



kaggle

# House Prices

- Advanced Regression Techniques

KT AIVLE 김진수 문현정 송정우 오지수 이윤우  
22.02.27~22.04.15 (10days)

# 스터디원 소개



오지수

변수 01~16



김진수

변수 17~32



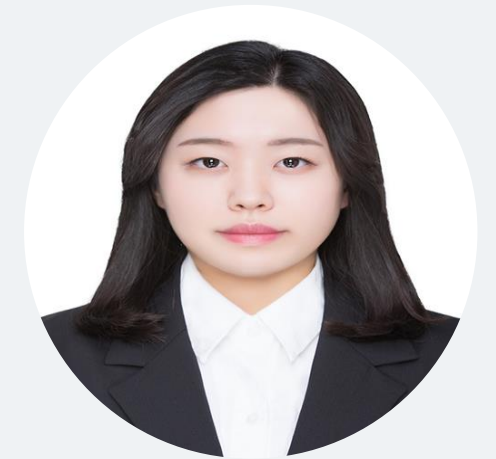
송정우

변수 33 ~ 48



이윤우

변수 49~ 64



문현정

변수 65 ~ 80



# CONTENTS

## Part 1. EDA & CDA

01. 가설수립

02. 이변량 분석

03. 추가 분석

## Part 2. Modeling

01. 데이터 전처리

변수 제거 | 열 변경/추가 | 가변수화  
train validation data 나누기 | 스케일링

02. 머신러닝

Linear Regression | Decision Tree | KNN  
SVM | Random Forest | XGBoost

03. 딥러닝

Fully connected Model  
Locally connected Model

04. test셋에 적용

최적 모델 선택  
전처리 코드 test셋에 적용 | 결측치 조치

05. 결론

예측 및 제출  
프로젝트 후기



## Part 1. EDA & CDA

01. 가설수립

02. 이변량 분석

03. 추가 분석

## | 가설 수립

### 오지수 (01~16)

'MSSubClass' 에 따라 'sale price'에 차이가 있다.

'LotArea' 에 따라 'sale price'에 차이가 있다.

'Neighborhood' 에 따라 'sale price'에 차이가 있다.

### 이윤우 (49~64)

'KitchenQual' 에 따라 'sale price'에 차이가 있다.

'TotRmsAbvGrd' 에 따라 'sale price'에 차이가 있다.

'GarageCars' 에 따라 'sale price'에 차이가 있다.

### 김진수 (17~32)

'YearRemodAdd' 에 따라 'sale price'에 차이가 있다.

'RoofStyle' 에 따라 'sale price'에 차이가 있다.

'BsmtExposure' 에 따라 'sale price'에 차이가 있다.

### 송정우 (33~48)

'TotalBsmtSF' 에 따라 'sale price'에 차이가 있다.

'HeatingQC' 에 따라 'sale price'에 차이가 있다.

'GrLivArea' 에 따라 'sale price'에 차이가 있다.

### 문현정 (65~80)

'WoodDeckSF' 에 따라 'sale price'에 차이가 있다.

'3SsnPorch' 에 따라 'sale price'에 차이가 있다.

'MoSold' 에 따라 'sale price'에 차이가 있다.



## | MSSubClass

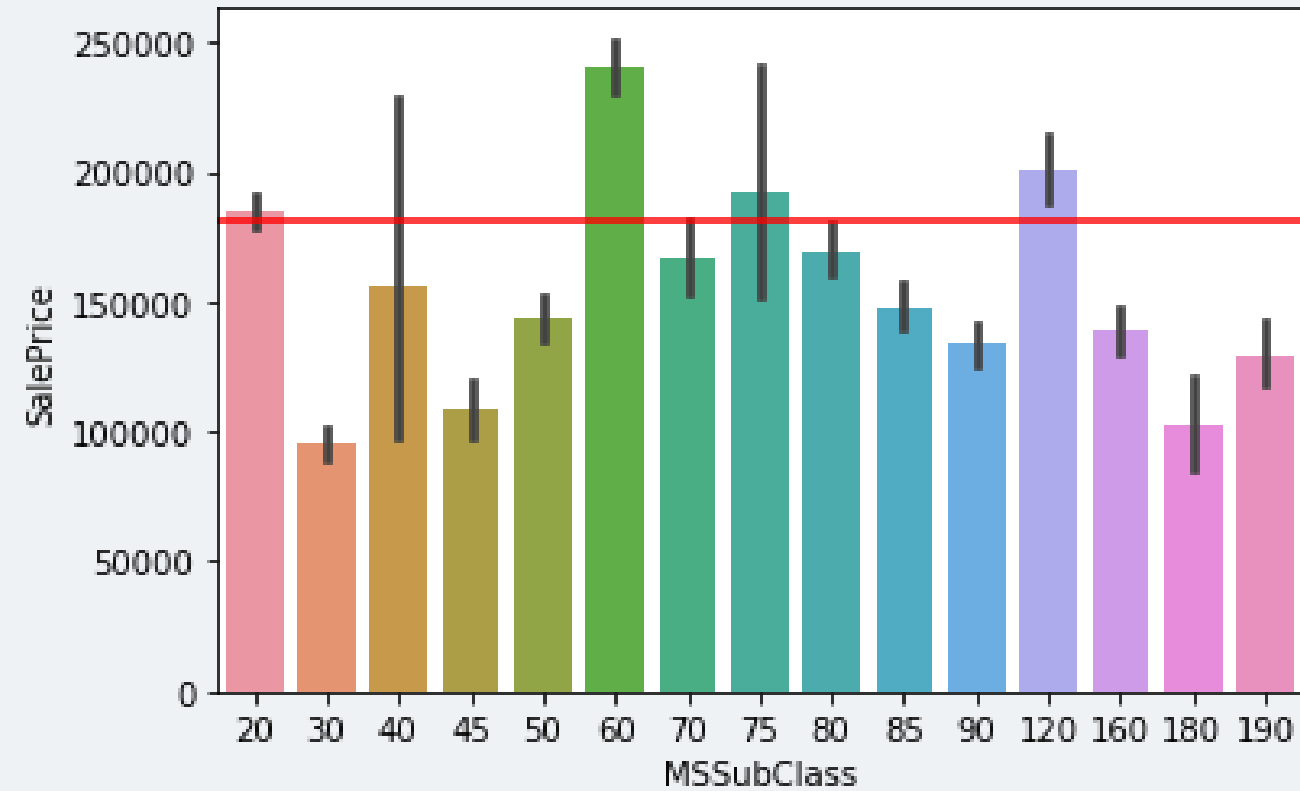
'MSSubClass' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 33.81366022603184$
- $p\text{-value} = 5.895420038257625e-79$

### | 분석 결과 |

- 상관계수가 약 0.34, p-value가 0으로 수렴
- 강한 상관관계



## | MSSubClass

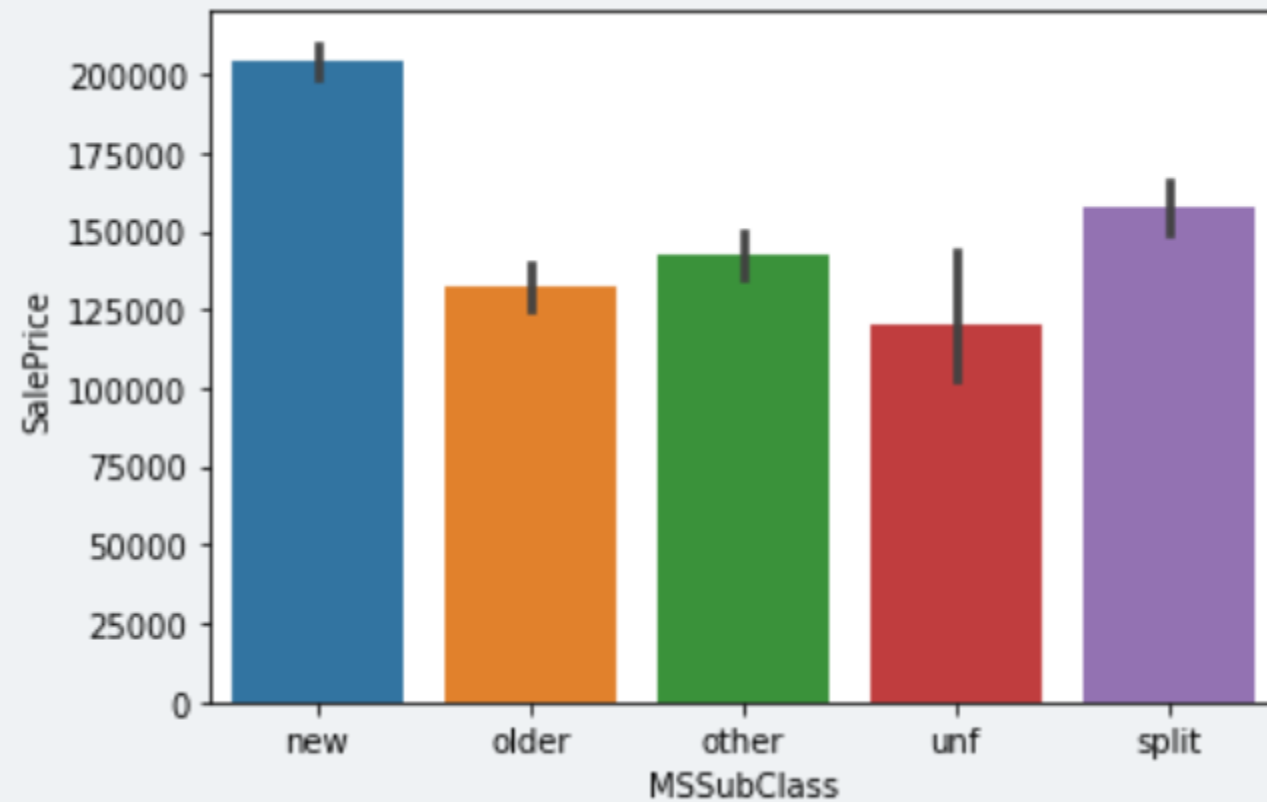
'MSSubClass' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 66.74751470999769$
- $p\text{-value} = 6.79852286119857e-52$

### | 분석 결과 |

- new의 평균 가격대가 가장 높고 이상치는 가장 적음
- unf는 데이터 수가 적어 신뢰구간이 넓은 것으로 추정
- 상관계수가 약 0.66, p-value가 약 0으로 수렴
- 중간 상관관계



## | LotArea

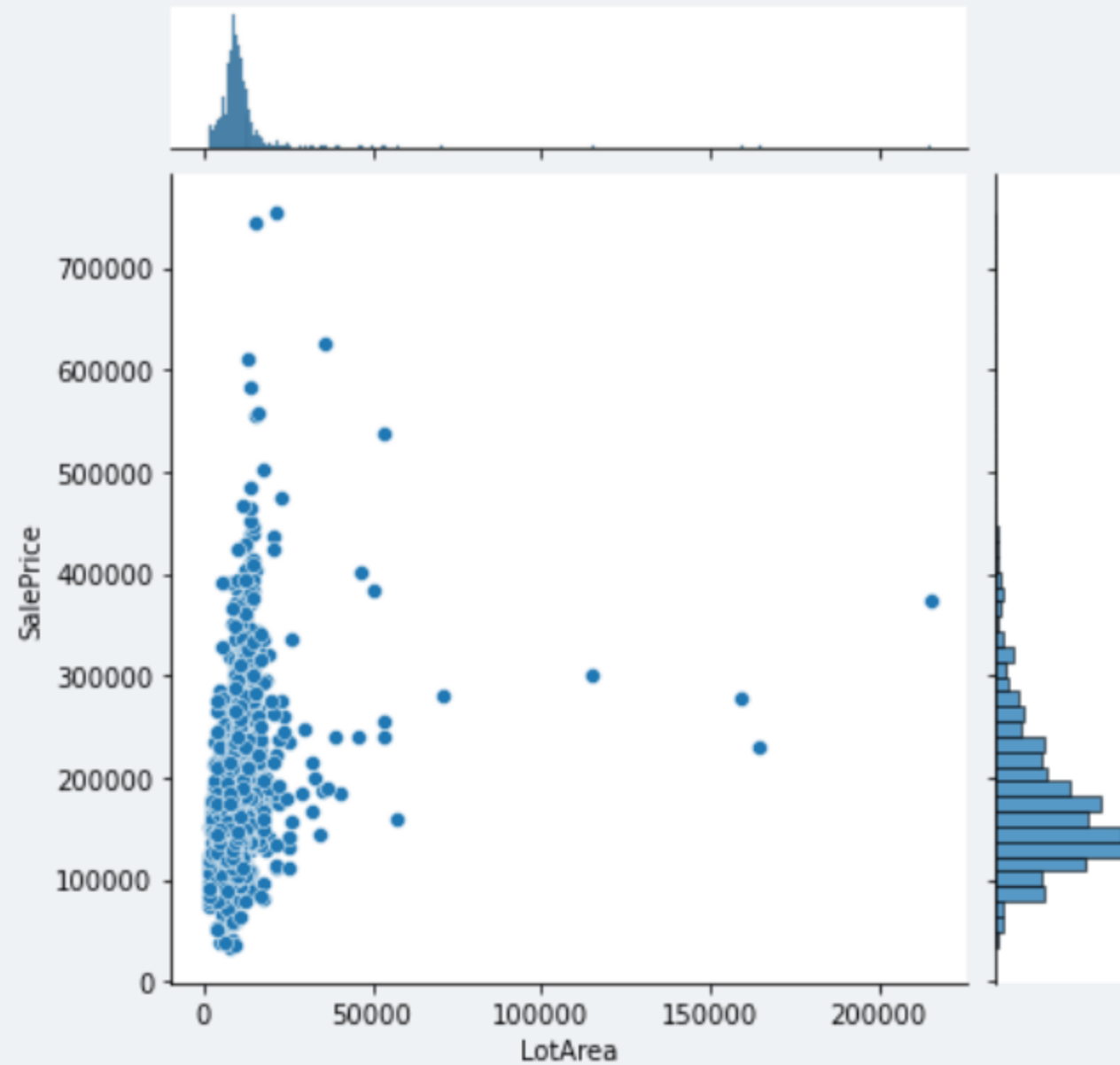
'LotArea' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $P\_statistic = 0.26686518893201316$
- $p\text{-value} = 3.4022092287236576e-25$

### | 분석 결과 |

- LotArea값이 0~50000사이에 대부분 몰려있음
- 산점도에서 직선형태
- 상관계수가 약 0.27, p-value가 약 0으로 수렴
- 중간 상관관계





## | Neighborhood

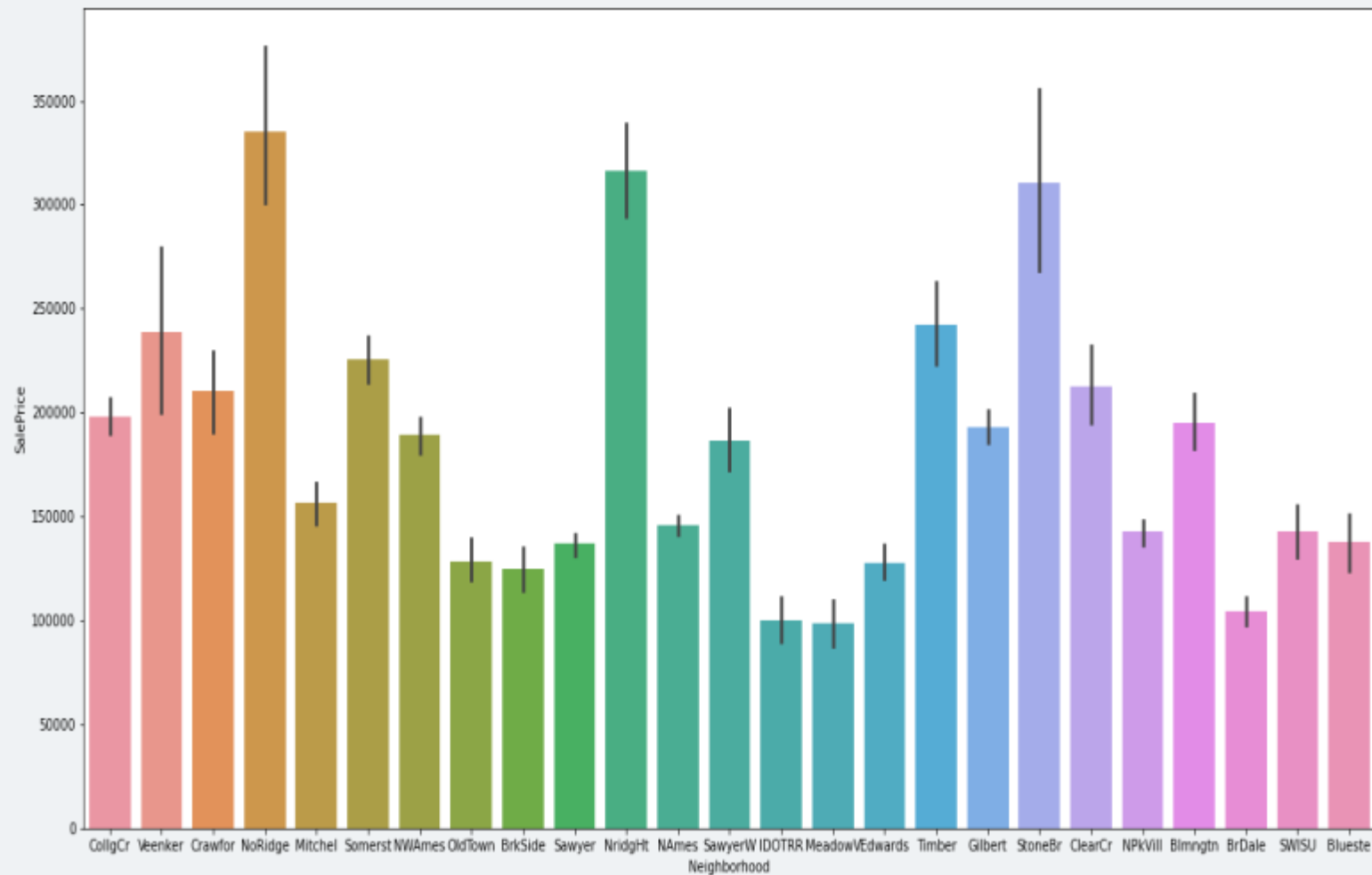
'Neighborhood' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 71.94962254131975$
- $p\text{-value} = 8.131159963685327e-226$

### | 분석 결과 |

- 노스리지, 네브래스카 링컨의 아파트 단지(NridgHt), Stone Brook(뉴욕 주립대)은 학군으로 유명해서 집값이 높은 곳으로 추정
- 상관계수가 약 0.72, p-value가 0에 수렴
- 중간 상관관계



