Real Estate Price Prediction

1. 21820103 Sumit Dube
2. 17u520 Sneha Waghmode
3. U1510600 Sagar Maske
4. 17u111 Suyog Salpure
5. 17u488 Vyankatesh Munde

**Objectives:**

* Data cleaning and processing.
* Feature engineering, outlier removal and dimensionally reduction.
* Model building and comparing various different models for best results.
* Reducing the error in the Model.

**Data Set Used:**

* Dataset is taken from kaggle.com
* It contain nine columns.
* The columns are area\_type, availability, location, size, society, total\_sqft, bath, balcony and price respectively

**Steps:**

* Data Load-
* Load banglore home prices into dataframe
* Drop-
* Drop features that are not required to build our model.
* Data Cleaning-
* Handle NA values.
* Feature Engineering-
* Add new feature for bhk.
* Add new feature called price per square feet.
* Dimensionally Reduction-
* Any location having less than 10 data points should be tagged other location. this way number of catagories can be reduced by huge amount .Later on when we do one hot encoding,it will help us with having fewer dummy columns.
* Outlier Removal Using Business Logic-
* As a data scientist when you have conversation with your business manager he will tell you that normally square ft per bedroom is 300 i.e 2 bhk apartment is minimum 600 sqft if you have for e.g 400 sqft apartment with 2 bhk than that seems suspicious and can be removed as an outlier.we will remove such outlier by keeping our minimum thresold per bhk to be 300 sqft
* Outlier Removal Using Standard Deviation and mean-
* Outlier Removal Using Bathrooms Feature-
* Build a Model-
* we will use linear regression model
* Find best model using GridSearchCV
* Test the model for few properties.

**Coding:**

1. **Client**

**App.html**

|  |
| --- |
| <!DOCTYPE html> |
|  | <html> |
|  | <head> |
|  | <title>Banglore Home Price Prediction</title> |
|  | <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script> |
|  | <script src="app.js"></script> |
|  | <link rel="stylesheet" href="app.css"> |
|  | </head> |
|  | <body> |
|  | <div class="img"></div> |
|  | <form class="form"> |
|  | <h2>Area (Square Feet)</h2> |
|  | <input class="area" type="email" id="uiSqft" class="floatLabel" name="Squareft" value="1000"> |
|  | <h2>BHK</h2> |
|  | <div class="switch-field"> |
|  | <input type="radio" id="radio-bhk-1" name="uiBHK" value="1"/> |
|  | <label for="radio-bhk-1">1</label> |
|  | <input type="radio" id="radio-bhk-2" name="uiBHK" value="2" checked/> |
|  | <label for="radio-bhk-2">2</label> |
|  | <input type="radio" id="radio-bhk-3" name="uiBHK" value="3"/> |
|  | <label for="radio-bhk-3">3</label> |
|  | <input type="radio" id="radio-bhk-4" name="uiBHK" value="4"/> |
|  | <label for="radio-bhk-4">4</label> |
|  | <input type="radio" id="radio-bhk-5" name="uiBHK" value="5"/> |
|  | <label for="radio-bhk-5">5</label> |
|  | </div> |
|  | </form> |
|  | <form class="form"> |
|  | <h2>Bath</h2> |
|  | <div class="switch-field"> |
|  | <input type="radio" id="radio-bath-1" name="uiBathrooms" |
|  |  |
|  |  |
|  |  |
|  | <input type="radio" id="radio-bath-5" name="uiBathrooms" value="5"/> |
|  | <label for="radio-bath-5">5</label> |
|  | </div> |
|  | <h2>Location</h2> |
|  | <div> |
|  |  |
|  |  |
|  |  |
|  |  |
|  | </select> |
|  | </div> |
|  | <button class="submit" onclick="onClickedEstimatePrice()" type="button">Estimate Price</button> |
|  | <div id="uiEstimatedPrice" class="result"> <h2></h2> </div> |
|  | </body> |
|  | </html> |

