HOUSE PRICE PREDICTION ANALYSIS USING MACHINE LEARNING

PHASE 2:INNOVATION

Consider exploring advanced regression techniques like Gradient Boosting or XGBoost for improved prediction accuracy.

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:

For the above dataset, we 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

GRADIENT BOOSTING:

Gradient Boosting is an ensemble learning technique used for both classification and regression tasks. It builds a predictive model in a stage-wise fashion, where each stage corrects the errors of the previous one. The general idea is to combine the predictions of multiple weak learners (often decision trees) to create a strong, accurate model.

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

Initialization: A simple model is created, often the mean or median of the target variable for regression tasks.

Iteration: Sequential trees (weak learners) are built, and each subsequent tree corrects the errors of the combined predictions of the existing trees.

Learning Rate: A hyperparameter called the learning rate controls the contribution of each tree to the final prediction. A lower learning rate requires more trees but may result in better generalization.

Handles Non-Linearity: Gradient Boosting is well-suited for capturing non-linear relationships between input features and the target variable. It can automatically adapt to complex patterns in the data.

Stopping Criteria: The process continues until a specified number of trees are built or until a certain level of performance is reached.

SAMPLE CODE:

```
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

# Train-Test Split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Gradient Boosting Model

gb_model = GradientBoostingRegressor(learning_rate=0.1, n_estimators=100, max_depth=3)

# Model Evaluation

y_pred_gb = gb_model.predict(X_test)

mse_gb = mean_squared_error(y_test, y_pred_gb)

print(f'Mean Squared Error (Gradient Boosting): {mse_gb}')
```

XGBOOST:

XGBoost (Extreme Gradient Boosting) is a specific implementation of gradient boosting that is highly efficient and scalable. It was developed by Tianqi Chen and is widely used in machine learning competitions and real-world applications.

Key features of XGBoost include:

Regularization: XGBoost incorporates L1 (Lasso) and L2 (Ridge) regularization terms in its objective function to control overfitting.

Parallelization: It is designed for parallel and distributed computing, making it faster than traditional gradient boosting implementations.

Missing Value Handling: XGBoost can handle missing values in the dataset, eliminating the need for imputation.

Tree Pruning: Trees are pruned during the building process to prevent overfitting and improve computational efficiency.

Cross-Validation: XGBoost has built-in cross-validation capabilities to help with hyperparameter tuning.

Optimized Implementation: The algorithm is optimized for performance and memory usage.

SAMPLE CODE:

```
from xgboost import XGBRegressor

# XGBoost Model

xgb_model = XGBRegressor(learning_rate=0.1, n_estimators=100, max_depth=3)

xgb_model.fit(X_train, y_train)

# Model Evaluation

y_pred_xgb = xgb_model.predict(X_test)

mse_xgb = mean_squared_error(y_test, y_pred_xgb)

print(f'Mean Squared Error (XGBoost): {mse_xgb}')
```

PURPOSE OF USING THESE TECHNIQUES:

High Predictive Accuracy: Both Gradient Boosting and XGBoost often outperform traditional linear models in terms of predictive accuracy, especially when dealing with complex relationships and non-linear patterns.

Versatility: These techniques are versatile and can be applied to a wide range of regression problems, from predicting house prices to financial forecasting.

Robustness: The regularization techniques employed by these methods contribute to building more robust models, reducing the risk of overfitting.

