

House Prices - Advanced Regression Techniques

주택 판매 가격 예측 -> SalePrice를 예측

1. 데이터 불러오기

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# 드라이브 사용 설정
from google.colab import drive
drive.mount('/content/drive')
```

We already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
# 데이터 로드
```

```
train=pd.read_csv('/content/drive/MyDrive/house/house-prices-advanced-regression-techniques/train.csv')
test = pd.read_csv('/content/drive/MyDrive/house/house-prices-advanced-regression-techniques/test.csv')
submission = pd.read_csv('/content/drive/MyDrive/house/house-prices-advanced-regression-techniques/sample_submission.csv')
```

train: 행 1460, 열 81, test: 행 1459, 열 80

```
1 print(train.shape,test.shape)
```

(1460, 81) (1459, 80)

```
1 train.head()
   Id MSSubClass MSZoning LotFrontage LotArea Street Alley LotShape LandContour Utilities ... PoolArea PoolQC Fence MiscFeature MiscVal MoSold YrSold SaleType SaleCondition
0  1       60      RL     65.0    8450  Pave  NaN    Reg      Lvl AllPub ...        0  NaN  NaN  NaN  0  2  2008      WD  Normal
1  2       20      RL     80.0    9600  Pave  NaN    Reg      Lvl AllPub ...        0  NaN  NaN  NaN  0  5  2007      WD  Normal
2  3       60      RL     68.0   11250  Pave  NaN    IR1      Lvl AllPub ...        0  NaN  NaN  NaN  0  9  2008      WD  Normal
3  4       70      RL     60.0    9550  Pave  NaN    IR1      Lvl AllPub ...        0  NaN  NaN  NaN  0  2  2006      WD  Abnorml
4  5       60      RL     84.0   14260  Pave  NaN    IR1      Lvl AllPub ...        0  NaN  NaN  NaN  0  12 2008      WD  Normal
5 rows × 81 columns
```

```
1 test.tail()
   Id MSSubClass MSZoning LotFrontage LotArea Street Alley LotShape LandContour Utilities ... ScreenPorch PoolArea PoolQC Fence MiscFeature MiscVal MoSold YrSold Sale
1454 2915       160      RM     21.0    1936  Pave  NaN    Reg      Lvl AllPub ...        0        0  NaN  NaN  NaN  0  6  2006
1455 2916       160      RM     21.0    1894  Pave  NaN    Reg      Lvl AllPub ...        0        0  NaN  NaN  NaN  0  4  2006
1456 2917       20       RL    160.0   20000  Pave  NaN    Reg      Lvl AllPub ...        0        0  NaN  NaN  NaN  0  9  2006
1457 2918       85       RL     62.0   10441  Pave  NaN    Reg      Lvl AllPub ...        0        0  NaN  MnPrv  Shed  700  7  2006
1458 2919       60       RL     74.0    9627  Pave  NaN    Reg      Lvl AllPub ...        0        0  NaN  NaN  NaN  0  11 2006
5 rows × 80 columns
```

2. 데이터 전처리

id 열은 제거

```
train.drop(['Id'], axis=1, inplace=True)
test.drop(['Id'], axis=1, inplace=True)
```

```
1 train.describe()

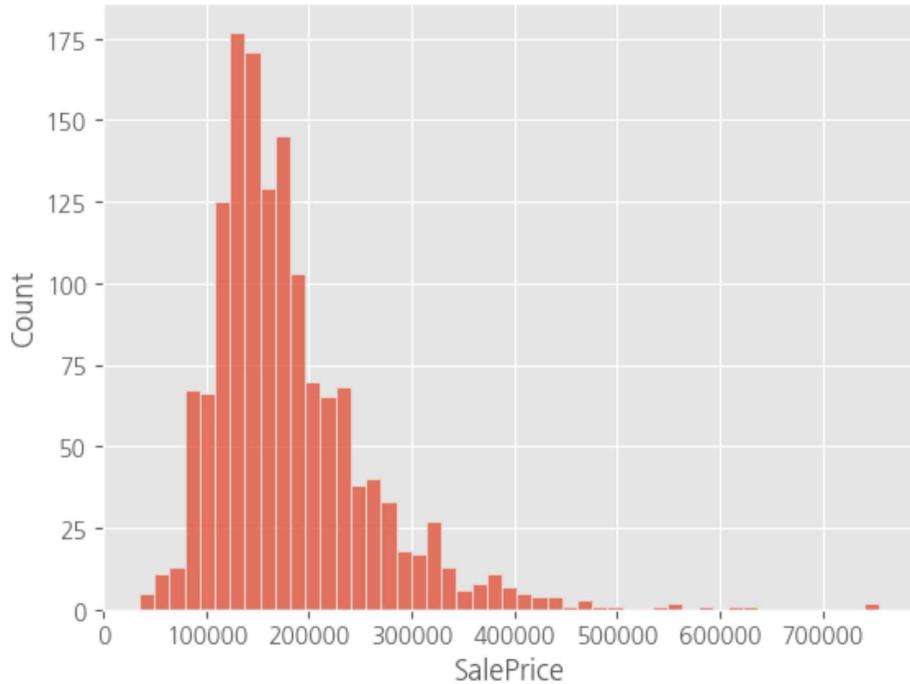
   MSSubClass LotFrontage LotArea OverallQual OverallCond YearBuilt YearRemodAdd MasVnrArea BsmtFinSF1 BsmtFinSF2 ... WoodDeckSF OpenPorchSF EnclosedPorch 3SsnPorch
count 1460.000000 1201.000000 1460.000000 1460.000000 1460.000000 1460.000000 1460.000000 1452.000000 1460.000000 1460.000000 ... 1460.000000 1460.000000 1460.000000
mean 56.897260 70.049958 10516.828082 6.099315 5.575342 1971.267808 1984.865753 103.685262 443.639726 46.549315 ... 94.244521 46.660274 21.954110 3.409589
std 42.300571 24.284752 9981.264932 1.382997 1.112799 30.202904 20.645407 181.066207 456.098091 161.319273 ... 125.338794 66.256028 61.119149 29.317331
min 20.000000 21.000000 1300.000000 1.000000 1.000000 1872.000000 1950.000000 0.000000 0.000000 0.000000 ... 0.000000 0.000000 0.000000 0.000000
25% 20.000000 59.000000 7553.500000 5.000000 5.000000 1954.000000 1967.000000 0.000000 0.000000 0.000000 ... 0.000000 0.000000 0.000000 0.000000
50% 50.000000 69.000000 9478.500000 6.000000 5.000000 1973.000000 1994.000000 0.000000 383.500000 0.000000 ... 0.000000 25.000000 0.000000 0.000000
75% 70.000000 80.000000 11601.500000 7.000000 6.000000 2000.000000 2004.000000 166.000000 712.250000 0.000000 ... 168.000000 68.000000 0.000000 0.000000
max 190.000000 313.000000 215245.000000 10.000000 9.000000 2010.000000 2010.000000 1600.000000 5644.000000 1474.000000 ... 857.000000 547.000000 552.000000 508.000000
8 rows x 37 columns
```

