VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science Engineering and Information Systems

Winter Semester –2023-24

B. Tech IT – Capstone Project

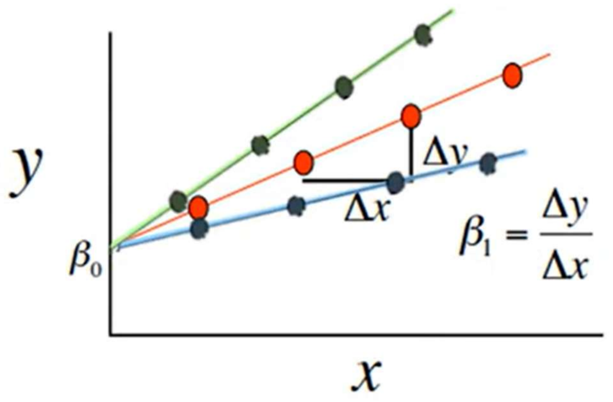
2nd Review

Register Number	20BIT0216
Student Name	VALLURU MOHAMMAD RASHEED
Project Domain (Capstone Project)	Artificial intelligence and Machine learning
Project Title (Capstone Project)	REALESTATE PREDICTOR: AI-DRIVEN REAL-TIME PROPERTY VALUATION
Guide Name	Dr. GUNDALA SWATHI MAM
Project Reviewers	Prof. Ganesan K SIR, Prof. CHANDRA MOULISWARAN S SIR.
Date of Reveiew-1	04-04-2024

Proposed Methodology:

Multiple Linear Regression:

This model builds links between the many variables being studied. Correlation coefficients or regression equations can be used to determine the correlation between variables. There are models that can identify the most important aspects for understanding the dependent variable. Multiple regression is used to collect data on both independent and dependent factors to anticipate certain prices. Multiple linear regression determines the link between dependent and independent variables.
This model operates by treating housing price projections as independent variables and data such as home price, size, property type, and bedrooms as dependent factors. As a result, the algorithm's aim is the house price, also known as the dependent variable, and the influencing elements are allocated as independent variables.
Thus, a correlation coefficient can be identified to determine the main variable.



$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

Lasso Regression (L1 Regularization):

Lasso regression is a linear regression approach that combines variable selection and regularization. It introduces a penalty term into the loss function, which is the absolute value of the coefficients multiplied by a regularization parameter (alpha). This promotes sparsity in the coefficients, thereby doing variable selection by setting certain coefficients to zero.

$$\text{Lasso} = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} |w_j|$$

Ridge Regression (L2 Regularization):

Ridge regression is another linear regression approach that regularizes the loss function by adding a penalty term equal to the square of the coefficients multiplied by a regularizations parameter (alpha). Ridge, unlike Lasso, does not pick variables, but rather lowers the coefficients to zero, diminishing their magnitude.

$$\text{Ridge} = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} w_j^2$$

Support vector machine (SVM):

It is a supervised learning method designed for classification and regression problems. In SVM, the algorithm seeks the ideal hyperplane that best separates or fits the data points into distinct classes or predicts the target variable in regression, all while maximizing the margin between the classes or data points.

In SVM regression, the aim is to create a function f(x) that predicts the target variable y based on input characteristics.

- **Hyperplane**: The hyperplane in SVM regression is defined as $f(x)=\langle w, x\rangle+b$, where w represents the weight vector, x is the input feature vector, and b is the bias factor. The hyperplane aims to suit the data points as closely as possible while maximising the margin.

- **Regularization**: SVM regression employs a regularisation parameter C to manage the trade-off between maximising margin and minimising mistakes. A greater value of C provides for more freedom in fitting the data, but it may result in overfitting, whereas a lower value of C imposes a bigger margin, but it may cause underfitting.

- **Loss Function**: In SVM regression, the loss function seeks to minimise the errors between projected and actual values while maximising the margin between the hyperplane and the data points. To avoid overfitting, the loss function often include a regularisation term.

- **Kernel Trick**: SVM regression can employ a variety of kernel functions (e.g., linear, polynomial, and radial basis functions) to translate input characteristics into a higher-dimensional space where the data may be more distinct. This enables SVM to identify complicated correlations between input characteristics and target variables.

