_re= StackingRegressor(regressors=[clf1, clf2, clf3], meta_regressor=RandomForestRegressor())
st_mod = st_re.fit(X_rbst, house_train_ohe_target)
st_pred = st_mod.predict(test_rbst)
```

```
submission = pd.DataFrame({
    "Id" :house_test['Id'],
    "SalePrice": np.expm1(st_pred)
})
submission.head()
```

	Id	SalePrice
0	1461	123117.347195
1	1462	164256.147276
2	1463	184467.838764
3	1464	192474.914422
4	1465	192505.184498

- 이것을 토대로 blending 시작

7.4 블랜딩

```
kfolds = KFold(n_splits=10, shuffle=True, random_state=42)
def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))

def cv_rmse(model, X=house_train_ohe):
    rmse = np.sqrt(-cross_val_score(model, X_rbst, house_train_ohe_target,
                                    scoring="neg_mean_squared_error",
                                    cv=kfolds))
    return (rmse)
```

```
alphas_ridge = [14.5, 14.6, 14.7, 14.8, 14.9, 15, 15.1, 15.2, 15.3, 15.4, 15.5]
alphas_lasso = [5e-05, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008]
alphas_enect = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007]
enect_l1ratio = [0.8, 0.85, 0.9, 0.95, 0.99, 1]
```

```
ridge = make_pipeline(RidgeCV(alphas=alphas_ridge, cv=kfolds))
```

```
lasso = make_pipeline(LassoCV(alphas=alphas_lasso,
                              random_state=2, cv=kfolds))
```

```
enet = make_pipeline(ElasticNetCV(alphas=alphas_enect,
                                  cv=kfolds, l1_ratio=enect_l1ratio))
```

```
svr = make_pipeline(SVR(C= 20, epsilon= 0.008, gamma=0.0003,))
```

```
stack_gen = StackingCVRegressor(regressors=(ridge, lasso, enet,
                                            GBC_best, XGB_best, LGB_best),
                                meta_regressor=XGB_best,
                                use_features_in_secondary=True)
```

```
score = cv_rmse(ridge)
print("Kernel Ridge score: {:.4f} ({:.4f})\n".format(score.mean(),score.std()))

score = cv_rmse(lasso)
print("Lasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(enet)
print("ElasticNet score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(svr)
print("SVR score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(GBC_best)
print("Lightgbm score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(XGB_best)
print("GradientBoosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(LGB_best)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )
```

```
Kernel Ridge score: 0.1239 (0.0245)

Lasso score: 0.1233 (0.0254)

ElasticNet score: 0.1231 (0.0254)

SVR score: 0.1225 (0.0269)

Lightgbm score: 0.1332 (0.0197)

GradientBoosting score: 0.1222 (0.0215)

Xgboost score: 0.1420 (0.0230)
```

- 각 모델링과 규제 : 점수(표준편차) 순으로 결과 도출

```
stack_gen_model = stack_gen.fit(np.array(X_rbst), np.array(house_train_ohe_target))
elastic_model_full_data = enet.fit(X_rbst, house_train_ohe_target)
lasso_model_full_data = lasso.fit(X_rbst, house_train_ohe_target)
ridge_model_full_data = ridge.fit(X_rbst, house_train_ohe_target)
svr_model_full_data = svr.fit(X_rbst, house_train_ohe_target)
gbr_model_full_data = GBC_best.fit(X_rbst, house_train_ohe_target)
xgb_model_full_data = XGB_best.fit(X_rbst, house_train_ohe_target)
lgb_model_full_data = LGB_best.fit(X_rbst, house_train_ohe_target)
```

- 각 모델링에 대해 제일 잘 나온 점수 추출

```
def blend_models_predict(X):
    return ((0.1 * elastic_model_full_data.predict(X)) + \
            (0.1 * lasso_model_full_data.predict(X)) + \
            (0.1 * ridge_model_full_data.predict(X)) + \
            (0.1 * svr_model_full_data.predict(X)) + \
            (0.1 * gbr_model_full_data.predict(X)) + \
            (0.15 * xgb_model_full_data.predict(X)) + \
            (0.1 * lgb_model_full_data.predict(X)) + \
            (0.25 * stack_gen_model.predict(np.array(X))))
```

- 각 모델링에 대해 가중치를 주는 함수 blend_models_predict 생성
- 각 모델링에 대해 가중치를 0.1 ~ 0.25 까지 준다.