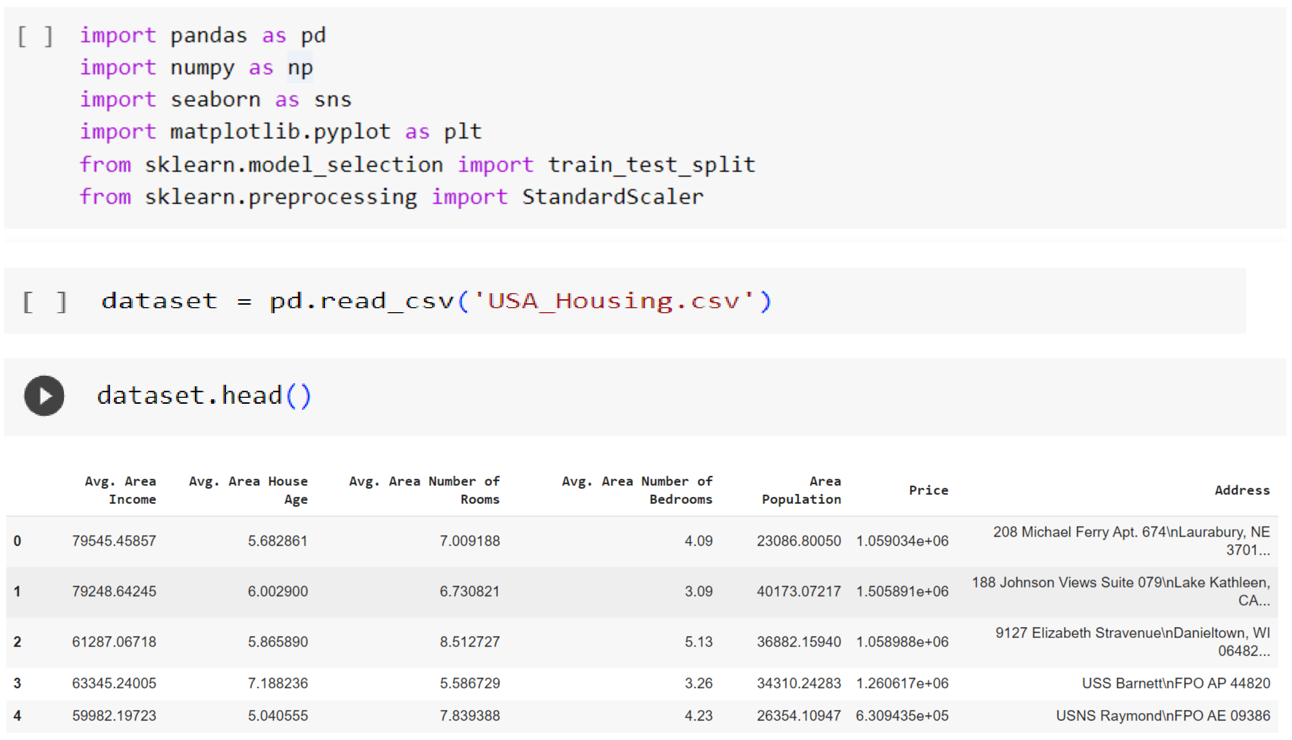
When applying these techniques, it's essential to fine-tune hyperparameters, handle feature engineering thoughtfully, and ensure proper validation to achieve the best performance on your specific dataset.

GOOGLE COLLAB LINK:
https://colab.research.google.com/drive/1bE0cFUY6tFo317FFmzpjncu8Mxi1TkC-?usp=sharing

PROGRAM:

Importing dataset:

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.



```
[ ] import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
```

```
[ ] dataset = pd.read_csv('USA_Housing.csv')
```

