멋쟁이집사자

서울시 아파트 가격 예측

Contents

1. 프로젝트 주제 소개

1. 프로젝트 주제 소개

최근 부동산 시장에 대한 관심이 높음

수도권 중심으로 부동산 가격이 급격히 상승

아파트 가격을 예측하는 것은

집값 상승률을 반영하여 투자 및 부동산 시장 분석에 많은 부분 활용될 수 있음

1. 프로젝트 주제 소개

아파트 매매가격 결정요인에 관한 선행 연구 분석

공공데이터 포털 및 수집가능한 자료를 기반으로 선정

선행연구를 바탕으로 가격결정요인 선정

모델의 정확도 향상을 위하여 가격결정요인 추가

입력 값을 바탕으로 모델 분석

아파트 매매가격 예측

<프로그램의 흐름>

1. 프로젝트 주제 소개

공공데이터 자료를 바탕으로 한 선행연구를 참고하여 가격결정 요인 설정 (다른 비슷한 연구에서도 매매가격 및 지수를 활용한 연구를 진행)

한국지적정보학회지 제21권 제1호 2019년 04월 pp. 3~12

연구논문

공공데이터를 활용한 아파트 매매 가격 결정 모형의 예측 능력 비교 : 서울 강남구 지역을 중심으로

A study on the sales price of apartment using public data : The apartment in Gangnam-gu Seoul

나 성 호* · 김 종 우**
Na, Seong Ho · Kim, Jong Woo

요 지

본 연구는 국토교통부의 아파트 실거래가 데이터 등 부동산 관련 공공데이터로 다양한 기계학습 알고리즘을 활용하여 서울특별시 강남구 지역의 아파트 매매 가격 결정 모형 간 예측 능력을 비교하였다. 다중선형회귀모형은 이해하기 쉽다는 장점이 있으나 오차의 정규성과 독립성 등의 가정을 충족하기 어렵다는 단점이 있으며 특히 예측 능력 면에서 기계학습 알고리즘에 비해 성능이 낮다. 본 연구에서는 아파트 매매 가격의 추정 성능을 비교하기 위하여 랜덤 포레스트 및 서포트 벡터 머신 알고리즘을 사용하였고, 그 결과 다중회귀분석 모형에 비해 예측 능력이 크게 향상된 결과를 보였다.

핵심용어 : 공공데이터, 다중선형회귀, 랜덤포레스트, 서포트벡터머신

연구에 활용된 데이터

- 아파트 매매거래 금액 사용
- 입력변수 중에서 전용면적, 층 그대로 사용
- 경과년수
  (입주년월로 부터 거래년월까지 개월수 개산)
- 공동주택관리정보시스템에서 아파트 시공능력 순위 데이터를 수집하여 입력 변수에 추가
- 아파트 시공능력 순위 20위 이내인 경우는 1
  그렇지 않은 경우는 0을 부여

1. 프로젝트 주제 소개

매매가격결정 요인

1. 2021 아파트 실거래가 추가
2. 매매가격지수 추가(연도별 평균)
3. 경과년수 추가(2021-건축년도)
4. 우수 시공사 가점 추가 (TOP 10 = 1점 or 0점)

우수 시공사 Top 10

1) 삼성물산 - 레미안
2) 현대건설 – 힐스테이트
3) GS건설 – 자이
4) 포스코 건설
5) 대우건설 –푸르지오
6) 현대엔지니어링
7) 롯데건설 – 롯데캐슬
8) DL E&C
9) HDC현대산업개발 – 아이파크
10) SK에코플랜트

2. 데이터 소개/탐색/시각화

2. 데이터 소개

- 2016년 ~ 2020년 서울특별시 부동산 실거래가 정보
  - 서울시 부동산 실거래 자료로 아파트/연립주택/단독주택/오피스텔 포함
  - 출처 <서울 열린데이터 광장>

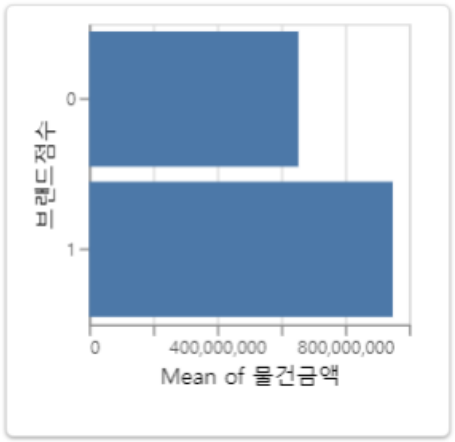
- 2021년 서울특별시 부동산 실거래가 정보
  - 서울시 부동산 실거래 자료로 아파트만 선택
  - 출처 <국토교통부 실거래가 공개시스템>

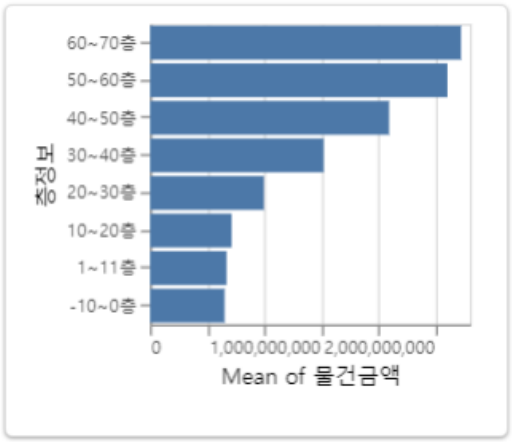
- 아파트 매매가격지수
  - 서울특별시 아파트 매매가격지수 월별 자료를 평균 연도별 자료로 수정
    (매매지수100% = 2021.6월 기준)
  - 출처 <KOSIS 국가통계포털>

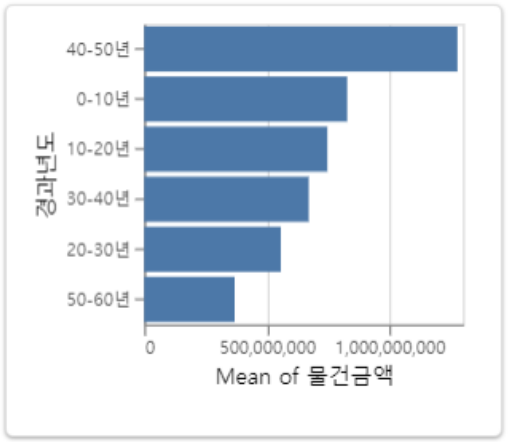
2. 데이터 탐색

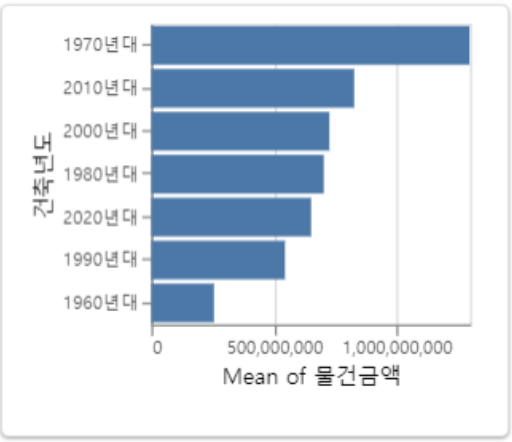
- 건물용도에 따라 데이터 분류 가능(단독주택/연립주택/오피스텔/아파트)
- 건물용도에 따라 Feature로 사용할 수 있는 값이 변경(대지권 면적, 층 정보 등)
- 데이터가 방대해 비교적 수요가 높은 아파트로 대상을 한정해 프로젝트 진행