|  |
| --- |
| value="1"/> |
|  | <label for="radio-bath-1">1</label> |
|  | <input type="radio" id="radio-bath-2" name="uiBathrooms" value="2" checked/> |
|  | <label for="radio-bath-2">2</label> |
|  | <input type="radio" id="radio-bath-3" name="uiBathrooms" value="3"/> |
|  | <label for="radio-bath-3">3</label> |
|  | <input type="radio" id="radio-bath-4" name="uiBathrooms" value="4"/> |
|  | <label for="radio-bath-4">4</label> |
|  | <input type="radio" id="radio-bath-5" name="uiBathrooms" value="5"/> |
|  | <label for="radio-bath-5">5</label> |
|  | </div> |
|  | <h2>Location</h2> |
|  | <div> |
|  | <select class="location" name="" id="uiLocations"> |
|  | <option value="" disabled="disabled" selected="selected">Choose a Location</option> |
|  | <option>Electronic City</option> |
|  | <option>Rajaji Nagar</option> |
|  | </select> |
|  | </div> |
|  | <button class="submit" onclick="onClickedEstimatePrice()" type="button">Estimate Price</button> |
|  | <div id="uiEstimatedPrice" class="result"> <h2></h2> </div> |
|  | </body> |
|  | </html> |

1. **Server**

**Sever.py**

|  |
| --- |
| from flask import Flask, request, jsonify |
|  | import util |
|  |  |
|  | app = Flask(\_\_name\_\_) |
|  |  |
|  | @app.route('/get\_location\_names', methods=['GET']) |
|  | 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']) |

|  |
| --- |
| response = jsonify({ |
|  | 'estimated\_price': util.get\_estimated\_price(location,total\_sqft,bhk,bath) |
|  | }) |
|  | response.headers.add('Access-Control-Allow-Origin', '\*') |
|  |  |
|  | return response |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| if \_\_name\_\_ == "\_\_main\_\_": |
|  | print("Starting Python Flask Server For Home Price Prediction...") |
|  | util.load\_saved\_artifacts() |
|  | app.run() |

1. **Model**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**from** **matplotlib** **import** pyplot **as** plt

%matplotlib inline

**import** **matplotlib**

matplotlib.rcParams["figure.figsize"] = (20,10)

df1 = pd.read\_csv("bengaluru\_house\_prices.csv")

df1.head()  
df1.shape  
df1.columns

df1['area\_type'].unique()

df1['area\_type'].value\_counts()

df2 = df1.drop(['area\_type','society','balcony','availability'],axis='columns')

df2.shape

df2.isnull().sum()  
df2.shape

df3 = df2.dropna()

df3.isnull().sum()  
df3.shape

df3['bhk'] = df3['size'].apply(**lambda** x: int(x.split(' ')[0]))

df3.bhk.unique()

**def** is\_float(x):

**try**:

float(x)

**except**:

**return** **False**

**return** **True**

2+3

df3[~df3['total\_sqft'].apply(is\_float)].head(10)

**def** convert\_sqft\_to\_num(x):

tokens = x.split('-')

**if** len(tokens) == 2:

**return** (float(tokens[0])+float(tokens[1]))/2

**try**:

**return** float(x)

**except**:

**return** **None**

df4 = df3.copy()

df4.total\_sqft = df4.total\_sqft.apply(convert\_sqft\_to\_num)

df4 = df4[df4.total\_sqft.notnull()]

df4.head(2)

df4.loc[30]

(2100+2850)/2

df5 = df4.copy()

df5['price\_per\_sqft'] = df5['price']\*100000/df5['total\_sqft']

df5.head()

df5\_stats = df5['price\_per\_sqft'].describe()

df5\_stats

df5.location = df5.location.apply(**lambda** x: x.strip())

location\_stats = df5['location'].value\_counts(ascending=**False**)

location\_stats

location\_stats.values.sum()

len(location\_stats[location\_stats>10])

len(location\_stats)

len(location\_stats[location\_stats<=10])

location\_stats\_less\_than\_10 = location\_stats[location\_stats<=10]

location\_stats\_less\_than\_10

len(df5.location.unique())

df5.head(10)

df5[df5.total\_sqft/df5.bhk<300].head()

df5.shape

df6 = df5[~(df5.total\_sqft/df5.bhk<300)]

df6.shape

df6.price\_per\_sqft.describe()

