lr

September 14, 2023

1 History of the Linear Regression

Francis Galton, a British scientist and cousin of Charles Darwin, made significant contributions to the field of statistics and is considered one of the pioneers of regression analysis. Galton's work on the relationship between parents' and children's heights laid the foundation for the concept of regression to the mean.

In his research, Galton observed that while there was a tendency for the height of a son to be related to the height of his father (suggesting a positive correlation), the son's height was also influenced by other factors, such as genetic variation and environmental factors. Galton coined the term "regression toward mediocrity" to describe the phenomenon where extreme values (either exceptionally tall or short) in a parent's height tended to be followed by offspring who were closer to the population average.

This observation laid the groundwork for the development of regression analysis, a statistical method used to model the relationship between variables, predict outcomes, and understand patterns in data. Today, regression analysis is a fundamental tool in various fields, including economics, social sciences, and machine learning. It allows researchers and data analysts to explore relationships between variables and make predictions based on data patterns.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib as plt
import seaborn as sns
```

[2]:
```
%matplotlib inline
```

[3]:
```
df= pd.read_csv('USA_Housing.csv')
```

1.1 Dataset Description

The dataset contains information related to real estate properties. Here is a breakdown of the dataset columns:

- **Avg. Area Income**: The average income of residents in the area where the property is located. (Numerical)
- **Avg. Area House Age**: The average age of houses in the area. (Numerical)
- **Avg. Area Number of Rooms**: The average number of rooms in houses in the area. (Numerical)

- **Avg. Area Number of Bedrooms**: The average number of bedrooms in houses in the area. (Numerical)
- **Area Population**: The population of the area where the property is located. (Numerical)
- **Price**: The price of the real estate property. (Numerical, Target Variable)
- **Address**: The address or location information for each property. (Categorical/Text)

This dataset contains both numerical and categorical features. The 'Price' column is the target variable that you may want to predict using regression techniques.

[4]:
```
df.head()
```

[4]:
```
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046             7.188236                   5.586729
4      59982.197226             5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
2                          5.13     36882.159400  1.058988e+06
3                          3.26     34310.242831  1.260617e+06
4                          4.23     26354.109472  6.309435e+05

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701…
1  188 Johnson Views Suite 079\nLake Kathleen, CA…
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482…
3                          USS Barnett\nFPO AP 44820
4                         USNS Raymond\nFPO AE 09386
```

[5]:
```
df.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
```

Describe function is used here to describe the data in summary. It will only shows numerical column.

```
[6]: df.describe()
```

```
[6]:        Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
     count       5000.000000          5000.000000                5000.000000
     mean       68583.108984             5.977222                   6.987792
     std        10657.991214             0.991456                   1.005833
     min        17796.631190             2.644304                   3.236194
     25%        61480.562388             5.322283                   6.299250
     50%        68804.286404             5.970429                   7.002902
     75%        75783.338666             6.650808                   7.665871
     max       107701.748378             9.519088                  10.759588

            Avg. Area Number of Bedrooms  Area Population         Price
     count                   5000.000000      5000.000000  5.000000e+03
     mean                       3.981330     36163.516039  1.232073e+06
     std                        1.234137      9925.650114  3.531176e+05
     min                        2.000000       172.610686  1.593866e+04
     25%                        3.140000     29403.928702  9.975771e+05
     50%                        4.050000     36199.406689  1.232669e+06
     75%                        4.490000     42861.290769  1.471210e+06
     max                        6.500000     69621.713378  2.469066e+06
```

2 To reference the columns

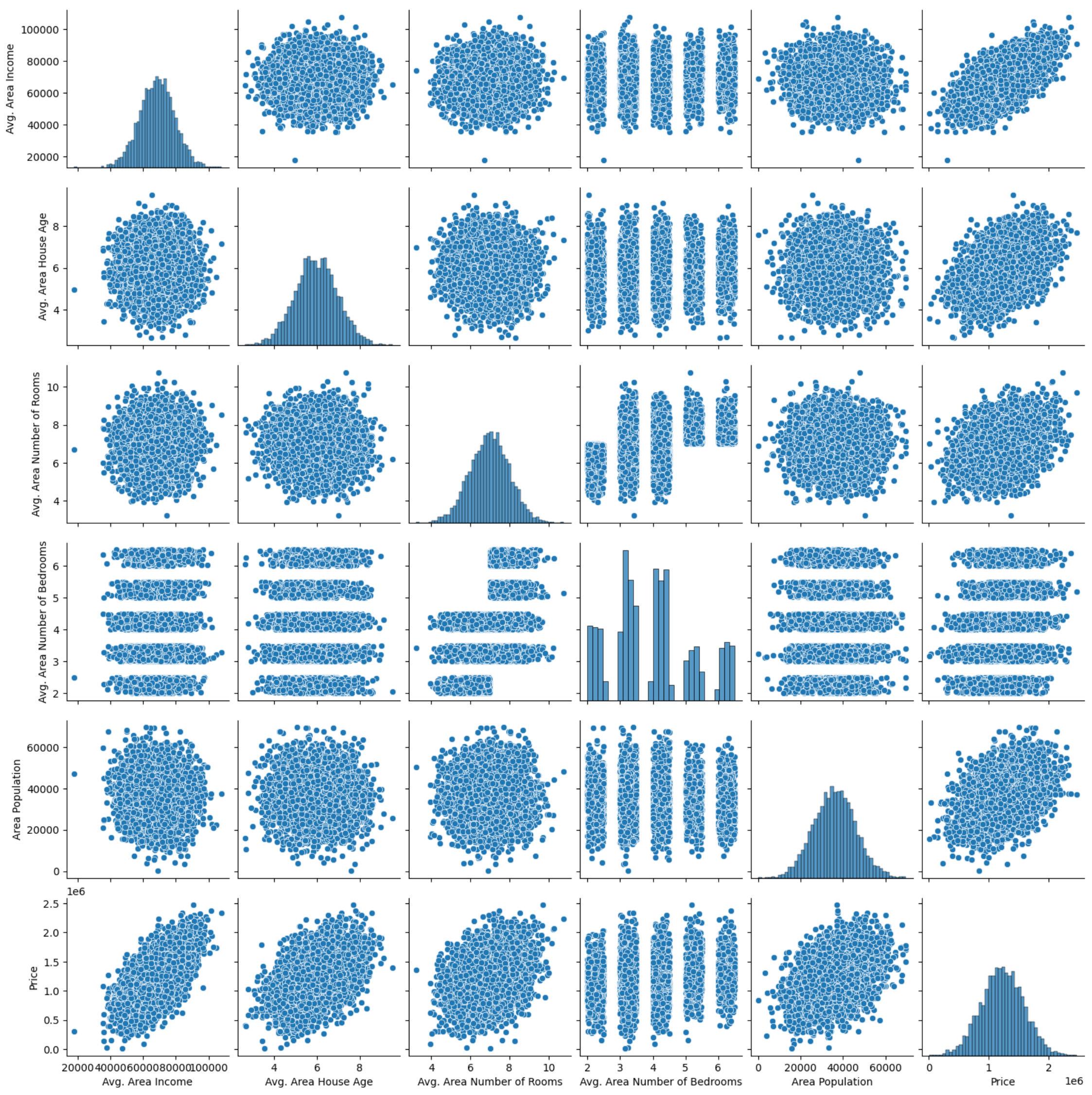
```
[7]: df.columns
```