2.1-1 왜도

- 원쪽으로 치우쳐있으며 오른쪽으로 긴 꼬리를 갖는다.

```
1 sns.histplot(train['SalePrice'])
```

```
<Axes: xlabel='SalePrice', ylabel='Count'>
```



```
1 # 왜도 값
2
3 train['SalePrice'].skew()
```

```
1.8828757597682129
```

2.1-2 로그 변환

부정적인 영향을 줄이기 위해 로그 변환하여 데이터의 범위를 줄이고, 이상치의 영향을 줄이며, 데이터의 분포를 더 정규 분포 형태로 만든다

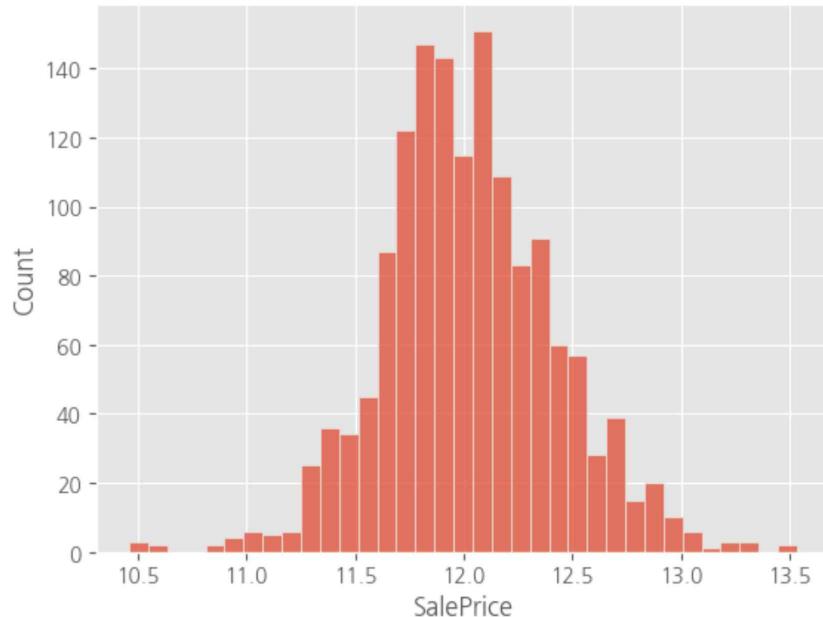
결과: 왜도가 1.88에서 0.12로 0에 가까운 왜도를 갖는다

```
[174] 1 y = np.log1p(train['SalePrice'])  
2 y.skew()
```

→ 0.12134661989685333

```
[175] 1 sns.histplot(y)
```

→ <Axes: xlabel='SalePrice', ylabel='Count'>



```
[176] 1 train["SalePrice"] = np.log1p(train['SalePrice'])
```

2.2 결측치 제거

train, test를 합쳐서 결측치를 제거하여 일관성 유지하기

```
1 train.isnull().sum()
```

```
MSSubClass      0
MSZoning        0
LotFrontage     259
LotArea          0
Street           0
...
MoSold           0
YrSold           0
SaleType          0
SaleCondition     0
SalePrice         0
Length: 80, dtype: int64
```

```
1 train_features = train.drop(['SalePrice'], axis=1)
```

```
1 features = pd.concat([train_features, test], ignore_index=True)
```

```
1 features
   MSSubClass MSZoning LotFrontage LotArea Street Alley LotShape LandContour Utilities LotConfig ...
0       60       RL       65.0    8450  Pave   NaN    Reg      Lvl    AllPub  Inside   ...
1       20       RL       80.0    9600  Pave   NaN    Reg      Lvl    AllPub  FR2     ...
2       60       RL       68.0   11250  Pave   NaN   IR1      Lvl    AllPub  Inside   ...
3       70       RL       60.0    9550  Pave   NaN   IR1      Lvl    AllPub  Corner   ...
4       60       RL      84.0   14260  Pave   NaN   IR1      Lvl    AllPub  FR2     ...
...
2914    160      RM      21.0   1936  Pave   NaN    Reg      Lvl    AllPub  Inside   ...
2915    160      RM      21.0   1894  Pave   NaN    Reg      Lvl    AllPub  Inside   ...
2916    20       RL     160.0   20000  Pave   NaN    Reg      Lvl    AllPub  Inside   ...
2917    85       RL      62.0   10441  Pave   NaN    Reg      Lvl    AllPub  Inside   ...
2918    60       RL      74.0   9627  Pave   NaN    Reg      Lvl    AllPub  Inside   ...
2919 rows x 79 columns
```

```
1 features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2919 entries, 0 to 2918
Data columns (total 79 columns):
 #   Column      Non-Null Count  Dtype  
 0   MSSubClass    2919 non-null   int64  
 1   MSZoning     2915 non-null   object 
 2   LotFrontage   2433 non-null   float64 
 3   LotArea       2919 non-null   int64  
 4   Street        2919 non-null   object 
 5   Alley          198 non-null   object 
 6   LotShape       2919 non-null   object 
 7   LandContour   2919 non-null   object 
 8   Utilities     2917 non-null   object 
 9   LotConfig     2919 non-null   object 
 10  LandSlope     2919 non-null   object 
 11  Neighborhood  2919 non-null   object 
 12  Condition1    2919 non-null   object 
 13  Condition2    2919 non-null   object 
 14  BldgType      2919 non-null   object 
 15  HouseStyle    2919 non-null   object 
 16  OverallQual   2919 non-null   int64  
 17  OverallCond   2919 non-null   int64  
 18  YearBuilt     2919 non-null   int64  
 19  YearRemodAdd  2919 non-null   int64  
 20  RoofStyle     2919 non-null   object 
 21  RoofMatl      2919 non-null   object 
 22  Exterior1st   2918 non-null   object 
 23  Exterior2nd   2918 non-null   object 
 24  MasVnrType    1153 non-null   object 
 25  MasVnrArea    2896 non-null   float64 
 26  ExterQual     2919 non-null   object 
 27  ExterCond     2919 non-null   object
```

범주 중 숫자형으로 되어있는 변수는 문자형으로 바꿔준다.

