Housing Price Predictor

Giovanni Rivera - giovannirivera578@gmail.com
GitHub: github.com/Gio578/HousingPricePredictor

June, 2024

Abstract

After conducting data exploration and performing some light cleaning on the dataset, I engaged in feature engineering to enhance model performance. This included creating features such as the ratio of bathrooms to rooms, location-based features like the distance to the city center and the presence of schools within one kilometer of the dwelling, and incorporating the year of construction as a feature. Following the feature engineering process, I tested several models and ultimately selected Random Forest Regression. I then proceeded with cross-validation, applied Principal Component Analysis (PCA), and concluded with hyper-parameter tuning.

Introduction - Data Exploration

I began with a preliminary analysis of the training dataset, identifying all the various features, data types, and non-null counts.

Following this initial examination, I focused on identifying outliers in the 'price' feature, forming an initial understanding of these outliers that I later explored further through visualization.

To identify the outliers, I utilized two basic visualization tools: scatterplots for each variable and a heatmap (as shown below).

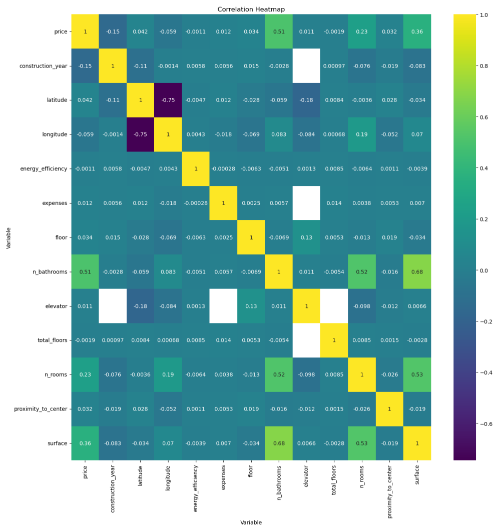

Figure 1: *Heatmap for the identification of outliers*

Afterward, I found it useful to conclude the exploration and visualization phase by clustering the data. I initially identified clusters within the training data and then performed targeted visualizations for individual cities.

After that I went and created a 'missing' dataframe

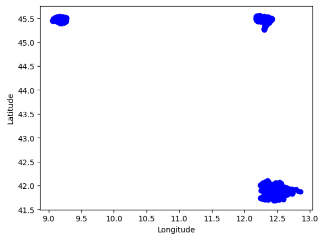

Figure 2: *Scatterplot to identify clusters in the dataframe*

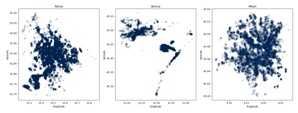

Figure 3: *Scatterplot to identify the distribution in each city*

showcasing the count and percentage of missing values for each variable within the DataFrame 'train'.

In the first part of the code, I calculated the count of missing values for each variable and sorted them in ascending order. In the second part, I calculated the percentage of missing values for each variable and sorted these in ascending order as well. I then performed the same operations on the test dataset.

Upon analyzing the missing values across various features, I observed that the features with the most null data. Additionally, within the dataframe, some data points were found to be unrealistic. For instance, some entries in the 'surface' feature were recorded as 0, the 'construction year' feature had a data point indicating a year greater than 2500, and the 'floor' feature included a building purported to have over 31,000 floors.

After this process, I went on to standardize the missing values through the utilization of median and other methods. I also repeated this process for the 'test' dataset.

Features	Percentage of NaN
price	0.000%
latitude	0.028%
longitude	0.028%
proximity_to_center	0.028%
n_rooms	0.762%
surface	0.957%
conditions	2.654%
floor	4.385%
n_bathrooms	5.217%
construction_year	30.113%
balcony	33.346%
expenses	34.881%
elevator	37.664%
total_floors	39.620%
energy_efficiency	42.976%

Table 1: *Percentage of NaN values for each feature.*

After this, I checked the unique values of the 'Conditions' feature and applied dummy encoding to transform it from a categorical variable into a set of binary variables using the one-hot encoding technique. This approach is particularly useful for enabling algorithms to handle categorical data effectively.

I applied the same process to the test dataset and checked for missing values, finding that 511 entries were missing in the 'Conditions' feature. To ensure a clean dataset, I dropped this feature.

