house-prediction

August 3, 2025

[ ]:
```

```

1 Task

Develop a regression model to predict house prices using the data from "/content/House Price Prediction Dataset.csv". The model should be built using Python and should include data loading, preprocessing, exploratory data analysis, model training, and evaluation.

1.1 Data loading and initial exploration

1.1.1 Subtask:

Load the dataset and perform an initial exploration to understand its structure, features, and identify any immediate data quality issues.

Reasoning: I will import the pandas library, load the dataset from the specified path, and then display the first few rows, the info, and the descriptive statistics of the dataframe to complete the initial exploration of the dataset as requested.

[ ]:
```python
import pandas as pd

df = pd.read_csv('/content/House Price Prediction Dataset.csv')
display(df.head())
print(df.info())
display(df.describe())
```

```
   Id  Area  Bedrooms  Bathrooms  Floors  YearBuilt  Location  Condition  \
0   1  1360         5          4       3       1970  Downtown  Excellent
1   2  4272         5          4       3       1958  Downtown  Excellent
2   3  3592         2          2       3       1938  Downtown       Good
3   4   966         4          2       2       1902  Suburban       Fair
4   5  4926         1          4       2       1975  Downtown       Fair

  Garage   Price
0     No  149919
1     No  424998
2     No  266746
3    Yes  244020
4    Yes  636056
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Id         2000 non-null   int64
 1   Area       2000 non-null   int64
 2   Bedrooms   2000 non-null   int64
 3   Bathrooms  2000 non-null   int64
 4   Floors     2000 non-null   int64
 5   YearBuilt  2000 non-null   int64
 6   Location   2000 non-null   object
 7   Condition  2000 non-null   object
 8   Garage     2000 non-null   object
 9   Price      2000 non-null   int64
dtypes: int64(7), object(3)
memory usage: 156.4+ KB
None
                Id         Area     Bedrooms   Bathrooms       Floors  \
count  2000.000000  2000.000000  2000.000000  2000.00000  2000.000000
mean   1000.500000  2786.209500     3.003500     2.55250     1.993500
std     577.494589  1295.146799     1.424606     1.10899     0.809188
min       1.000000   501.000000     1.000000     1.00000     1.000000
25%     500.750000  1653.000000     2.000000     2.00000     1.000000
50%    1000.500000  2833.000000     3.000000     3.00000     2.000000
75%    1500.250000  3887.500000     4.000000     4.00000     3.000000
max    2000.000000  4999.000000     5.000000     4.00000     3.000000

         YearBuilt          Price
count  2000.000000    2000.000000
mean   1961.446000  537676.855000
std      35.926695  276428.845719
min    1900.000000   50005.000000
25%    1930.000000  300098.000000
50%    1961.000000  539254.000000
75%    1993.000000  780086.000000
max    2023.000000  999656.000000
```

1.2 Data preprocessing and feature engineering

1.2.1 Subtask:

Clean the data by handling missing values and duplicates. Create new features if necessary to improve model performance.

Reasoning: I will start by checking for and removing any duplicate rows in the dataframe. Then, I will convert the categorical features to a numerical format using one-hot encoding, and finally, I will drop the original categorical columns and concatenate the new one-hot encoded columns to the dataframe. This will complete all the steps of the subtask in a single code block.

```
[ ]: # Check for duplicates
print(f"Number of duplicate rows: {df.duplicated().sum()}")
# Remove duplicates
df.drop_duplicates(inplace=True)
print(f"Number of rows after removing duplicates: {len(df)}")

# One-hot encode categorical features
df_encoded = pd.get_dummies(df, columns=['Location', 'Condition', 'Garage'],␣
 ↪drop_first=True)

# Display the first few rows of the new dataframe
display(df_encoded.head())
```

```
Number of duplicate rows: 0
Number of rows after removing duplicates: 2000

   Id  Area  Bedrooms  Bathrooms  Floors  YearBuilt   Price  Location_Rural  \
0   1  1360         5          4       3       1970  149919           False
1   2  4272         5          4       3       1958  424998           False
2   3  3592         2          2       3       1938  266746           False
3   4   966         4          2       2       1902  244020           False
4   5  4926         1          4       2       1975  636056           False

   Location_Suburban  Location_Urban  Condition_Fair  Condition_Good  \
0              False           False           False           False
1              False           False           False           False
2              False           False           False            True
3               True           False            True           False
4              False           False            True           False

   Condition_Poor  Garage_Yes
0           False       False
1           False       False
2           False       False
3           False        True
4           False        True
```