Random forest:

Random Forest regression is an ensemble learning approach that integrates the predictions of numerous decision trees to get a more accurate and stable result. During training, several decision trees are created from random subsets of the training data and features. Each tree is trained individually, and the final prediction is calculated by averaging the predictions of all individual trees (for regression tasks). This ensemble technique reduces overfitting and improves generalization performance. Furthermore, Random Forests incorporates unpredictability into the feature selection process, increasing the model's resilience and lowering the danger of overfitting.

XGBoost:

Extreme Gradient Boosting (XGBoost) is a well-known and strong machine learning technique that excels in regression and classification problems. It is part of the gradient boosting method family, which iteratively creates an ensemble of weak learners to minimize a given loss function.

- **Gradient Boosting Process**: XGBoost begins by fitting an initial decision tree to the data and then adds additional trees in a sequential order to rectify mistakes caused by earlier models. Each new tree is trained using the residuals.

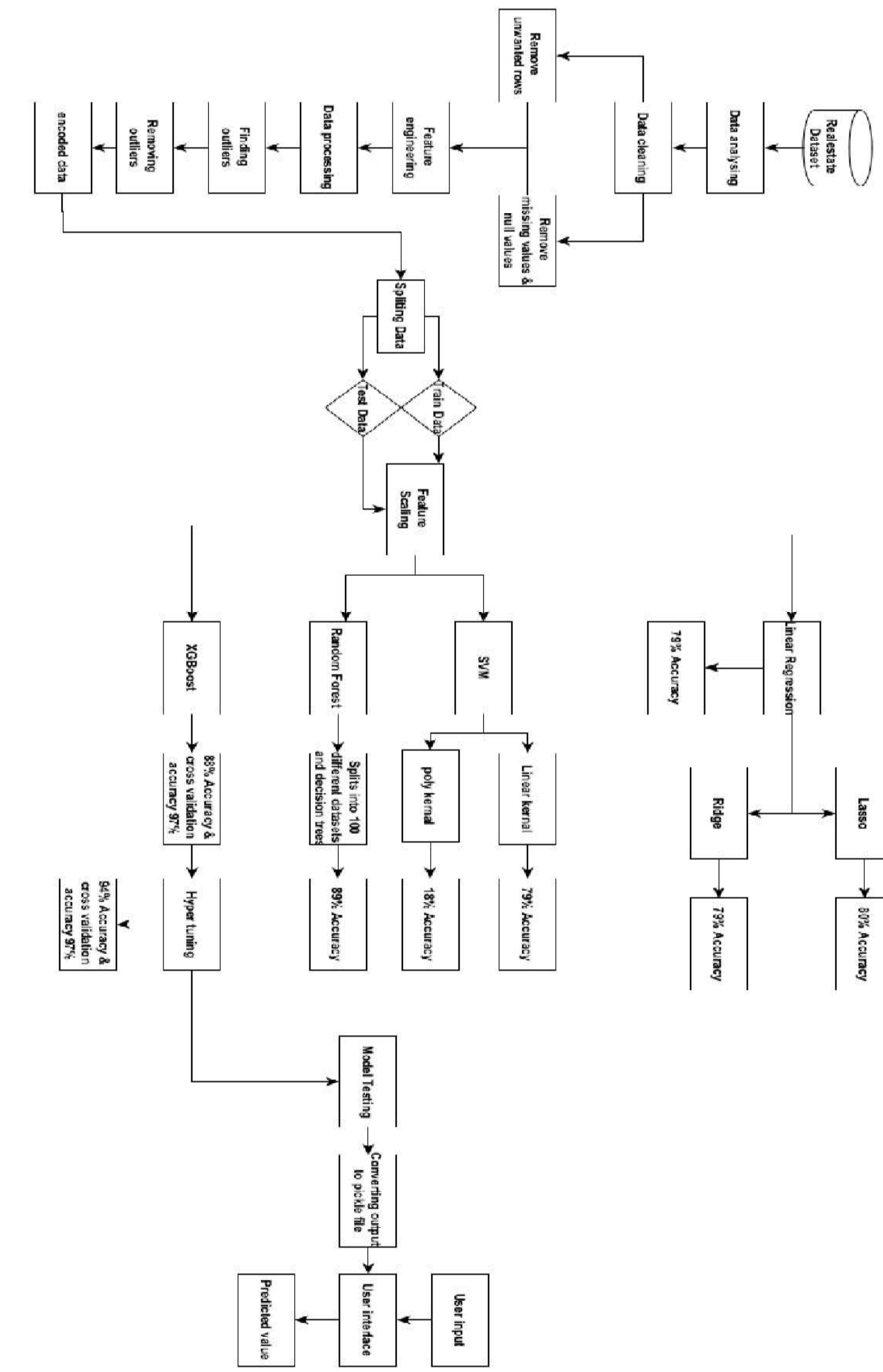
- **Regularization**: XGBoost uses regularisation approaches to reduce overfitting and increase generalisation performance. It has parameters like max_depth, min_child_weight, and gamma that govern the intricacy of the trees and keep them from growing too deep or splitting too frequently.

