서울특별시 아파트 실거래가 예측

Mini Project Team 4 조여재, 엄대용, 박상우

Background

서울 아파트 '역대급 거래 절벽'에 매매
수급지수 19주 째 하락

3년3개월 만에 최저...7월 실거래가지수도
13년 7개월 만에 최대 하락

금리 인상 등에 '급급매'만 일부 팔려...

전국 전세수급지수는 90선 회복

연합뉴스 - 서미숙 기자 2022-09-16

Data Set

아파트 매매 실거래가
국토교통부 실거래가 공개 시스템

서울시 어린이집 정보 (표준 데이터)
서울 열린데이터 광장

서울시 공원 (1인당 공원면적) 통계
공공데이터 포털

CONTENT

1 Data Cleansing

2 EDA

3 Modeling

4 Model Test

5 Conclusion

1 Data Cleansing

아파트 매매 실거래가 - Data 확인

```
apt.head()
```

	시군구	번지	본번	부번	단지명	전용면적(㎡)	계약년월	계약일	거래금액(만원)	층	건축년도	도로명
0	서울특별시 강남구 개포동	658-1	658.0	1.0	개포6차우성아파트1동~8동	79.97	202112	4	215,000	3	1987.0	언주로 3
1	서울특별시 강남구 개포동	658-1	658.0	1.0	개포6차우성아파트1동~8동	79.97	202204	12	220,000	4	1987.0	언주로 3
2	서울특별시 강남구 개포동	658-1	658.0	1.0	개포6차우성아파트1동~8동	79.97	202204	21	220,000	2	1987.0	언주로 3
3	서울특별시 강남구 개포동	658-1	658.0	1.0	개포6차우성아파트1동~8동	79.97	202205	27	216,000	2	1987.0	언주로 3
4	서울특별시 강남구 개포동	1282	1282.0	0.0	개포래미안포레스트	74.66	202109	16	253,000	2	2020.0	개포로 264

아파트 매매 실거래가 - Data 수정

```
drop_features = ['시군구', '본번', '부번', '도로명']
apt = apt.drop(drop_features, axis = 1)
```

```
apt = apt.rename(columns={"번지": 'jibun', '단지명': 'apt_name', '전용면적(㎡)': 'use_area', '계약년월': 'transaction_year_month',
                          '계약일': 'transaction_date', '거래금액(만원)': 'price', '층': 'floor', '건축년도': 'year_of_completion'})
```

```
apt['price'] = apt.price.apply(lambda x: x.replace(',', ''))
apt['price'] = apt.price.apply(pd.to_numeric)
```

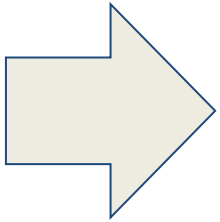
Modeling 할 때 필요 없는 column 삭제
Column명 변경
거래 금액의 ‘,’를 삭제하고 integer로 type 변경

아파트 매매 실거래가 - Null 값 확인

```
apt.isnull().sum()
```

jibun	139
apt_name	0
use_area	0
transaction_year_month	0
transaction_date	0
price	0
floor	0
year_of_completion	2
gu	0
dong	0

dtype: int64



```
apt.isnull().sum()
```

apt_name	0
use_area	0
transaction_year_month	0
transaction_date	0
price	0
floor	0
year_of_completion	0
gu	0
dong	0

dtype: int64

지번 변수에 결측치가 139개 있고, 분석에 유의미한 변수가 아니므로 삭제
완공년도의 결측치가 있는 행 2개를 삭제

아파트 매매 실거래가 - Data

```
apt.head()
```

	jibun	apt_name	use_area	transaction_year_month	transaction_date	price	floor	year_of_completion	gu	dong
0	658-1	개포6차우성아파트1동~8동	79.97	202112	4	215000	3	1987.0	강남구	개포동
1	658-1	개포6차우성아파트1동~8동	79.97	202204	12	220000	4	1987.0	강남구	개포동
2	658-1	개포6차우성아파트1동~8동	79.97	202204	21	220000	2	1987.0	강남구	개포동
3	658-1	개포6차우성아파트1동~8동	79.97	202205	27	216000	2	1987.0	강남구	개포동
4	1282	개포래미안포레스트	74.66	202109	16	253000	2	2020.0	강남구	개포동

서울시 어린이집 정보 - Data 확인

```
day_care.head()
```

	시군구명	어린이집명	어린이집유형	운영현황	상세주소	보육실수	보육실면적	놀이터수	CCTV총설치수	정원	현원	시설 위도(좌표값)	시설 경도(좌표값)	통학차량운영여부	인가일자	제공서비스	반수	아동수	보육교직원수
0	동작구	구립아리아어린이집	국공립	정상	서울특별시 동작구 보라매로5길 51 102호	4	103.0	0	6	33	28	37.490797	126.923171	미운영	2027-12-31	야간연장휴일보육시간제보육	7	28	12
1	구로구	달숲어린이집	국공립	정상	서울특별시 구로구 중앙로5길 59 고척아이파크 RD 내 관리동	6	478.0	2	0	89	0	37.566470	126.977963	NaN	2022-09-01	장애아통합	0	0	0
2	마포구	삼성아이마루어린이집	국공립	정상	서울특별시 마포구 백범로25길 63 염리삼성래미안 관리동	4	107.0	1	8	30	22	37.548530	126.946146	미운영	2022-09-01	일반	7	22	10

서울시 어린이집 정보 - Data 수정

```
# 필요없는 특성 지우기

drop_columns = ["상세주소", "시설 위도(좌표값)", "시설 경도(좌표값)", "보육실 면적", "제공서비스", "반수", "아동수"]

day_care.drop(drop_columns, axis=1, inplace=True)
```

필요 없는 column 삭제

```
# 컬럼명을 영어로 바꾸기

day_care.columns = ["gu", "day_care_name", "day_care_type", "day_care_status", "nursing_room_num", "playground_num",
                    "CCTV_num", "maximum_num", "current_num", "commuting_vehicle", "permission_date", "teacher_num"]
```

column명 변경

```
# 현황이 정상인 값만 남기기

day_care = day_care[day_care["day_care_status"] == "정상"]
```

현황이 정상인 행만 남기기

```
# 통학차량운영여부 NaN 값을 미운영으로 바꾸기

day_care["commuting_vehicle"].fillna("미운영", inplace=True)
```

통학차량운영여부의 Null값을 미운영으로 변경

```
# 정원이 0인 값들 지우기

day_care = day_care[(day_care["maximum_num"] != 0)]
```

정원, 교직원수가 0인 행 삭제

```
# 보육교직원수가 0인 값들 지우기

day_care = day_care[day_care["teacher_num"] != 0]
```

```
# 인가일자를 날자형으로 바꾸고 년도만 추출

day_care["permission_date"] = day_care["permission_date"].apply(pd.to_datetime)
day_care["permission_year"] = day_care["permission_date"].apply(lambda x : x.year)
day_care.drop("permission_date", axis=1, inplace=True)
```

인가일자의 년도만 추출

```
day_care.drop([0], inplace=True)  # 첫번째 데이터는 아직 개원을 안한 어린이집이므로 삭제
```

