Data Mining Challenge report

20200045 김건우

1.Load Base Dataset

```
1. Load Base Dataset

[3]  1 INPUT_DIR='/content/drive/MyDrive/ML IE/IE343-Kaggle-Project'
     2 OUTPUT_DIR = '/content/drive/MyDrive/ML IE/IE343-Kaggle-Project'

[4]  1 ## Do not change this cell.
     2 ## The reason for this is to prevent cheating using the original data from DACON.
     3 ## If an assertion error occurs in that cell when TAs evaluate the submitted code of each student, it is considered cheating.
     4
     5 train = pd.read_csv(os.path.join(INPUT_DIR, 'train.csv'))
     6 test = pd.read_csv(os.path.join(INPUT_DIR, 'test.csv'))
     7 park = pd.read_csv(os.path.join(INPUT_DIR, 'park.csv'))
     8 dcc = pd.read_csv(os.path.join(INPUT_DIR, 'day_care_center.csv'))
     9
    10 assert train.shape[0] == 329690 and train.shape[1] == 13, 'Do not change the format of the input data.'
    11 assert test.shape[0] == 85097 and test.shape[1] == 12, 'Do not change the format of the input data.'
    12 assert park.shape[0] == 1359 and park.shape[1] == 7, 'Do not change the format of the input data.'
    13 assert dcc.shape[0] == 7373 and dcc.shape[1] == 10, 'Do not change the format of the input data.'
```

2. Load additional dataset

추가적인 데이터셋은 사용하지 않았다.

3. EDA and preprocessing

데이터셋을 살펴보면 apartment_id(아파트id), city(도시), dong(동), house_area(면적), built_year(건설년도), floor(층), lat(위도), long(경도), transaction_year(거래년도), transaction_month(거래월), transaction_day(거래일), PRICE(가격) 으로 이루어져 있다. 우리의 최종 목표는 가격을 예측하는 것이므로 PRICE 열을 y로, 나머지를 x로 한다.

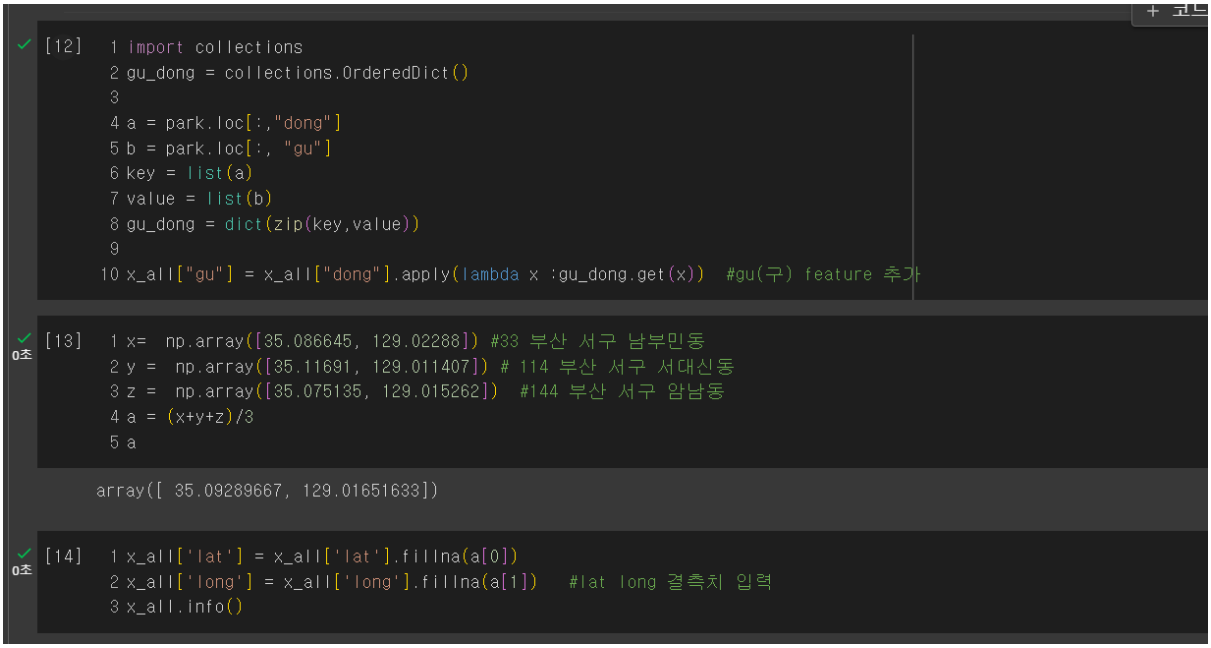
```
1 train= train.set_index(keys=['index'], inplace=False, drop=True)
2 test = test.set_index(keys=['index'], inplace=False, drop=True)

+ 코드  + 텍스트

[6]  1 y_train = np.log(train['PRICE'])
     2 x_train = train.drop('PRICE', axis =1)
     3 x_all = pd.concat([x_train,test], axis =0)
     4 x_all
```

