데이터과학 프로젝트 보고서

* Housing Price Prediction

2010210007 신동주

2014210049 강명훈

2014210045 강종욱

2014210029 임재현

# 1. 문제 정의

## 동기

* 독립을 준비하는 사회 초년생에게 첫 보금자리를 마련하는 일은 언제나 난해하다. 실제로 이런 상황을 마주한 입장에서, 가격과 거주 환경사이에서 적절히 타협점을 찾는 것이 가장 중요하다. 하지만, 별다른 배경지식이 없는 개인이 기존의 부동산 중개 플랫폼에서 적절한 매물임을 판단하는 것은 불가능에 가깝다. 따라서 이러한 상황에서 하나의 가이드라인으로 사용될 수 있는 객관적인 수치의 필요성을 체감하였다.

## 문제 정의

* 부동산 관련 지식이 부족한 개인이 원하는 조건에 부합하는 매물의 가격을 유추할 수 있도록 하는 것.
* 부동산 중개업자들이 실제 매물 소유주의 가격 책정에 신뢰성 높은 도움을 줄 수 있도록 하는 것.

# 2. Team Role

신동주 : 데이터 수집 및 파싱, Preprocessing

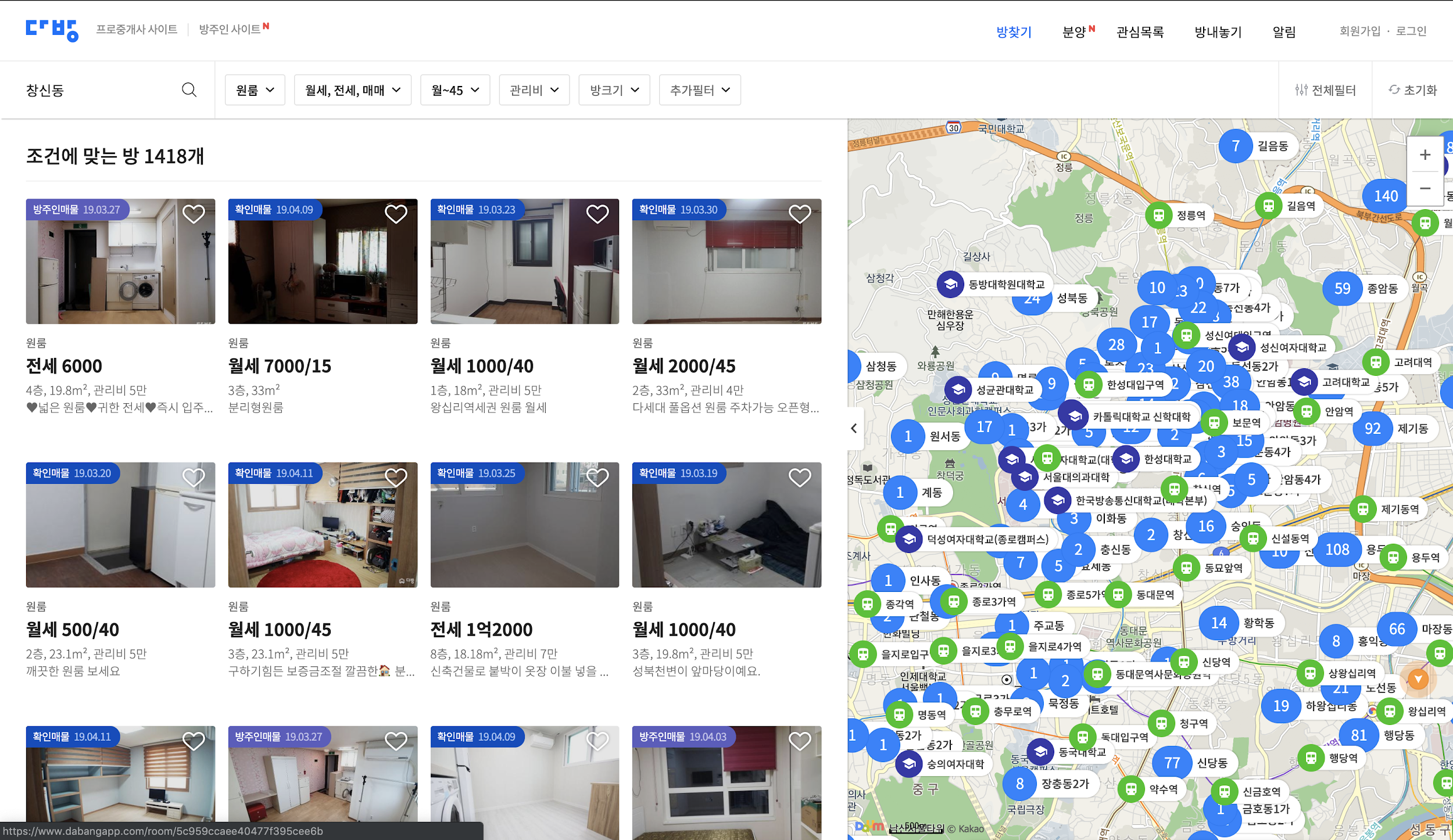
강명훈 : 데이터 모델링 및 피어 리뷰

강종욱 : 데이터 시각화 및 피어 리뷰

임재현 : 데이터 학습 및 피어 리뷰

# 3. Dataset

**Raw Data**



**Acquired Data by API**

* 데이터 : 약 25만개

# 

# 4. 개발 환경

* 데이터 수집 환경 : Python3(라이브러리:requests,Django,PyMysql….)
* 데이터 전처리 환경 : Microsoft Excel
* 데이터 개발 환경 : Anaconda Jupyter Notebook, Python3(라이브러리 : sklearn, numpy, pandas, matplotlib)

# 5. 과정

## 5 - 1) 데이터 수집

(1) 데이터 수집 API 분석

1. 현재 판매중인 매물 정보 API