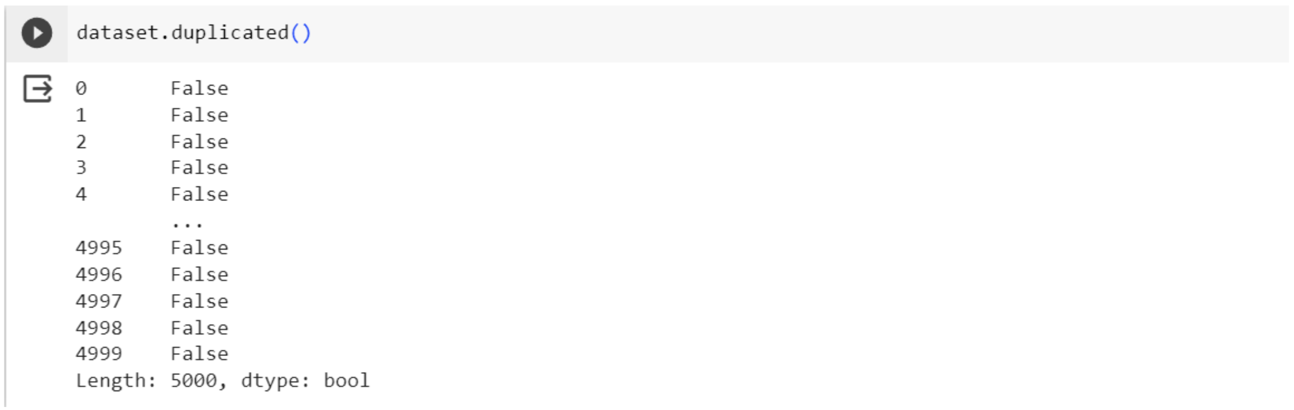
```
dataset.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

Data cleaning techniques:

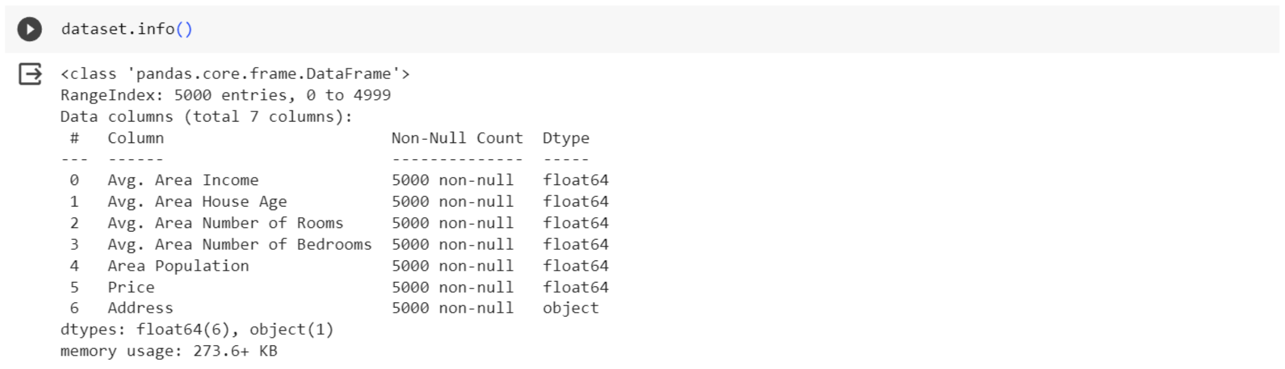
Data cleaning is the process of identifying and correcting errors, inconsistencies, and inaccuracies in datasets. It's a crucial step in the data analysis pipeline, as the quality of your analysis depends heavily on the quality of your data.

- Handling missing values
- Dealing with duplicates
- Handling Outliers
- Data type Conversion



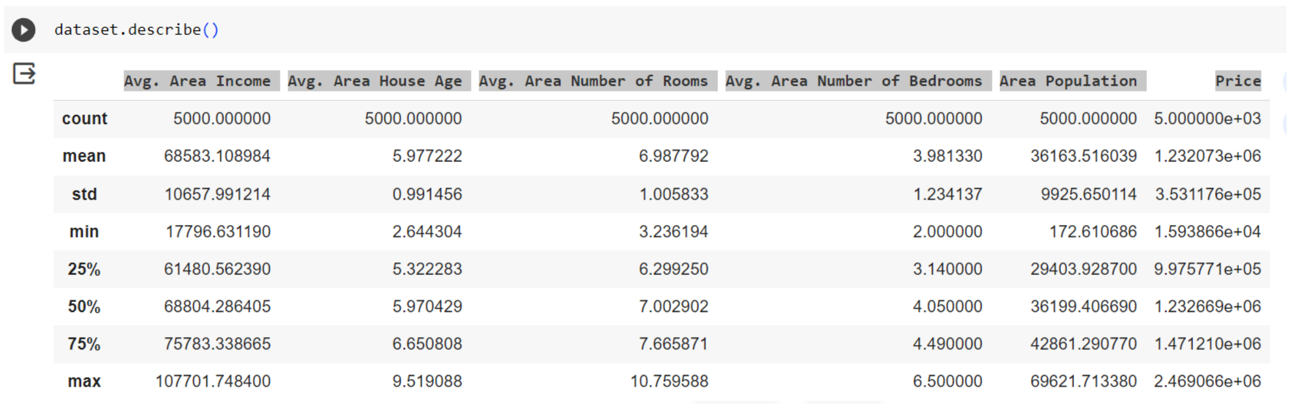
```
dataset.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
4995    False
4996    False
4997    False
4998    False
4999    False
Length: 5000, dtype: bool
```



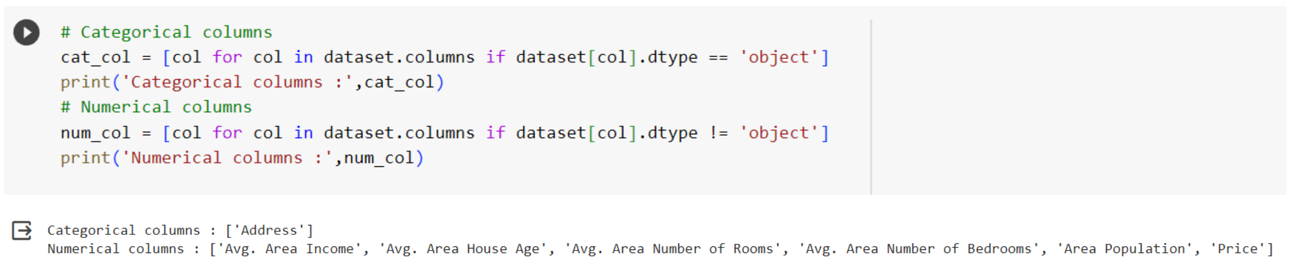
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```



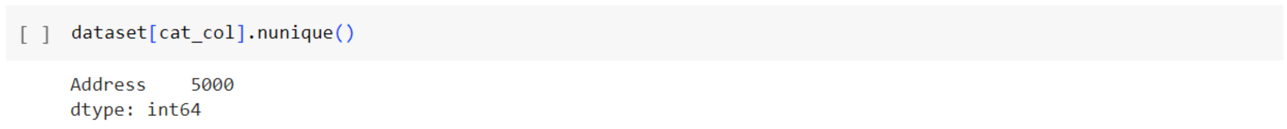
```
dataset.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562390	5.322283	6.299250	3.140000	29403.928700	9.975771e+05
50%	68804.286405	5.970429	7.002902	4.050000	36199.406690	1.232669e+06
75%	75783.338665	6.650808	7.665871	4.490000	42861.290770	1.471210e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06