index	apartment_id	city	dong	house_area	built_year	floor	lat	long	transaction_year	transaction_month	transaction_day
0	0	busan	197	125.865988	1993	5	35.149929	129.006071	2021	7	11~20
1	0	busan	197	101.647190	1993	12	35.149929	129.006071	2021	10	1~10
2	0	busan	197	91.511175	1993	6	35.149929	129.006071	2020	3	21~31
3	0	busan	197	101.647190	1993	13	35.149929	129.006071	2020	5	11~20
4	0	busan	197	101.647190	1993	4	35.149929	129.006071	2022	6	21~30

*기존 feature

(a) 가장 먼저 각 열의 결측치를 확인해보자. lat, long에 결측치가 존재하고 나머지 값들에 대해서는 존재하지 않는다. 위치정보는 집값에 있어서 매우 중요한 요소라고 할 수 있다. 따라서 위도, 경도의 결측치를 채워준 후 분석에 활용하고자 한다. 결측치를 채우는 방법에는 0으로 채우기, 평균값으로 채우기, 최빈값으로 채우기 등등이 있으나 본 데이터를 가장 잘 활용하기 위해 다른 위

치정보 (도시, 동, 구)를 활용하여 가장 가까운 아파트들의 위/경도 값의 평균으로 결측치를 채워주었다. 기존 데이터셋에는 구에 대한 데이터가 없다. 따라서 park 데이터셋의 정보를 이용하여 구에 대한 데이터를 만들어주었고, 이를 토대로 결측치가 있는 아파트의 위치가 부산 서구임을 추론하였다. 부산 서구 위치에 해당하는 아파트3개의 위치의 평균값으로 결측치를 채워주었다.



```
[12]  1 import collections
      2 gu_dong = collections.OrderedDict()
      3
      4 a = park.loc[:,"dong"]
      5 b = park.loc[:, "gu"]
      6 key = list(a)
      7 value = list(b)
      8 gu_dong = dict(zip(key,value))
      9
     10 x_all["gu"] = x_all["dong"].apply(lambda x :gu_dong.get(x))  #gu(구) feature 추가
```

```
[13]  1 x=  np.array([35.086645, 129.02288]) #33 부산 서구 남부민동
      2 y =  np.array([35.11691, 129.011407]) # 114 부산 서구 서대신동
      3 z =  np.array([35.075135, 129.015262])  #144 부산 서구 암남동
      4 a = (x+y+z)/3
      5 a
```

```
array([ 35.09289667, 129.01651633])
```

```
[14]  1 x_all['lat'] = x_all['lat'].fillna(a[0])
      2 x_all['long'] = x_all['long'].fillna(a[1])   #lat long 결측치 입력
      3 x_all.info()
```

(b) city(도시)는 서울 부산 2가지 종류가 있다. 서울은 1, 부산은 0으로 값을 변환해주었다.

(c) built_year(건설년도)와 floor(층) 열은 각각 1962~2016까지, -4~70까지의 값을 가지고 있다. 각각 -1961, +5를 하여 최솟값이 1이 되게 preprocessing해주었다.

(d) transaction_year(거래년도), transaction_month(거래월), transaction_day(거래일)은 시간 흐름에 따라 하나의 값으로 라벨링 할 수 있다. 1년은 12달, 1달은 3개의 일(1~10, 11~20, 21~)로 이뤄지고 거래년도는 2018년도부터 이므로 (년도-2018) * 36 + (달-1)*3 + 일 이란 식으로 거래일을 시간에 따라 하나의 정수값으로 나타내었다.

