Utilization of Machine Learning For Housing Purchase Prediction

Sherlin Narayanan (sn3870)

sn3870@nyu.edu

Giorgi Merabishvili (gm3386)

gm3386@nyu.edu

ECE-GY 6143 ML Final Project

Dr. Pei Liu

12/23/2023

Through this report, we will cover the following:

- ☐ Quad Chart with Summary
- ☐ Introduction
- ☐ Experimental Set-up
  - ☐ Dataset Retrieval
  - ☐ Data Cleaning
  - ☐ Data Transformation
  - ☐ Data Visualization
- ☐ Experimental Results
  - ☐ Linear Regression
  - ☐ KNN
  - ☐ Random Forest
  - ☐ Linear Regression with K-Fold
- ☐ Summary
- ☐ Errors/Lessons Learned
- ☐ Future Implementation
- ☐ Contributions to Society
- ☐ References
- ☐ Experimental Code

Quad Chart:

Motivation	Machine Learning Importance
• Ease the purchasing and selling process of future homeowners and developers/real estate agents respectively. • Especially useful in high demand locations (NY) where finding affordable housing is difficult	• Machine Learning escalates the analysis of large data to help a greater group of people on a larger scale • Using Machine Learning completes the task at hand at a much faster rate and saves time which can be used to accomplish much more
Contribution	**Results/Summary**
• Used ML Algorithms of linear regression, Random Forest, and K-Nearest Neighbor • Use of multiple algorithms helps identify most efficient method to identify trends and relationship of housing attributes to price • Made comparisons between algorithms and visualized performance metrics to understand and explain said differences and similarities • Used K-Fold Cross Validation to better analyze predictive model	• Analysis presented linear regression with K-fold cross validation as the most efficient algorithm with a correlation coefficient value of 0.74 • Future implementations consist of updated data, additional algorithms, and further analysis of the relationship between housing attributes and price • Lessons learned include Data visualization, preprocessing, evaluation, and analysis in addition to insight of specific algorithms used • Potential areas of error include dataset cleansing, feature extraction, model fitting, and catering to each algorithm.

I. Introduction:

Any individual requires shelter over their head. It is essential that a person find the right type of shelter to inhabit. On one side, there are real estate agents, housing developers, and sellers in general, while on the other hand, there are those buyers who are looking to purchase housing. For those looking to reside in areas such as New York City, Boston, California, housing can be very expensive and it may be difficult to find affordable pricing that adequately meets the

needs of everyone involved. Therefore, in order to benefit all those parties, developing predictive measures will assist with finding the perfect housing that accommodates the necessities and requirements that customers are looking for. With respect to this project, we are looking into predicting the prices of houses through machine learning means. Machine Learning is a type of artificial intelligence that obtains past data and can be trained to then apply its learnings to predict and accurately analyze new data based on the training data. As data around the world expands, and businesses require means to accommodate to those ongoing changes, it becomes crucial to implement measures that can keep up with the fast-paced nature of the world in order to make efficient decisions and grow. Therefore to grow in the housing industry and perform functions adequately machine learning is important for this specific field.

II. Experimental Set-up:

For the purpose of predictive analysis of housing prices, we conducted machine learning analysis using certain algorithms and technique (Linear Regression, Random Forest, K-Nearest Neighbors and K-fold).Linear regression algorithm serves to essentially determine the linear relationship between the independent or predictor and dependent variable. Random Forest algorithm constructs decision trees to essentially create a forest for predictive analysis where each output will be combined and followed until the end of the tree when the final prediction can be made. K-Nearest Neighbors (KNN) on the other hand works to determine the class that certain points belong to after obtaining the K nearest point by comparing training and testing data and it works best with complex data. Lastly, K-fold cross validation is a prediction technique that can be applied to certain models by splitting the data and analyzing the models that are being used with k groups. In addition to applying the chosen algorithms individually, hybrid combinations of the algorithm that offer higher performance with the K-fold technique

were made and applied to observe the result in performance. These specific algorithms and technique were used to best understand predictive analysis. By taking a look at the experimental code provided in a section below, the experimental procedure can be understood.

A. Dataset Retrieval:

Initially, for an analysis to be conducted, relevant data is required. Therefore for this analysis, a dataset from Kaggle was obtained. The dataset was found to contain 13 features and 545 samples, serving as a relatively small dataset in comparison with others, but a somewhat large dataset for the sake of the current stage of this analysis. The attributes that were included in the study consist of price, area, number of bedrooms, bathrooms, and stories, presence of a basement and/or guest room, immediate access to the main road, the inclusion of a hot water heater and air conditioning, number of parking spots, location preference, and furnished status. The data was categorized either quantitatively or qualitatively based on the respective categories, and presented through a comma-separated values formatted dataset. The proper retrieval of the csv dataset and the identification of the dataset components can be visualized through lines 1 through 4 in the source code below.

In [2]:
```
# Load, read the CSV file and handle exceptions

try:
    df = pd.read_csv("/Users/giorgi/Desktop/Housing.csv")
except FileNotFoundError:
    print("The file was not found, Please check the file path.")
except Exception as e:
    print("An error occurred:", e)
```

In [3]:
```
# Calculate and print the number of rows and columns

num_records = df.shape[0]
num_columns = df.shape[1]

print(f"The dataset contains {num_records} records (rows) and {num_columns} attributes
```

```
The dataset contains 545 records (rows) and 13 attributes (columns).
```

Although this dataset does appear as a simple dataset as it does not contain an immense number of data owing to its small size, it does pose as a complex dataset offering many areas to scope for analysis as a result of its multicollinearity with multiple variables that may possibly be interconnected. After obtaining the dataset, the data was then ready to be preprocessed.

B. Data Cleaning:

The first step of data preprocessing that was taken was cleaning of the housing dataset. As datasets cannot be trusted to completely contain clean data without any null values, missing values, duplicated values, or errors of any type, the data needs to be visualized and rid of those errors. Therefore lines 8 and 11 in the source code present the action of identifying the features that contain errors to then take the necessary steps to modify those null values or other types of error. Surprisingly, this dataset did not contain any such values, which allowed for us to go to the next step.

In the cell below, we checked and addressed null values in the dataset:

```
In [8]: # find and display rows that has some but not all null values

rows_with_some_nulls = df.isnull().any(axis=1)
rows_not_completely_null = ~df.isnull().all(axis=1)
null_entries_df = df[rows_with_some_nulls & rows_not_completely_null]

display(null_entries_df)
```

price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating

In cell 11th, we converted binary columns such as “mainroad” into numerical format that is either 1 or 0.

In [11]:

```
# Convert binary columns to (1/0)
# Handle unexpected values/missing columns

binary_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning

for col in binary_cols:
    if col in df.columns and df[col].isin(['yes', 'no']).all():
        df[col] = df[col].map({'yes': 1, 'no': 0})
    else:
        print(f"Column '{col}' is missing or has unexpected values.")

display(df)
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhe
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	
...	...	...	...	...	...	...	...	...	
540	1820000	3000	2	1	1	1	0	1	
541	1767150	2400	3	1	1	0	0	0	
542	1750000	3620	2	1	1	1	0	0	
543	1750000	2910	3	1	1	0	0	0	
544	1750000	3850	3	1	2	1	0	0	

C. Data Transformation:

As the data was small to begin with and already in a preferred format, there was no need to apply any type of reduction of data to make smaller or break into smaller sections. However, as there are categorical values, binary variables, and numerical values in the 13 features, in order to maintain consistency in the analysis, the features without numerical values or with binary columns were initially identified and then assigned values of 0 or 1 based on true or false/yes or no with encoding. Categorical variables were then taken care of next and were assigned dummy variables. Lines 11 to 14 in the source code section demonstrate this part of data preprocessing.

In line 12th, Categorical variables and identified.

```
In [12]: # Print a list of all categorical columns

cat_cols = df.select_dtypes(include='object').columns
cat_data = df[cat_cols]

print("Categorical Columns:", cat_cols.tolist())

Categorical Columns: ['furnishingstatus']
```

In line 13th, dummy variables are created.

```
In [13]: # Create dummy variables for categorical columns
         # Show the first few rows

         encoded_data = pd.get_dummies(df, columns=cat_cols)

         display(encoded_data.head())
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating
0	13300000	7420	4	2	3	1	0	0	0
1	12250000	8960	4	4	4	1	0	0	0
2	12250000	9960	3	2	2	1	0	1	0
3	12215000	7500	4	2	2	1	0	1	0
4	11410000	7420	4	1	2	1	1	1	0

D. Feature Extraction:

After the variables in the data were modified, the most important features that would contribute to this specific analysis were identified. The correlation between the features to the target variable of price were observed using a heatmap of a correlation matrix in line 14, followed by setting a correlation threshold. Correlation coefficients were calculated and displayed with a histogram to observe which categories go below the threshold. Once those categories were identified, they were either kept or dropped based on their standing. Categories that might not have many correlations with price can be removed in order to only observe the

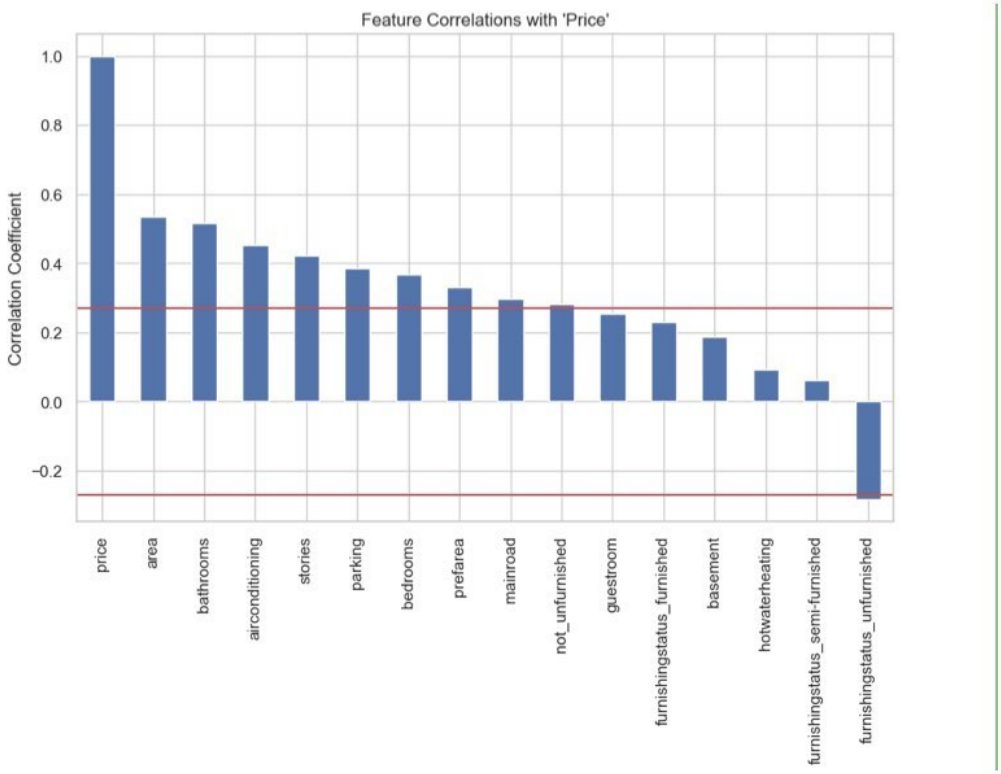
categories that would most likely impact the prices for better analysis of the resulting trends. Lines 18 and 19 demonstrate the steps previously explained.

