**Introduction:**

In this digital world where everything can be done using technology. A.I gives us room to grow in every aspect of our daily life. Buying/selling houses can be very difficult if we do not know what is the average price in that locality for the similar features like numer of rooms, balconies, area of the house etc., it will be difficult. So using machine learning to create a model which will be able to predict the house pricing based on the input data given we can try guessing the price of a house.

**Abstract:**

To apply this to practicality I tried using house price data of Bangalore and created the model to predict the house price from the given required data.

The parameters considered to predict the house pricing are:

* Area\_type
* Availability
* Location
* Size
* Society
* Total square foot
* No. of Bathrooms
* No.of Balconies
* And finally price for comparing with the predicted data while training

***Area******Type****: (area\_type)*

The area type is divided in to 3 categories:

1. Super built-up Area
2. Built-up Area
3. Plot Area

***Availability***: (*availability*)

This column gives us the information related to from when is the house ready to be moved into. It has 2 variations of data:

1. Ready to move:

If it is ready to be moved into.

1. Else date From when we can move into the house

***Location:*** *(location)*

This column consists of the location in which the house is, for example, Uttarahalli, Old Airport road, etc.

***Size:*** *(size)*

This column consists of the ‘BHK’ count. Example: 1 BHK etc.

***Society:*** *(society)*

Similar to the location this colum consists of the name of the society where the house is present.

***Total Square Foot:*** *(total\_sqft)*

This column consists of the area covered by the house or plot.(Most of the data in this column is in sqft units).

***No. of Bathrooms:*** *(bath)*

Consists of number of baths in the house in count.

***No. of Balconies:*** *(balcony)*

Consists of the number which denotes the number of balconies present.

***Price:*** *(price)*

Price of the house in lakhs (INR).

**Breif on data preprocessing:**

Firstly there are a lot of ***Nan*** values in the society column thus removing the column from our dataframe would be preferrable rather than dropping the rows with Nan values in the society column. Thus we remove the ‘society’ column. After dropping the ‘society’ column the number of nan values has reduced from 5502 which are the total in ‘society’ column to the maximum of 609 in ‘balcony’ column.

Replacing the Nan values would be messing our data for the outliers. Thus, the optimal way to reduce this problem is dropping those rows consisting of Nan values now.

By this procedure we are left with the dataframe which has 0 Nan values.

The data in columns ‘area\_type’, ‘location’, are in the object format and thus cannot be changed in to any numerical format with a single column to support our regression because, giving the object formats dummy values of whole numbers like 1,2,3... and so on would give some variables more preference due to higher int value than the others. Thus using ‘***One-hot-encoding***’ we change these columns in to numerical data.

The data input given in the ‘total\_sqft’ column is totally irregular. As it is observed that the values in the column are in acres, sq meters, sq yards, ghuntas and so on which need to be converted into the sqft units and into float datatype.

Similarly preprocessing the remaining data by convertin them in to float values the dataframe is ready to be used for model training.

**Model:**

**XGBoost:** XGBoost is one of the most popular machine learning algorithm these days. Regardless of the type of prediction task at hand; regression or classification.XGBoost is well known to provide better solutions than other machine learning algorithms. In fact, since its inception, it has become the "state-of-the-art” machine learning algorithm to deal with structured data.

Using the XGBoost’s XGBRegressor method with the following parameters and values, helped in creating a successful model with r2\_score of 0.72 for test data and 0.95 for training data.

Parameters:

objective ='reg:squarederror',

learning\_rate = 0.1,

max\_depth = 12,

alpha = 10

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code:**

*import pandas as pd*

*#importing libraries*

*import numpy as np*

*import re*

*import xgboost as xgb*

*from sklearn.metrics import mean\_squared\_error*

*from sklearn.metrics import r2\_score*

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

*df = pd.read\_csv('Bengaluru\_House\_Data.csv')*

*df = df.drop(["society"], axis = 1, inplace = True)*

*#one-hot-encoding*

*df = pd.concat([df, pd.get\_dummies(df[['area\_type']], prefix = 'area')], axis = 1)*

*df.drop(['area\_type'], axis = 1, inplace = True)*

*df[['availability']] = np.where(df[['availability']] == 'Ready To Move', 1, 0)*

*df = pd.concat([df, pd.get\_dummies(df[['location']])], axis = 1)*

*df.drop(['location'], axis = 1, inplace = True)*

*a = df[['size']]*

*b = df[['total\_sqft']]*

*a = list(a['size'])*

*b = list(b['total\_sqft'])*

*for i in range(len(a)):*

*a[i] = float(a[i].split()[0])*

*if('ac' in b[i] or "Ac" in b[i] or "AC" in b[i]): # Acres to sqft*

*b[i] = re.split(" |a|A", b[i])*

*b[i] = float(b[i][0])\*43560*

*elif('m' in b[i] or "M" in b[i]): #sq\_meters to sqft*

*b[i] = re.split(" |s|S", b[i])*

*b[i] = float(b[i][0])\*10.763910*

*elif('y' in b[i] or 'Y' in b[i]): #sqyards to sqft*

*b[i] = re.split(" |s|S", b[i])*

*b[i] = float(b[i][0])\*9*

*elif('ce' in b[i] or 'Ce' in b[i] or 'CE' in b[i]):*

*b[i] = re.split(" |c|C", b[i])*

*b[i] = float(b[i][0])\*435.54*

*elif("gu" in b[i] or "G" in b[i]):*

*b[i] = re.split(" |g|G", b[i])*

*b[i] = float(b[i][0])\*1089*

*else:*

*b[i] = re.split(" |s|S", b[i])*

*b[i] = float(b[i][0])*

*df['size'] = a*

*df['total\_sqft'] = b*

*#splitting data*

*X = pd.DataFrame(df)*

*X.drop(['price'], axis = 1, inplace = True)*

*Y = df[['price']]*

*xtrain, xtest, ytrain, ytest = tts(X, Y, test\_size = 0.2, random\_state = 0)*

*xg\_reg = xgb.XGBRegressor(objective ='reg:squarederror', learning\_rate = 0.1,*