다방의 현재 판매중 매물 정보 API는 법정동 주소별로 나누어져 있다.

(ex : 1129012500 = 안암동5가의 법정동 코드

https://www.dabangapp.com/api/2/room/list/region?&code=1129012500&page=1)

API의 데이터는 JSON 으로 제공되며 현재 판매중인 부동산의 가격정보,

방크기, 옵션, 주소와 같은 상세정보가 모두 포함되어있다.

API에서는 현재 판매중인 매물정보만 제공 해주므로 매물의 판매여부를 확인하기

위해서는 주기적인 데이터 수집을 통한 현재 판매중 매물 리스트 비교 방식이

필요하다.

2. 판매 상태 확인 API

매물의 아이디를 이용하여 방 1개의 상세정보를 제공해주는 API는 매물이 현재

판매완료, 삭제된 상태이여도 데이터를 제공해준다.

해당 데이터에는 판매완료/삭제/판매중을 구분 할 수 있는 데이터가 존재한다.

(2) 데이터 수집

1. 학습의 정확성과 신뢰성을 위해서 데이터 중에서 지역의 범위를 전국->서울로 한정지었고, 팔렸거나 안팔린 모든 데이터 중 팔린 데이터들만 수집하는 것을 목표로 했다.
2. Python의 requests 라이브러리를 이용하여 모든 법정동에 대해 현재 판매중인 부동산 데이터를 수집.  
   수집한 부동산 데이터는 판매매물 처리를 위해 mysql 데이터베이스를 사용하여 저장한다. 데이터 수집의 시간을 단축하기 위해 8개의 쓰레드를 사용하여 병렬처리하였다.
3. 1의 크롤링 모듈을 Django (Python web server Framework)를 이용하여 웹서버로 구축한뒤 원격 실행이 가능하도록 하였다. 또한 Linux의 Crontab을 사용하여 크롤링이 하루에 4번 주기적으로 실행되게 설정했다.
4. mysql에 저장한 매물과 2번에서 크롤링 한 매물을 비교하여 API에서 빠진 매물들의 ID를 확인한다. 방상세정보 확인 API를 이용하여 매물의 상태가 판매완료 또는, 삭제됨을 구분한다.
5. 약 40일동안 쌓은 데이터를 CSV로 추출하여 학습을 진행하였다.

