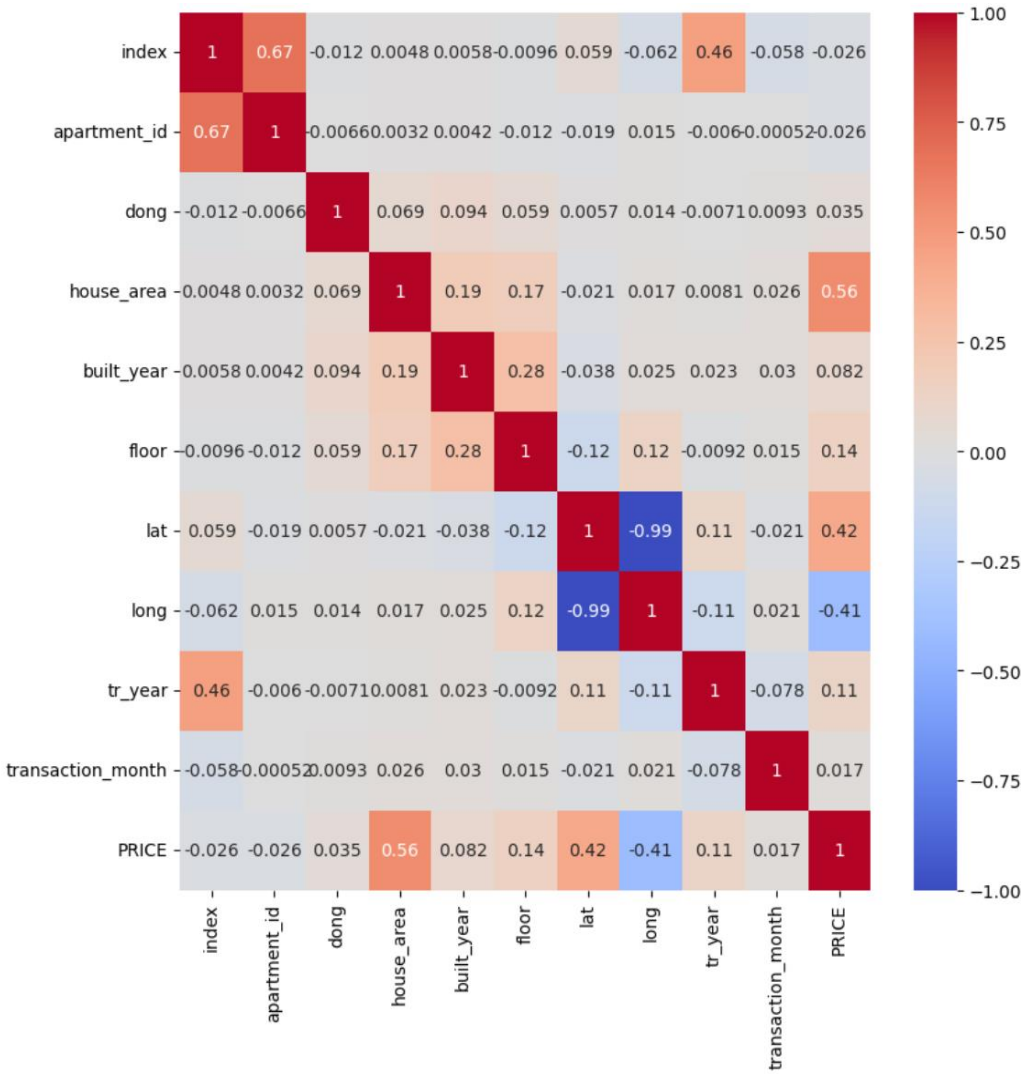
서울&부산 아파트 실거래가 예측 모델 보고서

20220487 이선규

I. EDA and Preprocessing



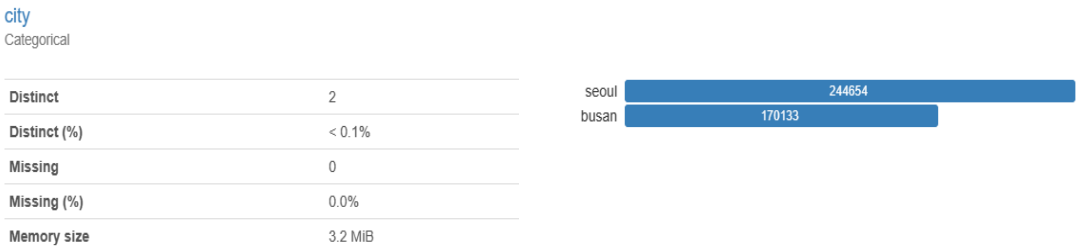
<그림 1> train feature간의 상관계수 행렬

가장 먼저, train.csv와 test.csv 데이터의 preprocessing를 간편하게 진행하기 위해, 두 데이터를 합친, 'all' 데이터프레임을 관리하도록 했다. 이는 train.csv와 test.csv가 'PRICE' 변수를 제외한 모든 feature가 동일하기 때문에 가능하다.

```
RangeIndex: 414787 entries, 0 to 414786
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   index              414787 non-null  int64
 1   apartment_id       414787 non-null  int64
 2   city               414787 non-null  object
 3   dong               414787 non-null  int64
 4   house_area         414787 non-null  float64
 5   built_year         414787 non-null  int64
 6   floor              414787 non-null  int64
 7   lat                414685 non-null  float64
 8   long               414685 non-null  float64
 9   tr_year            414787 non-null  int64
 10  transaction_month  414787 non-null  int64
 11  transaction_day    414787 non-null  object
 12  PRICE              329690 non-null  float64
dtypes: float64(4), int64(7), object(2)
memory usage: 41.1+ MB
```

데이터프레임 'all'은 총 13개의 변수를 가지고 있는 데이터 파일이다. 이 중, 'test.csv'에는 없는 'PRICE'변수는 제외하고, 각각의 변수 별로 다음과 같은 EDA & Preprocessing 작업을 해주었다.

1) city

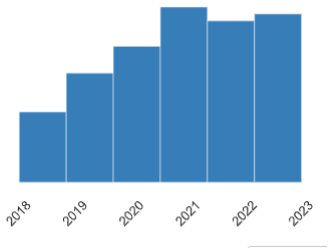


'city' 변수는 두 개의 값 중 하나를 가지는데, 각각 'seoul'과 'busan'이다. 이 변수의 타입은 'object'이므로 모델 학습의 편의를 위해 임의로 'seoul'에 1을, 'busan'에 0을 배정하였다.

2) transaction_year + transaction_month

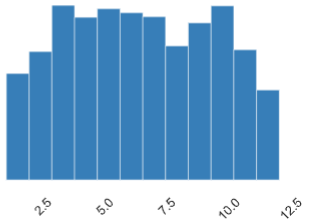
tr_year
Real number (ℝ)

Distinct	6	Minimum	2018
Distinct (%)	< 0.1%	Maximum	2023
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	2020.9183	Memory size	3.2 MiB



transaction_month
Real number (ℝ)

Distinct	12	Minimum	1
Distinct (%)	< 0.1%	Maximum	12
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	6.4115582	Memory size	3.2 MiB

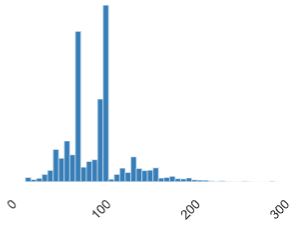


'transaction_year'은 2018~2021 중 하나의 자연수를 가지고, 'transaction_month'은 1 ~ 12 중 하나의 자연수를 가진다. 편의를 위해 "(transaction_year - 2018) * 12 + transaction_month" 식으로 변수를 추가하여, 2018년 1월부터 몇 개월 이후인지를 나타내는 변수를 추가했다.

