HOUSE PRICE PREDICTION

Abstract:

House price prediction is a crucial task in real estate and property markets. This study focuses on developing a machine learning model for accurately predicting house prices based on various features such as location, size, amenities, and historical sales data. The project aims to leverage a dataset of real estate transactions and employ regression techniques to create a predictive model. By analyzing the dataset, selecting relevant features, and implementing appropriate algorithms, this research aims to provide a reliable tool for both homebuyers and sellers to make informed decisions. The outcome of this project is expected to contribute to the efficiency and transparency of the housing market, assisting stakeholders in their property.

Data processing:

Data processing is a crucial step in house price prediction. Here are some key data processing steps:

1.Data Collection: Gather data on various factors that can affect house prices, such as square footage, number of bedrooms, location, and recent sale prices.

2.Data Cleaning: Remove or impute missing values, correct inconsistencies, and address outliers in the dataset to ensure data quality

3.Feature Selection: Choose the most relevant features (variables) for your prediction model. Some may have a stronger impact on house prices than others.

4.Data Transformation: Transform categorical variables into numerical values using techniques like one-hot encoding. You may also scale or normalize numerical features.

5.Feature Engineering: Create new features that may capture meaningful information, such as the age of the property, proximity to amenities, or a combination of existing variables.

6.Splitting Data: Divide your dataset into training and testing sets to evaluate your model's performance.

7.Data Preprocessing: Standardize, normalize, or apply other preprocessing techniques to make the data suitable for various machine learning algorithms.

8.Data Exploration: Visualize and explore the data to gain insights into the relationships between features and house prices.

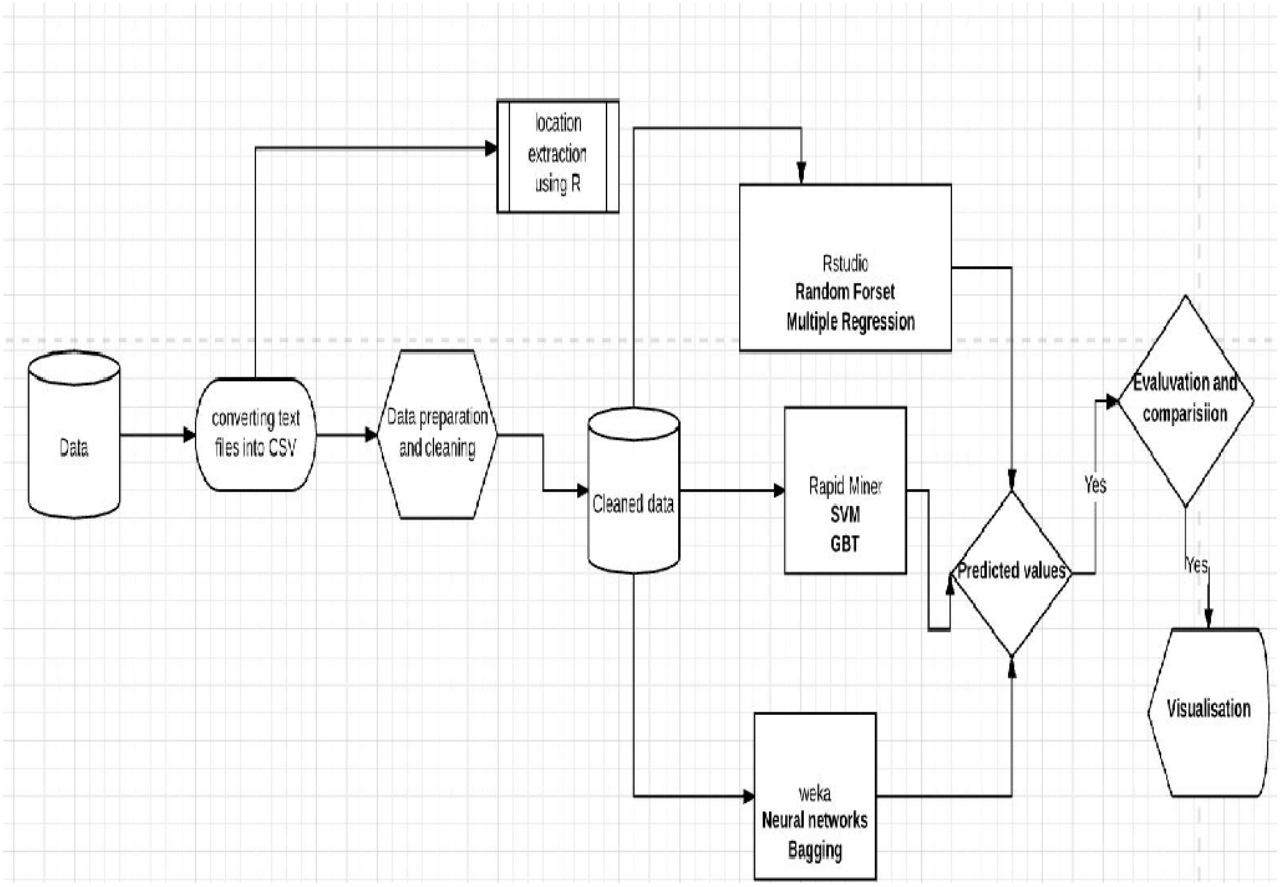
9.Model Training: Use machine learning algorithms to train a predictive model based on your processed data.