## 5 - 2) 데이터 전처리

1. 엑셀을 이용한 Raw Data 필터링

API로 추출한 데이터에 존재하는 feature 항목은 다음과 같다.  
id, platform\_id, sale\_id, user\_id, **addr\_d1 ~ addr\_d5**, **near\_subway**, **sell\_type**, **deposit**, **rent**, **room\_type**, room\_type\_str, **room\_floor**, building\_type, building\_total\_floors, **maintenance\_cost**, maintenance\_items, **room\_size\_m2**, **room\_size**, **options**, **parking**, **write\_date**, lat, lng, is\_selling, thumbnail, need\_update, sell\_date, sell\_by\_flag, cdate, viewcnt, sort, lat\_m1, lng\_m1

1. id, platform\_id, sale\_id, user\_id : Identify를 하기 위한 항목으로, unique한 속성을 가지고 있어서 쓰지 않을 feature이므로 삭제했다.
2. addr\_d1 ~ addr\_d5 : 주소상의 시, 군, 구, 동, 읍, 리 등을 구분해놓은 항목으로, 이 중 서울의 경우는 동으로 데이터들을 구분 가능하므로 addr\_d3만 남기고 나머진 삭제했다.
3. near\_subway : 역세권을 나타내는 항목으로, 근처에 역이 있으면 1, 없으면 0인데 공백의 경우도 존재하여 이는 0으로 통일했다.
4. sell\_type : 월세, 전세, 매매에 따라 one-hot vector로 변경했다.
5. room\_type\_str : room\_type과 같은 값을 가리키므로 room\_type으로 통합했다.
6. building\_type : 데이터가 존재하지 않으므로 삭제했다.
7. building\_total\_floors : 매물 건물의 최고 층수를 나타내는 항목인데, 공백도 많았고, room\_floor라는 가격 예측에 더 중요한 feature가 있었기 때문에 삭제했다.
8. maintenance\_cost : 아파트 유지비를 나타내는 항목으로, 0을 입력한 데이터도 있었고 공백으로 되어 있는 데이터도 존재했다. 보통 매물을 올릴 때 유지비가 0이라는 것을 강조하기 위해 0을 입력한 경우가 많을 것이라 생각하였고, 따라서 공백으로 되어 있는 데이터는 0으로 채워넣기 보다 전체 데이터의 평균으로 입력했다.
9. room\_size\_m2, room\_size : 둘 다 매물의 크기를 나타내는 항목으로, 제곱미터와 평방미터의 실제 값이 같을 것으로 예상되었으나, 변환을 하였을 때 일치하지 않았기 때문에, 이후 feature selection을 통해 하나를 선택하기로 결정했다.
10. options : 매물에 존재하는 에어컨, 전자레인지, 인터넷, TV 등 총 12개의 옵션들을 나타내는 항목으로, 각각의 option들을 one-hot vector로 변경, option이 아예 공백으로 되어 있는 데이터들은 일반적으로 거주지에 인터넷, TV 등 필수요소들이 존재하지 않는다고 보긴 어려우므로 one-hot vector를 변경한 후 각각의 옵션들마다의 평균값을 입력했다.
11. lat, lng, lat\_m1, lng\_m1 : 의미하는 값이 addr\_d3와 동일하므로 삭제했다.
12. is\_selling, sell\_by\_flag : 모든 데이터가 0이나 1로 동일하므로 삭제했다.
13. thumbnail, need\_update, cdate, viewcnt : 썸네일 이미지, 크롤링한 날짜 등 다방 플랫폼과 관련된 데이터이므로 삭제했다.
14. sell\_date : 크롤링 한 날 이전의 팔린 매물들은 모두 크롤링 한 날로 통일되어 추출되어서, 데이터 측정 오류가 아닌 수집 오류 이므로 삭제했다.
15. sort : 데이터가 아예 공백으로 되어 있었으므로 삭제했다.
16. deposit, rent : 우리가 예측하고자 하는 매물 가격을 의미하는 항목인데, deposit은 보증금, rent는 집세를 나타낸다. 매물의 type에 따라 데이터는 deposit에 전세, 매매 가격을 적고 rent에 0을 입력한 경우가 존재했기 때문에 이를 하나의 단위로 통일해야 했다. 따라서 KOSIS라는 국가통계포털의 전월세전환율을 이용하여 charge라는 새로운 칼럼을 만들었다. 하지만 이 때 데이터 샘플에는 2014년부터 등록된 매물이 존재했기 때문에 전월세전환율을 14년도부터 평균을 내기에는 정확도가 떨어질 것이라 예측했다. 또한 약 13만 여개의 데이터 샘플중에서 18년도 이전의 데이터는 7634개밖에 없었기 때문에 이 데이터는 버리고 2018.1~2019.3의 데이터만 남기고 이 기간 내의 전월세전환율 평균을 이용해 charge 칼럼을 추가 했고, 계산식은 deposit \* 5.322755977(=전월세전환율) / (12\*100) + rent와 같다.