3) house_area

house_area
Real number (ℝ)

Distinct	9240	Minimum	14.39972
Distinct (%)	2.2%	Maximum	325.10596
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	93.874969	Memory size	3.2 MiB



<그림1>을 살펴보면, 'PRICE'변수와 가장 밀접한 관련이 있는 변수가 'house_area' 라는 것을 확인할 수 있다. 면적을 나타내는 'house_area' 변수는 총 9,240개의 서로 다른 값을 가지고 있는 변수이다. 이를 그룹화하면 모델 학습과 EDA에 도움이 될 것 같아, <코드 1>을 통해 새로운 변수, 'pyung'을 만들었다. 10의 배수 단위로 그룹화했다.

```
all['pyung'] = all['house_area'] // 10 * 10          # 10제곱미터 단위
```

<코드 1> 'pyung' 변수를 추가하는 코드

또한, 극 소수의 10보다 작은 면적이나 300 이상의 값에 대해서는 각각 10, 300으로 값을 제한하도록 코드를 추가했다. 이에 대한 코드는 <코드2>와 같다.

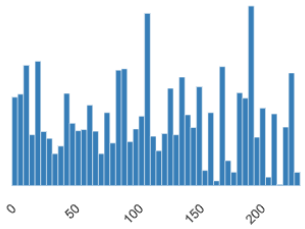
```
all.loc[all['pyung']<10, 'pyung'] = 10.0
all.loc[all['pyung']>=300, 'pyung'] = 300.0
```

<코드 2> 'pyung' 변수의 최소, 최대 값을 조절

4) dong

dong
Real number (ℝ)

Distinct	203	Minimum	0
Distinct (%)	< 0.1%	Maximum	236
Missing	0	Zeros	4561
Missing (%)	0.0%	Zeros (%)	1.1%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	113.08621	Memory size	3.2 MiB



<코드 3>과 <코드 4>를 통해 'all'의 'dong' 변수에 대해 추가적인 분석을 진행했다.

train, test data의 동 개수

```
train_dong = set(train['dong'])
test_dong = set(test['dong'])

print('*** 동 개수 ***')
print(f'train data: {len(train_dong)}')
print(f'test data : {len(test_dong)}')
print(f'test data only: {len(test_dong - train_dong)}')
```

```
*** 동 개수 ***
train data: 203
test data : 202
test data only: 0
```

<코드 3> train과 test데이터의 'dong' 종류 카운트

서울과 부산의 같은 동 이름 구분

```
cnt = len(all['dong'].unique())
print(f'Before: {cnt}')

seoul_dong = set(all.loc[all['city']==1, 'dong'])
busan_dong = set(all.loc[all['city']==0, 'dong'])
same_dong = seoul_dong & busan_dong
print(f'Same dong: {same_dong}')

for d in same_dong:
    all.loc[(all['city']==1) & (all['dong']==d), 'dong'] = 1000 + d
    all.loc[(all['city']==0) & (all['dong']==d), 'dong'] = 2000 + d

cnt = len(all['dong'].unique())
print(f'After: {cnt}')
```

```
Before: 203
Same dong: {120}
After: 204
```

<코드 4> 서울과 부산에 같은 'dong'이 존재하는지 확인

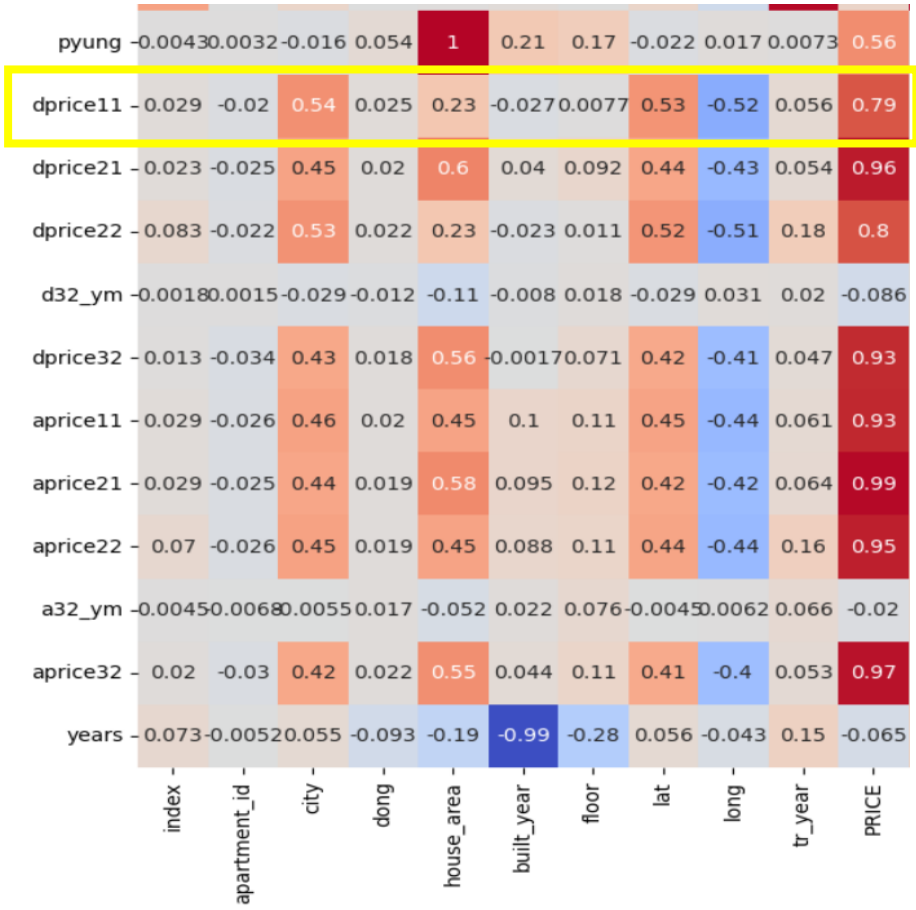
두 코드의 실행결과를 통해 알 수 있듯이, test.csv에만 등장하는 'dong'변수 값은 없으며, 부산과 서울에 120번 'dong'이 중복되게 존재한다는 것을 확인했다. 120번이 중복되는 것은 모델 학습에서 혼동을 줄 수 있으므로, 이 'dong'값을 각각 서울과 부산에 대해 각각 1120, 2120으로 임의 배정했다.

변수 'dong'(동)'을 같은 것끼리 묶어서, '같은 동에 속하는 아파트들의 실거래가의 평균'을 구한다면, 'PRICE' 변수와 밀접한 관련이 있을 것이라는 추론을 하게 되었다.

```
dong_df = trn.groupby('dong')['PRICE'].agg('mean').reset_index()
dong_df = dong_df.rename({'PRICE':'dprice11'}, axis=1)
all = pd.merge(all, dong_df, on='dong', how='left')
```

<코드 5> groupby를 사용하여 '같은 동의 집 값 평균'을 새로운 변수로 도출

<코드 5>을 통해 'dprice11'라는 변수를 새로 추가하였고, 이 변수는 'PRICE' 변수와 상관계수 0.79로 매우 높은 관련성을 지닌다는 것을 확인할 수 있었다. (그림 2)



<그림 2> 새로 추가한 변수와 'PRICE'간의 상관계수 행렬

'dong'과 관련된 변수를 몇 개 더 제작하여 'dprice21', 'dprice22', 'dprice32' 라는 이름을 붙였다. 각각의 변수의 정의는 <표 1>과 같다.