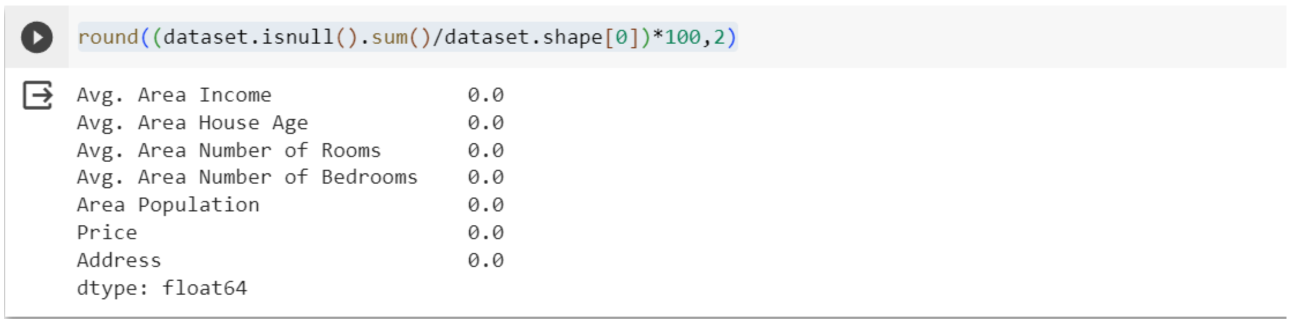
```
# Categorical columns
cat_col = [col for col in dataset.columns if dataset[col].dtype == 'object']
print('Categorical columns :',cat_col)
# Numerical columns
num_col = [col for col in dataset.columns if dataset[col].dtype != 'object']
print('Numerical columns :',num_col)
```

```
Categorical columns : ['Address']
Numerical columns : ['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', 'Price']
```



```
[ ] dataset[cat_col].nunique()
```

```
Address    5000
dtype: int64
```