**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

df7 = remove\_pps\_outliers(df6)

df7.shape

**def** plot\_scatter\_chart(df,location):

bhk2 = df[(df.location==location) & (df.bhk==2)]

bhk3 = df[(df.location==location) & (df.bhk==3)]

matplotlib.rcParams['figure.figsize'] = (15,10)

plt.scatter(bhk2.total\_sqft,bhk2.price,color='blue',label='2 BHK', s=50)

plt.scatter(bhk3.total\_sqft,bhk3.price,marker='+', color='green',label='3 BHK', s=50)

plt.xlabel("Total Square Feet Area")

plt.ylabel("Price (Lakh Indian Rupees)")

plt.title(location)

plt.legend()

plot\_scatter\_chart(df7,"Rajaji Nagar")

plot\_scatter\_chart(df7,"Hebbal")

**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')

df8 = remove\_bhk\_outliers(df7)

*# df8 = df7.copy()*

df8.shape

plot\_scatter\_chart(df8,"Rajaji Nagar")

plot\_scatter\_chart(df8,"Hebbal")

**import** **matplotlib**

matplotlib.rcParams["figure.figsize"] = (20,10)

plt.hist(df8.price\_per\_sqft,rwidth=0.8)

plt.xlabel("Price Per Square Feet")

plt.ylabel("Count")

df8.bath.unique()

plt.hist(df8.bath,rwidth=0.8)

plt.xlabel("Number of bathrooms")

plt.ylabel("Count")

df8[df8.bath>10]

df8[df8.bath>df8.bhk+2]

df9 = df8[df8.bath<df8.bhk+2]

df9.shape

df9.head(2)

df10 = df9.drop(['size','price\_per\_sqft'],axis='columns')

df10.head(3)

dummies = pd.get\_dummies(df10.location)

dummies.head(3)

df11 = pd.concat([df10,dummies.drop('other',axis='columns')],axis='columns')

df11.head()

df12 = df11.drop('location',axis='columns')

df12.head(2)

df12.shape

X = df12.drop(['price'],axis='columns')

X.head(3)

X.shape

y = df12.price

y.head(3)

len(y)

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=10)

**from** **sklearn.linear\_model** **import** LinearRegression

lr\_clf = LinearRegression()

lr\_clf.fit(X\_train,y\_train)

lr\_clf.score(X\_test,y\_test)

**from** **sklearn.model\_selection** **import** ShuffleSplit

**from** **sklearn.model\_selection** **import** cross\_val\_score

cv = ShuffleSplit(n\_splits=5, test\_size=0.2, random\_state=0)

cross\_val\_score(LinearRegression(), X, y, cv=cv)

**from** **sklearn.model\_selection** **import** GridSearchCV

**from** **sklearn.linear\_model** **import** Lasso

**from** **sklearn.tree** **import** DecisionTreeRegressor

**def** find\_best\_model\_using\_gridsearchcv(X,y):

algos = {

'linear\_regression' : {

'model': LinearRegression(),

'params': {

'normalize': [**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'])

find\_best\_model\_using\_gridsearchcv(X,y)

**def** predict\_price(location,sqft,bath,bhk):

loc\_index = np.where(X.columns==location)[0][0]

x = np.zeros(len(X.columns))

x[0] = sqft

x[1] = bath

x[2] = bhk

**if** loc\_index >= 0:

x[loc\_index] = 1

**return** lr\_clf.predict([x])[0]

predict\_price('1st Phase JP Nagar',1000, 2, 2)

predict\_price('1st Phase JP Nagar',1000, 3, 3)

predict\_price('Indira Nagar',1000, 2, 2)

predict\_price('Indira Nagar',1000, 3, 3)