10.Model Evaluation: Assess your model's performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).

11.Model Tuning: Adjust hyperparameters and explore different algorithms to improve the model's accuracy.

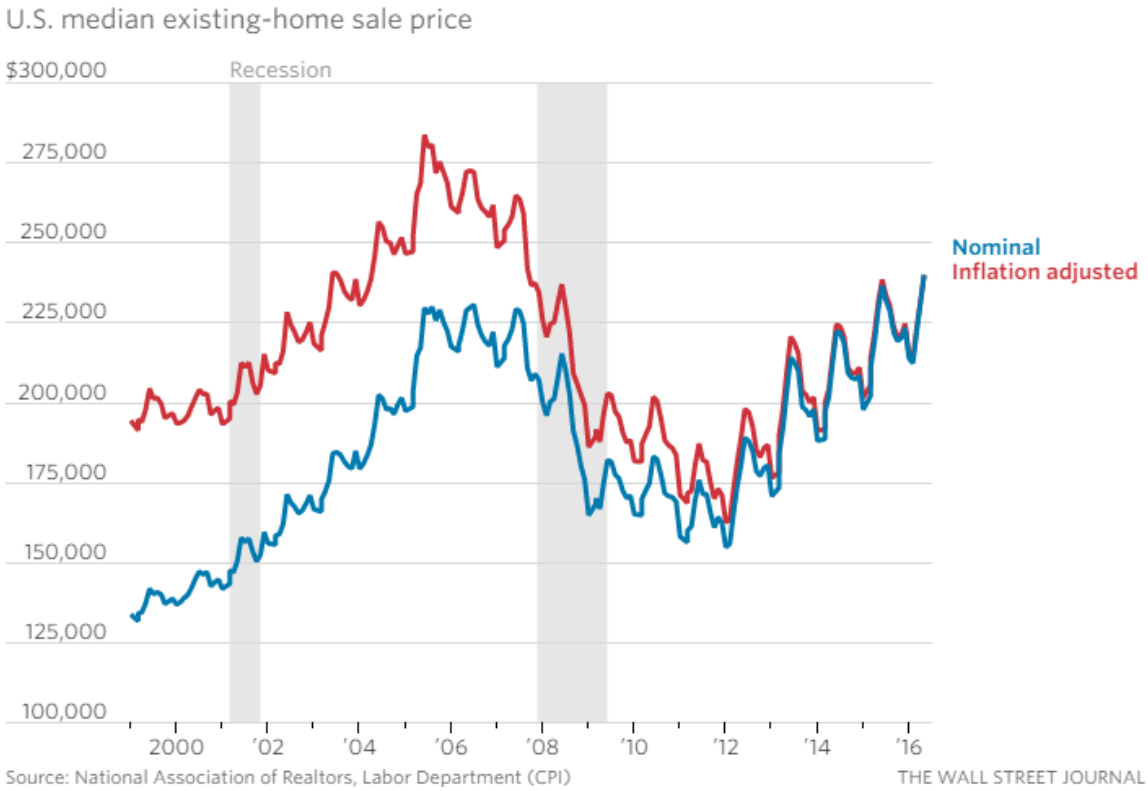
12.Deployment: Deploy the trained model to make predictions on new, unseen data, such as the price of a house that is not in the training dataset.