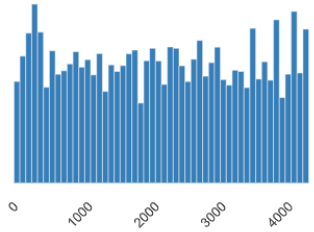
dprice21	같은 동, 같은 'pyung'을 가진 'PRICE'의 평균
dprice22	같은 동, 같은 'tr_year'을 가진 'PRICE'의 평균
dprice32	같은 동, 같은 'pyung', 가장 큰(나중) 'tr_ym'을 가진 'PRICE'의 평균

<표 1> 'dong'에서 파생된 변수의 정의

5) 'apartment_id'

apartment_id
Real number (ℝ)

Distinct	4420	**Minimum**	0
Distinct (%)	1.1%	**Maximum**	4419
Missing	0	**Zeros**	209
Missing (%)	0.0%	**Zeros (%)**	0.1%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	2201.1971	**Memory size**	3.2 MiB



'apartment_id' 변수에 대해서도 'dong' 변수에서 진행한 것처럼, <코드 6>을 통해 추가적인 분석을 진행했다.

train, test data의 apt 개수

```
train_apt = set(train['apartment_id'])
test_apt = set(test['apartment_id'])

print('*** Apt 개수 ***')
print(f'train data: {len(train_apt)}')
print(f'test data : {len(test_apt)}')
print(f'test data only: {len(test_apt - train_apt)}')
```

```
*** Apt 개수 ***
train data: 4419
test data : 3957
test data only: 1
```

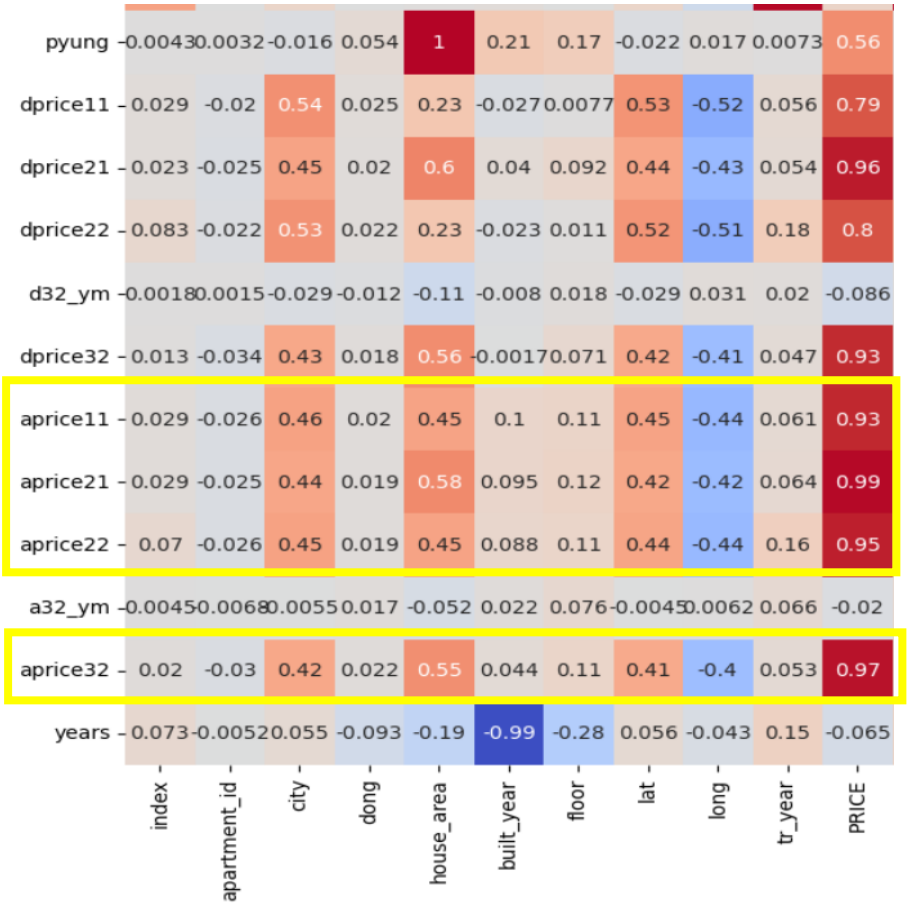
```
seoul_apt = set(all.loc[all['city']==1, 'apartment_id'])
busan_apt = set(all.loc[all['city']==0, 'apartment_id'])

print('*** Apt 개수 ***')
print(f'서울: {len(seoul_apt)}')
print(f'부산: {len(busan_apt)}')
print(f'공통: {len(seoul_apt & busan_apt)}')
```

```
*** Apt 개수 ***
서울: 2526
부산: 1894
공통: 0
```

<코드 6> train과 test데이터의 'apartment_id' 종류 카운트 (왼) 서울과 부산의 'apartment_id' 종류 카운트 (오)

test.csv에만 존재하는 'apartment_id'는 오직 하나라는 것을 확인할 수 있다. 서울과 부산에 공통적으로 있는 'apartment_id'는 없으므로, 이에 대해서는 추가적인 작업을 진행할 필요가 없다.

'dong'에서 변수를 추가한 것과 비슷한 아이디어를 적용하여, 'apartment_id' 또한 같은 아파트 단지의 평균을 이용해서 'aprice11', 'aprice21', 'aprice22', 'aprice32' 변수를 추가하였다. <그림 3>에서 확인할 수 있듯이, 이 변수들 또한 'PRICE' 변수와 상관계수 0.93~0.97로 매우 높은 관련성을 지닌다는 것을 확인할 수 있었다.



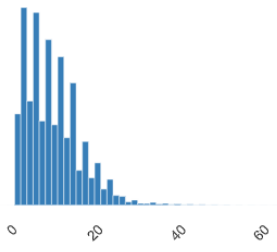
<그림 3> 'aprice' 변수들과 'PRICE' 간의 상관계수

6) floor

floor

Real number (ℝ)

Distinct	74	**Minimum**	-4
Distinct (%)	< 0.1%	**Maximum**	70
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	24
Infinite (%)	0.0%	**Negative (%)**	< 0.1%
Mean	9.6490295	**Memory size**	3.2 MiB



'floor' 변수는 총 24개의 데이터에 대해 음수 값을 가지고 있어 전처리 작업을 진행했다.

```
all[all['floor']<0].groupby('apartment_id')['df_index'].count().sort_values(ascending=False)
```

```
apartment_id
276     16
1526     5
2123     3
Name: df_index, dtype: int64
```

```
all[(all['PRICE'].isna()) & (all['floor']<0)]
```

	df_index	apartment_id	city	dong	house_area	built_year	floor	lat	long	tr_year
335106	335106	276	1	202	160.836735	2010	-1	37.642117	126.935430	2023
335121	335121	276	1	202	160.872615	2010	-3	37.642117	126.935430	2023
359502	359502	1526	1	205	101.324273	2014	-1	37.480951	126.838955	2023

3 rows × 23 columns

<코드 7> 'floor'이 음수인 행에 대한 분석

<코드 7>에서 확인해 본 결과, 음수 층은 3개 아파트에 대해 이루어졌고, 테스트 데이터의 음수 층은 2개 아파트의 3건이며, -1층과 -3층이다. 양수, 음수에 상관없이 넓이에 비례해 'PRICE'가 증가함을 확인할 수 있었다. 보통 아파트에 음수 층이 존재하지 않기 때문에, 양수 층의 잘못된 표기 또는 다른 이유로 음수 층으로 표기한 것일 뿐, 실제로는 양수 층 일 것이라고 판단해서 절대값을 씌워 양수 층으로 변환하였다.