인가일자가 현재 이후인 데이터 삭제

서울시 어린이집 정보 - Null 값 확인

```
day_care.isnull().sum()
```

```
gu                  0
day_care_name       0
day_care_type       0
day_care_status     0
nursing_room_num    0
playground_num      0
CCTV_num            0
maximum_num         0
current_num         0
commuting_vehicle   0
teacher_num         0
permission_year     0
dtype: int64
```

Null 값이 존재하지 않음

서울시 어린이집 정보 Data

```
day_care.head()
```

	gu	day_care_name	day_care_type	day_care_status	nursing_room_num	playground_num	CCTV_num	maximum_num	current_num	commuting_vehicle
0	동작구	구립아리아어린이집	국공립	정상	4	0	6	33	28	미운영
2	마포구	삼성아이마루어린이집	국공립	정상	4	1	8	30	22	미운영
5	영등포구	LG 에너지솔루션 지정 어린이집	직장	정상	9	1	0	108	60	미운영
6	강남구	케이티앤지 어린이집	직장	정상	4	0	8	49	8	미운영
7	양천구	나연어린이집	가정	정상	3	0	3	12	6	미운영

서울시 어린이집 정보 - New Table 생성

'구'로 groupby 해서 특성들을 대표값(평균값)으로 정리

```
# day_care.groupby("gu").describe().T를 한 결과 모두 평균값과 중앙값의 차이가 거의 없으므로 평균으로 설정
gu_day_care = day_care.groupby("gu").mean()
```

```
columns = ['gu', 'nursing_room_num', 'playground_num', 'CCTV_num', 'maximum_num', 'current_num', 'teacher_num', 'permission_year']

gu_day_care = gu_day_care[columns]
```

```
gu_day_care = round(gu_day_care, 2)
```

통학차량운영여부가 '운영'일 경우 True, '미운영'일 경우 False로 변경

```
gu_day_care['permission_year'] = round(gu_day_care['permission_year'])
```

```
day_care["commuting_vehicle"] = (day_care["commuting_vehicle"] == "운영")
gu_day_care["commuting_vehicle_sum"] = day_care.groupby("gu")["commuting_vehicle"].sum().tolist()
```

```
gu_day_care['day_care_num'] = day_care.groupby('gu').size().values
```

'구'로 groupby한 개수를 구한 column 추가

서울시 어린이집 정보 - New Table

gu_day_care

	gu	nursing_room_num	playground_num	CCTV_num	maximum_num	current_num	teacher_num	permission_year	commuting_vehicle_sum	day_care_num
0	강남구	4.78	1.16	13.31	58.40	37.31	12.43	2007.0	20	18(
1	강동구	4.91	0.82	9.63	52.03	40.53	11.17	2009.0	58	23:
2	강북구	5.11	0.93	8.21	52.76	38.00	10.63	2006.0	40	12:
3	강서구	4.40	0.56	8.06	44.90	33.87	9.97	2009.0	40	30:
4	관악구	4.63	0.76	8.63	48.16	33.57	9.83	2005.0	32	18:
5	광진구	4.79	0.83	8.14	44.27	32.38	9.82	2006.0	22	14:

서울시 공원 면적 통계 - Data

```
park.head()
```

	구	공원 면적 (천㎡)	1인당 공원면적 (㎡)	도시공원 면적 (천㎡)	1인당 도시공원면적 (㎡)	도보생활권공원 면적 (천㎡)	1인당 도보생활권공원면적 (㎡)
0	종로구	11402.58	71.72	6280.66	39.50	3024.28	19.02
1	중구	3163.83	23.50	3091.30	22.96	1344.78	9.99
2	용산구	1777.06	7.26	760.62	3.11	1612.72	6.59
3	성동구	3073.98	10.23	1231.01	4.10	2896.08	9.64
4	광진구	3451.04	9.58	2985.27	8.29	1350.93	3.75

서울시 공원 면적 통계 - Data 수정

```
# 컬럼명을 영어로 바꾸기
park = park.rename(columns={"구": 'gu', '공원 면적 (천㎡)': 'park_area', '1인당 공원면적 (㎡)': 'one_park_area',
                            '도시공원 면적 (천㎡)': 'citypark_area',
                            '1인당 도시공원면적 (㎡)': 'one_citypark_area',
                            '도보생활권공원 면적 (천㎡)': 'walkpark_area',
                            '1인당 도보생활권공원면적 (㎡)': 'one_walkpark_area'})
```

Column명 변경

서울시 공원 면적 통계 - Null 값 확인

Null 값이 존재하지 않음

```
park.isnull().sum()
```

```
gu                  0
park_area           0
one_park_area       0
citypark_area       0
one_citypark_area   0
walkpark_area       0
one_walkpark_area   0
dtype: int64
```

서울시 공원 면적 통계 - Data

```
park.head()
```

	gu	park_area	one_park_area	citypark_area	one_citypark_area	walkpark_area	one_walkpark_area
0	종로구	11402.58	71.72	6280.66	39.50	3024.28	19.02
1	중구	3163.83	23.50	3091.30	22.96	1344.78	9.99
2	용산구	1777.06	7.26	760.62	3.11	1612.72	6.59
3	성동구	3073.98	10.23	1231.01	4.10	2896.08	9.64
4	광진구	3451.04	9.58	2985.27	8.29	1350.93	3.75

Data 합치기

```
final = apt.copy()
```

```
final = pd.merge(final, gu_day_care, on="gu", how="left")
```

```
final = pd.merge(final, park, on="gu", how="left")
final.head()
```

	apt_name	use_area	transaction_year_month	transaction_date	price	floor	year_of_completion	gu	dong	nursing_room_num	...	teacher_num	permis
0	개포6차우성아파트1동~8동	79.97	202112	4	215000	3	1987.0	강남구	개포동	4.78	...	12.43	
1	개포6차우성아파트1동~8동	79.97	202204	12	220000	4	1987.0	강남구	개포동	4.78	...	12.43	
2	개포6차우성아파트1동~8동	79.97	202204	21	220000	2	1987.0	강남구	개포동	4.78	...	12.43	
3	개포6차우성아파트1동~8동	79.97	202205	27	216000	2	1987.0	강남구	개포동	4.78	...	12.43	
4	개포래미안포레스트	74.66	202109	16	253000	2	2020.0	강남구	개포동	4.78	...	12.43	

5 rows × 24 columns

아파트 매매 가격 데이터에 '구'가 같은 어린이집, 공원 데이터 추가 (by left join)

Data 합치기 - Null 값 확인

```
final.isnull().sum()
```

```
apt_name                  0
use_area                  0
transaction_year_month    0
transaction_date          0
price                     0
floor                     0
year_of_completion        0
gu                        0
dong                      0
nursing_room_num          0
playground_num            0
CCTV_num                  0
maximum_num               0
current_num               0
teacher_num               0
permission_year           0
commuting_vehicle_sum     0
day_care_num              0
park_area                 0
one_park_area             0
citypark_area             0
one_citypark_area         0
walkpark_area             0
one_walkpark_area         0
dtype: int64
```