BLOCK DIAGRAM:



DATA SET IN HOUSE PRICE PREDICTION

Sr. No.	Details	Price	Bedrooms	Bathrooms	Living Room Sqft	Floors	View	Waterfront	Grade	Basement sqft
1	23534368	221456	3	2	1008	1.00	0	0	6	410
2	89756456	321234	4	3	1342	2.00	0	0	7	700
3	45767857	134000	2	2	2001	1.00	0	0	6	0
4	25756756	214679	3	1	1200	1.00	0	0	6	0
5	23445466	213245	3	1	980	1.00	0	0	8	0



PHASES OF DEVELOPMENT:

Phase 1: data collection and loading

Problem Statement:

Gather a dataset of house prices for classification

Activities:

Collect the database from a various sources or be downloaded from the platform like Kaggle.load the dataset into pandas dataframe using the read_csv function.

OUTPUT:

A pandas Dataframe containing the house price dataset.

Phase 2: Exploratory Data Analysis (EDA)

Problem Statement:

Understand the dataset's characteristics and distribution.

Activities:

- Visualize the distribution of spam and non-spam messages using a bar plot.
- Analyze the distribution of message length and create a histogram to understand how message length varies by label.
- Create a scatter plot to explore the relationship between message length and labels.

Output:

Data visualizations that provide insights into the dataset's characteristics and distribution.

Phase 3: Data Preprocessing

Problem Statement:

Prepare the data for machine learning by removing inconsistencies.

Activities:

Remove duplicate messages to avoid data redundancy.
Convert the text in the 'v2' column to lowercase to ensure uniformity.

Output:

A preprocessed dataset with duplicate messages removed and text converted to lowercase.

Phase 4: Basic Data Processing Methods

Problem Statement:

Provide an overview of the preprocessed dataset.

Activities:

- Display the first five and last five rows of the dataset to provide a glimpse of the data.
- Share additional information about the dataset, including its shape (number of rows and columns), size, and presence of missing values.
- Provide descriptive statistics of the dataset.
- Analyze the number of unique values in the 'v1' column (label) and display value counts.
- List the columns present in the dataset.
- Save the preprocessed dataset to a CSV file for future use.

Output:

A detailed overview of the preprocessed dataset and a saved CSV file.

Phase 5: Data Splitting

Problem Statement:

Prepare the dataset for model training and evaluation by splitting it into training and testing sets.

Activities:

- Use the train_test_split function from Scikit-Learn to create training and testing sets.

Output:

Training and testing datasets.

Phase 6: Text Vectorization using TF-IDF

Problem Statement:

Convert text data into numerical features suitable for machine learning models.

Activities:

- Create TF-IDF vectors from text data using the TfidfVectorizer from Scikit-Learn. This involves removing stop words, tokenizing text, and calculating TF-IDF scores.

Output:

TF-IDF vectors for the training and testing sets.

Phase 7: Model Training and Evaluation

Problem Statement:

Develop and evaluate a machine learning model for house price classification.

Activities:

- Train a Multinomial Naive Bayes model using the TF-IDF vectors.
- Make predictions on the test data.
- Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1 score.

Output:

Evaluation metrics and the trained machine learning model.

Phase 8: Additional Metrics

Problem Statement:

Further assess the model's performance using crossvalidation, confusion matrix, and a classification report.

Activities:

- Perform cross-validation to assess the model's generalization performance.
- Calculate the confusion matrix to understand the model's true positives, true negatives, false positives, and false negatives.
- Generate a classification report, which includes metrics like precision, recall, F1 score, and support.

Output:

Cross-validation scores, confusion matrix, and a classification report to provide additional insights into the model's performance.

These phases together cover the complete development cycle for spam classification, from data collection and preprocessing to model training and evaluation, including additional performance metrics.