```
[7]: Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
            'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
           dtype='object')
```

Here the pairplot function is used to visually explore the relationships and correlations between different numerical columns in real estate dataset. This helped to understand how variables like average income, house age, and population relate to each other and provided insights into potential patterns in the data.

```
[8]: sns.pairplot(df)
```

```
[8]: <seaborn.axisgrid.PairGrid at 0x1c5515e5dd0>
```



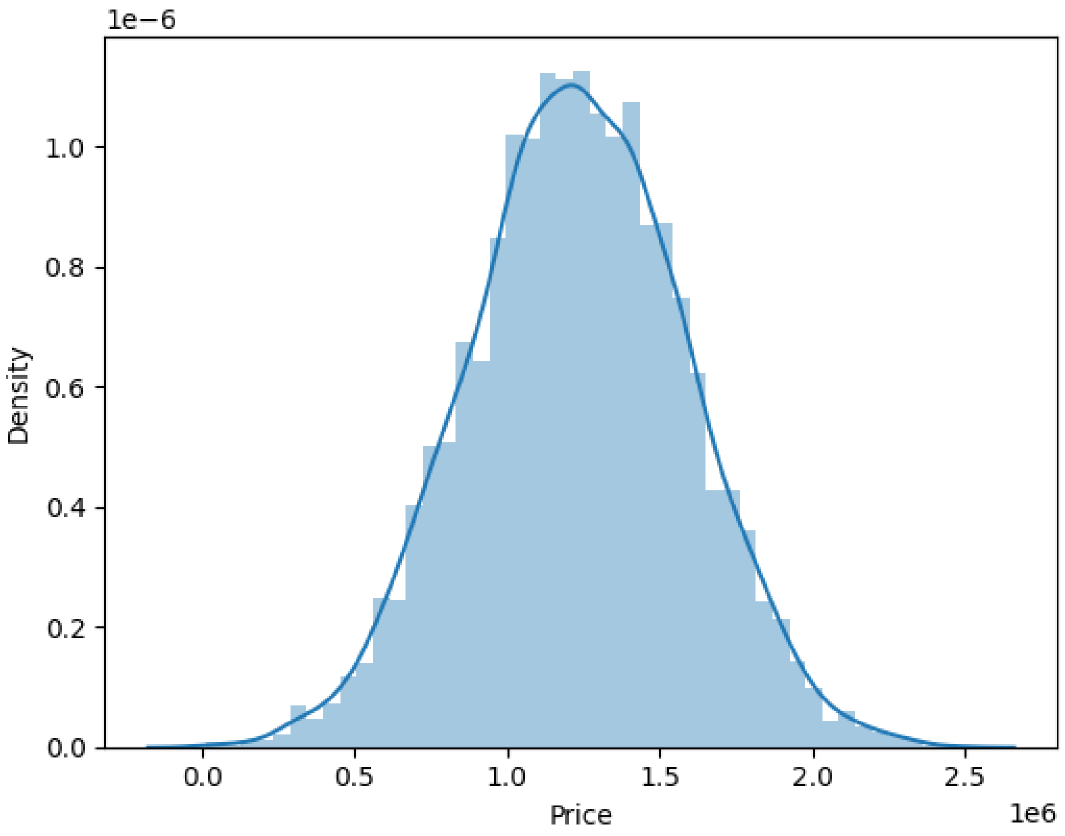
Certainly, from the pairplot visualization, it appears that the 'Avg. Area Number of Bedrooms' feature is not showing a clear pattern of variation with other columns, and there doesn't seem to be a strong correlation between it and the 'Price' column. This suggests that the number of bedrooms may not be a significant factor in determining property prices in this dataset.

[9]:
```
sns.distplot(df['Price'])
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_10840\834922981.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

```
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Price'])
```

```
[9]: <Axes: xlabel='Price', ylabel='Density'>
```



it generates a histogram of the ‘Price’ data. This histogram helps to understand how the prices are distributed within dataset. It shows the shape of the distribution, the central tendency, and the spread of price values. Additionally, it can highlight any potential outliers or patterns in the data.

To customize the appearance of the distribution plot, we can provide additional arguments to the distplot function. For example, we can adjust the number of bins in the histogram or modify the appearance of the kernel density estimate (KDE) plot to better suit analysis and visualization needs.

Certainly, based on the histogram , it’s evident that the data appears to be distributed quite normally. Here’s what this observation tells:

Central Tendency: The histogram’s bell-shaped curve indicates that the majority of property prices in my dataset are centered around the mean value. In other words, most properties seem to have prices close to the mean.