MSSubClass: 건물 클래스, MoSold: 판매월, YrSold: 판매년도

```
features['MSSubClass'] = features['MSSubClass'].apply(str)
features['YrSold'] = features['YrSold'].astype(str)
features['MoSold'] = features['MoSold'].astype(str)
```

2.2-1 MSZoning(일반적인 구역 분류)의 결측치 처리

범주형인 MSSubClass(건물 클래스) 별로 그룹화하여 그룹 별 MSZoning의 최빈값이 결측치를 대신한다.

```
1 features[features['MSZoning'].isnull()]

   MSSubClass MSZoning LotFrontage LotArea Street Alley LotShape LandContour Utilities LotConfig ... ScreenPorch PoolArea PoolQC Fence
1915      30     NaN       109.0    21780    Grvl     NaN     Reg        Lvl      NaN    Inside    ...      0       0     NaN     NaN
2216      20     NaN       80.0     14584    Pave     NaN     Reg        Low     AllPub   Inside    ...      0       0     NaN     NaN
2250      70     NaN      56600    56600    Pave     NaN    IR1        Low     AllPub   Inside    ...      0       0     NaN     NaN
2904      20     NaN      125.0    31250    Pave     NaN     Reg        Lvl     AllPub   Inside    ...      0       0     NaN     NaN
4 rows x 79 columns
```



```
1 features['MSZoning'] = features.groupby('MSSubClass')['MSZoning'].transform(lambda x: x.fillna(x.mode()[0]))
```

2.2-2 LotFrontage(부동산과 연결된 거리의 선형 피트) 결측치 처리

범주형인 Neighborhood(Ames시 경계 내의 물리적 위치) 별로 그룹화하여 그룹 별 LotFrontage의 중앙값이 결측치를 대신한다.

```
LotFrontage
[185] 1 features['LotFrontage'] = features.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.fillna(x.median()))
```

2.2-3 Alley(골목 접근 방식) 결측치 처리

```

1 features.dropna(subset=['Alley'])

   MSSubClass MSZoning LotFrontage LotArea Street Alley LotShape LandContour Utilities LotConfig
21        45      RM     57.0    7449    Pave    Grvl     Reg      Bnk    AllPub    Inside
30        70  C (all)    50.0    8500    Pave    Pave     Reg      Lvl    AllPub    Inside
56       160      FV    24.0    2645    Pave    Pave     Reg      Lvl    AllPub    Inside
79        50      RM    60.0   10440    Pave    Grvl     Reg      Lvl    AllPub    Corner
87       160      FV    40.0    3951    Pave    Pave     Reg      Lvl    AllPub    Corner
...
2790      20      RM    65.0    9750    Pave    Grvl     Reg      Lvl    AllPub    Inside
2795      30      RL    50.0   11672    Pave    Pave    IR2      Lvl    AllPub    Inside
2797      50      RM    60.0   10320    Pave    Grvl     Reg      Lvl    AllPub    Inside
2870      50      RL    45.0    8248    Pave    Grvl     Reg      Lvl    AllPub    Inside
2871      30      RL    60.0    8088    Pave    Grvl     Reg      Lvl    AllPub    Inside

```

198 rows × 79 columns

```

1 from scipy.stats import chi2_contingency
2
3 table = pd.crosstab(features['Alley'], features['LotShape'])
4
5 # 카이제곱 검정 수행
6 chi2, p, dof, expected = chi2_contingency(table)
7
8 print(f"Chi2: {chi2}, p-value: {p}")

```

Chi2: 23.680753784368243, p-value: 2.912228021021979e-05

범주형 변수를 카이제곱 검정해본 결과, p값이 0.05보다 작으므로 연관성이 있다고 판단

```

1 features.groupby('LotShape')['Alley'].value_counts()

LotShape Alley
IR1    Pave    20
         Grvl     7
IR2    Pave     4
IR3    Grvl     1
Reg    Grvl   112
         Pave    54
Name: count, dtype: int64

```

```

1 features['Alley'] = features.groupby('LotShape')['Alley'].transform(lambda x: x.fillna(x.mode()[0]))

```

범주형인 LotShape(부동산의 일반적인 형태) 별로 그룹화하여 그룹 별 Alley의 중앙 값이 결측치를 대신한다.

2.2-4 결측값이 별로없는 범주형의 결측치는 최빈값으로 대체

```

1 features['Utilities'].value_counts()

Utilities
AllPub    2916
NoSewr     1
Name: count, dtype: int64

1 d = ['Utilities', 'Exterior1st', 'Exterior2nd', 'KitchenQual', 'Functional', 'SaleType', 'Electrical']
2
3 for i in d:
4     features[i].fillna(features[i].mode()[0], inplace=True)

```

2.2-5 나머지 결측값은 숫자면 0, 문자면 'NA'로 채우기

이유: 나머지 결측값의 행은 대부분 다른 열도 nan이나 0값이 많기 때문에 이렇게 처리

```

1 features[['BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath']].isnull()

```

	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	BsmtFullBath	BsmtHalfBath
2120	NaN	NaN	NaN	NaN	NaN	NaN
2188	0.0	0.0	0.0	0.0	NaN	NaN


```

1 features[['GarageCars', 'GarageArea', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageQual', 'GarageCond']].isnull()

```

	GarageCars	GarageArea	GarageType	GarageYrBlt	GarageFinish	GarageQual	GarageCond
39	0.0	0.0	NaN	NaN	NaN	NaN	NaN
48	0.0	0.0	NaN	NaN	NaN	NaN	NaN
78	0.0	0.0	NaN	NaN	NaN	NaN	NaN
88	0.0	0.0	NaN	NaN	NaN	NaN	NaN
89	0.0	0.0	NaN	NaN	NaN	NaN	NaN
...
2893	0.0	0.0	NaN	NaN	NaN	NaN	NaN
2909	0.0	0.0	NaN	NaN	NaN	NaN	NaN
2913	0.0	0.0	NaN	NaN	NaN	NaN	NaN
2914	0.0	0.0	NaN	NaN	NaN	NaN	NaN
2917	0.0	0.0	NaN	NaN	NaN	NaN	NaN

159 rows × 7 columns

숫자형 결측값 0으로 처리, 범주형 결측값 NA로 처리하기

```

for i in features.columns:
    if features[i].dtype == object:
        features[i].fillna('NA', inplace=True)
    else:
        features[i].fillna(0, inplace=True)

```

모든 결측값 처리 완료

```
1 features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2919 entries, 0 to 2918
Data columns (total 79 columns):
 #   Column            Non-Null Count  Dtype  
 ---  -- 
 0   MSSubClass        2919 non-null   object 
 1   MSZoning          2919 non-null   object 
 2   LotFrontage       2919 non-null   float64
 3   LotArea           2919 non-null   int64  
 4   Street            2919 non-null   object 
 5   Alley              2919 non-null   object 
 6   LotShape          2919 non-null   object 
 7   LandContour       2919 non-null   object 
 8   Utilities         2919 non-null   object 
 9   LotConfig          2919 non-null   object 
 10  LandSlope          2919 non-null   object 
 11  Neighborhood       2919 non-null   object 
 12  Condition1        2919 non-null   object 
 13  Condition2        2919 non-null   object 
 14  BldgType          2919 non-null   object 
 15  HouseStyle         2919 non-null   object 
 16  OverallQual       2919 non-null   int64  
 17  OverallCond       2919 non-null   int64  
 18  YearBuilt         2919 non-null   int64  
 19  YearRemodAdd      2919 non-null   int64  
 20  RoofStyle          2919 non-null   object 
 21  RoofMatl          2919 non-null   object 
 22  Exterior1st        2919 non-null   object 
 23  Exterior2nd        2919 non-null   object 
 24  MasVnrType         2919 non-null   object 
 25  MasVnrArea         2919 non-null   float64
 26  ExterQual          2919 non-null   object 
 27  ExterCond          2919 non-null   object 
 28  Foundation         2919 non-null   object 
 29  BsmtQual          2919 non-null   object 
 30  BsmtCond          2919 non-null   object 
 31  BsmtExposure      2919 non-null   object 
 32  BsmtFinType1      2919 non-null   object 
 33  BsmtFinSF1         2919 non-null   float64
```