- **Loss Function Optimization**: XGBoost uses gradient descent to optimise a given loss function (for example, squared loss in regression). It computes the gradients of the loss function with respect to the model predictions and modifies the model parameters (tree structure and leaf scores) in the direction of least loss.

- **Parallel and Distributed Computing**: XGBoost is meant to be extremely scalable and efficient, with parallel and distributed computing options to handle huge datasets and expedite training. It uses techniques like approximation tree learning and histogram-based algorithms to boost computing performance.

- **Hyperparameter Tuning**: XGBoost has a variety of hyperparameters that may be adjusted to improve model performance, including the learning rate (eta), the number of trees (n_estimators), and the subsampling ratio (subsample). To discover the best hyperparameter combination, approaches such as grid search or random search are commonly used.

SYSTEM ARCHITECTURE:



The architecture provides the complete process of the model from beginning to end. We start with the data collection and cleaning the data and removes the unwanted data from the data set. Now we need to preprocess the data. After that we need to find any outliers present in the data and remove it after completing all the transformations now, we need to train the model and test that model and after that we need to deploy it flask.

Module Description:

- **Data analysis:** In this stage we need to check the data first and get detailed information about the data whether it is useful or not.
- **Data cleaning:** In this stage we need to search for null values if u find any null values in any column with minimum % then drop that row from the table and if u got so many null values in the column, then u need to drop that column, if u don't drop that column it leads to failure of our model. In my data society has 41.3% missing value so I have dropped that column.
- **Data preprocessing:** In this stage we need to convert the data into integer type by that we can easily train and test our regression model.
- **Model training:** In this stage I have used 6 models to compare the models which gives best accuracy. First, I have used linear regression in that I got an accuracy of 79% and rsme value of round 65-70% so it is overfitting. Then I used lasso, ridge and SVM too they also not gave much convenient outcome, so I used random forest and XGBoost models then I got some good accuracy. I used the XGBoost model to train and test the model.
- **Deployment:** In this stage I build a front-end model to showcase the output with the help of flask to combine the web application and the model.

Technology Used:

Hardware Requirements:

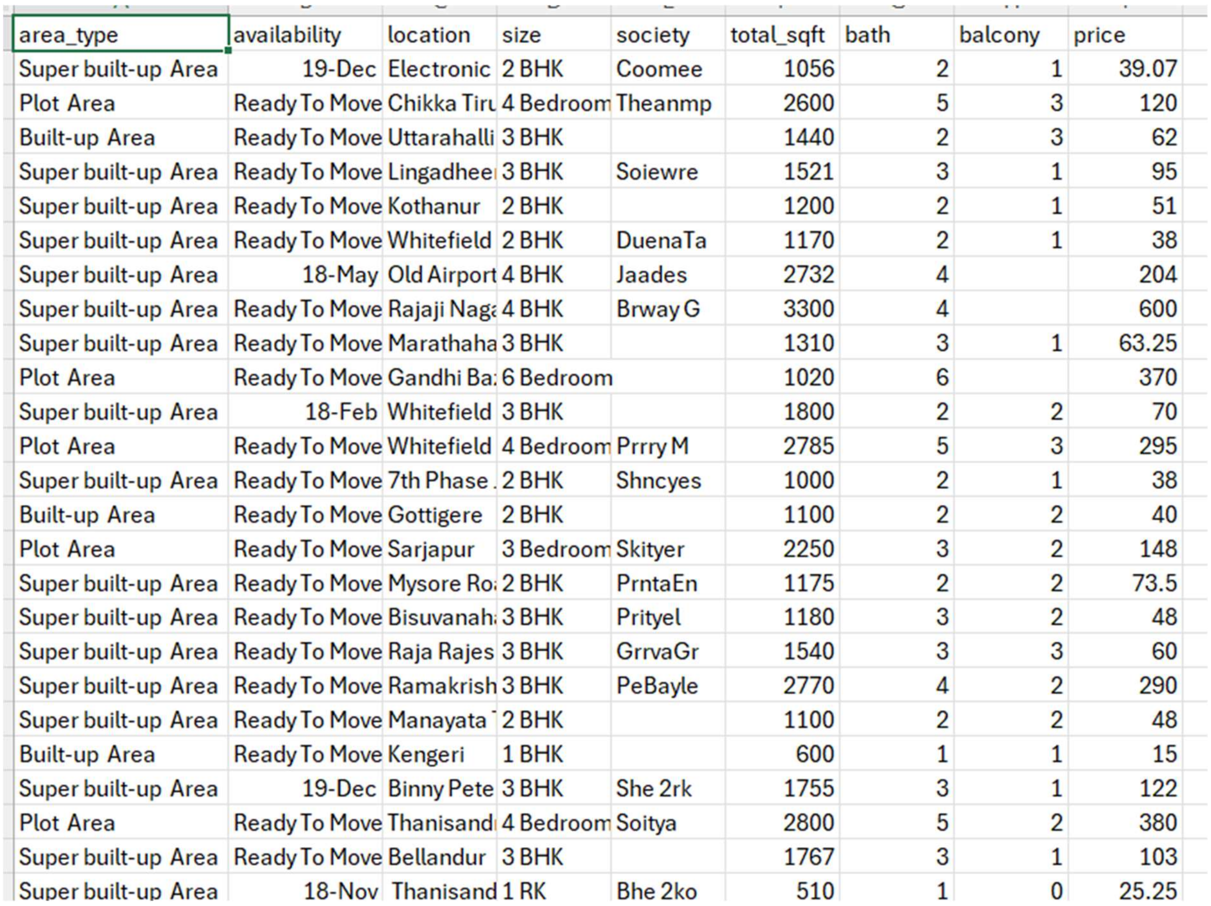
RAM: At least of 8 GBRAM is recommended, and more is better
Processor: A multi-core processor with a clock speed of 2 GHz or higher is recommended.
Storage space: a minimum of 2 GB of free storage space should be sufficient.

SOFTWARE REQUIREMENTS:

Editor: Jupiter Notebook
Language: Python
Libraries:

- NumPy: NumPy is a Python package that is used for scientific computing with Python.
- Pandas: Pandas are a Python package that is used for data manipulation and analysis.
- Matplotlib: Matplotlib is a plotting library for Python. It provides a wide range of 2D and 3D plots, charts, and graphs.
- Seaborn: Seaborn is a Python package for creating visually appealing and useful statistics graphs.