## | YearRemodAdd

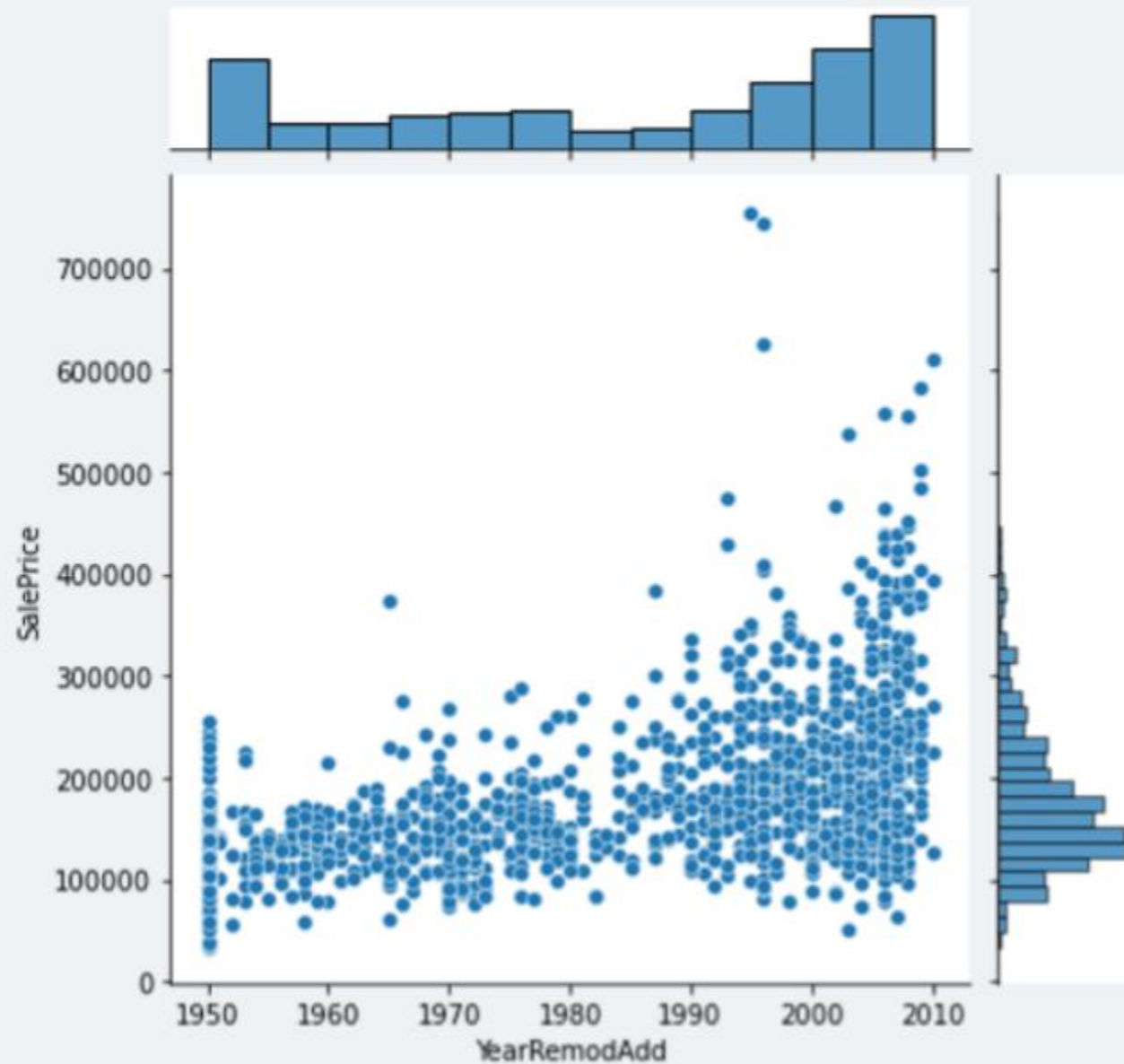
'YearRemodAdd' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $p\_statistic = 0.5071009671113862$
- $p\text{-value} = 3.1649482419200737e-96$

### | 분석 결과 |

- 상관계수가 약 0.5, p-value가 0에 수렴
- 연도에 상관없이 50,000\$에서 200,000\$사이에 골고루 분포되어 있지만, 200,000\$ ~ 450,000\$의 집값을 가지고 있는 집은 2010년도에 가까울수록 많이 분포



## | RoofStyle

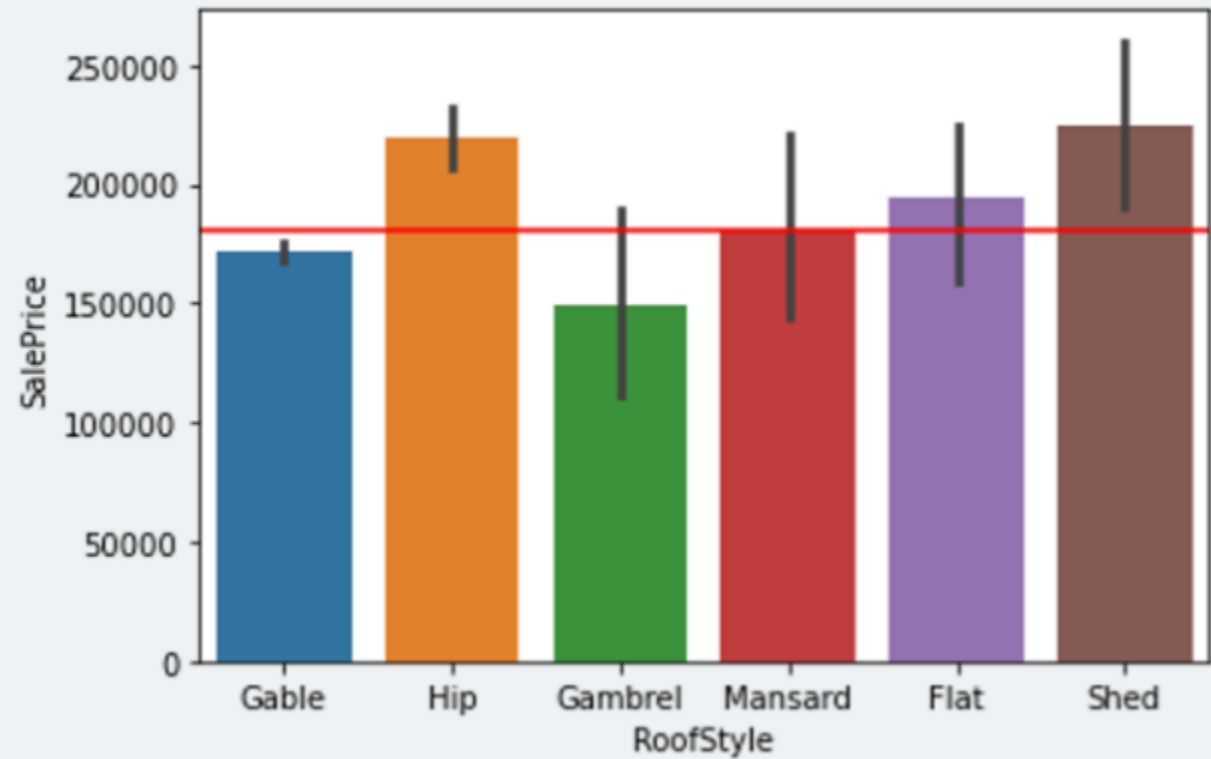
'RoofStyle' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 6.727304893420313$
- $p\text{-value} = 7.231444779987188e-08$

### | 분석 결과 |

- 상관계수가 약 7이고 p-value가 0에 수렴
- 약한 상관관계



## | BsmtExposure

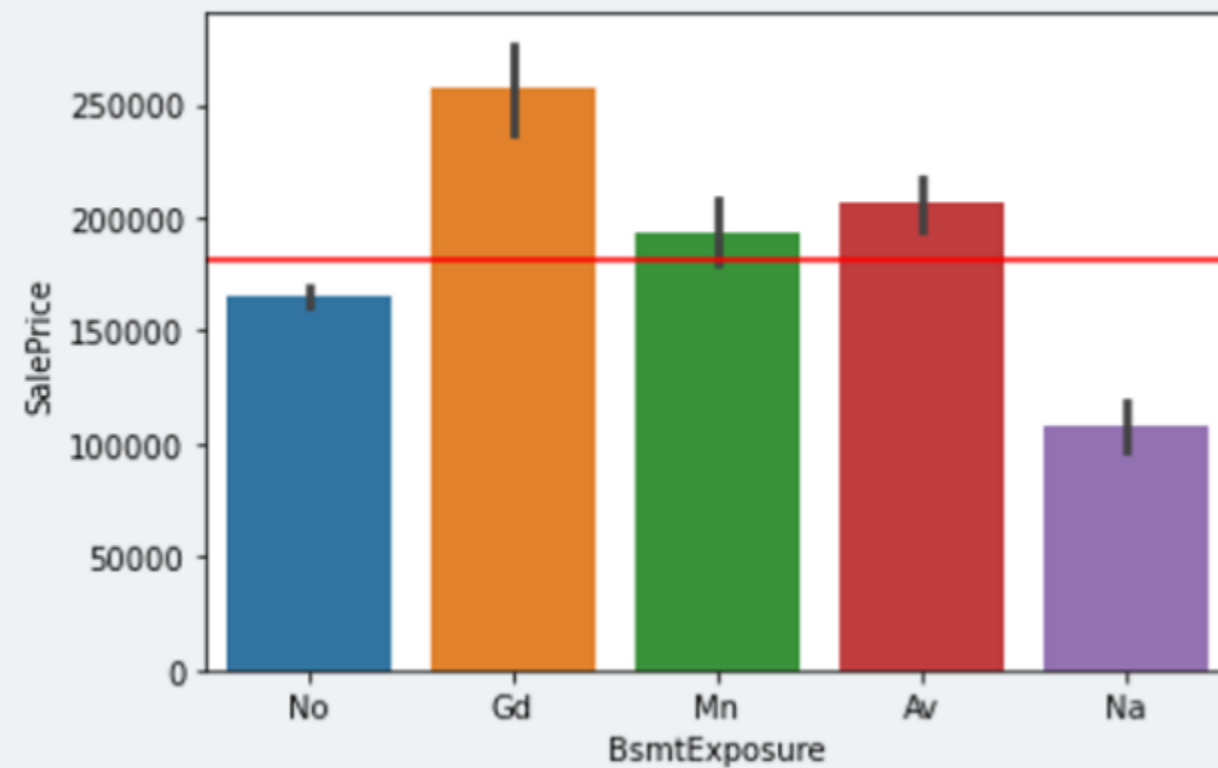
'BsmtExposure' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 63.939761270066455$
- $p\text{-value} = 7.557758359196251e-50$

### | 분석 결과 |

- 상관계수가 약 64이고, p-value가 0에 수렴
- 지하실이 없는 집의 평균이 가장 낮음



## | TotalBsmtSF

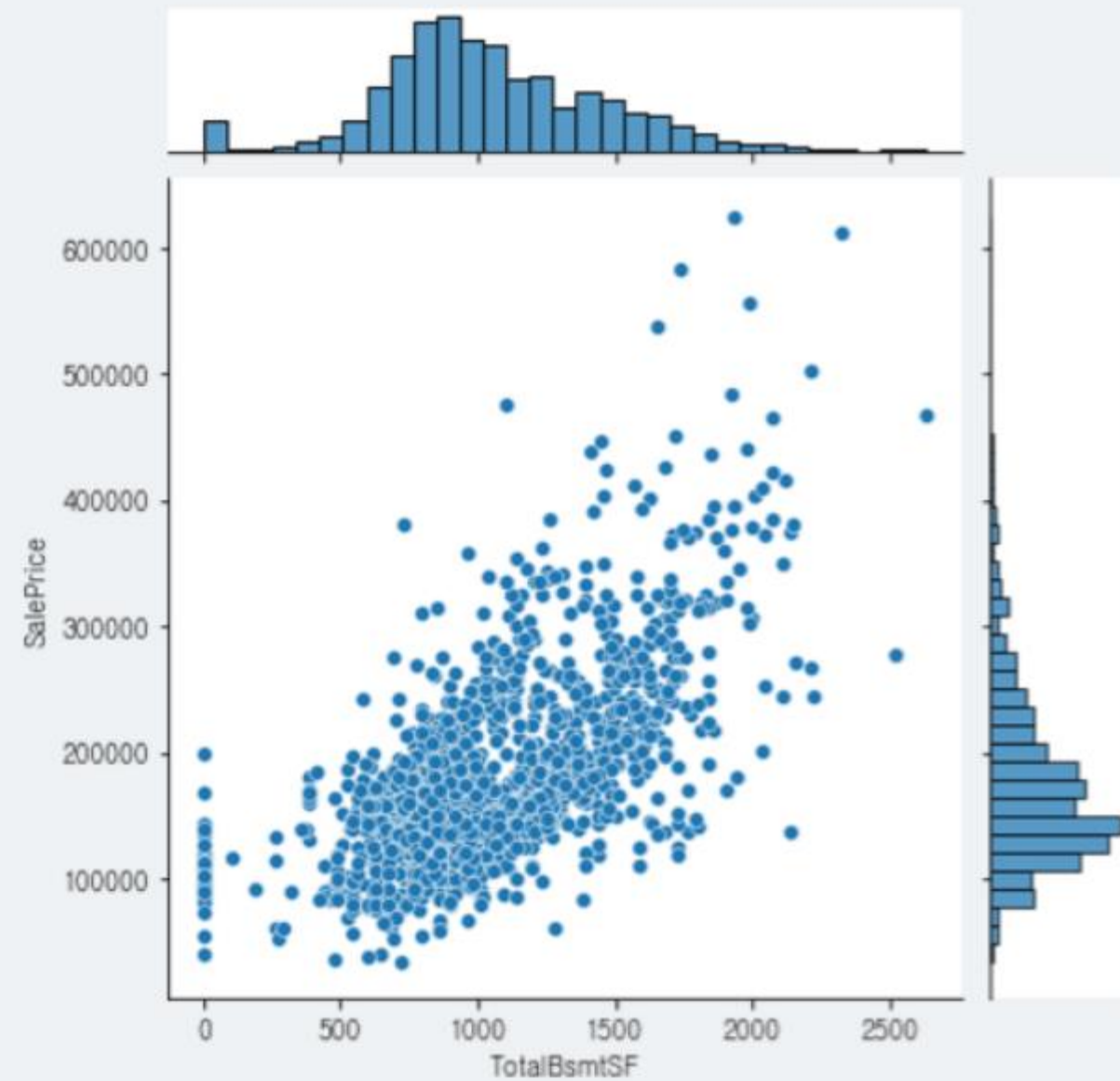
'TotalBsmtSF' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- P\_statistic = 0.6357078647094846
- p-value = 5.375162811876402e-165

### | 분석 결과 |

- 700~1500 사이에 대부분 몰려있음
- 산점도에서 직선형태
- 상관계수가 약 0.64, p-value가 약 0으로 수렴
- 강한 상관관계





## | HeatingQC

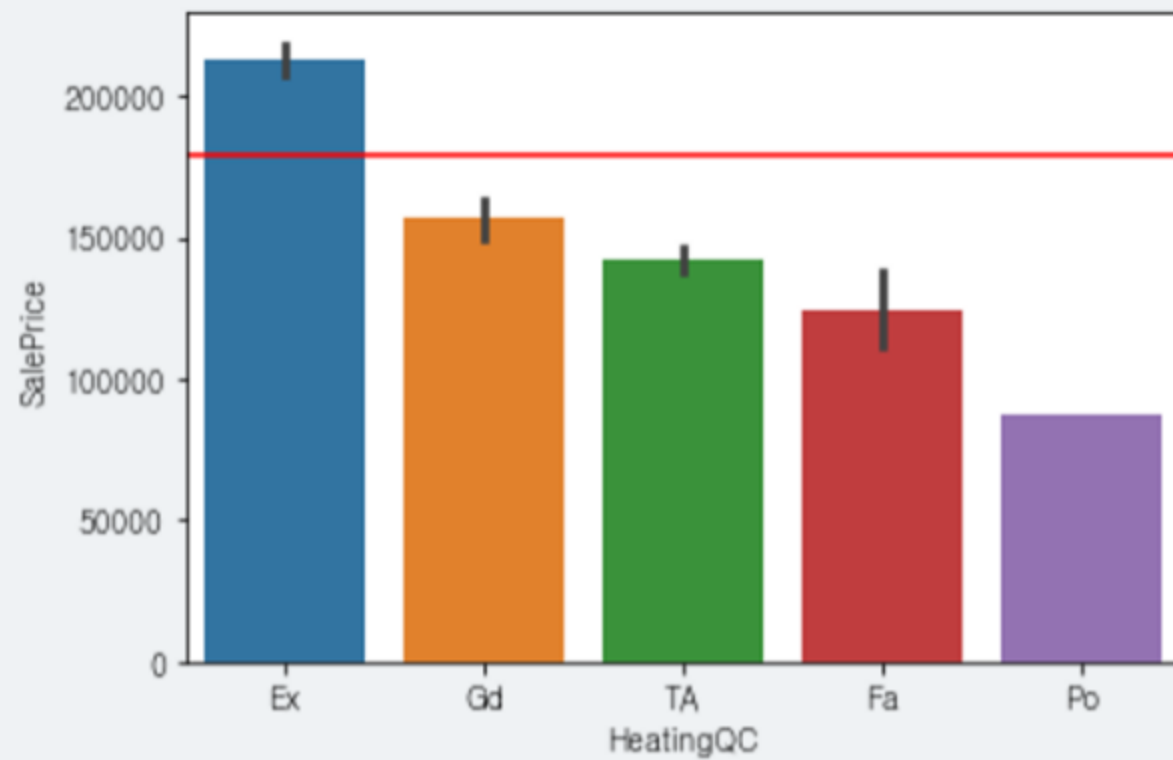
'HeatingQC' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 91.5842207923067$
- $p\text{-value} = 1.8524937490018277e-69$

### | 분석 결과 |

- 난방 품질 및 상태가 좋을수록 집값이 높게 형성
- 상관계수가 약 91.6, p-value가 약 0으로 수렴
- 강한 상관관계



## | GrLivArea

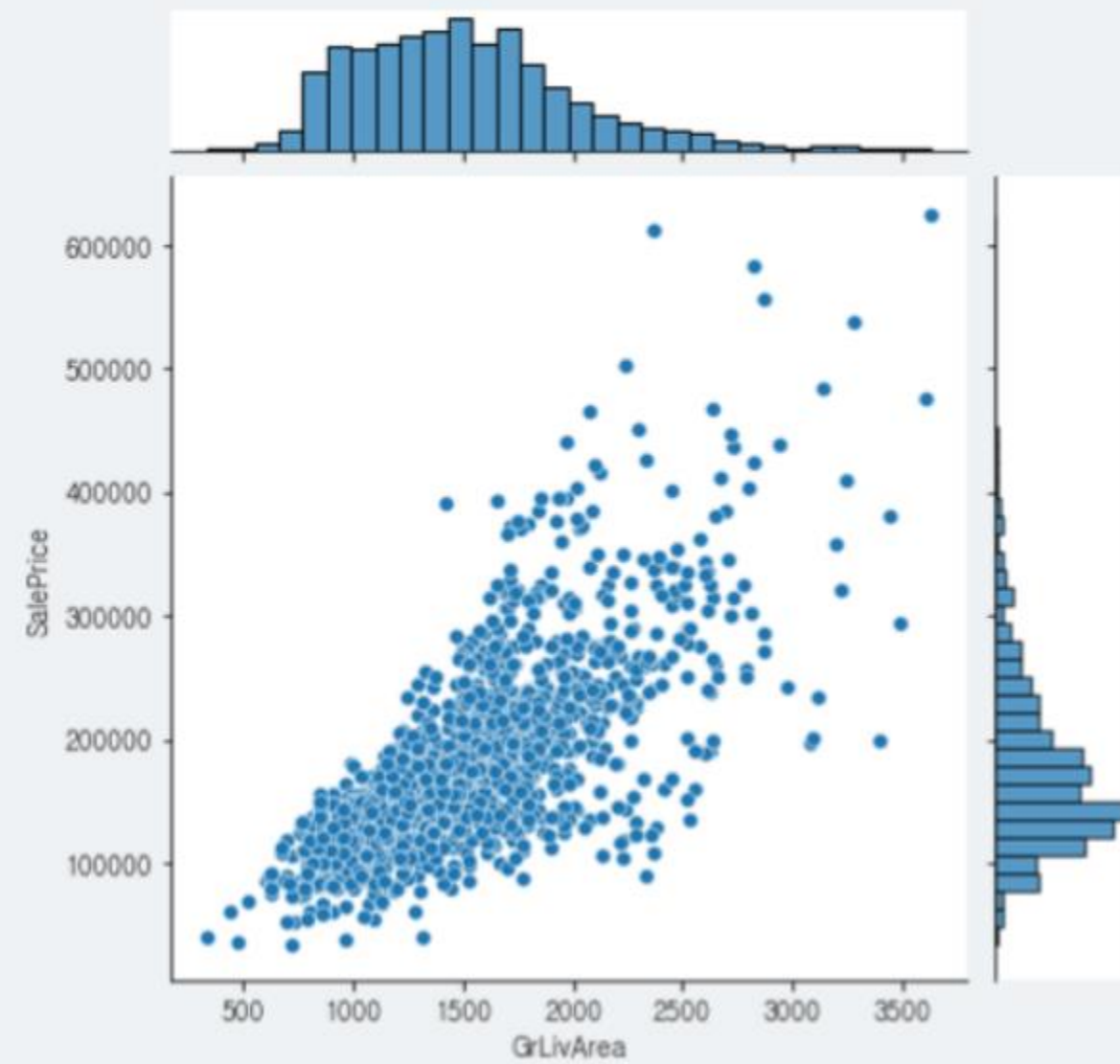
'GrLivArea' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- P\_statistic = 0.7176427345610046
- p-value = 8.612256299535791e-230