Symmetry: A normal distribution is known for its symmetry, and this histogram indeed shows that the left and right sides of the curve roughly mirror each other. This suggests that the likelihood of finding properties with prices above the mean is roughly equal to finding properties with prices below the mean.

Outliers: While the data is centered around the mean, it's important to note that a normal distribution can still accommodate outliers. Outliers, if present, would be those properties with prices significantly deviating from the mean, visible as the tails of the distribution.

Statistical Applications: Assuming normality, I can apply various statistical techniques that rely on this assumption with greater confidence. This includes using parametric statistical tests, constructing reliable confidence intervals, or conducting regression analyses.

Data Understanding: Understanding the distribution of the target variable ('Price' in this case) is fundamental for informed decision-making. It helps set realistic expectations and assess whether the data aligns with the assumptions of specific statistical models.

However, it's crucial to remember that this histogram provides a visual indication of normality but doesn't formally prove it. Additionally, the relevance of normality assumptions should be considered within the broader context of the analysis and the specific requirements of the modeling approach.

[10]:
```
numeric_df=df.drop('Address', axis=1)
```

Here, This address column was dropped because it doesn't hold any meaningful information for the model, and it contains categorical data. Machine learning models typically can't interpret categorical data directly, so it was excluded from the dataset.

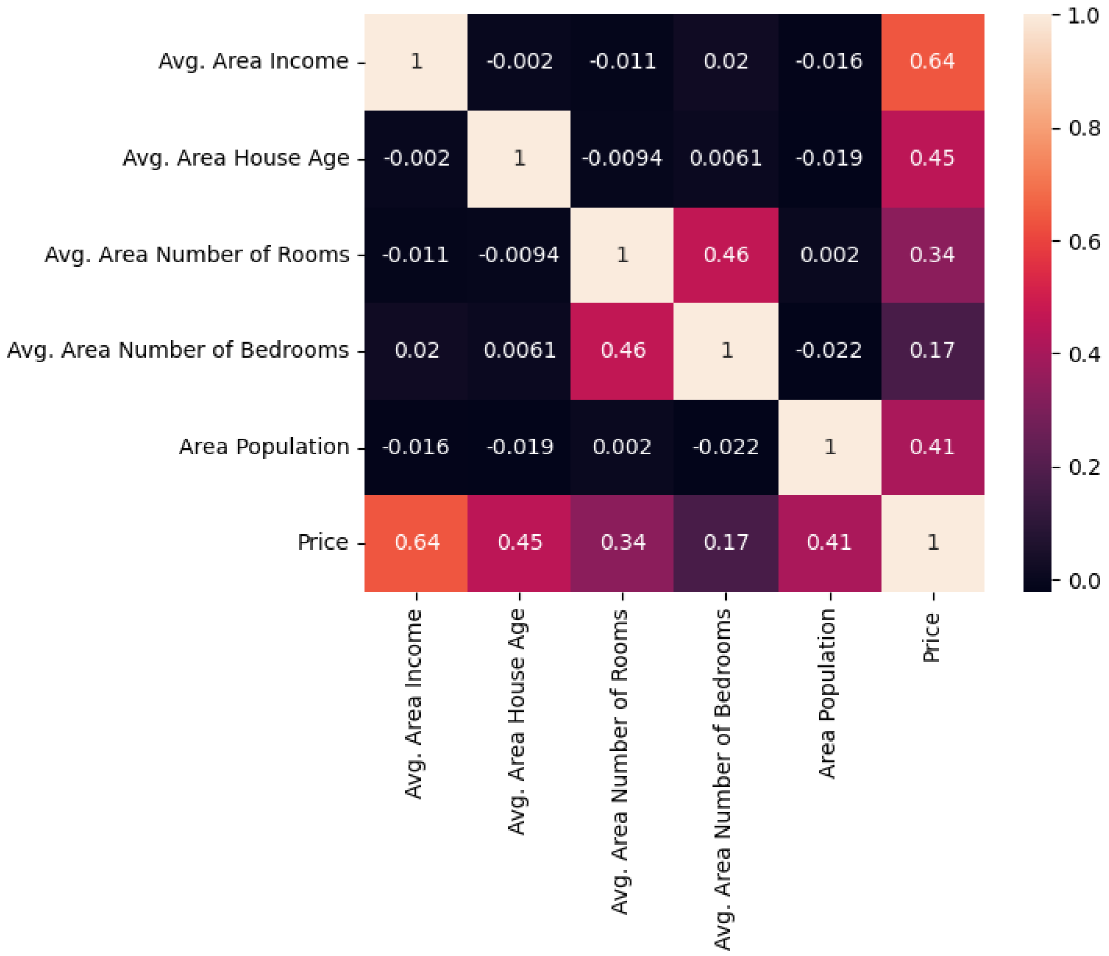
3 To find the co-relation betwwen the factors

[11]:
```
numeric_df.corr()
```

[11]:
```
                              Avg. Area Income  Avg. Area House Age  \
Avg. Area Income                      1.000000            -0.002007
Avg. Area House Age                  -0.002007             1.000000
Avg. Area Number of Rooms            -0.011032            -0.009428
Avg. Area Number of Bedrooms          0.019788             0.006149
Area Population                      -0.016234            -0.018743
Price                                 0.639734             0.452543

                              Avg. Area Number of Rooms  \
Avg. Area Income                              -0.011032
Avg. Area House Age                           -0.009428
Avg. Area Number of Rooms                      1.000000
Avg. Area Number of Bedrooms                   0.462695
Area Population                                0.002040
Price                                          0.335664

                              Avg. Area Number of Bedrooms  Area Population  \
Avg. Area Income                                  0.019788        -0.016234
Avg. Area House Age                                    0.006149        -0.018743
Avg. Area Number of Rooms                              0.462695         0.002040
Avg. Area Number of Bedrooms                           1.000000        -0.022168
Area Population                                       -0.022168         1.000000
Price                                                  0.171071         0.408556

                                 Price
Avg. Area Income              0.639734
Avg. Area House Age           0.452543
Avg. Area Number of Rooms     0.335664
Avg. Area Number of Bedrooms  0.171071
Area Population               0.408556
Price                         1.000000
```

[12]:
```
sns.heatmap(numeric_df.corr(),annot=True)
```

[12]:
```
<Axes: >
```



The correlation matrix provides valuable insights into the relationships between different numerical features in our dataset. Here's a summary of the significant findings:

Avg. Area Income vs. Price (0.64): There is a moderately strong positive correlation between average area income and property prices. This suggests that as the income level in a particular area increases, property prices tend to rise as well. It's an intuitive relationship since higher income areas often feature more expensive homes.