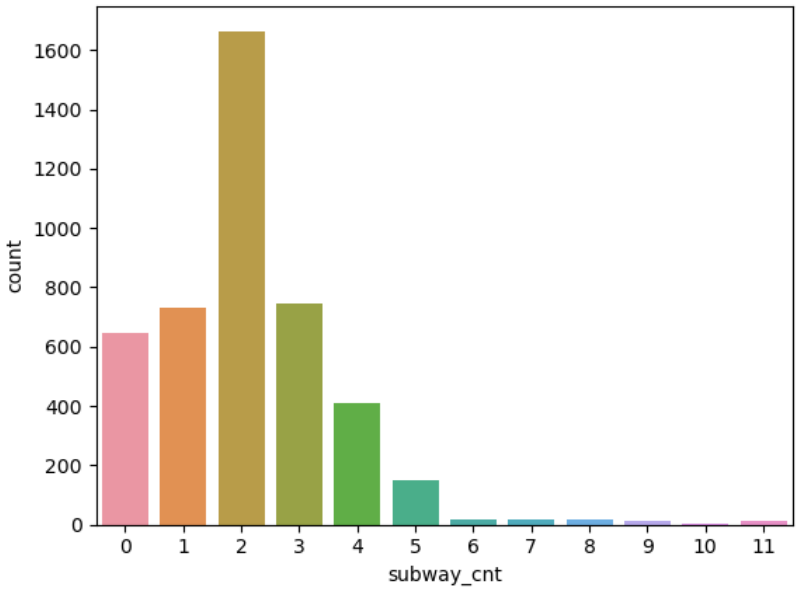
7) subway_seoul_busan.csv의 'lat', 'long'

아파트 실거래가에는 '교통이 얼마나 편리한지'가 큰 영향을 줄 것 같다는 생각을 했다. train.csv와 test.csv에서 제공하는 위도(lat)와 경도(long) 변수를 이용하기 위해서, 지하철 노선의 위도와 경도 데이터를 구하면 도움이 될 것이라고 생각했다. 공공 데이터 포털 data.go.kr에서 제공하는, '전국 도시 철도역사 정보 표준데이터'에서 대한민국의 모든 지하철 노선의 위치에 대한 정보를 가져올 수 있었다. 위도와 경도를 정리한 csv파일, subway_seoul_busan.csv의 첫 5행은 <그림4>와 같다.

	lat	long
0	37.477090	126.963506
1	37.487027	127.059475
2	37.481285	126.952695
3	37.610044	126.930302
4	37.796204	126.792563

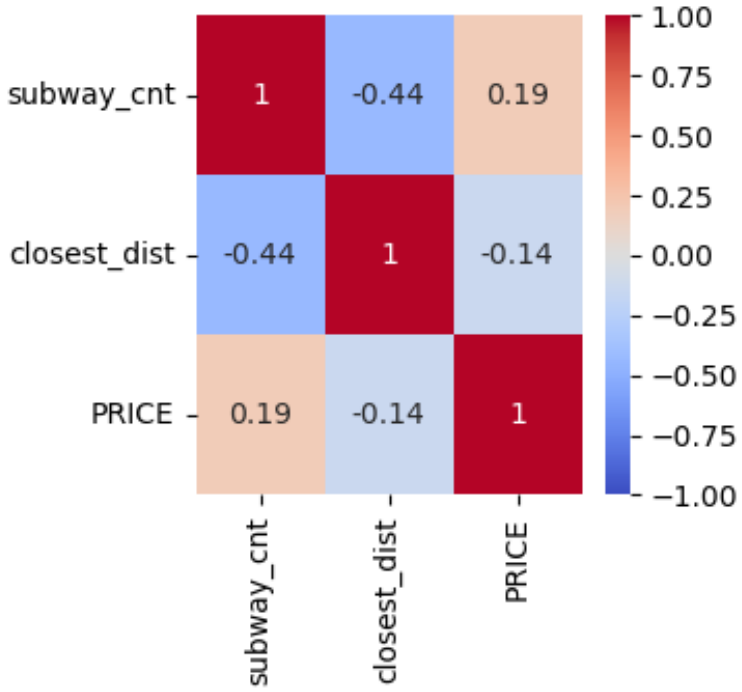
<그림4> subway_seoul_busan.csv의 첫 5행

교통이 좋다/나쁘다의 판단 기준은 '아파트로부터 일정 거리 이내에 지하철 노선이 몇 개가 있는가?'로 세웠다. 이것을 변수로 추가하기 위해, 모든 아파트, 지하철 노선 사이의 거리를 유클리디안 거리를 이용해서 구했다. 그 거리가 0.01 이내인 것의 개수를 각각의 아파트마다 구하고,이 값을 'subway_cnt' 라는 변수로 정의했다. 'subway_cnt'의 분포는 <그림 5>와 같다.



<그림 5> 'subway_cnt'의 분포

또한 'closest_dist' 변수를 정의하여, 각각의 아파트에 대해 '가장 가까운 지하철 노선과의 거리'로 정의했다. 추가한 두 변수와 'PRICE' 변수간의 상관계수는 <그림 6>과 같다.



<그림6> 추가한 'subway_cnt', 'closest_dist'와 'PRICE' 변수간의 상관계수

8) park.csv

```
RangeIndex: 1359 entries, 0 to 1358
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   city            1359 non-null   object
 1   gu              1359 non-null   int64
 2   dong            1359 non-null   int64
 3   park_name       1359 non-null   object
 4   park_type       1359 non-null   object
 5   park_area       1359 non-null   float64
 6   park_open_year  937 non-null    float64
dtypes: float64(2), int64(2), object(3)
memory usage: 74.4+ KB
```

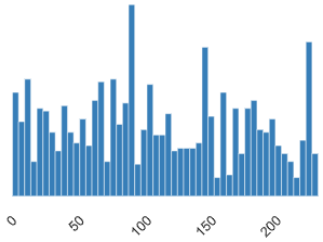
<그림 7> park.csv의 정보

문제에서 함께 제공되었던 park.csv에 대한 분석도 진행했다. park.csv는 7개의 변수를 가진 데이터이고, 'park_open_year'을 제외한 변수는 결측치가 없는 것을 확인할 수 있다.

park.csv에도 공원이 속한 동ID를 나타내는 'dong'에 대한 정보가 있어, 이 변수에 대한 EDA와 preprocessing을 진행했다.

dong
Real number (ℝ)

Distinct	237	**Minimum**	0
Distinct (%)	17.4%	**Maximum**	236
Missing	0	**Zeros**	13
Missing (%)	0.0%	**Zeros (%)**	1.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	112.14937	**Memory size**	10.7 KiB



먼저, 서울과 부산에서 겹치는 'dong'이 있는지 확인하고, 이들에 대해서는 각각 1000과 2000을 더해 서로 다른 'dong'이 되도록 임의 배정했다. (코드 8)

```
cnt = len(park['dong'].unique())
print(f'Before: {cnt}')

seoul_dong = set(park.loc[park['city']=='seoul', 'dong'])
busan_dong = set(park.loc[park['city']=='busan', 'dong'])
same_dong = seoul_dong & busan_dong
print(f'Same dong: {same_dong}')

for d in same_dong:
    park.loc[(park['city']=='seoul') & (park['dong']==d), 'dong'] = 1000 + d
    park.loc[(park['city']=='busan') & (park['dong']==d), 'dong'] = 2000 + d

cnt = len(park['dong'].unique())
print(f'After: {cnt}')
```

```
Before: 237
Same dong: {120, 208}
After: 239
```

