House Price Prediction Website with Bengaluru House price data

Personal Project

Annie Yin

Dataset Overview

Dataset: Kaggle's Bengaluru House Price Dataset

About the dataset

1. Location-Based Features with Neighborhood Information: Proximity to essential amenities like offices, schools, parks, restaurants, and hospitals.

2. Important Property Attributes: Total square footage, number of bedrooms (BHK), and bathrooms.

3. Availability of Units by Budget Segment: Detailed segmentation of housing units for sale across price categories, illustrating housing options within budget limits.

Dataset Challenges: High dimensionality, missing values, and outliers

area_type	availability	location	size	society	total_sqft	bath	balcony	price
Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00

Data Cleaning & Feature Engineering

1. Dimension reduction for location feature

```
location_stats_less_than_10 = location_stats[location_stats<=10]
location_stats_less_than_10
```

2. Outliers removal

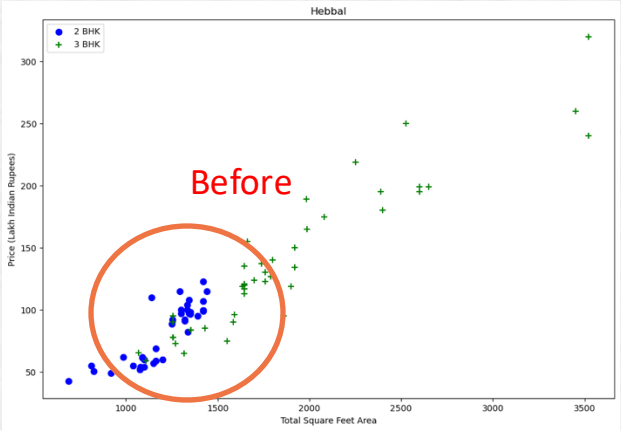
```
def remove_pps_outliers(df):
    df_out = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-st)) & (subdf.price_per_sqft<=(m+st))]
        df_out = pd.concat([df_out, reduced_df],ignore_index=True)
    return df_out
```

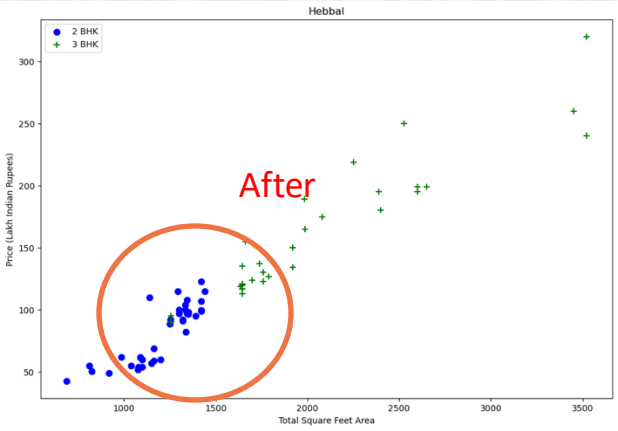
Data Cleaning & Feature Engineering (cont.)

3. **Handling the noisy of price and bhk**: As some price of 3 bhk are lower than that of 2 bhk

```python
def remove_bhk_outliers(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_df.price_per_sqft),
                'std': np.std(bhk_df.price_per_sqft),
                'count': bhk_df.shape[0]
            }

        for bhk, bhk_df in location_df.groupby('bhk'):
            stats = bhk_stats.get(bhk-1)
            if stats and stats['count'] > 5:
                exclude_indices = np.append(exclude_indices, bhk_df[bhk_df.price_per_sqft<(stats['mean'])].index.values)
    return df.drop(exclude_indices,axis='index')
```





Model Building

- Test Linear Regression, Lasso Regression, and Decision Tree Regressor
- Use GridSearchCV to find the best fit model

```
def find_best_model_gridsearchcv(X, y):
    algos = {
        'linear_regression' : {
            'model': LinearRegression(),
            'params': {
                'fit_intercept': [True, False],
                'copy_X': [True, False]
            }
    },
    'lasso': {
        'model': Lasso(),
        'params': {
            'alpha': [1,2],
            'selection': ['random', 'cyclic']
        }
    },
    'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {
                'criterion' : ['mse','friedman_mse'],
                'splitter': ['best','random']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
    for algo_name, config in algos.items():
        gs =  GridSearchCV(config['model'], config['params'], cv=cv, return_train_score=False)
        gs.fit(X,y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })
    return pd.DataFrame(scores,columns=['model','best_score','best_params'])
```

Flask API Structure

Designed RESTful API using Flask with two endpoints:

1. **/get_location_names**: Returns available location options.

2. **/predict_home_price**: Receives inputs and provides the predicted house price.

```python
@app.route('/get_location_names')
def get_location_names():
    response = jsonify({
        'locations': util.get_location_names()
    })
    response.headers.add('Access-Control-Allow-Origin', '*')
    return response

@app.route('/predict_home_price', methods=['GET','POST'])
def predict_home_price():
    total_sqft = float(request.form['total_sqft'])
    location = request.form['location']
    bhk = int(request.form['bhk'])
    bath = int(request.form['bath'])
```

util.py Module

Functions:

1. get_estimated_price: Fetches location index, applies one-hot encoding, and returns a prediction from the trained model.

2. load_saved_artifacts: Loads pre-trained model and necessary data columns for prediction.

```python
def get_estimated_price(location, sqft, bhk, bath):
    try:
        loc_index = __data_columns.index(location.lower())
    except:
        loc_index = -1  # 若地點不存在 columns 中，設為 -1 表示無效。且後續的 if loc_index >= 0 才會執行

    x = np.zeros(len(__data_columns))  # 建立一個與 data columns 長度相同的零向量
    x[0] = sqft
    x[1] = bath
    x[2] = bhk
    if loc_index >= 0:
        x[loc_index] = 1  # 代表 one hot encoding -> 找到對應位置的值設為 1

    return round(__model.predict([x])[0],2)


def load_saved_artifacts():
    print("loading saved artifacts...start")
    global __data_columns
    global __locations

    with open("artifacts/columns.json", "r") as f:
        __data_columns = json.load(f)['data_columns']
        __locations = __data_columns[3:]
```

UI/UX Design Considerations

Frontend included: HTML, CSS, JavaScript, jQuery
UI Components:

1. **Interactive Input Options**

2. **Dropdown Menu for location:** The **uiLocations** select element dynamically loads a list of available locations.

3. **Java Integration of backend:**
- **onPageLoad** fetches location names
- **onClickedEstimatePrice** triggers a price estimate request.
- **onClickedEstimatePrice()** for real-time predictions.

4. **Objective**: Created a scalable framework ready for additional features and enhanced user experience

Prediction Website

Draw-down menu

Predicted price will be shown here