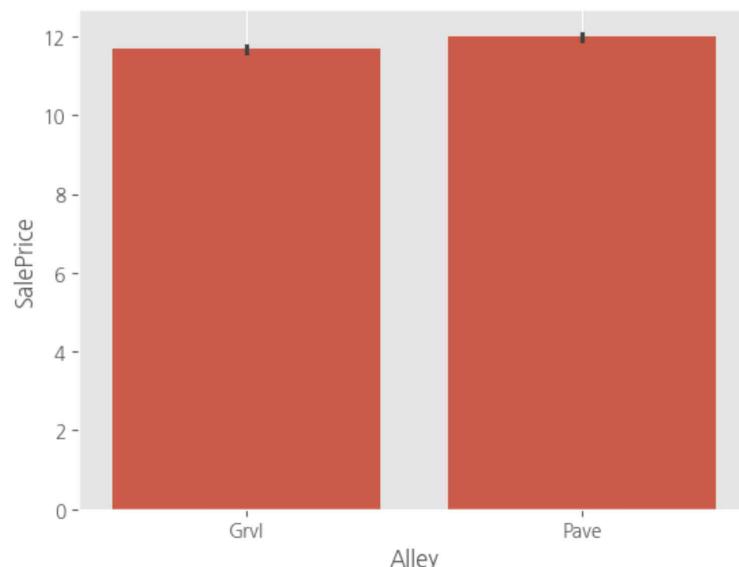
3. 피쳐엔지니어링

Anova를 이용해 범주형 데이터를 여러 그룹 간의 평균 차이를 비교하여 독립 변수가 종속 변수에 영향을 미치는지를 검정

p값이 0.05보다 작으면 유의미하다고 판단 -> 0.05보다 크면 제거

Alley(범주형)가 SalePrice에 유의미한 영향을 미친다.

```
1 import scipy.stats as spst  
  
1 sns.barplot(x="Alley", y="SalePrice", data=train)  
2 plt.grid()  
3 plt.show()
```



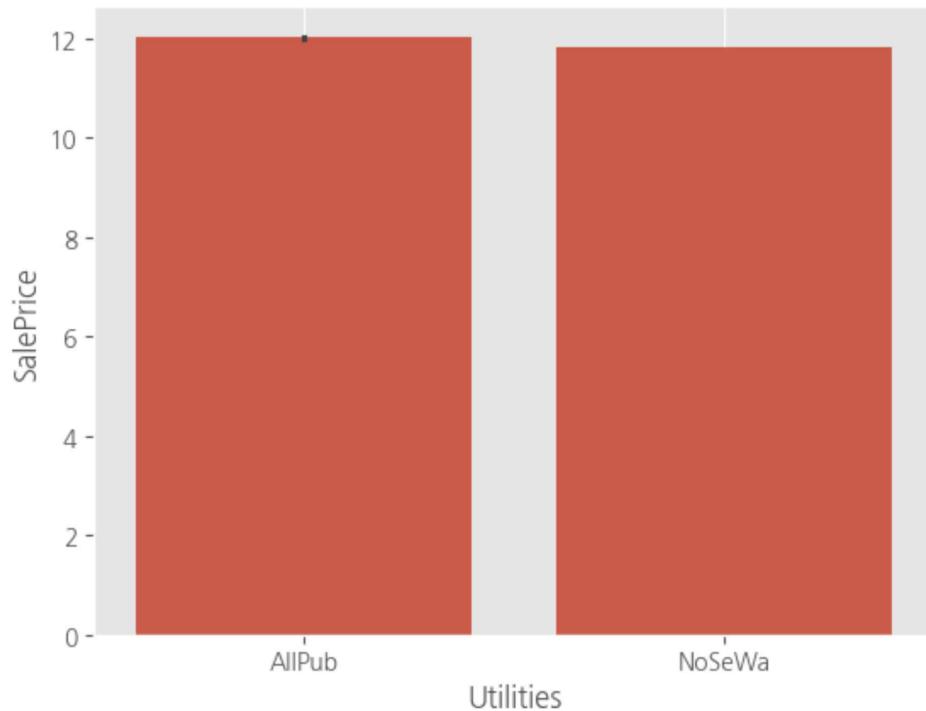
```
1 temp = train.loc[train['SalePrice'].notnull()]  
2 # 그룹별 저장  
3 P_1 = temp.loc[temp.Alley == 'Grvl', 'SalePrice']  
4 P_2 = temp.loc[temp.Alley == 'Pave', 'SalePrice']
```

```
1 spst.f_oneway(P_1, P_2)  
2  
3 # 유의미하다
```

```
F_onewayResult(statistic=27.007530111278424, pvalue=1.2770679752740598e-06)
```

Utilities(범주형)가 SalePrice에 유의미하지 않은 영향을 미친다.

```
1 sns.barplot(x="Utilities", y="SalePrice", data=train)
2 plt.grid()
3 plt.show()
```

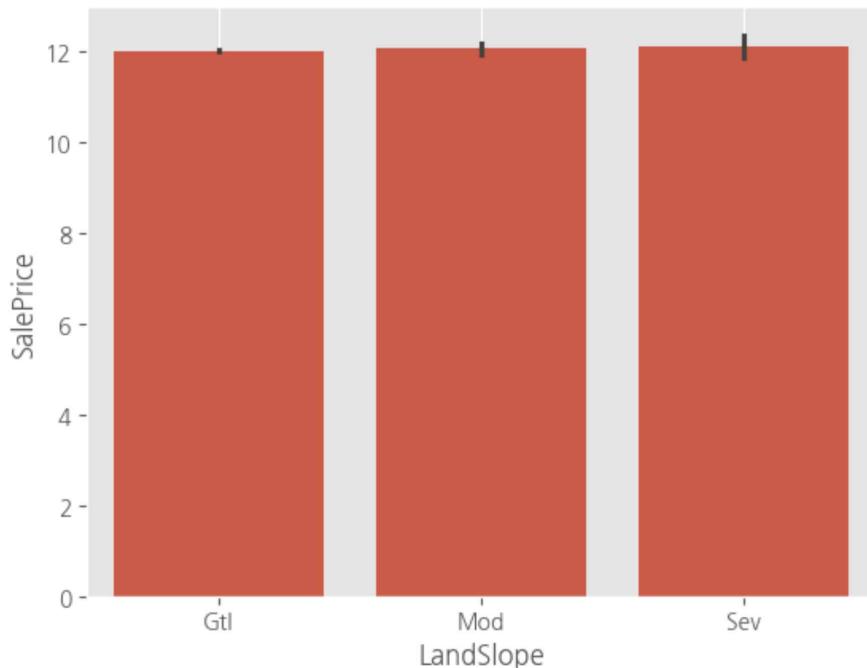


```
1 temp = train.loc[train['SalePrice'].notnull()]
2 # 그룹별 저장
3 P_1 = temp.loc[temp.Utilities == 'AllPub', 'SalePrice']
4 P_2 = temp.loc[temp.Utilities == 'NoSeWa', 'SalePrice']
5
6 spst.f_oneway(P_1, P_2)
7
8 # 유의미하지 않다 - 제거
```