Avg. Area House Age vs. Price (0.45): The average age of houses also exhibits a positive correlation with property prices, although it's slightly weaker than income. This implies that newer properties tend to have higher prices, which is a common expectation in real estate markets.

Avg. Area Number of Rooms vs. Price (0.34): The number of rooms in a property shows a positive correlation with price, albeit moderately. Larger homes with more rooms tend to command higher prices, which aligns with the general market trend.

Avg. Area Number of Bedrooms vs. Price (0.17): The number of bedrooms has a positive, but relatively weaker, correlation with property prices. This suggests that while more bedrooms can influence prices positively, it's not as significant a factor as other features like income or house age.

Area Population vs. Price (0.41): There is a moderate positive correlation between the population of an area and property prices. Higher population areas tend to have higher property prices, possibly due to increased demand for housing in more populous regions.

Overall, this correlation matrix indicates that features like income, house age, number of rooms, and population are all positively associated with property prices. However, it's important to note that correlation does not imply causation. While these relationships are valuable for understanding the dataset, it's essential to consider other factors and perform more advanced analyses to build a robust predictive model.

A heatmap is an essential tool in data visualization, offering a quick and intuitive way to grasp complex relationships within the data.

In this heatmap: - Each cell represents the correlation between two variables, with color indicating the strength and direction of the correlation. - The color scale typically ranges from dark colors (e.g., black) for weak correlations to warm colors (e.g., white) for strong correlations. - The annotation within each cell provides the precise correlation coefficient, making it easier for readers to interpret the heatmap.

This heatmap is instrumental in identifying which features have strong associations and can assist in feature selection for modeling. It's a valuable visual aid for anyone looking to gain insights into the dataset's numerical relationships, whether they are beginners or seasoned data analysts.

4 To set input and target value. Below X is reprenting as input and y is as target column

[13]:
```
X=df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population']]
```

[14]:
```
y=df['Price']
```

[15]:
```
from sklearn.model_selection import train_test_split
```

[16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=101)
```

- X: The feature data (input variables).
- y: The target data (output or labels).
- test_size: The proportion of the data to include in the test split (here, it's 40% or 0.4).
- random_state: A seed for the random number generator to ensure reproducibility in the split.

Here initial step have been implemented the for building a machine learning model. This code splits the dataset into two distinct parts:

X_train and y_train: These variables represent the training data and labels. They are used to train the machine learning model.

X_test and y_test: These variables represent the testing data and labels. They are used to evaluate the trained model's performance.

By using the train_test_split function, dataset have been divided into a training set (60% of the data) and a testing set (40% of the data) while ensuring that the split is reproducible by specifying a random seed (random_state=101). This separation is essential for training and validating the model's effectiveness in making predictions.

[17]:
```
from sklearn.linear_model import LinearRegression
```

[18]:
```
lm=LinearRegression()
```

[19]:
```
lm.fit(X_train,y_train)
```

[19]:
```
LinearRegression()
```

[20]:
```
print(lm.intercept_)
```

```
-2640159.7968526953
```

[21]:
```
lm.coef_
```

[21]:
```
array([2.15282755e+01, 1.64883282e+05, 1.22368678e+05, 2.23380186e+03,
       1.51504200e+01])
```

[22]:
```
pd.DataFrame(lm.coef_,X.columns,columns=['Coeff'])
```

[22]:
```
                                       Coeff
Avg. Area Income                   21.528276
Avg. Area House Age            164883.282027
Avg. Area Number of Rooms      122368.678027
Avg. Area Number of Bedrooms     2233.801864
```