Null 값이 존재하지 않음

2 EDA

Column Information

- gu : 구
- dong : 동
- apt_name : 아파트 이름
- use_area : 아파트 전용면적
- transaction_year_month : 아파트 거래년월
- transaction_date : 아파트 거래날짜
- price : 아파트 실거래가
- floor : 아파트 층
- year_of_completion : 아파트 설립일자

- nursing_room_num : 어린이집 보육실 수
- playground_num : 어린이집 놀이터 수
- CCTV_num : 어린이집 cctv개수
- maximum_num : 어린이집 정원
- current_num : 어린이집 현원
- teacher_num : 어린이집 교직원수
- permission_year : 어린이집 인가일자
- commuting_vehicle_sum : 어린이집 통학차량 운영 수
- day_care_num : 어린이집 수

- park_area : 공원 면적
- one_park_area : 1인당 공원 면적
- citypark_area : 도시공원 면적
- one_citypark_area : 1인당 도시공원 면적
- walkpark_area : 도보생활권공원 면적
- one_walkpark_area : 1인당 도보생활권공원 면적

Train Test Split

Data를 train set과 test set으로 분리

```
train, test = train_test_split(apt, test_size=0.2, random_state = 7916, stratify = apt['gu'])
```

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

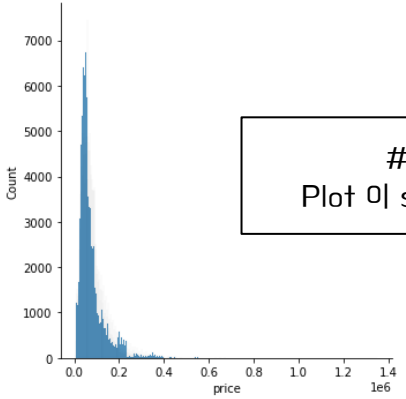
```
(257362, 24) (64341, 24)
```

Target Data 확인

target 값(price) 의 plot

```
sns.displot(train['price'])  # 타깃값이 skewed 하므로 회귀 모델의 성능을 높이기 위해서 변환이 필요
```

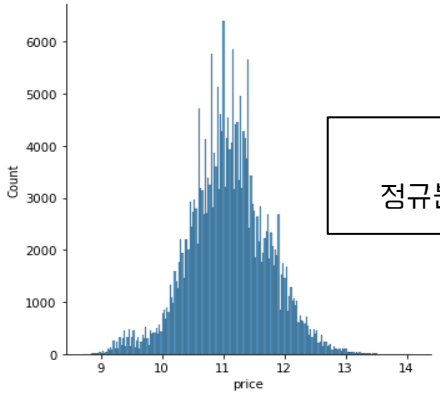
```
<seaborn.axisgrid.FacetGrid at 0x26680dfab50>
```



원본 target 값
Plot 이 skewed -> 변환이 필요

```
sns.displot(np.log(train['price']))  # log 변환을 해주면 정규분포를 따르므로 로그 변환 진행
```

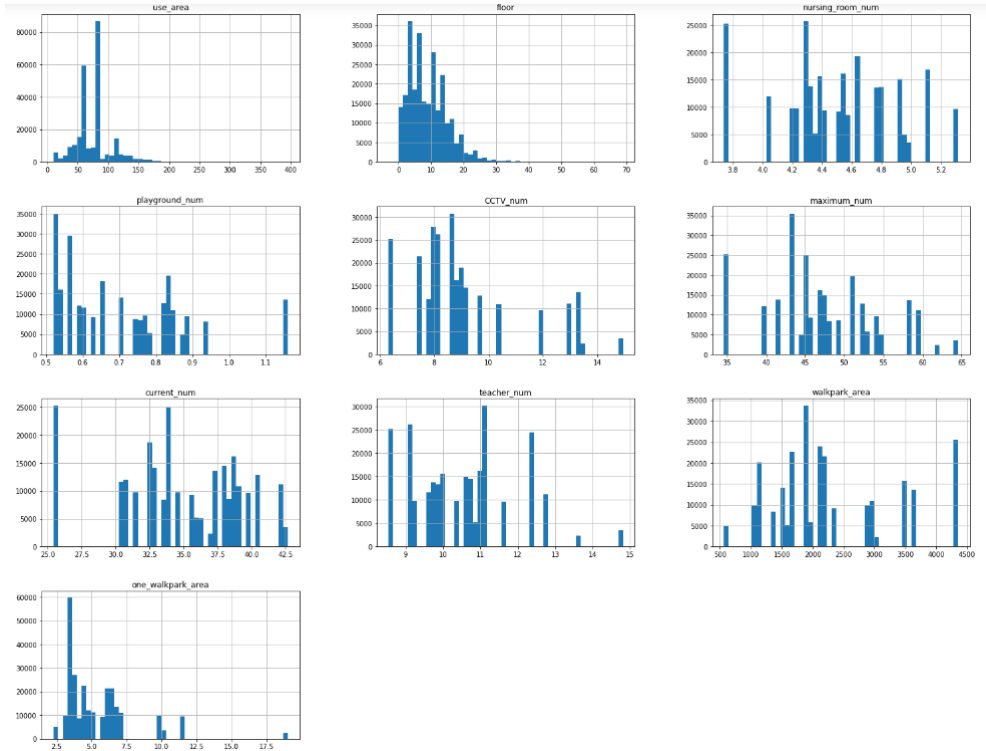
```
<seaborn.axisgrid.FacetGrid at 0x26689906850>
```



log(target)
정규분포를 따름 -> log 변환 진행

Independent Variables 확인

```
X_hist = X_train.hist(bins=50, figsize=(25,20))
```



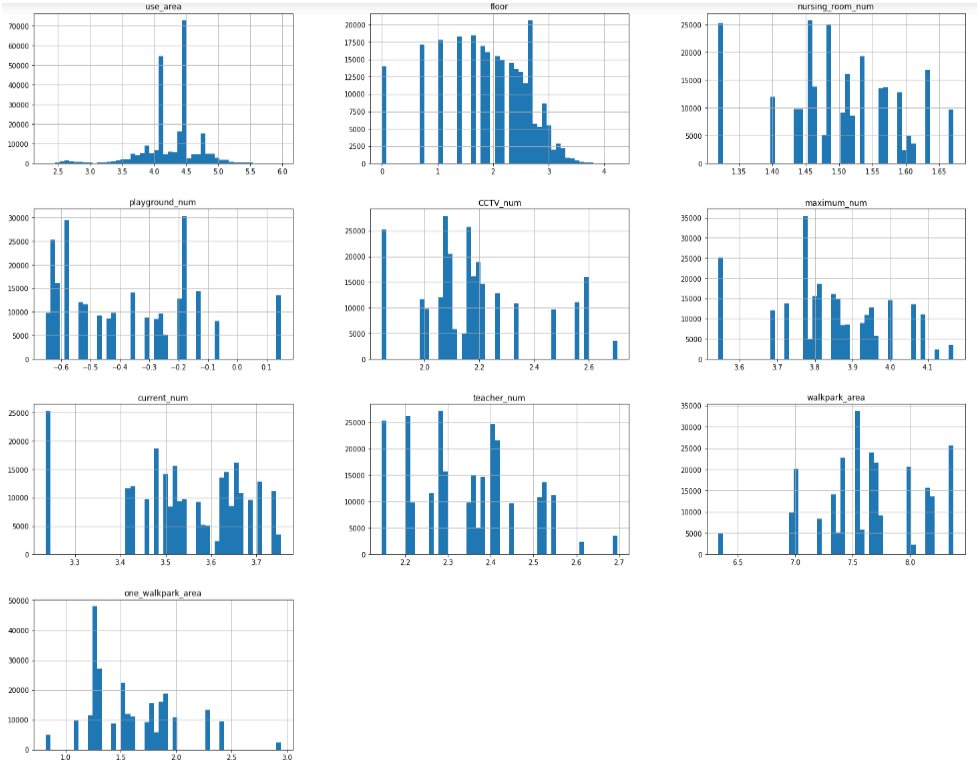
train data의 histogram 확인

Independent Variables 확인

log(X)의 histogram
원 데이터와 큰 차이를 보이지 않음

```
log_X = np.log(X_train)
```

```
log_hist =log_X.hist(bins=50, figsize=(25,20))
```