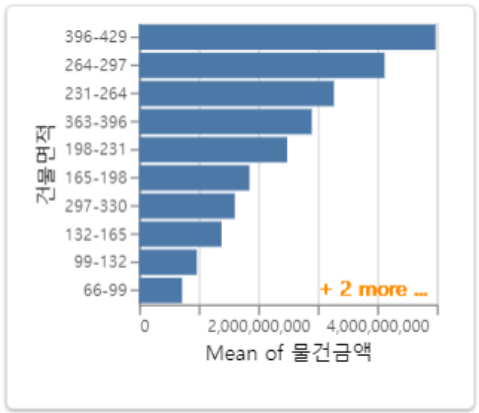
2. 데이터 시각화

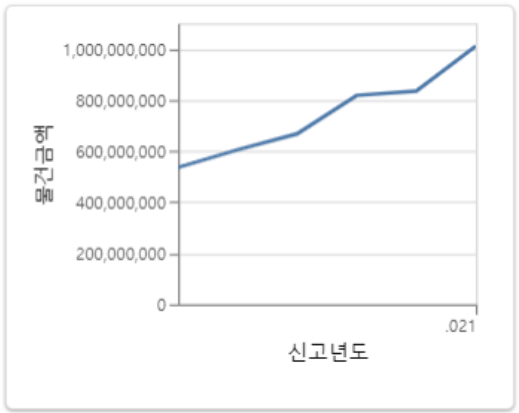

브랜드점수
0
1
0
400,000,000
800,000,000
Mean of 물건금액

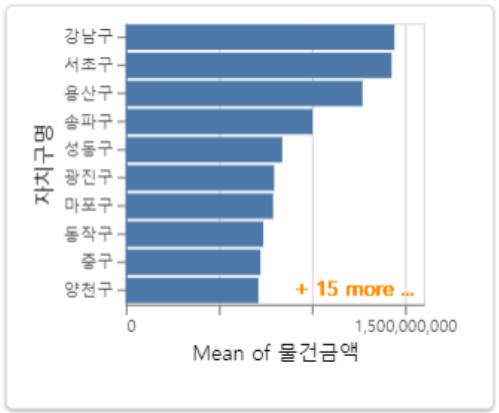

층정보
60~70층
50~60층
40~50층
30~40층
20~30층
10~20층
1~11층
-10~0층
0
1,000,000,000
2,000,000,000
Mean of 물건금액

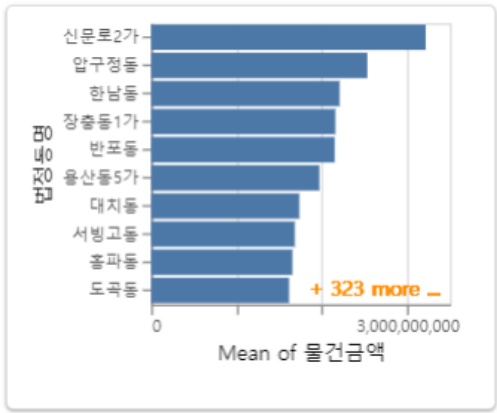

경과년도
40-50년
0-10년
10-20년
30-40년
20-30년
50-60년
0
500,000,000
1,000,000,000
Mean of 물건금액


건축년도
1970년대
2010년대
2000년대
1980년대
2020년대
1990년대
1960년대
0
500,000,000
1,000,000,000
Mean of 물건금액


건물면적
396-429
264-297
231-264
363-396
198-231
165-198
297-330
132-165
99-132
66-99
+ 2 more ...
0
2,000,000,000
4,000,000,000
Mean of 물건금액


물건금액
1,000,000,000
800,000,000
600,000,000
400,000,000
200,000,000
0
.021
신고년도


자치구명
강남구
서초구
용산구
송파구
성동구
광진구
마포구
동작구
중구
양천구
+ 15 more ...
0
1,500,000,000
Mean of 물건금액


법정동명
신문로2가
압구정동
한남동
장충동1가
반포동
용산동5가
대치동
서빙고동
홍파동
도곡동
+ 323 more ...
0
3,000,000,000
Mean of 물건금액

3. 데이터 전처리 과정

3. 데이터 전처리 과정

1) 불필요 Feature 삭제

· 예측과 관련 없는 Feature (실거래가아이디, 업무구분, 물건번호)
  > 예측에 필요하지 않다고 판단해 삭제

· 다른 Feature가 대체 가능한 Feature (지번코드, 시군구코드, … 관리구분코드, 건물주용도코드)
  > 기존에 있는 명칭으로 대체가 가능하기 때문에 삭제

· 건물주용도가 아파트인 경우
  > 예측 대상이 아파트로 아파트는 대지권면적이 없기 때문에 삭제

2) 2021년도 실거래 데이터 매칭 작업

· 기존 Feature 가공, 단위 변경 진행 (시군구, 거래금액)

· Feature 구성이 다르고 활용 불가한 부분은 삭제 (번지,본번,거래유형,…도로명,중개사소재지)

3) 2016~2020년 2021년 실거래 데이터 병합

· 실거래 데이터 병합
  > 전처리 된 데이터를 merge를 통해 병합 처리

3. 데이터 전처리 과정

4) 실거래 데이터에 매매지수 추가

· 실거래 데이터에 매매지수 추가
  > 매매지수 데이터를 신고년도와 자치구로 매칭해 기존 실거래 데이터에 새로운 Feature로 추가

5) 결측치 삭제

· 층정보 & 건축년도 결측치 보유 Row 삭제(24,616개)

6) 기존 자료를 이용한 새로운 Feature 생성

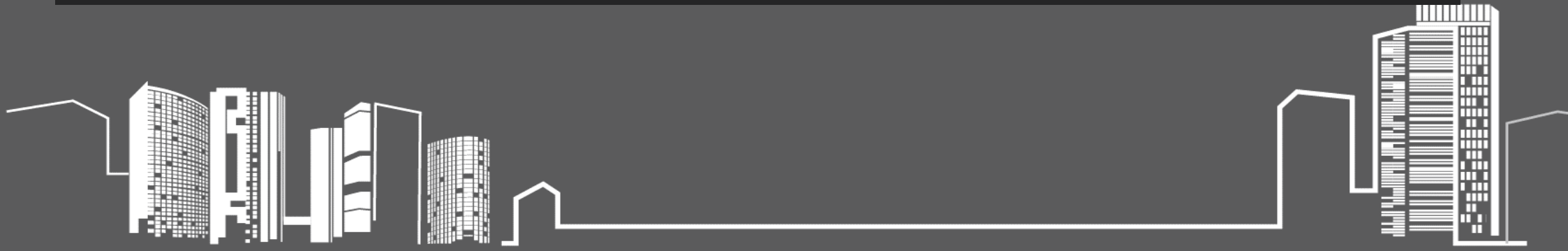
· 브랜드점수
  > 국토교통부 2021 시공사 능력 평가 Top 10에 해당 하는 브랜드는 1, 아닌 곳은 0