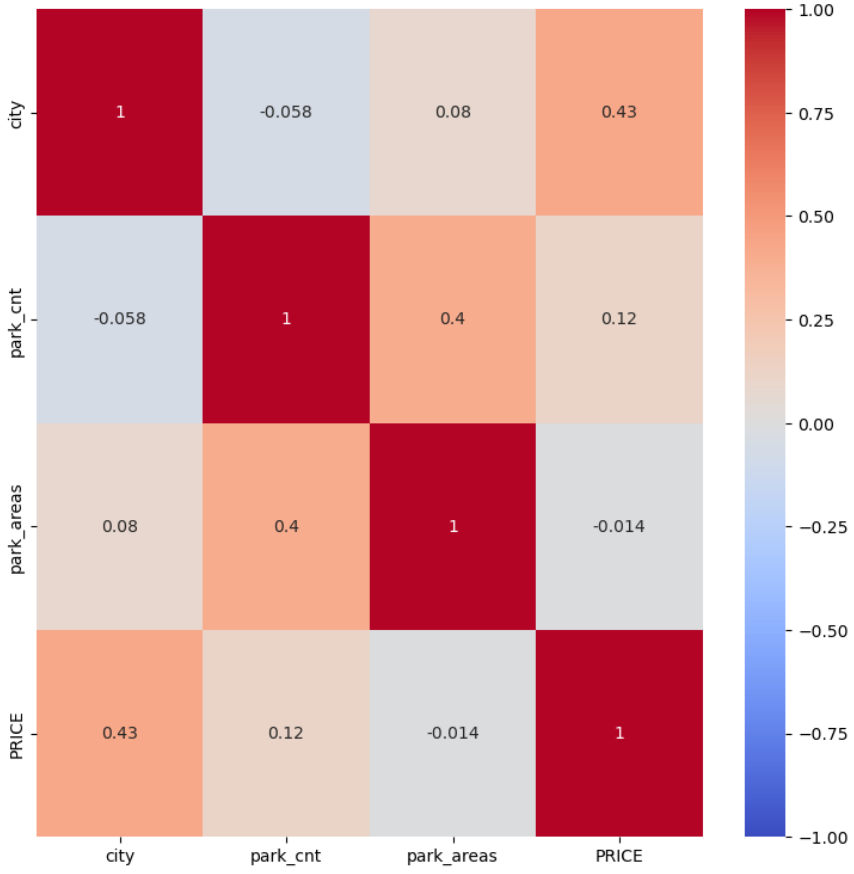
<코드 8> 서울과 부산에 공통되는 'dong' 제거

'dong'변수를 이용하여 두 개의 변수를 추가했다. 어떤 'dong'에 대해, 그 'dong'에 공원이 몇 개 있는지를 나타내는 'park_cnt' 변수와, 그 'dong'에 속해있는 공원의 평균 면적을 나타내는 'park_areas' 변수이다. 이 변수들을 기존의 train.csv와 test.csv파일을 합쳐 전처리한 all 데이터프레임에 추가하기 위해, park.csv 에서 같은 'dong'에 속하는 행의 정보를 merge를 통해 가져왔다. 자세한 코드는 <코드 9>과 같다.

```
park_df = park.groupby(['dong'])['park_name'].count().reset_index()
park_df = park_df.rename({'park_name':'park_cnt'}, axis=1)

park2 = park.groupby(['dong'])['park_area'].agg('sum').reset_index()
park2 = park2.rename({'park_area':'park_areas'}, axis=1)

park_df = pd.merge(park_df, park2, on='dong', how='left')
all = pd.merge(all, park_df, on='dong', how='left')
```

<코드 9> 'park_cnt', 'park_areas' 변수 값을 구하는 과정

이렇게 추가한 두 변수 'park_cnt', 'park_areas'에 대한 'PRICE' 상관계수 값은 <그림 10>과 같다.



<그림 10> 'park_cnt', 'park_areas'에 대한 'PRICE' 상관계수

II. 모델 만들기

XGBoost과 LightGBM 모델을 교차 검증(Cross Validation)을 이용해 학습하도록 구현했다. 각각의 모델에 대해서 'num_leaves', 'subsample'등의 매개변수를 제어하기 위해, hyperparameter tuning을 활용했다. 다음 <코드 10>와 <코드 11>은 각각 LightGBM과 XGBoost 모델의 hyperparameter tuning을 위한 세팅 코드이다. 대회의 평가함수가 MAE를 사용하므로, 'metric'은 'mae'로 설정했다. Tree_method를 설정할 때는 'hist'나 'approx', 'exact'의 옵션이 있는데, 튜닝 시간이 가장 긴 대신 높은 정확도를 보이는 'exact'를 Tree_method로 최종 선택했다. 기본 옵션이 'exact'이므로, 따로 명시를 하지 않아도 된다.

```
def objective(trial):
    params = {
        'objective': 'regression',
        'metric': 'mae',
        'learning_rate': 0.1,
        "n_estimators": 1000,
        "max_depth": -1,
        "num_leaves": trial.suggest_int("num_leaves", 8, 127),
        "subsample": trial.suggest_float("subsample", 0.6, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.6, 1.0),
        'random_state': 42
    }

    model = LGBMRegressor(**params)
    folds = KFold(n_splits=5, shuffle=True, random_state=42)

    score = cross_val_score(model, X, y, cv=folds, scoring="neg_mean_absolute_error")
    return score.mean()
```

<코드 10> LightGBM 모델의 hyperparameter tuning을 위한 세팅

```python
import optuna
from sklearn.model_selection import cross_val_score

def objective(trial):
    params = {
        'objective': 'reg:squarederror',
#         'tree_method': 'hist',
#         'tree_method': 'approx',
        'learning_rate': 0.1,
        'n_estimators': 300,
        'max_depth': trial.suggest_int("max_depth", 3, 12),
        'max_leaves': trial.suggest_int("max_leaves", 64, 1023),
        'subsample': trial.suggest_float("subsample", 0.6, 1.0),
        'colsample_bytree': trial.suggest_float("colsample_bytree", 0.6, 1.0),
        'random_state': 42
    }

    model = XGBRegressor(**params)
    folds = KFold(n_splits=5, shuffle=True, random_state=42)

    score = cross_val_score(model, X, y, cv=folds, scoring="neg_mean_absolute_error")
    return score.mean()
```

<코드 11> XGBoost 모델의 hyperparameter tuning을 위한 세팅

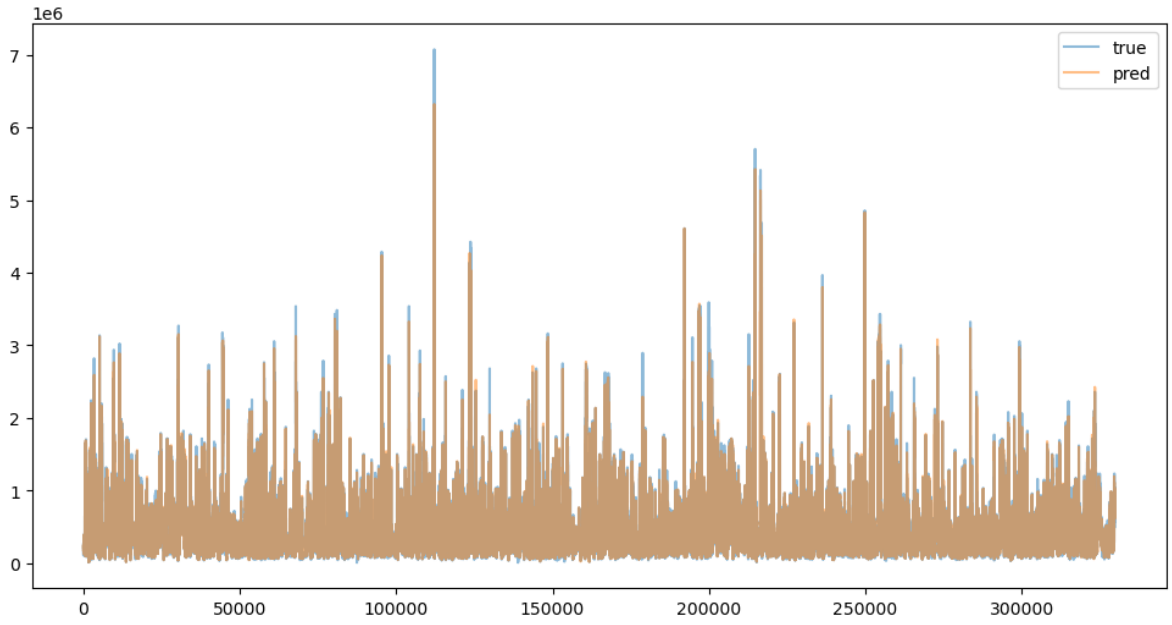
학습률, 반복 횟수 값을 직접 지정하고, hyperparameter tuning결과를 통해 얻은 best_parameters 값을 변수에 대입한다. 이후, 전체 데이터를 fold 5개로 분할하여, 각 fold에 대해서 모델을 학습하고 검증하는 교차 검증을 수행하도록 구현했다. 이를 통해 각 fold에서 얻은 validation 점수를 평균하여 모델의 일반화 성능을 추정할 수 있다. 이에 대한 코드는 <코드 12>와 같다.

```python
folds = KFold(n_splits=5, shuffle=True, random_state=42)
vld_pred = np.zeros(len(X))
tst_pred = np.zeros(len(X_tst))

params = {
    'objective': 'regression',
    'metric': 'mae',
    'learning_rate': 0.01,
    "n_estimators": 10000,
    "max_depth": -1,
    "num_leaves": study.best_params['num_leaves'],
    "subsample": study.best_params['subsample'],
    "colsample_bytree": study.best_params['colsample_bytree'],
    'random_state': 42,
}

for fold_id, (trn_idx, vld_idx) in enumerate(folds.split(X, y)):
    print('\n', '#'*40, f'Fold {fold_id+1} / Fold {folds.n_splits}', '#'*40)
    X_trn = X.loc[trn_idx]
    y_trn = y.loc[trn_idx]
    X_vld = X.loc[vld_idx]
    y_vld = y.loc[vld_idx]

    model = LGBMRegressor(**params)
    model.fit(X_trn, y_trn,
              eval_set=[(X_trn, y_trn), (X_vld, y_vld)],
              early_stopping_rounds=50,
              verbose=100
             )

    vld_pred[vld_idx] = model.predict(X_vld)
    tst_pred         += model.predict(X_tst) / folds.n_splits
```