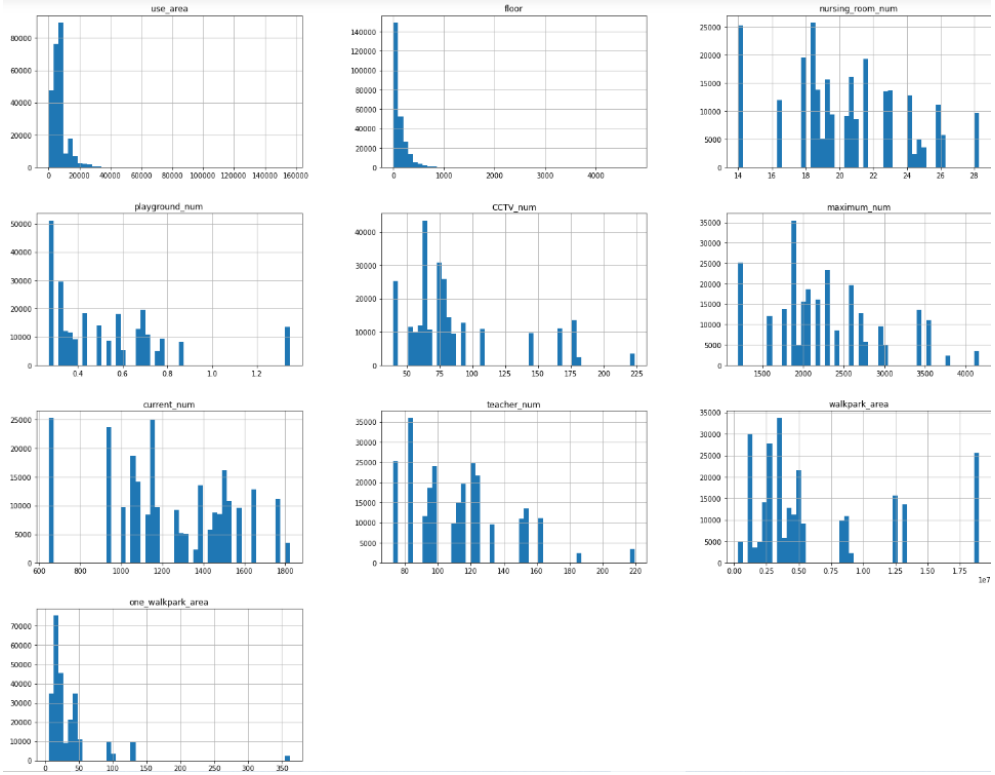
Independent Variables 확인

X^2 의 histogram
skewed한 결과가 나타남

-> 원 데이터가 더 좋음
-> tuning을 하지 않고 사용

```
X2 = (X_train)**2
```

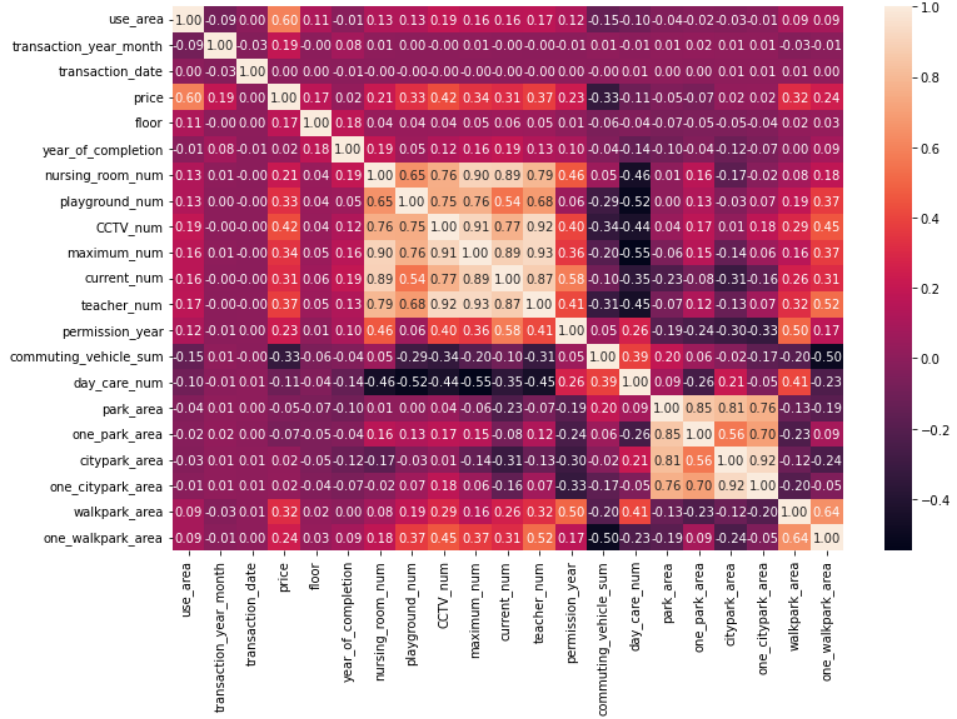
```
hist_X2 = X2.hist(bins=50, figsize=(25,20)) # 제곱변환을 하면 skewed한 결과가 나타남
```



Correlation Matrix

```
plt.figure(figsize=(12, 8))
corr_matrix = train.corr()
sns.heatmap(corr_matrix, annot=True, fmt=".2f")
```

```
<AxesSubplot:>
```



Correlation Matrix

```
corr_matrix = train.corr()
corr_matrix['price'].sort_values(ascending=False)
```

```
price                    1.000000
use_area                 0.595160
CCTV_num                 0.424423
teacher_num              0.374944
maximum_num              0.339786
playground_num           0.328529
walkpark_area            0.315364
current_num              0.305761
one_walkpark_area        0.239233
permission_year          0.230042
nursing_room_num         0.208940
transaction_year_month   0.191856
floor                    0.169619
one_citypark_area        0.023741
year_of_completion       0.023361
citypark_area            0.017898
transaction_date         0.000832
park_area               -0.050086
one_park_area           -0.072148
day_care_num            -0.114949
commuting_vehicle_sum   -0.329761
Name: price, dtype: float64
```

