WASSUP AI 모델 개발자 양성과정 2기

정형데이터 분석 프로젝트

서울시 아파트 매매가 예측

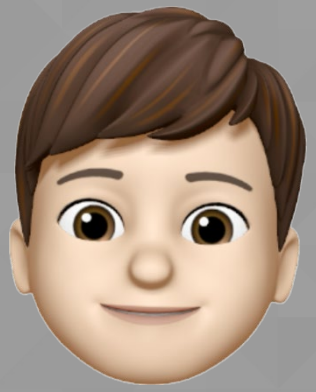
김세연
이구협
정우성
최한솔

다2조부

다2조부 팀원

팀장

이구협



팀원

김세연

정우성

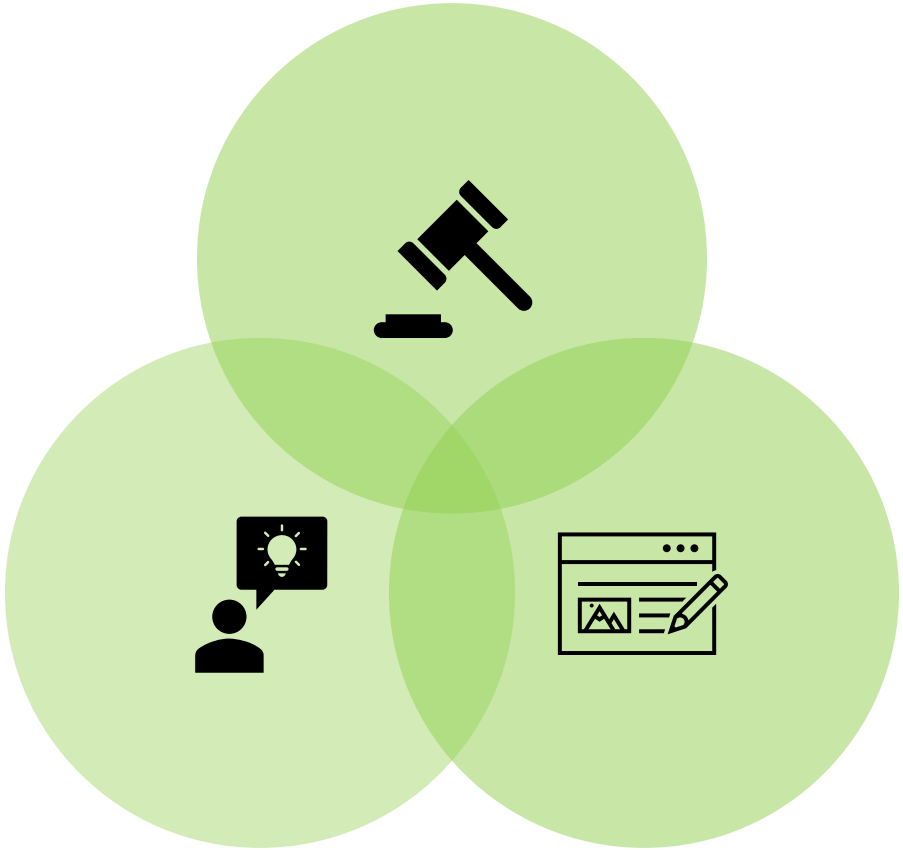
최한솔

Contents

01.주제 선정 및 데이터 준비

주제 선정

사회적 문제를 잘 다루는가?

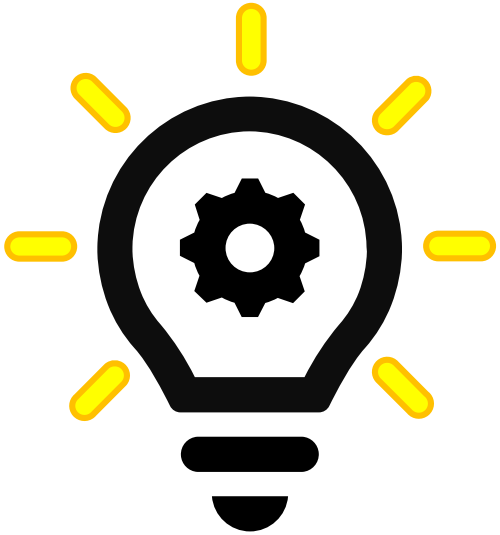


결과를 도출하기 쉬운가?

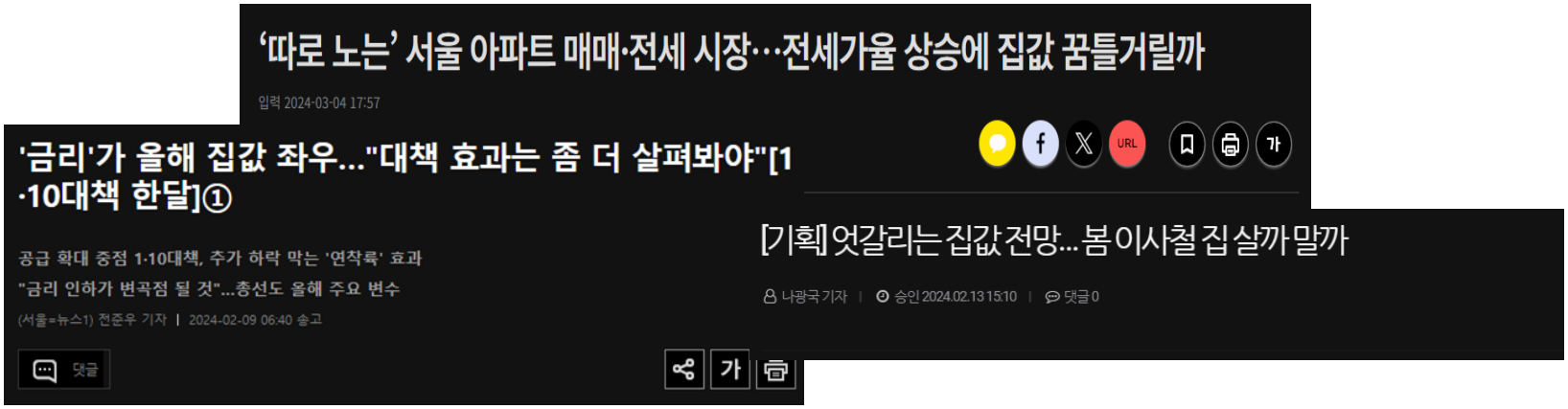
데이터는 적절한가?
데이터의 양은 충분한가?

주제 선정

부동산!



