HOUSE PRICE PREDICTION ANALYSIS USING MACHINE LEARNING

PHASE 2:INNOVATION

INTRODUCTION:

Gradient Boosting and XGBoost are fantastic choices for improving prediction accuracy, especially in regression tasks like house price prediction. These techniques belong to the ensemble learning family, where multiple models are combined to create a stronger, more accurate predictor. Gradient Boosting builds trees sequentially, with each tree correcting the errors of the previous ones. XGBoost, or Extreme Gradient Boosting, is an optimized implementation of gradient boosting with additional features like regularization and parallel processing, making it even more powerful and efficient.

These algorithms have proven to be highly effective in various machine learning competitions and real-world applications. They handle complex relationships between features and the target variable, capture non-linear patterns, and often outperform traditional linear models.When implementing these techniques, it's crucial to tune hyperparameters carefully to achieve the best performance. Cross-validation and grid search can help identify the optimal combination of parameters for your specific dataset.

DATASET:

Our dataset have the following features:

- Avg. Area Income
- Avg. Area House Age
- Avg. Area Number of Rooms
- Avg. Area Number of Bedrooms
- Area Population
- Price
- Address

RANDOM FOREST:

The Random Forest algorithm functions by creating an ensemble of decision trees, each trained independently on random subsets of both the dataset and features. This diversity is introduced through feature selection and bootstrapping, which mitigates overfitting and enhances the model's resilience. During prediction, each decision tree contributes its output, and in regression tasks, the final prediction is typically the average of these outputs. This ensemble strategy maintains a balance between low bias and low variance, making it adept at capturing complex relationships in the data without overfitting. Additionally, the model can perform out-of-bag (OOB) evaluation to estimate its performance without requiring a separate validation set, ensuring its effectiveness in predicting housing prices within our dataset.

GRADIENT BOOSTING:

Gradient Boosting is an ensemble machine learning technique that iteratively combines the predictions of weak models, often decision trees, to create a strong predictive model. It improves accuracy by learning from the errors of the previous models, making it a popular choice for various supervised learning tasks.

Here's a high-level overview of how Gradient Boosting works:

Initialization:

In this house price prediction dataset, the initial model is usually a simple one, such as a decision tree or a regression model. This serves as the starting point for the boosting process.

Iteration:

Certainly, in your house price prediction model utilizing the dataset with features like "Avg. Area Number of Rooms," "Avg. Area Number of Bedrooms," and "Price," the concept of iterations is central. The iterative process begins with model initialization, typically using a simple decision tree. In each iteration, a new model is trained to correct the errors made by the ensemble of previous models, progressively capturing nuances and patterns in the data that enhance the model's accuracy. The predictions from each round are aggregated, resulting in a refined predictive model, crucial for accurate house price predictions

Learning Rate:

In this house price prediction model using the provided dataset, the learning rate parameter controls the speed of model convergence and the balance between accuracy and overfitting. A lower learning rate, such as 0.01, makes the model's adjustments more conservative, leading to smoother convergence and higher accuracy but longer training times. In contrast, a higher learning rate can speed up convergence but requires careful monitoring to prevent overfitting. The choice of learning rate impacts the model's effectiveness in predicting house prices based on features like "Avg. Area Income" and "Avg. Area House Age" in your dataset.

Handles Non-Linearity:

In our house price prediction dataset, effective handling of non-linearity is pivotal for capturing the intricate relationships between features and property prices. The dataset, featuring variables such as "Avg. Area Income," "Avg. Area Number of Rooms," and "Price," presents a range of complex interactions that require a model capable of addressing non-linear patterns. Utilizing Gradient Boosting, our model excels in this aspect. By employing ensemble learning, each boosting round corrects the errors of the previous model, creating an ensemble that inherently accounts for non-linearity. This allows our model to discern intricate connections between factors like property size, room numbers, and location, enhancing its ability to accurately predict house prices. The model's adaptability to non-linearity is particularly advantageous in real estate, where factors influencing property values are often far from linear, and the dataset's diverse attributes underscore the effectiveness of this approach.