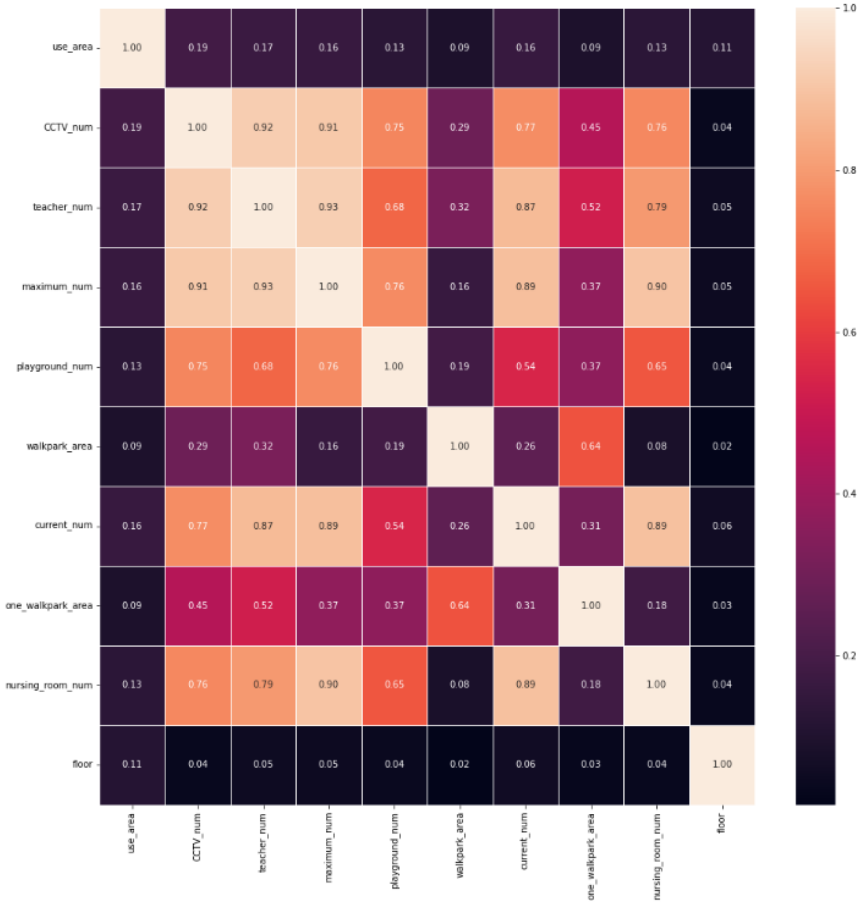
변수 선택 기준
상관계수가 0.2 이상
중요한 변수라고 판단이 되는 경우

Multicolinearity 확인

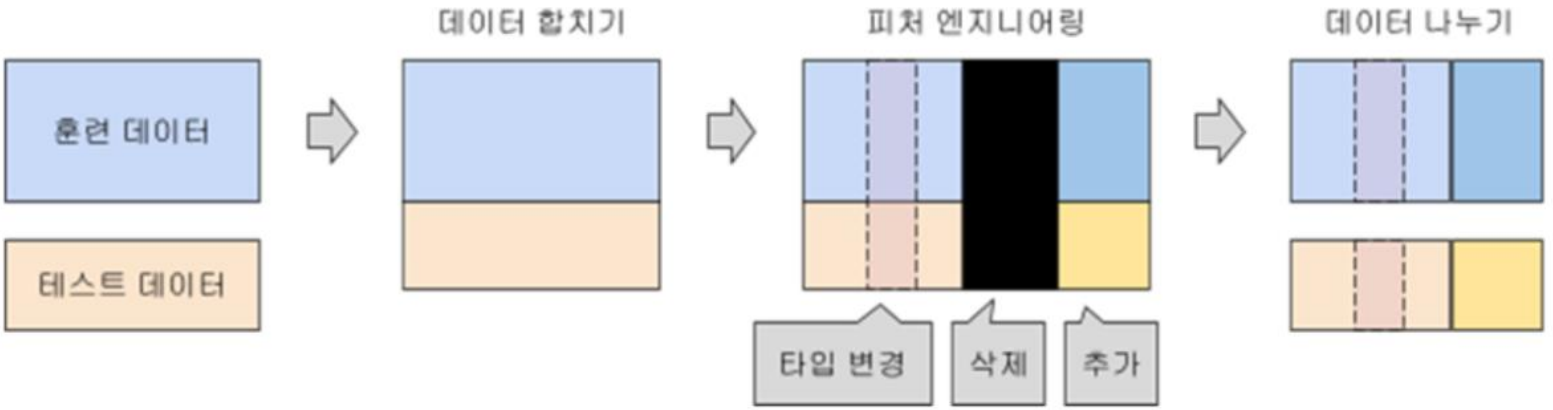
Multicolinearity가 높게 나온 경우

cctv수 & 보육실수
cctv수 & 놀이터수 / 정원
보육실수 / 교직원수
#보육실수 / 정원 & 놀이터수 등

이 값들은 담연하게도
어린이 집의 정원이 많으면
보육실수, 놀이터수, ccvtv 수, 교직원 수가
많게 나올 수 밖에 없음



Variable 선택



train set과 test set을 합친 후 변수 선택
변수 선택이 완료된 후 다시 train set과 test set 분리

Variable 선택

```
all_data = pd.concat([train, test], axis = 0, ignore_index = True)  # 행축으로 연결
all_data.tail()
```

	apt_name	use_area	transaction_year_month	transaction_date	price	floor	year_of_completion	gu	dong	nu		
321698	우방	114.79	202003	27	NaN	2	1999.0	관악구	신림동			
321699	한마을	149.97	201803	24	NaN	24	1999.0	구로구	개봉동		4.29	... 9.13
321700	정릉우성아파트101~105동	84.99	202002	22	NaN	8	2000.0	성북구	정릉동			
321701	양평현대6차	59.98	201806	21	NaN	5	2000.0	영등포구	양평동3가		4.63	... 12.32
321702	은평뉴타운마고정2단지	84.04	201806	23	NaN	2	2009.0	은평구	진관동		5.32	... 11.59

5 rows × 24 columns

```
target = test['price'].copy()
test = test.drop('price', axis = 1)
```

```
test.shape, target.shape
```

```
((64341, 23), (64341,))
```

데이터를 합치기 전에 test에서 target(price) 분리

target값이 제거된 test와 target값이 남아있는 train 데이터를 합침

```
all_data.shape
```

```
(321703, 24)
```

Variable 선택

```
drop_features = ['apt_name', 'gu', 'dong', 'transaction_year_month', 'transaction_date', 'year_of_completion', 'commuting_vehicle_sum',
                 'day_care_num', 'permission_year', 'park_area', 'one_park_area', 'citypark_area', 'one_citypark_area']
all_data = all_data.drop(drop_features, axis = 1)
```

변수 선택

```
X_train = all_data[~pd.isnull(all_data['price'])]
X_test = all_data[pd.isnull(all_data['price'])]
```

price의 null값을 이용해 X_train과 X_test로 분리

```
X_train.shape, X_test.shape
```

```
((257362, 11), (64341, 11))
```

```
# 타깃값(count) 제거, 특성만 가져오기
y_train = X_train['price']
X_train = X_train.drop(['price'], axis = 1)

X_test = X_test.drop(['price'], axis = 1)
```

target값(price)을 제거

```
print(X_train.shape, y_train.shape, X_test.shape)
```

```
(257362, 10) (257362,) (64341, 10)
```

```
print(target.shape)
```

```
(64341,)
```