(e) house_area(면적)의 경우 log를 취해 정규분포에 가까운 분포를 가지게끔 변환하여 분석에 사용하였다.

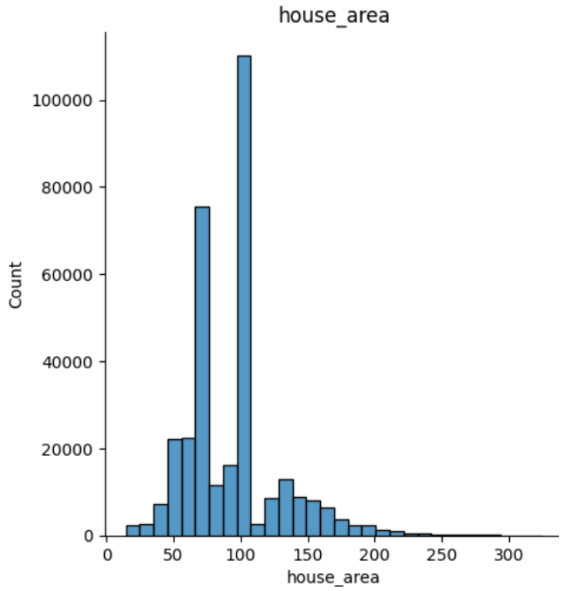


Figure1. 기존 데이터(house_area) 분포

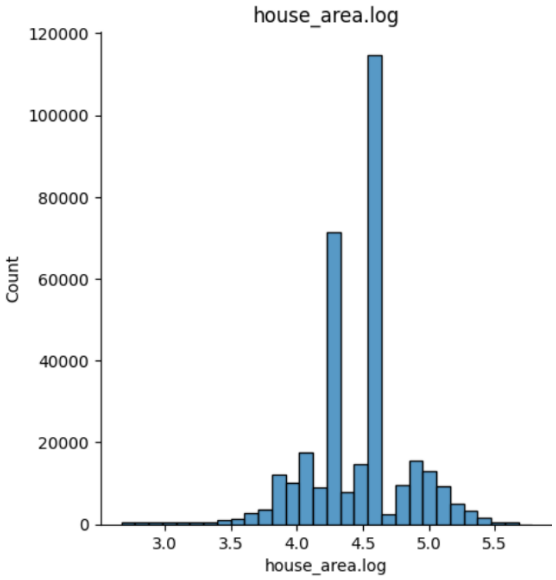


Figure2. Log를 취한 house_area 분포

```python
day_num_is_2 = x_all['transaction_day'] == '11~20'
day_num_is_1 = x_all['transaction_day'] == '1~10'
x_all.loc[day_num_is_2,'day_num'] = 2
x_all.loc[day_num_is_1,'day_num'] = 1
x_all['day_num'].replace(np.nan, 3, True)

x_all['transaction_year'] = x_all['transaction_year'].astype(int)
x_all['transaction_month'] = x_all['transaction_month'].astype(int)
x_all['day_num'] = x_all['day_num'].astype(int)

x_all['time_order'] = (x_all['transaction_year']-2018)*36 + (x_all['transaction_month']-1)*3 + x_all['day_num']  # 1,2,...

#########################
x_all['floor_prepro'] = x_all['floor'] + 5
x_all['built_year_prepro'] = x_all['built_year'] - 1961
x_all = x_all.replace({'city' : {'seoul':1, 'busan':0}})

##############################3
x_all['log_house_area'] = np.log(x_all['house_area'])
x_all
```

(f) PRICE의 경우에도 그래프의 분포를 살펴보면 상당히 치우쳐 있음을 알 수 있다. Log를 취해 분석에 사용한 후, 나중에 predict한 값에 exp함수를 사용하여 값을 추측하였다.