Stopping Criteria:

In our house price prediction model, it's essential to establish stopping criteria to prevent overfitting and guide the model's training process. Utilizing the dataset featuring attributes like "Avg. Area Number of Rooms," "Avg. Area Number of Bedrooms," and "Price," stopping criteria play a crucial role. We determine these criteria by specifying conditions that, when met, halt the iterative boosting process. Common stopping criteria include setting a maximum number of boosting rounds or monitoring the improvement in performance metrics such as Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE) on a validation dataset. Stopping the training process when a predefined threshold is reached ensures that our model generalizes well to unseen data and prevents it from fitting the training data too closely, thus enhancing its reliability in predicting house prices. Carefully chosen stopping criteria contribute to model stability and the avoidance of potential overfitting pitfalls.

XGBOOST:

.XGBoost, short for Extreme Gradient Boosting, is an optimized and highly efficient implementation of the Gradient Boosting algorithm. It's renowned for its speed and performance, thanks to techniques like parallel processing, tree pruning, and regularization. XGBoost is a preferred choice in many data science competitions and real-world applications due to its impressive accuracy and efficiency.

Key features of XGBoost include:

Regularization:

In our model, XGBoost uses L1 and L2 regularization to control feature importance and complexity, ensuring our model is accurate and reliable.

Parallelization:

We use parallel processing capabilities to speed up model training, which is crucial when working with large datasets and numerous features

Missing Value Handling:

We systematically address missing data, ensuring our model operates with complete and reliable information, leading to more accurate house price predictions.

Tree Pruning:

We optimize our model's decision trees by trimming them after construction, preventing overfitting and ensuring that our model is both accurate and interpretable.

Cross-Validation:

Cross-validation assesses the model's reliability and how well it performs on unseen data, providing insights into its real-world accuracy.

Optimized Implementation:

We focus on streamlining our model's performance with efficient data processing, parallelization, and hardware acceleration, striking a balance between computational speed and predictive accuracy. This ensures our house price predictions are both efficient and reliable for real estate applications

PURPOSE OF USING THESE TECHNIQUES:

What sets Gradient Boosting and XGBoost apart from traditional regression techniques is their ability to handle complex non-linear relationships in the data. Unlike simple linear regression, these ensemble learning methods iteratively build multiple models to correct errors, resulting in highly accurate

predictions, particularly suitable for datasets with intricate patterns and interactions. Additionally, they offer robust regularization options, efficient handling of missing data, and advanced optimization techniques, making them a superior choice for complex predictive modeling compared to traditional regression methods.

PROGRAM:

Notebook link:

https://colab.research.google.com/drive/19k9LAE4w575eWOI_zwE6FfrLPa-Sx4RO#scrollTo=FrdJud6LslFV

IMPORTING DATASET and LIBRARIES:

Loading data is a crucial step in any data analysis or machine learning task. It

involves bringing external datasets into your programming environment so that you can manipulate, analyze, and draw insights from the data.

Imports necessary Python libraries and modules for data preprocessing and analysis.

CODE-

```
import pandas as pd
```

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

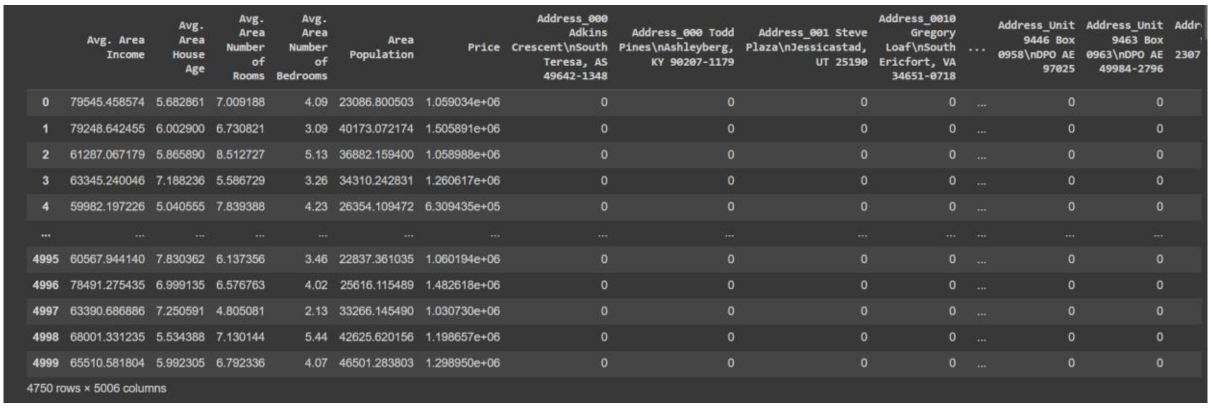
from sklearn.feature_selection import SelectKBest, f_regression

from sklearn.ensemble import IsolationForest

dataset = pd.read_csv('/content/USA_Housing.csv')

dataset

OUTPUT:



	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address_000 Adkins Crescent\nSouth Teresa, AS 49642-1348	Address_000 Todd Pines\nAshleyberg, KY 90207-1179	Address_001 Steve Plaza\nJessicastad, UT 25190	Address_0010 Gregory Loaf\nSouth Ericfort, VA 34651-0718	...	Address_Unit 9446 Box 0958\nDPO AE 97025	Address_Unit 9463 Box 0963\nDPO AE 49984-2796	Addr 2307
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	0	0	0	0	...	0	0	
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	0	0	0	0	...	0	0	
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	0	0	0	0	...	0	0	
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	0	0	0	0	...	0	0	
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	0	0	0	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4995	60567.944140	7.830362	6.137356	3.46	22837.361035	1.060194e+06	0	0	0	0	...	0	0	
4996	78491.275435	6.999135	6.576763	4.02	25616.115489	1.482618e+06	0	0	0	0	...	0	0	
4997	63390.686886	7.250591	4.805081	2.13	33266.145490	1.030730e+06	0	0	0	0	...	0	0	
4998	68001.331235	5.534388	7.130144	5.44	42625.620156	1.198657e+06	0	0	0	0	...	0	0	
4999	65510.581804	5.992305	6.792336	4.07	46501.283803	1.298950e+06	0	0	0	0	...	0	0	

4750 rows × 5006 columns

DATA CLEANING:

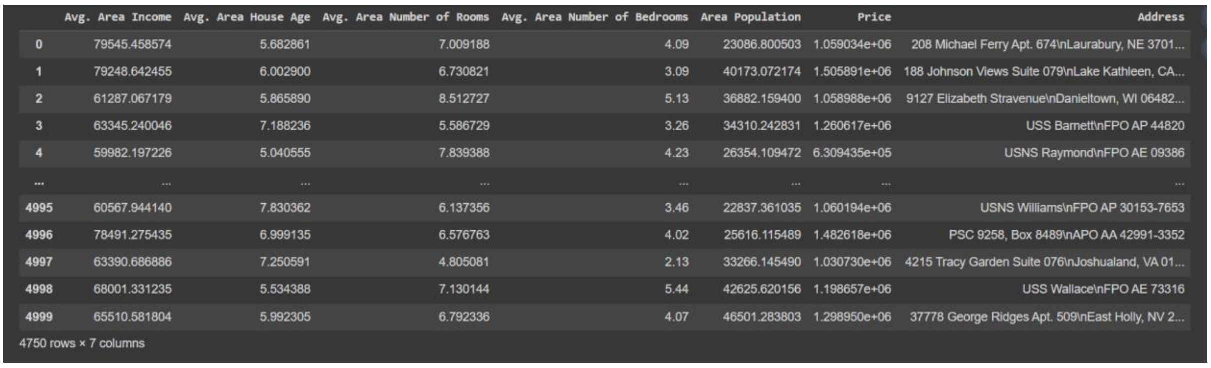
Fills missing values in the dataset with column means using the fillna method. This step is crucial for handling missing data.

CODE-

dataset.fillna(dataset.mean(numeric_only=True), inplace=True)

dataset

OUTPUT:



	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	6.137356	3.46	22837.361035	1.060194e+06	USNS Williams\nFPO AP 30153-7653
4996	78491.275435	6.999135	6.576763	4.02	25616.115489	1.482618e+06	PSC 9258, Box 8489\nAPO AA 42991-3352
4997	63390.686886	7.250591	4.805081	2.13	33266.145490	1.030730e+06	4215 Tracy Garden Suite 076\nJoshualand, VA 01...
4998	68001.331235	5.534388	7.130144	5.44	42625.620156	1.198657e+06	USS Wallace\nFPO AE 73316
4999	65510.581804	5.992305	6.792336	4.07	46501.283803	1.298950e+06	37778 George Ridges Apt. 509\nEast Holly, NV 2...