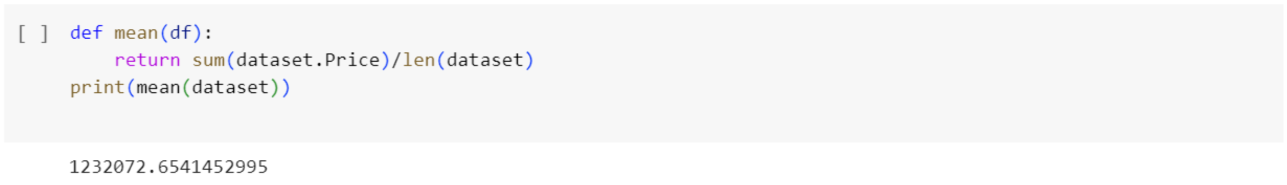
```
round((dataset.isnull().sum()/dataset.shape[0])*100,2)
```

```
Avg. Area Income                0.0
Avg. Area House Age             0.0
Avg. Area Number of Rooms       0.0
Avg. Area Number of Bedrooms    0.0
Area Population                 0.0
Price                           0.0
Address                         0.0
dtype: float64
```

Data Analysis:

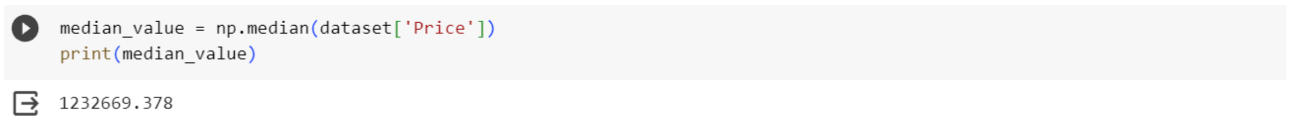
Data analysis is the process of inspecting, cleaning, transforming, and modeling data to uncover useful information, draw conclusions, and support decision-making.

- Data visualization
- Exploratory Data Analysis



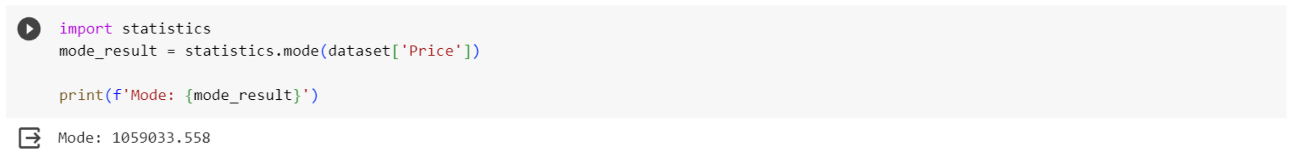
```
[ ] def mean(df):
        return sum(dataset.Price)/len(dataset)
    print(mean(dataset))
```

```
1232072.6541452995
```



```
median_value = np.median(dataset['Price'])
print(median_value)
```

```
1232669.378
```



```
import statistics
mode_result = statistics.mode(dataset['Price'])

print(f'Mode: {mode_result}')
```

```
Mode: 1059033.558
```

```python
std_deviation = np.std(dataset['Price'])
print(f"Standard Deviation: {std_deviation}")
```