· 경과년도
  > 2021년 – 건축년도

3. 데이터 전처리 과정

```
apart_2021 = apart_2021[apart_2021['해제사유발생일'].isnull()] # 계약 해제사유발생 건 삭제
apart_2021.drop(['번지','본번','부번','계약일','도로명','해제사유발생일','거래유형','중개사소재지'],axis=1, inplace=True) # 불필요 열 삭제

apart_2021['자치구명'] = apart_2021['시군구'].str.split(" ").str[1] # 자치구명 추출
apart_2021['법정동명'] = apart_2021['시군구'].str.split(" ").str[2] # 법정동명 추출
apart_2021.drop(['시군구'], axis='columns', inplace=True) # 시군구 열 삭제

apart_2021['계약년월'] = apart_2021['계약년월'] = 2021 # 계약년월 전체 2021년으로 대체

apart_2021['거래금액(만원)'] = apart_2021['거래금액(만원)'].str.replace(",", '') # 거래금액 열 내 ',' 제거
apart_2021['거래금액(만원)'] = apart_2021['거래금액(만원)'].astype(int) # 거래금액 열 타입 변경(object > int)
apart_2021['거래금액(만원)'] = apart_2021['거래금액(만원)'].apply(lambda x : x * 10000) # 거래금액 열 단위(만원 > 원) 변경

apart_2021 = apart_2021.rename(columns={'단지명':'건물명', '전용면적(㎡)':'건물면적', '거래금액(만원)':'물건금액', '층':'층정보', '계약년월':'신고년도'}) # 열 이름 변경
apart_2021 = apart_2021[['자치구명','법정동명','신고년도','건물면적','층정보','물건금액','건축년도','건물명']] # 열 순서 변경
```

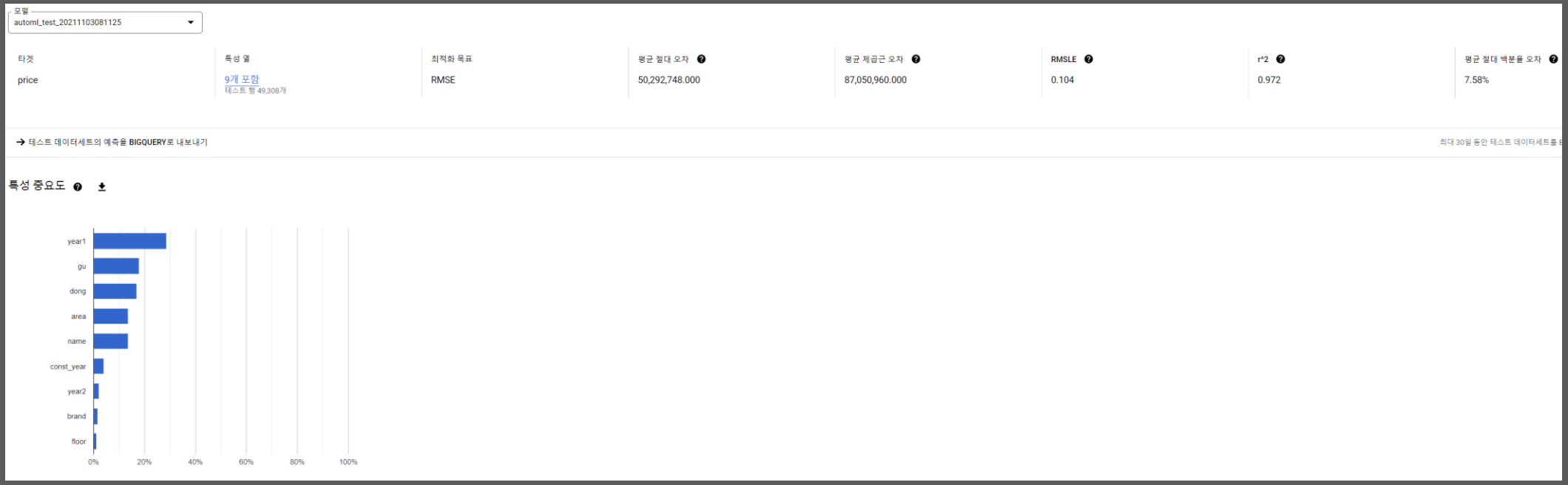
```
floor_drop_index = apart_all[apart_all['층정보'] == 0].index # 층정보 결측치 삭제
apart_all.drop(floor_drop_index, inplace=True)
const_year_index = apart_all[apart_all['건축년도'] == 0].index # 건축년도 결측치 삭제
apart_all.drop(const_year_index, inplace=True)

apart_all['경과년도'] = apart_all['건축년도'].apply(lambda x : int(datetime.datetime.now().year) - x) # 현재년도 - 건축년도
apart_all['경과년도'] = apart_all[['경과년도']].replace(int(datetime.datetime.now().year),0) # 0 값으로 생긴 현재년도 값 0으로 대체

brands = ['래미안','힐스테이트','자이','더샵','#','푸르지오','힐스테이트','롯데캐슬','아크로','아이파크','IPARK','에스케이뷰','SKVIEW'] # 시공사 브랜드 Top 10
brand = '|'.join(brands) # contains 기능을 활용해 한번에 찾아낼 수 있도록 List > Str 형태로 변형하며 브랜드 사이에 |(or) 입력
apart_all['브랜드점수'] = apart_all['건물명'].str.contains(brand) # 건물명에 Top10 브랜드가 있는 경우 True 아니면 False
apart_all['브랜드점수'] = apart_all['브랜드점수'].apply(lambda x : 1 if x == True else 0)

apart_all['법정동명'] = apart_all['법정동명'].apply(lambda x : x[:-2] if x[-1] in '가' else x) # 법정동명 값 중 OO동 뒤에 'X가' 같은 구획이 붙은 경우 삭제
```