```
pred = np.floor(np.expm1(blend_models_predict(test_rbst)))
submission = pd.DataFrame({
    "Id" :house_test['Id'],
    "SalePrice": pred
})
submission.head()
```

	Id	SalePrice
0	1461	119057.0
1	1462	158794.0
2	1463	184892.0
3	1464	196707.0
4	1465	191924.0

- 가중치를 준 후 결과 도출

```
elastic_model_full_data.predict(test_rbst)
```

```
array([11.64753114, 11.97650932, 12.10631274, ..., 12.02667728,
       11.66906623, 12.31516353])
```

```
lasso_model_full_data.predict(test_rbst)
```

```
array([11.64707644, 11.97661567, 12.10685632, ..., 12.02675115,
       11.67017745, 12.31671867])
```

```
ridge_model_full_data.predict(test_rbst)
```

```
array([11.64379928, 11.98247207, 12.09774165, ..., 12.02649693,
       11.65518687, 12.30157197])
```

```
svr_model_full_data.predict(test_rbst)
```

```
array([11.67314208, 12.00123614, 12.15590377, ..., 12.03642958,
       11.75403973, 12.32699407])
```

```
gbr_model_full_data.predict(test_rbst)
```

```
array([11.70404314, 11.99056045, 12.15908744, ..., 12.0294436 ,
       11.66925522, 12.27391436])
```

```
xgb_model_full_data.predict(test_rbst)
```

```
array([11.713395, 11.985395, 12.153102, ..., 11.891494, 11.571796,
       12.255717], dtype=float32)
```

```
lgb_model_full_data.predict(test_rbst)
```

```
array([11.72146223, 11.96380881, 12.10896123, ..., 11.96861011,
       11.6713542 , 12.32627102])
```

-각 모델링의 개별 예측점수 확인

```
from catboost import CatBoostRegressor
```

```
model_CBR = CatBoostRegressor()
parameters = {'depth'         : [6,8,10],
                  'learning_rate' : [0.01, 0.05, 0.1],
                  'iterations'    : [30, 50, 100]
                 }
grid = GridSearchCV(estimator=model_CBR, param_grid = parameters, cv = 2, n_jobs=-1)
catboost_full_data = grid.fit(X_rbst, house_train_ohe_target)
```

```
0:      learn: 0.3728723        total: 139ms    remaining: 13.8s
1:      learn: 0.3518754        total: 142ms    remaining: 6.98s
2:      learn: 0.3310955        total: 145ms    remaining: 4.7s
3:      learn: 0.3099935        total: 149ms    remaining: 3.57s
4:      learn: 0.2917979        total: 152ms    remaining: 2.89s
5:      learn: 0.2748116        total: 156ms    remaining: 2.44s
6:      learn: 0.2613147        total: 159ms    remaining: 2.11s
7:      learn: 0.2484537        total: 162ms    remaining: 1.87s
8:      learn: 0.2364814        total: 166ms    remaining: 1.68s
9:      learn: 0.2261383        total: 170ms    remaining: 1.53s
10:     learn: 0.2164213        total: 174ms    remaining: 1.41s
11:     learn: 0.2082791        total: 177ms    remaining: 1.3s
12:     learn: 0.2001852        total: 180ms    remaining: 1.21s
13:     learn: 0.1934332        total: 183ms    remaining: 1.12s
14:     learn: 0.1872461        total: 187ms    remaining: 1.06s
15:     learn: 0.1814153        total: 191ms    remaining: 1s
16:     learn: 0.1758818        total: 194ms    remaining: 947ms
17:     learn: 0.1712085        total: 198ms    remaining: 900ms
18:     learn: 0.1667409        total: 201ms    remaining: 858ms
```