F_onewayResult(statistic=0.23269025789394468, pvalue=0.6296085870536472)

LandSlope(범주형)가 SalePrice에 유의미하지 않은 영향을 미친다.

```
1 sns.barplot(x="LandSlope", y="SalePrice", data=train)
2 plt.grid()
3 plt.show()
```



```
1 temp = train.loc[train['SalePrice'].notnull()]
2 # 그룹별 저장
3 P_1 = temp.loc[temp.LandSlope == 'Gtl', 'SalePrice']
4 P_2 = temp.loc[temp.LandSlope == 'Mod', 'SalePrice']
5 P_3 = temp.loc[temp.LandSlope == 'Sev', 'SalePrice']
6
7 spst.f_oneway(P_1, P_2, P_3)
8
9 # 유의미하지 않다 - 제거
```

F_onewayResult(statistic=1.0830496097902718, pvalue=0.33883387690662126)

3.1 특정 범주형 변수 제거

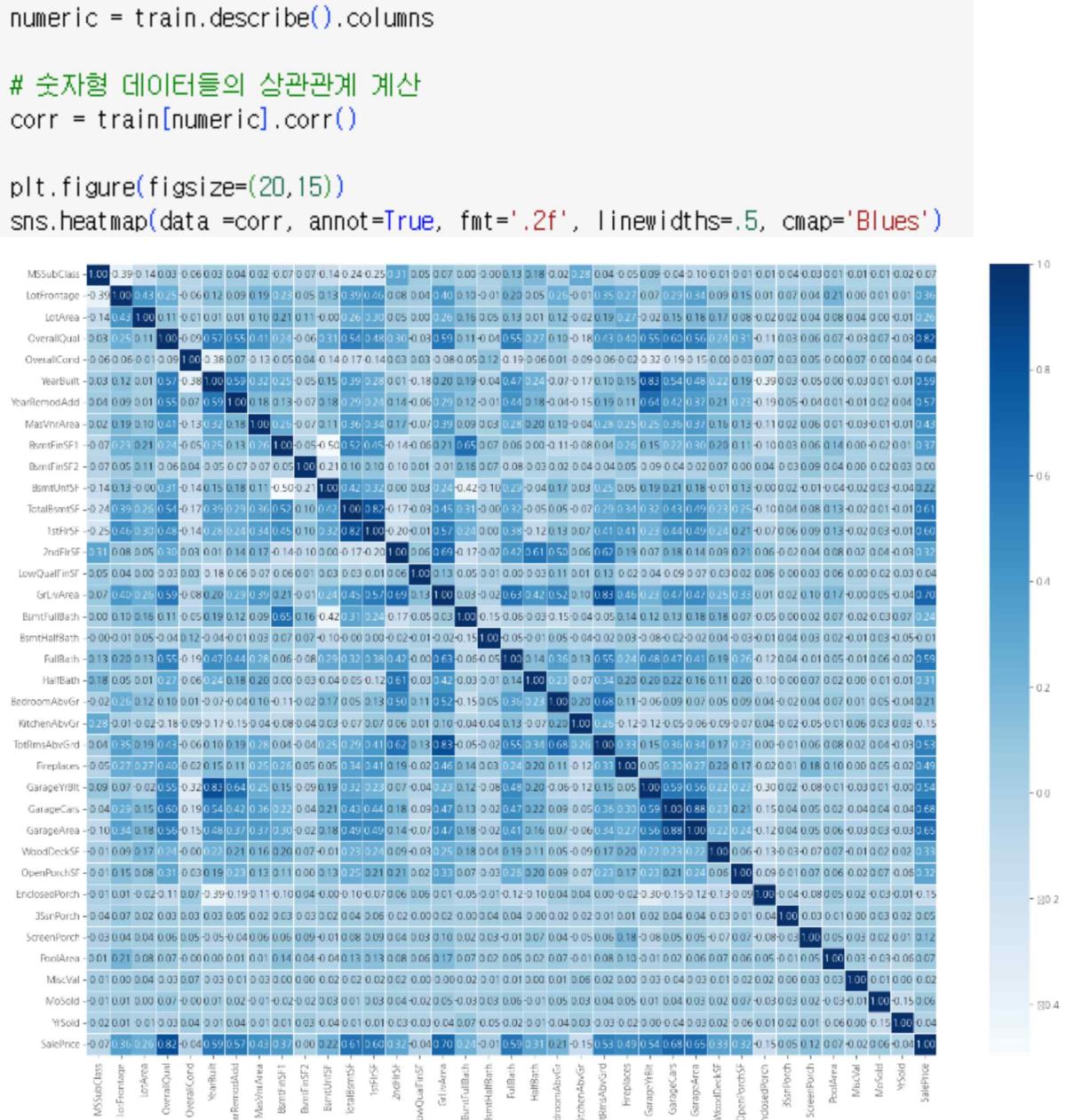
위의 그래프 결과로 유의미하지 않은 변수와 nan값이 많은 변수 제거

```
71 PoolQC      10 non-null    object
73 MiscFeature  105 non-null   object
```

PoolQC, MiscFeature은 nan값이 많으므로 제거

```
features = features.drop(['Utilities', 'Street', 'LandSlope', 'PoolQC', 'MiscFeature'], axis=1)
```

3.2 파생변수 생성



1: Very Poor ~ 10: Very Excellent

```
[ ] 1 features[['OverallQual', 'OverallCond']]
```

	OverallQual	OverallCond
0	7	5
1	6	8
2	7	5
3	7	5
4	8	5
...
2914	4	7
2915	4	5
2916	5	7
2917	5	5
2918	7	5

2919 rows × 2 columns

```
[ ] 1 features['Overall'] = features['OverallQual'] * features['OverallCond']
```

2) 파생변수 : 전체 피트(지하 + 1층 + 2층 평방 피트)

```
1 features[['TotalBsmtSF', '1stFlrSF', '2ndFlrSF']]
```

	TotalBsmtSF	1stFlrSF	2ndFlrSF
0	856.0	856	854
1	1262.0	1262	0
2	920.0	920	866
3	756.0	961	756
4	1145.0	1145	1053
...
2914	546.0	546	546
2915	546.0	546	546
2916	1224.0	1224	0
2917	912.0	970	0
2918	996.0	996	1004

2919 rows × 3 columns

```
1 features['TotalSF'] = features['TotalBsmtSF']+features['1stFlrSF']+features['2ndFlrSF']
```

3) 파생변수 : 욕실의 총 개수

```
1 features[['BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath']]
```

	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	
0	1.0	0.0	2	1	
1	0.0	1.0	2	0	
2	1.0	0.0	2	1	
3	1.0	0.0	1	0	
4	1.0	0.0	2	1	
...
2914	0.0	0.0	1	1	
2915	0.0	0.0	1	1	
2916	1.0	0.0	1	0	
2917	0.0	1.0	1	0	
2918	0.0	0.0	2	1	

2919 rows × 4 columns

```
1 features['Bath'] = features['BsmtFullBath']+features['BsmtHalfBath']+features['FullBath']+features['HalfBath']
```