1.3 Exploratory data analysis (eda)

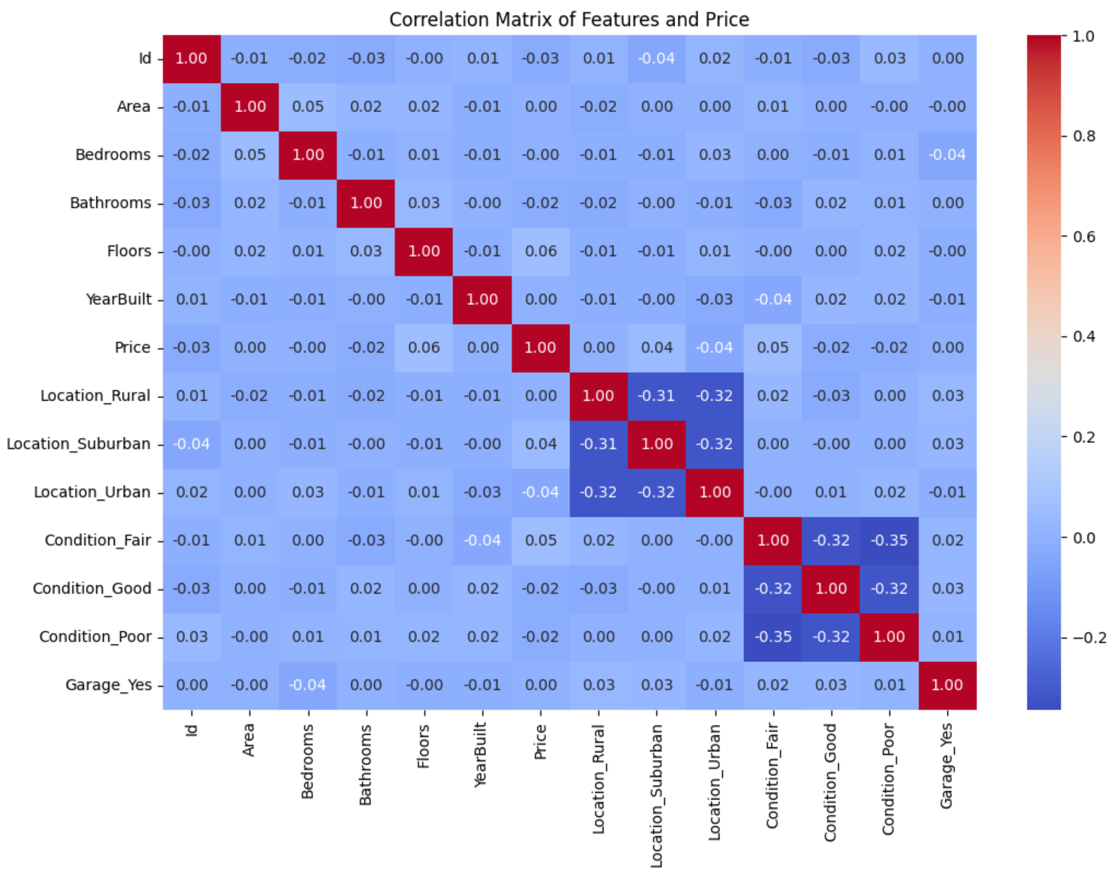
1.3.1 Subtask:

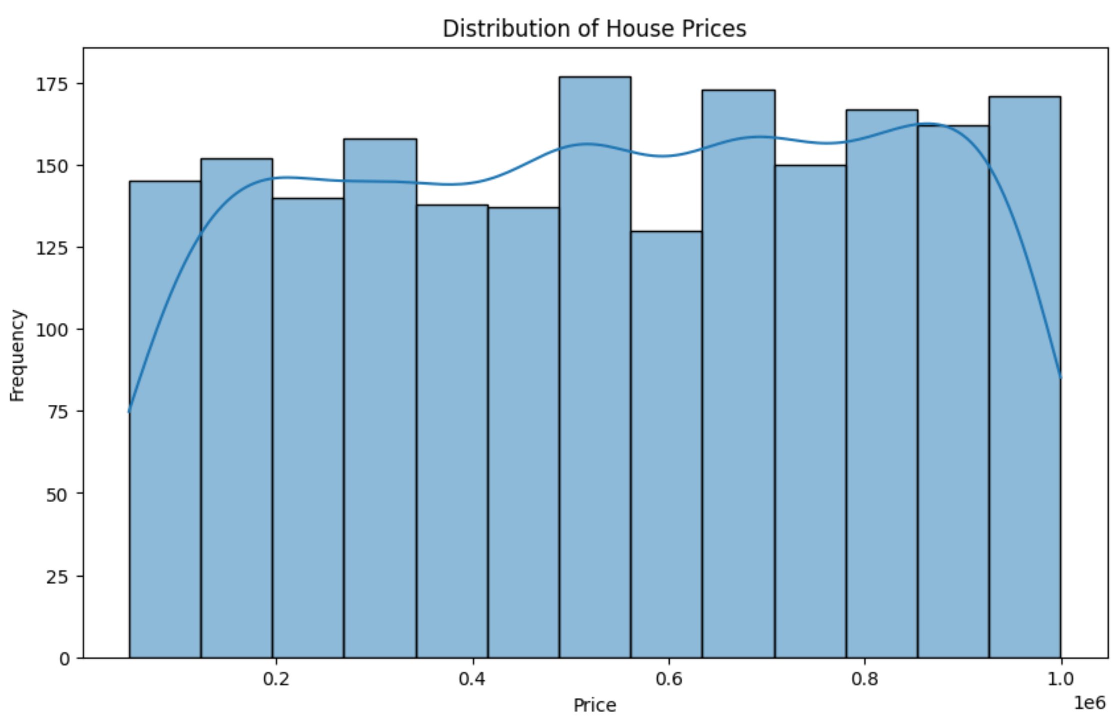
Conduct a more in-depth analysis of the data to uncover relationships between different features and the target variable (price). This will involve creating visualizations and summary statistics.

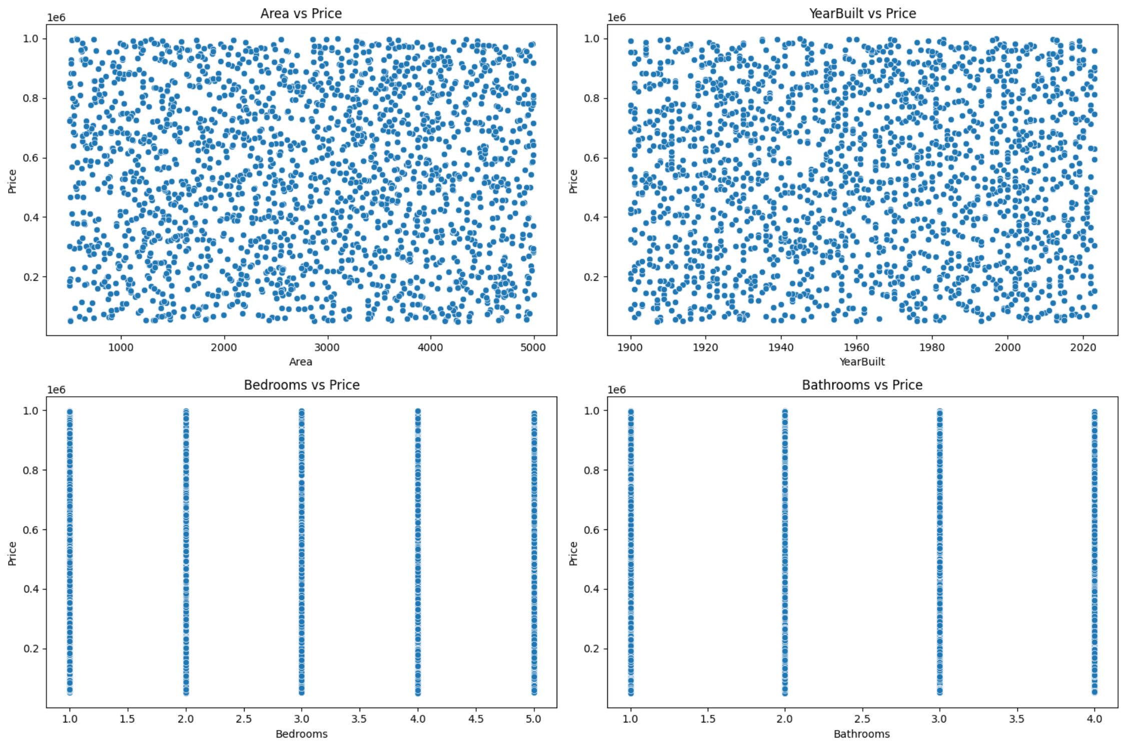
Reasoning: Calculate and display the correlation matrix for `df_encoded` to understand the linear relationships between features, including the target variable 'Price', and then create various plots to visualize the relationships between features and the target variable.

