

# 서울 부동산 가격 예측 모델링

# INDEX

A table of contents

---

**1** 주제 선정

**2** 데이터 수집 및 전처리

**3** 데이터 시각화 및 분석

**4** 예측 모델

**5** 회고 및 고찰



# Part 1

주제 선정

# 주제 선정 배경



혼돈의 부동산, '공급·금리·양극화' 극복할 수 있을까

거래절벽 속 집값 양극화 극심... 1위-2·3위 격차 커져

입력 2022.06.16. 오전 5:11 기사원문

부동산 관망세 지속... 지역 양극화 심화

송고 2022.06.15 16:21

'서울 아파트 시장의 두얼굴' 시세보다 수억 떨어진 급매물  
거래 VS 최고가 경신 계약 성사

입력 2022.06.16. 오전 7:02 수정 2022.06.16. 오전 11:24 기사원문

- 극심하게 변하는 서울 부동산 시장 속에서 어떠한 요인이 가격에 영향을 미치는지 알아보고자 한다.
- 어떤 모델이 가격 예측에 가장 높은 정확도를 보이는지 적용해 보고자 한다.

# 워크플로우





# Part 2

데이터 수집 및 전처리

# 데이터 수집 및 전처리 : 데이터 출처



The screenshot shows the Seoul Open Data Portal interface. On the left, there's a sidebar with a map icon and the text '도시관리' (City Management). Below it, under '공공데이터' (Public Data), are buttons for '활용갤러리 등록' (Register for Utilization Gallery) and 'URL 복사' (Copy URL). At the bottom is a blue button labeled '목록 이동' (Move to List). The main content area has a title '서울시 부동산 실거래가 정보' (Seoul City Real Estate Transaction Information) and a brief description about the data being provided by the Seoul Special City Government. It includes details like location, legal district, transaction date, and building information. Below this is a table titled '데이터 정보' (Data Information) with various metadata fields.

공개일자	2022.04.28	최신수정일자	2022.04.22	
갱신주기	매일1회	분류	도시관리	
원본시스템	서울 부동산 정보광장	바로가기	저작권자	서울특별시
제공기관	서울특별시	제공부서	도시계획국 토지관리과	
담당자	박인영 (02-2133-4672)			
원본형태	DB	제3저작권자	없음	
라이센스	 저작권자표시(BY): 이용이나 변경 및 2차적 저작물의 작성을 포함한 자유이용을 허락합니다.			
관련태그	실거래가, 부동산, 부동산, 실거래, 아파트, 건물			

출처 : 서울 열린데이터 광장

<http://data.seoul.go.kr/dataList/OA-21275/S/1/datasetView.do>

- 2017년부터 2022년 상반기까지의 부동산 실거래가 정보 조회
- 20개의 칼럼 중 8개의 칼럼만 추출
- 거래가격에 영향을 미치는 변수들이 무엇인지 탐색

# 데이터 수집 및 전처리 : 필요 칼럼 추출

```
# 2) 컬럼명 수정  
df = raw_data.rename(columns={'물건금액(만원)':'물건금액', '건물면적(㎡)':'건물면적'})  
  
# 3) 필요한 칼럼만 추출하기  
df = df.loc[:,['접수연도', '자치구명', '법정동명', '물건금액', '건물면적', '총', '건축년도', '건물용도']]  
df.info()  
...  
  
#   Column  Non-Null Count  Dtype    
---  
 0   접수연도      640000 non-null  int64  
 1   자치구명      640000 non-null  object - 인코딩 필요  
 2   법정동명      640000 non-null  object - 인코딩 필요  
 3   물건금액      640000 non-null  float64  
 4   건물면적      640000 non-null  float64  
 5   총             591406 non-null  float64  
 6   건축년도      637484 non-null  float64  
 7   건물용도      640000 non-null  object - 인코딩 필요  
dtypes: float64(3), int64(2), object(3)  
...  
  
물건금액과 건물면적의 단위를 제거 한 뒤, 필요한 8개의 칼럼만 추출
```

# 데이터 수집 및 전처리 : 데이터셋 소개

In [15]: df

Out[15]:

약 640,000개의 행과 8개의 열로 구성됨

	접수연도	자치구명	법정동명	물건금액	건물면적	총	건축년도	건물용도
0	2022	강동구	천호동	23750	29.93	2.0	2016.0	연립다세대
1	2022	강서구	화곡동	41500	29.96	5.0	2022.0	연립다세대
2	2022	송파구	잠실동	29500	24.42	10.0	2013.0	오피스텔
3	2022	구로구	구로동	12700	19.99	16.0	2011.0	아파트
4	2022	금천구	가산동	15000	17.16	13.0	2018.0	오피스텔
...	...	...	...	...	...	...	...	...
639995	2018	강서구	방화동	8280	26.86	3.0	1999.0	오피스텔
639996	2018	중랑구	면목동	23300	55.73	1.0	1994.0	연립다세대
639997	2018	송파구	방이동	26300	30.62	4.0	2013.0	연립다세대
639998	2018	강동구	명일동	41550	100.24	5.0	2005.0	오피스텔
639999	2018	관악구	신림동	21000	35.06	4.0	2014.0	연립다세대

# 데이터 수집 및 전처리 : 결측치 확인

```
# 4) 결측치 확인
df.isnull().sum()
...
접수연도      0
자치구명      0
법정동명      0
물건금액      0
건물면적      0
총          48594
건축년도      2516
건물용도      0
dtype: int64
...
```



```
#####
## 추가 확인
#####
# '총'의 null값 확인
df[df['총'].isnull()]['건물용도'].unique() # array(['단독다가구'], dtype=object)
# [해석] 총이 null로 된 부동산들은 모두 단독다가구이므로 null값을 1로 추후 대체 예정

# '건축년도'의 null값 확인
df[df['건축년도'].isnull()]['접수연도'].unique()
df[df['건축년도'].isnull()]['자치구명'].unique()
df[df['건축년도'].isnull()]['법정동명'].unique()
df[df['건축년도'].isnull()]['물건금액'].max()
df[df['건축년도'].isnull()]['물건금액'].min()
df[df['건축년도'].isnull()]['건물면적'].max()
df[df['건축년도'].isnull()]['건물면적'].min()
df[df['건축년도'].isnull()]['총'].max()
df[df['건축년도'].isnull()]['총'].min()
df[df['건축년도'].isnull()]['건물용도'].unique()
# 건축년도가 null로 된 부동산들은 속성 값이 다양하므로 대체하기 애매하니 추후 삭제 예정
```

전체 null 값 확인 후 '총'과 '건축년도'의 null값 추가 확인 한 뒤 애매한 결측치 추후 삭제

# 데이터 수집 및 전처리 : 이상치 확인

```
# 5) 이상치 확인
# 범주형 변수의 경우
df['접수연도'].unique() # 이상치 없음
df['자치구명'].unique() # 이상치 없음
df['층'].unique() # 이상치(층 = 0)
df['건물용도'].unique() # 이상치 없음
```

# 연속형변수의 경우  
df.describe()  
'''

	접수연도	물건금액	건물면적	층	건축년도
count	640000.00000	6.400000e+05	640000.00000	591406.00000	637484.00000
mean	2019.640220	6.029282e+04	71.161121	6.663972	1983.039118
std	1.209938	6.799329e+04	67.255728	5.796018	193.321607
min	2017.000000	1.700000e+03	5.070000	-3.000000	0.000000
25%	2019.000000	2.400000e+04	39.480000	3.000000	1993.000000
50%	2020.000000	4.000000e+04	59.490000	5.000000	2002.000000
75%	2021.000000	7.490000e+04	84.800000	10.000000	2012.000000
max	2022.000000	1.108778e+07	3619.840000	73.000000	2022.000000
'''					