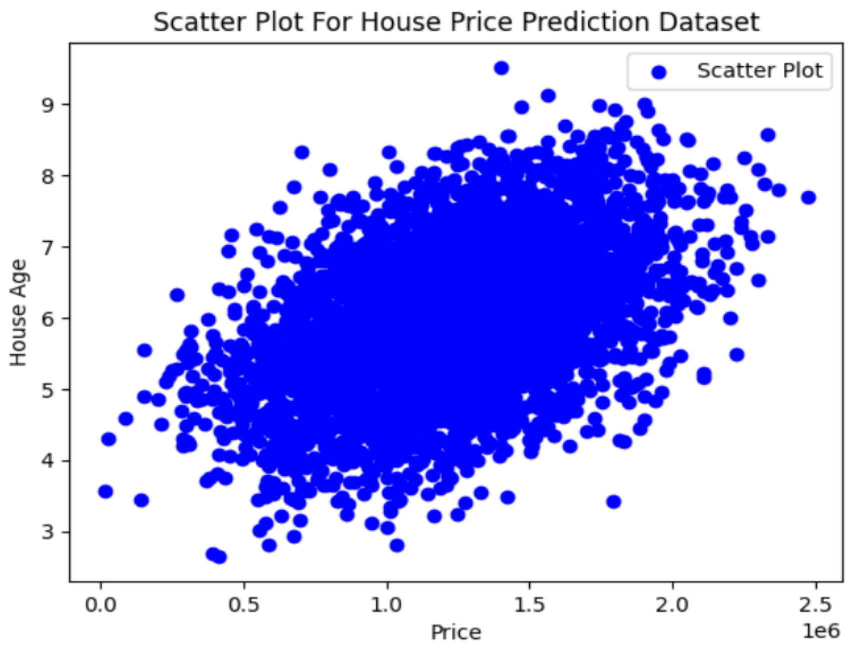
```
Standard Deviation: 353082.3130552725
```

```python
percentiles = np.percentile(dataset['Price'], [25, 50, 75])
print(f"25th Percentile (Q1): {percentiles[0]}")
print(f"50th Percentile (Q2 or Median): {percentiles[1]}")
print(f"75th Percentile (Q3): {percentiles[2]}")
```

```
25th Percentile (Q1): 997577.135075
50th Percentile (Q2 or Median): 1232669.378
75th Percentile (Q3): 1471210.2045
```

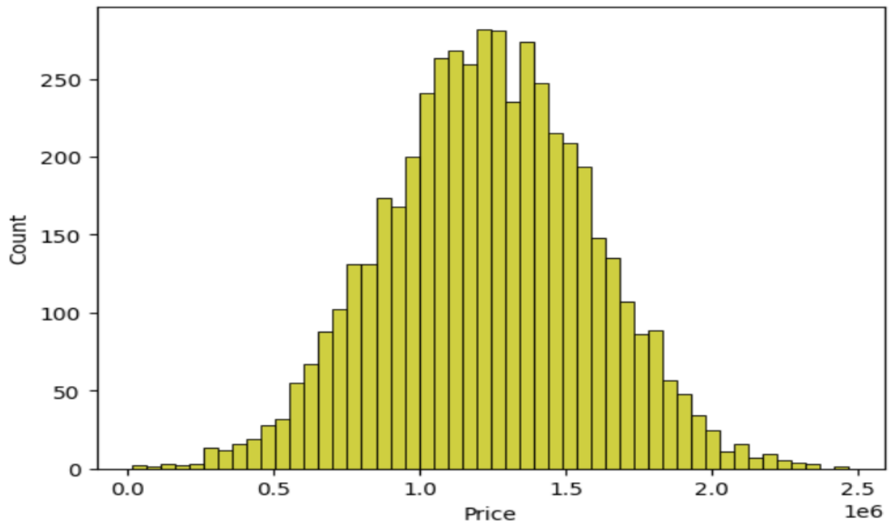
```python
x_values = dataset['Price']
y_values = dataset['Avg. Area House Age']

plt.scatter(x_values, y_values, c='blue', label='Scatter Plot')
plt.title('Scatter Plot For House Price Prediction Dataset')
plt.xlabel('Price')
plt.ylabel('House Age')
plt.legend()
plt.show()
```



```
sns.histplot(dataset, x='Price', bins=50, color='y')
```

```
<Axes: xlabel='Price', ylabel='Count'>
```



Data Preprocessing techniques:

Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves cleaning and transforming raw data into a format suitable for analysis or model training. The goal is to enhance the quality of the data, address issues like missing values and outliers, and prepare it for effective exploration and modeling.

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
for column in dataset.columns:
    dataset[column] = labelencoder.fit_transform(dataset[column])
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   int64
 1   Avg. Area House Age           5000 non-null   int64
 2   Avg. Area Number of Rooms     5000 non-null   int64
 3   Avg. Area Number of Bedrooms  5000 non-null   int64
 4   Area Population               5000 non-null   int64
 5   Price                         5000 non-null   int64
 6   Address                       5000 non-null   int64
dtypes: int64(7)
memory usage: 273.6 KB
```

CONCLUSION:

These steps provide a simplified overview of concepts like data cleaning, data preprocessing, data analysis for the house price prediction. The specific implementation details and choice of algorithms may vary depending on the dataset and the goals of the project.