### | 분석 결과 |

- 600~2000 사이에 대부분 값이 몰려있음
- 상관계수가 약 0.72, p-value가 약 0으로 수렴
- 강한 상관관계



## | KitchenQual

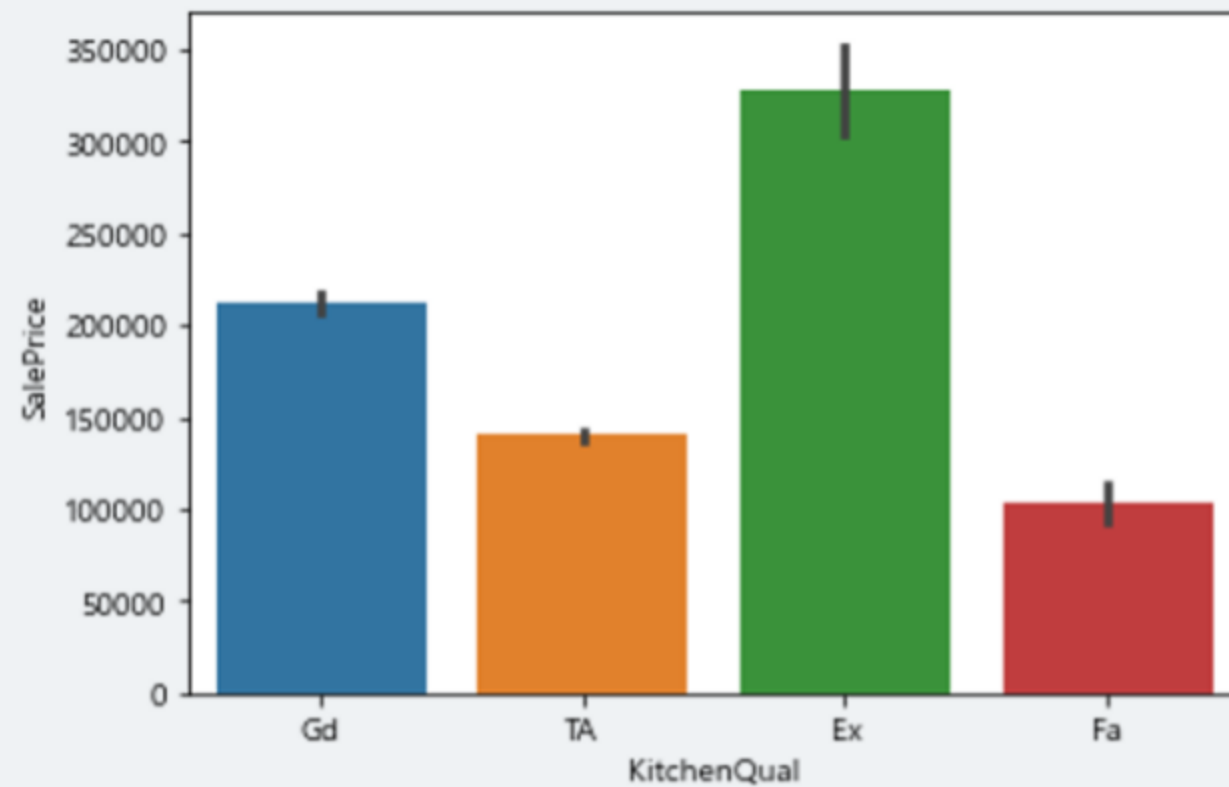
'KitchenQual' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 408.0748428508533$
- $p\_value = 3.0669301958454065e-192$

### | 분석 결과 |

- 뛰어난 수준의 집 값이 가장 높음
- 뛰어난 수준을 제외하고 표준편차가 작은 편
- 평균 이하의 데이터를 제외하고 이상치 확인
- 상관계수가 약 408, p-value가 0으로 수렴
- 강한 상관관계



## | TotRmsAbvGrd

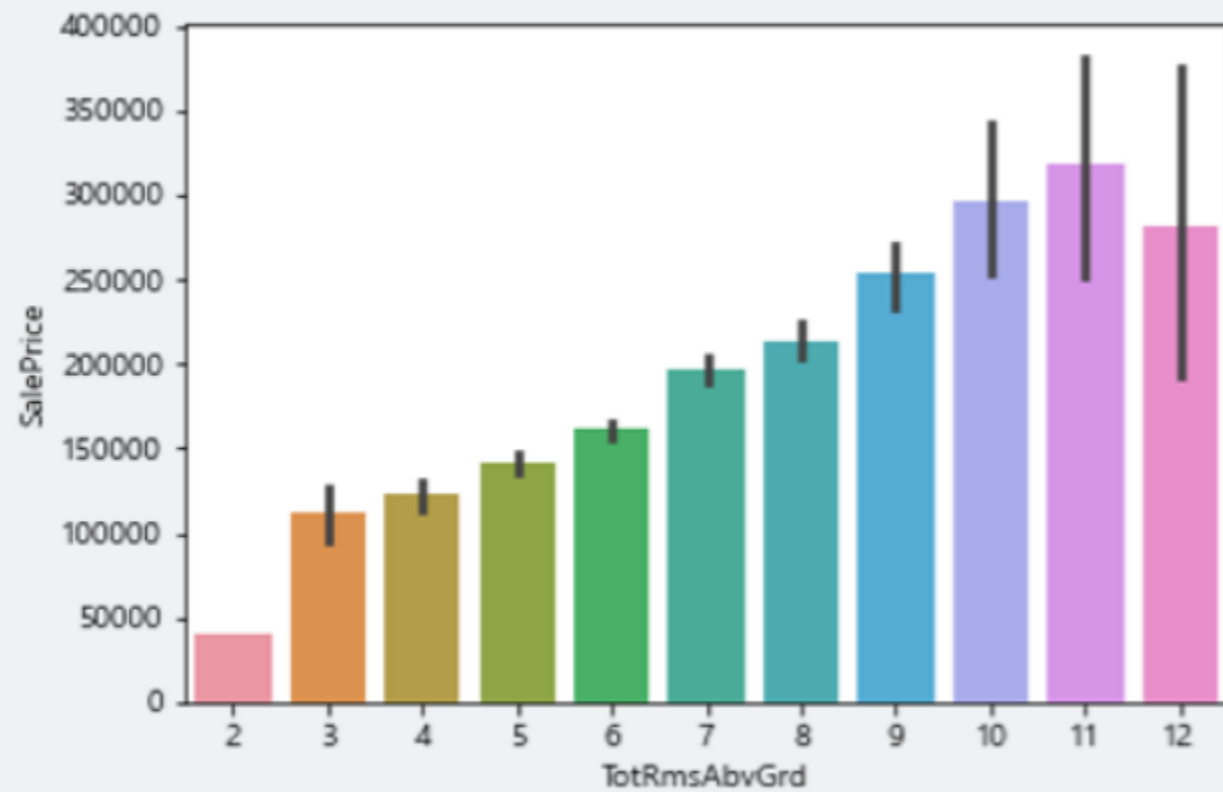
'TotRmsAbvGrd' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 61.710234688022716$
- $p\text{-value} = 2.368532467263881e-104$

### | 분석 결과 |

- 방이 많을수록 집값이 높음
- 9개 이상의 데이터는 표준편차가 큼
- 방 2개 데이터는 데이터가 거의 없음
- 상관계수가 약 62, p-value가 0으로 수렴
- 중간 상관관계



## | TotRmsAbvGrd

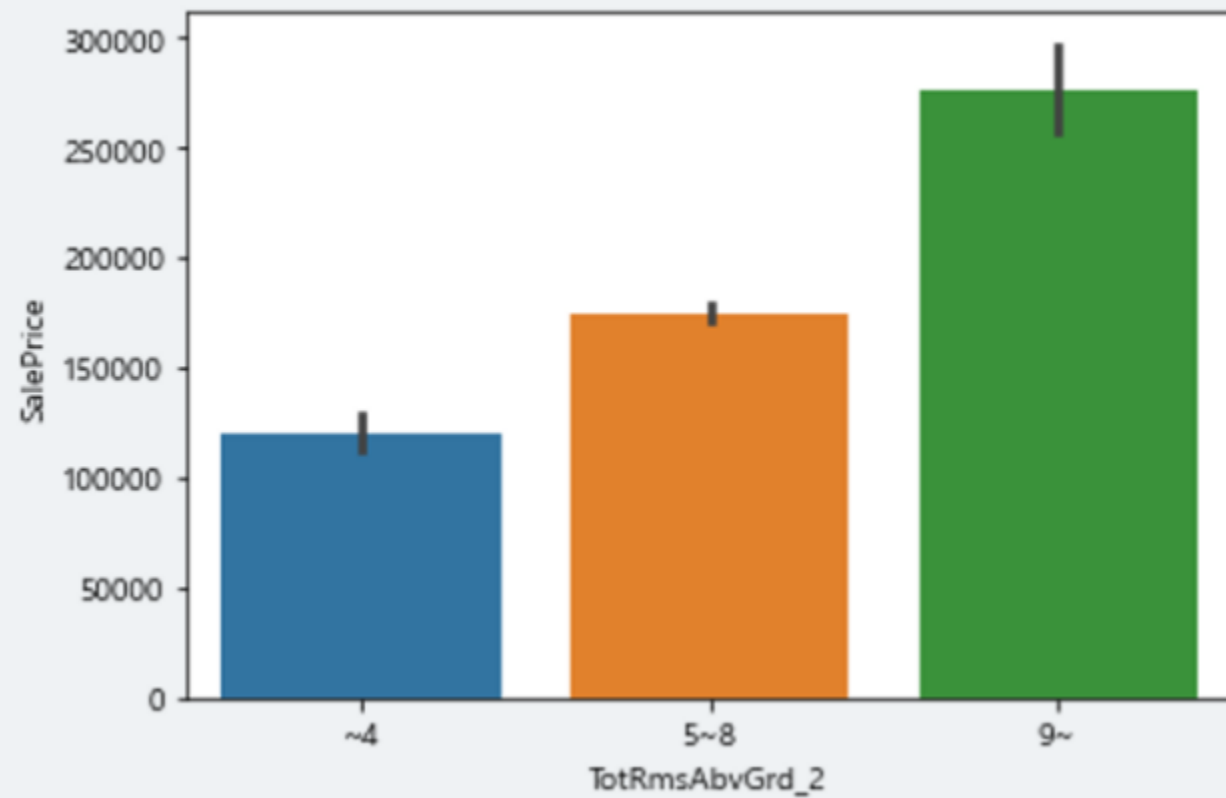
'TotRmsAbvGrd' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 181.29685514836837$
- $p\text{-value} = 5.057064072690088e-71$

### | 분석 결과 |

- 방이 많을수록 가격이 높고 9개 이상의 데이터는 표준편차가 큼
- 상관계수가 약 181, p-value가 0으로 수렴
- 강한 상관관계





## | GarageCars

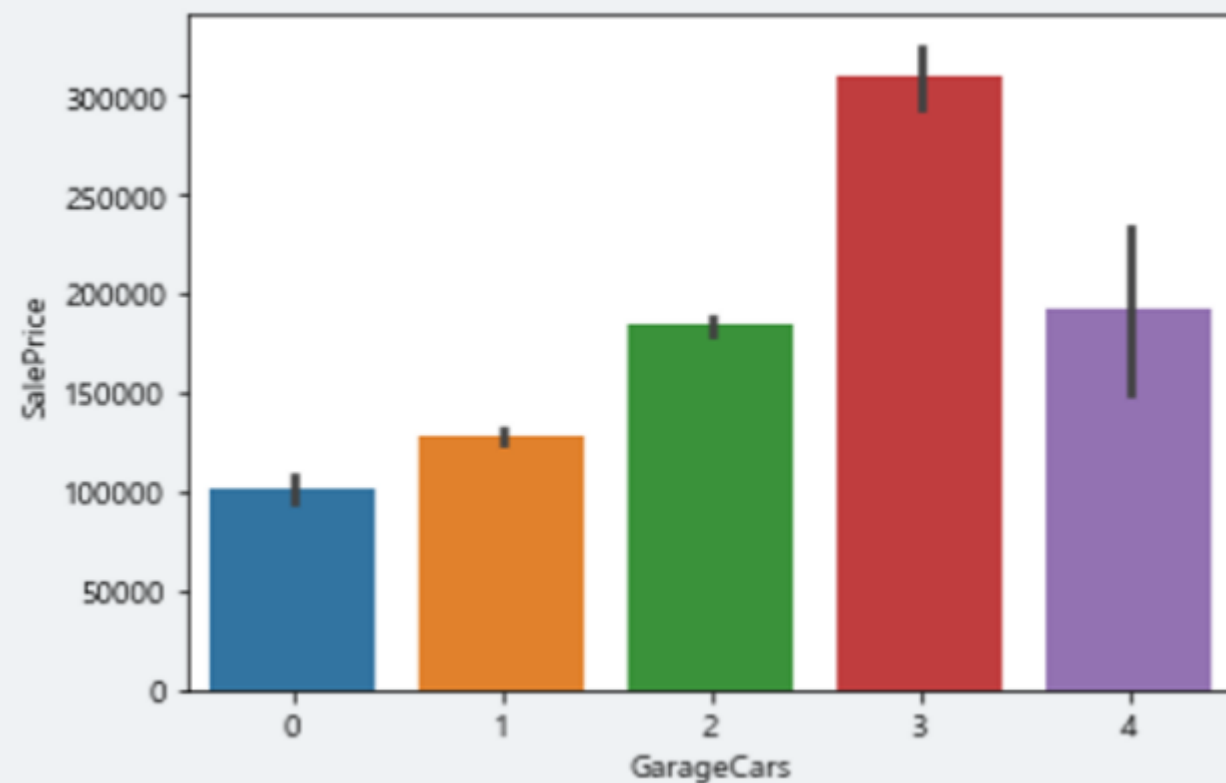
'GarageCars' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 352.0287016481662$
- $p\text{-value} = 6.423496866748168e-212$

### | 분석 결과 |

- 수용 가능한 차가 3대일 때, 집 값의 평균 가장 높음
- 3대인 데이터에 높은 이상치 데이터  
(4대인 경우가 5개의 데이터밖에 없기 때문)
- 차고가 없는 데이터의 평균이 가장 낮음
- 상관계수가 약 352, p-value가 0으로 수렴
- 강한 상관관계



## | GarageCars\_2

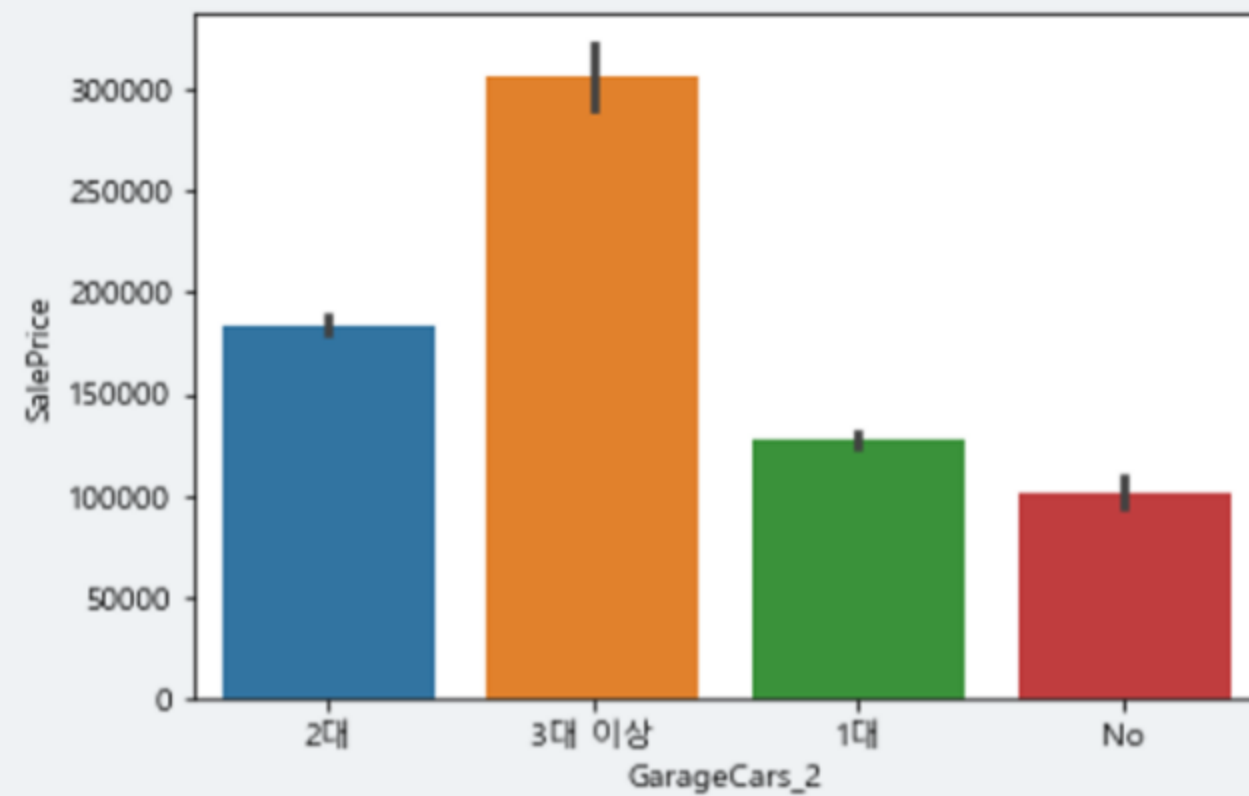
'GarageCars' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $F\_statistic = 456.2884885348042$
- $p\text{-value} = 7.904404068035168e-209$