4. 적용 분석기법 및 모델 소개

4. 분석기법 및 모델소개

전처리 완료된 데이터를 Google AutoML을 통해 대략적인 성능 확인



4. 분석기법 및 모델소개

회귀 모델을 랜덤하게 하나씩 선별해 기본 HyperParameter로 성능 확인

```
[70]
model1.fit(x_train_transformed, y_train)
model1_predict_y = model1.predict(x_test_transformed)
print(f'Catboost RMSE : {(mean_squared_error(y_test, model1_predict_y)**0.5):.1f}')
print(f'Catboost MAE : {(mean_absolute_error(y_test, model1_predict_y)**0.5):.1f}')
print(f'Catboost R2 score : {(r2_score(y_test, model1_predict_y)*100):.2f} %')

...
Catboost RMSE : 133106093.9
Catboost MAE : 9451.4
Catboost R2 score : 93.32 %
```

```
[71]
model2.fit(x_train_transformed, y_train)
model2_predict_y = model2.predict(x_test_transformed)
print(f'LightGBM RMSE : {(mean_squared_error(y_test, model2_predict_y)**0.5):.1f}')
print(f'LightGBM MAE : {(mean_absolute_error(y_test, model2_predict_y)**0.5):.1f}')
print(f'LightGBM R2 score : {(r2_score(y_test, model2_predict_y)*100):.2f} %')

...
LightGBM RMSE : 158839864.7
LightGBM MAE : 10319.1
LightGBM R2 score : 90.49 %
```

```
[72]
model3.fit(x_train_transformed, y_train)
model3_predict_y = model3.predict(x_test_transformed)
print(f'ExtraTree RMSE : {(mean_squared_error(y_test, model3_predict_y)**0.5):.1f}')
print(f'ExtraTree MAE : {(mean_absolute_error(y_test, model3_predict_y)**0.5):.1f}')
print(f'ExtraTree R2 score : {(r2_score(y_test, model3_predict_y)*100):.2f} %')

...
ExtraTree RMSE : 106738344.4
ExtraTree MAE : 7565.5
ExtraTree R2 score : 95.71 %
```

```
[73]
model4.fit(x_train_transformed, y_train)
model4_predict_y = model4.predict(x_test_transformed)
print(f'XGBoost RMSE : {(mean_squared_error(y_test, model4_predict_y)**0.5):.1f}')
print(f'XGBoost MAE : {(mean_absolute_error(y_test, model4_predict_y)**0.5):.1f}')
print(f'XGBoost R2 score : {(r2_score(y_test, model4_predict_y)*100):.2f} %')

...
XGBoost RMSE : 145695009.3
XGBoost MAE : 9906.4
XGBoost R2 score : 92.00 %
```

4. 분석기법 및 모델소개

- 모델들의 RMSE 값이 8천만원 ~ 1억대로 형성
- 지역 간 부동산 시세의 차이가 RMSE 값에 영향이 있다고 판단
- 서울 전체의 실거래 데이터를 구별 단위로 분리 후 모델 학습 시 RMSE 값을 낮출 수 있을지 테스트 진행

```
models = {'KNeighbors':KNeighborsRegressor(),'Randomforest':RandomForestRegressor(),'ExtraTree':ExtraTreeRegressor(),'MLP':MLPRegressor(), 'SGD':SGDRegressor(),
'SVM':SVR(),'CatBoost':CatBoostRegressor(),'LightGBM':LGBMRegressor(),'XGBoost':XGBRegressor()}
```

```
model_score = {} # 모델 성능 저장용 dict

# 모델간 점수 비교
for name, attr in models.items():
    model = attr
    print(f'{name} model tranning ...')
    model.fit(x_train_transformed, y_train)
    predict_y= model.predict(x_test_transformed)
    model_score_list = []
    model_score_list.append(f'{name} RMSE : {(mean_squared_error(y_test, predict_y)**0.5):.1f}')
    model_score_list.append(f'{name} MAE : {(mean_absolute_error(y_test, predict_y)**0.5):.1f}')
    model_score_list.append(f'{name} R2 score : {(r2_score(y_test, predict_y)*100):.2f} %')
    model_score[f'{name}'] = model_score_list
    model_score_list = []
```

4. 분석기법 및 모델소개

모델 성능 테스트 진행 중 R2 Score가 – 되는 값이 발생되는 모델에 HyperParameter 조정 후 정상화 확인

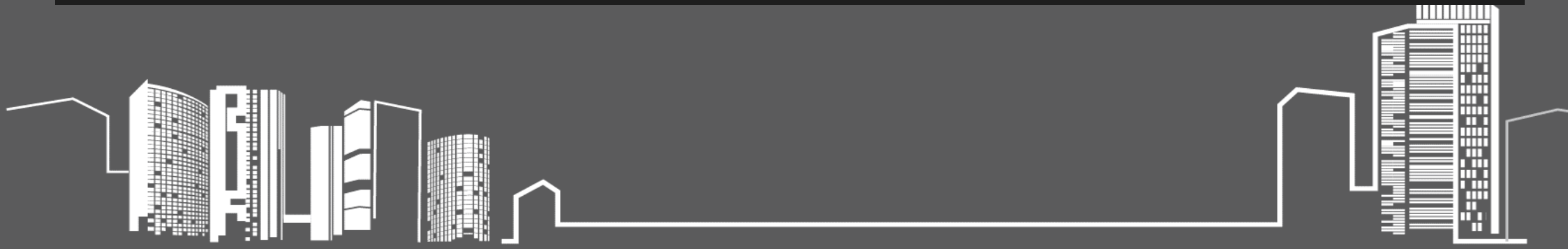
```
KNeighbors
['KNeighbors RMSE : 49512378.1', 'KNeighbors MAE : 5718.6', 'KNeighbors R2 score : 92.61 %']

Randomforest
['Randomforest RMSE : 45102920.2', 'Randomforest MAE : 5405.4', 'Randomforest R2 score : 93.87 %']

ExtraTree
['ExtraTree RMSE : 50804490.5', 'ExtraTree MAE : 5693.8', 'ExtraTree R2 score : 92.22 %']

MLP
['MLP RMSE : 451292082.4', 'MLP MAE : 20322.2', 'MLP R2 score : -513.97 %']

SGD
['SGD RMSE : 66167763.1', 'SGD MAE : 6945.6', 'SGD R2 score : 86.80 %']

SVM
['SVM RMSE : 186790996.8', 'SVM MAE : 11640.1', 'SVM R2 score : -5.18 %']

CatBoost
['CatBoost RMSE : 40794724.9', 'CatBoost MAE : 5279.9', 'CatBoost R2 score : 94.98 %']

LightGBM
['LightGBM RMSE : 43444983.6', 'LightGBM MAE : 5436.8', 'LightGBM R2 score : 94.31 %']

XGBoost
['XGBoost RMSE : 42993633.3', 'XGBoost MAE : 5424.0', 'XGBoost R2 score : 94.43 %']
```

4. 분석기법 및 모델소개

모델 성능 테스트 상위 5개를 GridSearch를 통한 HPO 진행

```
# MLP GridSearch
model1 = MLPRegressor()

param_grid_mlp={'hidden_layer_sizes': [(32,64),(128,64),(128,256)],
                'batch_size':  [50,100,200],
                'learning_rate_init': [0.01,0.05],
                'max_iter': [100,300,400]
                }

gs1 = GridSearchCV(model1, param_grid_mlp, scoring='neg_mean_squared_error', n_jobs=-1, cv=10, verbose=False)

gs1.fit(x_train_transformed, y_train)

gs1_test_score = mean_squared_error(y_train, gs1.predict(x_train_transformed))
print(f'Best RMSE {(-gs1.best_score_)**0.5} params {gs1.best_params_}')
print()
```

```
Best RMSE 41299066.89491322 params {'batch_size': 50, 'hidden_layer_sizes': (128, 256), 'learning_rate_init': 0.05, 'max_iter': 400}
```