Dataset used:



area_type	availability	location	size	society	total_sqft	bath	balcony	price
Super built-up Area	19-Dec	Electronic	2 BHK	Coomee	1056	2	1	39.07
Plot Area	Ready To Move	Chikka Tiru	4 Bedroom	Theanmp	2600	5	3	120
Built-up Area	Ready To Move	Uttarahalli	3 BHK		1440	2	3	62
Super built-up Area	Ready To Move	Lingadheei	3 BHK	Soiewre	1521	3	1	95
Super built-up Area	Ready To Move	Kothanur	2 BHK		1200	2	1	51
Super built-up Area	Ready To Move	Whitefield	2 BHK	DuenaTa	1170	2	1	38
Super built-up Area	18-May	Old Airport	4 BHK	Jaades	2732	4		204
Super built-up Area	Ready To Move	Rajaji Naga	4 BHK	Brway G	3300	4		600
Super built-up Area	Ready To Move	Marathaha	3 BHK		1310	3	1	63.25
Plot Area	Ready To Move	Gandhi Ba:	6 Bedroom		1020	6		370
Super built-up Area	18-Feb	Whitefield	3 BHK		1800	2	2	70
Plot Area	Ready To Move	Whitefield	4 Bedroom	Prrry M	2785	5	3	295
Super built-up Area	Ready To Move	7th Phase .	2 BHK	Shncyes	1000	2	1	38
Built-up Area	Ready To Move	Gottigere	2 BHK		1100	2	2	40
Plot Area	Ready To Move	Sarjapur	3 Bedroom	Skityer	2250	3	2	148
Super built-up Area	Ready To Move	Mysore Ro:	2 BHK	PrntaEn	1175	2	2	73.5
Super built-up Area	Ready To Move	Bisuvanah:	3 BHK	Prityel	1180	3	2	48
Super built-up Area	Ready To Move	Raja Rajes	3 BHK	GrrvaGr	1540	3	3	60
Super built-up Area	Ready To Move	Ramakrish	3 BHK	PeBayle	2770	4	2	290
Super built-up Area	Ready To Move	Manayata	2 BHK		1100	2	2	48
Built-up Area	Ready To Move	Kengeri	1 BHK		600	1	1	15
Super built-up Area	19-Dec	Binny Pete	3 BHK	She 2rk	1755	3	1	122
Plot Area	Ready To Move	Thanisandı	4 Bedroom	Soitya	2800	5	2	380
Super built-up Area	Ready To Move	Bellandur	3 BHK		1767	3	1	103
Super built-up Area	18-Nov	Thanisand	1 RK	Bhe 2ko	510	1	0	25.25

Sample code:

Data cleaning:

```
df.isnull().mean()*
plt.figure(figsize=(16,9))
sns.heatmap(df.isnull())
df2 = df.drop('society', axis='columns')
df2.shape
df2['balcony'] = df2['balcony'].fillna(df2['balcony'].mean())
df2.isnull().sum()
df3 = df2.dropna()
df3.shape
df3.isnull().sum()
```

Finding outlier:

```
def diagnostic_plots(df, variable):
    plt.figure(figsize=(16, 4))
     plt.subplot(1, 3, 1)
    sns.distplot(df[variable], bins=30)
    plt.title('Histogram')
    plt.subplot(1, 3, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.ylabel('Variable quantiles')
    plt.subplot(1, 3, 3)
    sns.boxplot(y=df[variable])
    plt.title('Boxplot')
    plt.show()
num_var = ["bath","balcony","total_sqft_int","bhk","price"]
for var in num_var:
  print("******* {} *******".format(var))
  diagnostic_plots(df7, var)
df7[df7['total_sqft_int']/df7['bhk'] < 350].head()
```

Removing outlier:

```
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

df9 = remove_pps_outliers(df8)
df9.shape
```

ML models:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
lr = LinearRegression()
lr_lasso = Lasso()
lr_ridge = Ridge()
def rmse(y_test, y_pred):
  return np.sqrt(mean_squared_error(y_test, y_pred))

lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test)
lr_rmse = rmse(y_test, lr.predict(X_test))
lr_score, lr_rmse

lr_lasso.fit(X_train, y_train)
lr_lasso_score=lr_lasso.score(X_test, y_test)
lr_lasso_rmse = rmse(y_test, lr_lasso.predict(X_test))
lr_lasso_score, lr_lasso_rmse