```
catboost_full_data.predict(test_rbst)
```

```
array([11.70350142, 11.95102766, 12.11257878, ..., 12.02086616,
       11.65328077, 12.3150755 ])
```

- 새로운 모델인 CatBoostRegressor 로 예측

- 각 모델링에 대해 가중치를 주는 함수 blend_models_predict 에 CatboostRegressor 의 예측을 포함시켜 blend_models_predict1 함수 생성

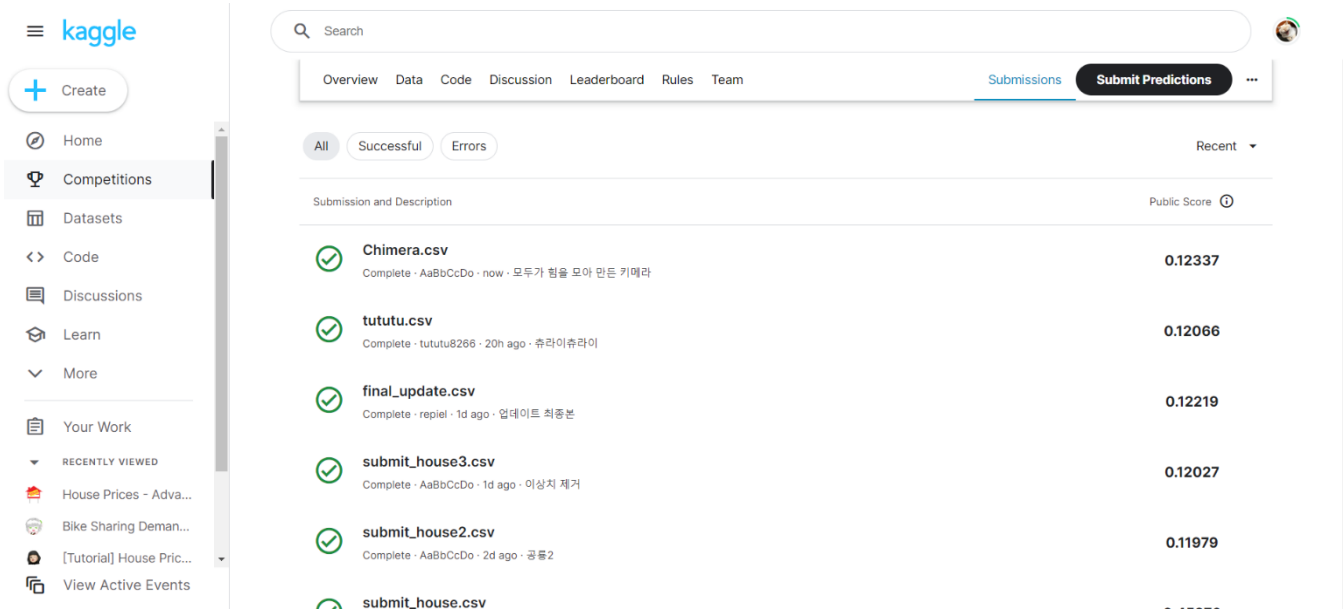
```
def blend_models_predict1(X):
    return ((0.1 * elastic_model_full_data.predict(X)) + \
            (0.1 * lasso_model_full_data.predict(X)) + \
            (0.1 * ridge_model_full_data.predict(X)) + \
            (0.1 * svr_model_full_data.predict(X)) + \
            (0.1 * gbr_model_full_data.predict(X)) + \
            (0.10 * xgb_model_full_data.predict(X)) + \
            (0.1 * lgb_model_full_data.predict(X)) + \
            (0.15 * stack_gen_model.predict(np.array(X)) + \
            (0.15 * catboost_full_data.predict(np.array(X))
            )))
```