*max\_depth = 14, alpha = 10, )*

*xg\_reg.fit(xtrain,ytrain)*

*preds = xg\_reg.predict(xtest)*

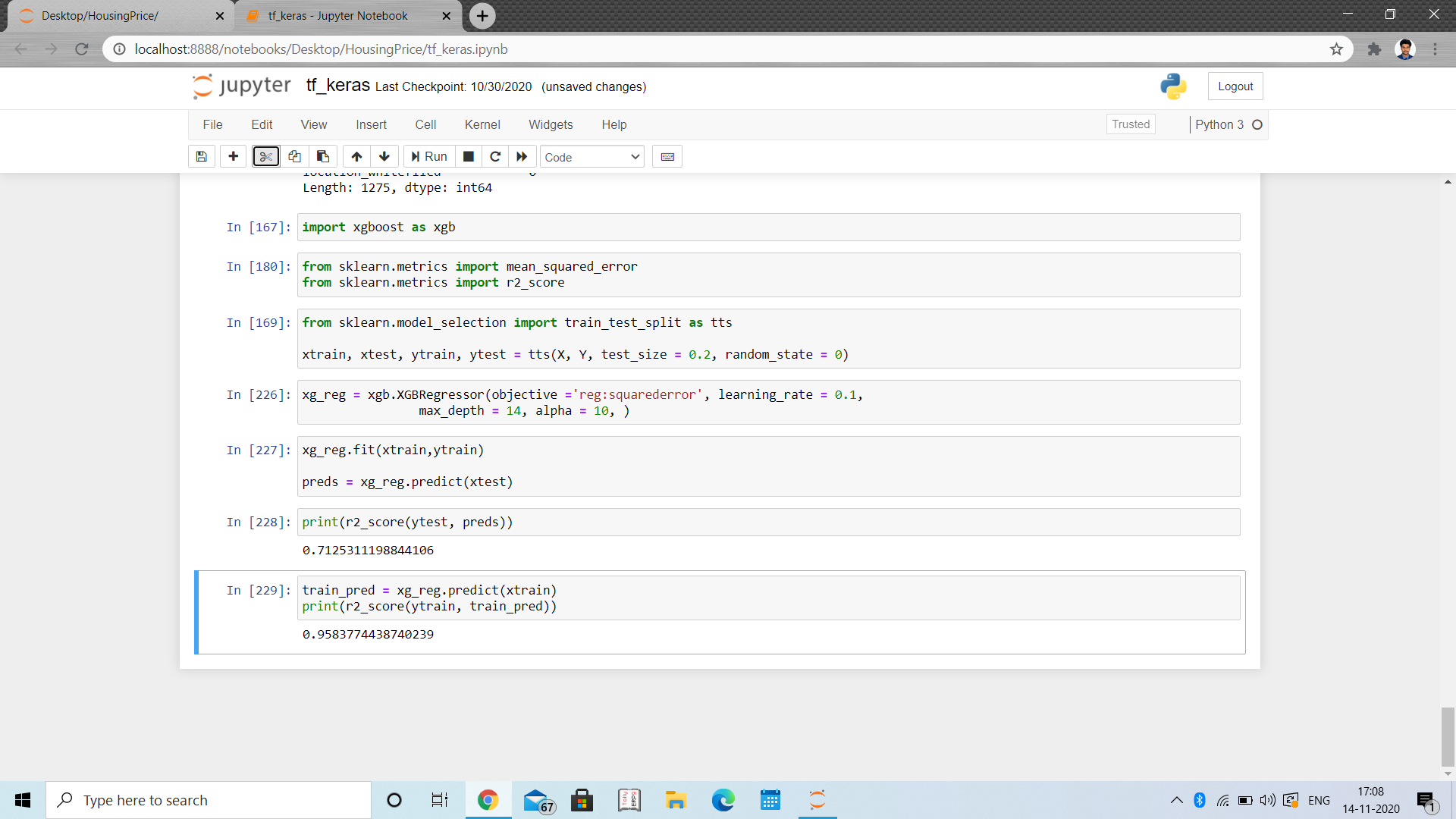
*print(r2\_score(ytest, preds))*

*train\_pred = xg\_reg.predict(xtrain)*

*print(r2\_score(ytrain, train\_pred))*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**Output:**

****

***Observations:***

LinearRegression(): Using the linear regression model gives us an r2\_score of 30% at maximum.

Trying to solve this problem I came up with 2 solutions, either creating a new network or using rgboost module.

New Network: The model which was created using a new neural network with 1 hidden layer could not help in improving the efficiency, rather it gave r2\_score in negative range of (-0.6 to -0.5) which shows that the model is very poorly fit and is being forced to predict the values. Thus instead of trying to increase the hidden layers, I straightly jumped to using XGBoost which is one of the most popular prediction algorithm.

XGBOOST: *Changing the learning\_rate from 0.1 to further down the road to 0.01 or 0.001 caused decrease In the model score significanly.*

*Maximum depth of the tree is 12 for us to get an optimal 70% score. Though increasing the depth might result in overfitting, in this case the data is too uncertain with lot* of outliers to be considered inside the data thus increasing the depth was the only solution to increase the efficiency.

Conclusion:

Using XGBOOST(eXtreme Gradient Boosting) algorithm I was able to create a model which gave around 70% efficiency.