lr_ridge.fit(X_train, y_train)
lr_ridge_score = lr_ridge.score(X_test, y_test)
lr_ridge_rmse = rmse(y_test, lr_ridge.predict(X_test))
```

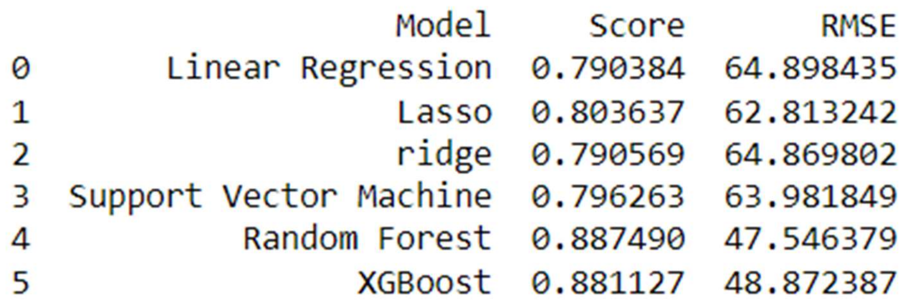
```
lr_ridge_score, lr_ridge_rmse

from sklearn.svm import SVR
svr = SVR()
svr.fit(X_train,y_train)
svr_score=svr.score(X_test,y_test)
svr_rmse = rmse(y_test, svr.predict(X_test))
svr_score, svr_rmse

svr = SVR(kernel='linear')
svr.fit(X_train,y_train)
svr_score=svr.score(X_test,y_test)
svr_rmse = rmse(y_test, svr.predict(X_test))
svr_score, svr_rmse

from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(X_train,y_train)
rfr_score=rfr.score(X_test,y_test)
rfr_rmse = rmse(y_test, rfr.predict(X_test))
rfr_score, rfr_rmse

import xgboost
xgb_reg = xgboost.XGBRegressor()
xgb_reg.fit(X_train,y_train)
xgb_reg_score=xgb_reg.score(X_test,y_test)
xgb_reg_rmse = rmse(y_test, xgb_reg.predict(X_test))
xgb_reg_score, xgb_reg_rmse
```

Hyper tuning:

```
from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBRegressor

xgb1 = XGBRegressor()
parameters = {'learning_rate': [0.1,0.03, 0.05, 0.07],
            'min_child_weight': [1,3,5],
            'max_depth': [4, 6, 8],
            'gamma':[0,0.1,0.001,0.2],
            'subsample': [0.7,1,1.5],
            'colsample_bytree': [0.7,1,1.5],
            'objective':['reg:linear'],
            'n_estimators': [100,300,500]}

xgb_grid = GridSearchCV(xgb1,
                    parameters,
                    cv = 2,
                    n_jobs = -1,
                    verbose=True)

xgb_grid.fit(X_train, y_train)
```

```
print(xgb_grid.best_score_)
print(xgb_grid.best_params_)
```

Model Output:



```
                    Model     Score       RMSE
0       Linear Regression  0.790384  64.898435
1                   Lasso  0.803637  62.813242
2                   ridge  0.790569  64.869802
3  Support Vector Machine  0.796263  63.981849
4           Random Forest  0.887490  47.546379
5                 XGBoost  0.881127  48.872387
```

Model testing:

```
def
predict_house_price(model,bath,balcony,total_sqft_int,bhk,price_per_sqft,area_type,availabil
ity,location):

  x =np.zeros(len(X.columns)) # create zero numpy array, len = 107 as input value for model

  # adding feature's value accorind to their column index
  x[0]=bath
  x[1]=balcony
  x[2]=total_sqft_int
  x[3]=bhk
  x[4]=price_per_sqft

  if "availability"=="Ready To Move":
    x[8]=1

  if 'area_type'+area_type in X.columns:
    area_type_index = np.where(X.columns=="area_type"+area_type)[0][0]
    x[area_type_index] =1

    #print(area_type_index)

  if 'location_'+location in X.columns:
    loc_index = np.where(X.columns=="location_"+location)[0][0]
    x[loc_index] =1

    #print(loc_index)

  #print(x)

  # feature scaling
  x = sc.transform([x])[0] # give 2d np array for feature scaling and get 1d scaled np array
  #print(x)

  return model.predict([x])[0]
```