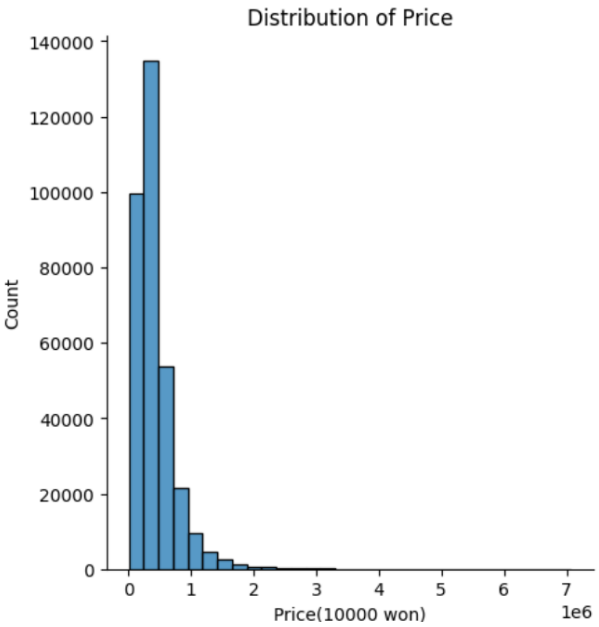


Figure3. 기존 데이터(PRICE) 분포

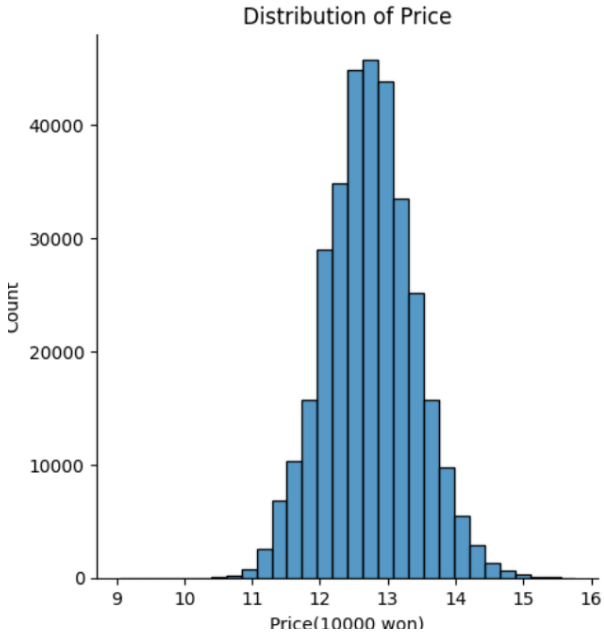


Figure4. Log를 취한 PRICE 분포

* 추가한 feature

(a) gu(구) 열 추가. 주어진 park 데이터셋에는 위치 정보 중 도시, 동 뿐만 아니라 구에 대한 정보도 나와있다. Park 데이터셋을 이용해 동을 key, 구를 value로 갖는 딕셔너리를 만들어 각 동이 어느 구에 포함되는지 매핑해주었다.

(b) num_dcc, num_park. 주어진 day_care_center데이터셋을 통해 각 구에 몇 개의 센터가 있는지를 센 후, 그 값을 num_dcc란 feature에 대입하였다. 예를들어 1번 구에 5개의 센터가 있다면 1번 구에 해당하는 모든 apt들의 num_dcc 값은 5인 셈이다. num_park도 같은 방식이나 park는 동까지의 정보가 나와있으므로 각 동에 몇 개의 공원이 있는지로 계산해주었다.

```
[15]  1 def get_occurrence_count(my_list):
      2   new_list = {}
      3   for i in my_list:
      4     try: new_list[i] += 1
      5     except: new_list[i] = 1
      6   return(new_list)

[18]  1 a = list(dcc["gu"])
      2 b = get_occurrence_count(a)
      3
      4 c = list(park["dong"])
      5 d = get_occurrence_count(c)

[20]  1 x_all["num_dcc"] = x_all["gu"].apply(lambda x:b.get(x))  #num_dcc, num_park feature 추가
      2 x_all["num_park"] = x_all["dong"].apply(lambda x:d.get(x))
```

(c) coord_pca1, coord_pca2. 위도와 경도 데이터를 pca를 통해 다시 표현한 feature이다. 추후에 클러스터링에도 사용하였다.

```
from sklearn.decomposition import PCA

coord = x_all[['lat','long']]
pca = PCA(n_components=2)
pca.fit(coord)

coord_pca = pca.transform(coord)

x_all['coord_pca1'] = coord_pca[:, 0]  #coord_pca1, coord_pca2 feature 추가
x_all['coord_pca2'] = coord_pca[:, 1]
```