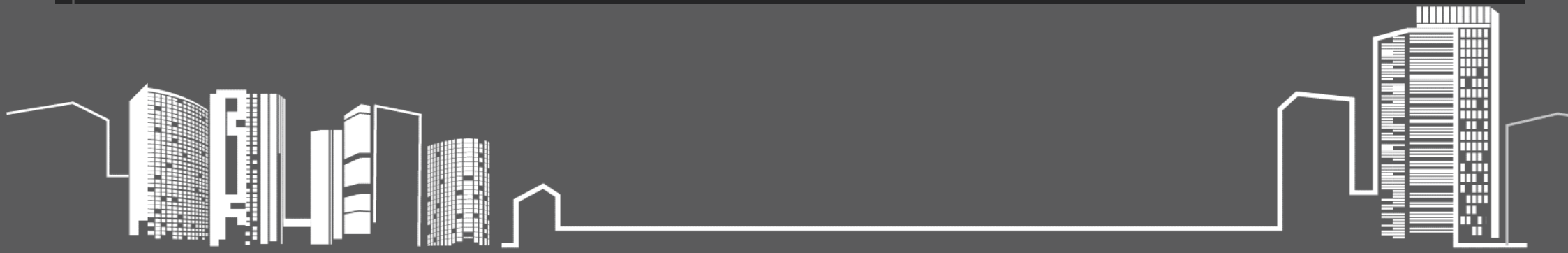
4. 분석기법 및 모델소개

Stacking 기법을 활용하기 위해 Top 5 모델 선정 시 비슷한 R2 Score를 가지고 있는 경우
알고리즘 기반이 다른 모델들 위주로 선별

전체 데이터에서 for문을 통해 25개 구에 대한 분리작업 진행

- DataFrame 구별 생성
- Stacking을 적용한 모델 학습
- Web 적용을 위한 Model/Pipeline/StackingTransformer pkl 파일 저장

```
for kor, eng in gu.items():
    globals()[f'df_{eng}'] = apart_sep[apart_sep['자치구명']==kor] # df_nowon, df_gangnam ...

    x = globals()[f'df_{eng}'].drop(['물건금액'],axis=1)
    y = globals()[f'df_{eng}']['물건금액']

    globals()[f'df_{eng}'].to_csv(f'{kor}.csv', encoding='cp949', index=False) # 구별 csv 저장
```



4. 분석기법 및 모델소개

```
# stacking 1 level model 지정
estimators = [
('LightGBM', LGBMRegressor(learning_rate=0.05, n_estimators=300, num_leaves=100)),
('Catboost', XGBRegressor(n_estimators=250)),
('Lightgbm', RandomForestRegressor(n_estimators=200))
('MLP', MLPRegressor(batch_size=50, hidden_layer_sizes=(128, 256), learning_rate_init=0.05, max_iter=400))]

# Initialize StackingTransformer
stack = StackingTransformer(estimators,
                            regression = True,
                            metric = mean_squared_error,
                            n_folds = 10, stratified = True, shuffle = True,
                            random_state = 123, verbose = 0)

# Fit
stack = stack.fit(x_train_transformed, y_train)

# Get your stacked features
S_train = stack.transform(x_train_transformed)
S_test = stack.transform(x_test_transformed)

# Use 2nd level estimator with stacked features
model = CatBoostRegressor(depth=10, l2_leaf_reg = 1, iterations=500, learning_rate = 0.05)
model = model.fit(S_train, y_train)
```

4. 분석기법 및 모델소개

```
    # 모델 성능평가 점수를 구별로 list에 모아둔 뒤 dict로 저장
    model_score_temp = []
    model_score_temp.append(f'{kor}')
    model_score_temp.append((f'RMSE : {(mean_squared_error(y_test, y_pred)**0.5):.1f}'))
    model_score_temp.append((f'MAE : {(mean_absolute_error(y_test, y_pred)**0.5):.1f}'))
    model_score_temp.append((f'R2 score : {(r2_score(y_test, y_pred)*100):.2f} %'))
    model_score[kor] = model_score_temp
    model_score_temp = [] # 성능평가 점수 초기화

    # 구별 pkl 파일 저장 [Model, Pipeline(feature scaling), stacking]
    joblib.dump(model, f'model_apart_{eng}.pkl', compress=True) # model save
    joblib.dump(preprocessor_pipe, f'pipeline_{eng}.pkl', compress=True) # feature scaling pipeline save
    joblib.dump(stack, f'stack_{eng}.pkl', compress=True) # stacking transform save

# 모델 성능 DataFrame화 후 csv로 저장
df2 = pd.DataFrame.from_dict(model_score, orient='index')
df2.to_csv('model_score(Stacking).csv', encoding='cp949', index=False)
```

5. 분석 및 모델링 결과

5. 분석 및 모델링 결과

HPO 전 + 높은 성능 모델만 Stacking

	RMSE	R2 Score
노원구	0.58억	89.96%
송파구	1.32억	93.37%
강서구	0.7억	92.43%
강남구	2.44억	91.14%
강동구	0.93억	91.05%
구로구	0.63억	92.02%
성북구	0.65억	89.71%
양천구	1.06억	93.84%
도봉구	0.54억	89.70%
서초구	2.3억	91.44%
영등포구	0.93억	94.57%
성동구	1.18억	93.35%
마포구	1.04억	91.27%
동작구	0.89억	91.06%
동대문구	0.67억	92.54%
은평구	0.68억	90.16%
중랑구	0.55억	89.92%
서대문구	0.81억	91.96%
관악구	0.65억	89.41%
용산구	1.95억	96.08%
강북구	0.57억	89.27%
광진구	1.16억	90.48%
금천구	0.5억	92.25%
중구	1.15억	89.72%
종로구	0.92억	95.53%

5. 분석 및 모델링 결과

HPO 후 + 여러 기반 모델 Stacking 1

```
# stacking 1 level model 지정
estimators = [
('LightGBM', LGBMRegressor(learning_rate=0.05, n_estimators=300, num_leaves=100)),
('XGBoost', XGBRegressor(n_estimators=250)),
('RandomForest', RandomForestRegressor(n_estimators=200)),]
# ('MLP', MLPRegressor(batch_size=50, hidden_layer_sizes=(128,256), learning_rate_init=0.05, max_iter=400))]

# Initialize StackingTransformer
stack = StackingTransformer(estimators,
                            regression = True,
                            metric = mean_squared_error,
                            n_folds = 10, stratified = True, shuffle = True,
                            random_state = 123, verbose = 0)

# Fit
stack = stack.fit(x_train_transformed, y_train)

# Get your stacked features
S_train = stack.transform(x_train_transformed)
S_test = stack.transform(x_test_transformed)

# Use 2nd level estimator with stacked features
model = CatBoostRegressor(depth=10, l2_leaf_reg = 1, iterations=500, learning_rate = 0.05)
model = model.fit(S_train, y_train)
```