Line 14:

```
In [14]: # Print the column names after encoding categorical variables

         column_names = encoded_data.columns

         print("Column Names in Encoded DataFrame:", list(column_names))

         Column Names in Encoded DataFrame: ['price', 'area', 'bedrooms', 'bathr
         ooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating
         ', 'airconditioning', 'parking', 'prefarea', 'furnishingstatus_furnishe
         d', 'furnishingstatus_semi-furnished', 'furnishingstatus_unfurnished']
```

Line 18:

```
In [18]: # Set correlation threshold and compute correlations
         correlation_threshold = 0.27
         correlations = interaction_df.corr()['price'].sort_values(ascending=False)

         # Plot the correlation coefficients
         plt.figure(figsize=(10, 6))
         correlations.plot(kind='bar')
         plt.axhline(y=correlation_threshold, color='r', linestyle='-')
         plt.axhline(y=-correlation_threshold, color='r', linestyle='-')
         plt.title("Feature Correlations with 'Price'")
         plt.xlabel('Features')
         plt.ylabel('Correlation Coefficient')
         plt.show()

         # List features to drop by threshold
         features_to_drop = correlations[correlations.abs() < correlation_threshold].index.tolis
         print("\nFeatures to drop (correlation < 0.27):", features_to_drop)
```



Line 19:

```
In [19]: # Set correlation threshold
         # Select columns with correlation above it
         correlation_threshold = 0.27
         selected_columns = correlations[correlations.abs() >= correlation_threshold].index

         print("Selected Columns (correlation >= 0.27):", list(selected_columns))

         Selected Columns (correlation >= 0.27): ['price', 'area', 'bathrooms', 'airconditionin
         g', 'stories', 'parking', 'bedrooms', 'prefarea', 'mainroad', 'not_unfurnished', 'furn
         ishingstatus_unfurnished']
```

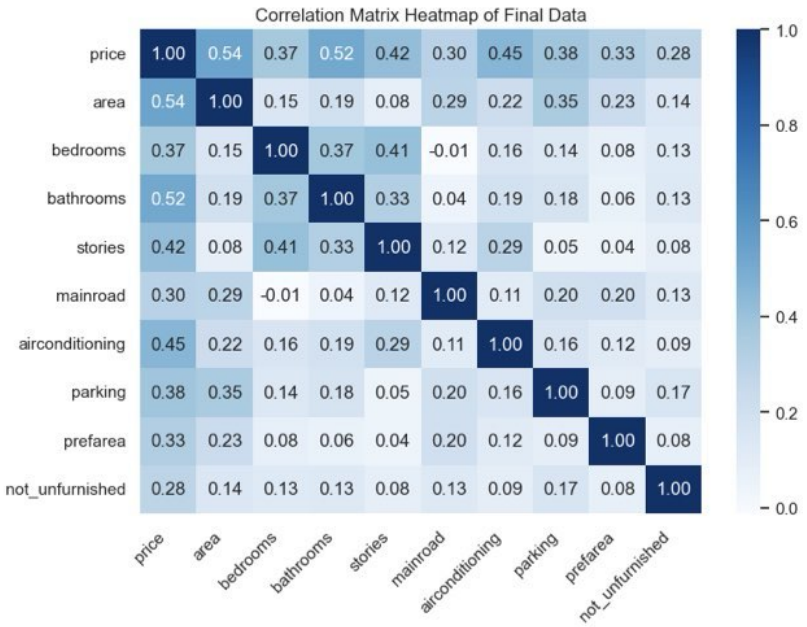
E. Data Visualization:

After appropriate measures were taken to prepare the dataset, and the unnecessary columns were dropped, the next step that was followed was data visualization. Therefore, lines 20 and 21 demonstrate the updated total columns list and correlation matrix heatmap for the final set of features that will be used for this analysis.

In [20]:
```
# Define columns to drop them

columns_to_drop = ['guestroom', 'basement', 'hotwaterheating',
                   'furnishingstatus_furnished', 'furnishingstatus_semi-furnished',
                   'furnishingstatus_unfurnished']

final_data = interaction_df.drop(columns_to_drop, axis=1)

# Show final dataset
display(final_data.head())
```

	price	area	bedrooms	bathrooms	stories	mainroad	airconditioning	parking	prefarea	not_
0	13300000	7420	4	2	3	1	1	2	1	
1	12250000	8960	4	4	4	1	1	3	0	
2	12250000	9960	3	2	2	1	0	2	1	
3	12215000	7500	4	2	2	1	1	3	1	
4	11410000	7420	4	1	2	1	1	2	0	

In [21]:
```
# Create & display a heatmap for correlation matrix for the final dataset

fig, ax = plt.subplots(figsize=(8, 6))

sns.heatmap(final_data.corr(), annot=True, ax=ax, fmt=".2f", cmap='Blues')

plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.title("Correlation Matrix Heatmap of Final Data")
plt.tight_layout()

plt.show()
```



F. Model Preparation:

Once the dataset was finalized, appropriate steps were then taken to further prepare for the model application by scaling the data using standard scaler and split into training and testing datasets. Lines 46 and 47 portray this part of data preparation.

In [46]:
```
# Extract target variable 'price' & feature matrix X

y = final_data['price']
X = final_data.drop(columns=['price'])

# Display first few rows of feature matrix X
display(X.head())

# Print number of rows & columns in X
print(f"\nTotal Rows: {X.shape[0]}, Total Columns: {X.shape[1]}")
```

	area	bedrooms	bathrooms	stories	mainroad	airconditioning	parking	prefarea	not_unfurnishec
0	7420	4	2	3	1	1	2	1	1
1	8960	4	4	4	1	1	3	0	1
2	9960	3	2	2	1	0	2	1	1
3	7500	4	2	2	1	1	3	1	1
4	7420	4	1	2	1	1	2	0	1

```
Total Rows: 545, Total Columns: 9
```

In [47]:
```
# Create StandardScaler object
scaler = StandardScaler()

# Scale feature matrix X with scaler
X_scaled = scaler.fit_transform(X)
```

III. Experimental Results:

In order to proceed with the actual experimental run, the three chosen algorithms were then applied individually on the data, and performance metrics of R^2, and Mean Squared Error (MSE) were calculated to compare the efficiency of the different algorithms. The R^2 value attempts to explain the variance that the dependent variable can essentially be explained by the

independent or predictor variables to understand the relationship or correlation that the two variables share.

A. Linear Regression:

The linear regression algorithm was applied on the data following the data preparation. The linear regression algorithm provided an R^2 value of 0.62 and a MSE of 1380628736112.17.

```
R-squared (R²): 0.62
Mean Squared Error (MSE): 1380628736112.17
```

B. KNN:

The KNN algorithm was applied on the data following the data preparation. This algorithm provided an R^2 value of 0.69 and a MSE of 1082097159693.61.

```
KNeighborsRegressor(n_neighbors=10) Model Evaluation:
R2 Score: 0.69
MAE: 752955.00
MSE: 1082097159693.61
```

C. Random Forest:

The Random Forest algorithm was also applied on the data following the data preparation. This algorithm provided an R^2 value of 0.63 and a MSE of 1285305313104.46.

```
RandomForestRegressor() Model Evaluation:
R2 Score: 0.63
MAE: 817874.24
MSE: 1285305313104.46
```

D. K-Fold Cross Validation with Linear Regression:

Following those three applications, a hybrid type of model was applied on the data with the algorithm that was found to be most compatible. Therefore as linear regression offered a simple procedure, it was combined with K-Fold cross validation. This hybrid model produced an R^2 value of 0.74 and a MSE of 916632279618.63.

```
LinearRegression() Model Evaluation:
R2 Score: 0.74
MAE: 733750.13
MSE: 916632279618.63
```

IV. Summary:

Therefore by analyzing the respective performance metrics, an understanding was reached regarding the algorithm that was found to be relatively more efficient than the other algorithms used in this study. Linear regression, as explained by its name, works with linear variables and as the binary features and other features such as number of bathrooms or bedrooms tend to typically relate linearly, there may be a correlation with the higher performance. For example, with a higher number of bedrooms, the price may increase, indicating a linear relationship. While this may be a simple concept, that may adequately explain the resulting metrics.

Furthermore, with slightly more complex algorithms such as KNN, or Random Forest, there is the chance of overfitting of data on the model to accommodate for its small size which may contradict the intended purpose. Therefore it is quite possible that for certain types of analysis, using the simpler algorithm may provide better results than complex algorithms. For this same type of predictive analysis, if the dataset was larger and included even more features, whether categorical, binary, and numerical, that made the dataset even more complex, using complex algorithms to accommodate for that change may benefit the outcome of the analysis.

Furthermore the utilization of K-Fold also presented a higher performance metric when associated with linear regression as it was able to provide the result after calculating the average of several R^2 values as well as further avoid overfitting which would provide better performance as it improves the algorithm by going a step further and tying up some loose ends to give a more reliable result.

V. Errors/Lessons Learned:

Although the appropriate measures taken throughout this study accomplished the intended motive of identifying efficient housing prediction algorithms. The best model was identified to be linear regression. Although this model produced performance metrics higher than the others that were used, it is still relatively low, which poses the possibility of potential errors.

Initially when applying data preprocessing, it is crucial to thoroughly clean the data before proceeding witht the next steps. With the dataset used in this analysis, there were no null values found, however there may been an error in identifying those instances which may have incorrectly provided that result. As a result, there may have been resulting changes in the performance metrics, skewing the analysis. However, additional steps of identifying potential errors in the dataset were implemented to completely clean the data.

Another potential drawback faced with this analysis was the multicollinearity of the dataset. With multiple types of categories or features provided, it makes it complex to make the data consistent and uniform. Therefore there was trouble working with the complexity of the dataset, however by transforming the categories based on their components into numerical values or assigning them dummy variables helped to work around that complexity. This problem also avoids the possibility of errors as if the features were maintained in the format they were and multiple steps were taken to individually compare them, there may have been higher chances of error with the inconsistencies. Ensuring that the chosen models were not overfitting the data or vice versa was also essential to avoiding errors in the data as it emphasized the need to balance the complexity of the model with the dataset when attempting to obtain accuracy performance metrics.

By understanding the errors that this analysis poses and trying to determine the best ways to avoid them enables further learning the methods to best work through datasets when preprocessing them, extracting features, visualizing them and running the actual models for evaluation.

Future Implementations:

As the economy changes owing to unavoidable circumstances such as pandemics, the housing market will also get affected. Therefore, when attempting to build a prediction model, it is best to stay updated with the trends. For the future, appropriate implementations can be established in order to keep the data updated with each period. Furthermore, additional algorithms can be utilized to observe whether there may be other algorithms that offer better performance and prediction results. Another future implementation may be the integration of additional housing categories to further understand the relationship between price and the respective categories.

Benefits to Society:

The economy is constantly wavering. There may be increases and falls in the stock market. People may become unemployed as a result of certain circumstances. On the other hand, someone may be starting a new job that pays really well in a popular city. Someone may desire to live somewhere near the water, or a place with 3 bathrooms. There may be different things that an individual looks for and they may come from different positions. Those individuals may all do their own research and attempt to find housing that fits them best. However that independent research may take a very long time, pose as a hassel and may not always be accurate. Moreover, there are real estate agents or housing developers that would like to know what features of

housing that most target customers are looking for and might find it difficult to do their own share of research over a large dataset.