(d) rand_dis. 랜드마크까지의 거리를 나타낸 feature이다. 각 지역의 랜드마크와 가까울수록 집값이 높을 것이라고 추측하고 생성하였다. 여기서 랜드마크는 실제 랜드마크를 사용한 것은 아니고 각 지역구에서 가장 평균집값이 높은 apartment_id의 위치를 랜드마크로 지정하였다. 거리는 km 단위로 계산하였다(Haversine Distance 공식 이용).

```
[23]
x_randmark =x_all[['apartment_id','city','dong','gu','lat','long']]
x_randmark =x_randmark.iloc[:train.shape[0],:]
x_randmark['price'] =  train['PRICE']
x_randmark.head()
```

index	apartment_id	city	dong	gu	lat	long	price
0	0	0	197	18	35.149929	129.006071	229250.8
1	0	0	197	18	35.149929	129.006071	215320.0
2	0	0	197	18	35.149929	129.006071	161740.0
3	0	0	197	18	35.149929	129.006071	199781.8
4	0	0	197	18	35.149929	129.006071	219606.4

```
[26]
x_randmark1 = x_randmark.groupby('apartment_id').mean()
#df.groupby('city').mean(price)
#x_randmark2 = x_randmark1.groupby('gu')

rand_lat_dic = {}
rand_long_dic = {}
for i in range(36): #36개 지역구
    if i!=13 and i!=30:  #13,30번 구는 없음
        df = x_randmark1[x_randmark1["gu"]==i]
        k = np.argmax(df['price'])
        max_lat = float(df.iloc[[k] , :]['lat'])
        max_long = float(df.iloc[[k] , :]['long'])
        rand_lat_dic[i] = max_lat  #각 지역구 별 랜드마크 위도
        rand_long_dic[i] = max_long #각 지역구 별 랜드마크 경도
```

```
[28]
x_all['rand_lat'] =  x_all["gu"].apply(lambda x:rand_lat_dic.get(x))  #'rand_lat', 'rand_long' feature 추가
x_all['rand_long'] =  x_all["gu"].apply(lambda x:rand_long_dic.get(x))
```

```
x_all['rand_lat_diff'] = x_all['lat'] - x_all['rand_lat']
x_all['rand_long_diff'] = x_all['long'] - x_all['rand_long']
```

```
[31]
AVG_EARTH_RADIUS = 6371
x_all['rand_dis'] =2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(np.sin(x_all['rand_lat_diff'] * 0.5) ** 2 + np.cos(x_all['lat']) * np.cos(x_all['rand_lat']) * np.sin(x_all['rand_long_diff'] * 0.5) ** 2 )) #'rand_dis' feature 추가
```

(e) 'coord_cluster'. 각 아파트의 위치를 그대로 클러스터링한다면 동, 구, 도시의 순서대로 클러스터링이 될 것이라고 예측했다. 따라서 위도 경도가 아닌 추가적으로 위에서 구한 coord_pca1, coord_pca2를 기준으로 클러스터링하여 새로운 지역구를 나눈다는 느낌으로 'coord_cluster' feature를 추가해주었다.

```
coord = x_all[['coord_pca1','coord_pca2']]

RANDOM_SEED = 631
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=100, random_state=RANDOM_SEED).fit(coord)
coord_cluster = kmeans.predict(coord)
x_all['coord_cluster'] = coord_cluster
```

4. modeling

데이터의 양이 상당히 많고, feature의 수도 많은 편이라고 생각해 분석 모델로는 LightGBM을 채택하였다. 최적의 파라미터는 GridSearchCV와 optuna를 사용하려 했으나 시간이 너무 많이 소요되어 처음에 optuna를 1번 사용해 파라미터의 범위를 대충 예상한 후 그 이후에는 하나씩 값을 바꿔가며 parameter를 튜닝해주었다. 'apartment_id','city','dong','gu','coord_cluster'는 categorical feature로 분류하여 분석에 사용하였다.

5. make submission

```
best_model = model
Y_exam = best_model.predict(X_exam)
Y_exam = np.exp(Y_exam)

submisson = pd.read_csv(os.path.join(INPUT_DIR, 'sample_submission.csv'))

submisson['PRICE'] = Y_exam

submisson.to_csv(os.path.join(OUTPUT_DIR,'sample_submission2358.csv'),sep = ',', na_rep= 'Nan', index = False)
```