

**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:**

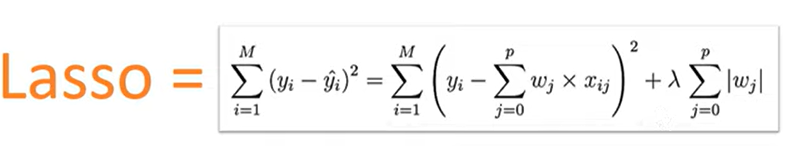
This model builds links between the many variables being studied. Correlation coefficients or regression equations can be used to determine the correlation between variables. There are models that can identify the most important aspects for understanding the dependent variable. Multiple regression is 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.

A graph of mathematical equations

Description automatically generated

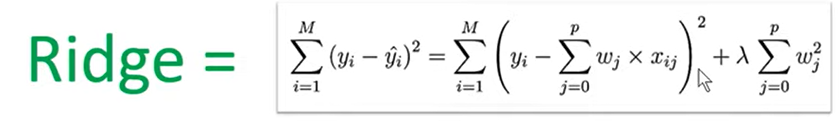
**Lasso Regression (L1 Regularization):**

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.



**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.



**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)=⟨w, x⟩+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.

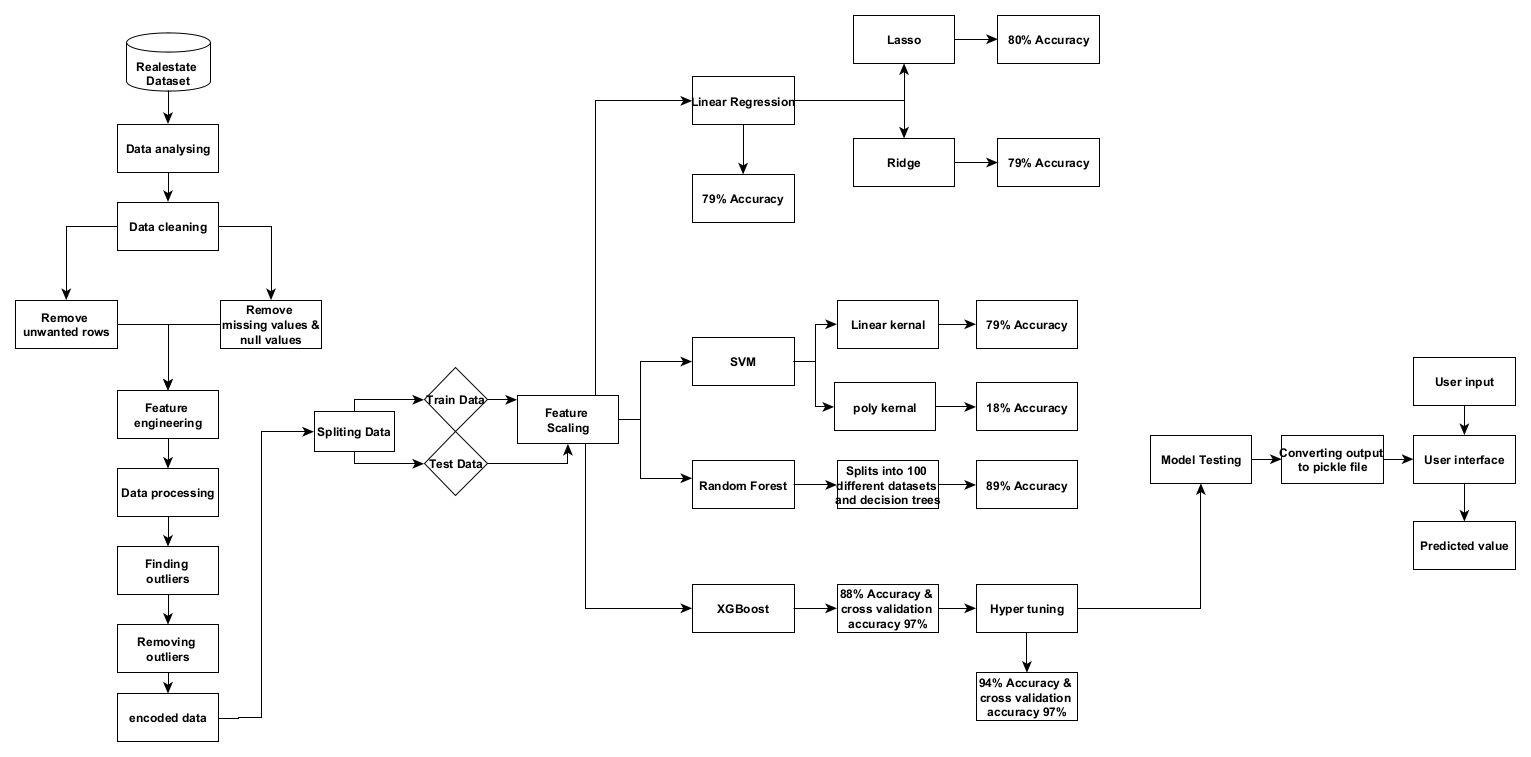
**Random forest:**  
Random Forest regression is an ensemble learning approach that integrates the predictions of numerous decision trees to get a more accurate and stable result. During training, several decision trees are created from random subsets of the training data and features. Each tree is trained 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

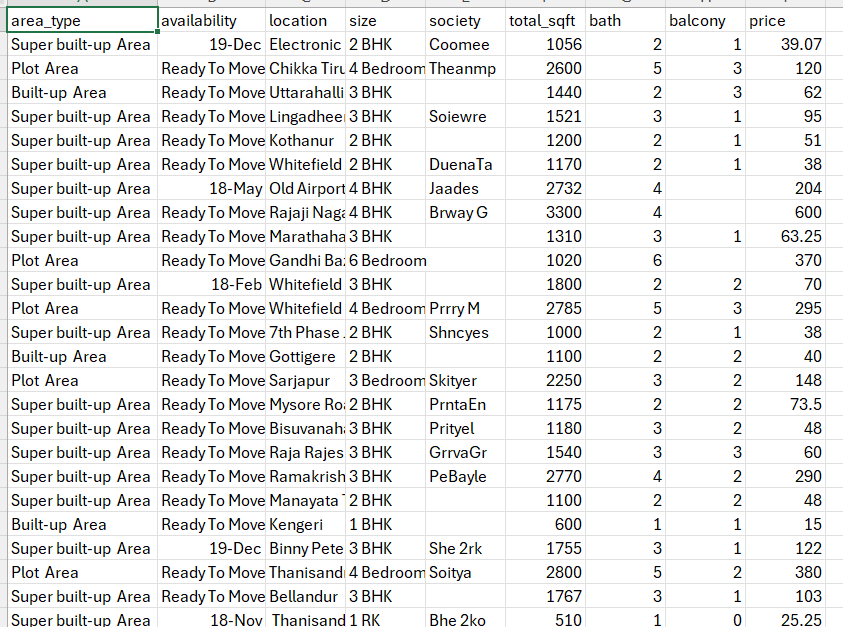
Libraries:

* 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:**



**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:**

A screenshot of a computer screen

Description automatically generated

**Model testing:**

def predict\_house\_price(model,bath,balcony,total\_sqft\_int,bhk,price\_per\_sqft,area\_type,availability,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]