MACHINE LEARNING ALGORITHM, MODEL TRAINING, AND EVALUATION METRICS:-

Machine Learning Algorithm:

Chosen Algorithm:

In the provided code, a Multinomial Naive Bayes (NB) classifier is chosen as the machine learning algorithm. Naive Bayes is a probabilistic algorithm that is particularly well-suited for text classification tasks like spam detection. The "Multinomial" variant of Naive Bayes is commonly used when dealing with discrete data like word counts in text documents.

Model Training:

- Training Data: The dataset is divided into training and testing sets using the train_test_split function. The training set is used to train the machine learning model, and the testing set is used to evaluate the model's performance.

- Feature Extraction: The text data is preprocessed, cleaned, and transformed into numerical feature vectors using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique. The TF-IDF vectors capture the importance of words in distinguishing spam from non-spam messages
- Model Training: The Multinomial Naive Bayes model is trained on the training data. During training, the model learns the probabilistic relationship between the TF-IDF features and the corresponding spam or non-spam labels. It estimates the likelihood of observing a particular set of features given the class label.

Evaluation Metrics:

- Accuracy: Accuracy measures the overall correctness of the model's predictions. It is the ratio of correctly classified instances to the total instances in the testing set.
- Precision: Precision is the ratio of true positives (correctly predicted spam) to the total predicted positives (true positives + false positives). It measures the ability of the model to make precise spam predictions.
- Recall: Recall is the ratio of true positives to the total actual positives (true positives + false negatives). It measures the model's ability to identify all actual spam messages.
- F1 Score: The F1 Score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is particularly useful when

there is an imbalance between the two classes (spam and nonspam).

Cross-Validation:

- Cross-validation is used to assess the performance of the model more robustly. In the code, a 5-fold crossvalidation is performed. It involves splitting the training data into five subsets, using four for training and one for validation iteratively. This helps in estimating the model's generalization performance and ensures that the model is not overfitting to the training data.

The choice of Multinomial Naive Bayes is suitable for text classification tasks, and the TF-IDF vectorization technique is a good way to represent text data numerically. The evaluation metrics provide a well-rounded view of the model's performance, taking into account both precision and recall, and cross-validation helps ensure the model's reliability.

1. Deep Learning Models: Innovative spam classifiers may employ deep learning models, such as recurrent neural networks (RNNs) or convolutional neural

networks (CNNs), to capture complex patterns in text data. These models have shown promising results in natural language processing tasks.

2. Word Embeddings: Advanced techniques like Word2Vec or GloVe pre-trained word embeddings can be used to represent words as dense vectors. These embeddings capture semantic relationships among words and can enhance the model's understanding of the text.
3. Ensemble Methods: Combining multiple models, such as Naive Bayes, Support Vector Machines (SVM), and deep learning models, into an ensemble can improve overall performance. Techniques like stacking or bagging can be used for this purpose.
4. Anomaly Detection: Unsupervised anomaly detection techniques, like one-class SVM or Isolation Forest, can be used to detect unusual patterns in messages, which may indicate spam.
5. Active Learning: Innovative approaches may incorporate active learning strategies to reduce the manual labeling effort required for building a labeled dataset. The model selects the most uncertain instances for manual labeling, optimizing the training process.

6. Temporal Analysis: Some spam messages exhibit temporal patterns. Innovative classifiers can analyze

message timestamps to identify spam messages that arrive at unusual times.

7. Graph-Based Models: Leveraging social network or communication graph structures can be a powerful technique to detect spam. Analyzing connections and interactions between users can help identify spam accounts or messages.
8. Transfer Learning: Models pre-trained on large text corpora, such as BERT or GPT, can be fine-tuned for spam classification. Transfer learning leverages the knowledge from general language tasks to improve performance on specific tasks.
9. Explainability Techniques: Innovative models can incorporate explainability techniques to provide insights into why a particular message is classified as spam. Interpretability is crucial for user trust and model transparency.
10. Handling Multimodal Data: In addition to text, some spam detection systems process images or other data types. Combining text and non-text features in a multimodal approach is an innovative approach.