주제 선정



경기침체와 고금리 여진이 상반기까지는 이어질 것이란 우려가 팽배

↓

올해 집값 전망과 관련해 상승과 하락으로 관측이 엇갈림

↓

시세 예측이 어느 때보다 어려운 상황

↓

실수요자들은 봄 이사철 내 집 마련을 놓고 고심이 깊어짐

이러한 사회적 이슈를 바탕으로

<서울시 아파트 매매가 예측>

주제 선정

데이터 준비

- **데이터 선정**

  2020년부터 거래된 서울시 아파트로 한정

- **Test셋**

  - 2024년에 거래된 데이터

- **부동산 가격에 영향력 있는 데이터 검토**

데이터 준비

01 부동산 자체 성격

건설사
연식
위치
층수
면적

02 주변 환경

공원
버스정류소
백화점 및 마트
병원
지하철역 및 기차역
한강뷰
학교

03 외부 영향

법정동별 인구
소비자 물가지수
아파트 매매 실거래지수
월별 기준 금리

02. 데이터 전처리 및 EDA

데이터 전처리

아파트 데이터 셋 컬럼 목록

```
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   접수연도               1332702 non-null  int64
 1   자치구코드              1332702 non-null  int64
 2   자치구명               1332701 non-null  object
 3   법정동코드              1332702 non-null  int64
 4   법정동명               1332702 non-null  object
 5   지번구분               1332575 non-null  float64
 6   지번구분명              1332575 non-null  object
 7   본번                1332575 non-null  object
 8   부번                1332575 non-null  object
 9   건물명                1332702 non-null  object
 10  계약일                1332702 non-null  int64
 11  물건금액(만원)           1332702 non-null  int64
 12  건물면적(㎡)            1332702 non-null  float64
 13  토지면적(㎡)            367888 non-null   float64
 14  층                 1332702 non-null  int64
 15  권리구분               55904 non-null    object
 16  취소일                7078 non-null     float64
 17  건축년도               1332700 non-null  float64
 18  건물용도               1332702 non-null  object
 19  신고구분               55256 non-null    object
 20  신고한 개업공인중개사 시군구명  50199 non-null    object
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 181507 entries, 0 to 181506
Data columns (total 12 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   자치구명       181506 non-null  object
 1   법정동명       181507 non-null  object
 2   본번         181485 non-null  object
 3   부번         181485 non-null  object
 4   건물명        181507 non-null  object
 5   계약일        181507 non-null  int64
 6   물건금액(만원)  181507 non-null  int64
 7   건물면적(㎡)   181507 non-null  float64
 8   층          181507 non-null  int64
 9   건축년도       181505 non-null  float64
 10  년          181507 non-null  int64
 11  년월         181507 non-null  int64
dtypes: float64(2), int64(5), object(5)
memory usage: 18.0+ MB
```

결측치는 이후, 행을 전체 제거

데이터 전처리

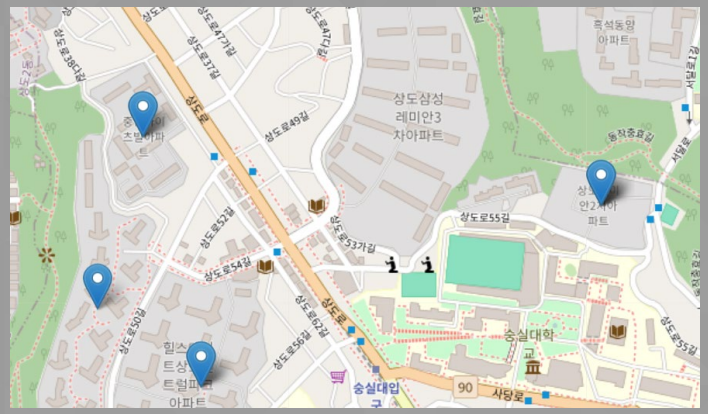
주소 데이터

본번 / 부번으로 주소 추출

→ 네이버 지도 API를 이용해 주소 Geocoding하여 위도 경도 매칭

Name	Address
블루힐하우스	서초구 잠원동 0071-0001
극동	서대문구 홍은동 0454-0000
e편한세상상도노빌리티	동작구 상도동 0903-0000
화랑대디오베이션	노원구 공릉동 0760-0000
율창	강서구 화곡동 0888-0001

```python
def transform_address(row):
    main_no = str(row['본번'])
    sub_no = (row['부번'])
    sub_no_str = '' if sub_no == 0 else f'-{sub_no}'
    return f"{row['자치구명']} {row['법정동명']} {main_no}{sub_no_str}"

df['주소'] = df.apply(transform_address, axis=1)
```



Geocoding

데이터 전처리

인구 데이터

법정동별 전체 인구수, 남녀 인구수, 나이당 인구수 존재

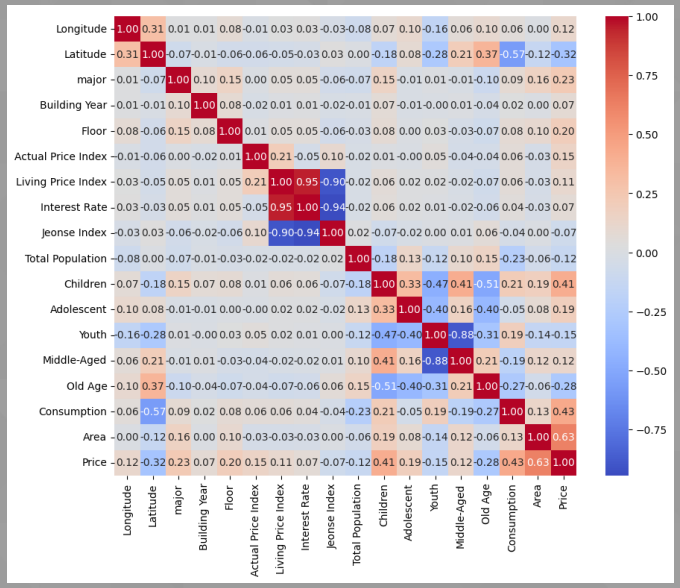
→ 어린이(0~12), 청소년(13~24), 청년(25~40), 중장년(41~65), 노년(66~109) 분류한 후 비율로 저장

```
population = Population[Population['시도명'] == '서울특별시']

population['어린이인구'] = population[[f"{age}세남자" for age in range(13)] + [f"{age}세여자" for age in range(13)]].sum(axis=1)
population['청소년인구'] = population[[f"{age}세남자" for age in range(13, 25)] + [f"{age}세여자" for age in range(13, 25)]].sum(axis=1)
population['청년인구'] = population[[f"{age}세남자" for age in range(25, 41)] + [f"{age}세여자" for age in range(25, 41)]].sum(axis=1)
population['중장년인구'] = population[[f"{age}세남자" for age in range(41, 66)] + [f"{age}세여자" for age in range(41, 66)]].sum(axis=1)
population['노년인구'] = population[[f"{age}세남자" for age in range(66, 109)] + [f"{age}세여자" for age in range(66, 109)]].sum(axis=1)
```

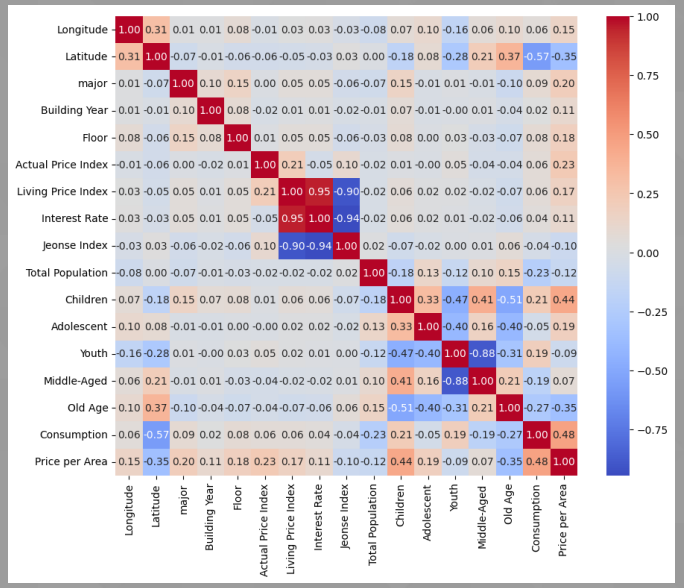
```
merged_df['Children'] = merged_df['Children']/merged_df['Total Population']
merged_df['Youth'] = merged_df['Youth']/merged_df['Total Population']
merged_df['Adolescent'] = merged_df['Adolescent']/merged_df['Total Population']
merged_df['Middle-Aged'] = merged_df['Middle-Aged']/merged_df['Total Population']
merged_df['Old Age'] = merged_df['Old Age']/merged_df['Total Population']
```

데이터 전처리

타켓변수 설정



Area(면적)의 상관관계가
다른 독립변수보다 높음



과적합 및 다른 독립변수 분석 방해를 고려하여
타겟변수를 '면적당 가격'으로 변경

데이터 전처리

```
m = folium.Map(location=[bridge.iloc[0]['Latitude'], bridge.iloc[0]['Longitude']], zoom_start=13)

for index, row in bridge.iterrows():
    folium.Marker([row['Latitude'], row['Longitude']]).add_to(m)
```

```
# 다리를 찍을 좌표를 선형보간으로 잇습니다.
interpolate_lon = np.linspace(hangang_sorted['Longitude'].min(), hangang_sorted['Longitude'].max(), 130)
linear_interp = interp1d(hangang_sorted['Longitude'], hangang_sorted['Latitude'], kind='linear')
interpolate_lat = linear_interp(interpolate_lon)

selected_coords = np.column_stack((interpolate_lon, interpolate_lat))
```