* 위와 같이 액셀 작업을 마친 뒤 최종적으로 남은 feature는 addr\_d3, room\_size\_m2, room\_size, near\_subways, room\_type, room\_floor, sell\_type0\_month, sell\_type1\_all sell\_type2\_sell, maintenance\_cost, op0\_aircond, op1\_induction, op2\_microwave, op3\_washer, op4\_doorlock, op5\_bed, op6\_bidet, op7\_desk op8\_closet, op9\_tv, op10\_refrig, op11\_gas, op12\_shoe, parking, write\_date, charge이고 총 26개이다.

1. sklearn API를 이용한 데이터 전처리  
   - LabelEncoder(): room\_type의 경우 값이 ‘옥탑’과 같은 문자열과 원룸을 의미하는 정수 1이 뒤섞여 있는 상태로 통합을 위해 방의 크기가 작은 것부터, room\_floor의 경우 반지하를 나타내는 ‘-1’가 있어서 정수를 입력으로 받는 OneHotEncoder를 위해, addr\_d3의 경우 문자열로 구성되어 있는 동명칭을 한글 자모음순서대로 0부터 시작하는 정수형 라벨로 변환했다. 이후에 사용할 OneHotEncoder를 위한 전처리 과정이다.  
     
   - OneHotEncoder(): addr\_d3, room\_type, room\_floor의 경우 연속적인 정수 값을 갖는 Feature지만, 실제로는 크기를 의미하는 값이 아닌 단순한 categorical value로, 해당 값을 그대로 학습시킨다면 의도하지 않은 오류가 발생할 수 있는 가능성 존재. 이를 해결하기 위해 OneHotEncoder를 이용하여 OneHotVector로 변환했다.
2. Normalizing  
   write\_date의 값과 연속적인 값을 갖는 또 다른 Feature들의 값 사이에 큰 차이가 존재한다. 이는 모델을 학습시킬 때 write\_date에 의도치 않은 가중치를 부여하는 것과 같은 효과를 야기할 수 있다. 따라서 정규화를 통해 모든 연속적인 값(room\_size, maintanence\_cost, write\_date)의 범위를 어느정도 통일시켜 이러한 문제를 해결했다. (Value - mean / std.)
3. Feature Selection  
   - **room\_size**: 기존 데이터에서 방의 크기를 나타내는 feature는 room\_size\_m2, room\_size로 두 개가 존재. 전자의 경우 평 단위 대신 m^2단위로 방의 크기를 나타낸 값이고, 후자의 경우 평 단위로 방의 크기를 나타낸 값이었다. 두 feature는 Correlation이 높은 feature이므로 둘 중 하나만 선택하기로 결정했다. 일반적으로 더 정확한 값인 m^2단위를 선택하여야겠지만 우리나라 부동산 시장의 특성상 부동산 거래시 평 단위를 중시하는 경향이 존재하므로 어느 단위가 더 중요한지 알 수 없기 때문에 validation을 통해 더 좋은 결과 값을 출력하는 쪽을 선택하기로 결정했다.  
     
   - **write\_date**: write\_date는 매물을 게시한 날짜를 표시하는 값인데, 부동산 시세는 일반적으로 등락을 반복하는 경향을 보이며, 시간과 반드시 정비례하지는 않는다. 그러나 장기적인 관점에서 보면 시간에 따라 상승하는 추세를 보이는 것은 사실이다. 따라서 위의 경우와 같이 validation을 통해 채택 여부를 결정했다.  
     
   Validation은 충분히 적절한 결과를 도출할 수 있는 Baseline Model을 선정하여 시행. 처음에는 Linear Regression을 사용해보았으나 이후 모델 평가에서도 언급하겠지만,   
    r2 score가 지나치게 좋지 않아 의미 없다고 판단되어 대체 모델로 Lasso Regression을 채택하여 시행했다. 결과적으로, 방의 크기를 나타내는 feature는 평 단위를 사용한 room\_size로 채택하였고, write\_date는 사용하기로 결정하였다.