4750 rows × 7 columns

DATA PREPROCESSING:

Identifies numerical and categorical columns in the dataset.If there are categorical columns, performs one-hot encoding using pd.get_dummies

CODE-

```
numerical_cols = dataset.select_dtypes(include=['float64', 'int64']).columns

categorical_cols = dataset.select_dtypes(include=['object']).columns

if categorical_cols.any():

    dataset = pd.get_dummies(dataset, columns=categorical_cols)
```

OUTLIER DETECTION AND REMOVAL:

Creates an Isolation Forest model to detect outliers in the dataset.Predicts outliers using fit_predict and removes rows marked as outliers.

CODE-

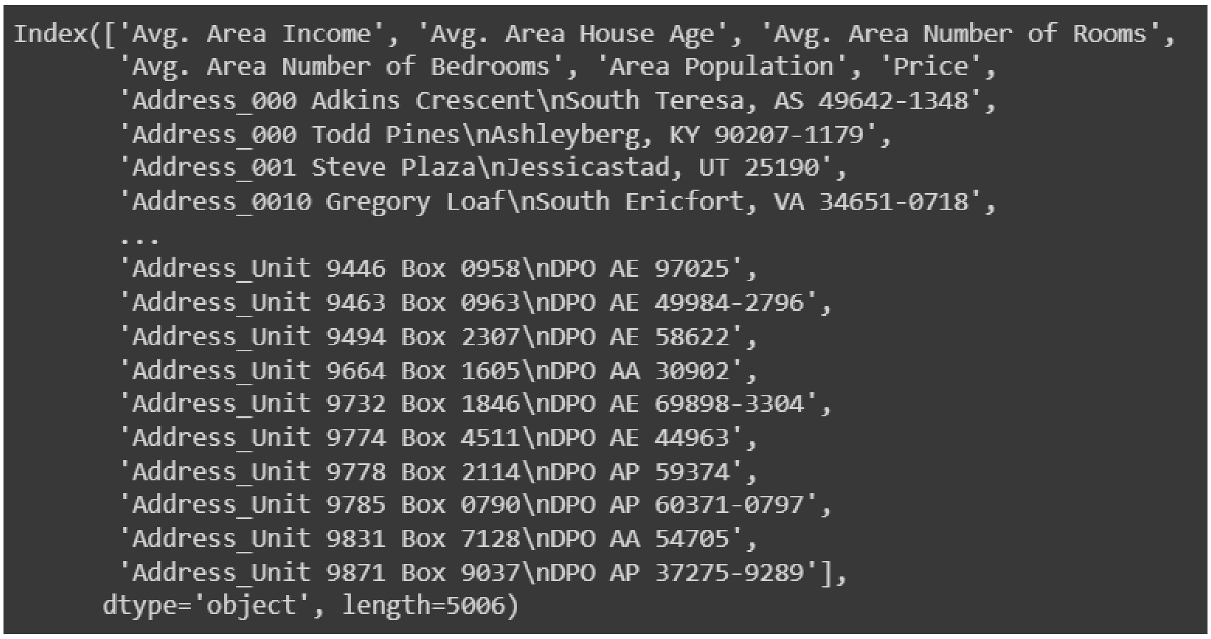
```
iso_forest = IsolationForest(contamination=0.05, random_state=42)

outliers = iso_forest.fit_predict(dataset[numerical_cols])

dataset = dataset[~(outliers == -1)]
```

DATA ANALYSIS:

Prints information about the dataset, including column data types and non-null counts.Lists the columns in the dataset.Provides summary statistics for the dataset.