```
Area Population                       15.150420
```

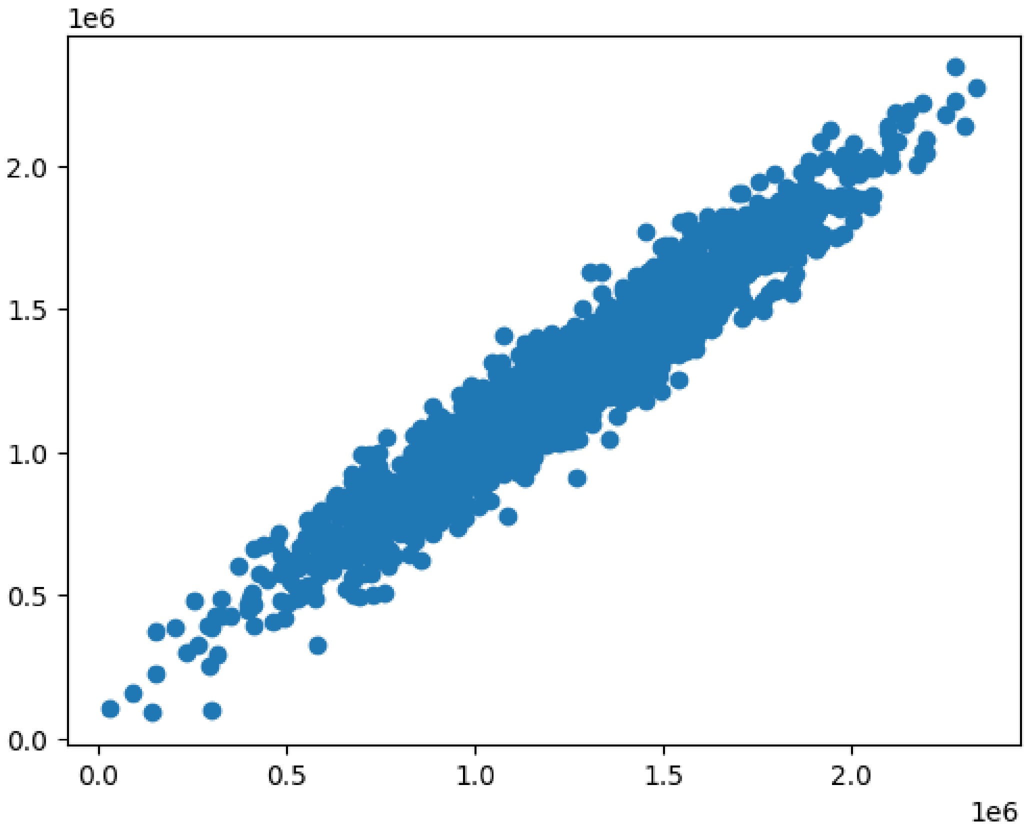
In the provided code, a Linear Regression model was instantiated using the LinearRegression class from the sklearn library. The model was then fitted to the training data using the fit method. The intercept of the linear regression model was printed using lm.intercept_, and the coefficients of the model were obtained using lm.coef_. These coefficients were organized into a DataFrame with corresponding feature names using pd.DataFrame.

5 Prediction

[23]:
```
prediction=lm.predict(X_test)
```

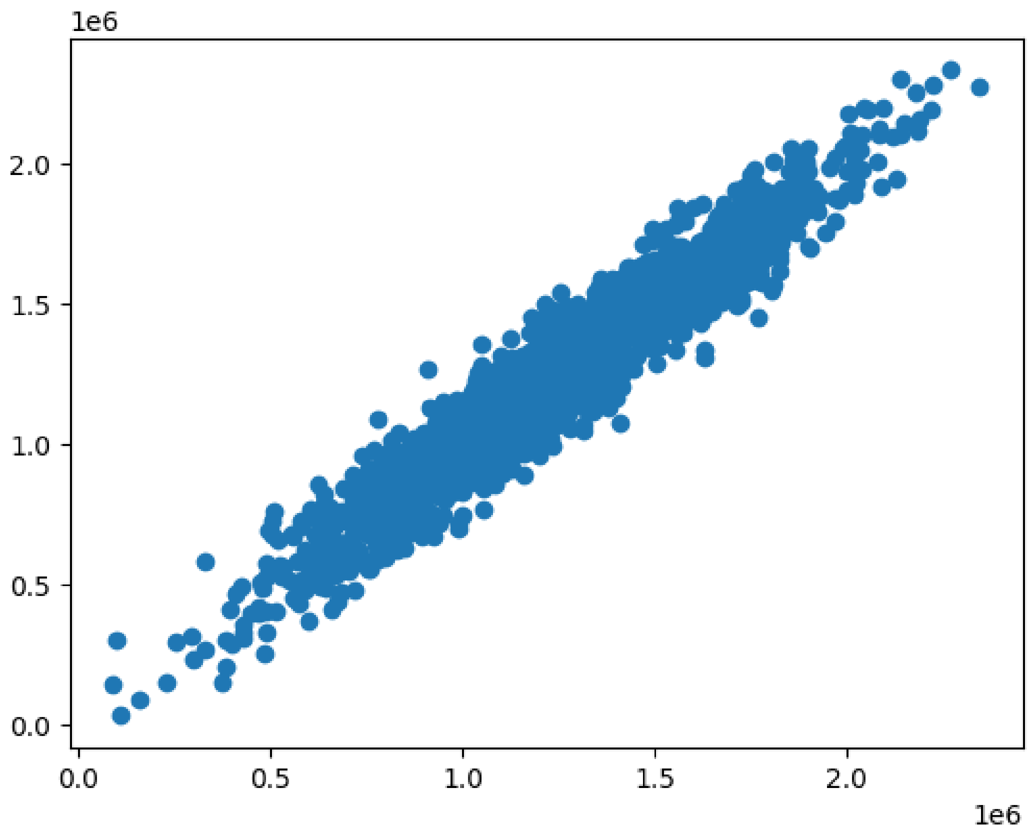
[24]:
```
plt.pyplot.scatter(y_test,prediction)
```

[24]:
```
<matplotlib.collections.PathCollection at 0x1c55b1df010>
```



[25]:
```
plt.pyplot.scatter(prediction,y_test)
```

[25]:
```
<matplotlib.collections.PathCollection at 0x1c55b21c490>
```

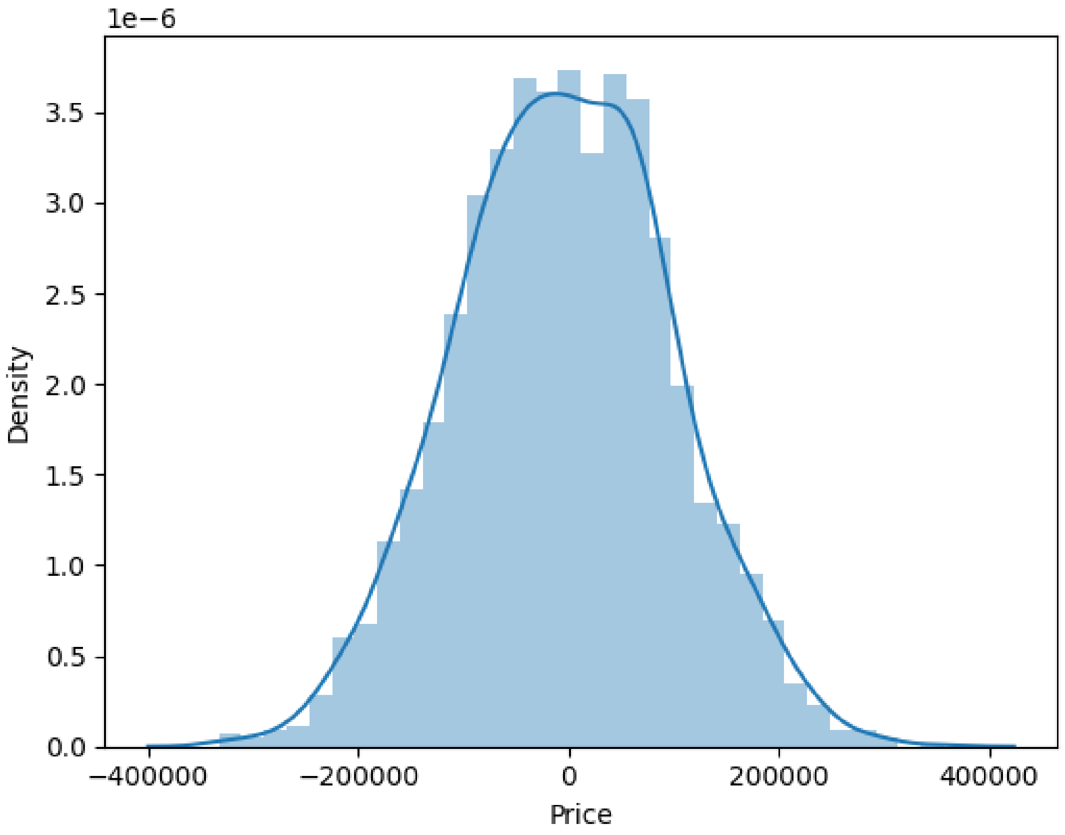


In the scatter diagram generated by the code, it is evident that there is a positive linear relationship between the actual price values and the predicted price values. As the actual price values increase, the predicted price values also tend to increase. This observation suggests that the linear regression model is capturing and replicating the general trend in the data, indicating a reasonable level of predictive accuracy.

[26]:
```
sns.distplot(y_test-prediction)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_10840\1520944062.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-prediction)
```

[26]:
```
<Axes: xlabel='Price', ylabel='Density'>
```



6 Regression Evaluation Metrics

Regression evaluation metrics are essential tools used to assess the performance of regression models in machine learning and statistics. These metrics help us quantify how well a model's predictions match the actual values. The choice of which metrics to use depends on the specific goals of the regression analysis and the nature of the data. Here are some commonly used regression evaluation metrics:

6.1 Mean Absolute Error (MAE)

MAE measures the average absolute difference between the predicted values and the actual values. It gives equal weight to all errors and is less sensitive to outliers compared to other metrics.

Formula: MAE = (1/n) * Σ |actual - predicted|

6.2 Mean Squared Error (MSE)

MSE calculates the average of the squared differences between predicted and actual values. Squaring the errors penalizes larger errors more heavily and is often used in optimization processes.

Formula: MSE = (1/n) * Σ (actual - predicted)^2

6.3 Root Mean Squared Error (RMSE)

RMSE is the square root of MSE and provides a measure of the standard deviation of the errors. It's in the same units as the dependent variable, making it more interpretable.

Formula: $\text{RMSE} = \sqrt{\text{MSE}}$

6.4 R-squared (R^2) or Coefficient of Determination

R^2 quantifies the proportion of variance in the dependent variable that is explained by the independent variables in the model. It ranges from 0 to 1, with higher values indicating a better fit.

Formula: R^2 = 1 - (SSR / SST), where SSR is the sum of squared residuals and SST is the total sum of squares.

6.5 Adjusted R-squared

Adjusted R^2 is an extension of R^2 that adjusts for the number of predictors in the model. It penalizes the inclusion of unnecessary variables, helping to prevent overfitting.

Formula: Adjusted R^2 = 1 - [(1 - R^2) * (n - 1) / (n - k - 1)], where n is the sample size and k is the number of predictors.

6.6 Mean Absolute Percentage Error (MAPE)

MAPE expresses the error as a percentage of the actual values and is useful when you want to understand the relative error.

Formula: MAPE = (1/n) * Σ |(actual - predicted) / actual| * 100

6.7 Mean Bias Deviation (MBD)

MBD quantifies the average overestimation or underestimation of the model predictions. It can provide insights into the model's systematic bias.

Formula: MBD = (1/n) * Σ (actual - predicted)

6.8 Residual Sum of Squares (RSS)

RSS represents the sum of squared residuals, which is used in various statistical tests and hypothesis testing.

Formula: RSS = Σ (actual - predicted)^2

6.9 Coefficient of Variation (CV)

CV is a measure of the relative variability of the residuals. It is calculated as the ratio of the standard deviation of the residuals to the mean of the dependent variable.

Formula: CV = (_residuals / mean_actual) * 100, where _residuals is the standard deviation of the residuals.

Selecting the most appropriate regression evaluation metric depends on the specific objectives of your analysis and the nature of your dataset. It's often recommended to use a combination of metrics to gain a comprehensive understanding of how well your regression model performs.