### | 분석 결과 |

- 상관계수가 약 456, p-value가 0으로 수렴
- 강한 상관관계



## | WoodDeckSF

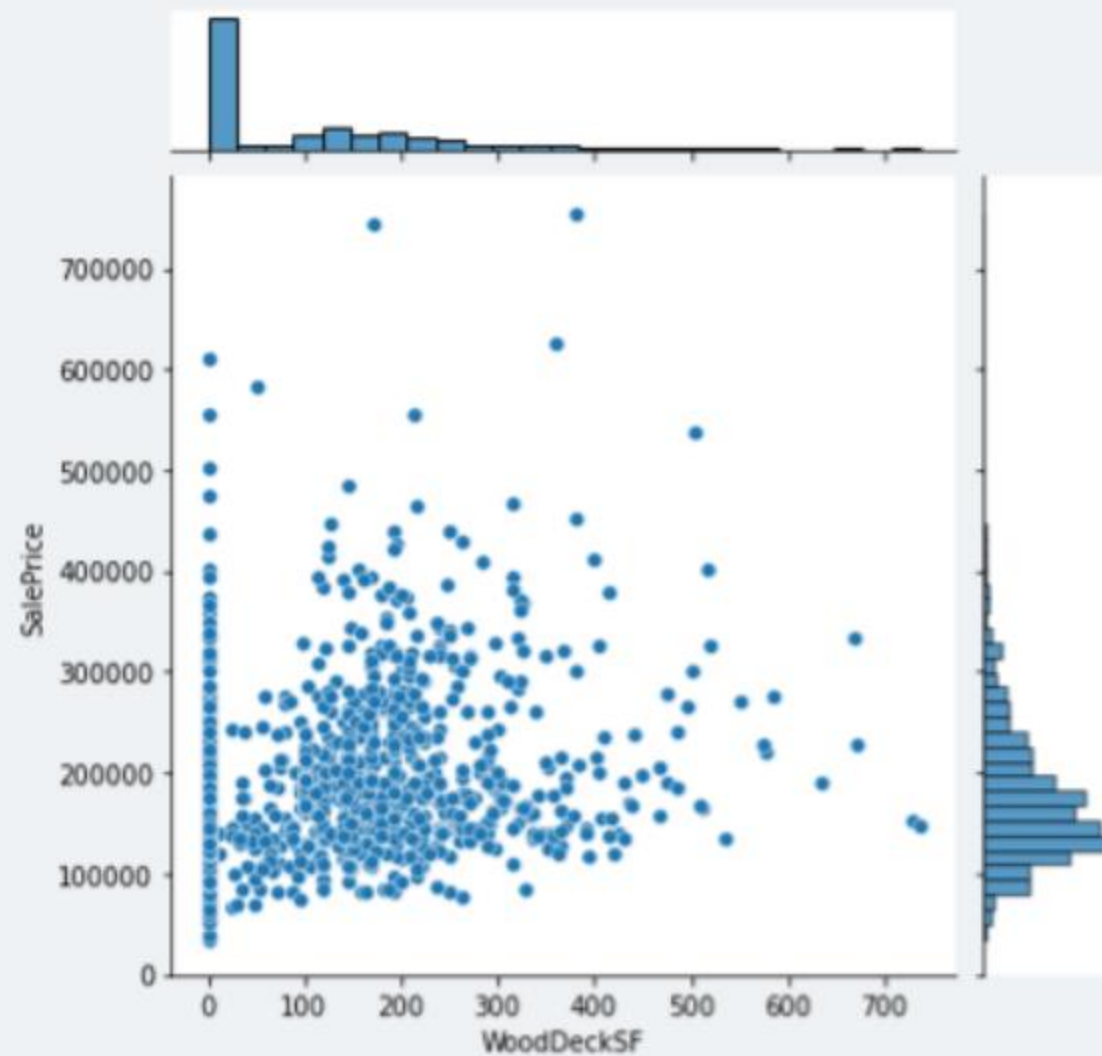
'WoodDeckSF' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

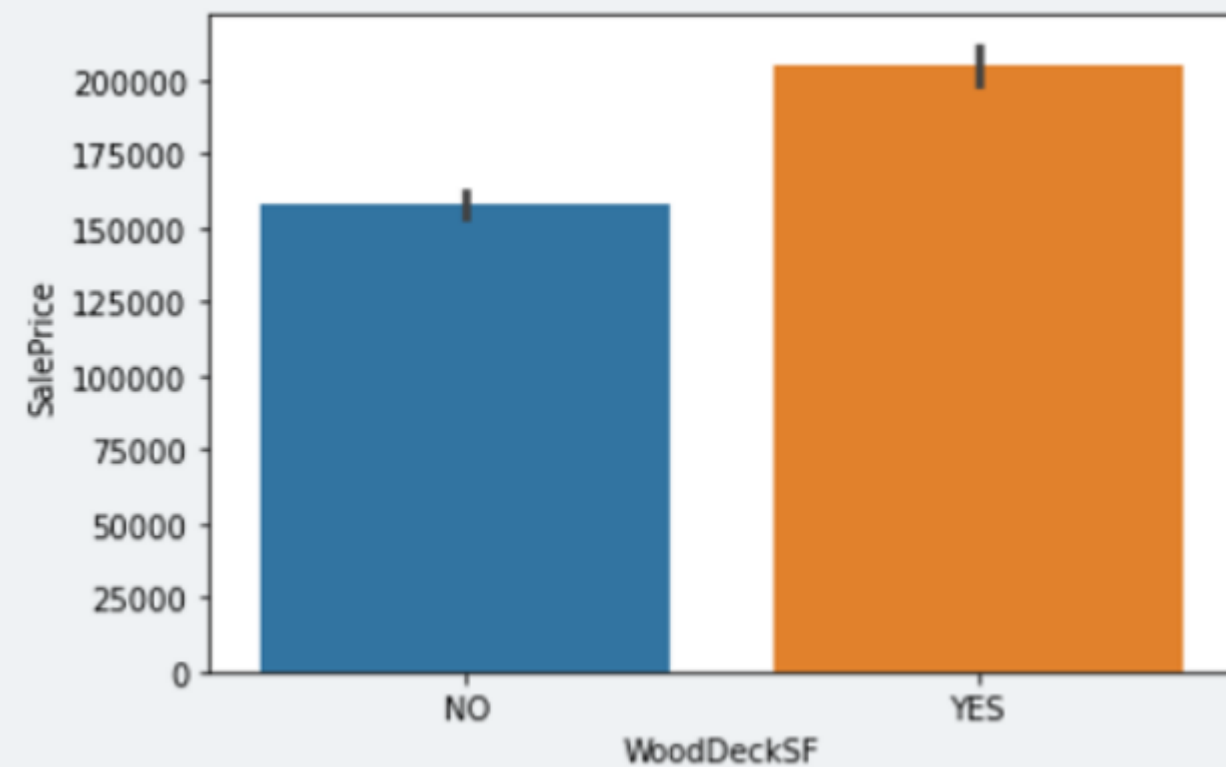
- $p\_statistic = 0.31938969671966416$
- $p\text{-value} = 7.747425741541596e-36$

### | 분석 결과 |

- 목재 데크 영역이 클수록 집값이 높아짐.
- 상관계수가 약 0.32, p-value가 0으로 수렴
- 중간 상관관계



## | WoodDeckSF\_cut



'WoodDeckSF' 에 따라 'sale price'에 차이가 있다.

### | 변수 추가 |

- WoodDeckSF\_cut : 목재 데크 여부로 그룹화

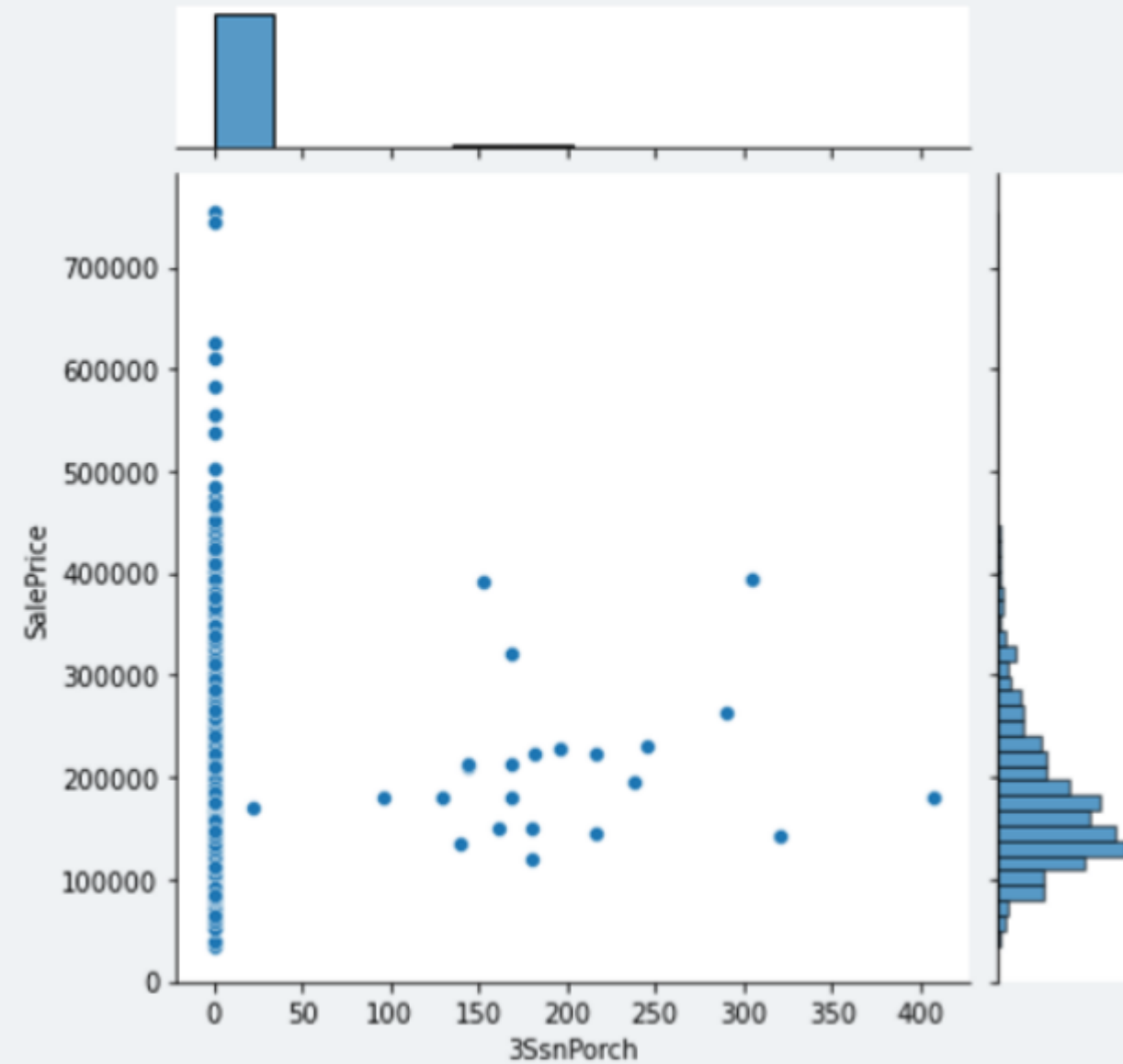
### | 수치 검정 |

- Ttest\_statistic= -11.82891153815244
- p-value = 6.889258790664898e-31

### | 분석 결과 |

- 목재 데크 영역이 있으면 판매가격이 상대적으로 높은 편
- WoodDeckSF에 따라 SalePrice에 차이가 있음
- 상관계수가 약 -12, p-value가 0으로 수렴하는 것으로 보아, 타겟과 강한 상관관계가 있다고 볼 수 있음

## | 3SsnPorch



'3SsnPorch' 에 따라 'sale price'에 차이가 있다.

### | 수치 검정 |

- $P\_statistic = 0.050606810476752334$
- $p\text{-value} = 0.05369577345446337$

### | 분석 결과 |

- 상관계수가 약 0.5, p-value가 0.053로 보아, 타겟과 상관관계가 없다고 볼 수 있음

### | 추가 분석 |

- 0은 3계절 현관 면적이 존재하지 않는 것으로 추측됨  
-> 3계절 현관 면적이 있다(24), 없다(1436)로 그룹화 후 분석



## | 3SsnPorch

'3SsnPorch' 에 따라 'sale price'에 차이가 있다.

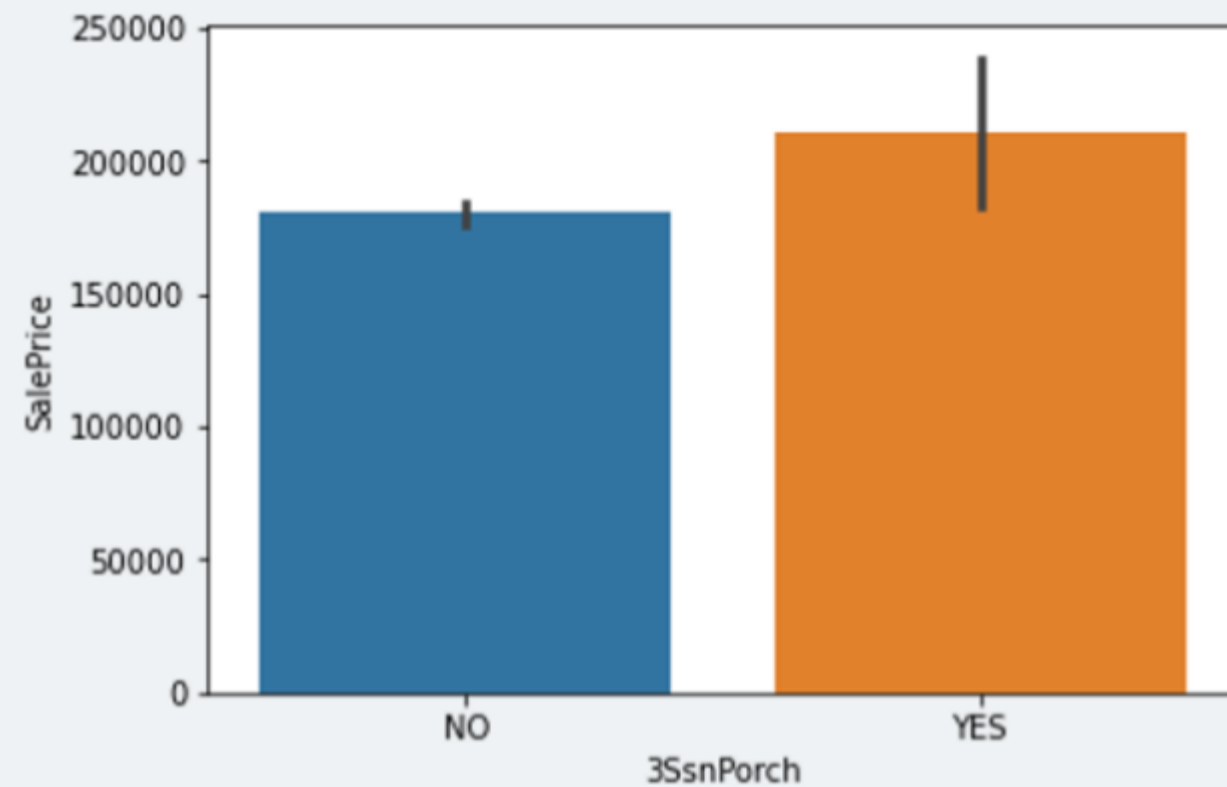
### | 수치 검정 |

- $F\_statistic = 0.9322672585401565$
- $pvalue = 0.5080059037091408$

### | 분석 결과 |

- 상관계수가 0.93, p-value가 0.0508으로, 타겟과 상관관계가 없다고 볼 수 있음

=> 변수 삭제



## | MoSold

'MoSold' 에 따라 'sale price'에 차이가 있다.

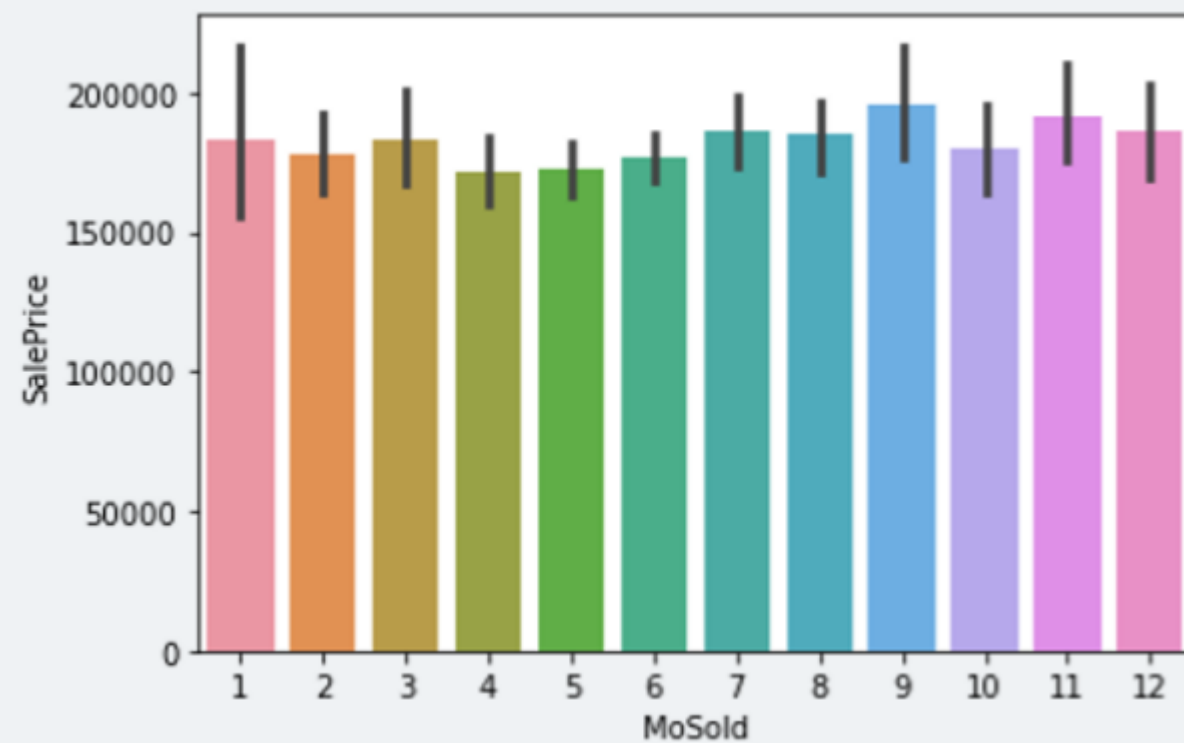
### | 수치 검정 |

- Ttest\_statistic = -1.816659084895562,
- p-value = 0.06947542900237297

### | 분석 결과 |

- 상관계수가 약 -1.8, p-value가 0.06으로,
- 타겟과 상관관계가 없다고 볼 수 있음

=> 변수 삭제





## Part 2. Modeling

### 01. 데이터 전처리

변수 제거 | 열 변경/추가 | 가변수화  
train validation data 나누기 | 스케일링

### 02. 머신러닝

Linear Regression | Decision Tree | KNN  
SVM | Random Forest | XGBoost

### 03. 딥러닝

Fully connected Model  
Locally connected Model

### 04. test셋에 적용

최적 모델 선택  
전처리 코드 test셋에 적용 | 결측치 조치

### 05. 결론

예측 및 제출  
프로젝트 후기

## | 변수 제거 - 이변량을 바탕으로 제거

### 1. 상관계수, T/F 통계량

| 상관관계 분석 기준 | P-value 0.05 이상 | 상관계수 절대값 0.1 미만 | T검정 절대값 2 미만 | F검정 3 미만

| 상관관계 | 0-1     | T  | t  >=  | F  | 2-3이상  |
|------|---------|----|--------|----|--------|
| 강함   | >=  0.5 | 강함 | >=  10 | 강함 | >= 100 |
| 중간   | >=  0.2 | 중간 | >=  5  | 중간 | >= 50  |
| 약함   | >=  0.1 | 약함 | >=  2  | 약함 | >= 3   |



기준에 미달되는 변수 제거

```
cols = ['Street', 'LandSlope', 'Condition2', 'OverallCond', 'BsmtFinSF2', 'LowQualFinSF',  
        '3SsnPorch', 'PoolArea', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold']  
x_train.drop(cols, axis=1, inplace=True)
```