```
pred = np.floor(np.expm1(blend_models_predict1(test_rbst)))
submission = pd.DataFrame({
    "Id" :house_test['Id'],
    "SalePrice": pred
})
submission.head()
```

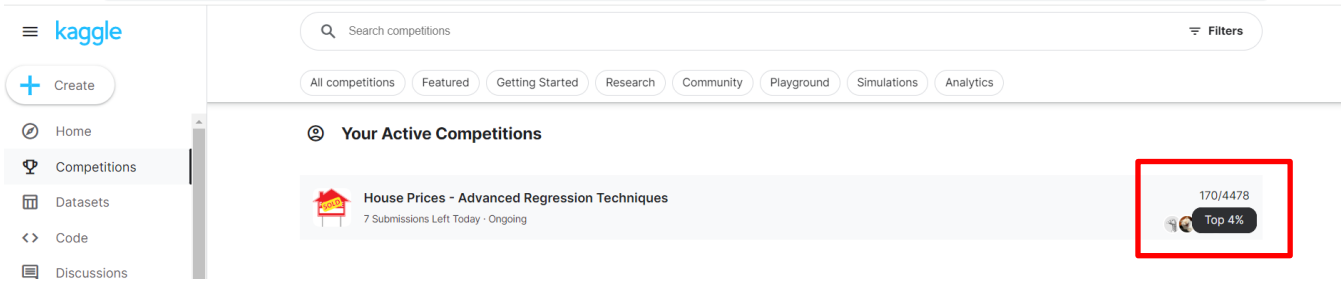
	Id	SalePrice
0	1461	118961.0
1	1462	158478.0
2	1463	184301.0
3	1464	195713.0
4	1465	192112.0

- 최종 예측값 도출

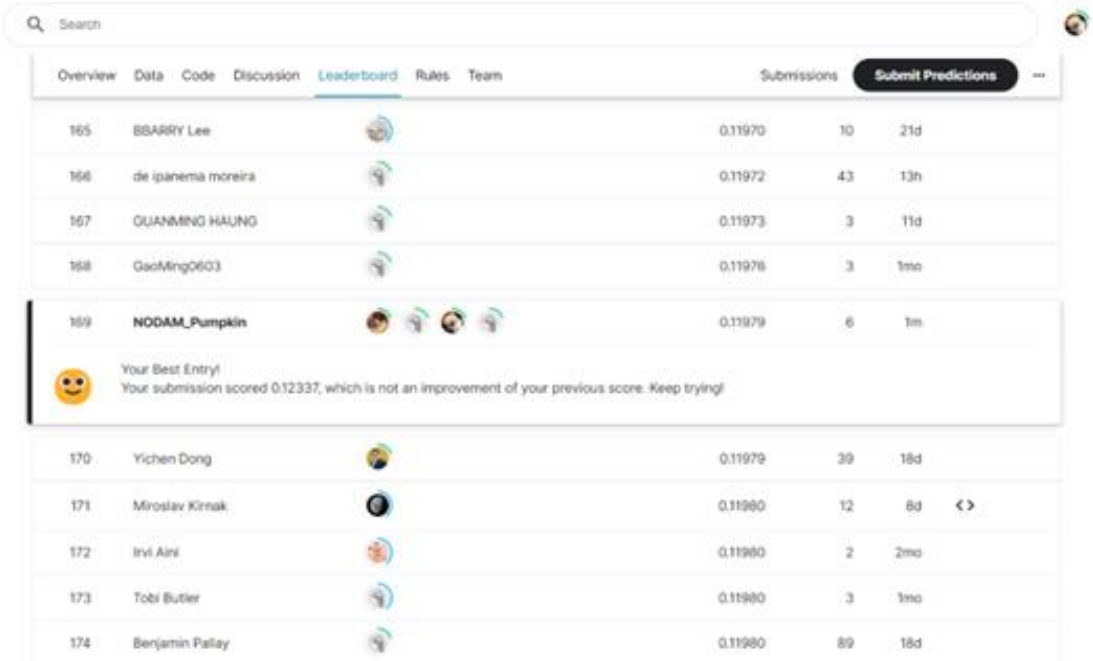
7.4 Kaggle 대회 진행 상황



결과물 제출 후 RMSE 환산 점수 목록



현재 총 참가팀 중 4%의 성적을 기록함



현재 참가팀 4495 중 169 위를 기록하고 있음