<코드 12> 교차 검증 수행

앙상블 모델도 활용했다. 앞에서 진행한 XGBoost와 LightGBM 모델에 각각 기여도를 설정하여, 기여도를 곱한 것을 더하여 예측값으로 내놓는 앙상블 모델을 제작하였다.

<AxesSubplot:>



<그림 11> 테스트 데이터에 대한 예측값과 실제값 비교

	city	dong	apartment_id	floor	house_area	tr_year	built_year	dprice21	aprice21	aprice32	PRED	PRICE	diff
112184	1	118	1484	44	325.105962	2022	2011	6.483714e+06	6.483714e+06	7073560.00	5.456030e+06	7073560.0	-1.617530e+06
214817	1	58	2877	55	197.194831	2020	2003	2.027760e+06	2.094736e+06	2251360.00	2.208623e+06	3681946.0	-1.473323e+06
214744	1	58	2877	54	290.721249	2020	2003	4.332943e+06	4.695501e+06	5037520.00	4.069015e+06	5428654.0	-1.359639e+06
214788	1	58	2877	52	290.793009	2020	2003	4.332943e+06	4.695501e+06	5037520.00	5.424212e+06	4073080.0	1.351132e+06
44599	1	90	548	1	127.085897	2018	1974	1.642445e+06	2.026850e+06	2551408.00	1.745834e+06	418924.0	1.326910e+06
...	...	...	...	...	...	...	...	...	...	...	...	...	...
41760	0	51	505	5	19.913592	2022	2015	1.035976e+05	1.035976e+05	118876.00	1.113749e+05	111374.8	1.453125e-01
105091	0	192	1370	23	38.989274	2020	2005	8.773832e+04	1.013877e+05	104891.62	1.049453e+05	104945.2	1.125000e-01
135801	1	3	1788	4	59.201505	2021	1992	2.943170e+05	2.998232e+05	407993.68	2.721149e+05	272114.8	1.062500e-01
84967	1	39	1082	2	71.544122	2022	1997	3.809491e+05	3.532885e+05	406064.80	3.599861e+05	359986.0	9.375000e-02
239005	1	138	3208	1	101.641210	2022	2004	4.815449e+05	4.442378e+05	502508.80	4.457141e+05	445714.0	6.250000e-02

329690 rows × 14 columns

<그림 12> diff = pred – PRICE 값 도표

또한, <그림 11>과 <그림 12>에서, 실제값을 의미하는 파란색의 'true'가 많이 보이고, 예측값에서 실제값을 뺀 값이 음수가 많다는 것을 확인하게 되었다. 이를 통해, 기존 모델의 예측값이 실제 'PRICE'값보다 전체적으로 낮게 예측하는 경향이 있는 것으로 판단되어, '기존 모델이 내놓는 예측 값 중 더 큰 값'을 최종 예측 값으로 도출하는 앙상블 모델도 제작하였다.

RandomForest 모델도 제작을 시도했으나, 실행 시간이 상당히 많이 소요가 되었다. 대처 방안으로 일부 변수만을 사용했지만, validation score에서 좋지 못한 성과를 냈고, 앙상블 모델에 적용하지 않았다.

III. 모델 학습 결과

Model	Major ver.	Minor ver.	변경 내용	Tuning				Training			Test	num leaves	sub sample	col sample	max depth
				Learn rate, Iterations, Trials	Best try	분	Tuning score	Learn rate, Iterations	분	Validation score	Public score				
LGBM	57	1	a/d price31 없음	0.1, 100		4		0.05, 1000	5	15,569.12374	42,933.00329				
LGBM	65	2	a/d price 5개씩 모두 사용	0.1, 1000		26	15,072.39	0.01, 10000	25	15,422.49303	43,014.37131				
		3		0.1, 3000, 30	20	196	14,948.11	0.01, 10000	17	15,538.84617	43,296.63513				
LGBM	66	1	a/d price31 제외 (=v.57)	0.1, 100, 10	3	4	17,113.66	0.05, 1000	5	15,569.09910	*42,933.00329*				
		2		0.1, 1000, 10	4	24	15,211.00	0.01, 10000	24	15,413.83913	42,777.14720				
		3		0.1, 3000, 30	1	152	14,979.34	0.01, 10000	21	15,607.32687	42,809.97444				
LGBM	68	2		0.1, 100, 100	63	49	16,768.22	0.01, 10000		15,616.02418	43,066.84276				
LGBM	70(66)	1	park 추가	0.1, 100, 10	4	4	17,006.00	0.05, 1000	5	15,475.80538	42,849.81808				
		2		0.1, 1000, 10	4	25	15,147.75	0.01, 10000	19	15,526.00260	42,796.01932				
		3		0.1, 100, 100	63	48	16,806.83	0.01, 10000	25	15,415.38986	42,887.77898				
LGBM	71	1	closest_dist 추가	0.1, 100, 10	4	2	17,077.55	0.05, 1000	2	15,530.25085					
		2		0.1, 1000, 10	4	15	15,170.38	0.01, 10000	40	14,992.32091	42,887.99053				
		3	subway_cnt 추가	0.1, 1000, 10	4	15	15,191.24	0.01, 10000	41	14,995.27950	42,868.85098				
LGBM	72	0	ver. 66.2, 피쳐 추가 테스트	0.1, 1000, 10	4	24	15,211.83	0.01, 10000	62	15,006.01408	**42,771.74242**	107	0.684	0.672	
		1		0.1, 1000, 10	4	14	15,211.83	0.01, 50000	42	14,937.73537	42,780.29741	107	0.684	0.672	
		2	지하철, 공원 정보 추가	0.1, 1000, 10	4	16	15,191.37	0.01, 10000	35	14,980.61908	42,788.14033	107	0.684	0.672	
		3	72.0과 동일 조건	0.1, 1000, 10	4	14	15,211.83	0.01, 10000	33	15,006.01408	42,771.77486	107	0.684	0.672	
		4	기본과 prices 15개 피쳐만 사용	0.1, 1000, 10	4	12	15,300.14	0.01, 10000	32	15,115.18327	43,104.51857	107	0.684	0.672	
		5	72.3에서 트레이닝 횟수 줄임	0.1, 1000, 10	4	14	15,211.83	0.01, 3000	9	15,771.76893	42,922.57729	107	0.684	0.672	