# | 변수 제거 - 이변량을 바탕으로 제거

## 2. 다중 공선성

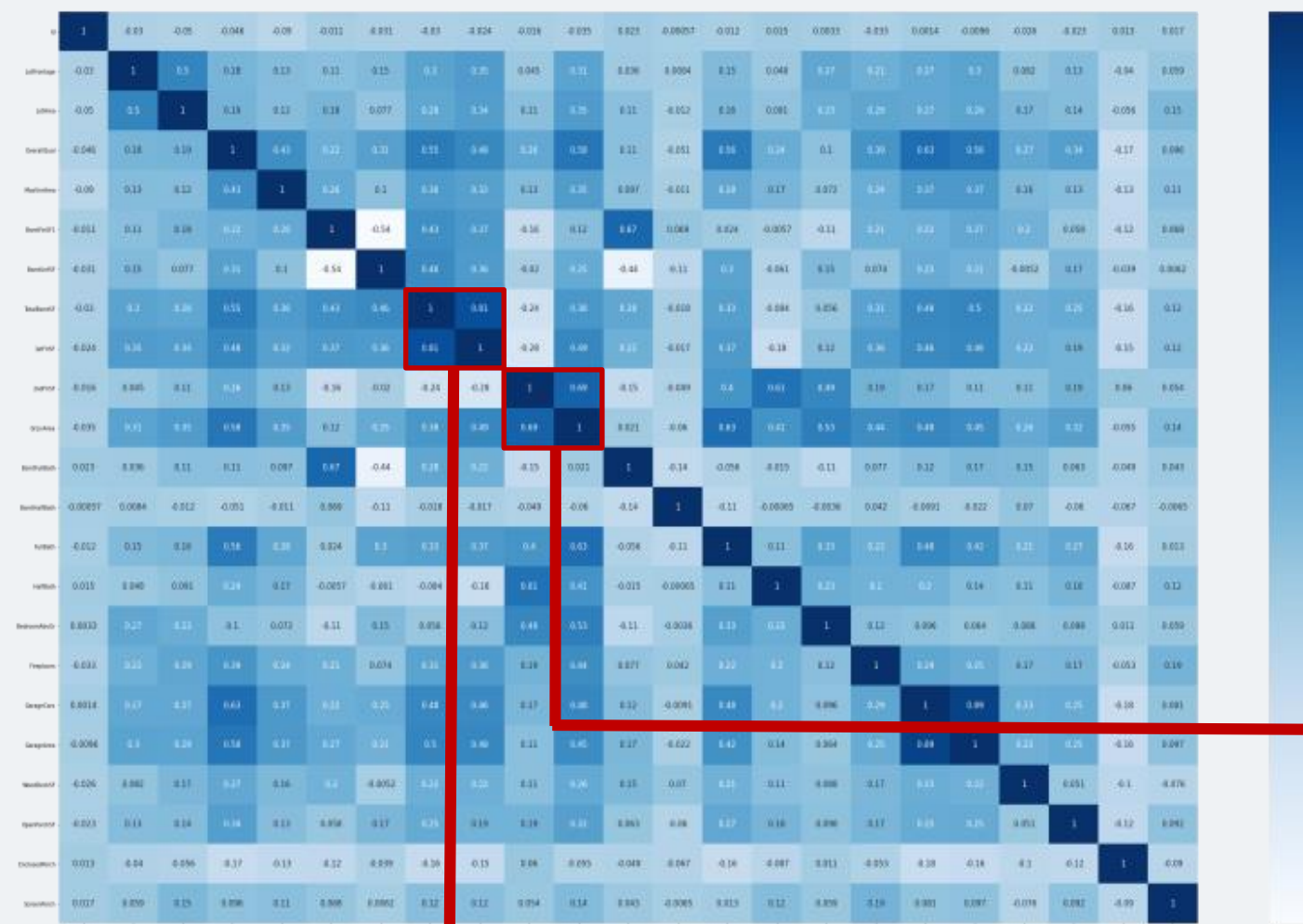
### | 판단 기준 |

- feature들 간 상관관계가 약 0.7 이상일 경우 다중 공선성 문제가 있는 것으로 판단

### | 기준에 해당하는 변수 |

- 1stFlrSF & TotalBsmtSF : 0.81
- GrLivArea & 2ndFlrSF : 0.69

\* GarageCars & GarageArea의 경우 0.89이지만, 범주형 vs 연속형임으로 다중공선성을 파악할 수 없기때문에 삭제하지 않음



|             |      |      |
|-------------|------|------|
| TotalBsmtSF | 1    | 0.81 |
| 1stFlrSF    | 0.81 | 1    |

|           |      |      |
|-----------|------|------|
| 2ndFlrSF  | 1    | 0.69 |
| GrLivArea | 0.69 | 1    |



## | 변수 제거 - 이변량을 바탕으로 제거

### 2. 다중 공선성

#### | VIF (분산팽창지수) |

|   | VIF Factor | features     |
|---|------------|--------------|
| 0 | 14.71      | Intercept    |
| 1 | 53.78      | firstFlrSF   |
| 2 | 74.11      | secondFlrSF  |
| 3 | 2.95       | TotalBsmntSF |
| 4 | 89.57      | GrLivArea    |

#### | 판단 기준 |

- 변수의 분산팽창지수가 10이상일 때, 해당 변수에 대하여 다중공선성이 있다고 판단

#### | 기준에 해당하는 변수 |

- firstFlrSF, secondFlrSF, GrLivArea

상관관계에서 분석한 결과와 동일하기 때문에,  
firstFlrSF와 secondFlrSF를 **삭제**하기로 결정

## | 변수 제거

### | 다중공선성 |

```
cols = ['firstFlrSF', 'secondFlrSF']  
x_train.drop(cols, axis=1, inplace=True)
```

### | 고유 변수 |

```
x_train.drop('ld', axis=1, inplace=True)
```

### | NaN 비중이 높았던 변수 |

```
x_train = x_train.drop('Alley', axis=1)  
x_train = x_train.drop('Utilities', axis=1)
```

## | 변수 변경/추가 - 1. 유사한 범주값 통합

### | 유사한 의미의 범주값을 합침 |

- MSSubClass : 집값에 영향을 줄 요인 별로 묶어 구분  
new (1946년 이후 건설) , older (1945년 이전) , split (엇층, 엇계단), unf (미완성) , other (나머지)
- LotShape : 대지 모양으로 묶어 구분 Reg( 모양이 규칙적 ), ir ( 모양이 불규칙적)
- LandSlope : 경사 유무로 묶어 구분 Gtl ( 경사가 없는 평지), ste (경사 유)
- Condition1 : 주변 상태의 긍,부정으로 묶어 구분 Norm (일반적), pos(긍정적), nag(부정적)
- HouseStyle : 층수로 묶어 구분 Sto1 (1층), sto2 (2층이상), unf (미완성)
- Exterior1st : 같은 재료별로 묶어 구분 Wood(나무류) , Panull(합판류) , Brick(벽돌류) , Shn(널판지) , Other
- GarageType : 차고 위치별로 묶어 구분 At ( 집에 붙어있는 ) , Dt ( 집에서 떨어져있는 ) , Others ( 다른 유형들 ) , No ( 차고 없음 )

### | 적은 데이터의 범주값을 합침 |

해당하는 변수들 - RoofStyle , RoofMatl , KitchenAbvGr, TotRmsAbvGrd, Functional, GarageCond

## | 변수 변경/추가 - 2. 연도 관련 변수를 범주화

### | YearBuilt\_grp |

```
# 지어진 년도: 1800 후반, ~1950, ~2000, 2000~  
bin = [-np.inf, 1900, 1951, 2001, np.inf]  
label = ['1800 후반', '~1950', '~2000', '2000~']  
x_train['YearBuilt_grp'] = pd.cut(x_train['YearBuilt'], bins=bin, right=False, labels=label)  
# 기존 변수 제거  
x_train.drop('YearBuilt', axis=1, inplace=True)
```

### | YearRemodAdd\_grp |

```
# 리모델링 년도: ~1950, ~2000, 2000~  
bin = [-np.inf, 1951, 2001, np.inf]  
label = ['~1950', '~2000', '2000~']  
x_train['YearRemodAdd_grp'] = pd.cut(x_train['YearRemodAdd'], bins=bin, right=False, labels=label)  
# 기존 변수 제거  
x_train.drop('YearRemodAdd', axis=1, inplace=True)
```

### | GarageYrBlt |

```
# 범주형 데이터로 판별하여 10년대씩 연도별로 묶어서 분석  
bin = [-np.inf, 1, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000, np.inf]  
label = ['No', '1900년대', '1910년대', '1920년대', '1930년대',  
        '1940년대', '1950년대', '1960년대', '1970년대', '1980년대', '1990년대', '2000년대']  
x_train['GarageYrBlt'] = pd.cut(x_train['GarageYrBlt'], bins=bin, labels=label, right=False)
```

## | 변수 변경/추가 - 3. 숫자형 변수 그룹화

### | BsmFinSF1\_grp |

```
bins = [-np.inf, 1, 500, 1000, 1500, np.inf]
labels = ['없음', '500미만', '1000미만', '1500미만', '1500이상']
x_train['BsmFinSF1_grp'] = pd.cut(
    x_train['BsmFinSF1'], bins, labels=labels, right=False)
```

### | BsmUnfSF\_grp |

```
bins = [-np.inf, 1, 500, 1000, 1500, 2000, np.inf]
labels = ['없음', '500미만', '1000미만', '1500미만', '2000미만', '2000이상']
x_train['BsmUnfSF_grp'] = pd.cut(
    x_train['BsmUnfSF'], bins, labels=labels, right=False)
```

### | WoodDeckSF\_cut |

```
bin = [-np.inf, 1, 200, np.inf]
label = ['No', '200미만', '200이상']
x_train['WoodDeckSF_cut'] = pd.cut(
    x_train['WoodDeckSF'], bins=bin, labels=label, right=False)
```

### | OpenPorchSF\_cut |

```
bin = [-np.inf, 1, 100, np.inf]
label = ['No', '100미만', '100이상']
x_train['OpenPorchSF_cut'] = pd.cut(
    x_train['OpenPorchSF'], bins=bin, labels=label, right=False)
```

### | EnclosedPorch\_cut |

```
bin = [-np.inf, 1, 100, 200, 300, np.inf]
label = ['No', '100미만', '200미만', '300미만', '300이상']
x_train['EnclosedPorch_cut'] = pd.cut(
    x_train['EnclosedPorch'], bins=bin, labels=label, right=False)
```

## | 가변수화

### | 카테고리 데이터 타입 변경 |

```
cat = {'MSSubClass': ['new', 'older', 'other', 'unf', 'split'],
       'MSZoning': ['A', 'C (all)', 'FV', 'RH', 'RL', 'RP', 'RM', 'I'],
       'LotShape': ['reg', 'ir'],
       'LandContour': ['Lvl', 'Bnk', 'HLS', 'Low'],
       'LotConfig': ['Inside', 'FR2', 'Corner', 'FR3', 'CulDSac'],
       'Neighborhood': ['CollgCr', 'Veenker', 'Crawfor', 'NoRidge', 'Mitchel', 'Somerst',
                        'OldTown', 'BrkSide', 'Sawyer', 'NridgHt', 'SawyerW', 'NAmes',
                        'IDOTRR', 'MeadowV', 'Edwards', 'Timber', 'StoneBr', 'ClearCr',
                        'Gilbert', 'NWAmes', 'NPkVill', 'Blmngtn', 'BrDale', 'SWISU', 'Blueste'],
       ...
       'Fence': ['NO', 'MnPrv', 'GdPrv', 'GdWo', 'MnWw'],
       'SaleType': ['WD', 'New', 'COD', 'ConLD', 'ConLI', 'CWD', 'ConLw', 'Con', 'Oth', 'VWD'],
       'SaleCondition': ['Normal', 'Abnorml', 'Partial', 'AdjLand', 'Alloca', 'Family']}

...

'Fence': ['NO', 'MnPrv', 'GdPrv', 'GdWo', 'MnWw'],
'SaleType': ['WD', 'New', 'COD', 'ConLD', 'ConLI', 'CWD', 'ConLw', 'Con', 'Oth', 'VWD'],
'SaleCondition': ['Normal', 'Abnorml', 'Partial', 'AdjLand', 'Alloca', 'Family']}

for k, v in cat.items():
    x_train[k] = pd.Categorical(x_train[k],
                              categories=v, ordered=False)
```

- Category 타입으로 변경하지 않고 가변수화를 진행할 경우 범주값이 누락된 채로 될 수도 있기때문에 Category 타입 변경을 먼저 실시
- 범주값이 숫자로 된 변수 중 그 값들이 동일한 가중치로 증감한다고 판단한 변수들은 가변수화에서 제외

\* 가변수화에서 제외한 변수들  
BsmtFullBath, BsmtHalfBath, BedroomAbvGr, Fireplaces, GarageCars

### | 범주형 변수 가변수화 |

```
cat_cols = [col for col in x_train.columns if x_train[col].dtype != 'float64' if x_train[col].dtype != 'int64']
x_train = pd.get_dummies(x_train, columns=cat_cols, drop_first=True)
x_train.tail(1)
```



## | train validation 나누기 | 스케일링

### | X, Y 전처리 전에 분리 |

```
target = 'SalePrice'

X = data.drop(target, axis=1)
y = data[target]
```

- X와 y를 언제 나눌 지에 대해 고민
- 이상치는 이슈로 보고 이상치를 먼저 제거한 후 x와 y를 분리
- 실무적인 관점에서 생각해봤을 때 y값이 포함된 val / test셋이 따로 없고 x값에 대해서 바로 예측해야할 것으로 예상하여 다른 전처리보다 x와 y를 먼저 분리하기로 결정

### | train, val 데이터 셋 나누기 |

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(
    x, y, test_size=0.2, random_state=2022)
```

- 데이터가 충분히 많지 않다고 판단하여 test\_size를 0.2로 설정
- 스터디원들과 분업(동일한 값을 확인하기 위해)하기 때문에 난수값인 random\_state는 임의로 2022로 고정

### | MinMaxScaler |

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train_s = scaler.fit_transform(x_train)
```

- KNN, SVM등의 정규화가 필요한 모델들을 위해 정규화를 진행
- 이상치 제거를 했기 때문에 정규화로 진행

# | Linear Regression

```
1 # 선언하기
2 lr = LinearRegression()
```

```
1 # 학습하기
2 lr.fit(x_train, y_train)
```

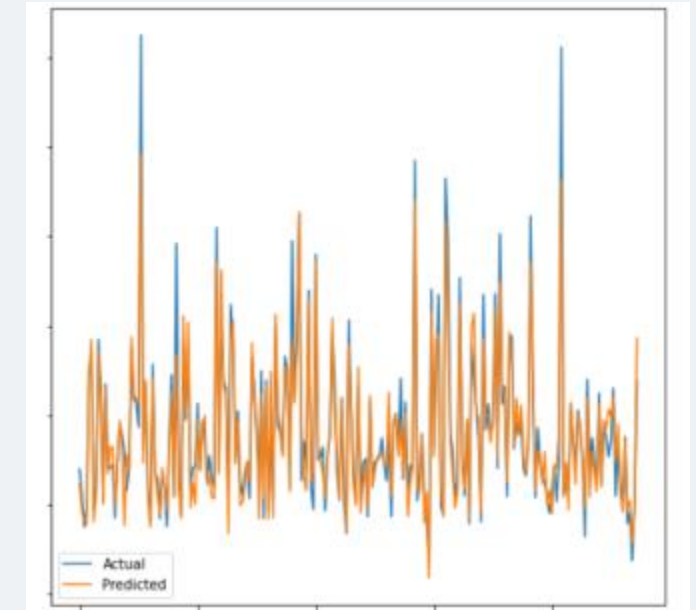
LinearRegression()

```
1 # 예측하기
2 lr_pred = lr.predict(x_val)
```

```
1 # 평가하기
2 print('MSE:', mean_squared_error(y_val, lr_pred))
3 print('RMSE:', math.sqrt(mean_squared_error(y_val, lr_pred)))
4 print('R2-Score:', r2_score(y_val, lr_pred))
```

MSE: 796069855.0227677  
RMSE: 28214.709904990475  
R2-Score: 0.9029058334160777

|                    | coefficient | abs_coefficient |
|--------------------|-------------|-----------------|
| <b>Id</b>          | 57.994362   | 57.994362       |
| <b>MSSubClass</b>  | 57.994362   | 57.994362       |
| <b>GarageType</b>  | 57.994362   | 57.994362       |
| <b>FireplaceQu</b> | 57.994362   | 57.994362       |
| <b>Fireplaces</b>  | 57.994362   | 57.994362       |



- MSE: 796069855.0227677
- RMSE: 28214.709904990475
- R2-Score: 0.9029058334160777
- 실제 값과 예측 값의 차이가 적음

# Decision Tree

```
1 # 선언하기 - max_depth, random_state 속성을 줄 수 있음
2 dt = DecisionTreeRegressor(max_depth=3)
```

```
1 # 학습하기
2 dt.fit(x_train, y_train)
```

```
DecisionTreeRegressor(max_depth=3)
```