건축년도 최소값 0 & 물건금액 최댓값  
이 이상치 같으므로 추가 확인 필요



```
#####
## 주가 확인
#####
# 건축년도 0으로 된 부동산 확인
df[df['건축년도'] == 0] # [5977 rows x 8 columns]
df[df['건축년도'] == 0]['접수연도'].unique()
df[df['건축년도'] == 0]['자치구명'].unique()
df[df['건축년도'] == 0]['법정동명'].unique()
df[df['건축년도'] == 0]['물건금액'].max() # 950000
df[df['건축년도'] == 0]['물건금액'].mean() # 95219.54810105404
df[df['건축년도'] == 0]['물건금액'].min() # 11480
df[df['건축년도'] == 0]['건물면적'].max() # 273.96
df[df['건축년도'] == 0]['건물면적'].min() # 12.71
df[df['건축년도'] == 0]['층'].max() # 73.0
df[df['건축년도'] == 0]['층'].min() # 0.0
df[df['건축년도'] == 0]['건물용도'].unique() # ['아파트', '단독다가구']
# [해석] 건축년도가 0으로 된 부동산들은 속성 값이 다양하므로 대체하기 애매하니 추후 삭제 예정

# 물건금액 최댓값 확인 전에 먼저 정상범주를 계산해 보기
# 방법1) maxval1 = 151250.0
maxval1 = df['물건금액'].describe()['75%'] + (df['물건금액'].describe()['75%'] - df['물건금액'].describe()['25%']) * 1.5
# 방법2) maxval2 = 264272.6734560396
maxval2 = df['물건금액'].describe()['mean'] + 3*df['물건금액'].describe()['std']

# 물건금액 최댓값 확인
df.sort_values(by="물건금액", ascending=False).head()
'''
```

	접수연도	자치구명	법정동명	물건금액	건물면적	층	건축년도	건물용도
24796	2022	용산구	한남동	11087780	1742.90	NaN	1970.0	단독다가구
30280	2022	용산구	한남동	11087780	1742.90	NaN	1970.0	단독다가구
34576	2022	강남구	역삼동	3000000	2536.72	NaN	2007.0	단독다가구
35565	2022	성동구	성수동1가	3000000	1494.00	NaN	1984.0	단독다가구
245915	2020	서초구	서초동	2900000	2804.97	NaN	1991.0	단독다가구
'''								

# [해석] 행(24796 & 30280)의 물건금액이 최댓값으로 확인됨. 정상범주에 벗어났으므로 추후 삭제 예정

# 데이터 수집 및 전처리 : 결측치, 이상치 처리

```
#####
##### 2. 전처리#####
#####

# 1) 결측치 처리
df['층'] = df['층'].fillna(1) # 층의 null값은 1로 대체
df = df.dropna() # 2516 rows(건축년도=null) 삭제됨
df.isnull().sum() # 결측치 없음

# 2) 이상치 처리
df = df[df['건축년도'] > 0] # 5977 rows(건축년도=0) 삭제됨
df = df.drop([24796, 30280], axis=0) # 물건금액 최댓값 row 삭제됨

# 3) 자치구명 순으로 데이터셋 정렬
df = df.sort_values(by='자치구명', ascending=True)
```

In [22]: df

Out[22]:

			접수연도	자치구명	법정동명	물건금액	건물면적	층	건축년도	건물용도
402811	2019	강남구	수서동	76000	34.44	14.0	1993.0		아파트	
86603	2021	강남구	논현동	1550000	1048.49	1.0	1989.0		단독다가구	
231574	2020	강남구	역삼동	31000	24.61	19.0	2015.0		오피스텔	
231560	2020	강남구	자곡동	17500	24.12	3.0	2014.0		오피스텔	
231554	2020	강남구	수서동	139000	59.81	2.0	2014.0		아파트	
		...	...	...	...	...	...	...	...	
19475	2022	중랑구	목동	19000	30.00	11.0	2013.0		아파트	
496262	2018	중랑구	면목동	24000	46.46	5.0	2016.0		연립다세대	
314437	2020	중랑구	망우동	20000	35.93	4.0	2016.0		연립다세대	
192129	2020	중랑구	면목동	64260	185.18	1.0	1993.0		단독다가구	
232171	2020	중랑구	신내동	43300	49.77	1.0	1996.0		아파트	



전처리 결과 631,505의 행과 8개의 열이 나타남

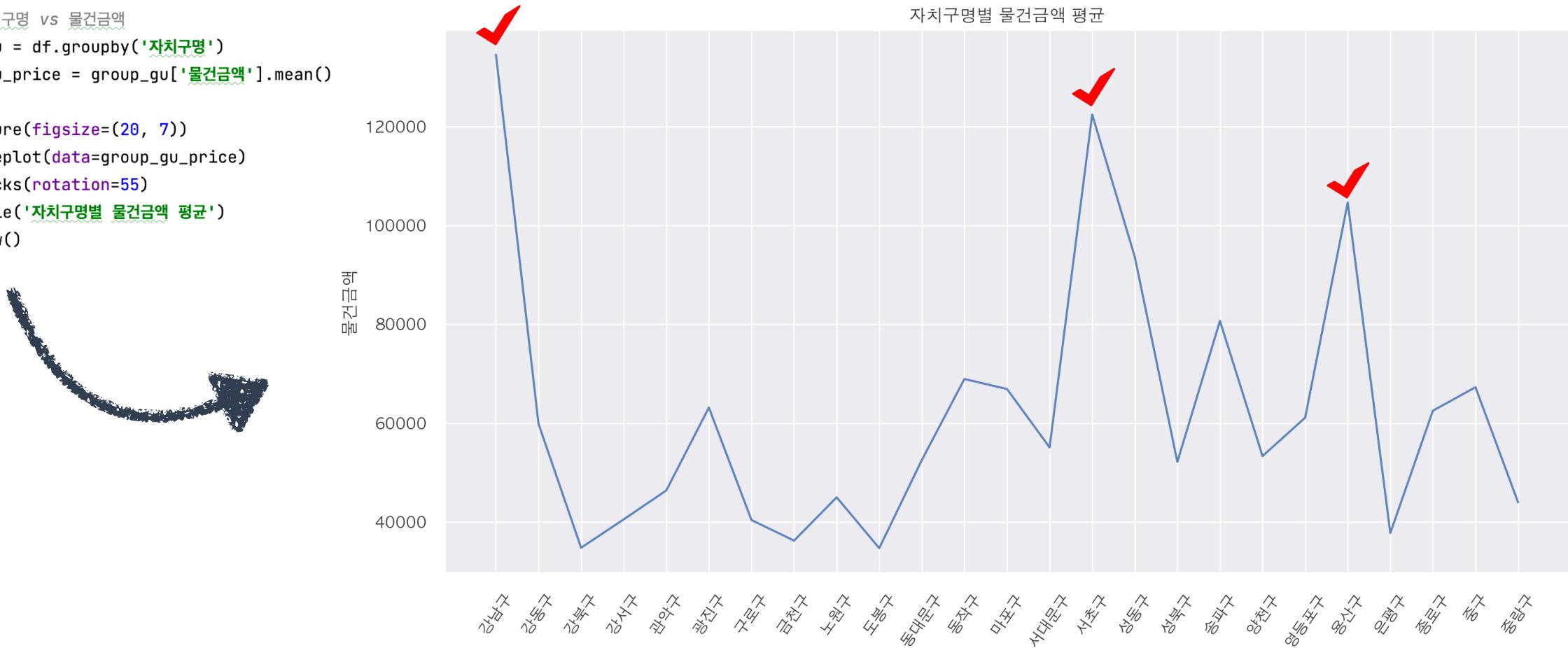


# Part 3

데이터 시각화 및 분석

# 데이터 시각화 및 분석 : 자치구명 vs 물건금액

```
# 1) 자치구명 vs 물건금액  
group_gu = df.groupby('자치구명')  
group_gu_price = group_gu['물건금액'].mean()  
  
plt.figure(figsize=(20, 7))  
sns.lineplot(data=group_gu_price)  
plt.xticks(rotation=55)  
plt.title('자치구별 물건금액 평균')  
plt.show()
```