3 Modeling

Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
lin_reg = LinearRegression()
```

```
log_y = np.log(y_train)
lin_reg.fit(X_train, log_y)
```

y값을 log 변환

```
LinearRegression()
```

```
from sklearn.model_selection import cross_val_score
```

```
# 교차 검증
scores = cross_val_score(lin_reg, X_train, log_y, scoring = 'neg_mean_squared_error', cv = 5)
-scores.mean()
```

```
0.18072040742132023
```

train mse = 0.18072040742132023

Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly_features.fit_transform(X_train)
X_train.shape, X_train_poly.shape
```

```
((257362, 10), (257362, 65))
```

```
# 다항회귀 (정규방정식)
lin_reg = LinearRegression()

# 교차검증
scores = cross_val_score(lin_reg, X_train_poly, log_y, scoring = 'neg_mean_squared_error', cv = 5)
-scores.mean()
```

```
0.13458429551660994
```

정규방정식 이용
train mse = 0.13458429551660994

Polynomial Regression

```
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
```

```
# 다항회귀 (경사하강법)
# Poly(제곱특성추가) -> SGDRegressor(경사하강법)

sgd_reg = SGDRegressor(penalty = 'None', random_state = 7916)

poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly_features.fit_transform(X_train)

# 교차검증
scores = cross_val_score(sgd_reg, X_train_poly, log_y, scoring = 'neg_mean_squared_error', cv = 5)
-scores.mean()
```

```
6.5906142188899053e+43
```

경사하강법 이용
train mse = 6.5906142188899053e+43

Ridge

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
```

```
ridge = Ridge()
```

```
ridge_params = {'alpha': [0.01, 0.05, 0.1, 1, 2, 3, 4, 10]}

gridsearch_ridge = GridSearchCV(ridge, ridge_params, scoring = 'neg_mean_squared_error', cv = 5, n_jobs = -1)
```

```
gridsearch_ridge.fit(X_train, log_y)
```

```
GridSearchCV(cv=5, estimator=Ridge(), n_jobs=-1,
             param_grid={'alpha': [0.01, 0.05, 0.1, 1, 2, 3, 4, 10]},
             scoring='neg_mean_squared_error')
```

```
gridsearch_ridge.best_params_
```

```
{'alpha': 0.1}
```

```
cvres = gridsearch_ridge.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params)
```

```
0.1807204074077289 {'alpha': 0.01}
0.1807204073560647 {'alpha': 0.05}
0.1807204072975609 {'alpha': 0.1}
0.18072040739787493 {'alpha': 1}
0.18072041006734557 {'alpha': 2}
0.18072041542155765 {'alpha': 3}
0.18072042345236378 {'alpha': 4}
0.18072052739285716 {'alpha': 10}
```

{alpha = 0.1}
train mse = 0.1807204072975609

Lasso

```
from sklearn.linear_model import Lasso
```

```
lasso = Lasso()
```

```
lasso_params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 1]}

gridsearch_lasso = GridSearchCV(lasso, lasso_params, scoring = 'neg_mean_squared_error', cv = 5, n_jobs = -1)
```

```
gridsearch_lasso.fit(X_train, log_y)
```

```
GridSearchCV(cv=5, estimator=Lasso(), n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.5,
                                   1]},
             scoring='neg_mean_squared_error')
```

```
gridsearch_lasso.best_params_
```

```
{'alpha': 0.0001}
```

```
cvres = gridsearch_lasso.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params)
```

```
0.1807230080705341 {'alpha': 0.0001}
0.18097913619268927 {'alpha': 0.001}
0.1883657128835511 {'alpha': 0.01}
0.19260633936299607 {'alpha': 0.05}
0.19732680820082193 {'alpha': 0.1}
0.19868997035712072 {'alpha': 0.2}
0.20666071753316134 {'alpha': 0.5}
0.2127306755308894 {'alpha': 1}
```

\# {alpha = 0.0001}
\# train mse = 0.1807230080705341

ElasticNet

```
from sklearn.linear_model import ElasticNet
```

```
elastic = ElasticNet()
```

```
elastic_params = {'alpha': [0.001, 0.01, 0.05 , 0.1, 0.2, 0.5, 1]}

gridsearch_elastic = GridSearchCV(elastic, elastic_params, scoring = 'neg_mean_squared_error', cv = 5, n_jobs = -1)
```

```
gridsearch_elastic.fit(X_train, log_y)
```

```
GridSearchCV(cv=5, estimator=ElasticNet(), n_jobs=-1,
             param_grid={'alpha': [0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 1]},
             scoring='neg_mean_squared_error')
```

```
gridsearch_elastic.best_params_
```

```
{'alpha': 0.001}
```

```
cvres = gridsearch_elastic.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params)
```

```
0.18084413448041486 {'alpha': 0.001}
0.184975762863777 {'alpha': 0.01}
0.19124012875794721 {'alpha': 0.05}
0.1928663683512557 {'alpha': 0.1}
0.19733617583935376 {'alpha': 0.2}
0.1997509249054698 {'alpha': 0.5}
0.20670993767099444 {'alpha': 1}
```

{alpha = 0.001}
train mse = 0.18084413448041486

Decision Tree(1)

```
from sklearn.tree import DecisionTreeRegressor
```

```
tree_reg = DecisionTreeRegressor(random_state=7916)
```

```
tree_params = {'max_features' : [2, 4, 6, 8, 10], 'max_depth' : [10, 15, 20]}
```

```
gridsearch_tree = GridSearchCV(tree_reg, tree_params, scoring='neg_mean_squared_error', cv=5, n_jobs=-1)
gridsearch_tree.fit(X_train, log_y)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(random_state=7916),
             n_jobs=-1,
             param_grid={'max_depth': [10, 15, 20],
                         'max_features': [2, 4, 6, 8, 10]},
             scoring='neg_mean_squared_error')
```

```
gridsearch_tree.best_params_
```

```
{'max_depth': 15, 'max_features': 10}
```

```
cvres = gridsearch_tree.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params)
```

```
0.11487183161593748 {'max_depth': 10, 'max_features': 2}
0.10969240983620512 {'max_depth': 10, 'max_features': 4}
0.10615512597111434 {'max_depth': 10, 'max_features': 6}
0.10383601613669557 {'max_depth': 10, 'max_features': 8}
0.10200807469009603 {'max_depth': 10, 'max_features': 10}
0.09692246014852732 {'max_depth': 15, 'max_features': 2}
0.09469887662502886 {'max_depth': 15, 'max_features': 4}
0.09017243267935694 {'max_depth': 15, 'max_features': 6}
0.08645867334017861 {'max_depth': 15, 'max_features': 8}
0.08326235831588577 {'max_depth': 15, 'max_features': 10}
0.09801115503097967 {'max_depth': 20, 'max_features': 2}
0.09592863038340424 {'max_depth': 20, 'max_features': 4}
0.09200732940088843 {'max_depth': 20, 'max_features': 6}
0.08892846373828527 {'max_depth': 20, 'max_features': 8}
0.08502185289684881 {'max_depth': 20, 'max_features': 10}
```

{max_depth: 15, max_features: 10}
train mse = 0.08326235831588577

Decision Tree(2)