## 5 - 3) 데이터 모델링

* Linear Regression

Linear Regression은 sklearn.linear\_model의 LinearRegression() 을 이용하였다. 특별히 설정해줘야 하는 parameter가 존재하지 않아서 validation 과정 없이 바로 학습시켰고, train dataset과 test dataset 각각 mean\_squeard\_error와 score를 함께 출력하도록 구현하였다.

* Ridge Regression

Ridge Regression은 sklearn.linear\_model의 Ridge()를 이용하였다. alpha값을 1e-06 ~ 1e+05까지 설정해 각각의 alpha마다 5-fold로 Cross Validation을 진행하였고, 이후 평균 score가 가장 잘 나왔던 alpha로 다시 학습을 시켰다. 이후 train dataset과 test dataset 각각 error를 분석해 score와 함께 출력하도록 구현하였다.

* Lasso Regression

Lasso Regression은 sklearn.linear\_model의 Lasso()를 이용해 구현하였다. Ridge Regression과 큰 틀은 거의 유사하지만 alpha값은 Ridge와 똑같이 설정해줬을 경우 alpha값이 너무 작다는 에러가 나왔기 때문에 1e-01 ~ 1e+05로 설정하였고 이 역시 5-fold로 Cross Validation을 진행 후, 가장 값이 잘 나왔던 alpha로 다시 학습을 시켜 error와 score를 함께 출력하였다.

* Logistic Regression

Logistic Regression은 sklearn.linear\_model의 LogisticRegression()을 이용하였다.   
Logistic Regression은 기본적으로 classification 문제에 적합한 모델이지만 API 설명에서 continuous 문제에도 적용시킬 수 있다고 되어 있기 때문에 구현하였다. 하지만 모델 학습 과정에서 하나의 parameter에 대한 결과가 10시간이 지나도 나오지 않았기 때문에 continuous 문제엔 최적화 되어 있지 않다고 생각하여 코드는 주석처리하고 스킵하였다.

* SVM

SVM은 sklearn.svm의 SVR()을 이용하였다. SVM의 각 kernel은 ‘rbf’, ‘poly’, ‘linear’, ‘sigmoid’로 총 4개를 학습에 사용하였고 polynomial은 degree=3으로 구현하였다. 모든 SVM은 3-fold로 Cross Validation을 진행하였고, 이후 각각의 score를 출력하도록 하였다.

* Neural Network

Neural Network는 sklearn.neural\_network의 MLPRegressor()를 이용해 구현하였다. 먼저 ‘identity’, ‘logistic’, ‘tanh’, ‘relu’의 4가지 activation function으로 5-fold로 Cross Validation을 진행해 가장 좋은 결과값을 출력하는 activation function을 구하였고, 이후 hidden layer node의 개수를 50 ~ 200으로 설정해 다시 5-fold로 Cross Validation을 진행하고, 가장 높은 score를 갖는 model을 찾을 수 있도록 출력했다.

## 5 - 4) 모델 평가

* Linear Regression

Train Score는 약 0.64로 준수한 반면에, Test 결과값이 굉장히 안 좋게 나왔다. 약 -85의 score를 출력하였는데, 이는 극단적인 outlier들에 의해 모델 가중치가 크게 변했기 때문으로 추정된다. 이후 Ridge, Lasso Regression에서 이 문제를 해결할 수 있었다.

* Ridge Regression

Train score는 약 0.638, Test score는 약 0.600의 값이 나왔다. Linear Regression의 극단적으로 잘못된 결과값에 비한다면 크게 나아진 모델이라고 평가할 수 있다. Linear Regression과 달리 penalty를 부과한 더 발전된 모델이기 때문에 더욱 원활한 학습이 가능해졌다고 볼 수 있겠다.

* Lasso Regression

Train score는 약 0.588, Test score는 약 0.497의 값이 나왔다. Lasso Regression은 Ridge Regression과 매우 유사하며, L2 regularization 대신 L1 regularization을 쓰는 차이만 존재하므로, Ridge Regression과는 유사하며 Linear Regression보다는 발전된 모델이라 할 수 있으며 결과값도 이를 비슷하게 증명한다.