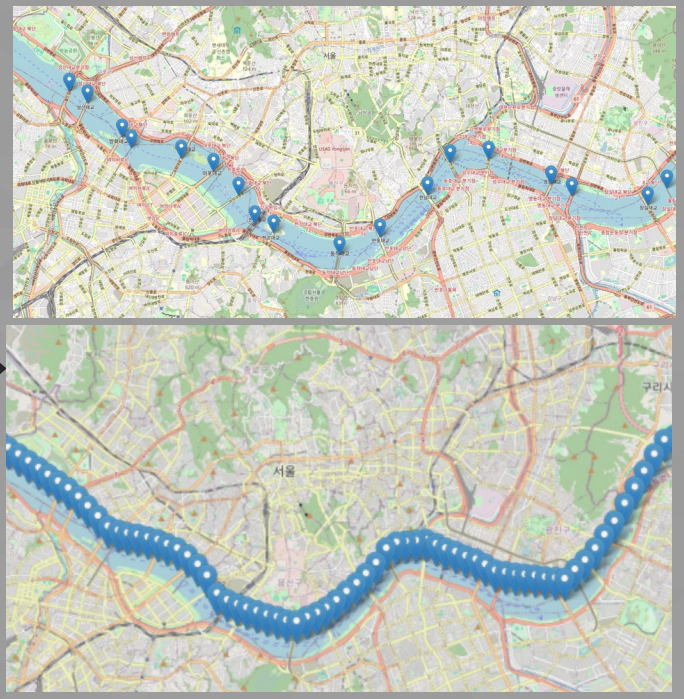
```
m = folium.Map(location=[selected_coords_df.iloc[0]['Latitude'], selected_coords_df.iloc[0]['Longitude']], zoom_start=13)

# DataFrame의 각 위치에 대한 마커 추가
for index, row in selected_coords_df.iterrows():
    folium.Marker([row['Latitude'], row['Longitude']]).add_to(m)
```

한강 데이터

서울 한강을 지나는 교량의 중심 좌표 체크 →

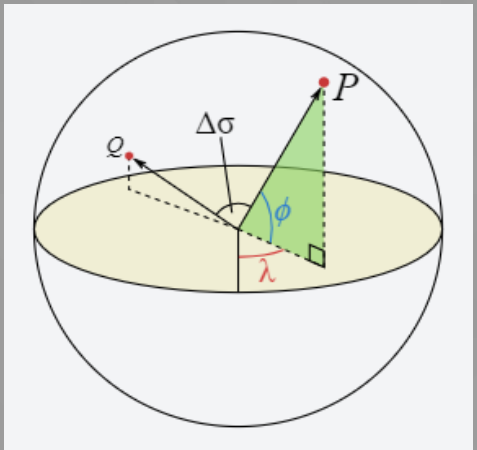
선형 보간 하여 좌표 추출

→ 위도, 경도를 이용해 한강과의 거리 추출



데이터 전처리

```python
def haversine(lat1, lon1, lat2, lon2): #Haversine 함수를 정의합니다.

    #속도를 빠르게하기 위해 데이터프레임을 직접 사용하지않고 넘파이 어레이를 사용했습니다.
    lat1, lon1, lat2, lon2 = map(np.radians, [np.array(x).astype(float) for x in [lat1, lon1, lat2, lon2]])

    dlat = lat2 - lat1
    dlon = lon2 - lon1

    a = np.sin(dlat / 2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    distance = 6371 * c # 6371 = 지구 반지름

    return distance
```