```
best_model_tree = gridsearch_tree.best_estimator_
best_model_tree
```

```
DecisionTreeRegressor(max_depth=15, max_features=10, random_state=7916)
```

```
sorted(zip(best_model_tree.feature_importances_, X_train.columns), reverse = True)
```

```
[(0.6146105430435348, 'use_area'),
 (0.1972381418777127, 'CCTV_num'),
 (0.10177582046377515, 'teacher_num'),
 (0.02247488383404033, 'floor'),
 (0.01522945917084668, 'current_num'),
 (0.013198929620090767, 'walkpark_area'),
 (0.012032288496765764, 'playground_num'),
 (0.011600445667100983, 'one_walkpark_area'),
 (0.006063645094064736, 'maximum_num'),
 (0.005775842732068011, 'nursing_room_num')]
```

Random Forest(1)

```
from sklearn.ensemble import RandomForestRegressor
```

```
rnd_forest = RandomForestRegressor(random_state=7916)
```

```
rf_params = {'n_estimators': [100, 120, 140]}
```

```
gridsearch_forest = GridSearchCV(rnd_forest, rf_params, scoring = 'neg_mean_squared_error', cv = 5, n_jobs = -1)
gridsearch_forest.fit(X_train, log_y)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=7916),
             n_jobs=-1, param_grid={'n_estimators': [100, 120, 140]},
             scoring='neg_mean_squared_error')
```

```
gridsearch_forest.best_params_
```

```
{'n_estimators': 140}
```

{'max_depth': 20, 'max_features': 10, 'n_estimators': 220}
train mse = 0.0729584701779201

```
cvres = gridsearch_forest.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params)
```

```
0.07700034640991632 {'n_estimators': 100}
0.07695854565748539 {'n_estimators': 120}
0.07692742043158787 {'n_estimators': 140}
```

Random Forest(2)

```
best_model_forest = gridsearch_forest.best_estimator_
best_model_forest
```

```
RandomForestRegressor(n_estimators=140, random_state=7916)
```

```
sorted(zip(best_model_forest.feature_importances_, X_train.columns), reverse = True)
```

```
[(0.6085758168800071, 'use_area'),
 (0.22275457305917637, 'CCTV_num'),
 (0.05799867691133111, 'floor'),
 (0.019287299360374262, 'current_num'),
 (0.018949525542086267, 'one_walkpark_area'),
 (0.018648489831402156, 'maximum_num'),
 (0.016804220709392224, 'teacher_num'),
 (0.014131990108862031, 'playground_num'),
 (0.011699584298100307, 'walkpark_area'),
 (0.011149823299204163, 'nursing_room_num')]
```

random forest model 은 black box model
feature importance를 확인하는 것이 중요

Gradient Boosting (Ensemble)

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
gbrt_params = {'learning_rate': [0.01, 0.02, 0.03, 0.04],  # 각 트리의 기여도
               'n_estimators': [1000, 1500],
               'subsample': [0.9, 0.5, 0.2],
               'max_depth': [2, 4, 6, 8]}

gbrt = GradientBoostingRegressor()

gridsearch_gbrt = GridSearchCV(gbrt, gbrt_params, scoring = 'neg_mean_squared_error', cv = 5, n_jobs = -1)
```

```
gridsearch_gbrt.fit(X_train, log_y)
```

```
GridSearchCV(cv=5, estimator=GradientBoostingRegressor(), n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.02, 0.03, 0.04],
                         'max_depth': [2, 4, 6, 8],
                         'n_estimators': [1000, 1500],
                         'subsample': [0.9, 0.5, 0.2]},
             scoring='neg_mean_squared_error')
```

```
gridsearch_gbrt.best_params_
```

```
{'learning_rate': 0.04, 'max_depth': 8, 'n_estimators': 1500, 'subsample': 0.5}
```

```
cvres = gridsearch_gbrt.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params)
```

{'learning_rate': 0.04, 'max_depth': 8,
'n_estimators': 1500, 'subsample': 0.5}
train mse = 0.07059033032043774

Gradient Boosting (Ensemble)

```
best_model_gbrt = gridsearch_gbrt.best_estimator_
```

```
sorted(zip(best_model_gbrt.feature_importances_, X_train.columns), reverse = True)
```

```
[(0.5985527661716553, 'use_area'),
 (0.2090611626501374, 'CCTV_num'),
 (0.03459397859901509, 'floor'),
 (0.03125444789522308, 'maximum_num'),
 (0.026732005392210993, 'one_walkpark_area'),
 (0.026218544756001577, 'teacher_num'),
 (0.022255562453814107, 'playground_num'),
 (0.018892869139067583, 'current_num'),
 (0.017468469362051042, 'walkpark_area'),
 (0.014970193580823757, 'nursing_room_num')]
```

Voting (Ensemble)

```
from sklearn.ensemble import VotingRegressor
```

```
lin_reg = LinearRegression()
ridge = Ridge(alpha=0.1, random_state=7916)
rnd_forest = RandomForestRegressor(max_depth=20, max_features=10, n_estimators=220, random_state=7916)
svm = SVR()
```

```
voting_reg = VotingRegressor(estimators = [('lin', lin_reg), ('rid', ridge), ('rnd', rnd_forest), ('svmr', svm)],
                n_jobs=-1)

scores = cross_val_score(voting_reg, X_train, log_y, scoring = 'neg_mean_squared_error', cv = 2)
-scores.mean()
```

```
0.13330274291081O6
```

random forest best model의 hyper parameter 이용
train mse = 0.13330274291081O6

Bagging (Ensemble)

```
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.ensemble import BaggingRegressor
```

```
rnd_forest = RandomForestRegressor(max_depth=20, max_features=10, n_estimators=220, random_state=7916)
```

```
bag_reg = BaggingRegressor(rnd_forest, n_jobs=-1, random_state=7916)
```

```
scores = cross_val_score(bag_reg, X_train, log_y, scoring = 'neg_mean_squared_error', cv = 3)
-scores.mean()
```

```
0.07514101865338980
```

random forest best model의 hyper parameter 이용
train mse = 0.07514101865338980

Linear SVR (Support Vector Regressor)

```
from sklearn.svm import LinearSVR
```

```
svm_reg = LinearSVR()
```

```
scores = cross_val_score(svm_reg, X_train, log_y, scoring = 'neg_mean_squared_error', cv = 5)
-scores.mean()
```

```
2.9375196592384363
```

train mse = 2.9375196592384363

4 Model Test

Final Model

Train 데이터로 fitting 해본 결과
최소의 train mse를 갖는 모델인
Gradient Boosting Model을
Final Model로 선택

Final Model - Gradient Boosting

```
log_target = np.log(target)
```

target 값에 log를 취함

```
gbrt = GradientBoostingRegressor(learning_rate=0.04, max_depth=8, n_estimators=1500, subsample=0.5, random_state=7916)
```

```
gbrt.fit(X_train, log_y)
```

```
GradientBoostingRegressor(learning_rate=0.04, max_depth=8, n_estimators=1500,
                          random_state=7916, subsample=0.5)
```

```
y_pred = gbrt.predict(X_test)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.06930499418814444
```