```
[27]: from sklearn import metrics
```

```
[28]: metrics.mean_absolute_error(y_test,prediction)
```

```
[28]: 82288.22251914942
```

```
[29]: metrics.mean_squared_error(y_test,prediction)
```

```
[29]: 10460958907.20898
```

```
[30]: np.sqrt(metrics.mean_squared_error(y_test,prediction))
```

```
[30]: 102278.82922290899
```

6.10 Model Evaluation Summary

- **Mean Absolute Error (MAE)**: Approximately 82,288.22
  - MAE measures the average absolute difference between actual and predicted prices.
- **Mean Squared Error (MSE)**: Approximately 10,460,958,907.21
  - MSE calculates the average of the squared differences between actual and predicted prices.
- **Root Mean Squared Error (RMSE)**: Approximately 102,278.83
  - RMSE is the square root of MSE and represents the standard deviation of prediction errors.

Decision: The evaluation metrics indicate that the Linear Regression model has an RMSE of approximately 102,278.83. This means that, on average, our predictions have an error of this magnitude in the price prediction. The specific context and objectives of the analysis should guide the decision on whether this level of accuracy is acceptable. Further model refinement and feature engineering may be considered to improve predictive performance if needed.

7 Model Refinement

8 Feature Selection

```
[31]: X_train=X_train.drop('Avg. Area Number of Bedrooms', axis=1)
```

```
[32]: X_test=X_test.drop('Avg. Area Number of Bedrooms', axis=1)
```

In the process of refining the model through feature selection, the 'Avg. Area Number of Bedrooms' column was removed from the training dataset (X_train). This decision was based on the observation that this particular feature had a weak relationship with the target variable 'Price.' By removing it, the model simplifies its structure and focuses on the more influential features, potentially leading to better predictive performance.