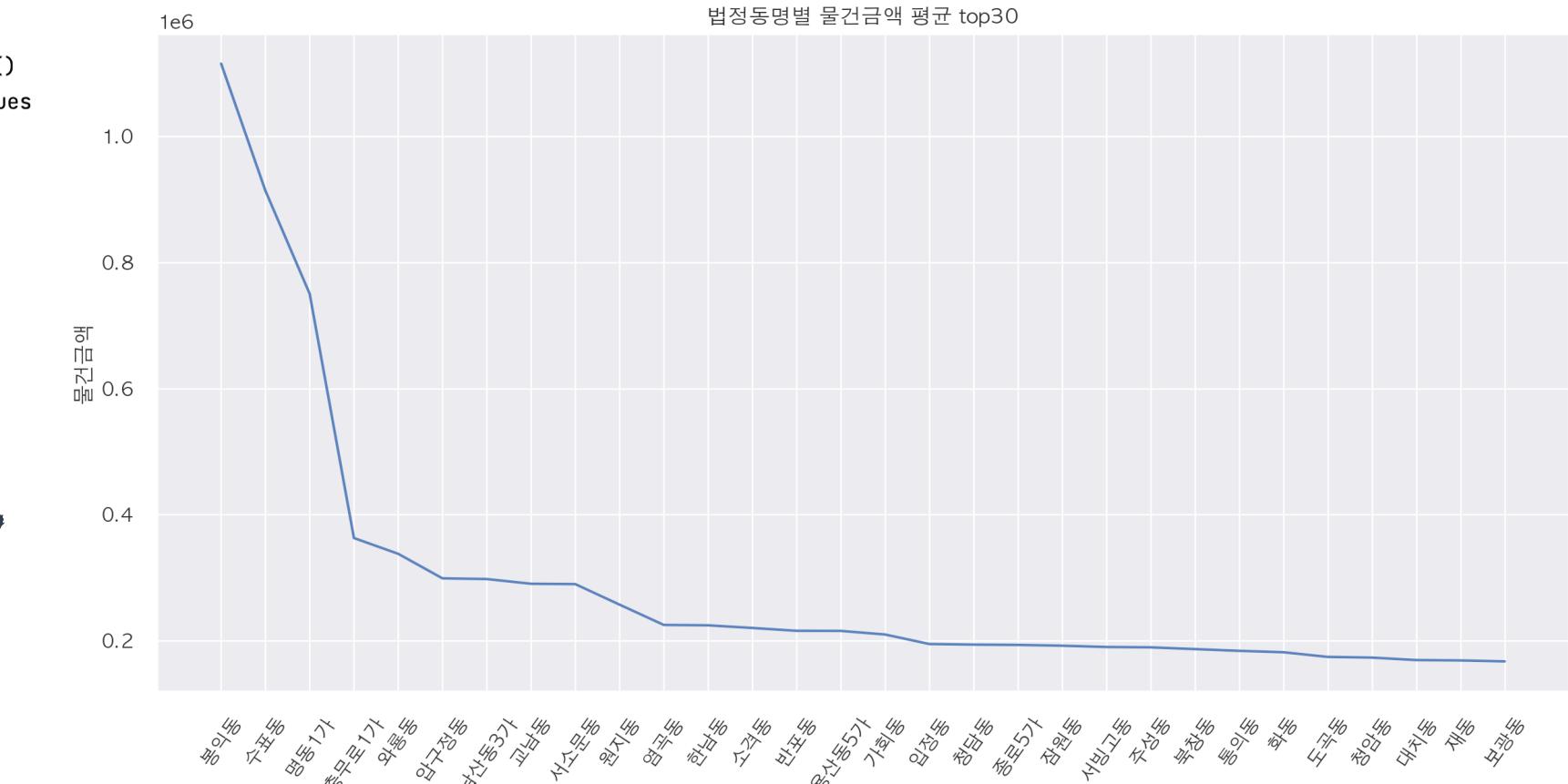


자치구별 부동산 거래가격 top3: 강남구, 서초구, 용산구

# 데이터 시각화 및 분석 : 법정동명 vs 물건금액

```
# 2) 법정동명 vs 물건금액
group_dong = df.groupby('법정동명')
group_dong_price = group_dong['물건금액'].mean()
group_dong_price = group_dong_price.sort_values

plt.figure(figsize=(20, 7))
sns.lineplot(data=group_dong_price[:30])
plt.xticks(rotation=55)
plt.title('법정동명별 물건금액 평균 top30')
plt.show()
```

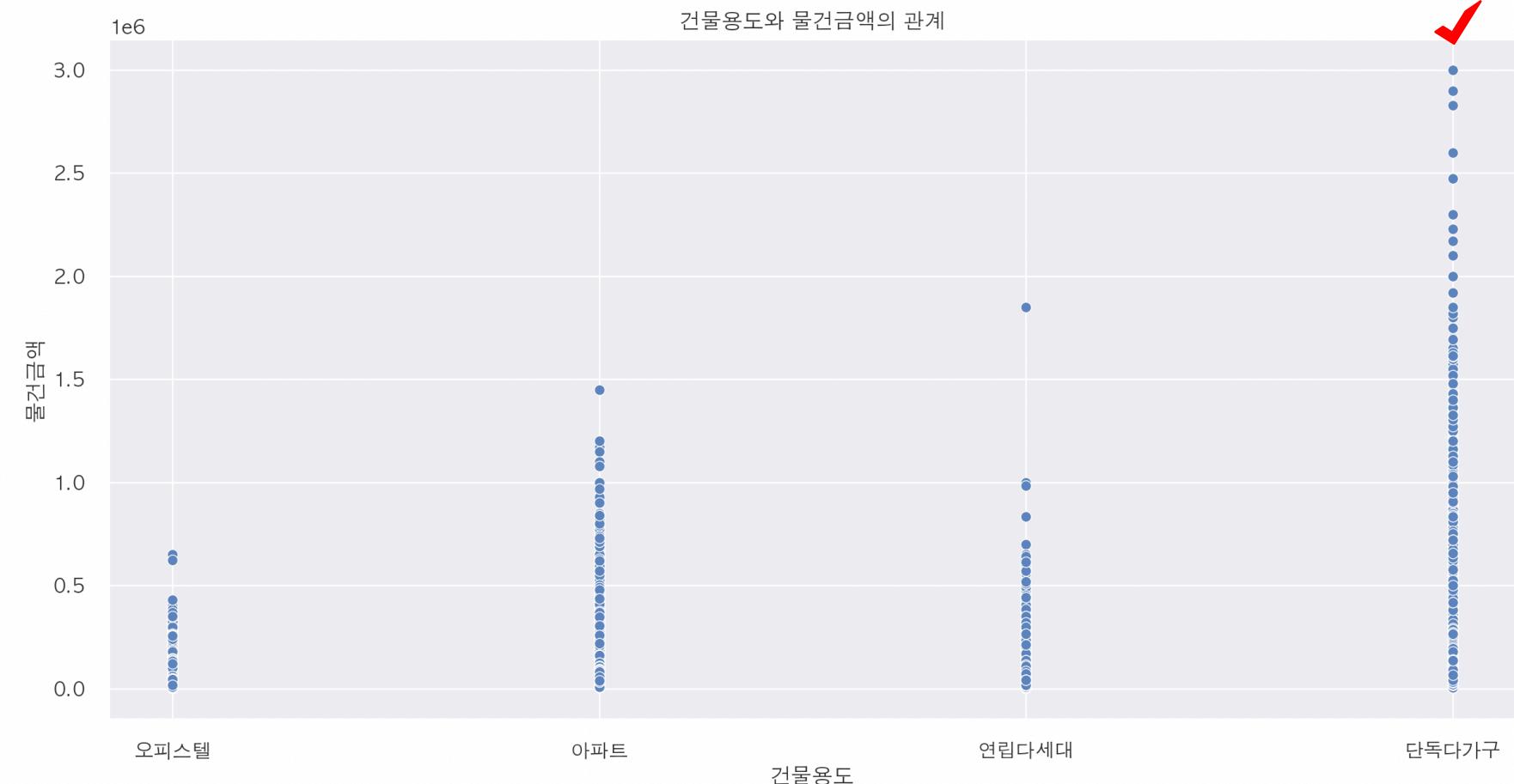
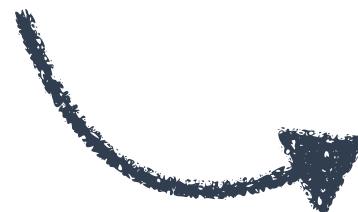