**import** **pickle**

**with** open('banglore\_home\_prices\_model.pickle','wb') **as** f:

pickle.dump(lr\_clf,f)

**import** **json**

columns = {

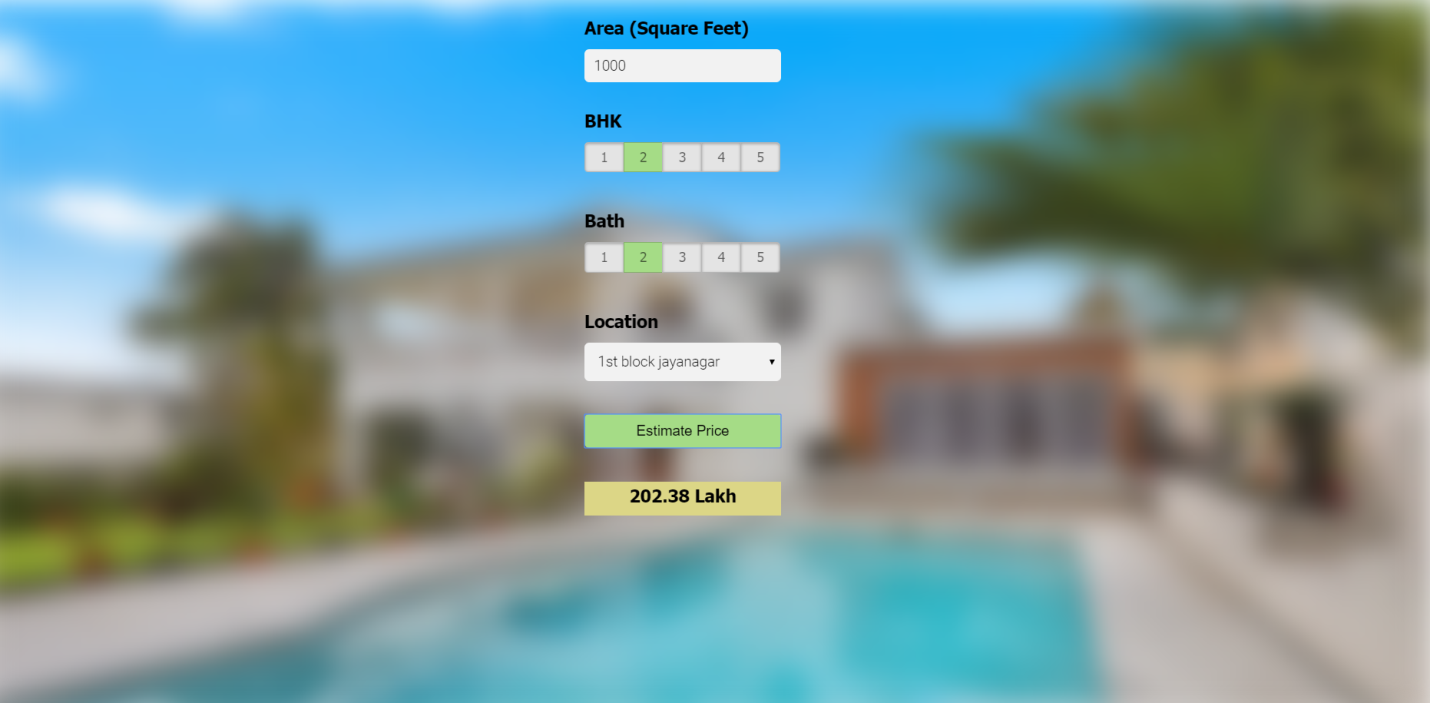
'data\_columns' : [col.lower() **for** col **in** X.columns]

}

**with** open("columns.json","w") **as** f:

f.write(json.dumps(columns))

**Results(Screenshot):**

****

**Observations:**

* We can make reasonable prediction about price a estate will sell for based on characterstics of property.
* Keys include handling NA values,normalizing variable,optimizing hyperparameter for candidate models,and chosing best model.