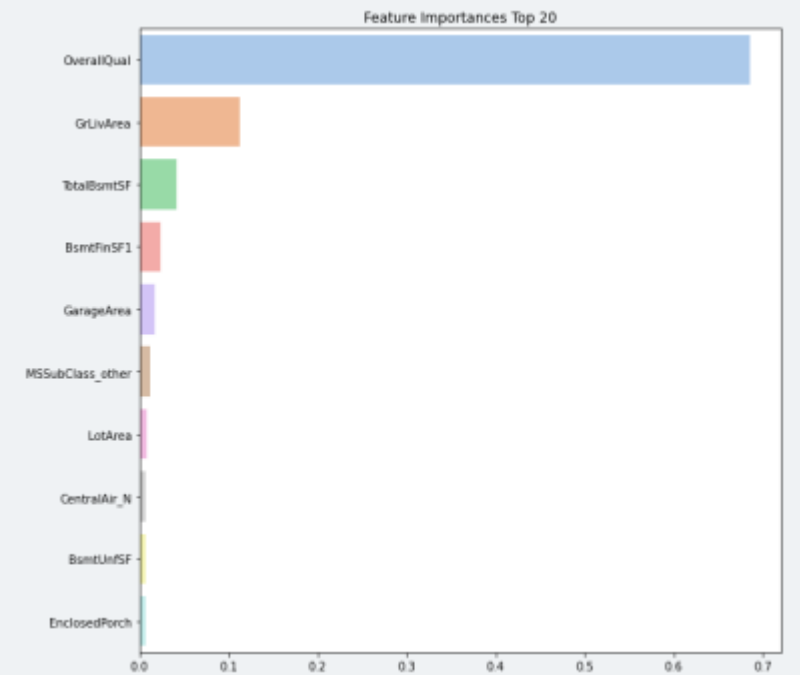
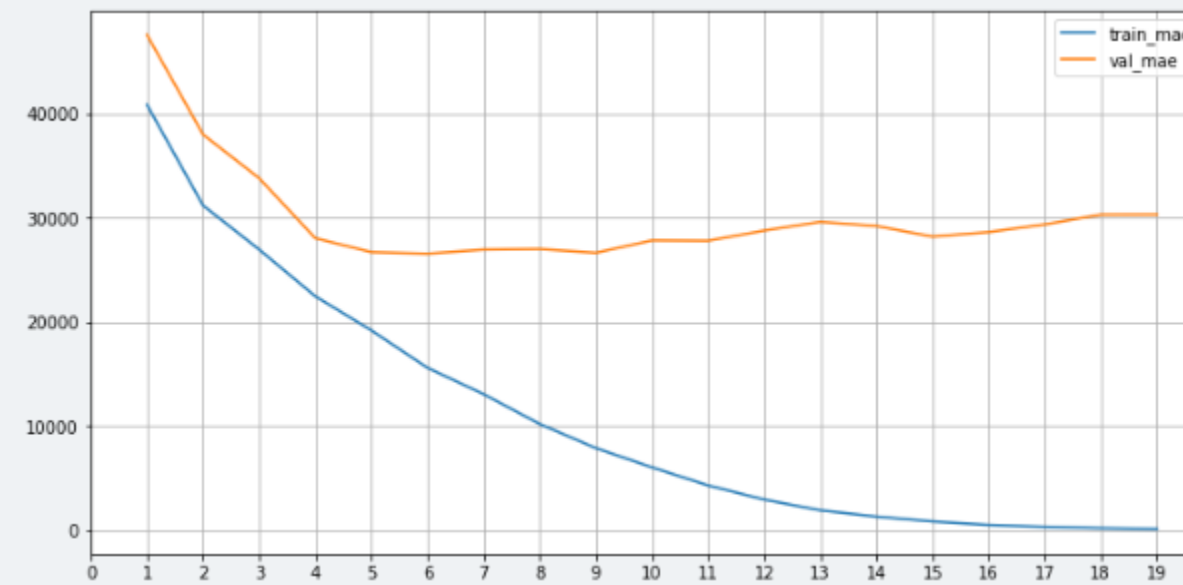
```
1 # 예측
2 dt_pred = dt.predict(x_val)
```

```
1 # 평가
2 print('r2 score:', r2_score(y_val, dt_pred))
3 print('RMSE:', math.sqrt(mean_squared_error(y_val, dt_pred)))
4 print('mse:', mean_squared_error(y_val, dt_pred))
```

r2 score: 0.7283200305695912

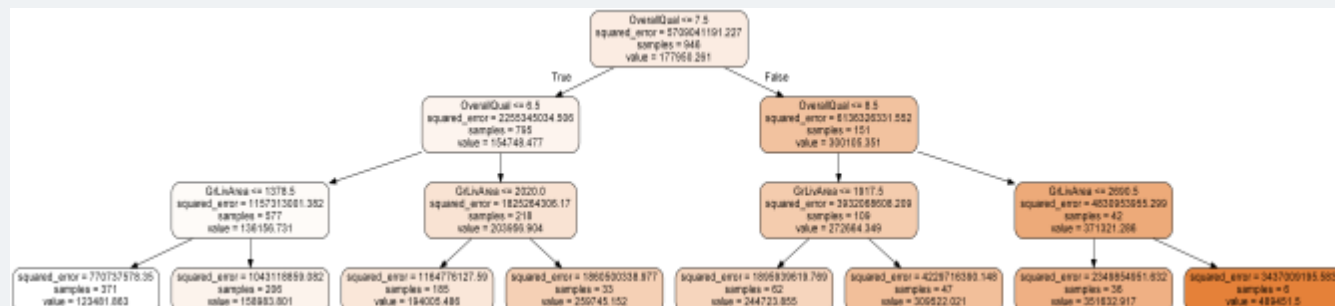
RMSE: 47196.28663563645

mse: 2227489472.1931553



- MSE: 2227489472.1931553
- RMSE: 47196.28663563645
- R2-Score: 0.7283200305695912

- max\_depth를 3으로 설정
- 사회과학 문제이므로 일반적인 기준(0.3)에 적용하면 유의미하다고 판단됨



# | KNN 및 튜닝

## RandomizedSearchCV

```

1 param = {'n_neighbors': range(3, 51)}
2 knn_t = RandomizedSearchCV(knn,
3                             param,
4                             cv=10,
5                             n_iter=30,
6                             scoring='r2',
7                             n_jobs=-1)
8
9 # 학습하기
10 knn_t.fit(x_train_s, y_train)
11
12 RandomizedSearchCV(cv=10, estimator=KNeighborsRegressor(), n_iter=30, n_jobs=-1,
13                    param_distributions={'n_neighbors': range(3, 51)},
14                    scoring='r2')
15
16 # mean_test_score 확인
17 print(knn_t.cv_results_['mean_test_score'])
18 print('-'*20)
19
20 # 최적 파라미터
21 print('최적파라미터:', knn_t.best_params_)
22
23 # 최고 성능
24 print('최고성능:', knn_t.best_score_)
25
26 [0.75858856 0.74142509 0.71800902 0.73269178 0.76404716 0.71006566
27  0.72456134 0.71146021 0.7448121  0.76302454 0.73899918 0.70795228
28  0.71562209 0.69905369 0.76448686 0.71900642 0.75629389 0.69743099
29  0.70399292 0.75429846 0.72641988 0.71952322 0.70080125 0.7219713
30  0.75897941 0.7581342  0.7498163  0.76000593 0.72874079 0.73747765]
31
32 최적파라미터: {'n_neighbors': 7}
33 최고성능: 0.7644868585990279

```

- 최적 파라미터: n\_neighbors 7
- 최고 성능: 0.7644868585990279

## GridSearchCV

```

1 # 변수 선언
2 param = {'n_neighbors': range(3, 10),
3          'leaf_size': range(5, 35)}
4
5 # 선언하기
6 knn_t2 = GridSearchCV(knn,
7                        param,
8                        cv=5,
9                        n_jobs=-1)
10
11 # 학습하기
12 knn_t2.fit(x_train_s, y_train)
13
14 GridSearchCV(cv=5, estimator=KNeighborsRegressor(), n_jobs=-1,
15              param_grid={'leaf_size': range(5, 35),
16                           'n_neighbors': range(3, 10)})
17
18 # 선택된 파라미터와 성능
19 print(knn_t2.best_params_)
20 print(knn_t2.best_score_)
21
22 {'leaf_size': 5, 'n_neighbors': 5}
23 0.7626146816804259

```

- 최적 파라미터: leaf\_size 5, n\_neighbors 5
- 최고 성능: 0.7626146816804259

- MSE: 2154209275.353418
- RMSE: 46413.4600665951
- R2-Score: 0.7372577884740864

# | SVM 및 튜닝

## GridSearchCV

```

1 # 선언
2 svm = SVR()

1 # 학습
2 svm.fit(x_train_s, y_train)

SVR()

1 # 예측
2 svm_pred = svm.predict(x_val_s)

1 # 평가      # 음수 -> 평균으로 예측하는 것보다 쓰레기 모델
2 print('r2 score:', r2_score(y_val, svm_pred))
3 print('RMSE: ', math.sqrt(mean_squared_error(y_val, svm_pred)))
4 print('mse:', mean_squared_error(y_val, svm_pred))

r2 score: -0.06294844914687836
RMSE: 93354.47082550317
mse: 8715057223.109724

```

- MSE: 8715057223.109724
- RMSE: 93354.47082550317
- R2-Score: -0.06294844914687836

```

1 # 선언
2 params = {'C': [0.01, 0.1, 1, 10, 100], 'gamma': [0.01, 0.1, 1, 10, 100]}
3 svm_t = GridSearchCV(svm, param_grid = params, cv = 5, scoring = 'neg_mean_squared_error')

1 # 학습
2 svm_t.fit(x_train_s, y_train)

GridSearchCV(cv=5, estimator=SVR(),
              param_grid={'C': [0.01, 0.1, 1, 10, 100],
                           'gamma': [0.01, 0.1, 1, 10, 100]},
              scoring='neg_mean_squared_error')

1 # 최적 파라미터 확인
2 svm_t.best_params_

{'C': 100, 'gamma': 0.01}

1 # 예측
2 svm_pred = svm_t.predict(x_val_s)

1 # 평가
2 print('r2 score:', r2_score(y_val, svm_pred))
3 print('RMSE: ', math.sqrt(mean_squared_error(y_val, svm_pred)))
4 print('mse:', mean_squared_error(y_val, svm_pred))

r2 score: -0.028142420298623794
RMSE: 91813.31310156196
mse: 8429684462.685448

```

- 최적 파라미터: C 100, gamma 0.01
- MSE: 8429684462.685448
- RMSE: 91813.31310156196
- R2-Score: -0.028142420298623794



# | Random Forest 및 튜닝

## 최적의 max\_depth 찾기

```
1 # 선언하기
2 rf = RandomForestRegressor(random_state=2022)
```

```
1 # 학습하기
2 rf.fit(x_train, y_train)
```

RandomForestRegressor(random\_state=2022)

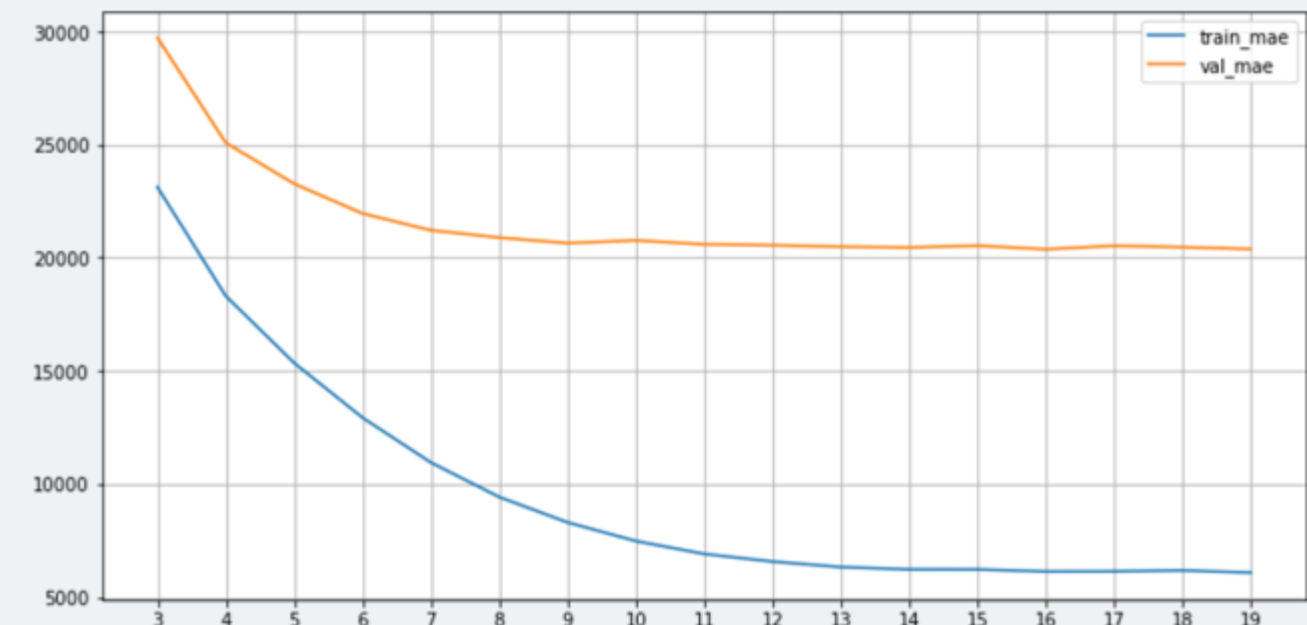
```
1 # 예측하기
2 rf_pred = rf.predict(x_val)
```

```
1 # 평가하기
2 print('MSE:', mean_squared_error(y_val, rf_pred))
3 print('RMSE: ', math.sqrt(mean_squared_error(y_val, rf_pred)))
4 print('R2-score:', r2_score(y_val, rf_pred))
```

MSE: 1082210645.025963  
RMSE: 32896.9701496348  
R2-score: 0.8680061303865856

- MSE: 1082210645.025963
- RMSE: 32896.9701496348
- R2-Score: 0.8680061303865856

```
1 train_mae, val_mae = [], []
2
3 depth = list(range(3, 20))
4 for d in depth:
5     rf_t = RandomForestRegressor(max_depth = d, random_state=2022)
6     rf_t.fit(x_train, y_train)
7     rf_tr_pred = rf_t.predict(x_train) # train 에러 계산을 위해 x_train도 예측
8     rf_pred = rf_t.predict(x_val)
9     train_mae.append(mean_absolute_error(y_train, rf_tr_pred))
10    val_mae.append(mean_absolute_error(y_val, rf_pred))
11    print(d)
```



# | Random Forest 및 튜닝

## GridSearchCV

```
1 param = {'max_depth':range(1, 6), # 5 기준으로 앞뒤 2~3개
2         'n_estimators':range(1,121,10)}
3
4 rf_r = RandomForestRegressor(max_depth=3, random_state=2022)
5
6 rf_g = GridSearchCV(rf_r,
7                     param,
8                     cv=10,
9                     n_jobs=-1)
10
11 rf_g.fit(x_train, y_train)
12
13 GridSearchCV(cv=10,
14             estimator=RandomForestRegressor(max_depth=3, random_state=2022),
15             n_jobs=-1,
16             param_grid={'max_depth': range(1, 6),
17                         'n_estimators': range(1, 121, 10)})
18
19 print('최적 파라미터:', rf_g.best_params_)
20 print('최고 성능:', rf_g.best_score_)
21
22 최적 파라미터: {'max_depth': 5, 'n_estimators': 31}
23 최고 성능: 0.8573397799251568
```

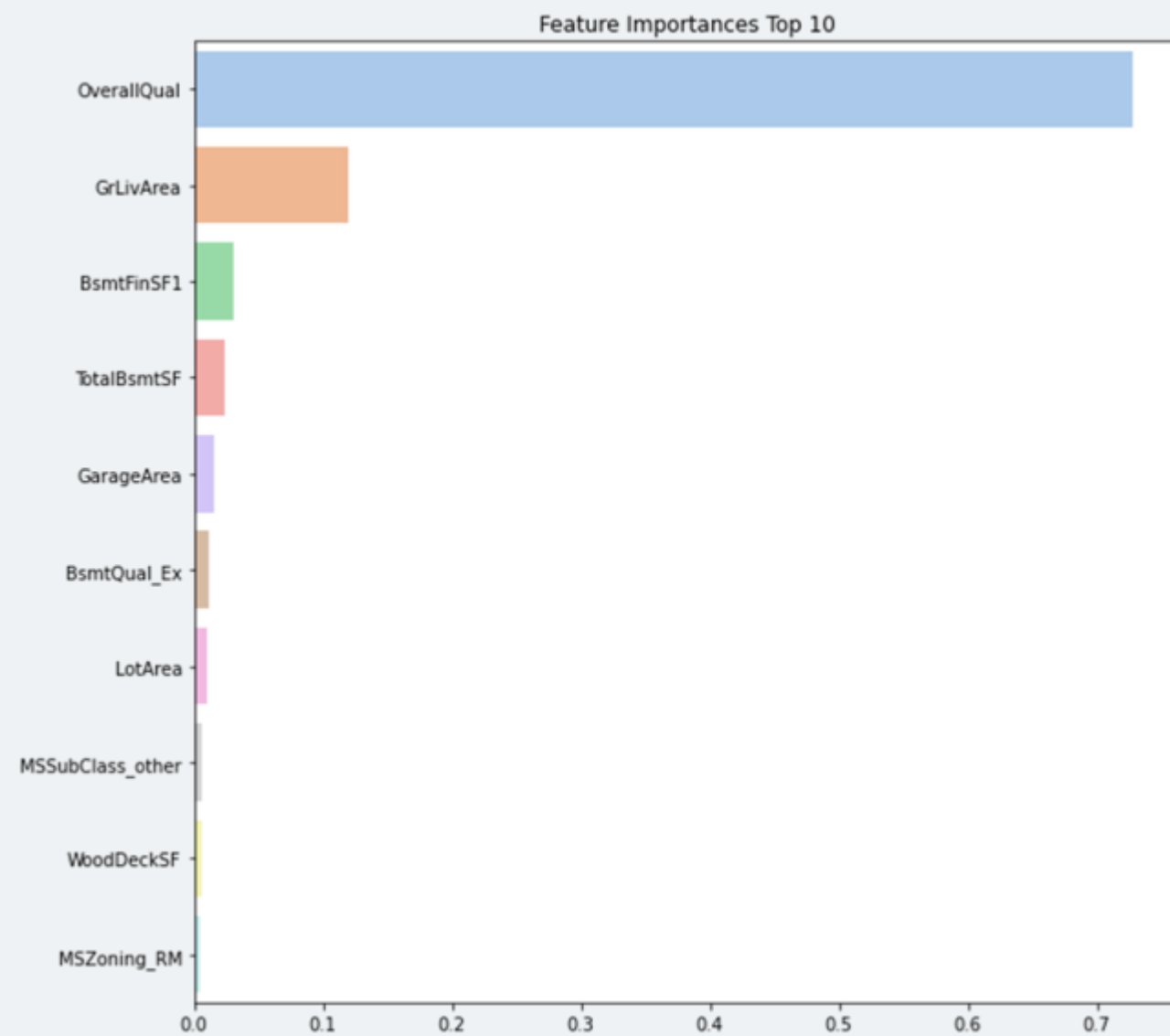
- 최적 파라미터: max\_depth 5,  
n\_estimators 31
- 최고 성능: 0.8573397799251568

```
1 # 선언
2 rf_t = RandomForestRegressor(max_depth=5,
3                             n_estimators=31,
4                             random_state=2022)
5
6 rf_t.fit(x_train, y_train)
7
8 RandomForestRegressor(max_depth=5, n_estimators=31, random_state=2022)
9
10 # 예측
11 rf_t_pred = rf_t.predict(x_val)
12
13 # 평가
14 print('MSE:', mean_squared_error(y_val, rf_t_pred))
15 print('RMSE: ', math.sqrt(mean_squared_error(y_val, rf_t_pred)))
16 print('R2-score:', r2_score(y_val, rf_t_pred))
17
18 MSE: 1297931608.5422838
19 RMSE: 36026.81790752944
20 R2-score: 0.841695314777698
```

- MSE: 1297931608.5422838
- RMSE: 36026.81790752944
- R2-score: 0.841695314777698



## | Random Forest 및 튜닝



### 변수 중요도

- OverallQual
- GrLivArea
- BsmtFinSF

순으로 변수 중요도를 차지하고 있음

# | XGBoost 및 튜닝

```
1 # 선언하기
2 xgb = XGBRegressor(random_state=2022)
```

```
1 # 학습하기
2 xgb.fit(x_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=2022,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
1 # 예측하기
2 xgb_pred = xgb.predict(x_val)
```

```
1 # 평가하기
2 print('MSE:', mean_squared_error(y_val, xgb_pred))
3 print('RMSE:', math.sqrt(mean_squared_error(y_val, xgb_pred)))
4 print('R2-score:', r2_score(y_val, xgb_pred))
```

```
MSE: 1125782943.141482
RMSE: 33552.689059768105
R2-score: 0.8626917525779302
```