노원구	RMSE : 39	MAE : 512	R2 score :	95.25 %
송파구	RMSE : 94	MAE : 801	R2 score :	96.65 %
강서구	RMSE : 48	MAE : 577	R2 score :	96.35 %
강남구	RMSE : 16	MAE : 102	R2 score :	95.73 %
강동구	RMSE : 68	MAE : 665	R2 score :	95.20 %
구로구	RMSE : 42	MAE : 519	R2 score :	96.28 %
성북구	RMSE : 47	MAE : 557	R2 score :	94.52 %
양천구	RMSE : 75	MAE : 703	R2 score :	96.85 %
도봉구	RMSE : 37	MAE : 497	R2 score :	94.91 %
서초구	RMSE : 15	MAE : 100	R2 score :	95.92 %
영등포구	RMSE : 69	MAE : 653	R2 score :	96.96 %
성동구	RMSE : 87	MAE : 730	R2 score :	96.32 %
마포구	RMSE : 75	MAE : 704	R2 score :	95.36 %
동작구	RMSE : 67	MAE : 653	R2 score :	94.75 %
동대문구	RMSE : 48	MAE : 562	R2 score :	96.02 %
은평구	RMSE : 46	MAE : 557	R2 score :	95.38 %
중랑구	RMSE : 39	MAE : 491	R2 score :	94.82 %
서대문구	RMSE : 56	MAE : 615	R2 score :	96.15 %
관악구	RMSE : 46	MAE : 551	R2 score :	94.61 %
용산구	RMSE : 15	MAE : 927	R2 score :	97.45 %
강북구	RMSE : 44	MAE : 533	R2 score :	93.40 %
광진구	RMSE : 80	MAE : 729	R2 score :	95.34 %
금천구	RMSE : 38	MAE : 483	R2 score :	95.35 %
중구	RMSE : 10	MAE : 719	R2 score :	91.95 %
종로구	RMSE : 75	MAE : 679	R2 score :	96.97 %

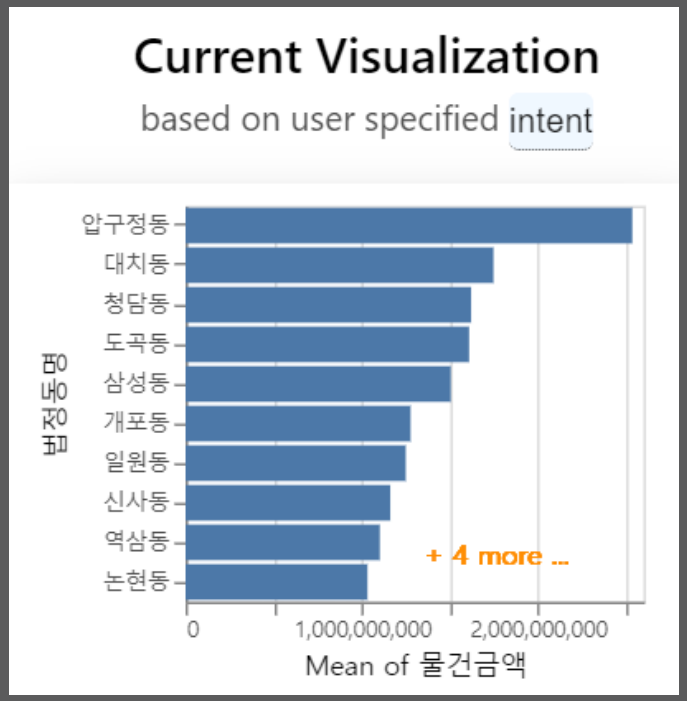
5. 분석 및 모델링 결과

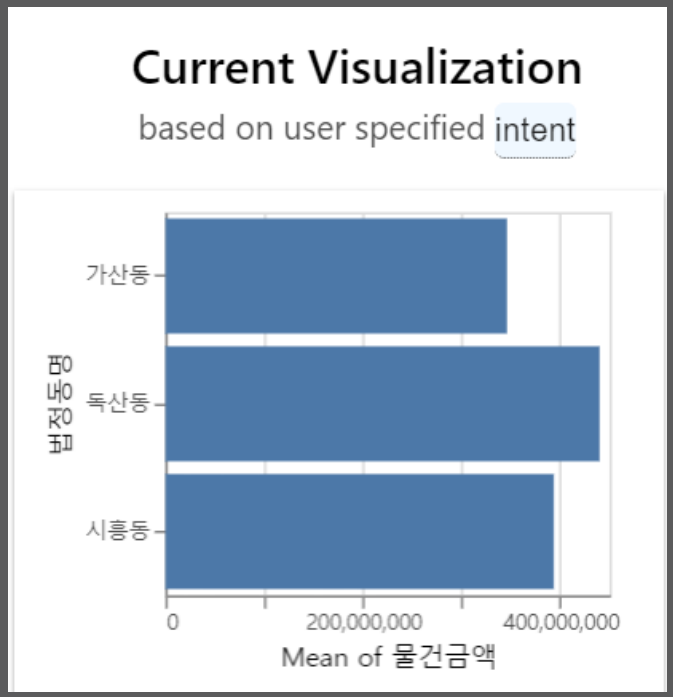
HPO 후 + 여러 기반 모델 Stacking 2

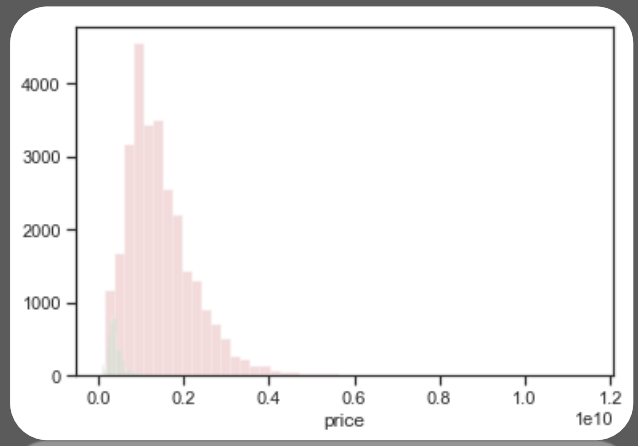
```
# stacking 1 level model 지정
estimators = [
('LightGBM', LGBMRegressor(learning_rate=0.05, n_estimators=300, num_leaves=100)),
('XGBoost', XGBRegressor(n_estimators=250)),
('RandomForest', RandomForestRegressor(n_estimators=200)),
('MLP', MLPRegressor(batch_size=50, hidden_layer_sizes=(128,256), learning_rate_init=0.05, max_iter=400))]

# Initialize StackingTransformer
stack = StackingTransformer(estimators,
                            regression = True,
                            metric = mean_squared_error,
                            n_folds = 10, stratified = True, shuffle = True,
                            random_state = 123, verbose = 0)

# Fit
stack = stack.fit(x_train_transformed, y_train)

# Get your stacked features
S_train = stack.transform(x_train_transformed)
S_test = stack.transform(x_test_transformed)

# Use 2nd level estimator with stacked features
model = CatBoostRegressor(depth=10, l2_leaf_reg = 1, iterations=500, learning_rate = 0.05)
model = model.fit(S_train, y_train)
```