haversine formula

각 데이터 거리 구하기

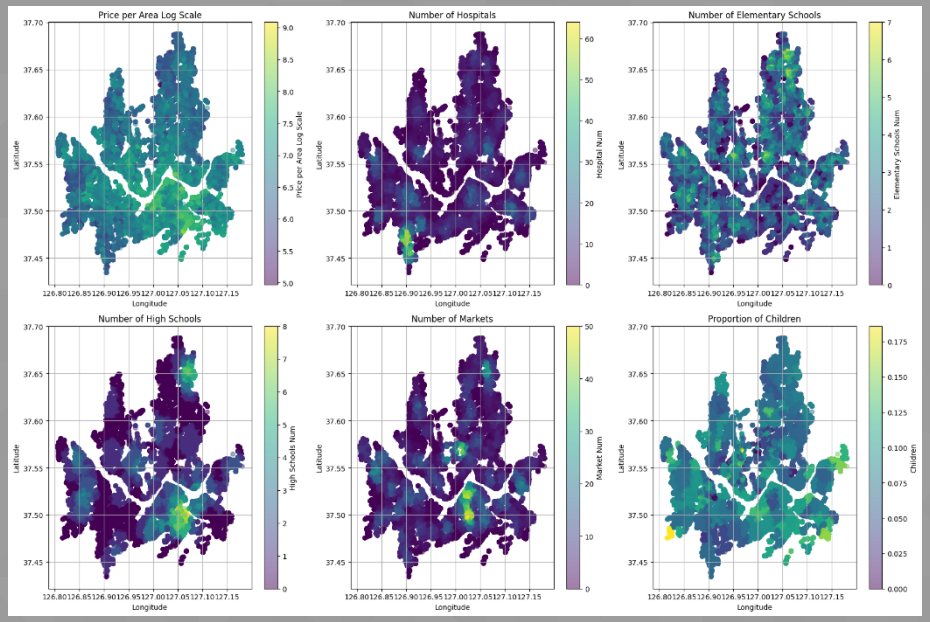
haversine 함수 사용

→ 가장 가까운 역, 종합병원의 거리 기록

→ 거리 내 초등학교, 고등학교, 상업시설, 병/의원 count

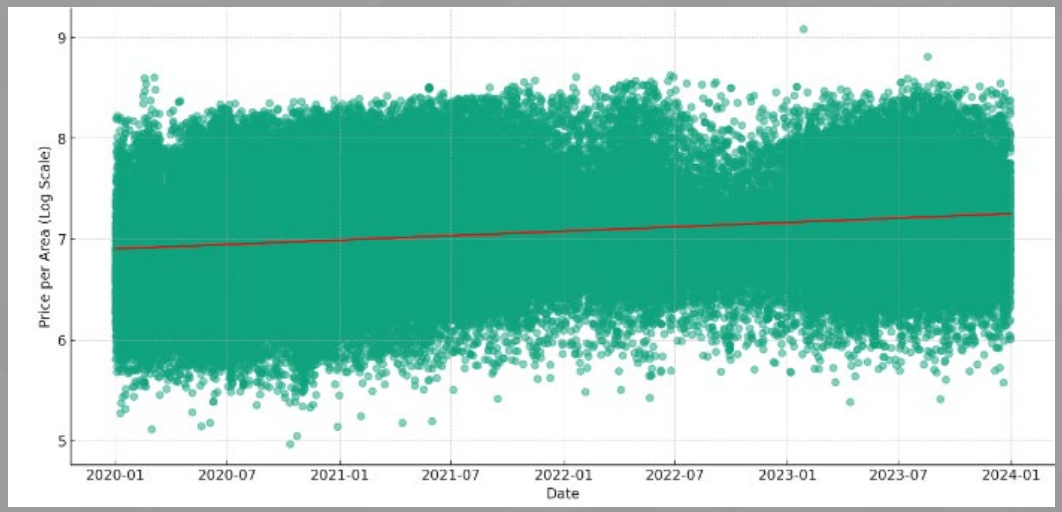
→ 거리 내 공원 및 하천이 있다면 1, 없다면 0 반환

데이터 전처리



몇가지 데이터를 가격과 비교

시간에 따른 가격 추세선 확인



EDA

이상치 확인

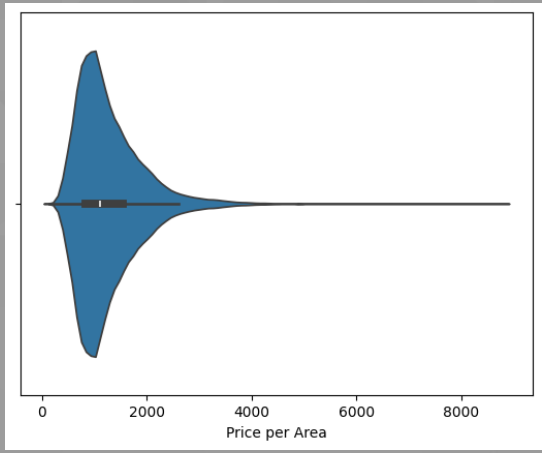
```
Q1 = df['Price per Area'].quantile(0.25)
Q3 = df['Price per Area'].quantile(0.75)
IQR = Q3 - Q1

# IQR을 이용해서 이상치를 구합니다
outliers = df['Price per Area'][((df['Price per Area'] < (Q1 - 1.5 * IQR)) |(df['Price per Area'] > (Q3 + 1.5 * IQR)))]
outliers.count()
```

```
7972
```

이상치가 존재하지만,

실제 거래를 다룬 데이터이므로

제거하지 않음.



EDA

VIF(다중공선성) 검사 실행

```
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X_const.values, i) for i in range(X_const.shape[1])]
vif["features"] = X_const.columns

vif["VIF Factor"] = vif["VIF Factor"].apply(lambda x: '{:.0f}'.format(x))
vif
```

1	1844566	Date
2	3526	Year
3	1856558	Month
10	82	Living Price Index
14	5812257	Children
15	6086071	Adolescent
16	35147531	Youth
17	9673519	Middle-Aged
18	11617455	Old Age

VIF 수치가 높은

‘Date’, ‘Year’, ‘Month’, ‘Living Price Index’ 제거

'Children', 'Adolescent', 'Youth', 'Middle-Aged', 'Old Age’ 중 목표 변수와 상관관계가 낮았던 'Adolescent', 'Middle-Aged' 제거

1	1	Longitude
2	2	Latitude
3	1	major
4	1	Building Year
5	1	Floor
6	1	Actual Price Index
7	10	Interest Rate
8	10	Jeonse Index
9	1	Total Population
10	5	Children
11	4	Youth
12	4	Old Age
13	2	Consumption
14	1	Distance to MC
15	1	Distance to NS
16	1	Elementary Schools Num
17	2	High Schools Num
18	2	Market Num
19	1	Hospital Num
20	1	Park Presence
21	1	Nearby Hangang
22	1	Bus Station
23	1	Gu_encoding

제거 후 VIF 수치가 모두 10 이하로 하향

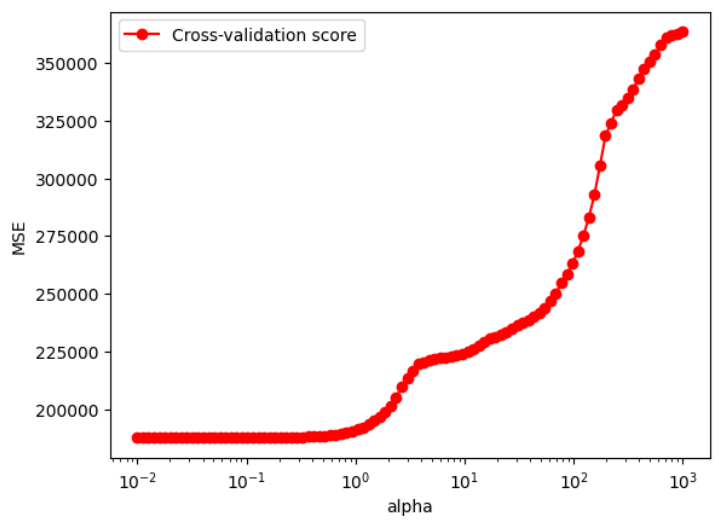
EDA

P-value, 라쏘 회귀

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3.197e+04	1657.951	-19.281	0.000	-3.52e+04	-2.87e+04
Longitude	964.7695	13.398	72.011	0.000	938.511	991.028
Latitude	-2443.7638	30.167	-81.009	0.000	-2502.890	-2384.638
major	67.6013	1.313	51.498	0.000	65.028	70.174
Building Year	0.1872	0.005	35.304	0.000	0.177	0.198
Floor	9.1966	0.179	51.342	0.000	8.846	9.548
Actual Price Index	7.9420	0.067	118.035	0.000	7.810	8.074
Interest Rate	23.2319	2.783	8.349	0.000	17.778	28.686
Jeonse Index	-6.0192	1.731	-3.478	0.001	-9.411	-2.627
Total Population	0.0002	2.17e-05	9.558	0.000	0.000	0.000
Children	4513.0277	103.101	43.773	0.000	4310.952	4715.103
Youth	-860.2029	40.339	-21.324	0.000	-939.266	-781.140
Old Age	-1498.6965	67.977	-22.047	0.000	-1631.930	-1365.462
Consumption	197.5609	2.017	97.925	0.000	193.607	201.515
Distance to MC	15.8899	1.572	10.108	0.000	12.809	18.971
Distance to NS	-152.9641	2.579	-59.316	0.000	-158.018	-147.910
Elementary Schools Num	25.9397	0.993	26.136	0.000	23.994	27.885
High Schools Num	48.0707	1.036	46.404	0.000	46.040	50.101
Market Num	5.9471	0.200	29.735	0.000	5.555	6.339
Hospital Num	-13.2151	0.204	-64.813	0.000	-13.615	-12.815
Park Presence	54.1592	8.430	6.425	0.000	37.637	70.681
Nearby Hangang	385.6604	5.383	71.642	0.000	375.110	396.211
Bus Station	-1.4570	0.264	-5.512	0.000	-1.975	-0.939
Gu_encoding	5.9191	0.178	33.212	0.000	5.570	6.268

P-value는 전부 0에 근접한 수치

```
# validation_curve 함수를 사용하여 검증 곡선을 계산합니다.
train_scores, test_scores = validation_curve(
    Lasso(), X_train, y_train, param_name="alpha", param_range=param_range,
    scoring="neg_mean_squared_error", cv=5)
```




```
alpha = 1  # 위의 그래프를 보고 값을 지정합니다.
lasso_model = Lasso(alpha=alpha)
lasso_model.fit(X_train, y_train)

print(f'Selected features: {np.where(lasso_model.coef_ != 0)[0]}')
```

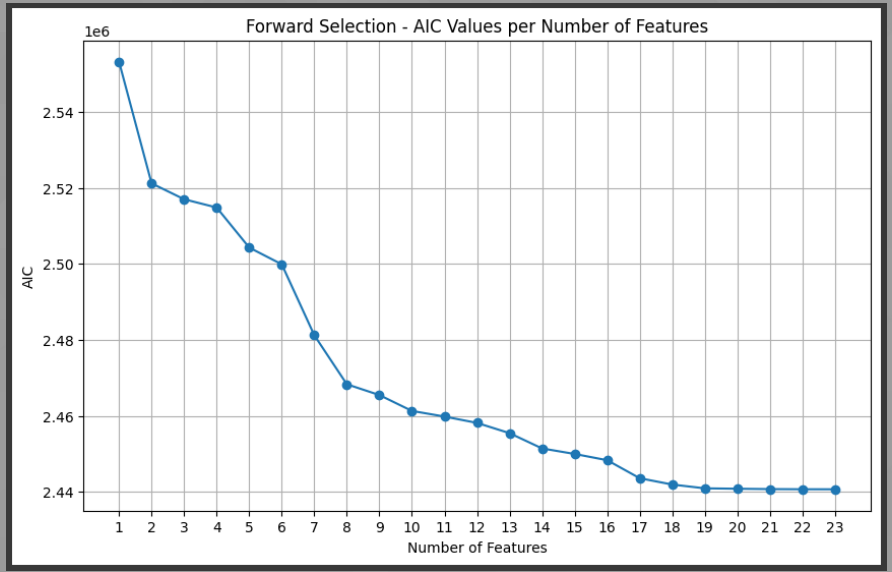
```
Selected features: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22]
```