봉익동(종로구), 수표동(중구), 명동1가(중구) 순으로 거래가격이 높고 주로 단독다가구(상가) 건물임

# 데이터 시각화 및 분석 : 건물용도 vs 물건금액

```
# 3) 건물용도 vs 물건금액
plt.figure(figsize=(20, 7))
sns.scatterplot(data=df, x="건물용도", y="물건금액")
plt.title("건물용도와 물건금액의 관계")
plt.show()
df['건물용도'].value_counts()
'''

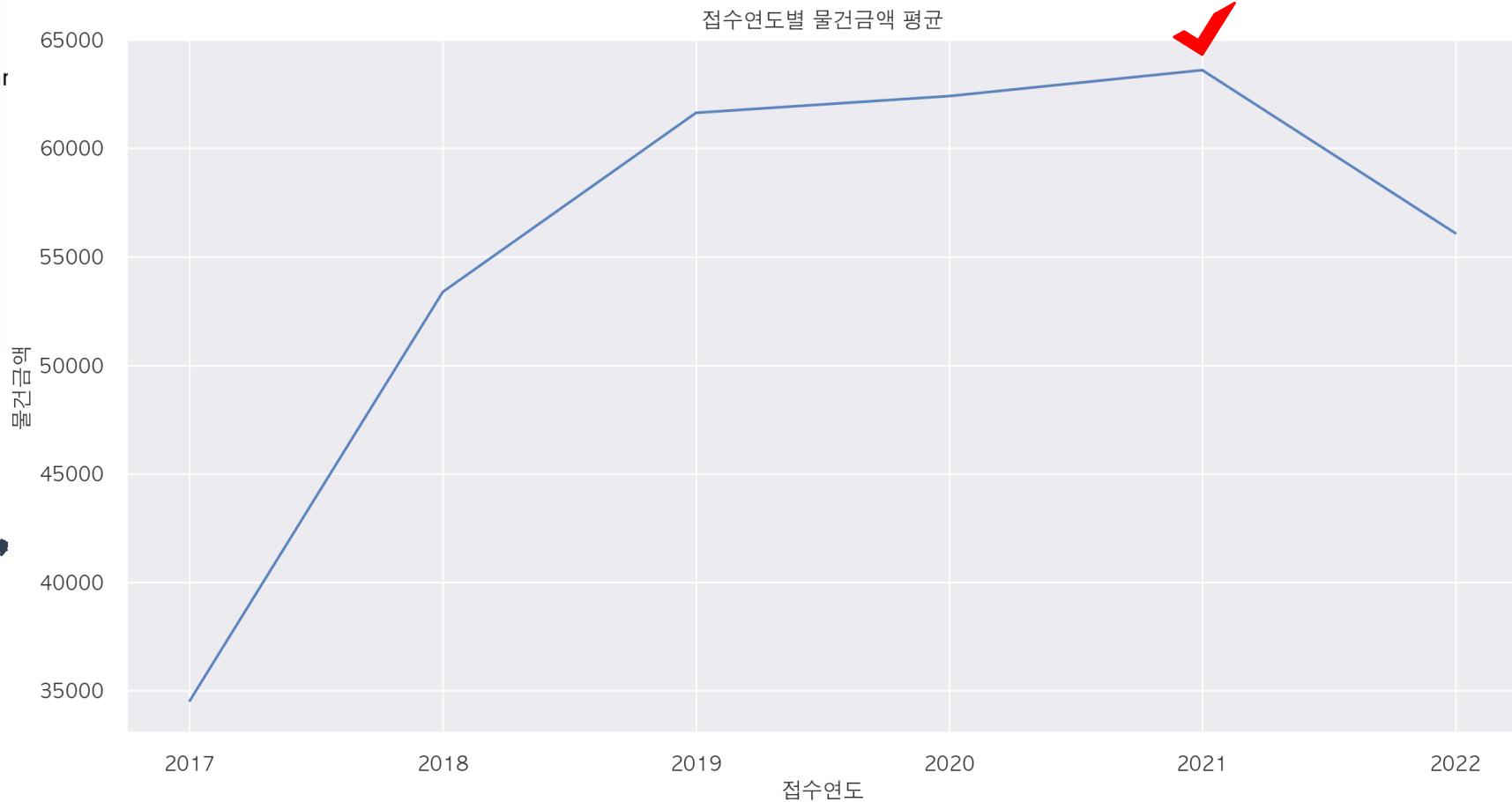
아파트      290943
연립다세대  229303
오피스텔    62717
단독다가구  48542
Name: 건물용도, dtype: int64
'''
```



단독다가구(상가) 개수는 다른 용도에 비하면 적지만, 거래가격은 상대적으로 높은 편

# 데이터 시각화 및 분석 : 접수연도 vs 물건금액

```
# 4) 접수연도 vs 물건금액  
group_year1 = df.groupby('접수연도')  
group_year1_price = group_year1['물건금액'].mean  
  
plt.figure(figsize=(20, 7))  
sns.lineplot(data=group_year1_price)  
plt.xlabel('접수연도')  
plt.ylabel('물건금액')  
plt.title('접수연도별 물건금액 평균')  
plt.show()
```

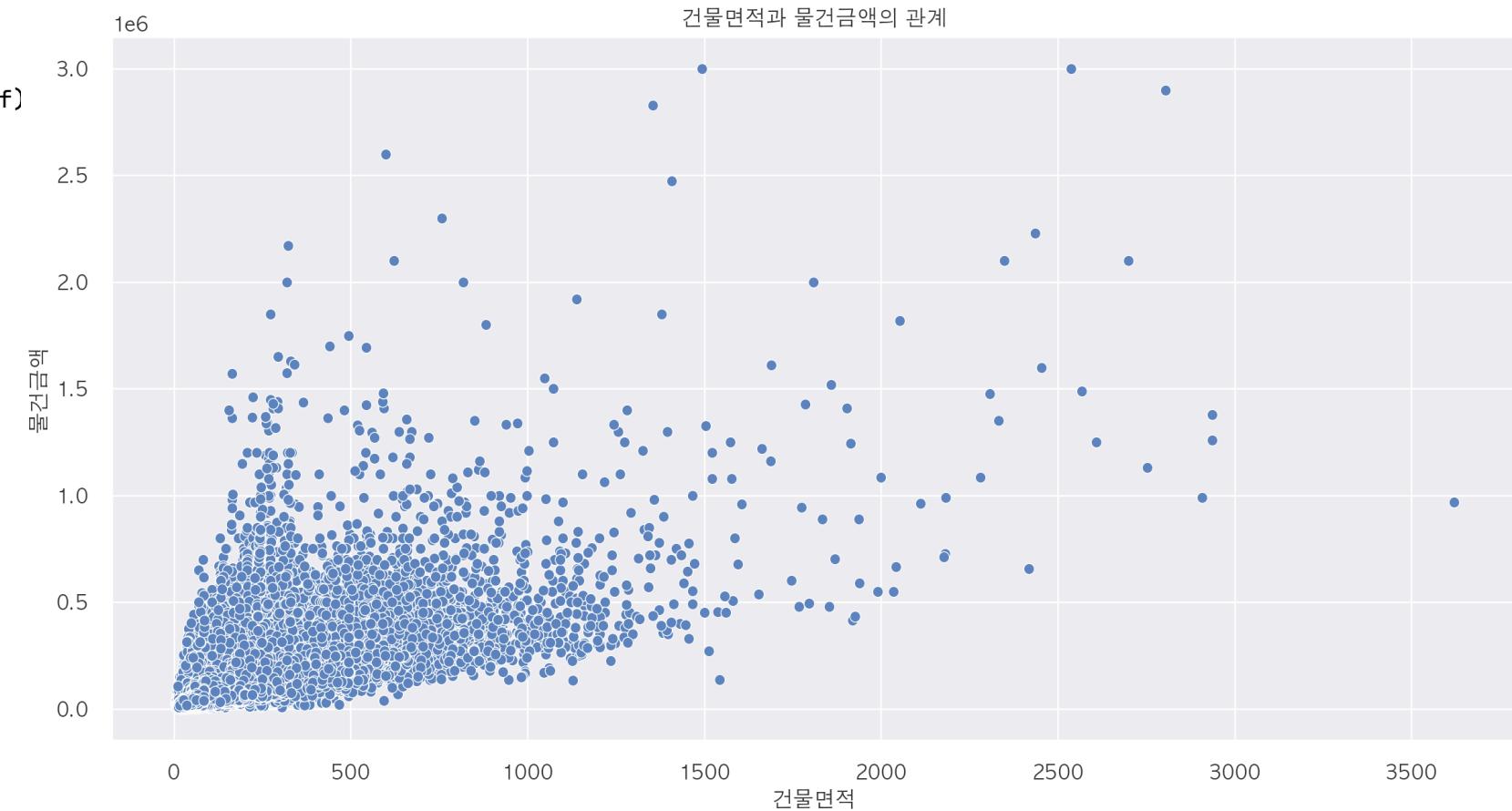
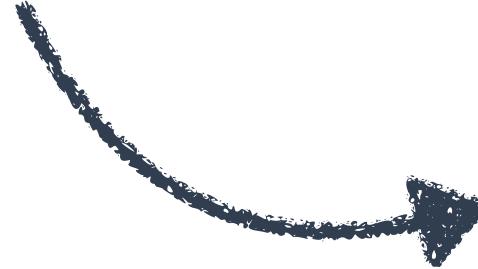