This predictive analysis essentially benefits society by offering a method which is much easier and efficient in terms of speed and time. Customers can find accurate housing much faster and that may be a much better fit than options found through independent research through other means. Real estate agents on the other hand can feel much better at their job in providing housing that serves their customers best and will also be able to understand the trends in housing to best market their products and its associated features. With more appropriate housing being catered to the right groups of people, customers will be able to live in affordable housing, and have more time and resources to take care of other needs. Society may then be able to function better in terms of problems of the economy, unemployment, population of homeless individuals as well as others.

***The code in this study was written from scratch, with code in one cell obtained from an external source for the sake of reduction of the number of cells.**

- **References:**

.Dataset: Housing Prices Dataset. (2022, January 12). Kaggle. https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

Source code for Cell 50: House Price Prediction. (n.d.). Github.com/ Line 87. https://github.com/kumod007/House-Price-Prediction/blob/main/House%20Price%20Prediction.ipynb

analysis-of-housing-project

December 23, 2023

Machine Learning Final Project

Dataset Used: https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

All the code has been developed from scratch, except for line 50th. The source is mentioned above the cell.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm

from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import KFold, train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

[2]:
```
# Load, read the CSV file and handle exceptions

try:
    df = pd.read_csv("/Users/giorgi/Desktop/Housing.csv")
except FileNotFoundError:
    print("The file was not found, Please check the file path.")
except Exception as e:
    print("An error occurred:", e)
```

[3]:
```
# Calculate and print the number of rows and columns

num_records = df.shape[0]
num_columns = df.shape[1]

print(f"The dataset contains {num_records} records (rows) and {num_columns}␣
 ↪attributes (columns).")
```

```
The dataset contains 545 records (rows) and 13 attributes (columns).
```

[4]:
```
# Display summary without verbose column details

df.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Columns: 13 entries, price to furnishingstatus
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

[5]:
```
# Summarize and display statistics for object-type columns

df_describe_objects = df.describe(include="object")
styled_table = df_describe_objects.style.set_table_attributes("style='display:
  ↪inline'").set_caption('Descriptive Statistics for Object-type Columns')
display(styled_table)
```

```
<pandas.io.formats.style.Styler at 0x13a9d81d0>
```

[6]:
```
# Generate and display statistics for numerical columns

df_describe_numerical = df.describe(include=[np.int64, np.float64])
styled_numerical_table = df_describe_numerical.style.
  ↪set_table_attributes("style='display:inline'").set_caption('Descriptive␣
  ↪Statistics for Numerical Columns')
display(styled_numerical_table)
```

```
<pandas.io.formats.style.Styler at 0x13c788f90>
```

[7]:
```
# Display sample 5 random rows

df_sample = df.sample(5, random_state=42)
styled_sample = df_sample.style.set_table_attributes("style='display:inline'").
  ↪set_caption('Random Sample of 5 Rows')
display(styled_sample)
```

```
<pandas.io.formats.style.Styler at 0x13cc48a50>
```

[8]:
```
# find and display rows that has some but not all null values

rows_with_some_nulls = df.isnull().any(axis=1)
rows_not_completely_null = ~df.isnull().all(axis=1)
null_entries_df = df[rows_with_some_nulls & rows_not_completely_null]

display(null_entries_df)
```

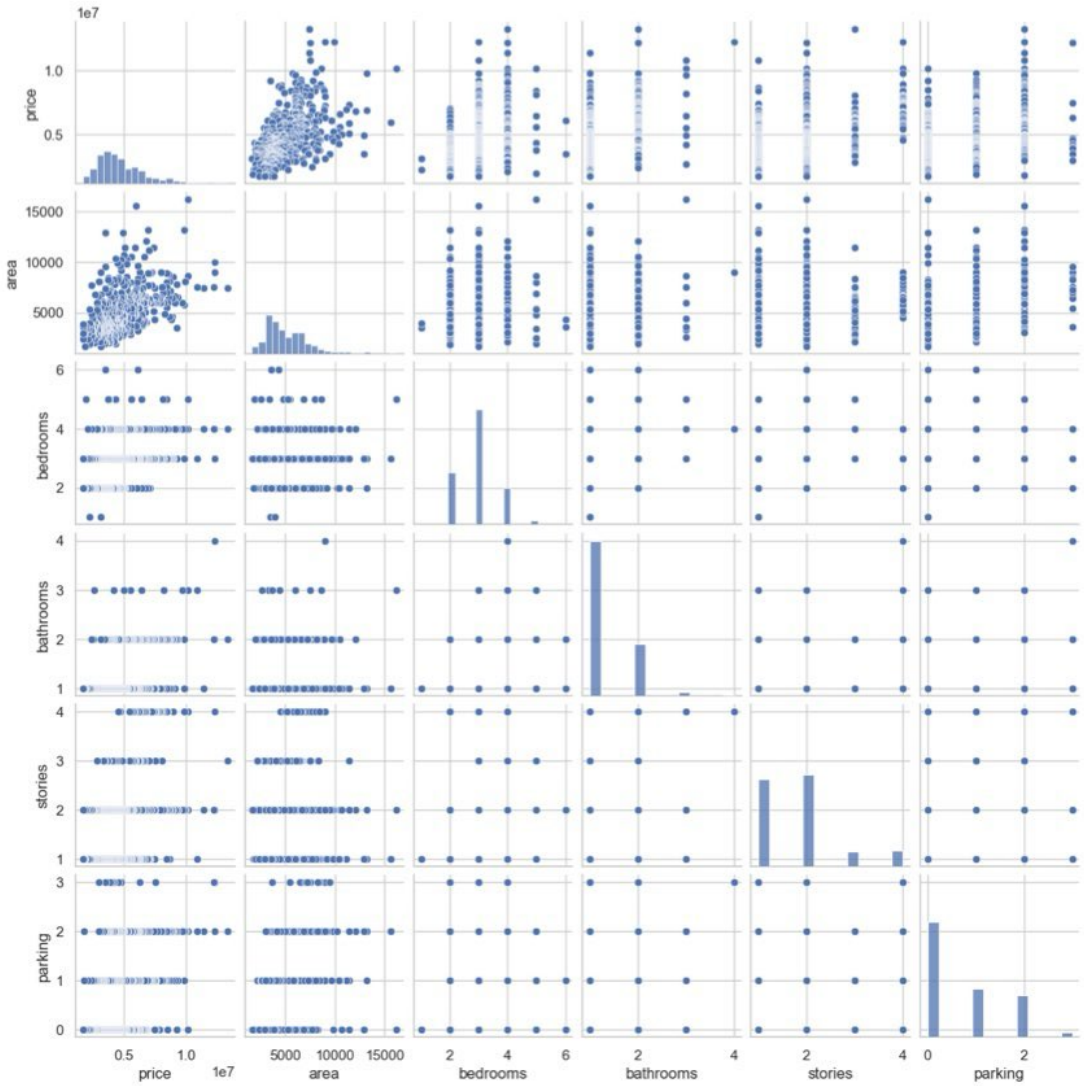
```
Empty DataFrame
```

```
Columns: [price, area, bedrooms, bathrooms, stories, mainroad, guestroom,␣
 ↪basement, hotwaterheating, airconditioning, parking, prefarea,␣
 ↪furnishingstatus]
Index: []
```

[9]:
```
# Create a histogram of the 'price' column

plt.style.use('ggplot')
plt.figure(figsize=(10, 6))

df['price'].plot(kind='hist', bins=10, title='Price Distribution')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Histogram of Price')

plt.show()
```

[10]:
```
# Display a pairplot from dataset

plt.figure(figsize=(10, 10))
sns.set(style="whitegrid")

ax = sns.pairplot(df, height=2)
plt.show()
```

<Figure size 1000x1000 with 0 Axes>



[11]:
```
# Convert binary columns to (1/0)
# Handle unexpected values/missing columns

binary_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']

for col in binary_cols:
    if col in df.columns and df[col].isin(['yes', 'no']).all():
        df[col] = df[col].map({'yes': 1, 'no': 0})
    else:
        print(f"Column '{col}' is missing or has unexpected values.")
```

```
display(df)
```

```
        price  area  bedrooms  bathrooms  stories  mainroad  guestroom  \
0    13300000  7420         4          2        3         1          0
1    12250000  8960         4          4        4         1          0
2    12250000  9960         3          2        2         1          0
3    12215000  7500         4          2        2         1          0
4    11410000  7420         4          1        2         1          1
..        ...   ...       ...        ...      ...       ...        ...
540   1820000  3000         2          1        1         1          0
541   1767150  2400         3          1        1         0          0
542   1750000  3620         2          1        1         1          0
543   1750000  2910         3          1        1         0          0
544   1750000  3850         3          1        2         1          0

     basement  hotwaterheating  airconditioning  parking  prefarea  \
0           0                0                1        2         1
1           0                0                1        3         0
2           1                0                0        2         1
3           1                0                1        3         1
4           1                0                1        2         0
..        ...              ...              ...      ...       ...
540         1                0                0        2         0
541         0                0                0        0         0
542         0                0                0        0         0
543         0                0                0        0         0
544         0                0                0        0         0

    furnishingstatus
0          furnished
1          furnished
2     semi-furnished
3          furnished
4          furnished
..               ...
540      unfurnished
541   semi-furnished
542      unfurnished
543        furnished
544      unfurnished

[545 rows x 13 columns]
```

[12]:
```
# Print a list of all categorical columns

cat_cols = df.select_dtypes(include='object').columns
```

```
cat_data = df[cat_cols]

print("Categorical Columns:", cat_cols.tolist())
```

```
Categorical Columns: ['furnishingstatus']
```

[13]:
```
# Create dummy variables for categorical columns
# Show the first few rows

encoded_data = pd.get_dummies(df, columns=cat_cols)

display(encoded_data.head())
```

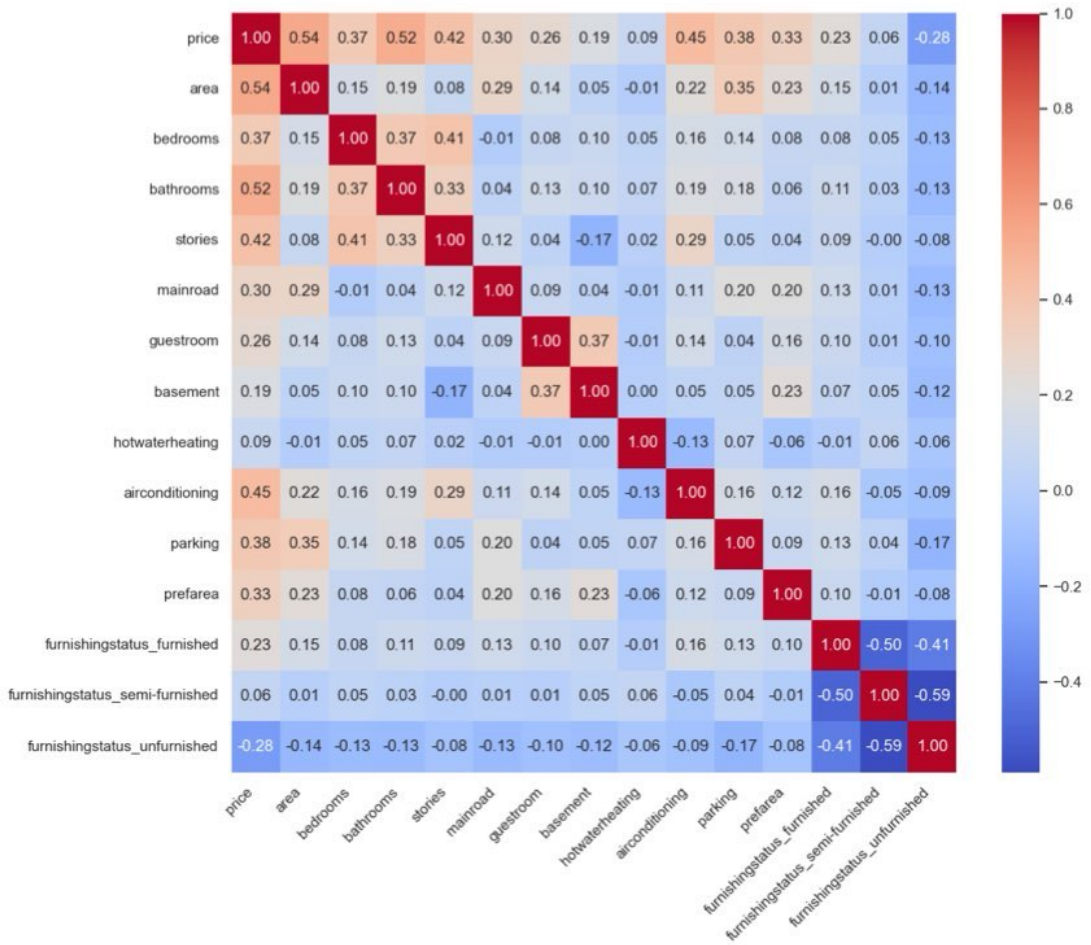
```
      price  area  bedrooms  bathrooms  stories  mainroad  guestroom  \
0  13300000  7420         4          2        3         1          0
1  12250000  8960         4          4        4         1          0
2  12250000  9960         3          2        2         1          0
3  12215000  7500         4          2        2         1          0
4  11410000  7420         4          1        2         1          1