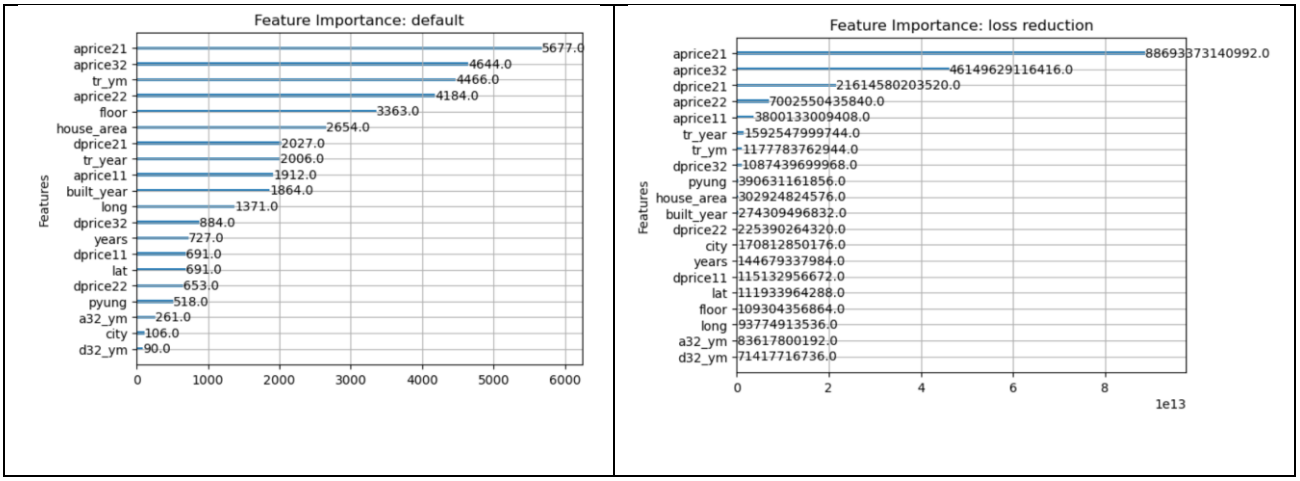
Model	Major ver.	Minor ver.	변경 내용	Tuning				Training			Test	num leaves	sub sample	col sample	max depth
				Learn rate, Iterations, Trials	Best try	분	Tuning score	Learn rate, Iterations	분	Validation score	Public score				
LGBM	73	1	num_leaves: 16~255 증가	0.1, 1000, 100	98	261	14,973.92	0.01, 50000	85	14,789.22866	42,805.45902	244	0.854	0.634	
LGBM	74	1	a/d price23 추가	0.1, 1000, 10	4	34	15,039.06	0.01, 50000	55	14,852.66137	42,959.13485	215	0.684	0.672	
LGBM	75	0	price22를 3개월 단위로 변경	0.1, 300, 10	4	8	15,701.28	0.05, 10000	12	14,707.29410	43,106.44845	107	0.684	0.672	
		1		0.1, 1000, 10	4	39	14,961.07	0.01, 50000	62	14,782.56747	43,119.71455	107	0.684	0.672	
XGB	57	1	a/d price31 없음	0.1, 100		5		0.05, 1000	8	15,673.84230	43,374.36504				
XGB	65	2	a/d price 5개씩 모두 사용	0.1, 1000		32	15,180.76	0.01, 10000	32	15,532.64516	43,172.66925				
		3		0.1, 3000, 30	25	147	15,051.28	0.01, 10000	19	15,698.31155	43,658.52003				
XGB	66	1	a/d price31 제외 (=v.57)								*43,374.36504*				
		2		0.1, 1000, 10	3	15	15,264.88	0.01, 10000	25	15,580.88162	42,864.17136				
		3		0.1, 3000, 30	29	145	15,003.05	0.01, 10000	29	15,449.53268	42,971.37779				
XGB	68	1	iter# 줄이고, Trial 횟수를 증가	0.1, 100, 100, h	94	50	15,937.35	0.01, 1000, e	53	15,276.11581	44,319.46815				
		2		0.1, 100, 100, h	94	50	15,937.35	0.01, 10000, e	93	15,086.10732	44,168.31169				
		3		0.1, 100, 10, e	3	30	16,082.62	0.01, 10000, e	85	15,460.88473	43,472.49149				
XGB	73	1	exact	0.1, 300, 30	21?	360+	15,113.55	0.01, 50000	320	14,693.46070	43,263.80955	403	0.764	0.600	12
		2	hist	0.1, 1000, 10	3	26	15,264.88	0.01, 50000	148	14,903.23695	42,863.92977	267	0.672	0.673	11
		3	hist, 지하철, 공원 정보 추가	0.1, 1000, 10	3	28	15,263.51	0.01, 3000	38	15,529.92757	**42,829.31004**	267	0.672	0.673	11
		4	exact		3			0.01, 500	37	16,277.52804	47,369.92955	267	0.672	0.673	11

Model	Major ver.	Minor ver.	변경 내용	Tuning				Training			Test	num leaves	sub sample	col sample	max depth
				Learn rate, Iterations, Trials	Best try	분	Tuning score	Learn rate, Iterations	분	Validation score	Public score				
Ensemble	65	3	L65.3 + X65.3								43,401.14261				
	73	1	L72.0 + X73.3 (best publc models)								42,682.02590				
		2	L72.1 + X73.1 (best Validation models)								42,944.65682				
		3	L72.0 + X73.3, 둘 중의 최대값								**41,435.35864**				
		4	L72.0 + X73.3, 7:3 비율로								42,690.76751				
		6	L72.0 + X73.3, 둘 중의 최소값								44,165.69381				
RF	66	1	속도 문제로 일부 피쳐만 사용	10회	4	60	17,626.76		89	17,594.61310	46,229.85832				

<표 2> 각 모델의 특징, validation score

변수를 추가하는 과정, hyperparameter tuning 초기 세팅 값, training 세팅 값에 따라 다양한 모델이 제작되었다. <표 2>는 각각의 모델의 추가 내용과 validation score, public score를 정리한 표이다. 최종적으로 대회 제출을 위해 선택한 두 모델은, public_score에서 가장 좋은 성능을 보인 ensemble_73_3 모델과 LGBM_72_0 모델이다. ensemble_73_3 모델은 LGBM_72_0 모델과 XGB_73_3 모델이 예측한 값 중 최대값으로 예측 값을 도출하는 앙상블 모델이다. 각각의 모델에 대한 private score는 41395.1279와 42737.15764였다. 모든 모델에 대한 private score를 비교해본 결과, ensemble_73_3 모델이 private score에서도 가장 좋은 성과를 냈다.