2017년부터 2021년까지 부동산 거래가격이 상승하다가 2021년부터 현재까지 떨어지고 있는 추세

# 데이터 시각화 및 분석 : 건물면적 vs 물건금액

# 5) 건물면적 vs 물건금액

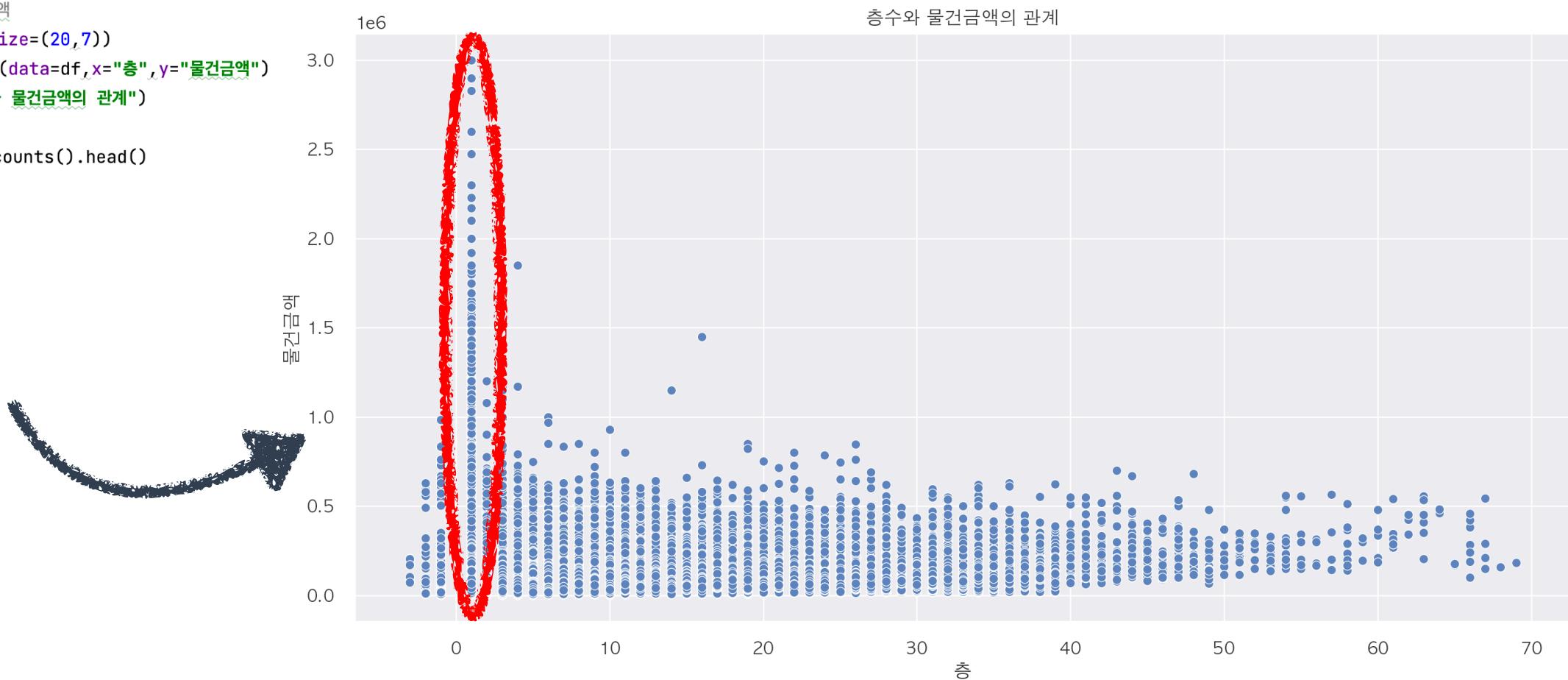
```
plt.figure(figsize=(20,7))
sns.scatterplot(x='건물면적', y='물건금액', data=df)
plt.title("건물면적과 물건금액의 관계")
plt.show()
```



건물 면적과 부동산 거래가격 간의 유의미한 상관성이 없음

# 데이터 시각화 및 분석 : 층 vs 물건금액

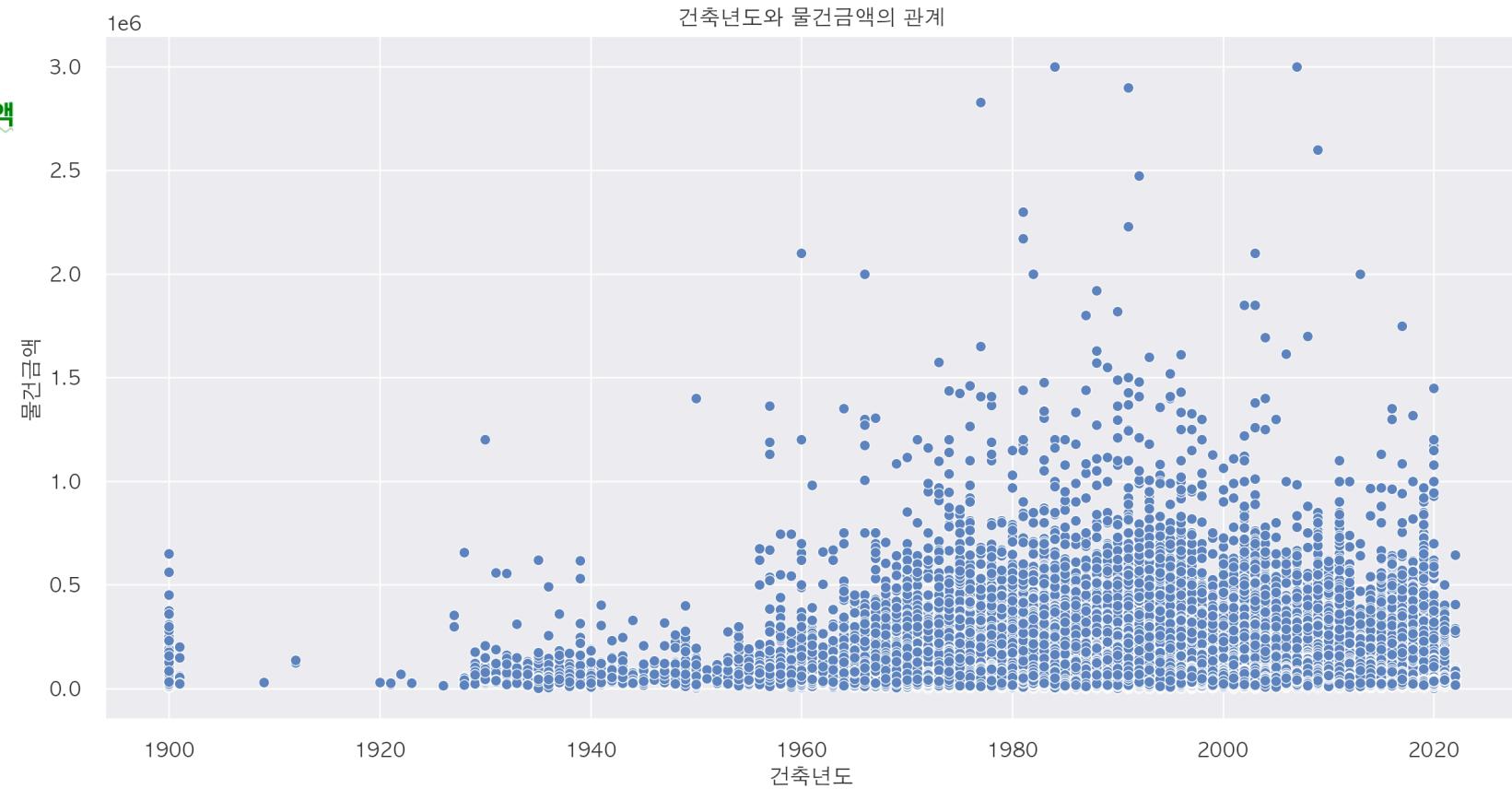
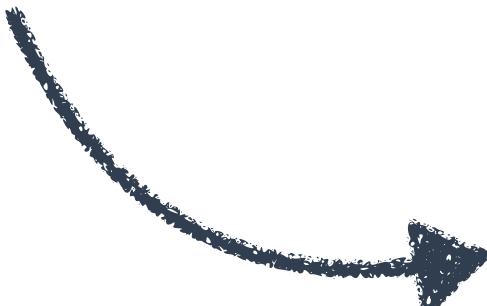
```
# 6) 층 vs 물건금액
plt.figure(figsize=(20,7))
sns.scatterplot(data=df,x="층",y="물건금액")
plt.title("층수와 물건금액의 관계")
plt.show()
df['층'].value_counts().head()
...
1.0    92724
2.0    80813
3.0    76032
4.0    65998
5.0    51319
...
```



# 데이터 시각화 및 분석 : 건축년도 vs 물건금액

# 7) 건축년도 vs 물건금액

```
plt.figure(figsize=(20, 7))
sns.scatterplot(data=df, x="건축년도", y="물건금액")
plt.title('건축년도와 물건금액의 관계')
plt.show()
```