SUBMISSION:

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_excel("HousePricePrediction.xlsx")
print(dataset.head(5)
```

OUTPUT :

	MSSubClass	MSZoning	LotArea	LotConfig	BldgType	OverallCond	YearBuilt
0	60	RL	8450	Inside	1Fam	5	2003
1	20	RL	9600	FR2	1Fam	8	1976
2	60	RL	11250	Inside	1Fam	5	2001
3	70	RL	9550	Corner	1Fam	5	1915
4	60	RL	14260	FR2	1Fam	5	2000

	YearRemodAdd	Exterior1st	BsmtFinSF2	TotalBsmtSF	SalePrice
0	2003	VinylSd	0.0	856.0	208500.0
1	1976	MetalSd	0.0	1262.0	181500.0
2	2002	VinylSd	0.0	920.0	223500.0
3	1970	Wd Sdng	0.0	756.0	140000.0
4	2000	VinylSd	0.0	1145.0	250000.0

Data Preprocessing

```
obj = (dataset.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:",len(object_cols))

int_ = (dataset.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:",len(num_cols))

fl = (dataset.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:",len(fl_cols))
```

Output:

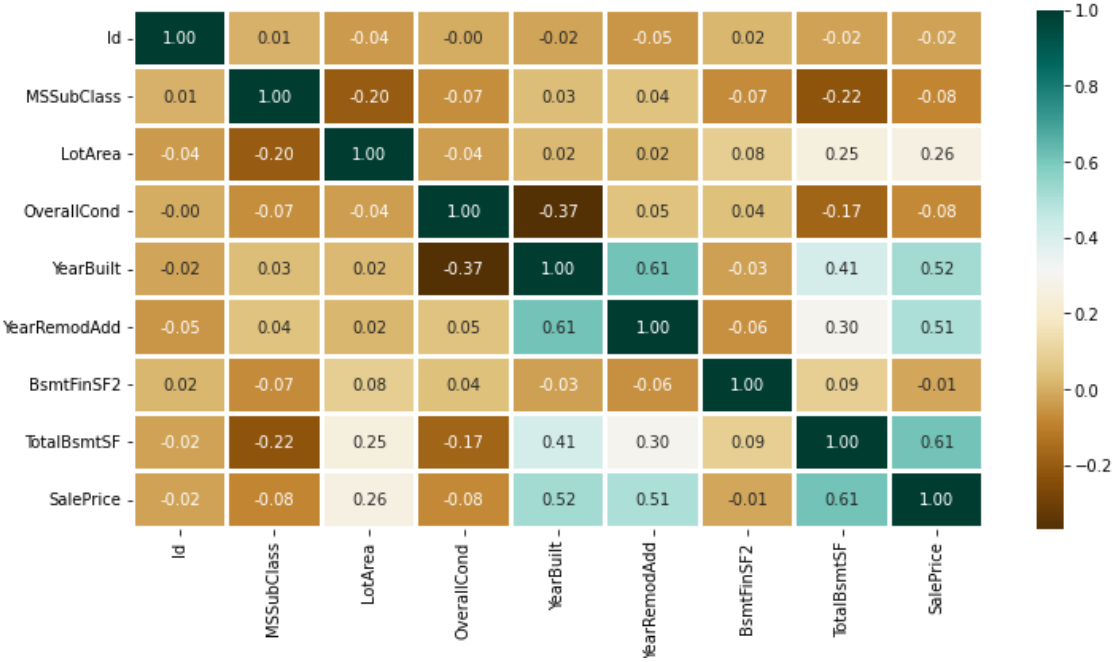
```
Categorical variables : 4