Additionally, I removed all rows in the training dataset where the 'surface' value was 0. Consequently, the initial dataset of 46,312 data points was reduced to a final training dataset of 46,215 data points.

Feature Engineering

Feature engineering is a crucial step in a data science project because it enables the extraction of maximum information value from raw data, thereby enhancing the quality of predictions obtained from machine learning models. Without proper feature preparation, a model may be inefficient and produce unreliable predictions. After testing various new features, the following ones provided the best results and were also practically useful for evaluation by potential buyers. The features I created were:

Bathroom-to-room ratio

This feature calculates the ratio of the number of bathrooms to the number of rooms in the house.

This feature helps determine if there are enough bathrooms compared to the number of rooms, which could be important for potential buyers. Naturally, I also applied this feature to the test dataset.

Location-based features

The purpose of this feature was to use latitude and longitude coordinates to extract location-based attributes such as the distance to popular landmarks, crime rates in the area, school districts, and more. This information can be valuable to potential buyers looking for location-specific features.

To achieve this, I used the latitude and longitude of the city centers and wrote a function to apply the Haversine formula. This formula calculates the distance between two points given their latitudes and longitudes on a sphere (such as the Earth). The Haversine formula is expressed as:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{1}$$

where:

- d is the distance between the two points in kilometers
- r is the radius of the Earth (mean radius = 6,371km)
- $\Delta\phi$ represents the difference between the latitudes of the two points in radians
- $\Delta\lambda$ represents the difference between the longitudes of the two points in radians

However, the issue with the Haversine function is that it requires extremely heavy computations.

Therefore, I opted in creat-ing four new features, focusing on the distance from city center.

- Milan
- Rome
- Venice
- Distance from the center of city

I created a dictionary containing the coordinates of the city centers and filled in the missing latitude and longitude data with their median values. To calculate the distance from the center, I used this formula:

$$d = \sqrt{(\phi_{house} - \phi_{city})^2 + (\lambda_{house} - \lambda_{city})^2} \tag{2}$$

I then wrote a function that creates three new columns in the dataframe. Each column is assigned a value of 1 if the house is closest to the center of one of the three cities (Milan, Rome, Venice) compared to the others, and 0 otherwise. While checking the dataset, I noticed some null values (NaN), which I filled using standard Pandas methods.

Count of schools within a radius:

I defined a function that takes the latitude and longitude of a house as input and created a dataframe called 'schools,' which contains the coordinates of the schools and a radius in kilometers using the 'poi' dataset. The function uses the 'cdist' function from the 'scipy.spatial.distance' module to calculate the distance between the house and all the schools in the dataframe.

Next, I selected only the schools within the specified radius and returned the number of schools found. Finally, I applied the function to all rows of the training dataframe using the apply method and created a new column containing the number of schools found within 1 km of each house.

Finally, I applied the function to all rows of the training dataframe using the apply method, creating a new column that contains the number of schools found within 1 km of each house.

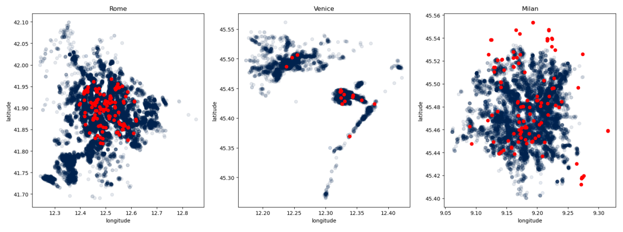

Figure 4: *Scatterplot to visualize the distribution of schools (red), correlated with the disposition of houses (blue).*

House age:

Instead of using the year of construction directly, I calculated the age of each house by subtracting the year of construction from the current year. This is useful because newer houses tend to have higher prices than older ones. To achieve this, I set the current year to 2023 and created a new column that shows the difference between the current year and the construction year.

Model Running

Regarding the workflow for running the model, after testing several options, I found that the Random Forest Regressor provided the best performance. Following this, I proceeded with cross-validation and hyperparameter tuning to further optimize the model.

Random Forest

Random forest regression is an ensemble machine learning algorithm used for regression tasks. It operates by combining multiple decision trees to make predictions. Each tree is trained on randomly selected subsets of the input features and bootstrapped samples of the training data. During prediction, the algorithm averages the predictions from all the individual trees in the forest to arrive at the final prediction. This approach is robust to overfitting, performs well with high-dimensional and noisy data, and can efficiently handle large datasets.