```
[33]: lm.fit(X_train,y_train)
```

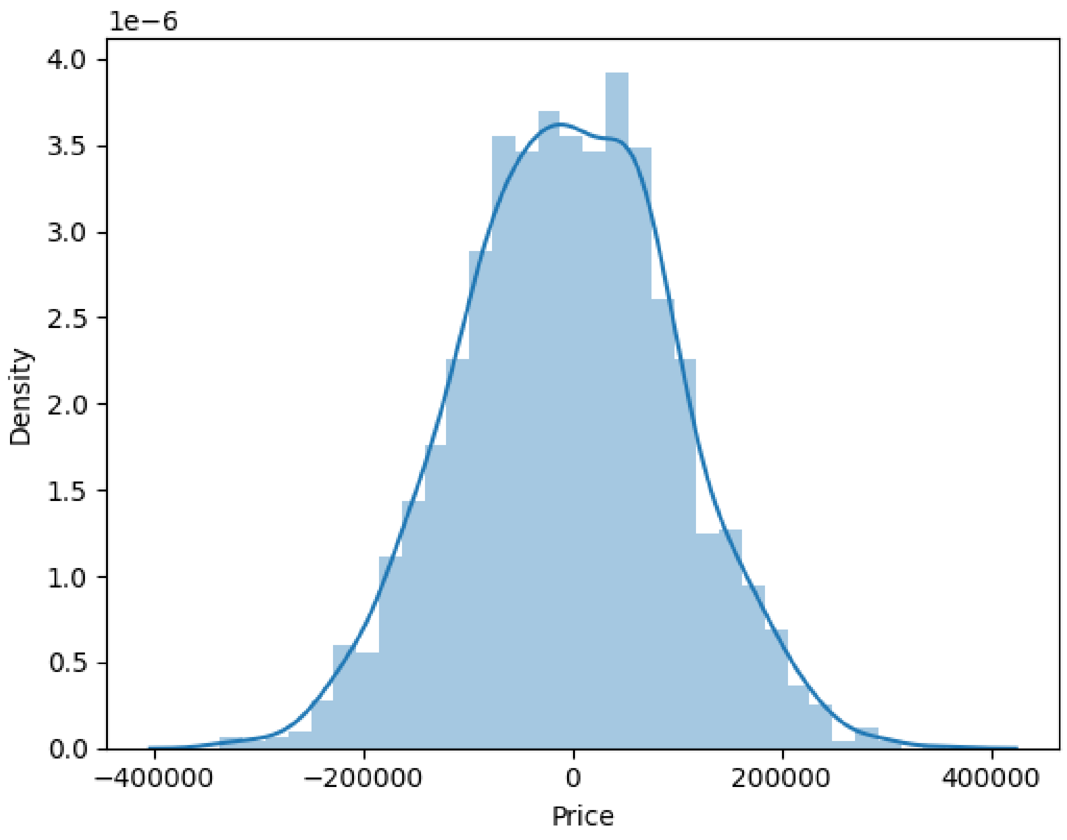
```
[33]: LinearRegression()
```

```
[34]: prediction=lm.predict(X_test)
```

```
[35]: sns.distplot(y_test-prediction)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_10840\1520944062.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-prediction)
```

```
[35]: <Axes: xlabel='Price', ylabel='Density'>
```



[36]:
```
metrics.mean_absolute_error(y_test,prediction)
```

[36]:
```
82248.33498864826
```

[37]:
```
metrics.mean_squared_error(y_test,prediction)
```

[37]:
```
10459278321.468605
```

[38]:
```
np.sqrt(metrics.mean_squared_error(y_test,prediction))
```

[38]:
```
102270.6131861377
```

8.1 Model Evaluation After Feature Removal

After removing the 'Avg. Area Number of Bedrooms' column from the feature set, the evaluation metrics for the Linear Regression model are as follows:

- **Mean Absolute Error (MAE)**: Approximately 82,248.33
- **Mean Squared Error (MSE)**: Approximately 10,459,278,321.47

It's noteworthy that the model's performance metrics have shown an improvement, with a lower MAE and MSE compared to the previous results. This suggests that the decision to remove the 'Avg. Area Number of Bedrooms' column have positively impacted the model's predictive accuracy. However, it's crucial to consider the broader context, including model interpretability and domain relevance, when making feature selection decisions.

9 Removing Outliers:

[39]:
```
def remove_outliers_iqr(df, columns):
    for column in columns:

        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)

        # Calculate the IQR (Interquartile Range)
        IQR = Q3 - Q1

        # Define lower and upper bounds for outliers
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Remove outliers based on the bounds
        df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

