# ***Bangalore House Price Prediction – Regression***

***By Hrithik***

### *Objective:*

To predict the price values of house data in the city of Bangalore, India

### *Description*

**The dataset contains the price of the houses in Banglore,India. The datasets give us information on land area type, availability, size, no of bedrooms, society, area (sq ft), no of balconies, no of bathrooms and the price in lakhs**

### *Source -*

**https://www.kaggle.com/datasets/saipavansaketh/pune-house-data?select=Bangalore++house+data.csv**

### *Summary of data exploration and actions taken for data cleaning and feature engineering*

It was found that there was a lot of missing data in each of the feature's columns and also it was found important numerical columns were of object data type with missing typos like sqft_area was given as ‘1000-1200’ range and ‘30 Acres’ which were to cleaned and converted to numeric datatypes. Also, the number of bedrooms were given as string (4 Bedrooms or 2 BHK) which had to be converted to numeric. Columns like society, availability, etc. were categorical and had to be one hot encoded which led to dimensionality increase from ten columns to 3200 columns.

Standard Scaler was applied to one linear regression model but it made the model to be overfitting with negative r2 score and thus not used for other models. Also, Boxcox was used on one of the models but there wasn’t an improvement in r2_score or the Root mean sqaure error.

**Before Data-Cleaning**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   area_type     13320 non-null  object 
 1   availability  13320 non-null  object 
 2   location      13319 non-null  object 
 3   size          13304 non-null  object 
 4   society       7818 non-null   object 
 5   total_sqft    13320 non-null  object 
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(3), object(6)
memory usage: 936.7+ KB
```

**After Cleaning**

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12610 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   area_type     12610 non-null  object 
 1   availability  12610 non-null  object 
 2   location      12610 non-null  object 
 3   size          12610 non-null  float64
 4   society       12610 non-null  object 
 5   total_sqft    12610 non-null  float64
 6   bath          12610 non-null  float64
 7   balcony       12610 non-null  float64
 8   price         12610 non-null  float64
dtypes: float64(5), object(4)
memory usage: 1.2+ MB
```

## *Summary of Models*

Initially a simple linear regression model was made with an r2_score of 0.46 and rmse of 100 lakhs

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
def rmse(ytrue, ypredicted):
    return np.sqrt(mean_squared_error(ytrue, ypredicted))
lr = LinearRegression()
lr.fit(X_train,Y_train)
y_pred = lr.predict(X_test)
```

```
print(rmse(Y_test,y_pred))
```

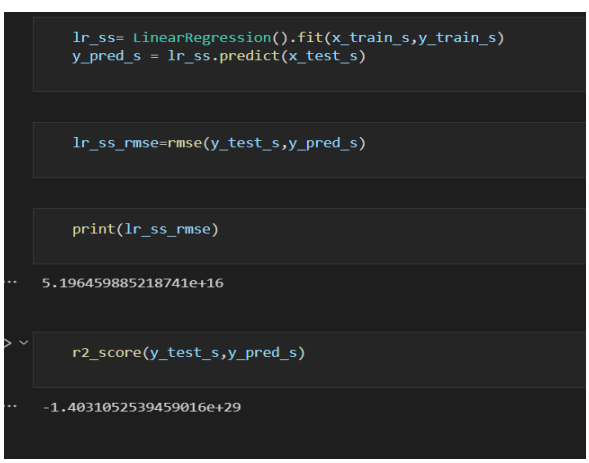
```
100.46097194296561
```

+ Code + Markdown

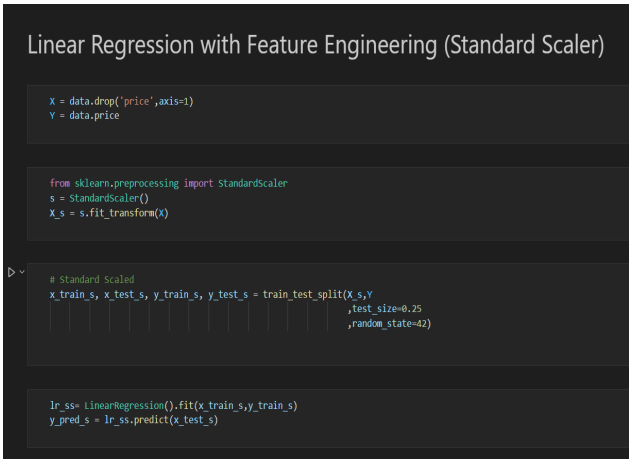
```
from sklearn.metrics import r2_score
r2_score(Y_test,y_pred)
```

```
0.46234278817825347
```

Then with standard scaler a linear regression model was made with a negative r2_score and rmse of 5 Lakhs. The negative r2_score meant that the model is overfitting and is biased to the training set



```
lr_ss= LinearRegression().fit(x_train_s,y_train_s)
y_pred_s = lr_ss.predict(x_test_s)

lr_ss_rmse=rmse(y_test_s,y_pred_s)

print(lr_ss_rmse)

5.196459885218741e+16

r2_score(y_test_s,y_pred_s)