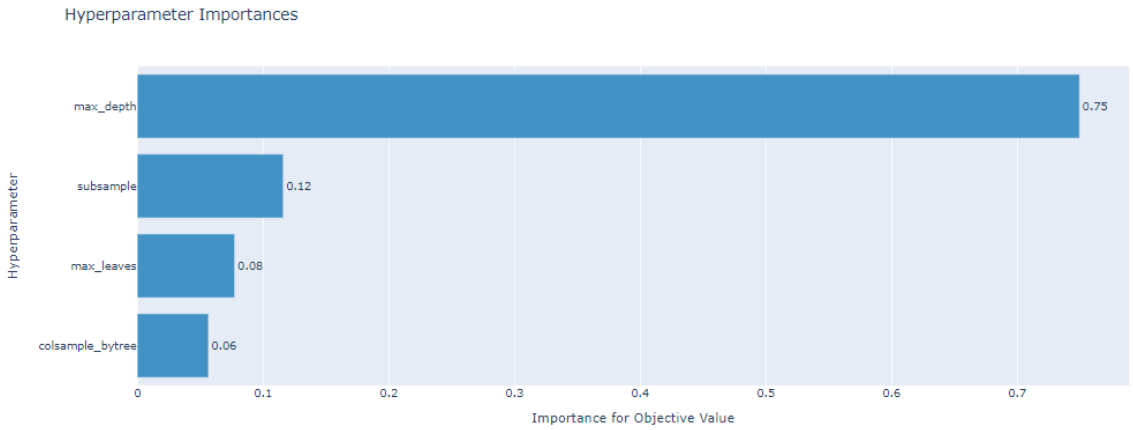
다음 <그림 13>은 XGBoost 라이브러리를 사용하여 모델의 특성 중요도를 시각화한 결과이다. 양쪽 그림에서 볼 수 있듯이, 같은 아파트 id와 평을 가진 집 값 평균을 도출한 'aprice21'이 가장 큰 영향을 끼치는 변수라는 것을 확인할 수 있었다.



<그림 13> 모델의 특성 중요도

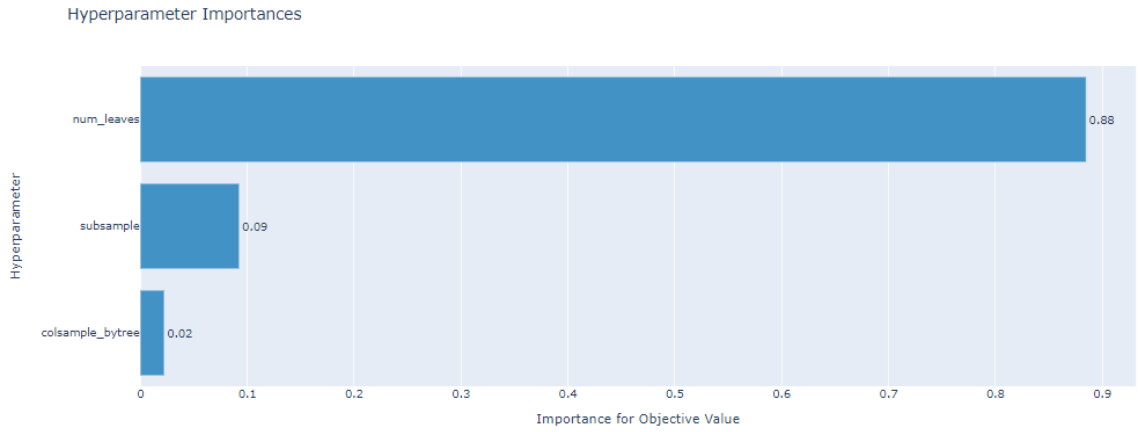
다음 <그림 14>와 <그림 15>는 각각 XGBoost 모델과 LightGBM 모델에서의 hyperparameter tuning 결과로 도출된, 변수의 중요도 그래프이다. 이 그래프를 통해 XGBoost에서는 max_depth가 75%의 중요도, LightGBM에서는 num_leaves가 88%의 중요도로 가장 큰 영향을 미친다는 것을 확인할 수 있다.

```
[ ] # 하이퍼파라미터 중요도
    optuna.visualization.plot_param_importances(study)
```



<그림 14> XGBoost hyperparameter tuning 변수 중요도

```
[ ] # 하이퍼파라미터 중요도
    optuna.visualization.plot_param_importances(study)
```



<그림 15> LightGBM hyperparameter tuning 변수 중요도

IV. 결론 및 제언

모델을 제작하면서 크게 XGBoost 모델과 LightGBM 모델, 2개를 활용하였는데, 시간적으로 LightGBM 모델이 우수했고, 정확도 측면에서도 큰 차이점이 없어 놀랐다. 오히려 대회의 후반부에서는 LightGBM 모델이 근소 우위의 정확도를 보였다.

이번 프로젝트에서 가장 핵심적으로 이용한 아이디어는 train.csv의 'apartment_id'와 'dong' 변수를 활용한 아이디어였다. 각각의 변수는 직접적으로는 'PRICE' 변수와 관련이 없었지만, 변수에서 같은 값을 가지는 것들끼리 그룹을 묶고, 이들의 평균을 구했을 때 큰 관련성을 맺

게 할 수 있었다. 한 가지 신기했던 점은, 'apartment_id' 변수로부터 비슷한 아이디어를 적용하여 무려 'aprice11', 'aprice21', 'aprice22', 'aprice32' 총 4개의 변수를 추가하였는데, 각각의 변수를 추가할 때마다 정확도가 점점 더 우수해졌다는 것이다. 이 변수들은 서로 독립적이지 않고, 연관성이 있기 때문에 과적합의 문제가 발생할 것으로 걱정 했으나, public 데이터에 대해서는 그렇지 않았다. 대회 종료 이후 private score도 확인해보았으나, public score에 비례하는 경향을 띄었다. 즉, 과적합의 문제는 전체 데이터에 대해 발생하지 않았다는 것을 확인할 수 있었다.

또한, 'aprice'와 'dprice' 변수들을 비교해보았을 때, 'aprice'가 'dprice'보다 PRICE값에 더 중요한 영향을 미쳤다. 이는 'aprice' 변수가, 같은 동을 그룹화한 'dprice' 변수보다 더 세밀한 정보를 가진 아파트ID를 그룹화했기 때문이다. 같은 아이디어로, park.csv를 통해 '구'에 대한 변수를 추가했을 때에도 '구(gu)'의 정보는 큰 기여를 하지 못했다. 기존 모델에서는 'gu' 변수를 추가하여 모델 학습을 진행하려 했으나, 위의 이유를 바탕으로 '구'에 대한 변수는 고려하지 않게 되었다.

프로젝트를 마감하면서 아쉬웠던 점이 있다면, 제공된 훈련 데이터 3개의 파일 중 park.csv와 daycare.csv를 충분히 활용하지 못했다는 것이다. 이 두 데이터에 대한 EDA를 진행해보았으나, 상관계수에서 좋은 변수를 찾아내기가 어려웠다. 이번 프로젝트에서는 지하철에 대한 변수를 추가해보았으나, 만약 버스 정류장과 같은 대중교통에 대한 접근성, 근처 초등학교와의 거리에 대한 정보를 참고할 수 있다면, 보다 더 정밀한 모델 제작에 도움이 될 것이다. 이런 아이디어도 train.csv의 위도, 경도 데이터를 통해 활용이 가능할 것이다.

대회 종료 직전 앙상블 모델을 수정하던 도중, 예측 모델이 전체적으로 실제값보다 낮게 값을 예측한다는 것을 깨달았다. '예측 모델들의 최댓값'으로 최종 예측 값을 도출하는 모델로 많이 향상된 성능을 보였다. 즉, 기본 예측 모델들이 실제 답보다 작은 값을 예측 값으로 도출한다는 것을 public score로 확인한 것인데, 이 점을 너무 늦게 깨달은 것 같아 아쉬웠다. 왜 이런 현상이 발생했는지 분석해보고, 좀 더 큰 값을 예측하도록 모델을 수정한다면, 더 좋은 예측 모델이 되지 않았을까 기대한다. RandomForest 모델 또한 제작하였지만, 결과를 도출하는 데까지 상당한 시간이 소요되어 좋은 모델을 만들지 못했고, 앙상블에도 이 모델은 적용하지 않았다. RandomForest 모델을 더 발전시키는 방법을 연구해보는 것도 도움이 될 것이다. 이 프로젝트가 종료된 이후에도 연습 세션이 열린다면, 방학 동안 이런 아이디어를 적용하여 예측모델을 개선하고 더 높은 순위를 받고 싶다. 다양한 데이터 유형과 변수가 제공되어 흥미로운 대회였고, 이로부터 많은 깨달음과 경험을 얻어간 유익한 프로젝트였다.