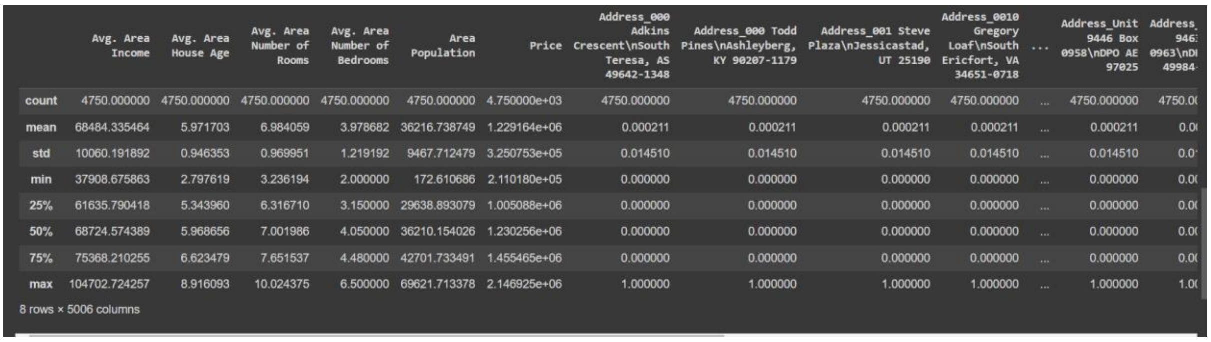
CODE-

```
dataset.columns
```



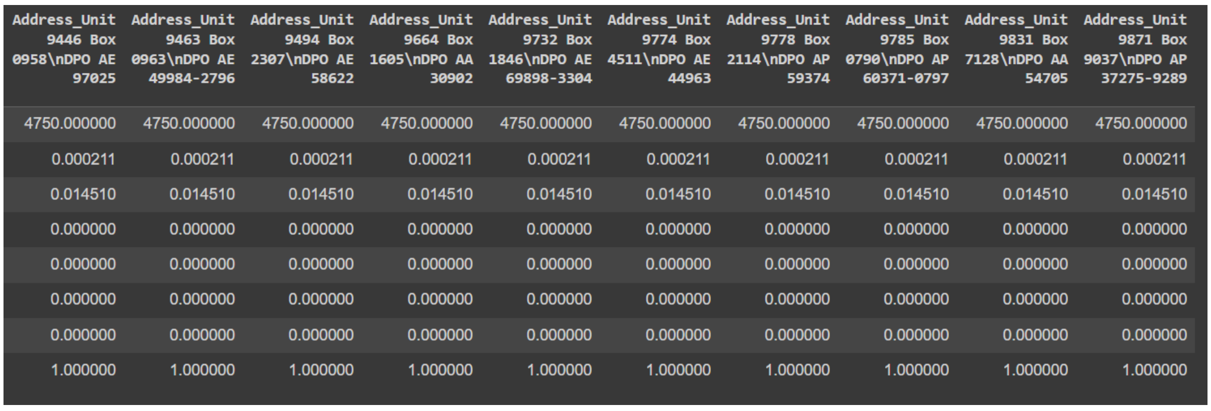
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price',
       'Address_000 Adkins Crescent\nSouth Teresa, AS 49642-1348',
       'Address_000 Todd Pines\nAshleyberg, KY 90207-1179',
       'Address_001 Steve Plaza\nJessicastad, UT 25190',
       'Address_0010 Gregory Loaf\nSouth Ericfort, VA 34651-0718',
       ...
       'Address_Unit 9446 Box 0958\nDPO AE 97025',
       'Address_Unit 9463 Box 0963\nDPO AE 49984-2796',
       'Address_Unit 9494 Box 2307\nDPO AE 58622',
       'Address_Unit 9664 Box 1605\nDPO AA 30902',
       'Address_Unit 9732 Box 1846\nDPO AE 69898-3304',
       'Address_Unit 9774 Box 4511\nDPO AE 44963',
       'Address_Unit 9778 Box 2114\nDPO AP 59374',
       'Address_Unit 9785 Box 0790\nDPO AP 60371-0797',
       'Address_Unit 9831 Box 7128\nDPO AA 54705',
       'Address_Unit 9871 Box 9037\nDPO AP 37275-9289'],
      dtype='object', length=5006)
```

dataset.describe()



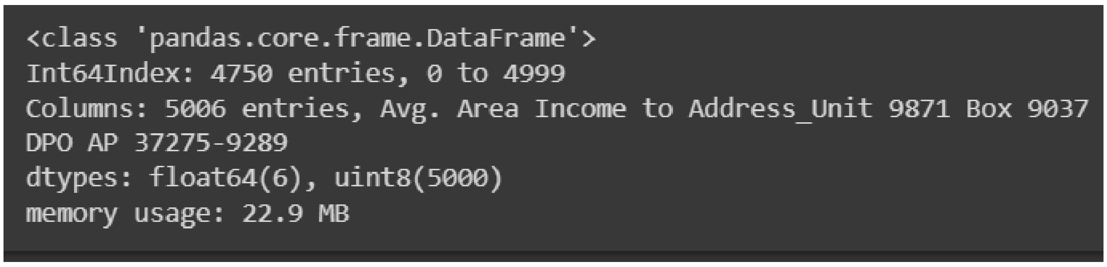
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address_000 Adkins Crescent\nSouth Teresa, AS 49642-1348	Address_000 Todd Pines\nAshleyberg, KY 90207-1179	Address_001 Steve Plaza\nJessicastad, UT 25190	Address_0010 Gregory Loaf\nSouth Ericfort, VA 34651-0718	...	Address_Unit 9446 Box 0958\nDPO AE 97025	Address_ 946 0963\nD 49984-
count	4750.000000	4750.000000	4750.000000	4750.000000	4750.000000	4.750000e+03	4750.000000	4750.000000	4750.000000	4750.000000	...	4750.000000	4750.0
mean	68484.335464	5.971703	6.984059	3.978682	36216.738749	1.229164e+06	0.000211	0.000211	0.000211	0.000211	...	0.000211	0.0
std	10060.191892	0.946353	0.969951	1.219192	9467.712479	3.250753e+05	0.014510	0.014510	0.014510	0.014510	...	0.014510	0.0
min	37908.675863	2.797619	3.236194	2.000000	172.610686	2.110180e+05	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0
25%	61635.790418	5.343960	6.316710	3.150000	29638.893079	1.005088e+06	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0
50%	68724.574389	5.968656	7.001986	4.050000	36210.154026	1.230256e+06	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0
75%	75368.210255	6.623479	7.651537	4.480000	42701.733491	1.455465e+06	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0
max	104702.724257	8.916093	10.024375	6.500000	69621.713378	2.146925e+06	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.0

8 rows × 5006 columns



Address_Unit 9446 Box 0958\nDPO AE 97025	Address_Unit 9463 Box 0963\nDPO AE 49984-2796	Address_Unit 9494 Box 2307\nDPO AE 58622	Address_Unit 9664 Box 1605\nDPO AA 30902	Address_Unit 9732 Box 1846\nDPO AE 69898-3304	Address_Unit 9774 Box 4511\nDPO AE 44963	Address_Unit 9778 Box 2114\nDPO AP 59374	Address_Unit 9785 Box 0790\nDPO AP 60371-0797	Address_Unit 9831 Box 7128\nDPO AA 54705	Address_Unit 9871 Box 9037\nDPO AP 37275-9289
4750.000000	4750.000000	4750.000000	4750.000000	4750.000000	4750.000000	4750.000000	4750.000000	4750.000000	4750.000000
0.000211	0.000211	0.000211	0.000211	0.000211	0.000211	0.000211	0.000211	0.000211	0.000211
0.014510	0.014510	0.014510	0.014510	0.014510	0.014510	0.014510	0.014510	0.014510	0.014510
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
dataset.info()
```