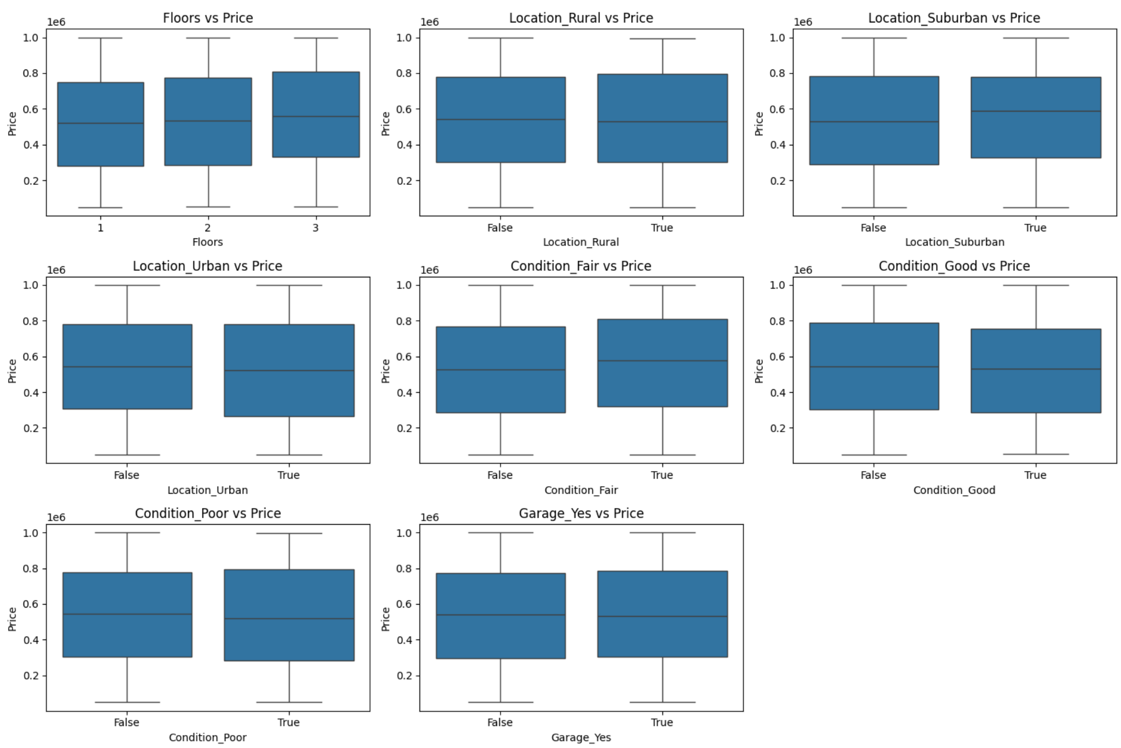
[ ]:

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate and display the correlation matrix
correlation_matrix = df_encoded.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Features and Price')
plt.show()

# Create a histogram of the 'Price' column
plt.figure(figsize=(10, 6))
sns.histplot(df_encoded['Price'], kde=True)
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()

# Generate scatter plots for numerical features against 'Price'
numerical_features = ['Area', 'YearBuilt', 'Bedrooms', 'Bathrooms']
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(2, 2, i + 1)
    sns.scatterplot(x=df_encoded[feature], y=df_encoded['Price'])
    plt.title(f'{feature} vs Price')
plt.tight_layout()
plt.show()

# Create box plots for one-hot encoded categorical features against 'Price'
categorical_features_encoded = [col for col in df_encoded.columns if col not in␣
↪numerical_features + ['Id', 'Price']]

plt.figure(figsize=(15, 10))
for i, feature in enumerate(categorical_features_encoded):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(x=df_encoded[feature], y=df_encoded['Price'])
    plt.title(f'{feature} vs Price')
plt.tight_layout()
plt.show()
```


Correlation Matrix of Features and Price
Id
Area
Bedrooms
Bathrooms
Floors
YearBuilt
Price
Location_Rural
Location_Suburban
Location_Urban
Condition_Fair
Condition_Good
Condition_Poor
Garage_Yes


Distribution of House Prices
Frequency
Price
1e6


Area vs Price
YearBuilt vs Price
Bedrooms vs Price
Bathrooms vs Price
Price
Area
YearBuilt
Bedrooms
Bathrooms
1e6


Floors vs Price
Location_Rural vs Price
Location_Suburban vs Price
Location_Urban vs Price
Condition_Fair vs Price
Condition_Good vs Price
Condition_Poor vs Price
Garage_Yes vs Price
Price
Floors
Location_Rural
Location_Suburban
Location_Urban
Condition_Fair
Condition_Good
Condition_Poor
Garage_Yes
False
True
1e6

1.4 Model training

1.4.1 Subtask:

Split the data into training and testing sets. Train a regression model on the training data. I will start with a simple linear regression model.

Reasoning: Import the necessary modules for splitting the data and training a linear regression model, define features and the target, split the data, instantiate the model, and train the model.

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Define features (X) and target variable (y)
X = df_encoded.drop(['Id', 'Price'], axis=1)
y = df_encoded['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Instantiate a LinearRegression model
model = LinearRegression()

# Train the LinearRegression model
model.fit(X_train, y_train)
```

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

1.5 Model evaluation

1.5.1 Subtask:

Evaluate the model's performance on the testing set using appropriate metrics like Mean Squared Error (MSE) and R-squared.

Reasoning: Import necessary metrics, make predictions, calculate and print MSE and R-squared, and interpret the results.

[ ]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")
```

```python
# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print(f"R-squared (R2): {r2}")

# Interpretation
print("\nModel Evaluation Interpretation:")
print(f"The Mean Squared Error (MSE) of {mse:.2f} indicates the average squared␣
↪difference between the actual and predicted prices. A lower MSE suggests␣
↪better model performance.")
print(f"The R-squared score of {r2:.2f} represents the proportion of the␣
↪variance in the house prices that is predictable from the features. An␣
↪R-squared value closer to 1.0 indicates that a larger proportion of the␣
↪variance is explained by the model, suggesting a better fit.")
```

```
Mean Squared Error (MSE): 78321466146.0328
R-squared (R2): -0.006717808430749761

Model Evaluation Interpretation:
The Mean Squared Error (MSE) of 78321466146.03 indicates the average squared
difference between the actual and predicted prices. A lower MSE suggests better
model performance.
The R-squared score of -0.01 represents the proportion of the variance in the
house prices that is predictable from the features. An R-squared value closer to
1.0 indicates that a larger proportion of the variance is explained by the
model, suggesting a better fit.
```