EDA

전진선택법

```
def forward_selection(data, target, significance_level=0.05):
    initial_features = data.columns.tolist()
    best_features = []
    aic_values = []

    while len(initial_features) > 0:
        remaining_features = list(set(initial_features) - set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.add_constant(data[best_features + [new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if min_p_value < significance_level:
            best_feature = new_pval.idxmin()
            best_features.append(best_feature)
            aic_values.append(sm.OLS(target, sm.add_constant(data[best_features])).fit().aic)
        else:
            break

    return best_features, aic_values
```

```
['major',
 'Children',
 'Old Age',
 'Bus Station',
 'Actual Price Index',
 'Longitude',
 'Latitude',
 'Consumption',
 'Youth',
 'Nearby Hangang',
 'Interest Rate',
 'Building Year',
 'Floor',
 'Hospital Num',
 'Elementary Schools Num',
 'Market Num',
 'Distance to NS',
 'High Schools Num',
 'Gu_encoding',
 'Distance to MC',
 'Total Population',
 'Park Presence',
 'Jeonse Index']
```



EDA

후진소거법

```python
#후진소거법
def backward_elimination(data, target, significance_level = 0.05):
    features = data.columns.tolist()
    while len(features) > 0:
        features_with_constant = sm.add_constant(data[features])
        p_values = sm.OLS(target, features_with_constant).fit().pvalues[1:]
        max_p_value = p_values.max()
        if max_p_value >= significance_level:
            excluded_feature = p_values.idxmax()
            features.remove(excluded_feature)
        else:
            break
    return features
```

```
['Longitude',
 'Latitude',
 'major',
 'Building Year',
 'Floor',
 'Actual Price Index',
 'Interest Rate',
 'Jeonse Index',
 'Total Population',
 'Children',
 'Youth',
 'Old Age',
 'Consumption',
 'Distance to MC',
 'Distance to NS',
 'Elementary Schools Num',
 'High Schools Num',
 'Market Num',
 'Hospital Num',
 'Park Presence',
 'Nearby Hangang',
 'Bus Station',
 'Gu_encoding']
```

교차선택법

```python
#교차선택법
def stepwise_selection(data, target, SL_in=0.05, SL_out=0.05):
    initial_features = data.columns.tolist()
    best_features = []
    while len(initial_features) > 0:
        changed=False
        # 전진 선택
        remaining_features = list(set(initial_features) - set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.add_constant(data[best_features + [new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if min_p_value < SL_in:
            best_features.append(new_pval.idxmin())
            changed=True

        # 후진 소거
        model = sm.OLS(target, sm.add_constant(data[best_features])).fit()
        p_values = model.pvalues.iloc[1:]
        max_p_value = p_values.max()
        if max_p_value > SL_out:
            changed=True
            worst_feature = p_values.idxmax()
            best_features.remove(worst_feature)

        if not changed:
            break

    return best_features
```

```
['major',
 'Children',
 'Old Age',
 'Actual Price Index',
 'Longitude',
 'Latitude',
 'Consumption',
 'Youth',
 'Nearby Hangang',
 'Interest Rate',
 'Building Year',
 'Floor',
 'Hospital Num',
 'Elementary Schools Num',
 'Market Num',
 'Distance to NS',
 'High Schools Num',
 'Gu_encoding',
 'Total Population',
 'Distance to MC',
 'Park Presence',
 'Bus Station',
 'Jeonse Index']
```

03. 모델링 및 모델 선정

개발 환경

모델링

사용 모델

```
models = {
    "KNN": KNeighborsRegressor(),
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(),
    "XGB": XGBRegressor(),
    "CatBoost": CatBoostRegressor(verbose=0),
    "Gradient Boosting": GradientBoostingRegressor(),
    "Decision Tree": DecisionTreeRegressor(),
    "LightGBM": LGBMRegressor()
}
```

모델링

사용한 데이터 목록

```
X = df[['Longitude',
        'Latitude', 'major', 'Building Year', 'Floor', 'Interest Rate', 'Jeonse Index', 'Total Population',
        'Children', 'Youth', 'Old Age',
        'Consumption', 'Distance to MC', 'Distance to NS', 'Elementary Schools Num',
        'High Schools Num', 'Market Num', 'Hospital Num', 'Park Presence',
        'Nearby Hangang', 'Bus Station', 'Gu_encoding']]
y = df['Price per Area']
```

위도, 경도, 상위 10개 건설사 여부, 연식, 층수, 기준금리, 매매가 대비 전세가,
법정동 전체 인구, 법정동 어린이, 청년, 노년 비율,
자치구별 연간 소비액, 가장 가까운 종합병원과의 거리, 가장 가까운 지하철역과의 거리, 0.7km내 초등학교 개수,
2km내 명문 일반고 개수, 1km내 일반 병의원과 상업시설의 개수, 0.8km내 공원 및 하천의 존재 여부,
한강변에서 0.4km이내에 있는지 여부, 0.5km내 버스 정류장 수의 절반, 라벨 인코딩 된 자치구

타겟변수 : 1 제곱미터당 가격

모델링

학습 데이터 분할

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

Test Set이 별도로 존재하므로 (2024년 데이터), 전체 데이터를 Train셋과 Validation셋으로 분리함

스케일링

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

스케일링이 필요한 모델만
스케일링을 적용하였음

```
results = {}
for name, model in models.items():
    if name in ["KNN", "Linear Regression"]:
        model.fit(X_train_scaled, y_train)
        predictions = model.predict(X_val_scaled)
    else:
        model.fit(X_train, y_train)
        predictions = model.predict(X_val)

    r2 = r2_score(y_val, predictions)
    mae = mean_absolute_error(y_val, predictions)
    mse = mean_squared_error(y_val, predictions)
    rmse = np.sqrt(mse)
```

모델링

머신러닝 모델

```
results = {}
for name, model in models.items():
    if name in ["KNN", "Linear Regression"]:
        model.fit(X_train_scaled, y_train)
        predictions = model.predict(X_val_scaled)
    else:
        model.fit(X_train, y_train)
        predictions = model.predict(X_val)

    r2 = r2_score(y_val, predictions)
    mae = mean_absolute_error(y_val, predictions)
    mse = mean_squared_error(y_val, predictions)
    rmse = np.sqrt(mse)

    results[name] = {"R2": r2, "MAE": mae, "RMSE": rmse}

for name, metrics in results.items():
    print(f"{name} - R² : {metrics['R2']:.4f}, MAE: {metrics['MAE']:.4f}, RMSE: {metrics['RMSE']:.4f}")
```

딥러닝 모델

```
regressor = TabNetRegressor()

X_train_np = X_train.values
X_val_np = X_val.values
y_train_np = y_train.values.reshape(-1, 1)
y_val_np = y_val.values.reshape(-1, 1)

regressor.fit(
  X_train_np, y_train_np,
  eval_set=[(X_val_np, y_val_np)],
  eval_name=['test'],
  eval_metric=['rmse'],
  max_epochs=200,
  patience=10,
  batch_size=2048,
  virtual_batch_size=1028
)
```

```
mlp = MLPRegressor(hidden_layer_sizes=(100,50,50), activation='relu', max_iter=200,
                   alpha=0.01, solver='adam', verbose=10,
                   random_state=42, tol=0.0000001)
mlp.fit(X_train, y_train)

predictions = mlp.predict(X_val)

r2 = r2_score(y_val, predictions)
n = X_val.shape[0]  # 테스트 샘플의 수
p = X_val.shape[1]  # 변수의 수
adjusted_r2 = 1 - (1-r2) * (n - 1) / (n - p - 1)
mse = mean_squared_error(y_val, predictions)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_val, predictions)

# 성능 지표 출력
print(f'Adjusted R^2: {adjusted_r2:.4f}')
print(f'MSE: {mse:.4f}')
print(f'RMSE: {rmse:.4f}')
print(f'MAE: {mae:.4f}')
```