4) 파생변수 : 침실과 주방을 제외한 방의 총 개수

```
1 features[['BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd']]
```

	BedroomAbvGr	KitchenAbvGr	TotRmsAbvGrd	
0	3	1	8	
1	3	1	6	
2	3	1	6	
3	3	1	7	
4	4	1	9	
...
2914	3	1	5	
2915	3	1	6	
2916	4	1	7	
2917	3	1	6	
2918	3	1	9	

2919 rows × 3 columns

```
1 features['Room'] = features['TotRmsAbvGrd'] - features['KitchenAbvGr'] - features['BedroomAbvGr']
```

5) 파생변수 : 현관의 총 면적

```
[1]: features[['WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch']]
```

	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	grid icon
0	0	61	0	0	0	bar icon
1	298	0	0	0	0	
2	0	42	0	0	0	
3	0	35	272	0	0	
4	192	84	0	0	0	
...	
2914	0	0	0	0	0	
2915	0	24	0	0	0	
2916	474	0	0	0	0	
2917	80	32	0	0	0	
2918	190	48	0	0	0	

2919 rows × 5 columns

```
[1]: features['Total_Porch'] = features['WoodDeckSF'] + features['OpenPorchSF'] + features['EnclosedPorch'] + features['3SsnPorch'] + features['ScreenPorch']
```

6) 건축한 연도와 리모델링 연도 파생변수

파생변수1 : 리모델링 연도와 건축한 연도의 차이(몇 년 만에 리모델링하였는지, 0이면 리모델링X)

파생변수2 : 리모델링 연도와 건축한 연도의 합(값이 클수록 최근 주택)

```
[ ] 1 features[['YearBuilt', 'YearRemodAdd']]
```

	YearBuilt	YearRemodAdd	grid icon
0	2003	2003	bar icon
1	1976	1976	
2	2001	2002	
3	1915	1970	
4	2000	2000	
...	
2914	1970	1970	
2915	1970	1970	
2916	1960	1996	
2917	1992	1992	
2918	1993	1994	

2919 rows × 2 columns

```
[ ] 1 features['YearDiffer'] = features['YearRemodAdd'] - features['YearBuilt']
[ ] 2 features['YearSum'] = features['YearRemodAdd'] + features['YearBuilt']
```

3.3 숫자형 데이터 제거

(순서형이거나 숫자이지만 문자형인 데이터는 제외)

상관계수가 0.1이하는 관계가 거의 없다고 보기 때문에 상관계수 0.05이하는 제거

```
features = features.drop(['3SsnPorch', 'MiscVal', 'BsmtFinSF2', 'BsmtFullBath', 'LowQualFinSF'], axis=1)
```

3.4 더미 변수

범주형 변수를 이진형(0과 1) 변수로 변환

다중공선성 방지를 위해 첫 번째 열은 제외

```
features = pd.get_dummies(features, drop_first=True)
```

3.5 데이터 분리 및 이상치 제거

```
train = pd.concat([features.iloc[:len(train)], train[['SalePrice']]], axis=1)
test = features.iloc[len(train):]
```

박스플롯으로 독립 변수 별 이상치 확인

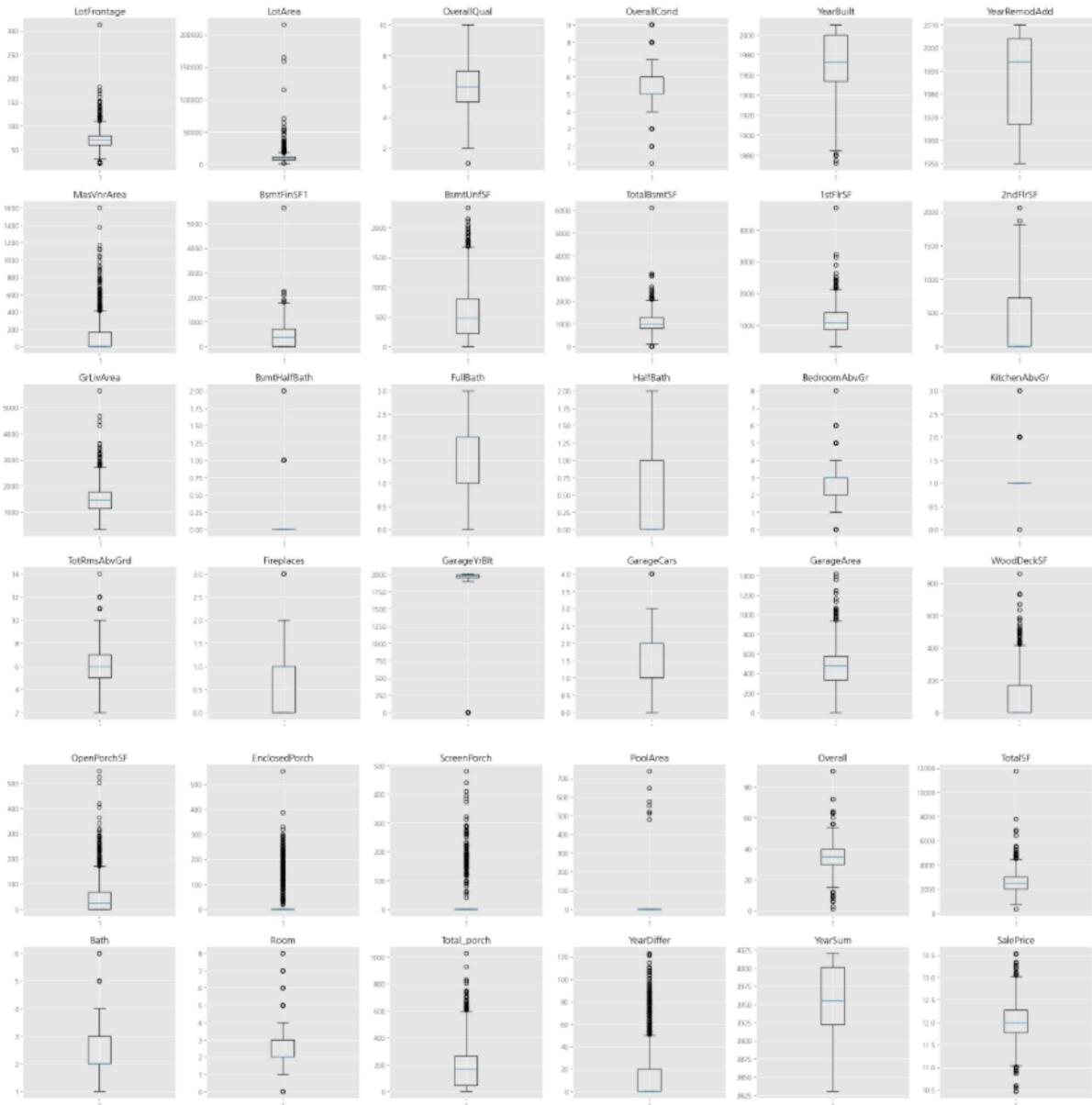
```
plt.style.use("ggplot")

# 숫자형
feature = train.describe().columns

plt.figure(figsize=(30,30))
plt.suptitle("독립변수별 이상치 확인", fontsize = 30)

for i in range(len(feature)):
    plt.subplot(6,6,i+1)
    plt.title(feature[i])
    plt.boxplot(train[feature[i]])
plt.show()
```