{learning_rate = 0.04, max_depth = 8, n_estimators = 1500, subsample = 0.5}
test mse = 0.06930499418814444

Final Model - Gradient Boosting

```
data = np.c_[(log_target, y_pred)]
```

```
data = pd.DataFrame(data, columns=['log_target', 'y_pred'])
```

```
data
```

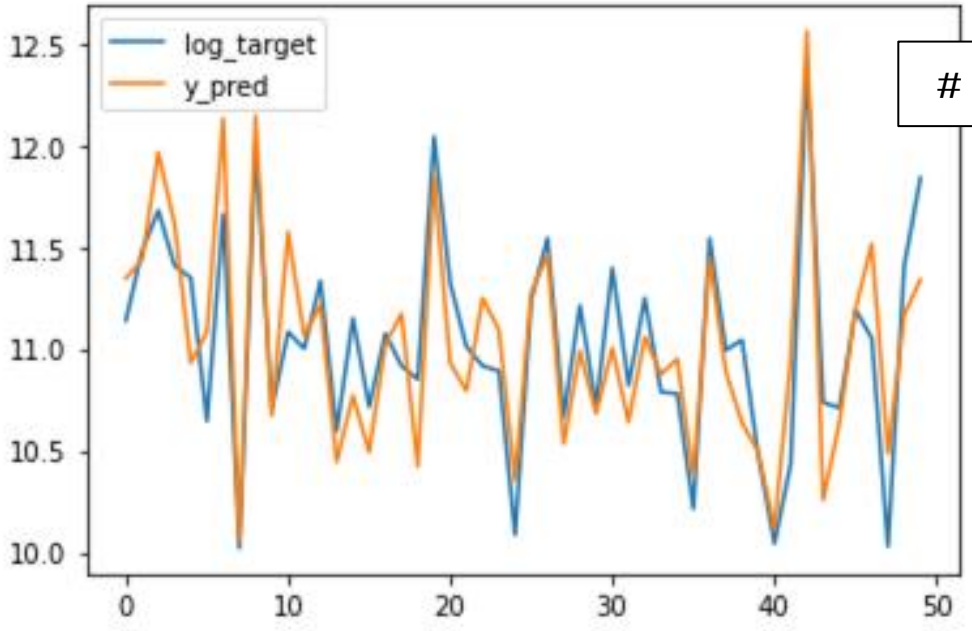
	log_target	y_pred
0	11.138959	11.345376
1	11.482466	11.436821
2	11.678440	11.964233
3	11.407565	11.612626
4	11.348051	10.931428
...	...	...
64336	10.976782	11.082403
64337	10.993732	11.189953
64338	10.896739	11.097455
64339	10.922335	11.235841
64340	11.016988	11.182772

64341 rows × 2 columns

target 값과 prediction 값 비교를 위해 테이블 생성

Final Model - Gradient Boosting

```
plt.plot(data.iloc[:50, 0], label = 'log_target')
plt.plot(data.iloc[:50, 1], label = 'y_pred')
plt.legend()
plt.show()
```



target 과 prediction plot

추가 Model Test

Random Forest
Linear Regression
Polynomial Regression
Ridge
Lasso
ElasticNet
Decision Tree
Linear SVR

Random Forest

```
rnd_forest = RandomForestRegressor(n_estimators=220, max_depth=20, max_features=10, random_state=7916)
```

```
rnd_forest.fit(X_train, log_y)
```

```
RandomForestRegressor(max_depth=20, max_features=10, n_estimators=220,
                      random_state=7916)
```

```
y_pred = rnd_forest.predict(X_test)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.07015279714123512
```

{max_depth = 20, max_features = 10, n_estimators = 220}
test mse = 0.07015279714123512

Linear Regression

```
lin_reg = LinearRegression()
```

```
lin_reg.fit(X_train, log_y)
```

```
LinearRegression()
```

```
y_pred = lin_reg.predict(X_test)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.17907320704468227
```

test mse = 0.17907320704468227

Polynomial Regression

```
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly_features.fit_transform(X_train)
X_train.shape, X_train_poly.shape

lin_reg_poly = LinearRegression()
```

```
lin_reg_poly.fit(X_train_poly, log_y)
```

```
LinearRegression()
```

```
X_test_poly = poly_features.fit_transform(X_test)
```

```
y_pred = lin_reg_poly.predict(X_test_poly)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.13370975594612017
```

test mse = 0.13370975594612017

Ridge

```
ridge = Ridge(alpha=0.1, random_state=7916)
```

```
ridge.fit(X_train, log_y)
```

```
Ridge(alpha=0.1, random_state=7916)
```

```
y_pred = ridge.predict(X_test)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.17907321527937733
```

{alpha = 0.1}
test mse = 0.17907321527937733

Lasso

```
lasso = Lasso(alpha=0.0001, random_state=7916)
```

```
lasso.fit(X_train, log_y)
```

```
Lasso(alpha=0.0001, random_state=7916)
```

```
y_pred = lasso.predict(X_test)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.179079231813746
```

{alpha = 0.0001}
test mse = 0.17907923181374 6

ElasticNet

```
elastic = ElasticNet(alpha=0.001, random_state=7916)
```

```
elastic.fit(X_train, log_y)
```

```
ElasticNet(alpha=0.001, random_state=7916)
```

```
y_pred = elastic.predict(X_test)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.17922363511430414
```

{alpha = 0.001}
test mse = 0.17922363511430414

Decision Tree Regressor

```
tree_reg = DecisionTreeRegressor(random_state=7916, max_depth=15, max_features=10)
```

```
tree_reg.fit(X_train, log_y)
```

```
DecisionTreeRegressor(max_depth=15, max_features=10, random_state=7916)
```

```
y_pred = tree_reg.predict(X_test)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.08082504271135 68
```

{max_depth = 15, max_features = 10}
test mse = 0.08082504271135 68

Linear SVR

```
lin_svr = LinearSVR()
```

```
lin_svr.fit(X_train, log_y)
```

```
LinearSVR()
```

```
y_pred = lin_svr.predict(X_test)
```

```
mean_squared_error(log_target, y_pred)
```

```
0.4841539298846815
```

test mse = 0.4841539298846815

5 Conclusion

Conclusion

Model	Train MSE	Test MSE
Gradient Boosting	0.07059033032043774	0.06930499418814444
Random Forest	0.07692742043158787	0.07015279714123512
Linear Regression	0.18072040742132023	0.17907320704468227
Polynomial Regression	0.13458429551660994	0.13370975594612017
Ridge	0.18072040729756 09	0.17907321527937733
Lasso	0.1807230080705341	0.179079231813746
ElasticNet	0.18084413448041486	0.17922363511430414
Decision Tree	0.08326235831588577	0.0808250427113568
Linear SVR	2.9375196592384363	0.4841539298846815

Final Model
Gradient Boosting
Best Train MSE
Best Test MSE

전체적으로 Train MSE와 Test MSE가 비슷한 경향성을 가지고 있음


THANK YOU