* Logistic Regression

Logistic Regression은 코드를 돌리는 과정에서 12시간 이상 코드를 돌렸으나, 결과값이 나오지 않았다. Logistic Regression은 본래 Regression보다는 Classification에 중점을 두고 있는 모델이기 때문에 Logistic Regression을 결론적으로 사용하지 않기로 결정하였다.

* SVM

SVM은 총 4개의 SVM을 돌리는데 약 12시간 가량이 소모되었고, Train score값 역시 0.478, 0.452, 0.464, -6.895로 크게 좋은 결과를 출력하지 못하였다. SVM 역시 Logistic Regression과 비슷하게 분류에 더 최적화된 모델임을 감안하였을 때 낮은 Train score가 어느정도 예상되는 부분이었다.

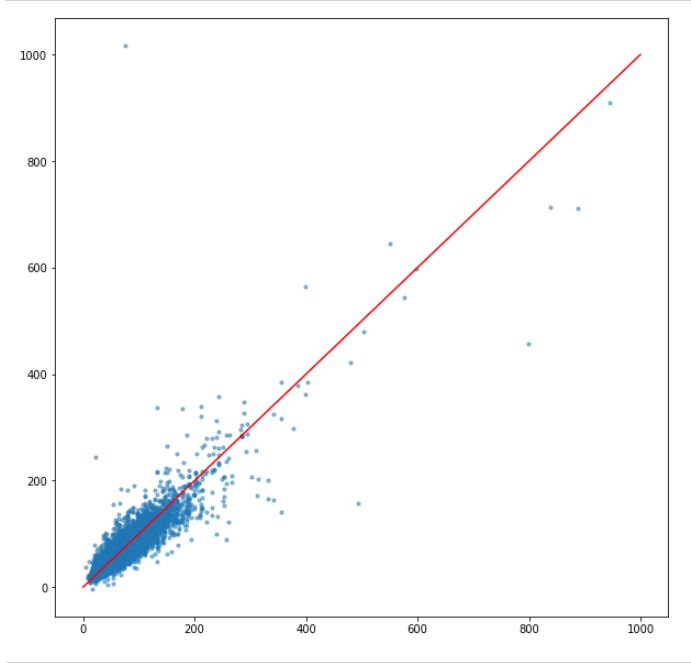
* Neural Network

먼저 activation function을 고르기 위해 Cross Validation을 시행한 결과, identity는 Train score 0.629, logistic은 Train score 0.738, tanh는 Train score 0.694, relu는 Train score 0.774로 relu activation function이 가장 좋은 결과를 출력하는 것으로 확인되었다. 이후 relu를 바탕으로 hidden layer를 50, 100, 150, 200으로 두고 각각 시행한 결과 hidden layer 50에서 0.784로 가장 좋은 결과를 출력하였다. 이는 앞서 설명한 모든 모델과 비교해도 가장 좋은 Train score를 뽑아내므로, 최종적으로 relu activation function을 이용한 hidden layer 50의 Neural Network를 Prediction에 이용하기로 하였다.