독립변수별 이상치 확인



```

1 train.describe().columns

Index(['LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
       'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF',
       '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', 'ScreenPorch', 'PoolArea', 'Overall',
       'TotalSF', 'Bath', 'Room', 'Total_porch', 'YearDiffr', 'YearSum',
       'SalePrice'],
      dtype='object')

```

3.5-1 IQR로 이상치 확인

박스플롯을 토대로 IQR을 계산하여 이상치를 확인하였지만 발견X

```

k = train.describe().columns

for i in k:
    # IQR 계산
    Q1 = np.percentile(train[i], 25)
    Q3 = np.percentile(train[i], 75)
    IQR = Q3 - Q1

    # 이상치 기준
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)

    if (len(train[i][(train[i] < lower_bound) & (train[i] > upper_bound)])>0):
        print(i)
        print(train[i][(train[i] < lower_bound) & (train[i] > upper_bound)])

    # 이상치 없음

```

3.5-2 과적합을 유발할 수 있는 열 제거

값의 빈도가 전체 데이터의 99%를 초과하는 데이터 열 뽑아내기

```

1 overfit = []
2 for i in train.columns:
3     counts = train[i].value_counts()
4     zeros = counts.iloc[0]
5     if zeros / len(train)*100 > 99: # 값의 빈도가 전체 데이터의 99%를 초과
6         overfit.append(i)
7
8 print(overfit)

['PoolArea', 'MSSubClass_150', 'MSSubClass_180', 'MSSubClass_40', 'MSSubClass_45', 'LotShape_IR3', 'LotConfig_FR3', 'Neighborhood_Blueste', 'Neighborhood_NPKWII',
train = train.drop(overfit, axis=1)
test = test.drop(overfit, axis=1)

```

3.6 스케일링

```
1 from sklearn.preprocessing import MinMaxScaler
2
3 X = train.drop('SalePrice', axis=1)
4 y = train['SalePrice']
5
6 # 스케일러 생성
7 scaler = MinMaxScaler()
8 X_scaled = scaler.fit_transform(X)
9 test_scaled = scaler.transform(test)
10
11 # 데이터프레임으로 변환
12 X = pd.DataFrame(X_scaled, columns=X.columns, index=X.index)
13 test = pd.DataFrame(test_scaled, columns=test.columns, index=test.index)
```

```
1 X.head()
```

	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtUnfSF	TotalBsmtSF	Mo...
0	0.150685	0.033420	0.666667	0.500	0.949275	0.883333	0.12250	0.125089	0.064212	0.140098	...
1	0.202055	0.038795	0.555556	0.875	0.753623	0.433333	0.00000	0.173281	0.121575	0.206547	...
2	0.160959	0.046507	0.666667	0.500	0.934783	0.866667	0.10125	0.086109	0.185788	0.150573	...
3	0.133562	0.038561	0.666667	0.500	0.311594	0.333333	0.00000	0.038271	0.231164	0.123732	...
4	0.215753	0.060576	0.777778	0.500	0.927536	0.833333	0.21875	0.116052	0.209760	0.187398	...

5 rows × 205 columns

```
1 test.head()
```

	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtUnfSF	TotalBsmtSF	Mo...
1460	0.202055	0.048246	0.444444	0.625	0.644928	0.183333	0.0000	0.082920	0.115582	0.144354	...
1461	0.205479	0.060609	0.555556	0.625	0.623188	0.133333	0.0675	0.163536	0.173801	0.217512	...
1462	0.181507	0.058566	0.444444	0.500	0.905797	0.800000	0.0000	0.140149	0.058647	0.151882	...
1463	0.195205	0.040562	0.555556	0.625	0.913043	0.800000	0.0125	0.106662	0.138699	0.151555	...
1464	0.075342	0.017318	0.777778	0.500	0.869565	0.700000	0.0000	0.046598	0.435360	0.209493	...

5 rows × 205 columns

4. 모델링

4.1 그리드서치

하이퍼파라미터 최적화를 위한 기법

사용자가 지정한 여러 하이퍼파라미터의 값들을 조합하여 모든 가능한 경우의 수를 만듦.

각각에 대해 모델을 학습시킨 후 검증 데이터셋을 사용하여 성능을 평가

가장 좋은 성능을 보이는 하이퍼파라미터의 조합을 찾아냄

```
1  from sklearn.model_selection import GridSearchCV
2
3  # GradientBoostingRegressor
4  gbr_param_grid = {
5      'n_estimators': [500, 1000, 3000],
6      'learning_rate': [0.01, 0.1],
7      'max_depth': [3, 6],
8      'min_samples_split': [5,10],
9      'min_samples_leaf': [5,10]
10 }
11
12 # RandomForestRegressor
13 rfr_param_grid = {
14     'n_estimators': [500, 1000, 3000],
15     'max_depth': [None, 5, 10],
16     'min_samples_split': [5,10],
17     'min_samples_leaf': [5,10]
18 }
19
20 # XGBRegressor
21 xgb_param_grid = {
22     'n_estimators': [500, 1000, 3000],
23     'learning_rate': [0.03, 0.3],
24     'max_depth': [4,6,8],
25     'min_child_weight': [0,1]
26 }
27
28 # LGBMRegressor
29 lgbm_param_grid = {
30     'n_estimators': [500, 1000, 3000],
31     'learning_rate': [0.01, 0.1],
32     'num_leaves': [10,30]
33 }
```

```

1 from sklearn.model_selection import KFold, train_test_split, cross_val_score
2 from sklearn.metrics import mean_squared_error
3 from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
4 from sklearn.linear_model import LinearRegression
5 from lightgbm import LGBMRegressor
6 from xgboost import XGBRegressor
7
8 X = train.drop('SalePrice', axis=1)
9 y = train['SalePrice']
10
11 Grid_gbr = GradientBoostingRegressor()
12 Grid_gbr2 = GridSearchCV(estimator=Grid_gbr, param_grid=gbr_param_grid, cv=3, n_jobs=-1, scoring='neg_mean_squared_error', verbose=1)
13 Grid_gbr2.fit(X,y)
14 print(f"Grid_gbr Best Param: {Grid_gbr2.best_params_}")
15
16 Grid_rfr = RandomForestRegressor()
17 Grid_rfr2 = GridSearchCV(estimator=Grid_rfr, param_grid=rfr_param_grid, cv=3, n_jobs=-1, scoring='neg_mean_squared_error', verbose=1)
18 Grid_rfr2.fit(X,y)
19 print(f"Grid_rfr Best Param: {Grid_rfr2.best_params_}")
20
21 Grid_xgb = XGBRegressor()
22 Grid_xgb2 = GridSearchCV(estimator=Grid_xgb, param_grid=xgb_param_grid, cv=3, n_jobs=-1, scoring='neg_mean_squared_error', verbose=1)
23 Grid_xgb2.fit(X,y)
24 print(f"Grid_xgb Best Param: {Grid_xgb2.best_params_}")