- 건축년도에 따른 부동산 가격에 대한 지표를 보는 것은 의미가 없음
- 1950년대 한국전쟁, 1960년대 아파트 공급 시작을 비롯해 입지별로 가격차이가 나타남
- 신축 건물이라 해서 가격이 비싼 것은 아님

# 데이터 시각화 및 분석 : 연속형 변수 기준 상관계수 확인

```
# 연속형 변수를 대상으로 상관계수 확인 및 시각화
corr = df.corr(method = 'pearson')

...
    접수연도 물건금액 건물면적 총   건축년도
접수연도 1.000000 -0.011696 -0.056327 -0.056069  0.140506
물건금액 -0.011696 1.000000  0.645417  0.168094 -0.179911
건물면적 -0.056327  0.645417 1.000000 -0.046254 -0.254733
총      -0.056069  0.168094 -0.046254 1.000000  0.168133
건축년도  0.140506 -0.179911 -0.254733  0.168133 1.000000
...

heat = sns.heatmap(corr, cbar=True, annot=True,
                    annot_kws={'size':20}, fmt='.{2f}',
                    square=True, cmap='Blues')
```



상관계수로 보았을 때 물건금액과 건물면적의 상관성이 가장 높음



# Part 4

## 예측 모델

# 예측 모델 : 인코딩 및 스케일링

```
# 1) 인코딩
df_encoded = BinaryEncoder(cols=['자치구명', '법정동명', '건물용도']).fit_transform(df)
df_encoded.shape # (631505, 22)

# 2) 스케일링
# 2-1) subset로 X변수 생성 -> X변수 스케일링
X = df_encoded.drop(columns='물건금액')
X.shape # (631505, 21)

X_not_scaled = X.drop(columns=['접수연도', '건물면적', '층', '건축년도'])
X_not_scaled.reset_index(drop=True, inplace=True)
X_scaled = scale(X[['접수연도', '건물면적', '층', '건축년도']])
X_scaled = pd.DataFrame(X_scaled, columns=['접수연도', '건물면적', '층', '건축년도'])
X_new = pd.concat(objs=[X_not_scaled, X_scaled], axis=1)
X_new.shape # (631505, 21)

# 2-2) subset로 y변수 생성 -> y변수 로그변환
y = df_encoded['물건금액']
y.shape # (631505,)
y_log = np.log(y)

# 3) train/test split
X_train, X_test, y_train, y_test = train_test_split(X_new, y_log, test_size=0.3, random_state=42)

print("X Train:", X_train.shape) # X Train: (442053, 21)
print("X_test:", X_test.shape) # X_test: (189452, 21)
print("y_train:", y_train.shape) # y_train: (442053,)
print("y_test:", y_test.shape) # y_test: (189452,)
```

- Binary Encoding으로 X변수를 스케일링 하였음.
- Binary Encoding 이란 이진법을 활용해 변수를 나누는 것으로 one-hot 인코딩과 유사하지만 늘 어나는 변수의 개수가 훨씬 적음.
- X변수는 표준화로 스케일링
- Y변수는 로그변환으로 스케일링
- 훈련셋과 테스트셋을 7 : 3 비율로 나눔

# 예측 모델 : 6개의 모델 생성

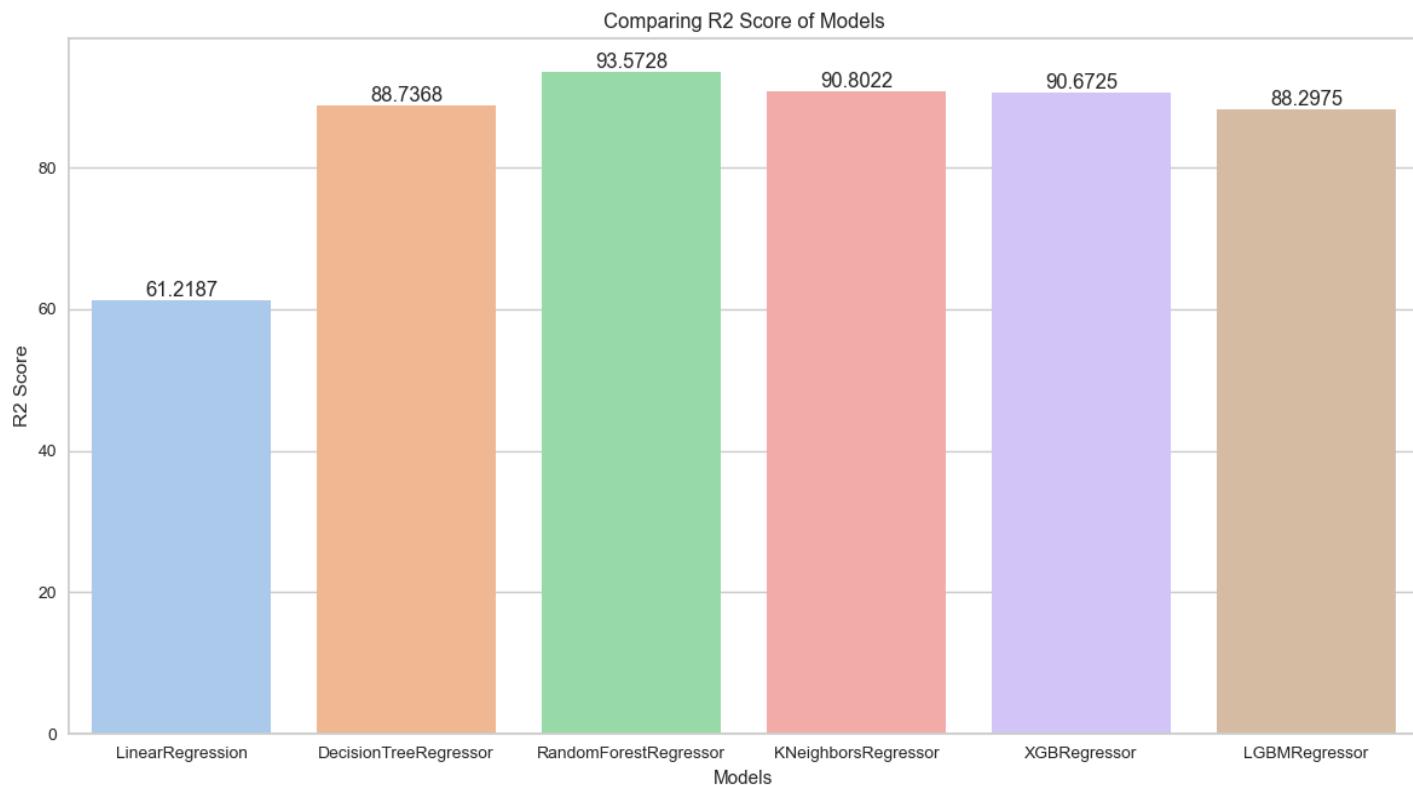
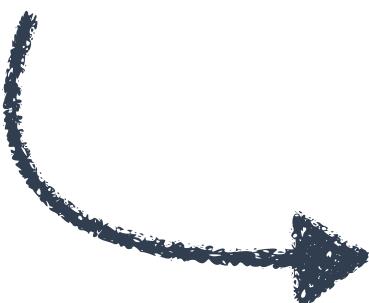
```
LR = LinearRegression()
DTR = DecisionTreeRegressor(random_state=42)
RFR = RandomForestRegressor(random_state=42)
KNR = KNeighborsRegressor()
XGB = XGBRegressor(random_state=42)
LGB = lgb.LGBMRegressor(random_state=42)

li = [LR, DTR, RFR, KNR, XGB, LGB]
d = {}
for i in li:
    model = i.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(i, ":", r2_score(y_test, y_pred)*100)
    print(i, ":", mean_squared_error(y_test, y_pred))
    d.update({str(i):i.score(X_test, y_test)*100})
...
LinearRegression() : 61.21873279835921
LinearRegression() : 0.2478906454911216
DecisionTreeRegressor(random_state=42) : 88.73676777187892
DecisionTreeRegressor(random_state=42) : 0.0719948085457922
RandomForestRegressor(random_state=42) : 93.57277913552876
RandomForestRegressor(random_state=42) : 0.041082925952981
KNeighborsRegressor() : 90.80217378996667
KNeighborsRegressor() : 0.05879269144211598
XGBRegressor(random_state=42) : 90.67247429588016
XGBRegressor(random_state=42) : 0.05962173323545928
LGBMRegressor(random_state=42) : 88.29748387949567
LGBMRegressor(random_state=42) : 0.0748027200838692
...
```