Integer variables : 6

Float variables : 3
```

Exploratory Data Analysis

```
plt.figure(figsize=(12, 6))
sns.heatmap(dataset.corr(),
            cmap = 'BrBG',
            fmt = '.2f',
            linewidths = 2,
            annot = True)
```

OUTPUT:

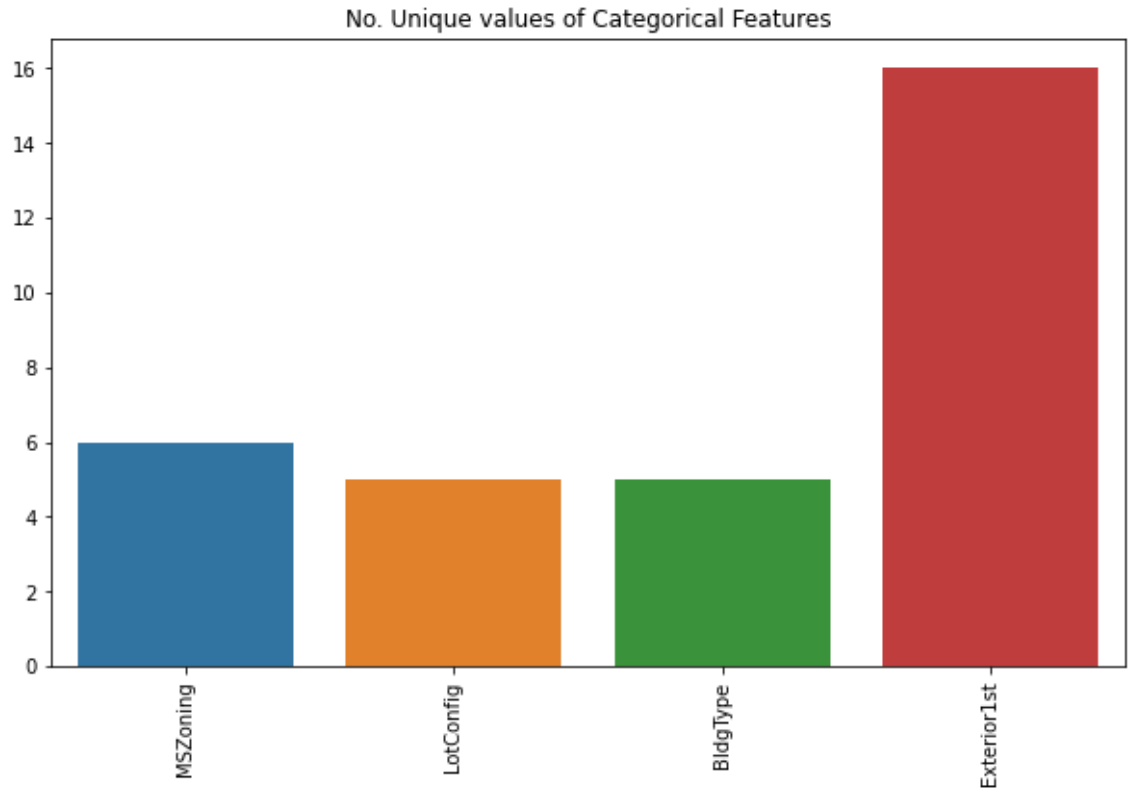
```
unique_values = []
for col in object_cols:
  unique_values.append(dataset[col].unique().size)
plt.figure(figsize=(10,6))
plt.title('No. Unique values of Categorical Features')
plt.xticks(rotation=90)
sns.barplot(x=object_cols,y=unique_values)
```




```
unique_values = []
for col in object_cols:
  unique_values.append(dataset[col].unique().size)
plt.figure(figsize=(10,6))
plt.title('No. Unique values of Categorical Features')
plt.xticks(rotation=90)
```

```
sns.barplot(x=object_cols,y=unique_values)
```

OUTPUT:



Data Cleaning

```
dataset['SalePrice'] = dataset['SalePrice'].fillna(
  dataset['SalePrice'].mean())
new_dataset = dataset.dropna()

new_dataset.isnull().sum()
```

OUTPUT:

```
MSSubClass      0
MSZoning        0
LotArea         0
LotConfig       0
BldgType        0
OverallCond     0
YearBuilt       0
YearRemodAdd    0
Exterior1st     0
BsmtFinSF2      0
TotalBsmtSF     0
SalePrice       0
dtype: int64
```

SVM – Support vector Machine

```
from sklearn import svm

from sklearn.svm import SVC
from sklearn.metrics import mean_absolute_percentage_error

model_SVR = svm.SVR()
model_SVR.fit(X_train,Y_train)
Y_pred = model_SVR.predict(X_valid)

print(mean_absolute_percentage_error(Y_valid, Y_pred))
```

Output :

```
0.18705129
```

CONCLUSION:

Clearly, SVM model is giving better accuracy as the mean absolute error is the least among all the other regressor models i.e. 0.18 approx. To get much better results ensemble learning techniques like bagging and boasting can also be used.By this project the house price prediction has predicted successfully.