노원구	RMSE : 39337719.0	R2 score : 95.34 %
송파구	RMSE : 91207431.4	R2 score : 96.86 %
강서구	RMSE : 47190315.3	R2 score : 96.56 %
강남구	RMSE : 159646531.9	R2 score : 96.21 %
강동구	RMSE : 66535497.3	R2 score : 95.45 %
구로구	RMSE : 40502386.9	R2 score : 96.65 %
성북구	RMSE : 46164904.3	R2 score : 94.84 %
양천구	RMSE : 70759591.7	R2 score : 97.23 %
도봉구	RMSE : 37073519.9	R2 score : 95.06 %
서초구	RMSE : 151424035.5	R2 score : 96.28 %
영등포구	RMSE : 67864105.1	R2 score : 97.12 %
성동구	RMSE : 85243101.7	R2 score : 96.51 %
마포구	RMSE : 72753442.3	R2 score : 95.71 %
동작구	RMSE : 65591506.8	R2 score : 95.10 %
동대문구	RMSE : 45953154.0	R2 score : 96.46 %
은평구	RMSE : 45229096.2	R2 score : 95.70 %
중랑구	RMSE : 38675513.3	R2 score : 95.07 %
서대문구	RMSE : 54084668.4	R2 score : 96.44 %
관악구	RMSE : 44487558.9	R2 score : 94.98 %
용산구	RMSE : 151582481.3	R2 score : 97.64 %
강북구	RMSE : 43352811.7	R2 score : 93.84 %
광진구	RMSE : 78356343.0	R2 score : 95.62 %
금천구	RMSE : 35889685.1	R2 score : 95.97 %
중구	RMSE : 98376788.3	R2 score : 92.44 %
종로구	RMSE : 68043774.0	R2 score : 97.55 %

5. 분석 및 모델링 결과


Current Visualization
based on user specified intent
법정동명
압구정동
대치동
청담동
도곡동
삼성동
개포동
일원동
신사동
역삼동
논현동
+ 4 more ...
0
1,000,000,000
2,000,000,000
Mean of 물건금액


Current Visualization
based on user specified intent
법정동명
가산동
독산동
시흥동
0
200,000,000
400,000,000
Mean of 물건금액


4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
price
1e10

5. 분석 및 모델링 결과

· 구 별로 모델을 분리 후 RMSE 편차 발생

· 동 별로 부동산가격 편차 발생 이유 추측
  1) 형성된 부동산 시세의 차이
  2) 동 사이의 부동산 시세의 차이

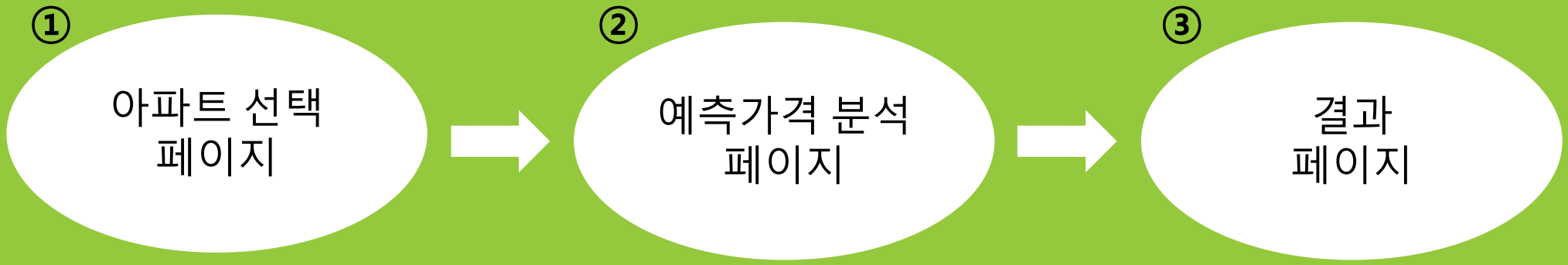
· RMSE 값이 가장 높은 강남구와 가장 낮은 금천구의 동 별 부동산금액을 확인

· 동 별 부동산 가격비교 시 RMSE 값은 줄어들 수 있으나 데이터도 줄어 언더피팅 가능성 존재

6. 모델을 활용한 웹 서비스 소개

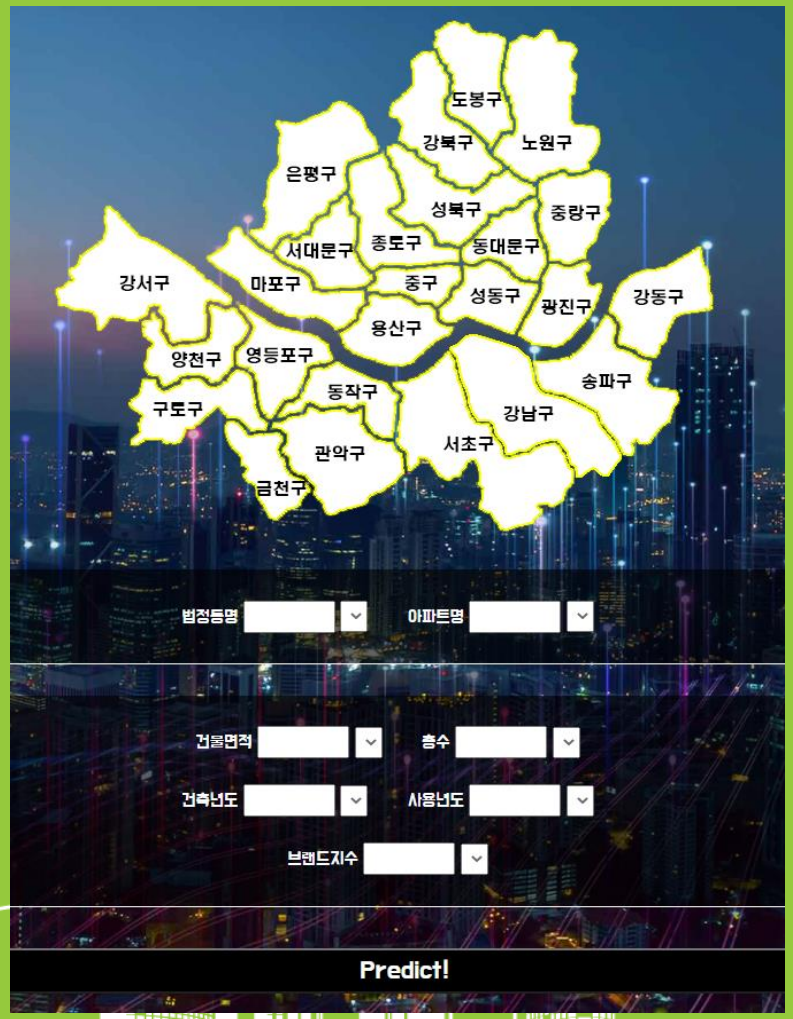
1-1 . 서비스 Flow

User experience를 최우선으로 고려해 간편하게 결과를 볼 수 있도록 구성



1-2-1 . 화면 예시 및 주요 소스코드

1. 아파트 선택 페이지



[map tag를 통해 구별로 click 가능]

```
<div class="seoul_map" id='seoul_map'>
    <img src="static/predict_price/images/0.png" alt="서울시지도"
    <map name='city-map' id="city-map-link">
        <area shape="poly" coords="350,9,368,8,375,21,387,15,402,
```