-1.4031052539459016e+29
```



```
Linear Regression with Feature Engineering (Standard Scaler)

X = data.drop('price',axis=1)
Y = data.price

from sklearn.preprocessing import StandardScaler
s = StandardScaler()
X_s = s.fit_transform(X)

# Standard Scaled
x_train_s, x_test_s, y_train_s, y_test_s = train_test_split(X_s,Y
                                                            ,test_size=0.25
                                                            ,random_state=42)

lr_ss= LinearRegression().fit(x_train_s,y_train_s)
y_pred_s = lr_ss.predict(x_test_s)
```

Then the Standard Scaler was removed and a RidgeCV model with alpha as 3 was made which gave an r2_score of 0.53 and rmse of 92 lakhs which is better than the linear regression model

```
Ridge Regression

from sklearn.linear_model import RidgeCV

alphas = [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 80]

ridgeCV = RidgeCV(alphas=alphas,
                  cv=4).fit(X_train, Y_train)

ridgeCV_rmse = rmse(Y_test, ridgeCV.predict(X_test))

print(ridgeCV.alpha_, ridgeCV_rmse)

3.0 93.68938269571552

r2_score(Y_test, ridgeCV.predict(X_test))

0.5323817208599473
```

Similarly, a LassoCv model was made with an alpha value of 0.0005 which gave an r2_score of 0.46 and rmse of 100 lakhs which led to the conclusion that

RidgeCV suit our data better. ElasticNet was also tried but the code took 1 hour to run and still didn't able to get a model .

## Lasso Regression

```
from sklearn.linear_model import LassoCV

alphas2 = np.array([1e-5, 5e-5, 0.0001, 0.0005])

lassoCV = LassoCV(alphas=alphas2,
                  max_iter=5e4,
                  cv=3).fit(X_train, Y_train)
```

```
lassoCV_rmse = rmse(Y_test, lassoCV.predict(X_test))
print(lassoCV.alpha_, lassoCV_rmse)
```

```
0.0005 100.03457809010952
```

```
r2_score(Y_test, lassoCV.predict(X_test))
```

```
0.4668971381203416
```

In order to improve the r2_score outlier analysis was done and it was found out that the target variable and one of the feature variables had extreme outliers which were removed by using the Turkey Interquartile Range approach

```
df.describe()
```

| | size | total_sqft | bath | balcony | price |
|---|---|---|---|---|---|
| count | 12610.000000 | 12610.000000 | 12610.000000 | 12610.000000 | 12610.000000 |
| mean | 2.742744 | 1521.920762 | 2.622046 | 1.592942 | 106.265130 |
| std | 1.201074 | 1174.541201 | 1.223033 | 0.811507 | 132.018454 |
| min | 1.000000 | 5.000000 | 1.000000 | 0.000000 | 8.000000 |
| 25% | 2.000000 | 1100.000000 | 2.000000 | 1.000000 | 49.602500 |
| 50% | 3.000000 | 1265.000000 | 2.000000 | 2.000000 | 70.000000 |
| 75% | 3.000000 | 1650.000000 | 3.000000 | 2.000000 | 115.000000 |
| max | 43.000000 | 52272.000000 | 40.000000 | 3.000000 | 2912.000000 |

```
Q1 = df['price'].quantile(0.25)
Q2 = df['price'].quantile(0.5)
Q3 = df['price'].quantile(0.75)
print(Q1,Q2,Q3)
```

```
49.6025 70.0 115.0
```

```
IQR = Q3 - Q1
IQR
```

```
65.39750000000001
```

```
low_limit = Q1-IQR*1.5
upper_limit  = Q3+IQR*1.5
low_limit,upper_limit
```

```
(-48.49375000000001, 213.09625)
```

No Outlier in lower limit

```
df = df[df.price<upper_limit]
```

After the Outlier Removal another RidgeCV model was trained with and without boxcox transformation on target variable.
The RidgeCV without boxcox transformation did better with an R2 score of 0.71 and rmse of only 19 lakhs compared to the very high 100 lakhs of our first model.

```
alphas = [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 80]

ridgeCV = RidgeCV(alphas=alphas,
                  cv=4).fit(X_otrain, Y_otrain)
```

```
y_opred = ridgeCV.predict(X_otest)
rmse(Y_otest,y_opred)
```

```
19.410842611035136
```

```
r2_score(Y_otest,y_opred)
```

```
0.7135500412409608
```

## *Recommended Model*

I would recommend the RidgeCv model which was trained on the dataset after outlier removal without the boxcox transformation since it got the highest r2_score of 0.71 which is phenomenal in the field of finance and also the lowest Root Mean Sqaure Error of 19 Lakhs which is lower than the other models

### *Key Findings and Insights*

The dataset had a lot of outliers and missing typos like area of a house is 5 sqaure feet which practically makes no sense. It was found out that the target variable was not normally distributed and was left skewed which resulted in a negative lower limit while finding Interquartile range.

### *Suggestions to Be done in Future*

In order to improve the R2_score multicollinearity analysis can be done to find the z score and remove specific columns. Also, Principal Component Analysis can be done which might give us better insight to improve the R2 score and decrease the RMSE.