   basement  hotwaterheating  airconditioning  parking  prefarea  \
0         0                0                1        2         1
1         0                0                1        3         0
2         1                0                0        2         1
3         1                0                1        3         1
4         1                0                1        2         0

   furnishingstatus_furnished  furnishingstatus_semi-furnished  \
0                           1                                0
1                           1                                0
2                           0                                1
3                           1                                0
4                           1                                0

   furnishingstatus_unfurnished
0                             0
1                             0
2                             0
3                             0
4                             0
```

[14]:
```
# Print the column names after encoding categorical variables

column_names = encoded_data.columns

print("Column Names in Encoded DataFrame:", list(column_names))
```

```
Column Names in Encoded DataFrame: ['price', 'area', 'bedrooms', 'bathrooms',
```

```
'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating',
'airconditioning', 'parking', 'prefarea', 'furnishingstatus_furnished',
'furnishingstatus_semi-furnished', 'furnishingstatus_unfurnished']
```

[15]:
```
# Create & show a heatmap of the correlation matrix for the encoded data

fig, ax = plt.subplots(figsize=(12, 10))

corr_matrix = encoded_data.corr()
sns.heatmap(corr_matrix, annot=True, ax=ax, fmt=".2f", cmap='coolwarm')

plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()

plt.show()
```



[16]:
```
# Sum columns to create a new feature
# Handle missing columns

columns_to_sum = ['furnishingstatus_semi-furnished',
 'furnishingstatus_furnished']

if all(col in encoded_data.columns for col in columns_to_sum):
    interaction_df = encoded_data.copy()
    interaction_df['not_unfurnished'] = interaction_df[columns_to_sum].
 sum(axis=1)
else:
    print("required columns are missing")

display(interaction_df.head())
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	\
0	13300000	7420	4	2	3	1	0	
1	12250000	8960	4	4	4	1	0	
2	12250000	9960	3	2	2	1	0	
3	12215000	7500	4	2	2	1	0	
4	11410000	7420	4	1	2	1	1	

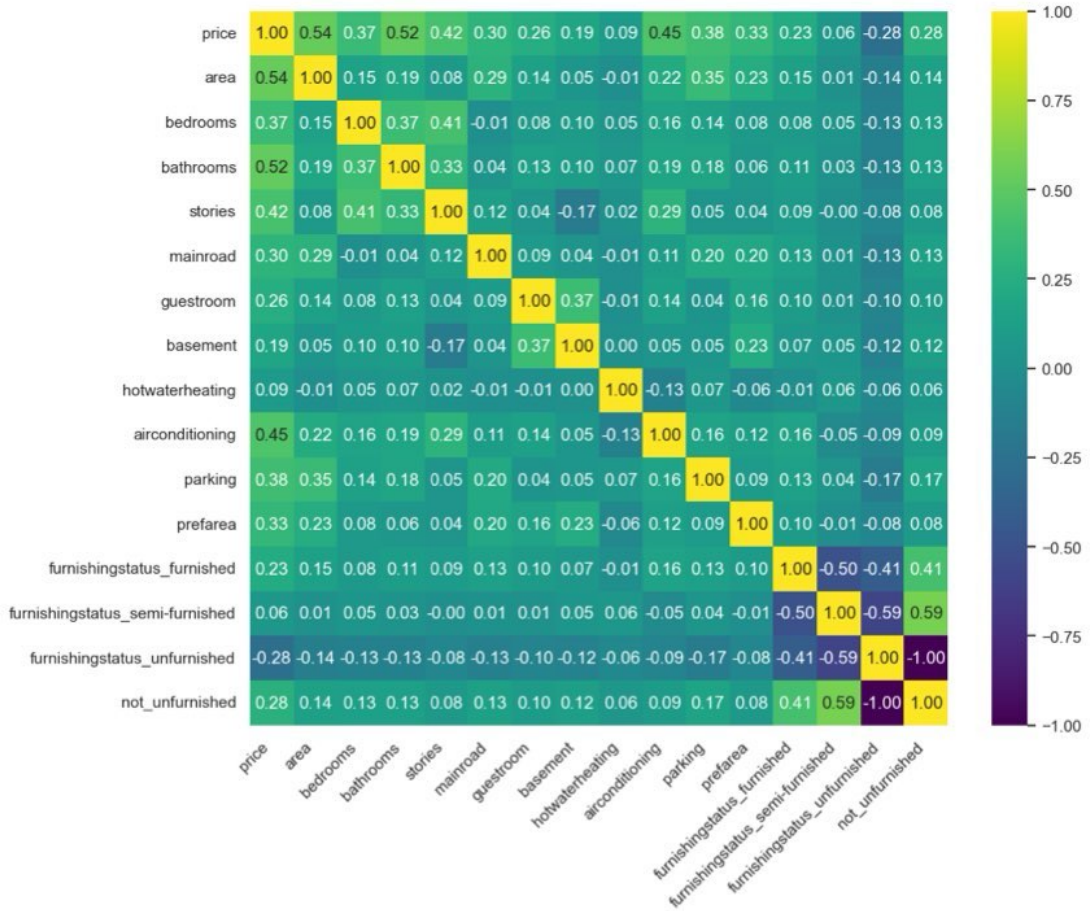
	basement	hotwaterheating	airconditioning	parking	prefarea	\
0	0	0	1	2	1	
1	0	0	1	3	0	
2	1	0	0	2	1	
3	1	0	1	3	1	
4	1	0	1	2	0	

	furnishingstatus_furnished	furnishingstatus_semi-furnished	\
0	1	0	
1	1	0	
2	0	1	
3	1	0	
4	1	0	

	furnishingstatus_unfurnished	not_unfurnished
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

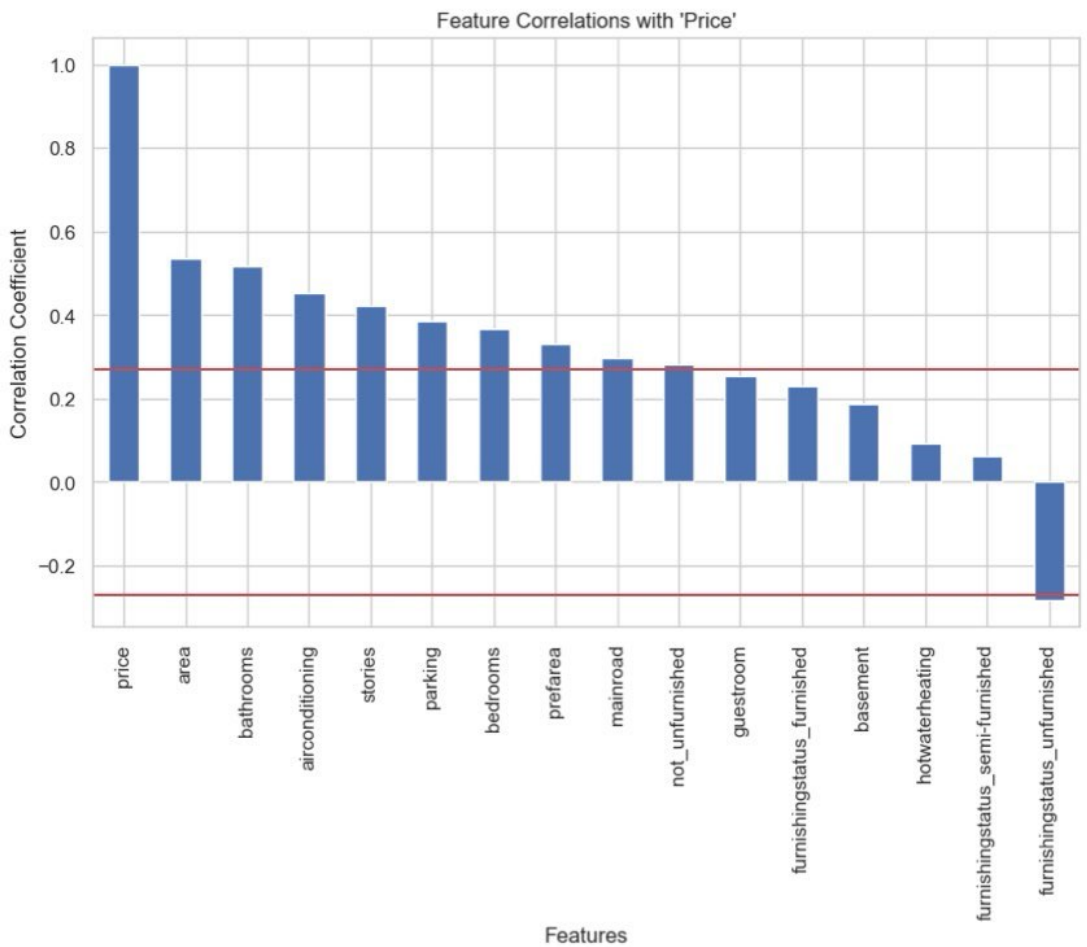
[17]:
```
# Create & display a heatmap of the correlation matrix with new feature

fig, ax = plt.subplots(figsize=(11, 9))
```

```
corr_matrix_interaction = interaction_df.corr()
sns.heatmap(corr_matrix_interaction, annot=True, ax=ax, fmt=".2f",␣
 ↪cmap='viridis')

plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()

plt.show()
```



[18]:
```
# Set correlation threshold and compute correlations
correlation_threshold = 0.27
correlations = interaction_df.corr()['price'].sort_values(ascending=False)

# Plot the correlation coefficients
plt.figure(figsize=(10, 6))
correlations.plot(kind='bar')
```

```
plt.axhline(y=correlation_threshold, color='r', linestyle='-')
plt.axhline(y=-correlation_threshold, color='r', linestyle='-')
plt.title("Feature Correlations with 'Price'")
plt.xlabel('Features')
plt.ylabel('Correlation Coefficient')
plt.show()

# List features to drop by threshold
features_to_drop = correlations[correlations.abs() < correlation_threshold].
  ↪index.tolist()
print("\nFeatures to drop (correlation < 0.27):", features_to_drop)
```




```
Features to drop (correlation < 0.27): ['guestroom',
'furnishingstatus_furnished', 'basement', 'hotwaterheating',
'furnishingstatus_semi-furnished']
```