```

Fitting 3 folds for each of 48 candidates, totalling 144 fits
Grid_gbr Best Param: {'learning_rate': 0.01, 'max_depth': 3, 'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 3000}
Fitting 3 folds for each of 36 candidates, totalling 108 fits
Grid_rfr Best Param: {'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 1000}
Fitting 3 folds for each of 36 candidates, totalling 108 fits
Grid_xgb Best Param: {'learning_rate': 0.03, 'max_depth': 4, 'min_child_weight': 0, 'n_estimators': 500}

```

1 Grid_lgb = LGBMRegressor()
2 Grid_lgb2 = GridSearchCV(estimator=Grid_lgb, param_grid=lgbm_param_grid, cv=3, n_jobs=-1, scoring='neg_mean_squared_error', verbose=1)
3 Grid_lgb2.fit(X,y)
4 print(f"Grid_lgb Best Param: {Grid_lgb2.best_params_}")

```

Fitting 3 folds for each of 12 candidates, totalling 36 fits
[LightGBM] [Warning] Found whitespace in feature_names, replace with underscores
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001511 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4139
[LightGBM] [Info] Number of data points in the train set: 1460, number of used features: 196
[LightGBM] [Info] Start training from score 12.024057
Grid_lgb Best Param: {'learning_rate': 0.01, 'n_estimators': 1000, 'num_leaves': 10}

```

1 gbr = GradientBoostingRegressor(learning_rate = 0.01, max_depth = 3, min_samples_leaf = 5, min_samples_split = 10, n_estimators = 3000)
2
3 rfr = RandomForestRegressor(max_depth = None, min_samples_leaf = 5, min_samples_split = 5, n_estimators = 1000)
4
5 xgb = XGBRegressor(learning_rate = 0.03, max_depth = 4, min_child_weight = 0, n_estimators = 500)
6
7 lr = LinearRegression()
8
9 lgb = LGBMRegressor(learning_rate = 0.01, n_estimators = 1000, num_leaves = 10)

```

위의 그리드서치를 통해 가장 성능이 좋은 하이퍼파라미터 조합을 찾아냄

4.2 스태킹

스태킹은 여러 다른 모델(regressors)의 예측을 결합하여 최종 예측을 개선하는 앙상블 기법

첫 번째 레벨의 모델들이 데이터에 대한 예측을 수행하고, 그 예측 결과를 기반으로 두 번째 레벨의 메타 모델(meta_regressor)이 최종 예측을 수행

K-Fold Cross Validation: 데이터를 k개의 부분으로 나누고, k-1개의 부분으로 모델을 학습시킨 후, 나머지 부분으로 모델을 검증. 이 과정을 k번 반복

cross_val_score 함수 사용: 교차 검증을 수행하고 각 분할에 대한 점수를 반환.

scoring 파라미터로 "neg_mean_squared_error"를 사용 => 음수 값 반환

교차 검증에서는 높은 점수가 더 좋은 성능을 의미하는데, MSE는 낮을수록 좋으므로 음수를 사용

그 후, MSE는 항상 양수이므로 음수 값을 양수로 변환하기 위해 앞에 -를 붙여준다.

```
1 from mlxtend.regressor import StackingCVRegressor
2
3 stack = StackingCVRegressor(regressors=(gbr, rfr, xgb, lr, lgb),
4 | | | | | meta_regressor=gbr,
5 | | | | | use_features_in_secondary=True)
6
7 kfolds = KFold(n_splits=10, shuffle=True, random_state=42)
8
9 def rmse(y, y_pred):
10 | return np.sqrt(mean_squared_error(y, y_pred))
11
12 def cv_rmse(model, X=X):
13 | rmse = np.sqrt(-cross_val_score(model, X, y, scoring="neg_mean_squared_error", cv=kfolds))
14 | return (rmse)
```

```
1 # 모델에 대한 RMSE 점수 계산
2
3 score = cv_rmse(gbr)
4 print(f"Gradient Boosting Regressor의 CV RMSE 점수: {score.mean():.4f} ({score.std():.4f})")
5
6 score = cv_rmse(rfr)
7 print(f"RandomForestRegressor의 CV RMSE 점수: {score.mean():.4f} ({score.std():.4f})")
8
9 score = cv_rmse(xgb)
10 print(f"XGBRegressor의 CV RMSE 점수: {score.mean():.4f} ({score.std():.4f})")
11
12 score = cv_rmse(lr)
13 print(f"LinearRegression의 CV RMSE 점수: {score.mean():.4f} ({score.std():.4f})")
14
15 score = cv_rmse(lgb)
16 print(f"LGMBMRegressor의 CV RMSE 점수: {score.mean():.4f} ({score.std():.4f})")
```

Gradient Boosting Regressor의 CV RMSE 점수: 0.12474771239865767 (±0.02146951683521616)

RandomForestRegressor의 CV RMSE 점수: 0.1390381409987062 (±0.0246809797855042)

XGBRegressor의 CV RMSE 점수: 0.1268217133191866 (±0.019889384595957006)

LinearRegression의 CV RMSE 점수: 0.13945803742816318 (±0.03315972125967303)

LGMBMRegressor의 CV RMSE 점수: 0.12642650186028948 (±0.021767295338698128)

```
1 print('stack')
2 stack_model = stack.fit(np.array(X), np.array(y))
3
4 print('gbr')
5 gbr_data = gbr.fit(X, y)
6
7 print('rfr')
8 rfr_data = rfr.fit(X, y)
9
10 print('xgb')
11 xgb_data = xgb.fit(X, y)
12
13 print('lr')
14 lr_data = lr.fit(X, y)
15
16 print('lgb')
17 lgb_data = lgb.fit(X, y)
```

4.3 블렌딩

여러 개의 다른 모델을 조합하여 최종 예측

```
1 def blend_models_predict(X):
2     return ((0.2 * gbr_data.predict(X)) + #
3             (0.06 * rfr_data.predict(X)) + #
4             (0.15 * xgb_data.predict(X)) + #
5             (0.04 * lr_data.predict(X)) + #
6             (0.15 * lgb_data.predict(X)) + #
7             (0.4 * stack_model.predict(np.array(X))))
```

```
1 print(rmse(y, blend_models_predict(X)))
```

0.058259734060433256

4.4 예측

```
1 prediction = blend_models_predict(test)
2 y_pred = np.exp(prediction)
```

```
1 y_pred
```

```
array([127545.36851038, 161693.32575017, 176179.34386647, ...,
       160228.53849285, 117246.22150468, 221781.26232366])
```

```
1 submission['SalePrice'] = y_pred
2 submission
```

	Id	SalePrice	
0	1461	127545.368510	
1	1462	161693.325750	
2	1463	176179.343866	
3	1464	195195.488133	
4	1465	189509.688028	
...	
1454	2915	83788.164898	
1455	2916	84335.812272	
1456	2917	160228.538493	
1457	2918	117246.221505	
1458	2919	221781.262324	

1459 rows × 2 columns

4.5 최종 제출

```
submission.to_csv('submission.csv', index = False)
```