# 6. 결론

## 6 - 1) 최종 결과

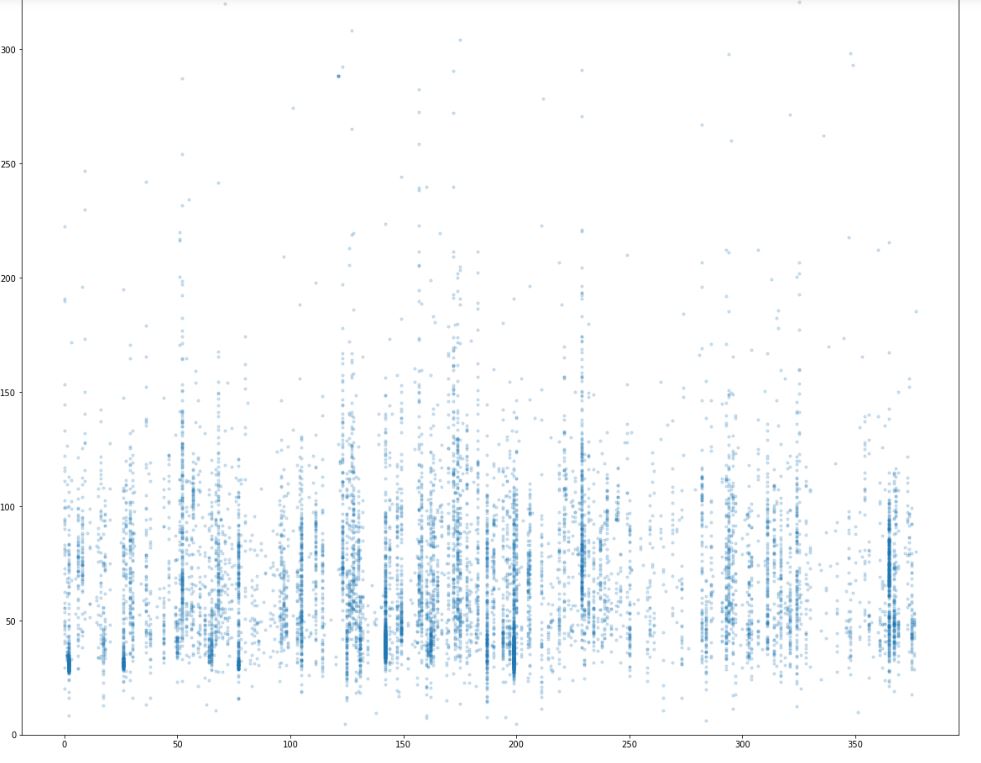
최종 모델: MLPRegressor(Neural Network)  
**with** activation function = “relu”, hidden layer size = 50  
최종 모델에 Test dataset을 넣어 평가를 한 결과, Test score = 0.777, Test RMSE = 19.963의 결과를 얻을 수 있었다.

(y\_test - y\_predicted graph)

위의 그래프를 보면 알 수 있듯이 매우 오차가 큰 몇몇 outlier가 존재하는 것을 확인할 수 있다.   
이러한 오류가 발생한 이유를 추측하자면, maintanence 개수나 각 option의 값을 imputation하는 과정에서 특정 data에 적절치 않은 수치가 적용되었거나, 판매자가 적절하지 않은 조건의 매물을 게시한 것으로 보인다.

## 6 - 2) 분석

아래 그래프는 test set에 대한 모델의 예측값을 지역별로 나타낸 것이다. x축이 지역을 나타내며, 동의 한글 명칭 순으로 나열되어있다.



즉, 위의 그래프는 각 동 별간의 시세 차이를 대략적으로 확인할 수 있는 그래프이다. 이를 분석해보면 대체로 모든 지역들이 비슷한 가격대를 보이는 것으로 보이지만, 최저 y가 50 이상인 지역이 있고, 유독 y=50 이하의 가격 예측이 많은 동네가 몇 군데 존재한다. 전자는 지역의 땅 값 자체가 비싼 동네이고, 후자는 땅 값이 싼 동네로 볼 수 있다. 우리 모델에서 후자로 예측한 동은 순서대로 용산동 3가, 묵정동, 오류동, 창신동이었는데 용산동 3가는 거주지역이 거의 없어 data set이 절대적으로 부족하여 오류가 발생한 것으로 추측 할 수있고, 이후로 각각 하위 28위, 42위, 31위(부동산 랭킹(부킹) buking.kr 사이트 기준, 212 개 동 중)로 상대적으로 하위권에 위치한 것을 확인할 수 있었다.

해당 기준은 매매 평당가 순으로, 방 값 순위와는 차이가 존재할 수 있고, 우리가 사용한 dataset과 구역 분할 기준이 달라 정확한 순위 비교는 어려운 것을 고려하면, 지역별 가격 편차를 대체로 표현할 수 있는 모델이라고 볼 수 있다.

또한 y=200이상으로 예측하는 데이터를 일부 확인할 수 있는데, 이는 데이터 전처리를 하는 단계에서 charge라는 y를 전월세전환율을 통해 새로 정의를 하는 과정에서 전세와 월세뿐만 아니라 매매에 해당하는 데이터 또한 전월세전환율로 계산을 했기 때문에 가격을 비싸게 예측한 것으로 보인다.

해당 모델을 통한 각 지역의 정확한 시세 분석은 만족스러운 결과가 아니었으나, 각 지역의 대략적인 상대적 시세 수준은 반영한다는 결과를 얻을 수 있었다. 추후에 해당 모델을 발전시켜 지역을 시세에 따라 상위/중위/하위 수준으로 분류하는 작업도 가능할 것으로 보인다.