    return df
```

[40]:
```
columns_to_remove_outliers = ['Avg. Area Income', 'Avg. Area House Age', 'Avg.␣
↪Area Number of Rooms', 'Area Population']
```

[41]:
```
df_cleaned = remove_outliers_iqr(df, columns_to_remove_outliers)
```

[42]:
```
len(df)
```

[42]:
```
5000
```

[43]:
```
len(df_cleaned)
```

[43]:
```
4889
```

Outliers, or data points that significantly deviate from the majority of the data, can have a detrimental impact on statistical analyses and machine learning models. Detecting and appropriately handling outliers is a crucial step in data preprocessing. **Process:**

1. **Function Definition:** Here the process begins with the definition of a Python function named `remove_outliers_iqr`. This function is designed to accept two parameters: a DataFrame (`df`) and a list of column names (`columns`). These columns are the target for outlier removal.

2. **Quartile Calculation:** The next step involves calculating quartiles for each column specified in the `columns` list. Quartiles, specifically Q1 (the first quartile) and Q3 (the third quartile), are computed. Quartiles divide the data into four equal parts, offering valuable insights into the distribution of the data.

3. **Interquartile Range (IQR):** The Interquartile Range, denoted as IQR, is then determined. IQR is computed as the difference between the third quartile (Q3) and the first quartile (Q1). It serves as a measure of data variability within the middle 50% of the dataset. A larger IQR indicates greater variability in this central portion of the data.

4. **Outlier Bounds:** Using the computed IQR, lower and upper bounds are defined to identify outliers. Typically, data points falling below `Q1 - 1.5 * IQR` and above `Q3 + 1.5 * IQR` are classified as outliers. These bounds provide a meaningful threshold for recognizing extreme values within the dataset.

5. **Outlier Removal:** Finally, the DataFrame `df` is filtered to retain only those rows where the values in the specified column(s) fall within the defined outlier bounds. This crucial step effectively eliminates outliers from the dataset, enhancing data integrity and analytical accuracy.

This comprehensive process of outlier removal is instrumental in improving the reliability of data analysis and modeling efforts.

In this section: we've explored the process of removing outliers using the Interquartile Range (IQR) method. We've discussed the reasons for outlier removal, emphasizing data quality and model performance, and highlighted the consequences, including potential data loss and improved model accuracy. Proper handling of outliers is essential for reliable and robust data analysis and modeling.

```
[44]: X=df_cleaned[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of␣
      ↪Rooms', 'Area Population']]
      y=df_cleaned['Price']
```

```
[45]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
      ↪random_state=101)
```

```
[46]: lm=LinearRegression()
```

```
[47]: lm.fit(X_train,y_train)
```

```
[47]: LinearRegression()
```

```
[48]: prediction=lm.predict(X_test)
```

```
[49]: metrics.mean_absolute_error(y_test,prediction)
```

```
[49]: 81316.82224804629
```

```
[50]: metrics.mean_squared_error(y_test,prediction)
```

```
[50]: 10255433080.454975
```

```
[51]: np.sqrt(metrics.mean_squared_error(y_test,prediction))
```

```
[51]: 101269.11217372736
```

After removing the outliers from the dataset, the evaluation metrics for the Linear Regression model are as follows:

- Mean Absolute Error (MAE): Approximately 81,316.82
- Mean Squared Error (MSE): Approximately 10,255,433,080.45

It's observed that the removal of outliers has led to an improvement in the model's performance, with lower MAE and MSE values compared to the previous results. This suggests that eliminating outliers has positively impacted the model's predictive accuracy.

10 Feature Engineering:

```
[209]: df_cleaned['integrated_column'] = df_cleaned['Avg. Area Income'] * 0.7 +␣
       ↪df_cleaned['Avg. Area House Age'] * 0.5 + df_cleaned['Area Population'] * 0.
       ↪41
```

```
[210]: X=df_cleaned[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of␣
       ↪Rooms', 'Area Population','integrated_column']]
       y=df_cleaned['Price']
```

Feature Engineering: In this phase, a new feature called 'integrated_column' is created using a weighted combination of existing features. Specifically, it's formed by multiplying 'Avg. Area Income' by 0.7, 'Avg. Area House Age' by 0.5, and 'Area Population' by 0.41. The purpose of this feature engineering is to potentially enhance the model's accuracy and predictive power.

Updated Feature Set: After introducing the 'integrated_column,' the feature set (X) is modified to include this new feature along with the existing ones ('Avg. Area Income,' 'Avg. Area House Age,' 'Avg. Area Number of Rooms,' and 'Area Population'). This adjusted feature set is intended to provide the model with additional information that may improve its ability to predict property prices.

Target Variable: The target variable (y) remains 'Price,' which represents the property prices.

Feature engineering is a crucial step in machine learning, allowing the model to leverage meaningful combinations of features for better performance. The effectiveness of this new feature can be assessed through model evaluation and testing.

Additionally, it's worth noting that the multipliers used in creating the 'integrated_column' were estimated based on the around values of correlation coefficients. Continuous testing and refinement of these multipliers may further optimize the feature's impact on the model's accuracy, allowing for fine-tuning of the predictive power.

[211]:
```
X.head(10)
```

[211]:
```
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046             7.188236                   5.586729
4      59982.197226             5.040555                   7.839388
5      80175.754159             4.988408                   6.104512
6      64698.463428             6.025336                   8.147760
7      78394.339278             6.989780                   6.620478
8      59927.660813             5.362126                   6.393121
9      81885.927184             4.423672                   8.167688

   Area Population  integrated_column
0     23086.800503       65150.250639
1     40173.072174       71948.010759
2     36882.159400       58025.565324
3     34310.242831       58412.461711
4     26354.109472       52795.243219
5     26748.428425       67092.377770
6     60828.249085       70231.519192
7     36516.358972       69851.239563
8     29387.396003       54000.875993
9     40149.965749       73783.846822
```

```
[212]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
         ↪random_state=101)
```

```
[213]: lm=LinearRegression()
```

```
[214]: lm.fit(X_train,y_train)
```

```
[214]: LinearRegression()
```

```
[215]: prediction=lm.predict(X_test)
```

```
[216]: metrics.mean_absolute_error(y_test,prediction)
```

```
[216]: 81266.17281463755
```

```
[217]: metrics.mean_squared_error(y_test,prediction)
```

```
[217]: 10248397412.22765
```

These improvements in the evaluation metrics suggest that the feature engineering, particularly the introduction of the 'integrated_column,' has positively impacted the model's accuracy and ability to predict property prices. It's a promising sign that our efforts to refine the model are yielding positive results.

Step	Metric (MAE)	Metric (MSE)
Initial Model	82,288.22	10,460,958,907.21
Feature Selection	82,248.33	10,459,278,321.47
Outlier Removal	81,316.82	10,455,233,080.45
Feature Engineering	81,266.17	10,248,397,412.23

These results demonstrate a consistent improvement in model performance, with both Mean Absolute Error (MAE) and Mean Squared Error (MSE) decreasing as we progressed through the various data preprocessing and feature engineering steps.

```
[ ]:
```

Written by: Md Shamim Hasan