- 6개의 모델로 정확도를 구해봄
- 결과 RandomForestRegressor가 가장 높은 정확도를 나타냈음
- 그 다음으로 KNN 모델, XGBoost 모델 순서로 높은 정확도가 나옴
- 다음 페이지에 각 알고리즘의 정확도를 시각화하여 나타냄

# 예측 모델 : 알고리즘 vs 정확도

```
# 모델 r2 score 비교 시각화
x_label = ['LinearRegression', 'DecisionTreeRegressor', 'RandomForestRegressor',
           'KNeighborsRegressor', 'XGBRegressor',
           'LGBMRegressor']
plt.figure(figsize=(20, 7))
r2 = list(d.values())
ax = sns.barplot(x_label, r2, palette='pastel')
ax.bar_label(ax.containers[0])
plt.xlabel('Models')
plt.ylabel('R2 Score')
plt.title('Comparing R2 Score of Models')
```



RandomForestRegressor 모델의 정확도가 약 94%로 가장 높음

# 예측 모델 : 최적의 파라미터 확인

```
# 1) GridSearchCV 모델 생성
model_RFR = RFR.fit(X_train, y_train)

parmas = {'n_estimators' : [100, 200],
          'max_depth' : [None, 5],
          'min_samples_split' : [2, 6],
          'min_samples_leaf' : [1, 6]} # dict 정의

grid_model = GridSearchCV(model_RFR, param_grid=parmas,
                           scoring='neg_mean_squared_error', cv=3, n_jobs=-1) # neg_mean_squared_error

grid_model = grid_model.fit(X_train, y_train)

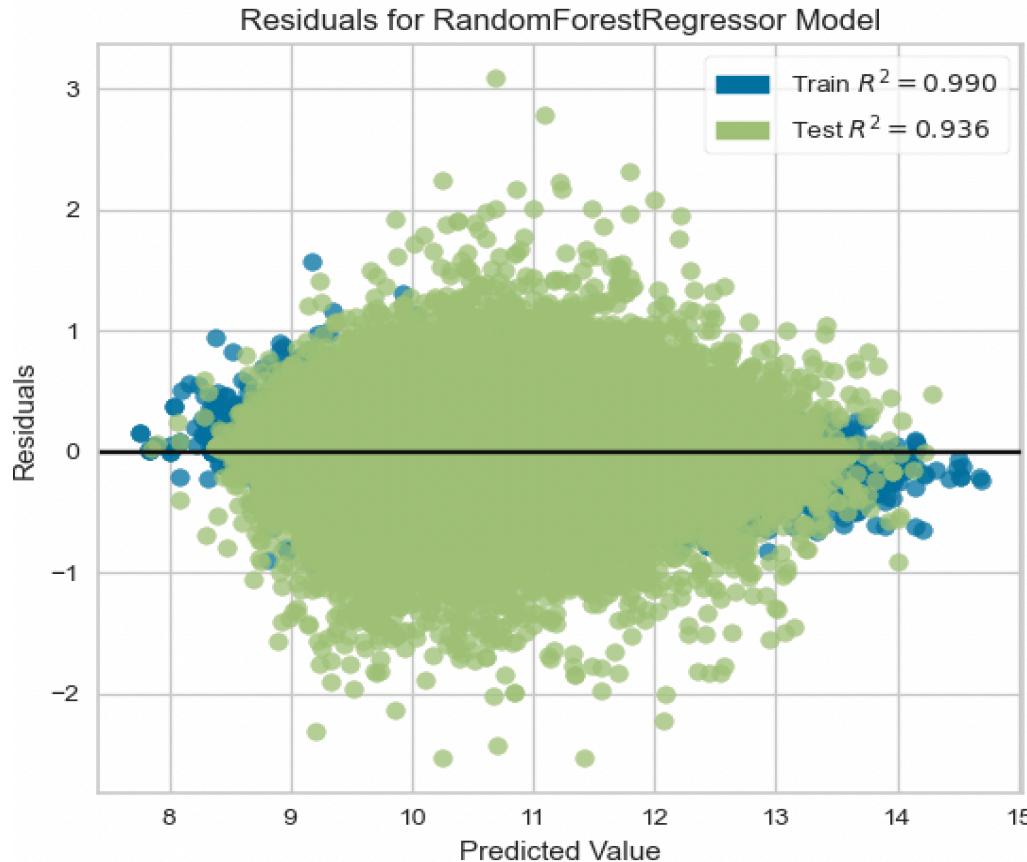
# 2) Best score & parameters
print('best score =', grid_model.best_score_) # best score = -0.04422525910036088

print('best parameters =', grid_model.best_params_)
# best parameters = {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

# 3) Best parameters 적용하여 모델 만들기
best_model = RandomForestRegressor(random_state=42, n_estimators=200).fit(X_train, y_train)
score = best_model.score(X_test, y_test)
print('best score =', score*100) # best score = 93.60748540518264
```

- 최적의 파라미터: 'max\_depth': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 200
- 최적의 파라미터를 적용한 모델의 정확도는 93.61(이전의 93.57보다 조금 높음)

# 예측 모델 : overfitting 유무 확인



# 4) over-fitting 유무 확인 및 시각화

```
train_score = best_model.score(X_train, y_train)  
print('train score:', train_score) # train score: 0.9898107495063089
```

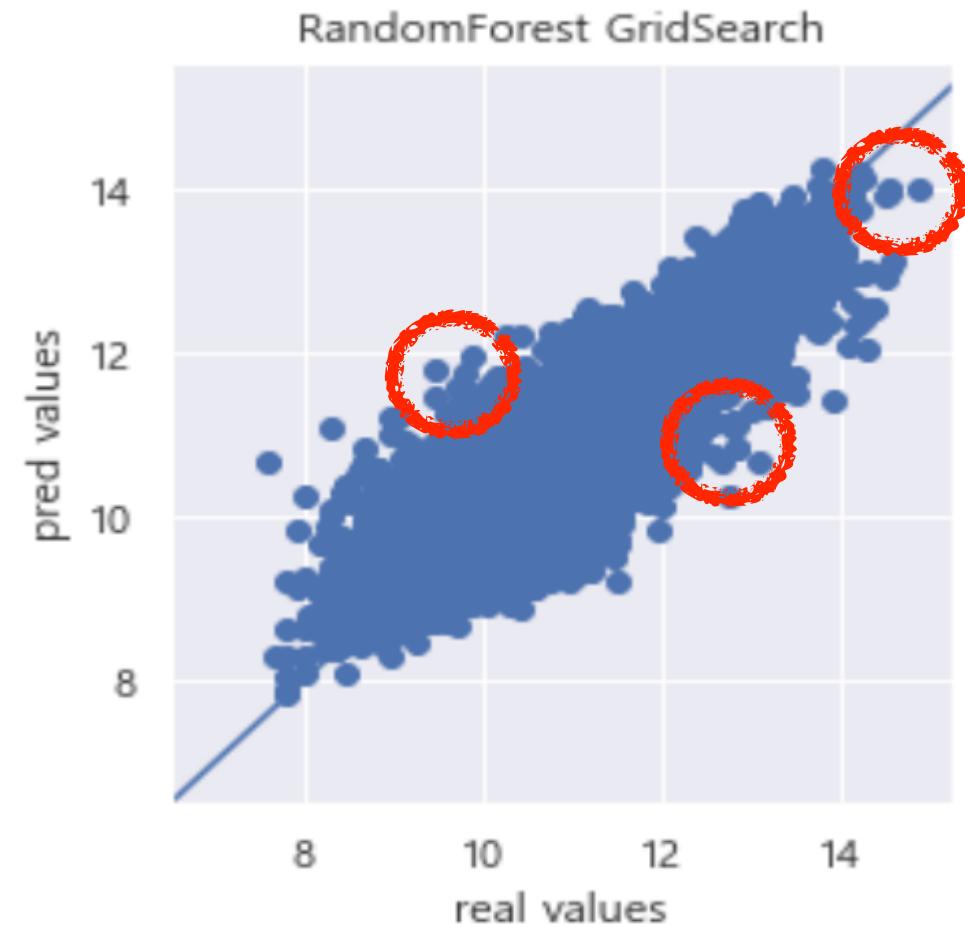
```
test_score = best_model.score(X_test, y_test)  
print('test score:', test_score) # test score: 0.9360748540518264
```

- 모델의 예측력이 높아 overfitting이 의심되어 훈련셋과 테스트셋의 r2 score를 확인한 후에 시각화함
- 훈련셋과 테스트셋의 결정계수가 약 0.05 정도의 차이가 나므로 overfitting이라고 볼 수 없음