모델링

모델	R-Squared	MAE	RMSE
KNN Regressor	0.8765	146.5663	217.2874
Linear Regression	0.4727	335.8102	448.9306
Random Forest Regressor	0.9278	105.0667	166.1151
XGBoost Regressor	0.8962	143.6255	199.1470
CatBoost Regressor	0.8971	143.5425	198.3046
Gradient Boosting Regressor	0.7429	227.5855	313.4634
Decision Tree Regressor	0.8818	136.3858	212.5834
LightGBM Regressor	0.8548	173.6649	235.6139
TabNet Regressor	-	-	248.3215
MLP Regressor	0.6198	281.9763	381.0954

상위 5개 모델 선정
KNN, RF, XGB, CatBoost, DT

모델링

```python
def objective(trial):
    # 탐색할 하이퍼파라미터 설정
    param = {
        'iterations': trial.suggest_int('iterations', 50, 300),
        'depth': trial.suggest_int('depth', 4, 10),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3),
        'random_strength': trial.suggest_int('random_strength', 1, 10),
        'bagging_temperature': trial.suggest_float('bagging_temperature', 0, 1),
        'border_count': trial.suggest_int('border_count', 1, 255),
        'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 1e-2, 10, log=True),
        'loss_function': 'RMSE',
        'verbose': 1
    }
```

```python
# Optuna 최적화
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=150)
```

Optuna 라이브러리 이용

하이퍼파라미터 최적화

그리드 서치나, 랜덤 서치를 사용 안한 이유:

로컬 PC에서 돌리기에는 데이터셋이 너무 많아 시간이 오래걸림.

```python
models = {
    "KNN": KNeighborsRegressor(n_neighbors=5, p=1, n_jobs=-1),
    "Random Forest": RandomForestRegressor(n_estimators=200, random_state=42, n_jobs=-1),
    "XGB": XGBRegressor(n_estimators=895, max_depth=10, learning_rate=0.11341, min_child_weight=2,
                        gamma=1.204458,subsample=0.9588896, colsample_bytree=0.8822700, reg_alpha=1.6237099, reg_lambda=0.0025671, random_state=42),
    "CatBoost": CatBoostRegressor(iterations=299, depth=10, learning_rate=0.2953026, random_strength=7,
                                  bagging_temperature=0.02308666, border_count=130, l2_leaf_reg=0.042969, random_state=42),
    "Decision Tree": DecisionTreeRegressor(random_state=42)
}
```

KNN, RF, DT는 수동으로 진행

모델링

```python
base_models = [
    ('Random Forest', RandomForestRegressor(n_estimators=200, random_state=42, n_jobs=-1)),
    ('CatBoost', CatBoostRegressor(iterations=299, depth=10, learning_rate=0.2953026, random_strength=7, bagging_temperature=0.02308666, border_count=130, l2_leaf_reg=0.042969, random_state=42)),
    ('KNN', knn_pipeline),
    ('Decision Tree', DecisionTreeRegressor(random_state=42))
]

meta_model = XGBRegressor(n_estimators=563, max_depth=3, learning_rate=0.0151, min_child_weight=1, subsample=0.8967, colsample_bytree=0.9831, reg_alpha=3.029,reg_lambda=0.9275, random_state=42)

stacking_model = StackingRegressor(estimators=base_models, final_estimator=meta_model)

stacking_model.fit(X_train, y_train)
```

```python
base_models = [
    ('Random Forest', RandomForestRegressor(n_estimators=200, random_state=42, n_jobs=-1)),
    ('KNN', knn_pipeline),
    ('XGB', XGBRegressor(n_estimators=895, max_depth=10, learning_rate=0.11341, min_child_weight=2, gamma=1.204458 ,subsample=0.9588896, colsample_bytree=0.8822700, reg_alpha=1.6237099, reg_lambda=0.0025671, random_state=42)),
    ('Decision Tree', DecisionTreeRegressor(random_state=42))
]

meta_model = CatBoostRegressor(iterations=299, depth=10, learning_rate=0.2953026, random_strength=7, bagging_temperature=0.02308666, border_count=130, l2_leaf_reg=0.042969, random_state=42)
```

Stacking Ensemble을 사용하여, 선정된 5가지 모델중에 한가지를 메타모델로 설정하고, 나머지를 기본모델로 설정하였음.

메타모델도 마찬가지로 Optuna로 최적화 하였음.

모델 선정

모델	Adjusted R^2	MAE	RMSE
KNN Regressor	0.8885	139.7465	206.4142
Random Forest Regressor	0.9282	104.7726	165.6209
XGBoost Regressor	0.9267	107.9752	167.3737
CatBoost Regressor	0.9193	122.9984	175.5749
Decision Tree Regressor	0.8849	125.8278	209.7002
Stacking Model (Meta: XGB)	0.9311	106.5422	162.2298
Stacking Model (Meta: RF)	0.9272	110.1101	166.7844
Stacking Model (Meta: CatBoost)	0.9256	106.9464	168.5928

하얀색으로 표시된 4가지 모델을 선택하였음.