[19]:
```
# Set correlation threshold
# Select columns with correlation above it
correlation_threshold = 0.27
selected_columns = correlations[correlations.abs() >= correlation_threshold].
 ↪index

print("Selected Columns (correlation >= 0.27):", list(selected_columns))
```

```
Selected Columns (correlation >= 0.27): ['price', 'area', 'bathrooms',
'airconditioning', 'stories', 'parking', 'bedrooms', 'prefarea', 'mainroad',
'not_unfurnished', 'furnishingstatus_unfurnished']
```

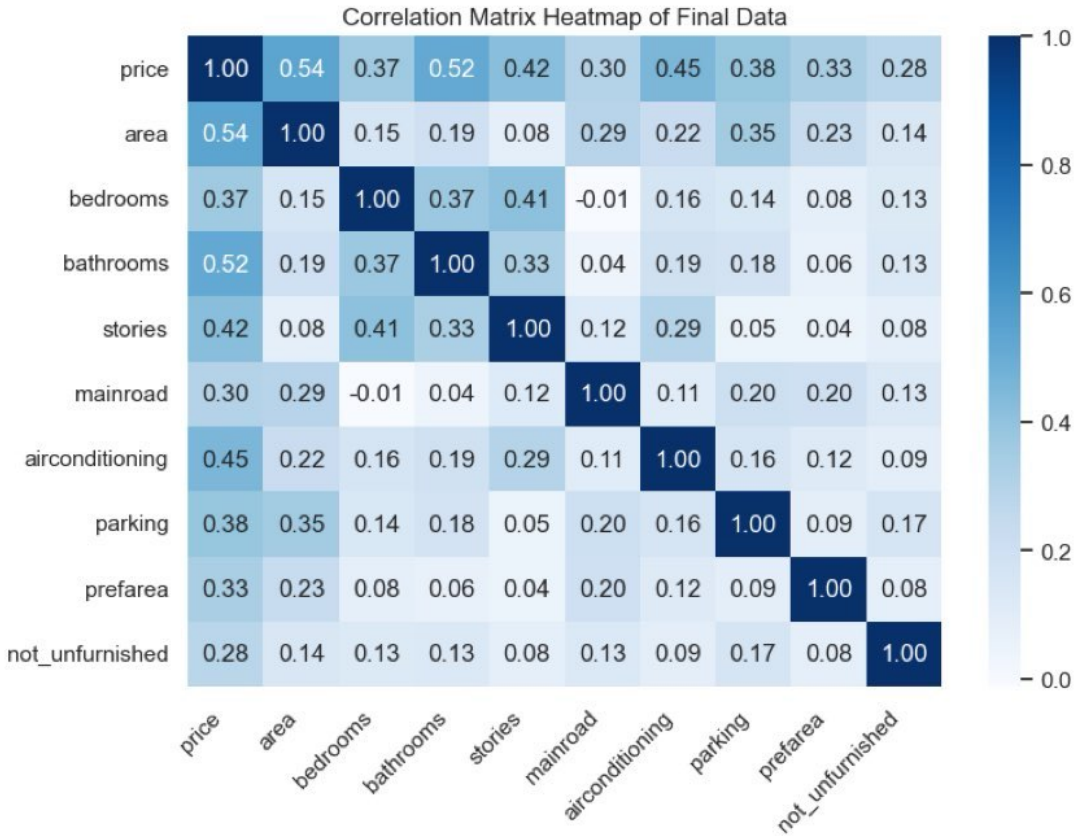
[20]:
```
# Define columns to drop them

columns_to_drop = ['guestroom', 'basement', 'hotwaterheating',
                   'furnishingstatus_furnished',␣
 ↪'furnishingstatus_semi-furnished',
                   'furnishingstatus_unfurnished']

final_data = interaction_df.drop(columns_to_drop, axis=1)

# Show final dataset
display(final_data.head())
```

	price	area	bedrooms	bathrooms	stories	mainroad	airconditioning	\
0	13300000	7420	4	2	3	1	1	
1	12250000	8960	4	4	4	1	1	
2	12250000	9960	3	2	2	1	0	
3	12215000	7500	4	2	2	1	1	
4	11410000	7420	4	1	2	1	1	

	parking	prefarea	not_unfurnished
0	2	1	1
1	3	0	1
2	2	1	1
3	3	1	1
4	2	0	1

[21]:
```
# Create & display a heatmap for correlation matrix for the final dataset

fig, ax = plt.subplots(figsize=(8, 6))

sns.heatmap(final_data.corr(), annot=True, ax=ax, fmt=".2f", cmap='Blues')

plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.title("Correlation Matrix Heatmap of Final Data")
```

```
plt.tight_layout()

plt.show()
```



1 Linear Regression

[22]:
```
# Define feature columns and check for missing

feature_columns = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
                   'airconditioning', 'parking', 'prefarea', 'not_unfurnished']

missing_columns = [col for col in feature_columns if col not in final_data.
  ↪columns]
if missing_columns:
    # Print missing feature columns
    print(f"Missing columns: {missing_columns}")
else:
    # Select features and target variable
```

```
    X = final_data[feature_columns]
    y = final_data['price']
```

[23]:
```
# Split data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪shuffle=True)

# Print shapes
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
```

```
Shape of X_train: (381, 9)
Shape of X_test: (164, 9)
```

[24]:
```
# Define and print numerical columns

num_cols = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']

print("Numerical Columns:", num_cols)
```

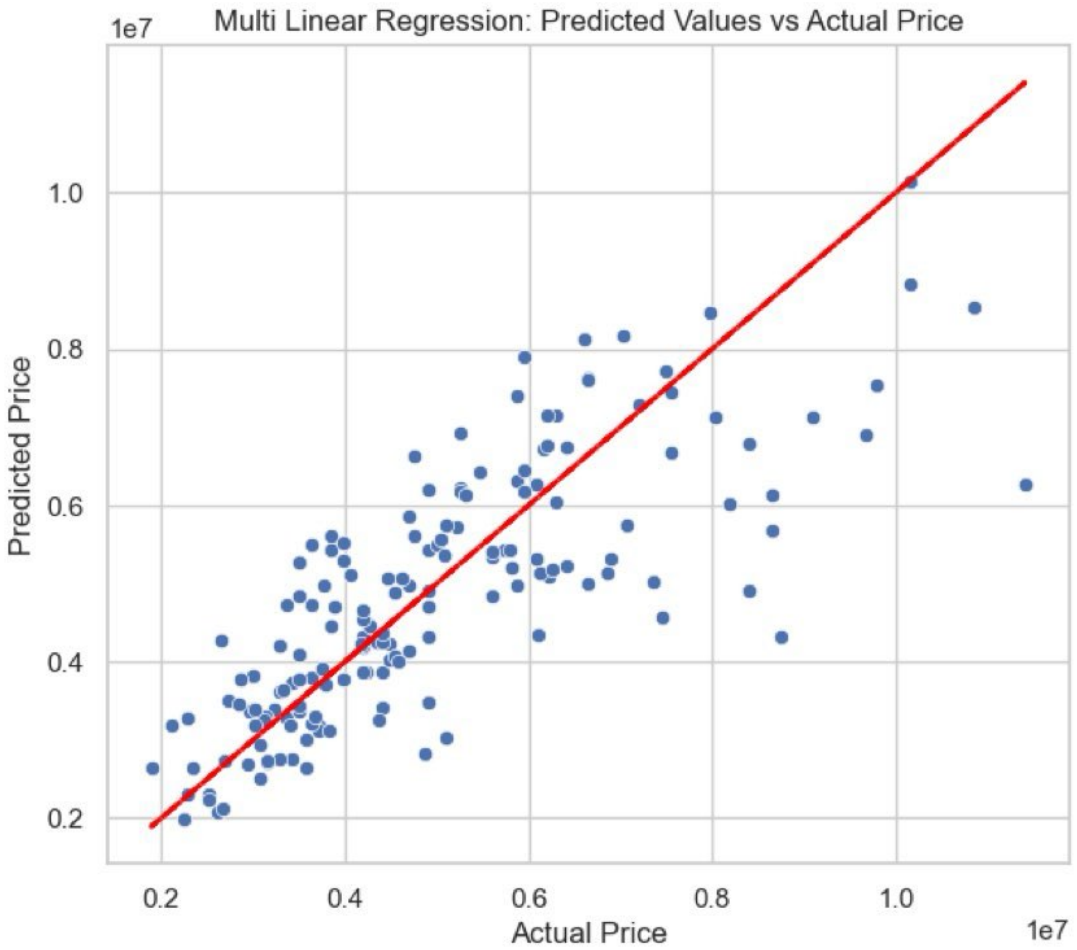
```
Numerical Columns: ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
```

[25]:
```
# Create and fit a linear regression model
reg_multi = LinearRegression()

reg_multi.fit(X_train, y_train)
```

[25]:
```
LinearRegression()
```

[26]:
```
# Predict house prices for test data by trained linear regression model

y_pred = reg_multi.predict(X_test)
```

[27]:
```
# Plot actual against predicted prices and reference line

fig, ax = plt.subplots(figsize=(7, 6))

sns.scatterplot(x=y_test, y=y_pred, ax=ax)
ax.set(xlabel="Actual Price", ylabel="Predicted Price", title="Multi Linear␣
 ↪Regression: Predicted Values vs Actual Price")

plt.plot(y_test, y_test, color='red', linewidth=2, linestyle="--")

plt.show()
```

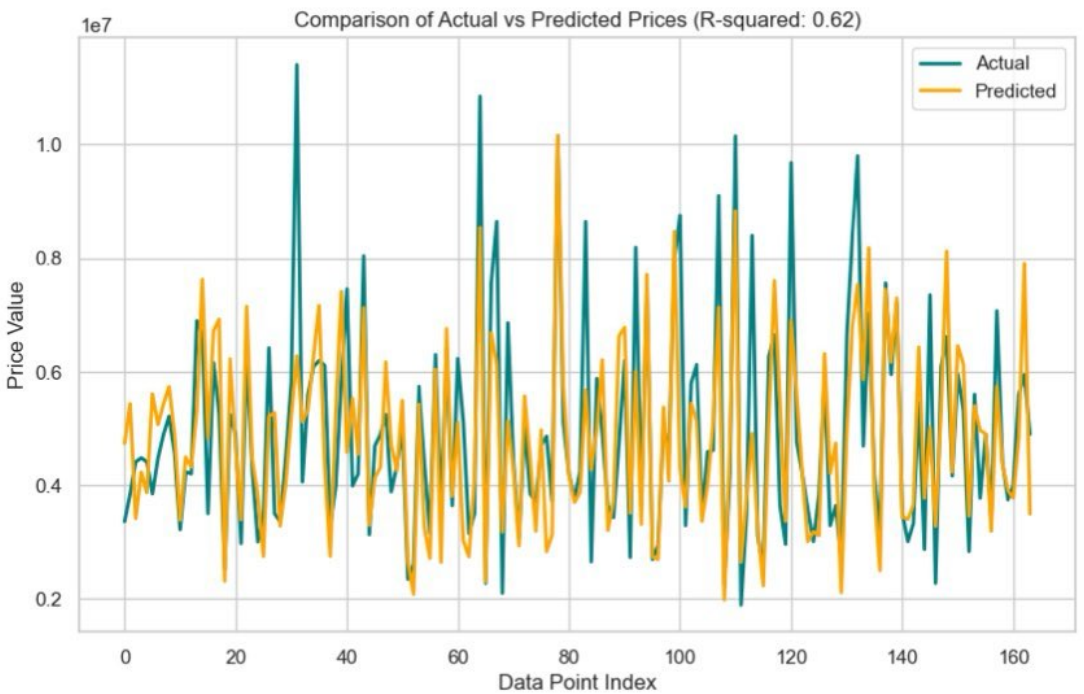


[28]:
```
# Calculate & print R squared and MSE (Mean Squared Error)

r2 = metrics.r2_score(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)

print(f"R-squared (R²): {r2:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
```

```
R-squared (R²): 0.62
Mean Squared Error (MSE): 1380628736112.17
```