I created the input data X and the output data y from the train2 DataFrame. The input X contains all columns except for the 'price' column, while the output y contains only the 'price' column.

Next, I split the data into a training set and a test set, with 33% of the observations used for testing and the remaining 67% for training. I also set a random seed of 42 for reproducibility.

I trained the model on the training set and made predictions on the test set. Calculating the Mean Squared Error (MSE) between the Random Forest predictions and the actual values in the test set resulted in an MSE of 541,255,415,097.57446.

Cross Validation

I continued by performing cross-validation on a linear regression model to calculate the Mean Squared Error (MSE). I set the number of folds to 5, dividing the dataset into five equal parts for this process.

I initialized the linear regression model and created an empty array to hold the MSE values obtained from each fold. Then, I introduced the KFold method, setting the number of folds to 5, enabling shuffling of the data before dividing it into folds, and setting the random seed to 42 for reproducibility.

For each fold, the following was conducted:

- Extracted the training data
- Extracted the test data
- Split the input and output data into train and test sets

To train and test the linear regression model, it is used to make predictions on the test data, and the MSE is calculated using the MSE function. The MSE value for the current fold is then added to the 'MSE Score' array.

At the end of the loop over all folds, the average MSE valueis calculated by summing the MSE values and dividing by the number of folds.

This average value is then assigned to the variable 'AVG MSE'.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) aims to reduce the dimensionality of the dataset while preserving the most important information. This technique involves finding a new set of orthogonal variables, called principal components, that capture the maximum amount of variance in the dataset.

I used the PCA class from the 'sklearn.decomposition' module and calculated the variance ratios for each component.

The process is based on the Elbow rule, which suggests selecting the number of components at the point where the explained variance starts to level off. This represents the point of diminishing returns, where additional components contribute little to the total explained variance and may even introduce noise into the model.

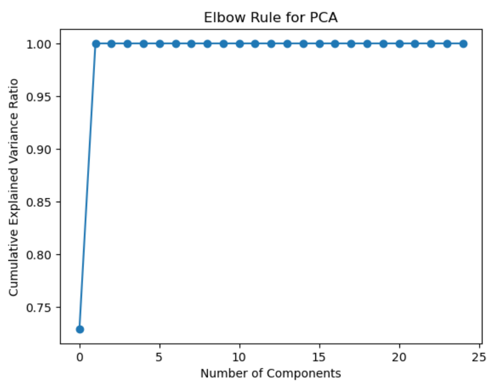

Figure 5: *Visualization of the Principal Component Analysis*

Hyper-Parameters

Following this, I applied a Randomized Search Cross-Validation to select the best hyperparameters for testing in the Random Forest Regression model. This process combines cross-validation with a random selection of hyperparameters, aiming to minimize the error on the training dataset.

Randomized Search is performed over 10 iterations with a cross-validation of 5 folds. Additionally, a random state value is set to ensure reproducibility of the results.

At the end of the Randomized Search, the model is trained on the training dataset using the best-selected hyperparameters and then used to make predictions on the test dataset.

After this, the best parameter was:

- Max Depth: 8
- Max Features: Sqrt
- Min Samples Leaf: 3
- Min Samples Split: 5
- Numbers of Estimators: 64

And finally, I determined the final best score to be 0.37966934926038276

Final Considerations

While Mean Squared Error (MSE) is a common metric used to evaluate the goodness of fit for a regression model, it may not always be the best choice, particularly in contexts such as house price estimation. MSE penalizes forecast errors quadratically, meaning larger errors are weighted much more heavily than smaller ones. In house price estimation, this results in an uneven impact on model accuracy, as errors that lead to oversupply or undersupply can differ significantly in their consequences. Additionally, MSE does not account for the scale of the problem. Therefore, in house price estimation, it is often preferable to use alternative metrics such as Mean Absolute Error (MAE) or Mean Absolute Percentage Error (MAPE), which give proportional weight to forecast errors and consider the scale of the problem.

Additionally, MSE does not account for the scale of the problem. For instance, if house prices in the dataset vary widely, MSE may be disproportionately affected by forecast errors in particularly expensive or low-cost areas.