# 예측 모델 : 모델 튜닝 전후 비교



VS



모델 튜닝 후의 정확도가 더 높게 나왔지만 큰 차이는 없음

# 예측 모델 : DNN 모델 생성

```
import tensorflow as tf
import numpy as np
import random as rd
from keras import Sequential
from keras.layers import Dense

tf.random.set_seed(42)
np.random.seed(42)
rd.seed(42)

# DNN model layer 구축
model = Sequential()
model.add(Dense(256, input_shape=(21,), activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```

- 좀 더 예측력이 높은 모델을 도출하기 위해 딥러닝을 시도함
- Keras 내부 weight seed 적용함
- 레이어 수를 3개~10개로 지정하여 가장 최적의 레이어 수를 구함
- 그 결과 최적의 레이어 수는 5개로 판단됨

# 4 예측 모델 : DNN 모델 학습

```
# train set / test set / val set split
X_train, X_test, y_train, y_test = train_test_split(X_new, y_log, test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.3, random_state=42)

# model 학습과정 설정
model.compile(loss='mse', optimizer='adam', metrics=['mse'])

# model training
model_fit = model.fit(x=X_train, y=y_train,
                       epochs=30, verbose=1,
                       batch_size=20000, validation_data=(X_val, y_val))
```

epochs(반복학습 횟수)를 10~3,000회까지 지정해 보았고, batch\_size(1회 공급 데이터량)는 50~20,000개까지  
지정해 본 결과 epochs은 30회, batch\_size는 20,000개가 가장 적합함

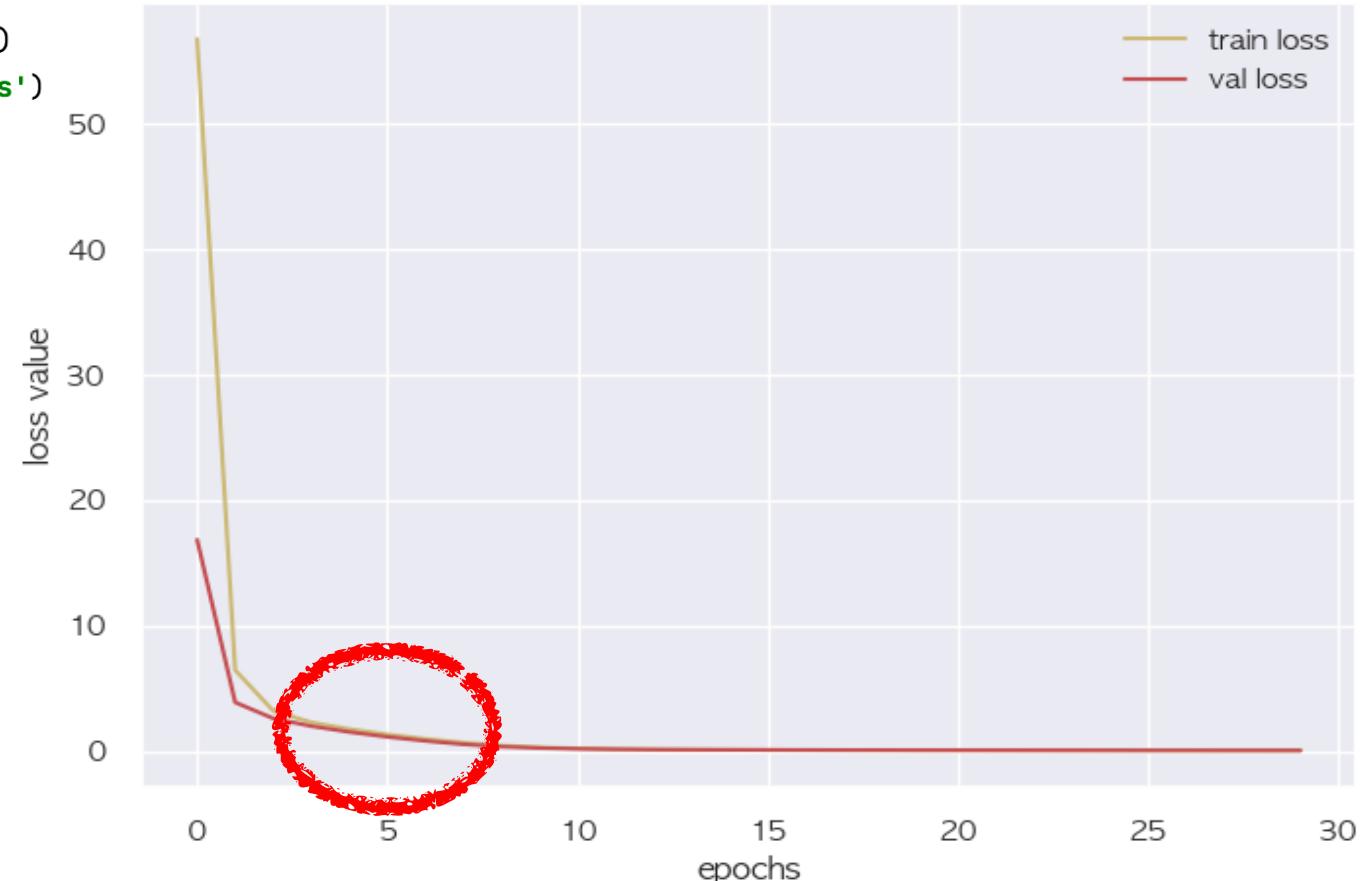
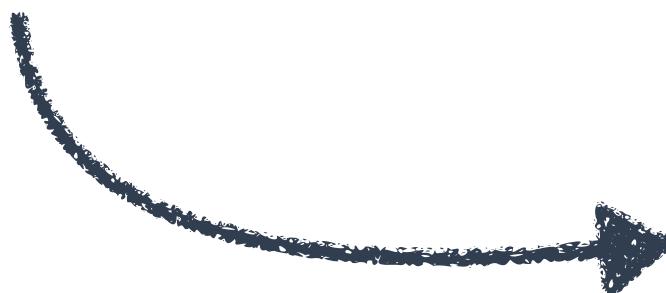
```
# model evaluation/prediction
loss, mse = model.evaluate(X_test, y_test)
print('mse : ', mse) # mse : 0.0881928876042366

y_predict = model.predict(X_test)
r2_y_predict = r2_score(y_test, y_predict)
print('R2 : ', r2_y_predict) # R2 : 0.8618141660484626
```

RandomForestRegressor 모델의 mse는 0.04이고 r2 score는 93.57인 반면,  
DNN model의 mse는 0.09이고 r2 score는 86.18로 예상과는 달리  
RandomForestRegressor 모델의 예측력이 더 좋음

# 예측 모델 : DNN 모델 loss value 시각화

```
# loss vs val_loss  
plt.plot(model_fit.history['loss'], 'y', label='train loss')  
plt.plot(model_fit.history['val_loss'], 'r', label='val loss')  
plt.xlabel('epochs')  
plt.ylabel('loss value')  
plt.legend(loc='best')  
plt.show()
```



그래프 결과 epochs 5이후로부터 loss value의 차이가 거의 없음

# Part 5

회고 및 고찰

- 총 6가지 모델 중 RandomForestRegressor 모델의 예측력이 **약 94%**로 가장 높음.
- 가장 높은 가격의 부동산의 위치는 **자치구별**로는 강남구, 서초구, 용산구 순으로 나타났고, **법정동별**로는 봉익동(종로구), 수표동, 명동1가(중구) 순으로 높게 나타났음.
- 부동산 가격과 가장 높은 상관관계를 나타난 변수는 **건물면적, 자치구명, 건물용도**로 나타났음.
- 부동산 가격은 2017년부터 2021년 상반기까지는 상승하다가 **2021년부터 현재까지는 하락**하고 있는 추세임.

- 부동산 가격에 영향을 줄 수 있는 데이터셋에 포함되어 있지 않은 방의 개수나 편의시설 등과 같은 변수들이 있는데, 이러한 정보들이 있었으면 모델의 예측능력을 더욱 향상시킬 수 있었을 텐데 그러지 못해 아쉬움.
- 그리드 서치와 랜덤 서치를 이용하여 최적의 파라미터를 구하기 위해선 충분한 시간과 반복적인 트레이닝이 필요한데 데이터셋의 용량이 큼과 동시에 장비가 이러한 과정을 받쳐주지 못하여 트레이닝하는데 절대적인 시간이 부족하여 모델 예측력에 큰 변화는 없었음.