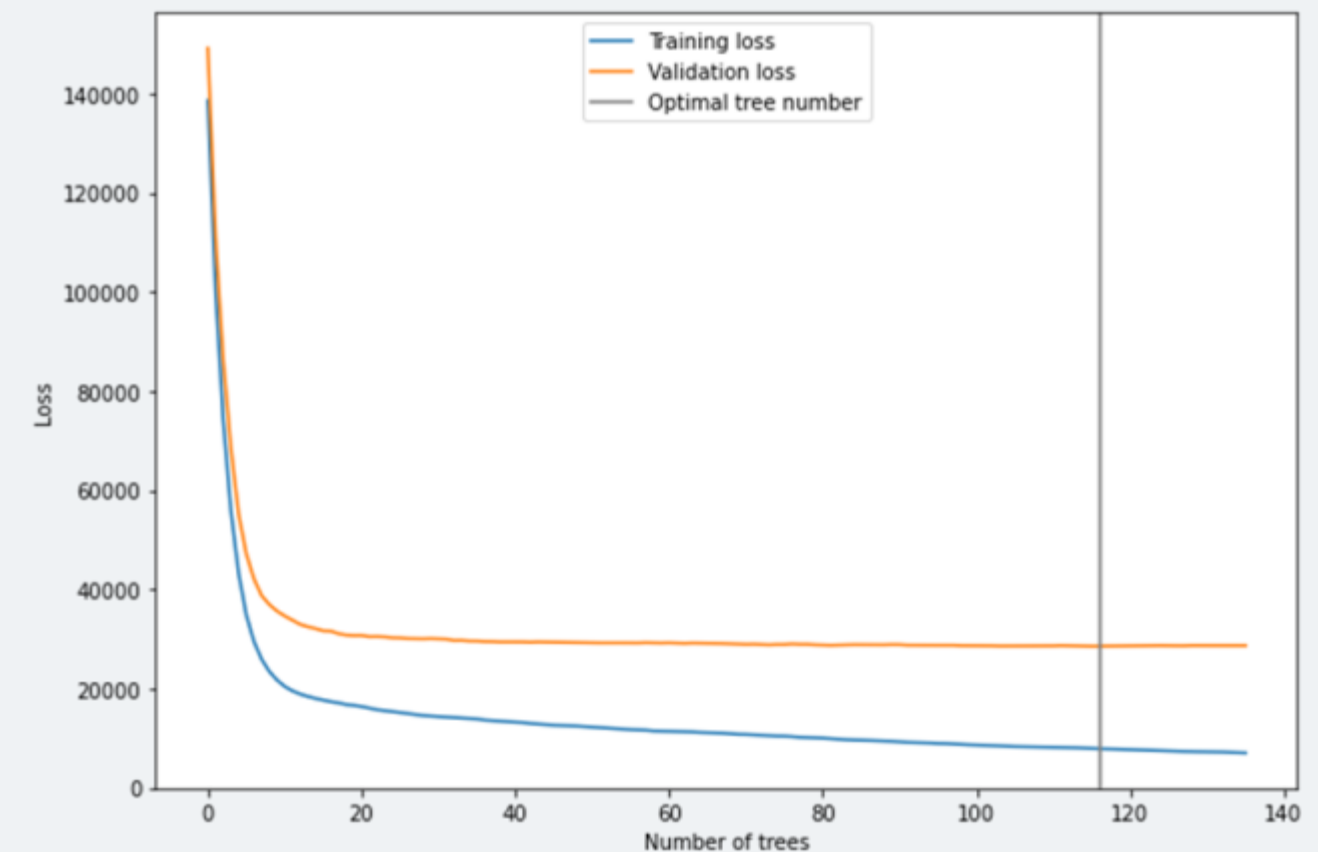
- MSE: 1125782943.141482
- RMSE: 33552.689059768105
- R2-Score: 0.8626917525779302

## 최적의 파라미터 조정

```
1 # 선언
2 xgbr = XGBRegressor(n_estimators=500, max_depth=3, objective='reg:squarederror', random_state=2022)
```

```
1 x_val.shape
(237, 233)
```

```
1 # 학습
2 xgbr.fit(x_train, y_train, eval_metric="rmse", eval_set=[(x_train, y_train), (x_val, y_val)],
3         early_stopping_rounds=20, verbose=True)
```



# | XGBoost 및 튜닝

## 최고 성능

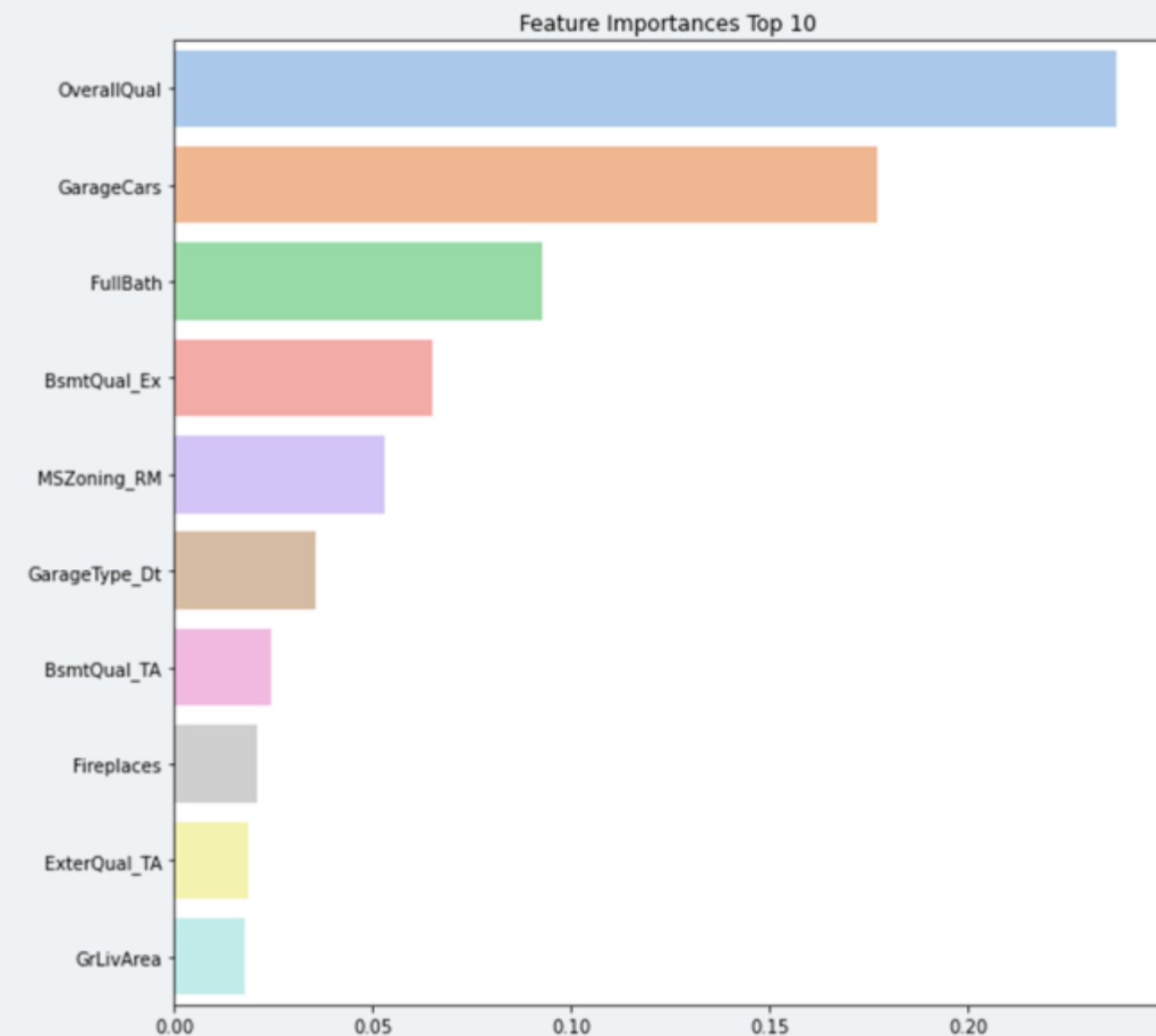
```
1 # 예측
2 xgbr_pred = xgbr.predict(x_val)

1 # 평가
2 print('MSE:', mean_squared_error(y_val, xgbr_pred))
3 print('RMSE:', math.sqrt(mean_squared_error(y_val, xgbr_pred)))
4 print('R2-score:', r2_score(y_val, xgbr_pred))
```

MSE: 812831740.0635613  
RMSE: 28510.204139282505  
R2-score: 0.90086143838197

- MSE: 812831740.0635613
- RMSE: 28510.204139282505
- R2-Score: 0.90086143838197

## 변수 중요도



## | DL1 ( Fully connected Model )

```
# 체인처럼 레이어 연결
il = Input(shape=(220,), name='input_layer')
# 51에 제일 인접한 2의 제곱수 32를 노드 수로 선택
hl = Dense(256, activation='relu', name='hidden_layer1')(il)
hl = Dense(256, activation='relu', name='hidden_layer2')(hl)
hl = Dense(128, activation='relu', name='hidden_layer3')(hl)
hl = Dense(128, activation='relu', name='hidden_layer4')(hl)
hl = Dense(64, activation='relu', name='hidden_layer5')(hl)
hl = Dense(64, activation='relu', name='hidden_layer6')(hl)
hl = Dense(32, activation='relu', name='hidden_layer7')(hl)
hl = Dense(32, activation='relu', name='hidden_layer8')(hl)
hl = Dense(16, activation='relu', name='hidden_layer9')(hl)
ol = Dense(1, name='output_layer')(hl)
```

```
# 모델 선언 (시작과 끝 지정)
d11 = Model(il, ol)
```

```
# 컴파일
d11.compile(loss=keras.losses.mean_squared_error,
            optimizer=Adam(learning_rate=0.01))
```

...

```
es = EarlyStopping(monitor='val_loss',
                  min_delta=0,
                  patience=10,
                  verbose=1,
                  restore_best_weights=True)
```

```
# d11 학습하기
d11.fit(x_train, y_train, validation_split=0.2, epochs=1000, verbose=1, callbacks=[es])
```

## | 수치 검정 |

- $R^2 = 0.8431342491350231$

## | 모델링 이유 |

51에 가장 인접한 2의 제곱수인 32를 노드 수로 선택하였으나, 정확도가 떨어져 제곱수를 높여나가 256으로 결정함

## | DL2 ( Fully connected Model )

```
# 모델의 레이어를 엮기
il = Input(shape=(220,), name='input_layer')
hl = Dense(512, activation='relu', name='hidden_layer1')(il)
hl = Dense(256, activation='relu', name='hidden_layer2')(hl)
hl = Dense(128, activation='relu', name='hidden_layer3')(hl)
hl = Dense(64, activation='relu', name='hidden_layer4')(hl)
hl = Dense(64, activation='relu', name='hidden_layer5')(hl)
hl = Dense(32, activation='relu', name='hidden_layer6')(hl)
hl = Dense(32, activation='relu', name='hidden_layer7')(hl)
hl = Dense(16, activation='relu', name='hidden_layer8')(hl)
ol = Dense(1, name='output_layer')(hl)

# 모델의 처음과 끝 지정
dl2 = Model(il, ol)

# 컴파일
dl2.compile(loss='mse', optimizer = 'adam')
```

...

```
# 학습
dl2.fit(x_train, y_train, epochs=1000, verbose=1, validation_split=0.15, callbacks=[es])
```

## | 수치 검증 |

- $R^2 = 0.8410952626167307$

## | 모델링 이유 |

- 기존 DL1 모델보다 노드 수를 늘려 512부터 시작함
- 둘을 비교했을 때, 무엇이 높은 지 비교하기 위해 결정

## | DL3 ( Fully connected Model )

```
# 체인처럼 레이어 연결
il = Input(shape=(220,), name='input_layer')
hl = Dense(128, activation='relu', name='hidden_layer1')(il)
hl = Dense(128, activation='relu', name='hidden_layer2')(hl)
hl = Dense(128, activation='relu', name='hidden_layer3')(hl)
hl = Dense(128, activation='relu', name='hidden_layer4')(hl)
hl = Dense(128, activation='relu', name='hidden_layer5')(hl)
ol = Dense(1, name='output_layer')(hl)

# 모델선언(시작과 끝 지정)
dl3 = Model(il, ol)

# 컴파일
dl3.compile(loss = 'mse', optimizer=Adam())
```

...

```
dl3.fit(x_train, y_train, validation_split=0.2,
        epochs=1000, verbose=1, callbacks=[es])
```

## | 수치 검정 |

- $R^2 = 0.8625568559611131$

## | 모델링 이유 |

- 기존 DL1,DL2 모델보다 노드 수를 줄여 128부터 시작함
- 셋을 비교했을 때, 무엇이 높은 지 비교하기 위해 결정



## | DL4 ( Locally connected Model )

```
A = ['LandContour_HLS', 'LandContour_Low', 'LandContour_Lvl',  
     'Condition1_nag', 'Condition1_pos',  
     'LotFrontage', 'LotArea', 'MasVnrArea', 'GrLivArea', 'FullBath',  
     'HalfBath', 'BedroomAbvGr', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch',  
     'ScreenPorch', 'Fireplaces', 'Heating_GasW', 'Heating_Grav',  
     'Heating_OthW', 'Heating_Wall', 'HeatingQC_Fa', 'HeatingQC_Gd',  
     'HeatingQC_Po', 'HeatingQC_TA', 'CentralAir_Y', 'Electrical_FuseF',  
     'Electrical_FuseP', 'Electrical_Mix', 'Electrical_SBrkr', 'KitchenAbvGr_2개 이상',  
     'KitchenQual_Fa', 'KitchenQual_Gd', 'KitchenQual_TA',  
     'TotRmsAbvGrd_5~8', 'TotRmsAbvGrd_9~',  
     'FireplaceQu_Fa', 'FireplaceQu_Gd', 'FireplaceQu_No', 'FireplaceQu_Po',  
     'FireplaceQu_TA', 'PoolQC_Gd', 'PoolQC_NO', 'Fence_GdWo', 'Fence_MnPrv',  
     'Fence_MnWo', 'Fence_NO', 'WoodDeckSF_cut_200미만', 'WoodDeckSF_cut_200이상',  
     'OpenPorchSF_cut_100미만', 'OpenPorchSF_cut_100이상',  
     'EnclosedPorch_cut_100미만', 'EnclosedPorch_cut_200미만',  
     'EnclosedPorch_cut_300미만', 'EnclosedPorch_cut_300이상']
```

...

```
D = ['YearBuilt', 'LotConfig_CulDSac', 'LotConfig_FR2', 'LotConfig_FR3',  
     'LotConfig_Inside', 'Neighborhood_Blueste', 'Neighborhood_BrDale',  
     'Neighborhood_BrkSide', 'Neighborhood_ClearCr', 'Neighborhood_CollgCr',  
     'Neighborhood_Crawfor', 'Neighborhood_Edwards', 'Neighborhood_Gilbert',  
     'Neighborhood_IDOTRR', 'Neighborhood_MeadowV', 'Neighborhood_Mitchel',  
     'Neighborhood_NAmes', 'Neighborhood_NPKVill', 'Neighborhood_NWAmes',  
     'Neighborhood_NoRidge', 'Neighborhood_NridgHt', 'Neighborhood_OldTown',  
     'Neighborhood_SWISU', 'Neighborhood_Sawyer', 'Neighborhood_SawyerW',  
     'Neighborhood_Somerst', 'Neighborhood_StoneBr', 'Neighborhood_Timber',  
     'Neighborhood_Veenker', 'GarageYrBlt_1900년대', 'GarageYrBlt_1910년대',  
     'GarageYrBlt_1920년대', 'GarageYrBlt_1930년대', 'GarageYrBlt_1940년대',  
     'GarageYrBlt_1950년대', 'GarageYrBlt_1960년대', 'GarageYrBlt_1970년대',  
     'GarageYrBlt_1980년대', 'GarageYrBlt_1990년대', 'GarageYrBlt_2000년대',  
     'BldgType_2fmCon', 'BldgType_Duplex', 'BldgType_Twnhs', 'BldgType_TwnhsE',  
     'HouseStyle_sto2', 'HouseStyle_unf', 'RoofStyle_Hip', 'RoofStyle_Others',  
     'RoofMatl_Others', 'RoofMatl_Wood', 'YearBuilt_grp_~1950', 'YearBuilt_grp_~2000',  
     'YearBuilt_grp_2000~', 'YearRemodAdd_grp_~2000', 'YearRemodAdd_grp_2000~']
```

## | 기준 설정 |

A : 지상, 주방, 난방(벽난로), 풀, 울타리

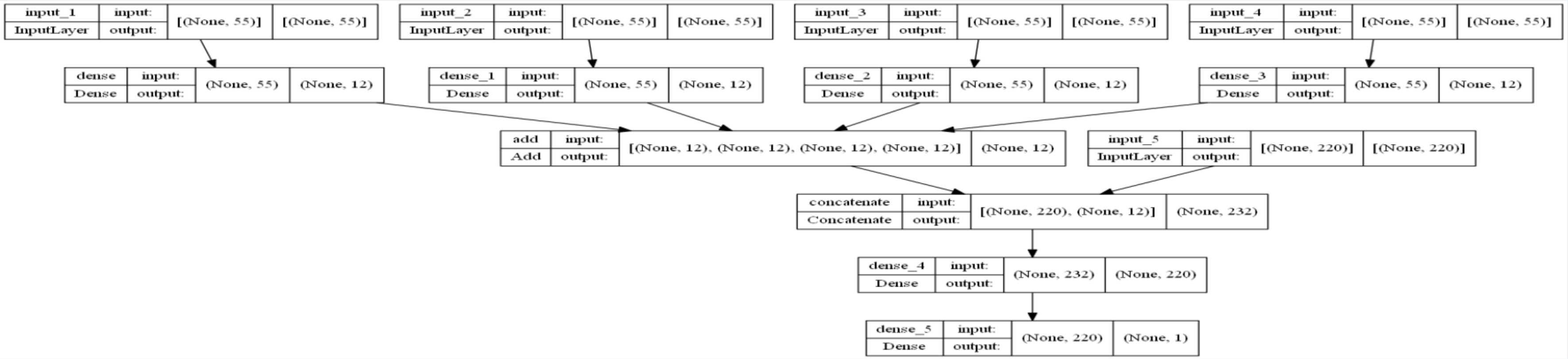
B : 지하, 차고(피트, 개수)

C : 자재 및 상태(퀄리티), 유형(형태)

D : 날짜, 평탄도, 부지 구성, 위치



| DL4 ( Locally connected Model )



| 기준 설정 |

A : 지상, 주방, 난방(벽난로), 풀, 울타리  
B : 지하, 차고(피트, 개수)

C : 자재 및 상태(퀄리티), 유형(형태)  
D : 날짜, 평탄도, 부지 구성, 위치

## | DL4 ( Locally connected Model )

```
# 2. 레이어 사슬처럼 엮기 : Input 2개!
il_A = Input(shape=(55,))
hl_A = Dense(12, activation='relu')(il_A)

il_B = Input(shape=(55,))
hl_B = Dense(12, activation='relu')(il_B)

il_C = Input(shape=(55,))
hl_C = Dense(12, activation='relu')(il_C)

il_D = Input(shape=(55,))
hl_D = Dense(12, activation='relu')(il_D)

add_l = Add()([hl_A, hl_B, hl_C, hl_D])

il = Input(shape=(220,))

co_l = Concatenate()([il, add_l])
hl = Dense(220, activation='relu')(co_l)
ol = Dense(1)(hl)

# 3. 모델 시작과 끝 지정
dl4 = Model([il_A, il_B, il_C, il_D, il], ol)

# 4. 모델 컴파일
dl4.compile(optimizer=Adam(), loss='mse')
```

...

```
# 학습하기
dl4.fit([tr_A, tr_B, tr_C, tr_D, x_train],
        y_train, validation_split=0.3, epochs=1000, verbose=1, callbacks=[es])
```

### | 수치 검정 |

- $R^2 = 0.8970018385272847$

### | 모델링 이유 |

- 집값에 영향을 주는 요인을 4가지로 구분  
A(지상요인), B(지하, 차고), C(품질, 유형), D(그 외)
- 요인끼리 묶어준 후, 전체와 다시 합치는 구조로 모델링