[29]:
```
# Plot & compare actual against predicted prices with R-squared value

plt.figure(figsize=(10, 6))
```

```
plt.plot(np.arange(len(y_test)), y_test, label='Actual', color='teal',
 ↪linewidth=2)

plt.plot(np.arange(len(y_test)), y_pred, label='Predicted', color='orange',
 ↪linewidth=2)

plt.xlabel('Data Point Index')
plt.ylabel('Price Value')
plt.title(f'Comparison of Actual vs Predicted Prices (R-squared: {r2:.2f})')

plt.legend()

plt.show()
```



2 Kfold

[30]:
```
# Select features and extract values

selected_features = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
 ↪'airconditioning', 'parking', 'prefarea', 'not_unfurnished']

X = final_data[selected_features].values
```

```
# Extract the target variable

y = final_data['price'].values
```

[31]:
```
# Split the data into training and test sets
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.2, shuffle=True)

# Print shapes
print("Shape of X_train:", X_tr.shape)
print("Shape of X_test:", X_ts.shape)
```

```
Shape of X_train: (436, 9)
Shape of X_test: (109, 9)
```

[32]:
```
# Define variables for cross-validation & tracking the best model

nfold = 30

r2_train = np.zeros(nfold)
r2_val = np.zeros(nfold)
r2_test = np.zeros(nfold)

best_r2 = 0
best_training_data = None
```

[33]:
```
# Cross-validation with KFold & track R-squared values

kf = KFold(n_splits=nfold, shuffle=True)
kf.get_n_splits(X_tr)


for isplit, idx in enumerate(kf.split(X_tr)):

    X_kfold_train, X_val, y_kfold_train, y_val = X_tr[idx[0]], X_tr[idx[1]],
 ↪y_tr[idx[0]], y_tr[idx[1]]

    # Fit a linear regression model
    model = LinearRegression()
    model.fit(X_kfold_train, y_kfold_train)

    # Predictions for training, validation, and test sets
    y_train_pred = model.predict(X_kfold_train)
    y_val_pred = model.predict(X_val)
    y_test_pred = model.predict(X_ts)

    # Calculate R-squared values for each fold
    r2_train[isplit] = metrics.r2_score(y_kfold_train, y_train_pred)
```

```
    r2_val[isplit] = metrics.r2_score(y_val, y_val_pred)
    r2_test[isplit] = metrics.r2_score(y_ts, y_test_pred)

    # Update best R-squared & training data if a better model is found
    if r2_val[isplit] > best_r2:
        best_r2 = r2_val[isplit]
        best_training_data = (X_kfold_train, y_kfold_train)
```

[34]:
```
# Print R-squared values for training data

print("R-squared values (Training):")
print(r2_train)
```

```
R-squared values (Training):
[0.65697311 0.66172963 0.66216135 0.66027583 0.66430575 0.66664779
 0.66587328 0.66559731 0.66310398 0.66159455 0.67858337 0.66220783
 0.67923265 0.67590089 0.66979189 0.6634278  0.66434526 0.66992916
 0.65686018 0.66115538 0.66131917 0.66421263 0.65923713 0.64956501
 0.65602365 0.66414256 0.66461833 0.67048215 0.6601457  0.66313575]
```

[35]:
```
# Print R-squared values for validation data

print("R-squared values (Validation):")
print(r2_val)
```

```
R-squared values (Validation):
[ 0.75056065  0.70886107  0.58208257  0.78106378  0.59249719  0.61309456
  0.61228148  0.37110882  0.49156242  0.65540905  0.02173647  0.68192553
  0.1500783   0.3006959   0.48301536  0.67375029  0.63549881  0.38118362
  0.83520726  0.74442805  0.78688384  0.60915241  0.69047993  0.89074611
  0.88960099  0.63744845  0.61027812 -0.35441142  0.7454444   0.66084802]
```

[36]:
```
# Find the best model number with the highest R-squared value

best_model_number = np.argmax(r2_val)
best_model_number

print("Best Model Number:", best_model_number)
```

```
Best Model Number: 23
```

[37]:
```
# Print R-squared values for the test data

print("R-squared values (Test):")
print(r2_test)
```

```
R-squared values (Test):
[0.61695543 0.62409523 0.62539906 0.62265229 0.62145712 0.619296
```

```
0.62134541 0.62627866 0.62452548 0.62721068 0.62599373 0.62126498
0.62753572 0.6236213  0.62205403 0.62564639 0.62261379 0.62346717
0.62392203 0.62660491 0.62317169 0.62359564 0.62001863 0.62452035
0.62555038 0.62605766 0.62356798 0.62459933 0.62272752 0.6185338 ]
```

[38]:
```
# Print the best R-squared from validation data

print("Best R-squared (Validation):", best_r2)
```

```
Best R-squared (Validation): 0.8907461118831921
```

[39]:
```
# Calculate R-squared value for the test data with best model from validation

r2_test_best_model = r2_test[best_model_number]
r2_test_best_model

print(f"R-squared (Test) for Best Model (Validation): {r2_test_best_model:.2f}")
```

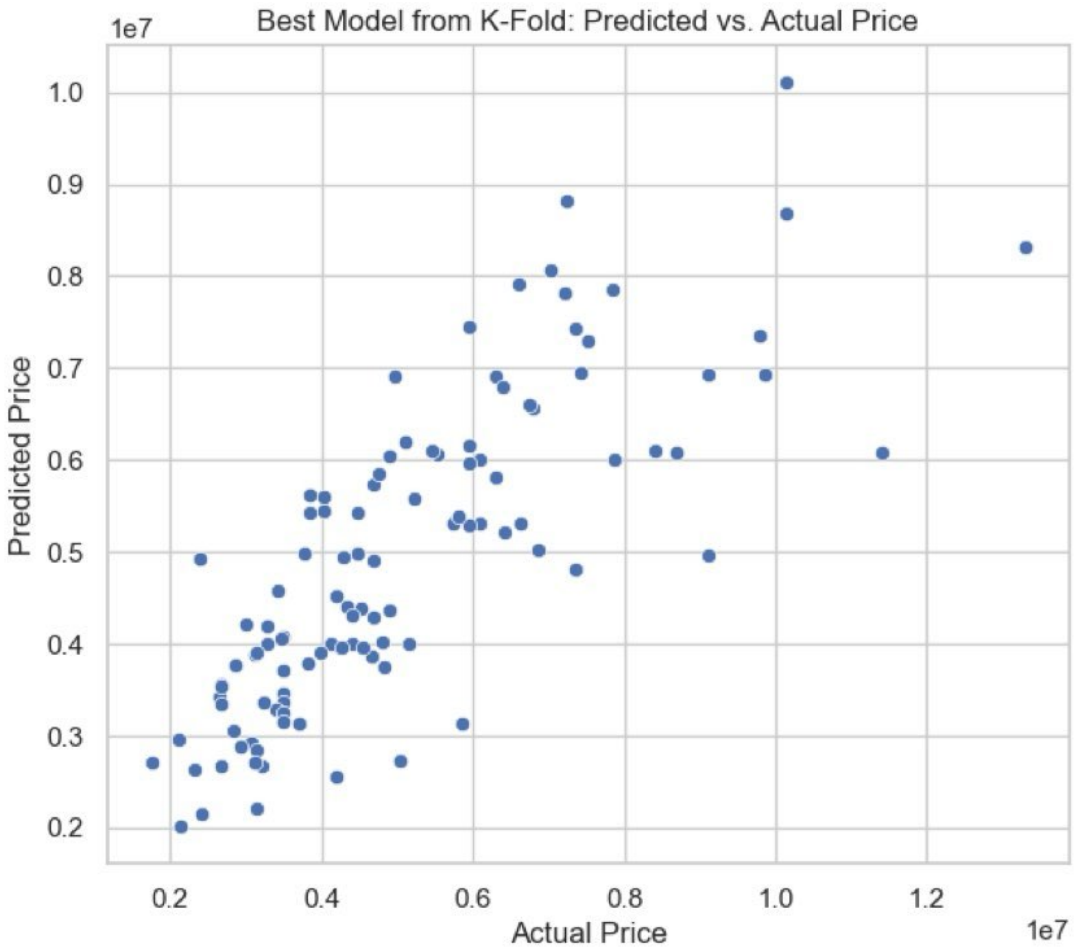
```
R-squared (Test) for Best Model (Validation): 0.62
```

[40]:
```
# Calculate the average R-squared value from validation data

r2_avg = np.mean(r2_val)
r2_avg

print(f"Average R-squared (Validation): {r2_avg:.2f}")
```

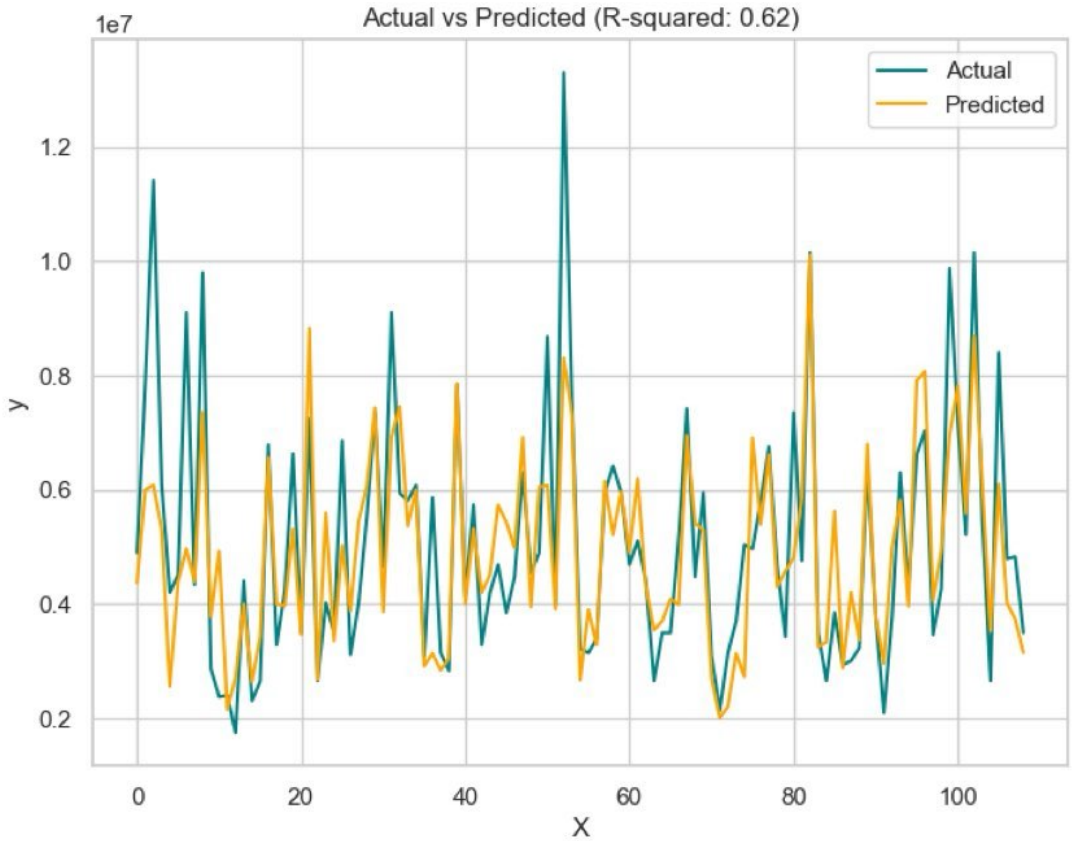
```
Average R-squared (Validation): 0.57
```