04. 최종 결과 및 고찰

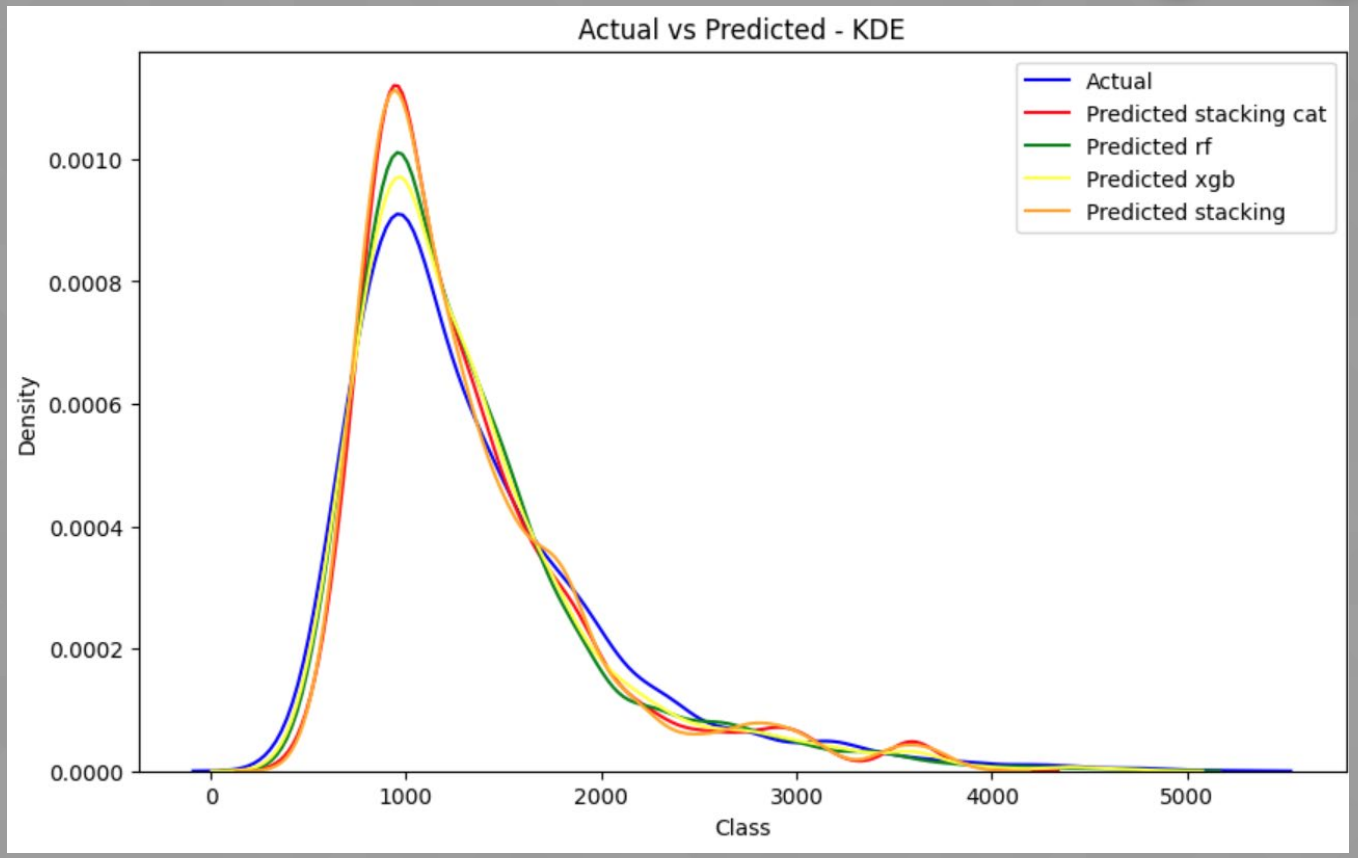
최종 결과

테스트 결과

모델	Adjusted R^2	MAE	RMSE
Random Forest Regressor	0.9004	138.6914	212.5829
XGBoost Regressor	0.9158	135.3869	195.4482
Stacking Model (Meta: XGB)	0.9057	137.5756	206.9193
Stacking Model (Meta: CatBoost)	0.8808	153.3266	232.5355

모델	MAE(검증)	MAE(실제)
Random Forest Regressor	104.7726	138.6914
XGBoost Regressor	107.9752	135.3869
Stacking Model (Meta: XGB)	106.5422	137.5756
Stacking Model (Meta: CatBoost)	106.9464	153.3266

최종 결과



모델의 예측 값과 실제 값의 비교

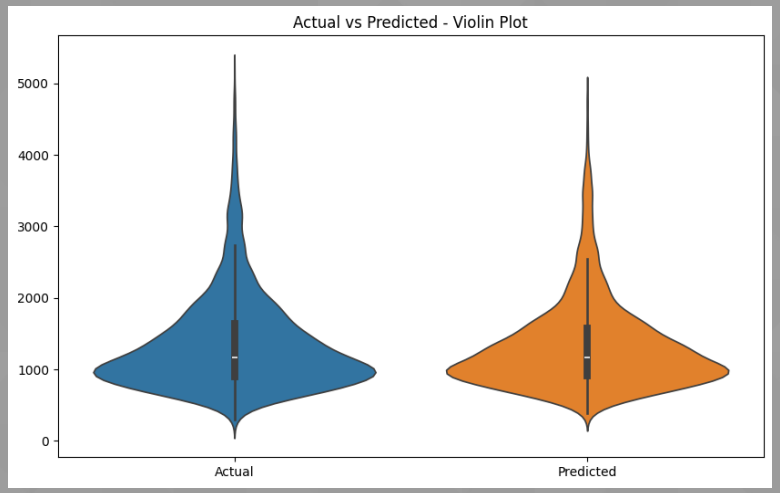
최종 결과

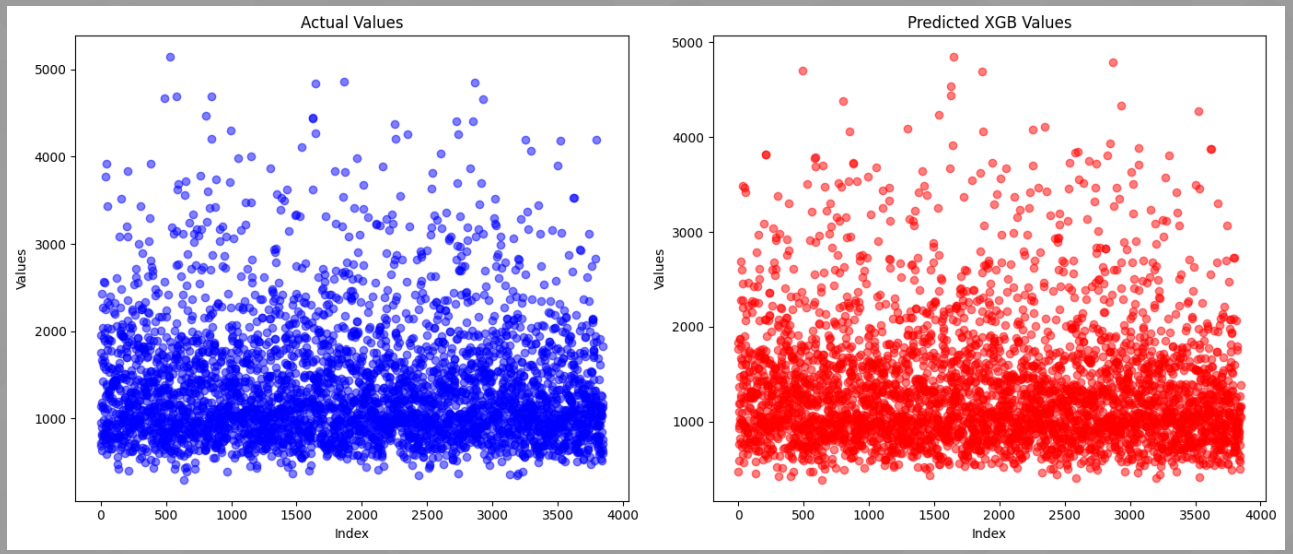
최종 모델 : XGBoost

Test셋에 모델을 적용한 결과

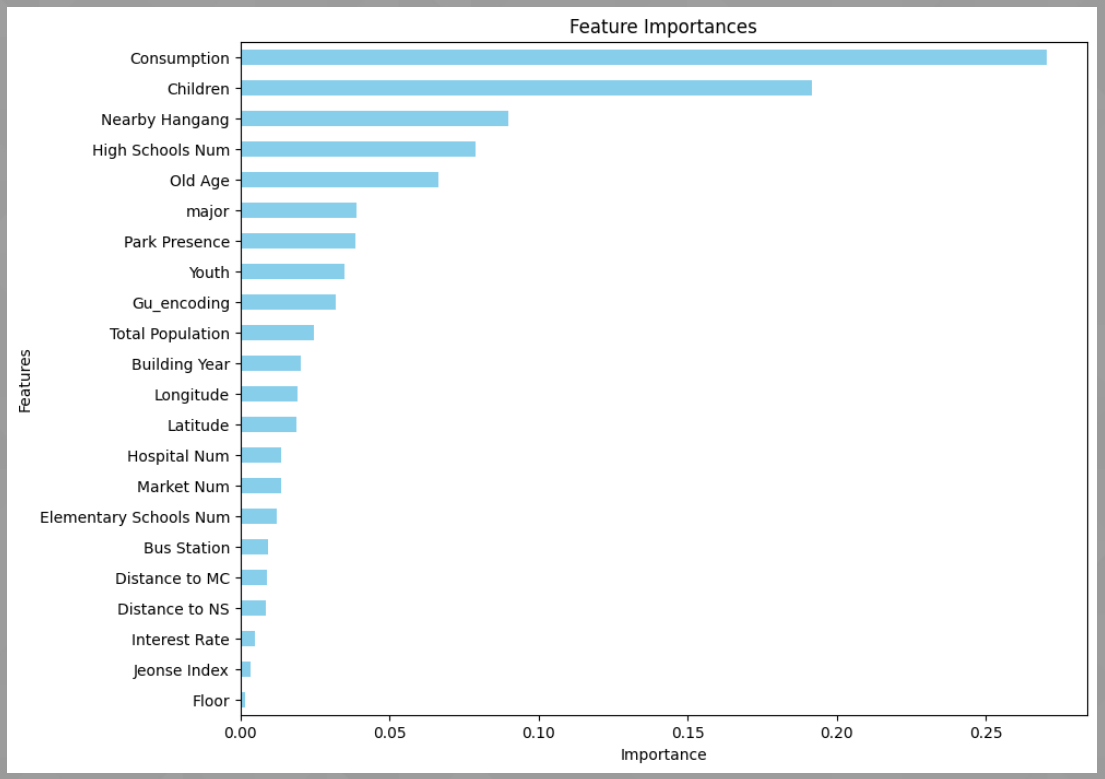
KDE 그래프 상 XGBoost 모델이 최빈값에 가장 근접,

설명을 가장 잘 한다고 판단





결과 해석



XGB의 Feature Importances

결과 해석

결과 해석

사용 지표

사용한 지표들이 대체적으로 집값에 영향을 미치는 것으로 보임
시간에 따른 추세는 큰 차이 없음

변수 중요도

소비자 물가 지수 > 어린이 비율 > 한강뷰 > 주위 명문고 순으로 모델이 평가

변수

변수 선택 과정에서 변수가 줄지 않았기에 더 다양한 변수를 포함한다면 더 좋은 결과를 만들 가능성이 있다고 판단됨

변수 중요도

소비가 높고 소득이 많은 지역이 가중치가 높고 그 다음 어린이 인구인 걸 토대로 어린이가 많은 젊은 지역의 집값이 높다고 판단 가능

예측과 실제

2020-2023 데이터로 판단한 예측 가격보다 실제 가격이 낮은 것은 평균 집값이 하락했다고 판단할 수 있음

고찰 및 개선점

고찰 및 개선점

아직 24년 2월 실질 매매가 지수 데이터가 없어서 테스트셋에 대한 특징으로 사용 불가능

아파트 단지 하나를 좌표 하나로 변환한 것에 데이터 왜곡이 존재할 수 있음

병/의원 및 시장, 공원을 설정하는 기준이 다소 애매

주변 지리에 따라 공원 및 한강 직선 거리 데이터에 오차가 존재할 수 있음

당시 부동산 정책 및 사회적 이슈를 데이터화 하기 어려움

공공기관 데이터에 많은 이상치 존재

아쉬움

데이터가 많아 오랜 시간 소요되고,
실행 환경에 많은 영향을 받아 데이터를 축소함

좋은 컴퓨터 성능과 시간적 여유가 있었다면 다양한 데이터와 특징을 가지고 더 의미 있는 결과를 도출할 수 있었을 아쉬움

하이퍼파라미터 튜닝을 더 많이 하지 못 한 점

복잡한 딥러닝 모델을 사용해보지 못 한 점

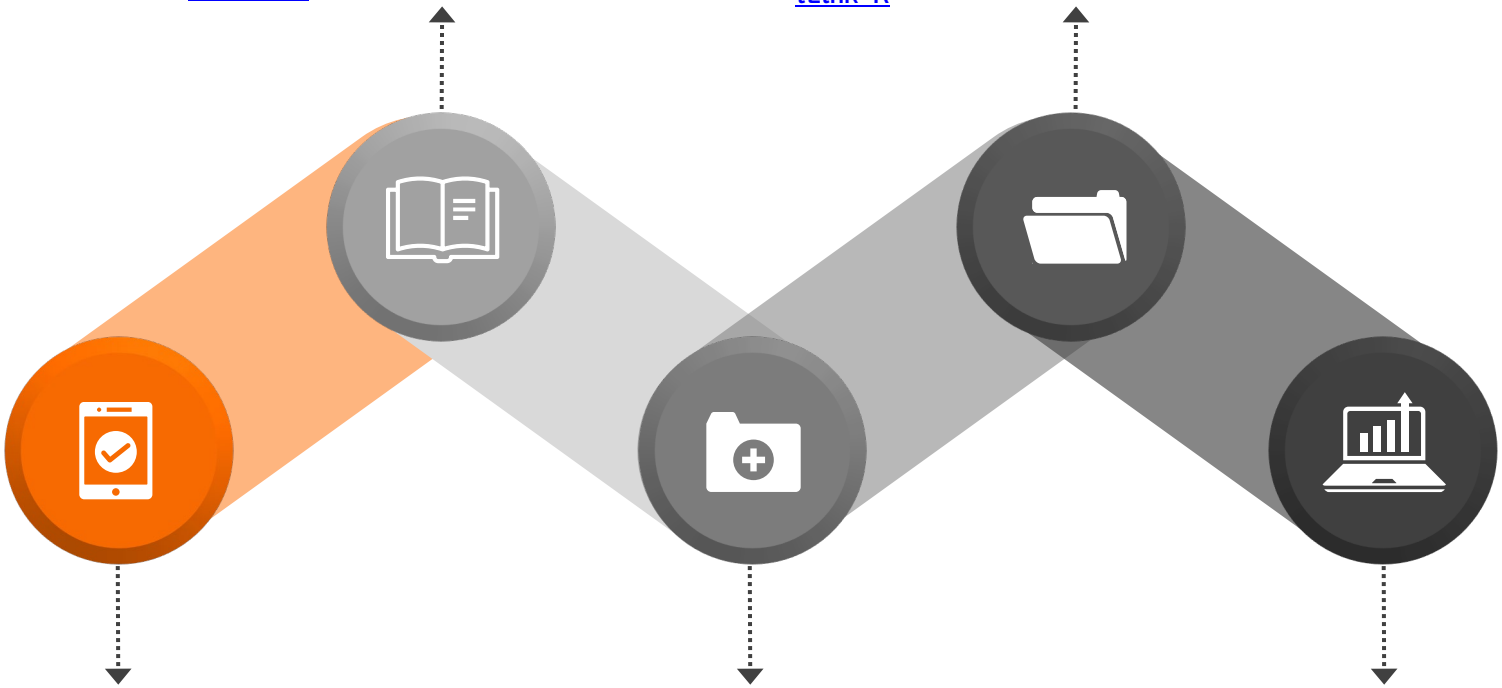
구를 수치형으로 변환할 때 라벨 인코딩을 사용하였으나 모델이 순서관계가 있다고 잘못 해석할 여지가 있음

05. 참고 문헌

참고 문헌

딥러닝과 머신러닝을 이용한 아파트 실거래가 예측 논문
https://m.riss.kr/search/detail/DetailView.do?p_mat_type=1a0202e37d52c72d&control_no=416ccb17bd7974f0b7998d826d417196

딥러닝을 이용한 주택가격 예측모형 연구: 서울시 주택 매매 실거래가를 중심으로
https://www.riss.kr/search/detail/DetailView.do?p_mat_type=be54d9b8bc7cdb09&control_no=87210321df907a06ffe0bdc3ef48d419&outLink=K



변수선택법
https://blog.naver.com/962300/2222928074

다중공선성과 VIF
https://bkshin.tistory.com/entry/DATA-20-%EB%8B%A4%EC%A4%91%EA%B3%B5%EC%84%A0%EC%84%B1%EA%B3%BC-VIF

Optuna
https://github.com/optuna/optuna

참고 문헌

데이터 출처

KOSIS 국가 통계 포털

소비자 물가 지수

매매가 대비 전세가 비율

아파트 실거래 매매가 지수

법정동별 인구수 통계

서울 열린 데이터 광장

서울시 아파트 부동산 실거래가

학교 정보

시설물 정보

병원 정보

지하철역 정보

버스정류소 정보

지표 누리

기준 금리 추이

네이버 지도 API

한강 교량의 위치 데이터

아파트의 위도 및 경도

여기서 잠깐!

프로그램 시연이 있겠습니다

THANK YOU