## | DL5 ( Locally connected Model )

```
# 집유형
house = ['MSSubClass_older', 'MSSubClass_other', 'MSSubClass_split', 'MSSubClass_unf',
        'MSZoning_FV', 'MSZoning_RH', 'MSZoning_RL', 'MSZoning_RM', 'LotShape_reg',
        'LandContour_HLS', 'LandContour_Low', 'LandContour_Lvl', 'LotConfig_CulDSac',
        'LotConfig_FR2', 'LotConfig_FR3', 'LotConfig_Inside', 'BldgType_2fmCon',
        'BldgType_Duplex', 'BldgType_Twnhs', 'BldgType_TwnhsE', 'HouseStyle_sto2',
        'HouseStyle_unf', 'RoofStyle_Hip', 'RoofStyle_Others', 'ExterQual_Fa',
        'ExterQual_Gd', 'ExterQual_TA', 'ExterCond_Fa', 'ExterCond_Gd', 'ExterCond_Po',
        'ExterCond_TA', 'YearBuilt', 'YearBuilt_grp_~1950', 'YearBuilt_grp_~2000',
        'YearBuilt_grp_2000~', 'YearRemodAdd_grp_~2000', 'YearRemodAdd_grp_2000~',
        'PoolQC_Gd', 'PoolQC_NO', 'Fence_GdWo', 'Fence_MnPrv', 'Fence_MnWw',
        'Fence_NO', 'OpenPorchSF_cut_100미만', 'OpenPorchSF_cut_100이상',
        'EnclosedPorch_cut_100미만', 'EnclosedPorch_cut_200미만', 'EnclosedPorch_cut_300미만',
        'EnclosedPorch_cut_300이상', 'KitchenAbvGr_2개 이상', 'KitchenQual_Fa',
        'KitchenQual_Gd', 'KitchenQual_TA', 'FireplaceQu_Fa', 'FireplaceQu_Gd',
        'FireplaceQu_No', 'FireplaceQu_Po', 'FireplaceQu_TA', 'Heating_GasW',
        'Heating_Grav', 'Heating_OthW', 'Heating_Wall', 'HeatingQC_Fa', 'HeatingQC_Gd',
        'HeatingQC_Po', 'HeatingQC_TA', 'GrLivArea', 'BedroomAbvGr', 'Fireplaces',
        'OpenPorchSF', 'EnclosedPorch', 'ScreenPorch', 'CentralAir_Y', 'Electrical_FuseF',
        'Electrical_FuseP', 'Electrical_Mix', 'Electrical_SBrkr',
        'Functional_Min', 'Functional_Typ']
```

...

```
# 판매관련
sale=['SaleType_CWD', 'SaleType_Con', 'SaleType_ConLD', 'SaleType_ConLI', 'SaleType_ConLw',
      'SaleType_New', 'SaleType_Oth', 'SaleType_WD', 'SaleCondition_AdjLand',
      'SaleCondition_Alloca', 'SaleCondition_Family', 'SaleCondition_Normal',
      'SaleCondition_Partial']
```

## | 기준 설정 |

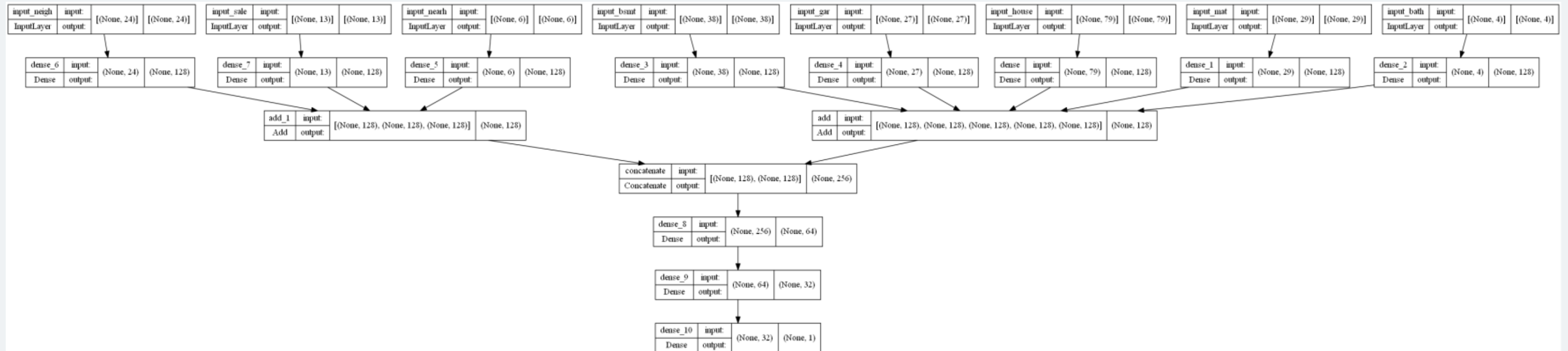
### [ 내부 요인 ]

- house : 집 스타일, 건설 연도, 집 옵션 등  
집의 유형이나 옵션 관련을 묶음
- mat : 집 건설에 사용된 자료를 묶음
- bath : 화장실 관련을 묶음
- bsmt : 지하실 관련을 묶음
- gar : 차고 관련을 묶음

### [ 외부 요인 ]

- nearth : 집 주변의 상태나 요인을 묶음
- neigh : 집의 지역 요인을 묶음
- sale : 집 판매 관련 요인을 묶음

## | DL5 ( Locally connected Model )



### | 기준 설정 | [ 내부 요인 ]

- house : 집 스타일, 건설 연도, 집 옵션 등  
집의 유형이나 옵션 관련을 묶음
- mat : 집 건설에 사용된 자료를 묶음
- bath : 화장실 관련을 묶음
- bsmt : 지하실 관련을 묶음
- gar : 차고 관련을 묶음

### [ 외부 요인 ]

- nearth : 집 주변의 상태나 요인을 묶음
- neigh : 집의 지역 요인을 묶음
- sale : 집 판매 관련 요인을 묶음

# | DL5 ( Locally connected Model )

```
# 레이어별 레이어 연결
il_house = Input(shape=(79,), name='input_house')
hl_house = Dense(120, activation='relu')(il_house)

il_mat = Input(shape=(29,), name='input_mat')
hl_mat = Dense(128, activation='relu')(il_mat)

il_bath = Input(shape=(4,), name='input_bath')
hl_bath = Dense(128, activation='relu')(il_bath)

il_bsmt = Input(shape=(34,), name='input_bsmt')
hl_bsmt = Dense(128, activation='relu')(il_bsmt)

il_gar = Input(shape=(27,), name='input_gar')
hl_gar = Dense(128, activation='relu')(il_gar)

il_nearh = Input(shape=(6,), name='input_nearh')
hl_nearh = Dense(128, activation='relu')(il_nearh)

il_neigh = Input(shape=(24,), name='input_neigh')
hl_neigh = Dense(128, activation='relu')(il_neigh)

il_sale = Input(shape=(13,), name='input_sale')
hl_sale = Dense(128, activation='relu')(il_sale)

# hidden layer add : 내부요인과 외부요인으로 구분하여 묶고자 함.
add_1 = Add()([hl_house, hl_mat, hl_bath, hl_bsmt, hl_gar])
add_2 = Add()([hl_nearh, hl_neigh, hl_sale])

# add끼리 연결
cl = Concatenate()([add_1, add_2])
hl = Dense(64, activation='relu')(cl)
hl = Dense(32, activation='relu')(hl)
ol = Dense(1)(hl)
```

```
# d15 학습하기
...
d15.fit([tr_x_house, tr_x_mat, tr_x_bath, tr_x_bsmt, tr_x_gar, tr_x_nearh, tr_x_neigh, tr_x_sale],
        y_train, validation_split=0.3, epochs=1000, verbose=1, callbacks=[es])
```

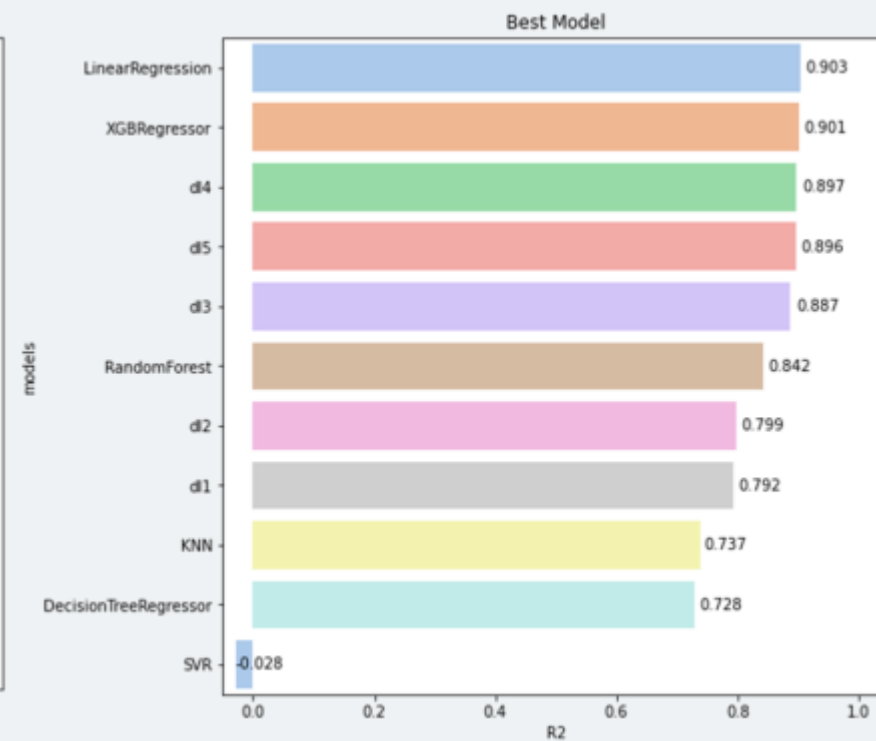
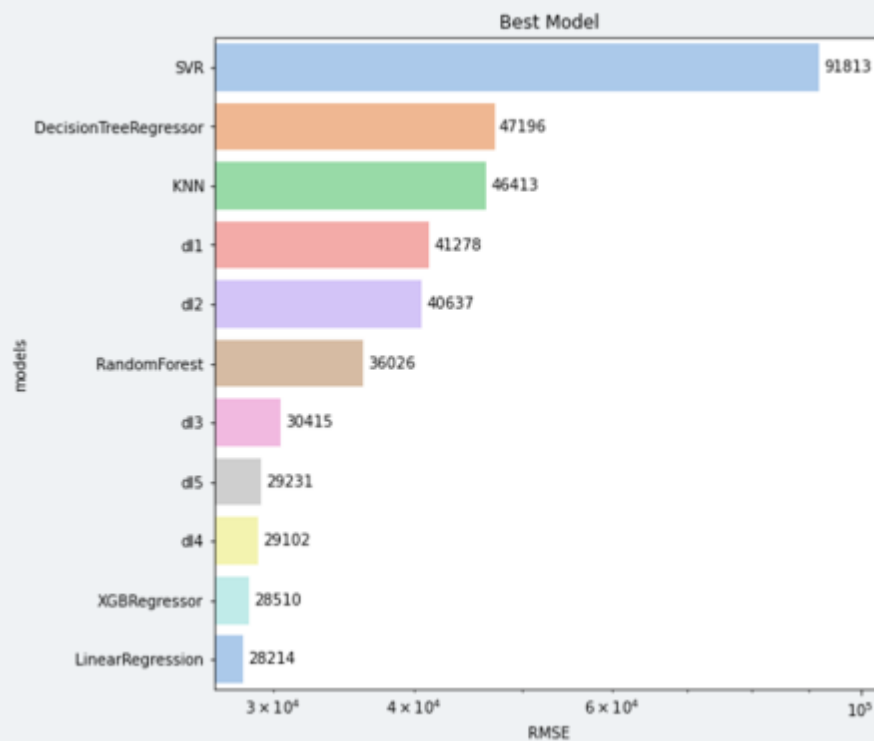
## | 수치 검정 |

- $R^2 = 0.8937901657107363$

## | 모델링 이유 |

- 집값에 영향을 주는 요인을 집 내부요인과 집 외부요인으로 구분
- 내부요인으로 집 건설 자재, 집 인테리어, 집 유형 등을 포함
- 외부요인으로 지역, 판매 관련 요인, 주변 상태 등을 포함
- 내부요인끼리 묶어주고, 외부요인끼리 묶어준 후  
두 요인을 합치는 구조로 모델링

## | 최적모델 선택



- \* 모델 성능을 한 번에 비교하기 위해 평가지표 RMSE를 사용
- \* R2 스코어를 통해 모델들의 정확도를 확인

### <최적모델 선택>

R2 스코어가 0.9이상인 모델 LinearRegerssion과 XGBRegressor를 선택하여 test셋에 적용



## | 전처리 코드 test셋에 적용

### <전처리 코드 함수화>

```
1 # 변수 1~16
2 def fe1(df):
3     temp = df.copy()
4
5     fre_D = temp['LotFrontage'].value_counts().idxmax() # 최빈값
6     temp['LotFrontage'].fillna(fre_D, inplace=True)
7
8     temp = temp.drop('Alley', axis=1)
9     temp = temp.drop('Utilities', axis=1)
10
11     temp['MSSubClass'].loc[temp['MSSubClass'].isin([20,60,120])] = 'new'
12     temp['MSSubClass'].loc[temp['MSSubClass'].isin([30,70,160])] = 'older'
13     temp['MSSubClass'].loc[temp['MSSubClass'].isin([40,45])] = 'unf'
14     temp['MSSubClass'].loc[temp['MSSubClass'].isin([80,85,180])] = 'split'
15     temp['MSSubClass'].loc[temp['MSSubClass'].isin([90,75,90,150,190])] = 'other'
16
17     temp['LotShape'].loc[temp.LotShape == 'Reg'] = 'reg'
18     temp['LotShape'].loc[temp['LotShape'].isin(['IR1','IR2','IR3'])] = 'in'
19
20     temp['LandSlope'].loc[temp.LotShape == 'Gtl'] = 'Gtl'
21     temp['LandSlope'].loc[temp['LandSlope'].isin(['Sev','Mod'])] = 'ste'
22
23     temp['Condition1'].loc[temp.LotShape == 'Norm'] = 'Norm'
24     temp['Condition1'].loc[temp['Condition1'].isin(['PosN','PosA'])] = 'pos'
25     temp['Condition1'].loc[temp['Condition1'].isin(['Feedr','Artery','RRue','RRin','RRAn','RRNe'])] = 'neg'
26
27     temp['HouseStyle'].loc[temp['HouseStyle'].isin(['1Story','1.5Fin'])] = 'sto1'
28     temp['HouseStyle'].loc[temp['HouseStyle'].isin(['2Story','SFoyer','SLvl1','2.5Fin'])] = 'sto2'
29     temp['HouseStyle'].loc[temp['HouseStyle'].isin(['2.5Unf','1.5Unf'])] = 'unf'
30
31
32     return temp
```

### \* 전처리 코드를 함수화하고 Test셋에 적용

- fe1 : 변수 1~16에 해당하는 전처리 함수화
- fe2 : 변수 17~32에 해당하는 전처리 함수화
- fe3 : 변수 33~48에 해당하는 전처리 함수화
- fe4 : 변수 49~64에 해당하는 전처리 함수화
- fe5 : 변수 65~80에 해당하는 전처리 함수화
- del\_fe : 이변량 분석과 다중공선성 분석을 통한 변수 제거 및 고유값 변수 제거 함수화

### <Test셋에 전처리 코드 적용>

```
1 data = pd.read_csv('test.csv')
2 data.tail(2)
```

```
1 data = fe1(data)
2 data = fe2(data)
3 data = fe3(data)
4 data = fe4(data)
5 data = fe5(data)
6 data = del_fe(data)
```

## | 전처리 코드 test셋에 적용

### <결측치 이슈 확인>

```
1 tmp = data.isna().sum()  
2 print(tmp[tmp>0])
```

```
MSZoning      4  
Exterior1st   1  
Exterior2nd   1  
BsmtFinSF1    1  
BsmtUnfSF     1  
TotalBsmtSF   1  
BsmtFullBath  2  
BsmtHalfBath  2  
KitchenAbvGr  2  
KitchenQual   1  
Functional    2  
GarageCars    1  
GarageArea    1  
SaleType      1  
BsmtFinSF1_grp 1  
BsmtUnfSF_grp  1  
dtype: int64
```

### <결측치 조치>

```
1 from sklearn.impute import SimpleImputer  
  
1 imputer_list = ['MSZoning', 'Exterior1st', 'Exterior2nd', 'BsmtFullBath', 'BsmtHalfBath', 'KitchenAbvGr',  
2                 'KitchenQual', 'Functional', 'GarageCars', 'SaleType', 'BsmtFinSF1_grp', 'BsmtUnfSF_grp']  
3 imputer = SimpleImputer(strategy = 'most_frequent')  
4 data[imputer_list] = imputer.fit_transform(data[imputer_list])  
  
1 imputer_list2 = ['BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', 'GarageArea']  
2 imputer2 = SimpleImputer(strategy = 'mean')  
3 data[imputer_list2] = imputer2.fit_transform(data[imputer_list2])
```

- \* 결측치 이슈 확인 후 조치
- \* 결측치 조치 후 (Categorical 후) 가변수화 적용
- \* 가변수화 후 DataFrame으로 변경

### <가변수화>

```
1 data = dumm_fe(data, cat)  
2 data = df(data)  
3 data.tail(2)
```

- 전처리 코드 적용 후 결측치 확인했을 때, 결측치가 없을 것으로 기대했지만 결측치가 많은 열에서 존재함을 확인
- 해당 결측치들을 이슈로써 접근하기로 함
- SimpleImputer를 사용하여 연속형(숫자형)이면 평균으로, 범주형이면 최빈값으로 결측치 조치



## | 예측 및 제출

### <최적모델 이용한 예측>

```
1 lr_pred = lr.predict(data)

1 lr_pred = pd.Series(lr_pred)
2 lr_pred



1 lr_pred.to_csv('predict_lr.csv')
```

```
1 xgbr_pred = xgbr.predict(data)

1 xgbr_pred = pd.Series(xgbr_pred)
2 xgbr_pred

1 xgbr_pred.to_csv('predict_xgbr.csv')
```

### <캐글 제출>

| Submission and Description  |                              |   |         |   | Public Score               |
|---|------------------------------|---|---------|---|----------------------------|
| <a href="#">predict_xgbr.csv</a><br>22 days ago by <a href="#">songjeongwoo</a><br><a href="#">add submission details</a>   |                              |   |         |   | 0.14675                    |
| <a href="#">predict_lr.csv</a><br>22 days ago by <a href="#">songjeongwoo</a><br><a href="#">add submission details</a>   |                              |   |         |   | 0.19483                    |
| 2011  | <a href="#">songjeongwoo</a> |  | 0.14675 | 2 | 22d                        |
|  Your Best Entry!<br>Your most recent submission scored 0.14675, which is an improvement of your previous score of 0.19483. Great job! |                              |   |         |   | <a href="#">Tweet this</a> |

## | 프로젝트 후기

train, test 데이터 셋을 나누어 받았을 때 test 셋에 대한 전처리를 진행해주는 타이밍과 validation 데이터를 나눠주는 타이밍, 다중공선성에 대한 처리, 가변수화 작업에 대한 고민을 깊게 할수록 더 공부할 게 많고, 아직도 많이 부족하다는 생각이 들었습니다. 새로운 문제점에 마주할 때, 잠드는 시간이 늦어지는 것에 대한 불평을 토로하는 사람없이 문제를 해결해냈습니다. 그렇게 확실한 복습과 더 나아간 학습이 되고, 저희가 성장하고 있음을 느낄 수 있어 좋은 경험이 되었다고 생각합니다.

또한, 실무적인 입장에서 데이터를 다루다 보니 그냥 지나쳤던 과정들도 하나씩 신중하게 생각하고 결정하였습니다.

그렇게 하면서 데이터 분석을 더 깊게 이해했던 시간이었습니다.

다섯 명의 스터디원이 모여 유의미한 변수를 선별하는 과정부터 전처리, 분석, 예측까지 직접 해보면서 모르는 부분이 생기면 서로 의견을 나눌 수 있는 점이 좋았습니다.



THANK YOU