[41]:
```
# Fit the best model using the training data with highest validation R-squared

best_model = LinearRegression().fit(best_training_data[0],
 best_training_data[1])

# Predict house prices for the test data with the best model
y_best_pred = best_model.predict(X_ts)

# Calculate R-squared for the test data with the best model
r2_best = metrics.r2_score(y_ts, y_best_pred)
```

[42]:
```
# Show scatterplot comparing actual against predicted prices with the best model

plt.figure(figsize=(7, 6))
sns.scatterplot(x=y_ts, y=y_best_pred)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Best Model from K-Fold: Predicted vs. Actual Price")
plt.show()
```



[43]:
```
# Create plot comparing actual against predicted prices with the best model␣
↪with R-squared

plt.figure(figsize=(8, 6))
plt.plot(np.arange(len(y_ts)), y_ts, label='Actual', color='teal')
plt.plot(np.arange(len(y_ts)), y_best_pred, label='Predicted', color='orange')
plt.xlabel('X')
plt.ylabel('y')
plt.title(f'Actual vs Predicted (R-squared: {r2_test[best_model_number]:.2f})')
plt.legend()
plt.show()
```



[44]:
```
# Print the shape of training data from best fold
print("Shape of the Training Data from the Best Fold:", best_training_data[0].
  ↪shape)
```

```
Shape of the Training Data from the Best Fold: (422, 9)
```

[45]:
```
# Add constant column to the training data & perform OLS regression
X_train_kf = sm.add_constant(best_training_data[0])

lr = sm.OLS(best_training_data[1], X_train_kf).fit()

sorted_indices = np.argsort(lr.params)[::-1]

print("Index\tValue")
for index in sorted_indices:
    value = lr.params[index]
    print(f"{index}\t{value:.4f}")
```

```
Index   Value
3       1151805.4674
```

```
6       789191.5354
8       698605.3255
9       594497.8366
5       586268.5398
4       375868.5821
7       229879.2354
2       127004.2418
1       229.2580
0       -457214.2456
```

3 More Models

[46]:
```
# Extract target variable 'price' & feature matrix X

y = final_data['price']
X = final_data.drop(columns=['price'])

# Display first few rows of feature matrix X
display(X.head())

# Print number of rows & columns in X
print(f"\nTotal Rows: {X.shape[0]}, Total Columns: {X.shape[1]}")
```

```
   area  bedrooms  bathrooms  stories  mainroad  airconditioning  parking  \
0  7420         4          2        3         1                1        2
1  8960         4          4        4         1                1        3
2  9960         3          2        2         1                0        2
3  7500         4          2        2         1                1        3
4  7420         4          1        2         1                1        2

   prefarea  not_unfurnished
0         1                1
1         0                1
2         1                1
3         1                1
4         0                1


Total Rows: 545, Total Columns: 9
```

[47]:
```
# Create StandardScaler object
scaler = StandardScaler()

# Scale feature matrix X with scaler
X_scaled = scaler.fit_transform(X)
```

[48]:
```
# Split scaled feature matrix & target variable into training/test sets
test_size = 0.3
random_state = 0
shuffle = True

x_tra, x_tes, y_tra, y_tes = train_test_split(X_scaled, y, test_size=test_size,
    random_state=random_state, shuffle=shuffle)
```

[49]:
```
# Create lists to store evaluation metrics

r2_value = []
adjusted_r2_value = []
mae_value = []
mse_value = []
rmse_value = []
```

4 The cell below is taken from another project to minimize the number of cells.

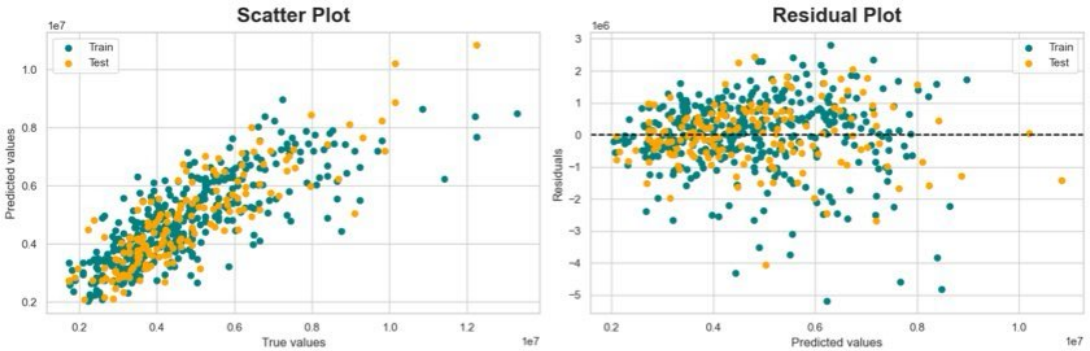
Source of the code below:

https://github.com/kumod007/House-Price-Prediction/blob/main/House%20Price%20Prediction.ipynb (Line 87)

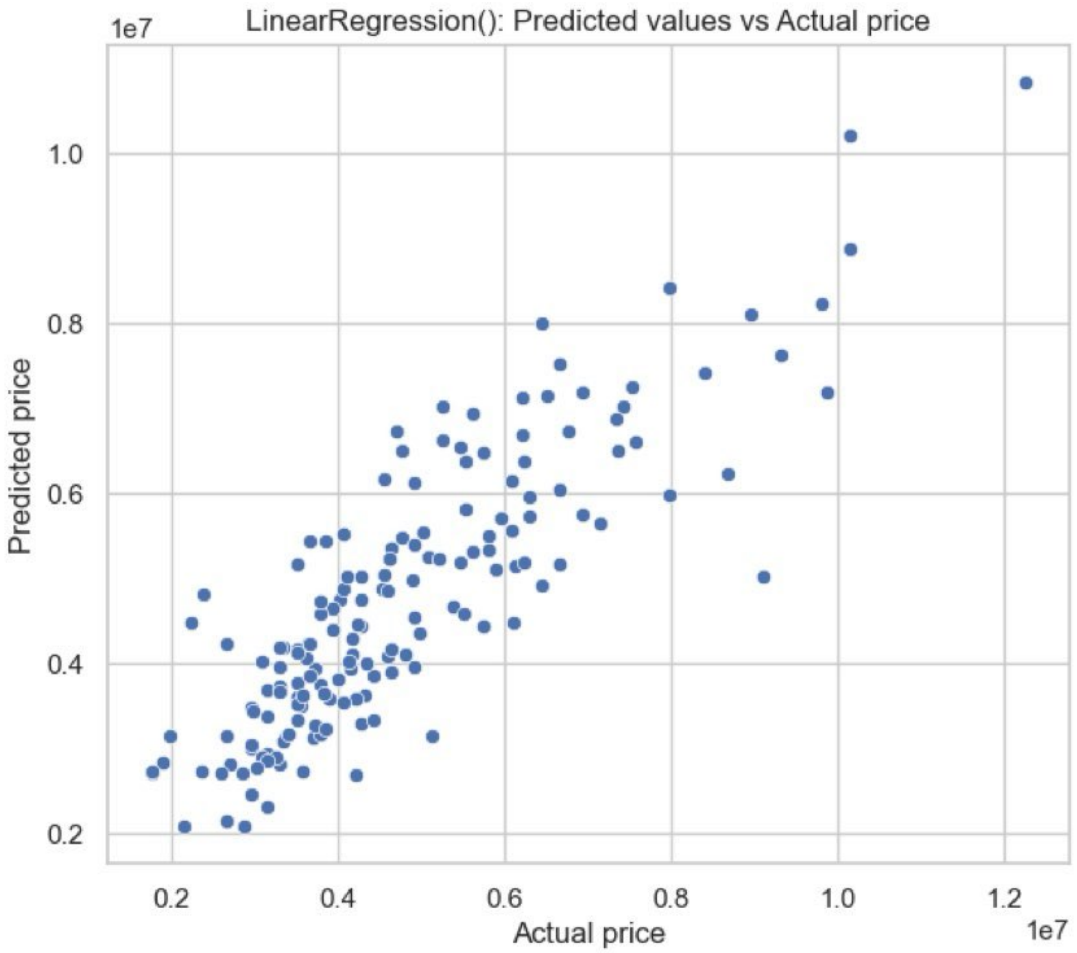
[50]:
```
def model_evaluation(model):

    # Fit the model to training data
    model.fit(x_tra, y_tra)

    # Make predictions on both training & test sets
    y_tr_pred = model.predict(x_tra)
    y_ts_pred = model.predict(x_tes)

    # Calculate evaluation metrics
    mae = mean_absolute_error(y_tes, y_ts_pred)
    mse = mean_squared_error(y_tes, y_ts_pred)
    r2 = r2_score(y_tes, y_ts_pred)

    # Append metrics to lists
    mae_value.append(mae)
    mse_value.append(mse)
    r2_value.append(r2)

    # Print evaluation metrics
    print(f"{model} Model Evaluation:")
    print(f"R2 Score: {r2:.2f}")
    print(f"MAE: {mae:.2f}")
```

```
    print(f"MSE: {mse:.2f}")

    # Create & display scatter and residual plots
    plt.figure(figsize=(15, 5))

    plt.subplot(121)
    plt.scatter(y_tra, y_tr_pred, color='teal', label='Train')
    plt.scatter(y_tes, y_ts_pred, color='orange', label='Test')
    plt.xlabel('True values')
    plt.ylabel('Predicted values')
    plt.legend()
    plt.title('Scatter Plot', fontweight="bold", size=20, pad=10)

    plt.subplot(122)
    plt.scatter(y_tr_pred, y_tr_pred - y_tra, color='teal', label='Train')
    plt.scatter(y_ts_pred, y_ts_pred - y_tes, color='orange', label='Test')
    plt.axhline(y=0, color='black', linestyle='--')
    plt.xlabel('Predicted values')
    plt.ylabel('Residuals')
    plt.legend()
    plt.title('Residual Plot', fontweight="bold", size=20, pad=10)

    plt.tight_layout()
    plt.show()

    # Display scatterplot of actual against predicted prices
    plt.figure(figsize=(7, 6))
    sns.scatterplot(data=x_tes, x=y_tes, y=y_ts_pred).set(xlabel="Actual␣
↪price", ylabel="Predicted price", title=f"{model}: Predicted values vs␣
↪Actual price")

    # Perform linear regression
    # Print parameter values
    X_train_lm = sm.add_constant(x_tra)
    lr = sm.OLS(y_tra, X_train_lm).fit()
    params_sorted = lr.params.sort_values(ascending=False)
    print(f"\nParameters of {model} Model:")
    print(params_sorted)

# Call model_evaluation function
model_evaluation(model)
```

```
LinearRegression() Model Evaluation:
R2 Score: 0.74
MAE: 733750.13
MSE: 916632279618.63
```