[Click event 발생 시 Java Script로 '동' list 생성]

```
<script>
    function selectImg(gu_dong_group, gu){
        $("#change_input_gu").val(gu);
    //     alert(gu);

        var data_dong = gu_dong_group[gu];
        var select_dong = document.querySelector('#change_test_dong');

        var disabled = document.createElement('option');
        disabled.innerText = "Please select one!";
        select_dong.append(disabled);

        for (var i = 0; i < data_dong.length; i++) {
            var option_dong = document.createElement('option');
            option_dong.innerText = data_dong[i];
            select_dong.append(option_dong);
        }
    }
```

1-2-2 . 화면 예시 및 주요 소스코드

2. 예측가격 분석 페이지

[Bootstrap을 사용하여 간편하게 Spinner 생성]

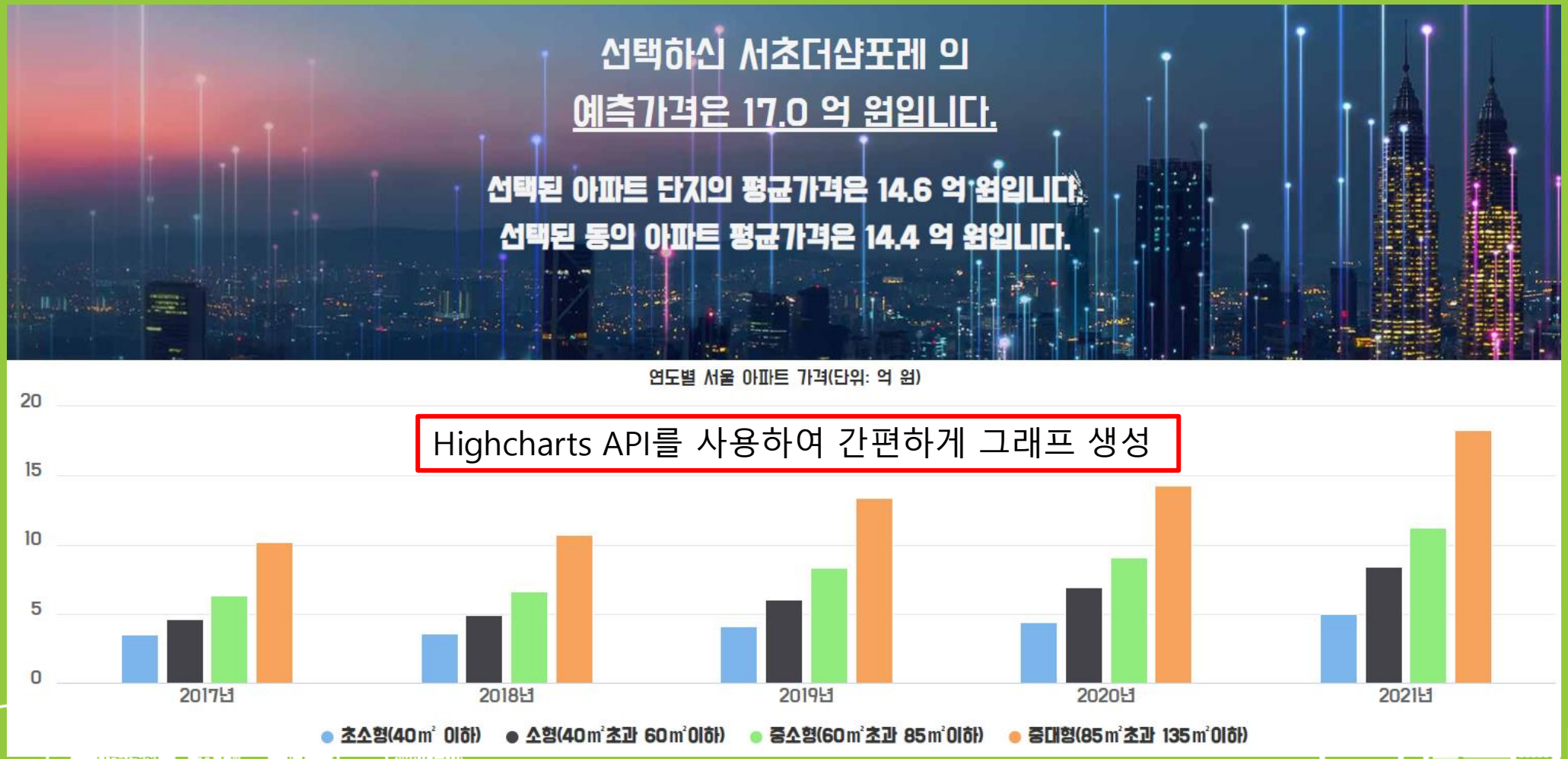
```
<div class="text-center">
    <div class="spinner-border text-success" role="status"
                                    style="width: 240px; height: 240px;">
        <span class="visually-hidden"></span>
    </div>
</div>
```

[View.py에서 계산된 값을 3초 뒤에 결과 페이지로 넘김]

```
<script language = "javascript">

  var result = '{{ predict_result }}';
  var apt = '{{ value_apt }}'; // added
  var apt_average = '{{ apt_avg }}'
  var dong_average = '{{ dong_avg }}'

  setTimeout(function(){
    location.href="/prediction/" + apt + "/" + result
  }, 5000);
</script>
```

1-2-3 . 화면 예시 및 주요 소스코드

3. 결과 페이지



7. 팀 구성 및 역할 & 후속과제

7. 후속과제

- 단독주택 / 연립주택 / 오피스텔까지 포함한 서울시 부동산 가격 예측
- 구 별로 나누어진 모델을 동 단위까지 세분화 시키고 데이터를 추가해 예측 성능 향상
- 유저 입력 창의 데이터를 현재는 기존 데이터 내 그룹화 된 형태로 제공해 지정된 값만 선택할 수 있도록 하고 있으나 추후에는 값을 유저가 직접 입력할 수 있도록 제공
- 최종 페이지에서 예측 가격만 제공하는 것이 아닌 실제 아파트 시세를 스크래핑을 통해 예측 금액과 실제 금액을 비교할 수 있도록 제공
- 유저가 입력한 구/동을 기준으로 스크래핑 된 뉴스를 같이 보여주기
- 최근 5~6년 간 구별 아파트 값의 증가추세를 수치나 색상으로 표현

감사합니다.