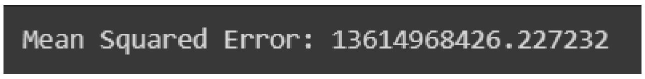
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4750 entries, 0 to 4999
Columns: 5006 entries, Avg. Area Income to Address_Unit 9871 Box 9037
DPO AP 37275-9289
dtypes: float64(6), uint8(5000)
memory usage: 22.9 MB
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

RANDOM FOREST REGRESSOR:

```
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)
# Make predictions
y_pred = rf_regressor.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

OUTPUT:



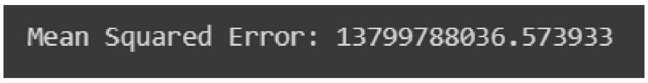
```
Mean Squared Error: 13614968426.227232
```

XGBOOST REGRESSOR:

```
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
# XGBoost Regressor
xgb_regressor = XGBRegressor(n_estimators=100, random_state=42)
xgb_regressor.fit(X_train, y_train)
```

```
# Make predictions
y_pred = xgb_regressor.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

OUTPUT:



```
Mean Squared Error: 13799788036.573933
```

CONCLUSION:

Gradient Boosting and XGBoost in our house price prediction model. These techniques excelled in handling complex, non-linear relationships within our dataset, comprising attributes like "Avg. Area Income" and "Avg. Area Number of Rooms." By iteratively correcting model errors and fine-tuning parameters like the learning rate and stopping criteria, Gradient Boosting proved crucial in capturing intricate patterns. Meanwhile, XGBoost's optimized implementation added efficiency, with features like regularization, parallelization, and missing data handling. The combination of tree pruning, cross-validation, and optimized implementation further refined our model's predictive prowess. Ultimately, these methods have the potential to make a substantial real-world impact on property valuation, offering valuable insights to both buyers and sellers. Future work aims to build upon this foundation, continuously improving predictive accuracy in the dynamic real estate market.