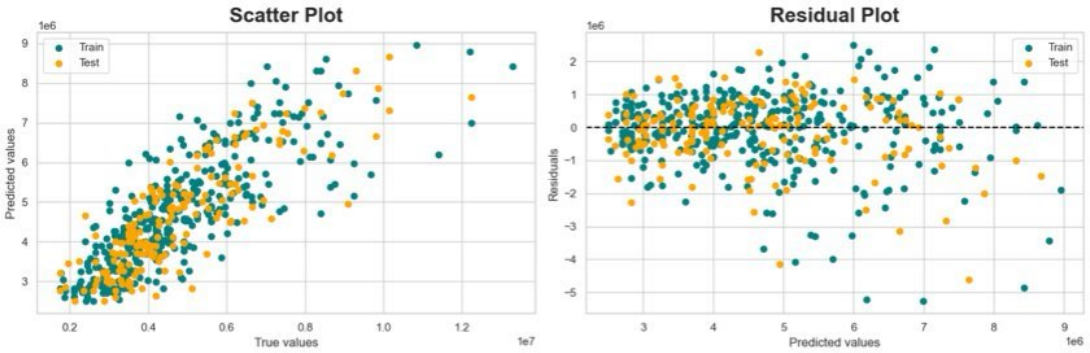



```
Parameters of LinearRegression() Model:
const    4.763927e+06
x3       5.400685e+05
x1       5.224118e+05
x6       3.958922e+05
x8       3.409127e+05
x4       3.216599e+05
x7       2.245776e+05
x9       2.103494e+05
x5       1.713070e+05
x2       1.437099e+05
dtype: float64
```

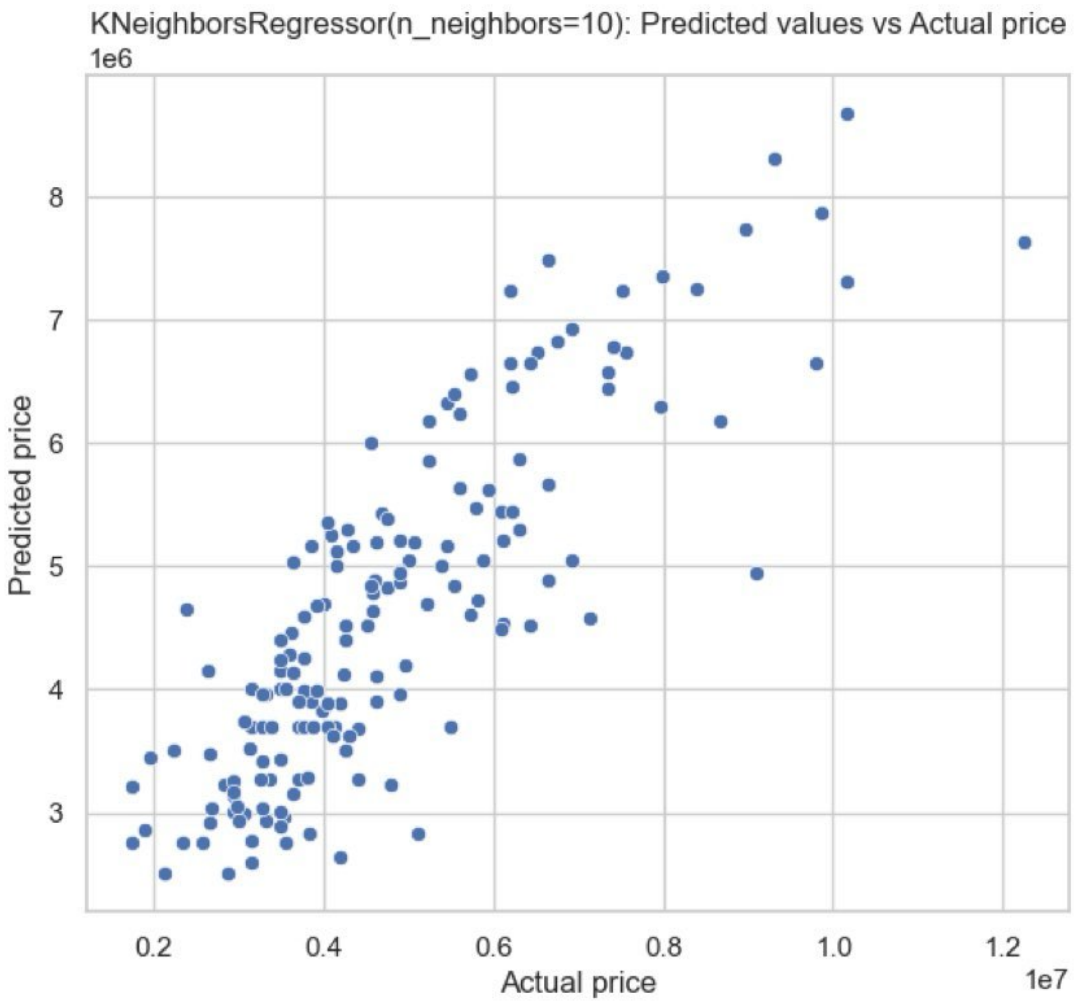


[51]:
```
# Call model_evaluation function with KNeighborsRegressor

model_evaluation(KNeighborsRegressor(n_neighbors=10))
```

```
KNeighborsRegressor(n_neighbors=10) Model Evaluation:
R2 Score: 0.69
MAE: 752955.00
MSE: 1082097159693.61
```

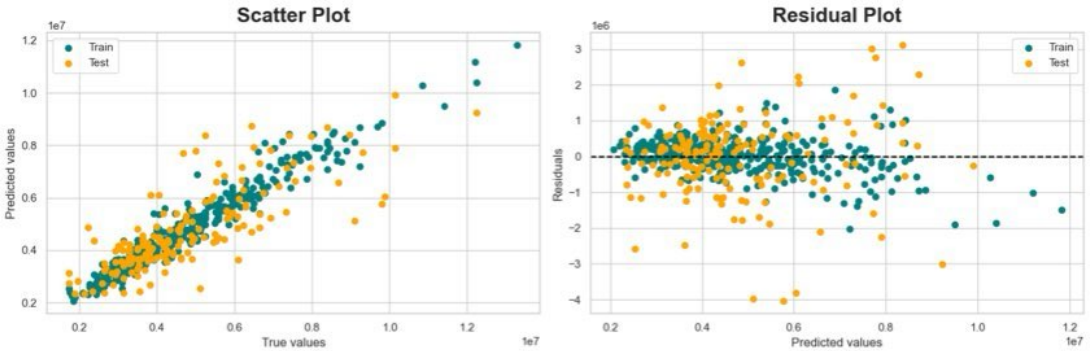



```
Parameters of KNeighborsRegressor(n_neighbors=10) Model:
const    4.763927e+06
x3       5.400685e+05
x1       5.224118e+05
x6       3.958922e+05
x8       3.409127e+05
x4       3.216599e+05
x7       2.245776e+05
x9       2.103494e+05
x5       1.713070e+05
x2       1.437099e+05
dtype: float64
```

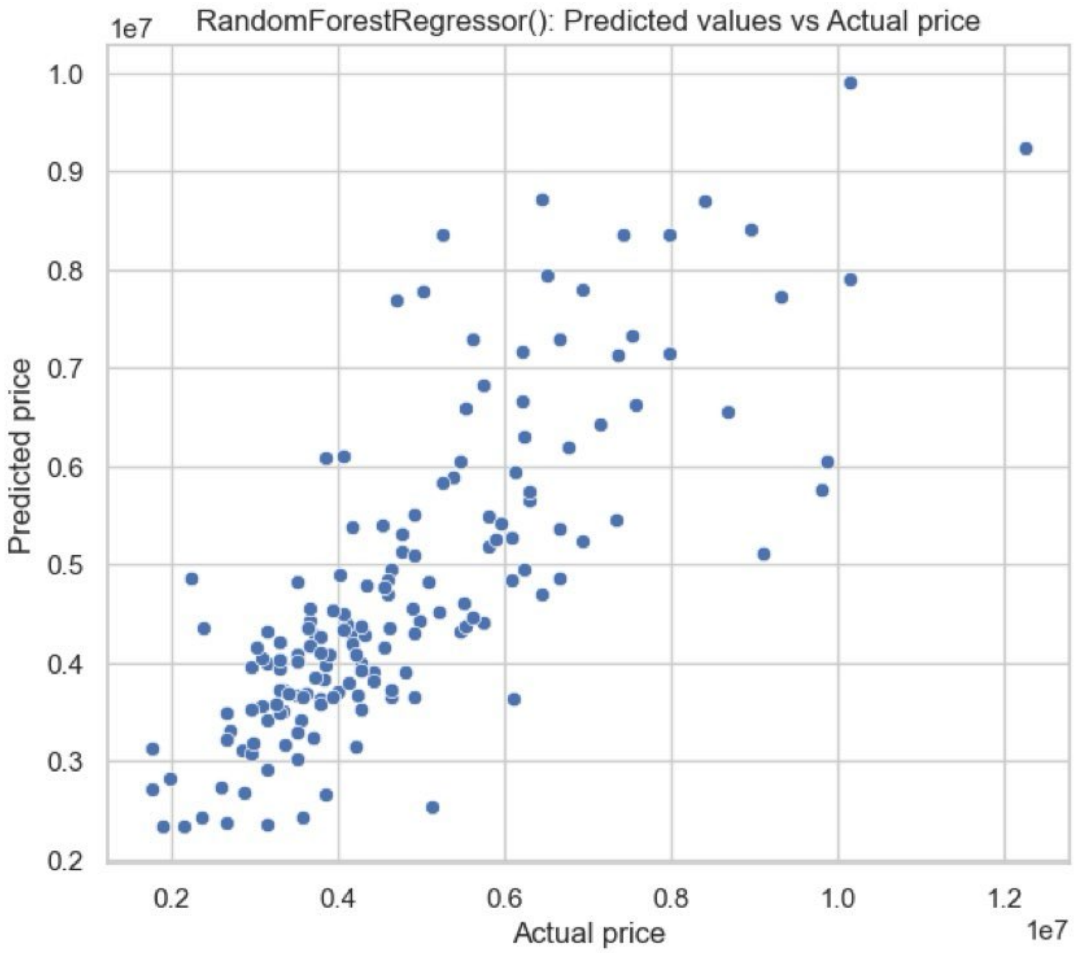


[52]:
```
# Call model_evaluation function with RandomForestRegressor
model_evaluation(RandomForestRegressor())
```

```
RandomForestRegressor() Model Evaluation:
R2 Score: 0.63
MAE: 817874.24
MSE: 1285305313104.46
```




```
Parameters of RandomForestRegressor() Model:
const    4.763927e+06
x3       5.400685e+05
x1       5.224118e+05
x6       3.958922e+05
x8       3.409127e+05
x4       3.216599e+05
x7       2.245776e+05
x9       2.103494e+05
x5       1.713070e+05
x2       1.437099e+05
dtype: float64
```


RandomForestRegressor(): Predicted values vs Actual price
1e7
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Predicted price
0.2
0.4
0.6
0.8
1.0
1.2
1e7
Actual price

[ ]: