What Lead to The Rising House Prices in Beijing?

LEE Jihyeon

University of Science and Technology, Department of Social Science

Abstract

This study aims to identify the key determinants of housing prices and develop an optimal price forecasting model. Three datasets are used in this analysis: the transaction dataset of properties in Beijing, the monthly data of CPI of Beijing, and the daily data of the number of residential buildings sold in Beijing. In the explanatory analysis, multiple linear regressions were used to explore the impacts of internal and external features on housing prices. In the predictive analysis, five machine learning models, two feature selection methods, and hyperparameter tuning were introduced to optimize the prediction of housing prices. Results indicate that type of building and the location of the houses have significant impacts on housing prices, and the XGBoost model generated the highest prediction performance scores. The findings of this study can help individuals make informed decisions about buying or selling property and contribute to the development of more effective policy interventions aimed at promoting greater stability and affordability in the housing market.

1. Introduction

1.1. Research Objective

The purpose of this study is to investigate the impacts of various factors on housing prices in Beijing, China, and build an efficient predictive model based on regression analysis. The independent variables in this research can be divided into two categories: (1) internal housing features that constitute the main parts of buildings; (2) external housing features that exist outside the buildings. To achieve these research objectives, multiple methodologies are employed in this study.

The study begins by developing multiple linear regression models to explore the linear relationship between the dependent variable (housing price) and each independent variable. Next, five predictive models are developed, and their results are compared. After the best

model is chosen through R-squared and Root Mean Square Error (RMSE), interaction and polynomial terms, two different feature selection methods, and hyperparameter tuning with 5-fold cross validation are conducted to optimize the model performance.

1.2. Research Significance

As shelter is the basic and crucial need for human survival, the price of real estate has a direct impact on people across all socio-economic groups. Therefore, the findings of this study could have important implications for buyers, sellers, policymakers, and researchers seeking to understand the housing market and its broader economic and social implications.

This research provides a comparative analysis of various factors that influence housing prices, which can help enhance the public's understanding of real estate. Although clients should have a comprehensive understanding of how housing prices are determined and what factors affect their fluctuations, this information is often not readily available for the public. Therefore, this study can aid individuals in comprehending and predicting housing market volatility and identifying key factors to consider when purchasing properties in the future. Additionally, monitoring and managing housing prices is crucial for governments as they are intimately connected to a country's economic well-being. While the Chinese government has implemented several housing policies to stabilize the real estate market, their effectiveness has varied across different cities and times. Therefore, policymakers can benefit from understanding not only the determinants of housing prices but also prediction mechanisms to adopt timely and proactive policies. Lastly, this study provides a valuable overview of the real estate market in China, which can serve as a foundation for future research to conduct more in-depth analysis by investigating a wider range of features and economic indicators.

2. Background

2.1. Housing System and House-Purchase Restrictions

The transformation of the Chinese housing market was facilitated by two major housing reforms: the 1988 reforms and the 1998 reforms. While the housing privatization had been promoted after the housing reform in 1988, the market-based system of housing supply was initiated by the reforms in 1998 which completely terminated the former housing system. Along with the economic reform in 1978, the socioeconomic status in China had transformed

rapidly, and housing had become a crucial component of the Chinese economy. Additionally, the affordability of housing consequent upon the rapid growth of housing prices in China was an urgent challenge faced by both central and local governments. (Man, 2011)

On April 30, 2010, the Chinese government announced the House-purchase restriction (HPR) for the first time in Beijing as an adjustment to the rapid rise in housing prices. Accordingly, many relevant research studies were published regarding the impact of the House-purchase restriction policy on the Housing market in China. However, the results of the study vary depending on the research design and dataset that researchers employed in their analyses. For instance, Li et al. (2020) applied a regression discontinuity design (RDD) with 11,597 second-hand housing data in five countries located near Beijing and found that the HPR policy failed to reduce but rather increased second-hand housing prices. On the contrary, Lu et al. (2021) estimated a structural model to examine the transformations of demand, supply, and market equilibrium in five major Chinese cities after the implementation of the HPR policy, and they concluded that HPR successfully stabilized the housing market in those cities.

2.2. Predictive Model of Housing Price

Yu et al. (2018) conducted quantitative research to build predictive models using deep learning strategies and investigated the determinants of housing prices in Beijing. While Convolution Neural Network (CNN), Long Short-Term Memory (LSTM), and logical regression algorisms were employed to predict housing prices based on characteristic factors, LSTM neural network and Auto-Regressive and Moving Average (ARMA) models were to include time factors in forecasting models. Although models based on time series had higher prediction accuracy, they could only forecast the trends of price, whereas models based on characteristic factors could predict the specific price of houses with different features. However, researchers explored only four characteristic features, including the age of the building, the number of subways, the number of schools, and the location of the house, which might degrade the reliability of the findings.

Similarly, Xu and Zhang (2021) developed forecasting models of housing prices in China using the neural network approach. Researchers collected data on 99 cities from the China Real Estate Index System (CREIS) and compared several neural network models with different algorithms and hyperparameters such as delay, hidden neuron, and data spitting

ratio. The findings revealed that a simple neural network with four delays and three hidden neurons yielded stable prediction results across the 99 cities. This research contributed to deep learning approaches used for price forecasting in the Chinese housing market, but it could be further improved by developing separate models for each city to achieve more accurate predictions of specific housing prices.

3. Data

The “Housing Price in Beijing” dataset (source: uploaded in Kaggle, fetched from Lianjia) which has 318851 transaction records is combined with the monthly data of the Beijing Consumer Price Index (source: National Bureau of Statistics) and daily data of the number of residential buildings sold in Beijing (source: Beijing Municipal Commission of Housing & Urban-Rural Development) based on the date of transactions.

To enhance the quality of data and analysis, the following preprocessing steps were processed:

- Remove attributes that are not used in the analysis
- Remove all records having NA (Completes Analysis)
- Remove the fraud/bad data
- Remove transactions traded in 2008, 2009, 2010, and 2018 (only 0.13 percent of total records)
- Remove transactions of prices below 1000 (set the minimum value)
- Create a ‘totalRoom’ variable by adding the number of rooms including kitchen, bedrooms, living rooms, and bathrooms
- Create an ‘age’ variable by subtracting the ‘constructionTime’ variable from the ‘year’ of the transaction
- Create ‘year’, ‘month’, and 'day' variables from 'transactionTime'
- The number of buildings sold of residential existing houses and houses in advance are summed together to estimate the number of total buildings sold in Beijing

After cleansing the raw data, 10 independent variables and 1 dependent variable remained in

the final dataset. Table 1 provides basic information about attributes that will be used for regression analysis in this research.

Table 1. Dependent and Independent Variables

	Attribute Name	Type	Description
Dependent Variable	price	Continuous	The average price by square
Independent Variables (Internal)	age	Continuous	The age of houses
	totalRoom	Continuous	The total number of rooms including kitchen, bedrooms, living rooms, and bathrooms
	renovationCondition	Nominal	The renovation condition of house: - refined - simple - rough - other
	buildingType	Nominal	The type of building: - bungalow - tower - plate - combination of plate and tower
	elevator	Binary	If the house has any elevator or not: - have (1) - not have (0)
Independent Variables (External)	cpi	Continuous	Monthly Consumer Price Index (CPI) of Beijing
	buildingSold	Continuous	The number of residential buildings sold in Beijing at the day of transaction
	fiverYearsProperty	Continuous	If the owner has the property for less than 5 years - yes (1)

			- no (0)
	year	Continuous	The year of transaction
	district	Nominal	Thirteen districts in Beijing: - Dongcheng - Fengtai - Tongzhou - Daxing - Fangshan - Changping - Chaoyang - Haidian - Shijingshan - Xicheng - Pinggu - Mentougou - Shunyi * Yanqing, Huairou, Miyun disctricts are not included

4. Hypotheses

The main interest of this study is to explore the two kinds of factors that have decisive influences on housing prices in Beijing. The hypotheses of each relationship of interest have been formulated as follows.

4.1. Internal Features

H1: Houses with more rooms have higher prices.

The larger houses with more rooms require more resources and materials to construct, which increases their overall cost. In addition, houses with more rooms may be preferred by buyers who are willing to pay a premium for the added space and functionality, which may lead to higher transaction values.

H2: Houses that have better renovation conditions have higher prices.

Houses in better renovation conditions typically require less maintenance and repair work, which can be costly for residents. Additionally, houses in better condition may be more visually appealing and more likely to attract potential buyers.

H3: Newer houses have a higher value than older houses.

Newer houses are generally built with more modern materials and construction techniques, resulting in improved energy efficiency, structural integrity, and overall functionality. These features can be particularly appealing to buyers who are willing to pay more for the latest designs and amenities.

H4: There is an association between types of buildings and housing prices.

Different types of buildings can have different characteristics that can impact desirability among buyers. In addition, the demand and supply of properties would vary depending on the types of buildings, which may lead to different prices.

H5: Houses with one or more elevators have higher prices

Elevators can enhance the house's accessibility and convenience, particularly for individuals who have difficulty climbing stairs. Therefore, the presence of an elevator could potentially increase the demand for such property in the housing market.

4.2. External Features

H6: If the daily number of residential buildings sold in Beijing increases, the housing prices on that day will increase.

An increase in the number of residential buildings sold may indicate a higher demand in the housing market, which will cause housing prices to rise due to the limited supply of available properties and ultimately lead to a sense of urgency among buyers, who may be willing to pay more for a property to secure it before prices increase further.

H7: The higher CPI of Beijing leads to lower housing prices.

CPI is a measure of inflation, which reflects the general increase in prices of goods and services in an economy. Therefore, an increase in the cost of living may decrease the purchasing power of potential buyers, which can lead to a decrease in demand for housing

and a subsequent decrease in housing prices.

H8: Housing prices increase if the owner has owned the property for less than five years

According to the policy outlined on Liangjia's website, the owners who have owned their properties for over five years and do not own any other real estate in Beijing can waive the value-added tax (VAT) on the transaction when they sell their property. Thus, if the owners have the property for less than five years, selling prices would increase as they have to pay the VAT when selling properties.

H9: Housing prices are positively associated with the year of the transaction.

As the economy grows and inflation occurs over time, the cost of building materials, labor, and other factors related to construction and maintenance may increase, which can lead to higher housing prices.

H10: The houses located in the central area of Beijing have higher prices.

Due to the higher levels of economic activity, cultural significance, and social status in central areas of Beijing, these districts tend to be densely populated with limited available space for new development. As a result, the housing market in central districts may have high demand and low supply, leading to higher prices.

5. Methodologies

5.1. Preprocessing

5.1.1. Z-normalization

Z-normalization is a technique that transforms numerical data to have a mean of zero and a standard deviation of one. This is done by subtracting the mean from each value and dividing by the standard deviation of the data. The transformed data allows the comparison of features at different scales.

$$x_{scaled} = \frac{x - mean}{sd}$$

5.1.2. One-Hot Encoding

One-hot encoding is used to represent categorical features as a set of binary features,

where each possible category is represented by a unique binary feature. This allows for the use of categorical data in machine learning algorithms, which typically require numerical inputs.

5.2. Explanatory Analysis

5.2.1. Multiple Linear Regression

Multiple linear regression is a statistical technique used to model the relationship between a dependent variable and two or more independent variables. In this methodology, the emphasis is on using multiple linear regression to interpret correlation coefficients as a means of explanatory analysis, rather than prediction. Interpreting correlation coefficients involves analyzing the sign and magnitude of the coefficients to determine their relationship to the dependent variable.

The followings are equations of two multiple linear regressions that include internal and external features as independent variables:

$$\hat{y}_{price} = \beta_0 + \beta_1 x_{age} + \beta_2 x_{totalRoom} + \beta_3 x_{renovationCondition} + \beta_4 x_{buildingType} + \beta_5 x_{elevator} + \varepsilon$$

$$\hat{y}_{price} = \beta_0 + \beta_1 x_{cpi} + \beta_2 x_{buildingSold} + \beta_3 x_{fiveYearsProperty} + \beta_4 x_{year} + \beta_5 x_{district} + \varepsilon$$

5.3. Predictive Model

To build predictive models, five machine learning techniques are utilized in this study. The performance of each model is evaluated and compared using two metrics of evaluation.

5.3.1. Linear Regressions

Multiple Linear Regression and Lasso Regression are both linear models. They assume a linear relationship between the input variables and the output variable. Multiple Linear Regression uses ordinary least squares to estimate the coefficients of the linear relationship, while Lasso Regression adds a regularization term to the cost function to shrink some of the coefficients to zero and effectively remove them from the model.

5.3.2. Tree-based *Regressions*

Decision Tree, XGBoost and Random Forest are used to build a tree-like structure to

model the relationships between input and output variables, without making any assumptions about their relationship. Unlike decision trees that build a single tree, XGBoost and Random Forest use boosting and bagging techniques to improve their performance by building an ensemble of decision trees. Ensemble algorisms are methodologies used to address overfitting and underfitting problems of classification. While the Random Forest is the combination of multiple decision trees (weak classifiers) such that each tree votes for the most popular class as its final prediction (Breiman, 2001), Extreme Gradient Boosting (XGBoost) is a machine learning system for tree boosting which implements a gradient boosting framework and has high scalability in all scenarios (Chen & Guestrin, 2016).

5.4. Feature Selection

In this study, two feature selection methods are used to identify and select the most important and relevant features. By discarding the irrelevant or redundant features, these methods can potentially improve the accuracy, efficiency, generalization, and interpretability of the model, while mitigating the risk of the curse of dimensionality.

5.4.1. Principal Component Analysis

Principal Component Analysis (PCA) is used to effectively reduce the dimensionality of a dataset. It works by identifying the underlying patterns in the data and transforming the variables into a smaller set of uncorrelated variables, known as principal components. These principal components are ordered in terms of their variance, with the first component having the highest variance and subsequent components having decreasing variance. By selecting a subset of the principal components, it can effectively reduce the number of variables in the dataset while retaining most of the information.

5.4.2. Feature importance

Feature importance gives a relative score that indicates the contribution of each feature to the performance of a tree-based machine learning model. It is calculated based on how much the inclusion of a specific feature reduces the impurity of the model. In this study, models of top k features with the highest feature importance scores are evaluated and compared using two evaluation metrics. This can identify the optimal number of features that can maximize the performance score.

5.5. Hyperparameter Tuning

To improve the accuracy of the predictive model, 5-fold cross validation is performed to tune the hyperparameters of the model. In this approach, the dataset is divided into 5 disjoint partitions known as folds. One fold is then used for evaluation as the validation set, while the other four are used for training. This process is repeated five times, with each fold being used as the validation set once. The average performance of the model across all five folds is then computed. This process is repeated for each hyperparameter combination, and the combination that yields the best average performance is selected as the optimal hyperparameters. Once they are selected, the model is trained on the entire dataset using these hyperparameters. In the end, the performance of the final model is then tested again on a separate test set to evaluate its generalization capability.

5.6. Evaluation Metrics

5.6.1. R-squared (R2)

R-squared measures the proportion of variance in the dependent variable that is explained by the regression models. R-squared values range from 0 to 1, with a higher R-squared value indicating that the regression model fits well with the data values.

$$R^2 = 1 - \frac{SS_{regression}}{SS_{total}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

5.6.2. Root Mean Square Error (RMSE)

RMSE is interpreted as the standard deviation of the residuals. It measures the average distance between the predicted values and the actual values. RMSE values range from 0 to infinity, with a lower RMSE value indicating that the predicted values are close to the actual values.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

6. Descriptive Statistics

6.1. Distribution

6.1.1. Continuous Variables

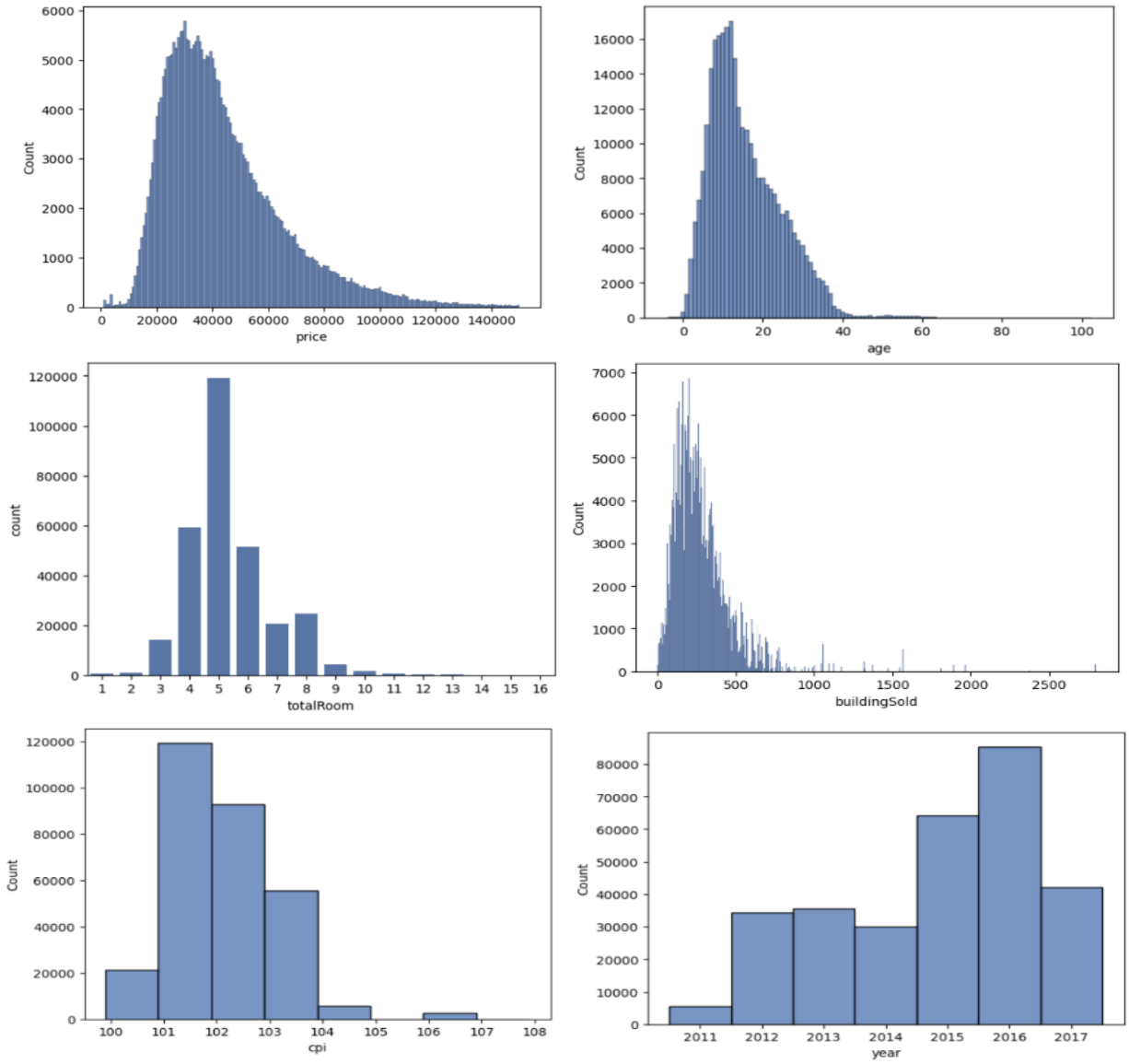


Fig 1. Distribution of Continuous Variables

Table 2. Descriptive Summary of Continuous Variables

	mean	std	min	25%	50%	75%	max
price	43889.193	21761.154	1000	28274	39063	54278	150000
age	15.622	8.866	-4	9	14	21	102
totalRoom	5.359	1.422	1	4	5	6	16
buildingSold	279.513	198.624	0	156	240	350.382	2801
cpi	102.042	0.993	100.4	101.3	101.9	102.7	106.6
year	2014.810	1.652	2011	2013	2015	2016	2017

6.1.2. *Nominal Variables*

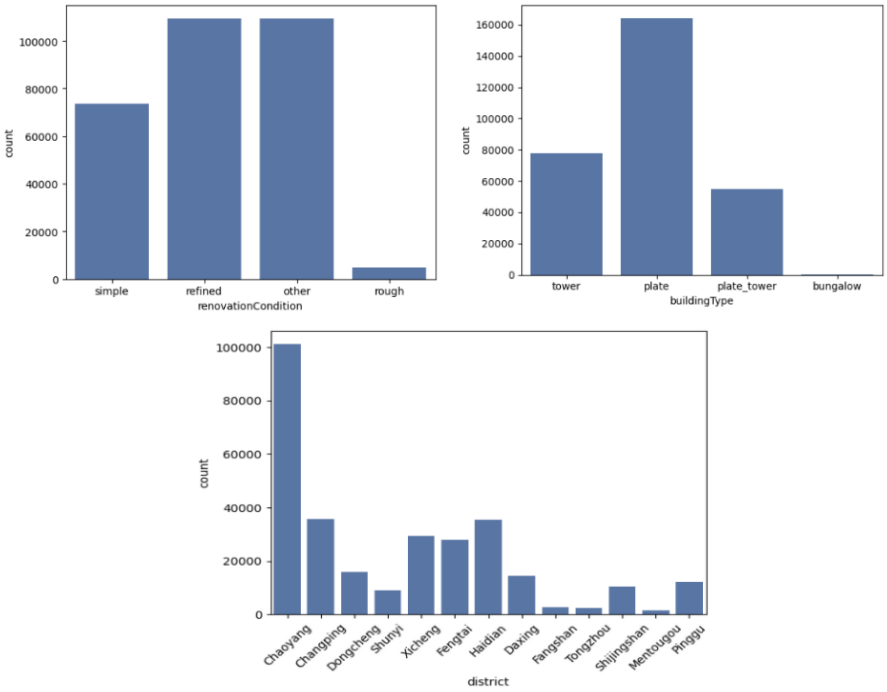


Fig 2. Distribution of Categorical Variables

Table 3. Descriptive Summary of Categorical Variables

	unique	top	freq of top	bottom	freq of bottom
renovationCondition	4	refined	0.368	rough	39063
buildingType	4	plate	0.553	bungalow	0.00032
district	13	Chaoyang	0.340	Mentougou	0.00486

6.1.3. *Binary Variables*

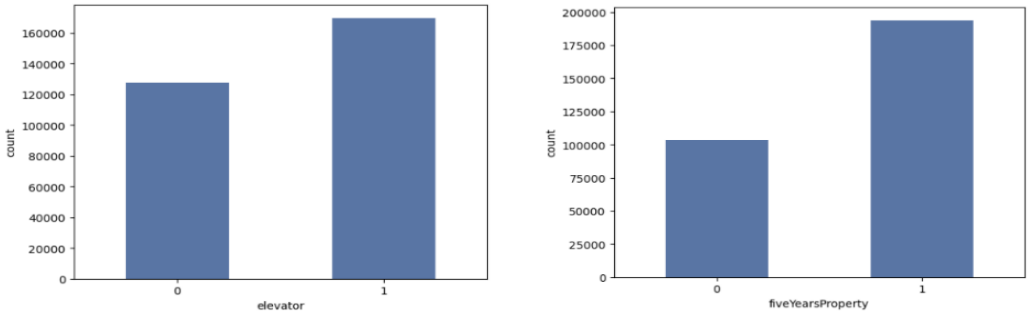


Fig 3. Distribution of Binary Variables

Table 4. Descriptive Summary of Binary Variables

	unique	top	freq of top	bottom	freq of bottom
elevator	2	1	0.571	0	0.429
fiveYearsProperty	2	1	0.652	0	0.348

6.2. *Correlation between the dependent and independent variables*

6.2.1. *Continuous Independent Variables*

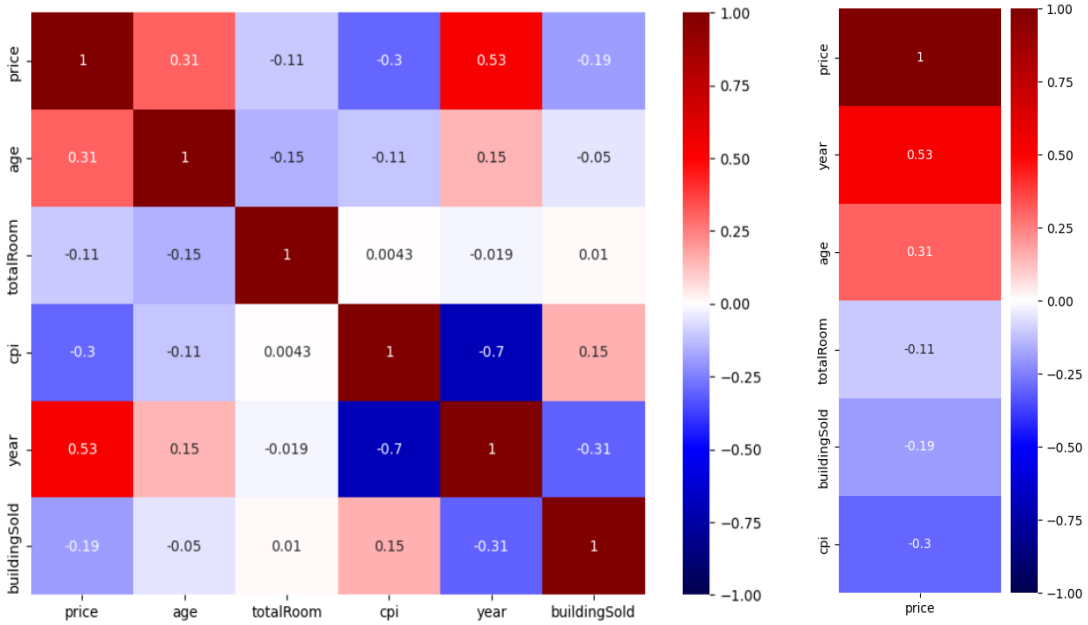


Fig 4. Pearson Correlation Coefficients between Continuous Independent Variables

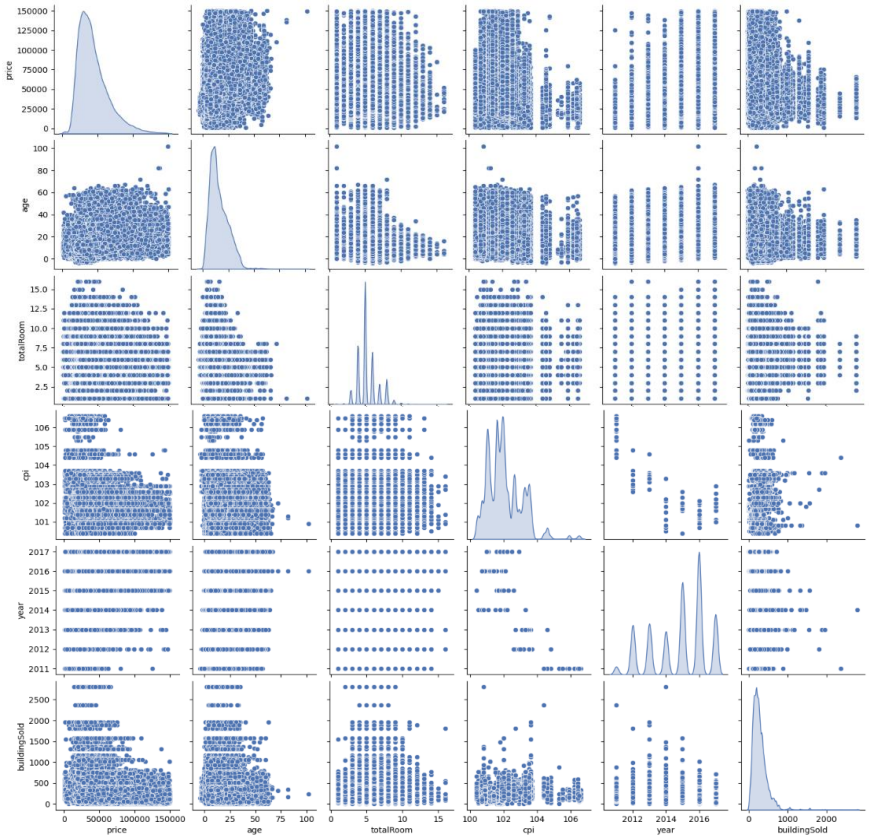


Fig 5. Scatterplots of All Continuous Variables

6.2.2. *Binary Independent Variables*

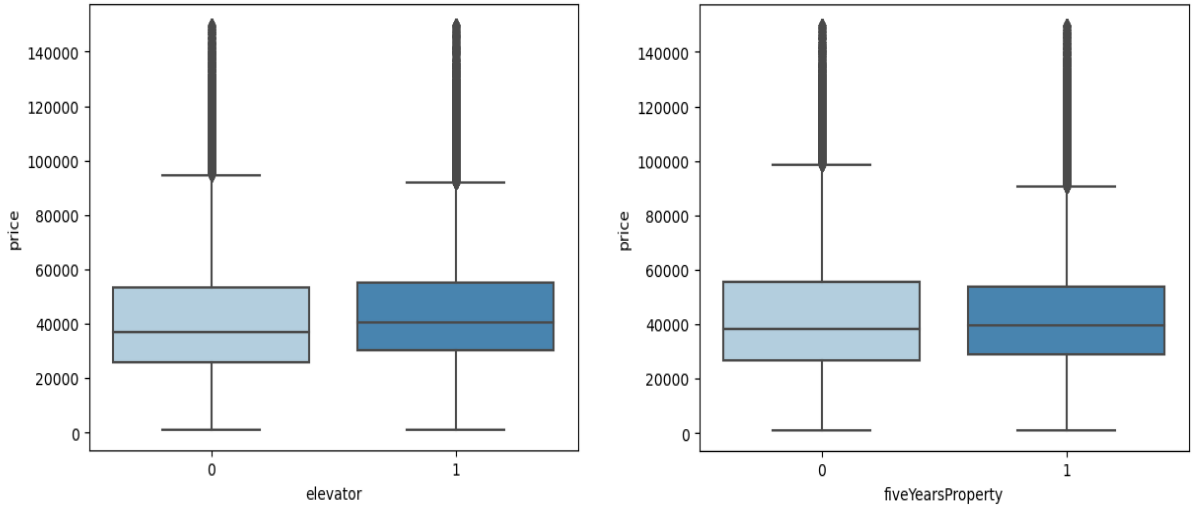


Fig 6. Correlation Between Binary Independent Variables and Housing Price

6.2.3. *Nominal Independent Variables*

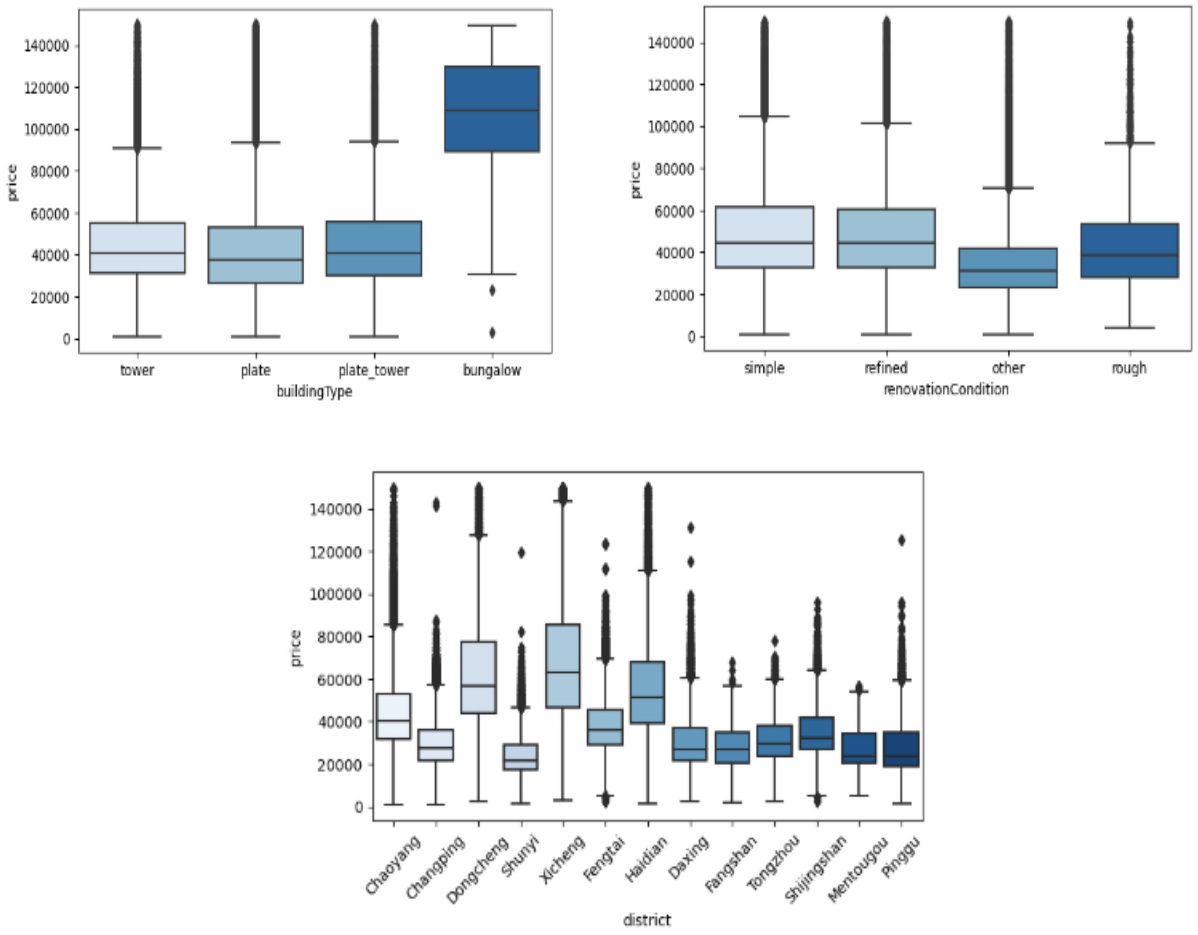


Fig 7. Correlation Between Categorical Independent Variables and Housing Price

6.2.4. *Regional Locations and Housing Prices*

(* longitude and latitude variables only used for descriptive purpose)

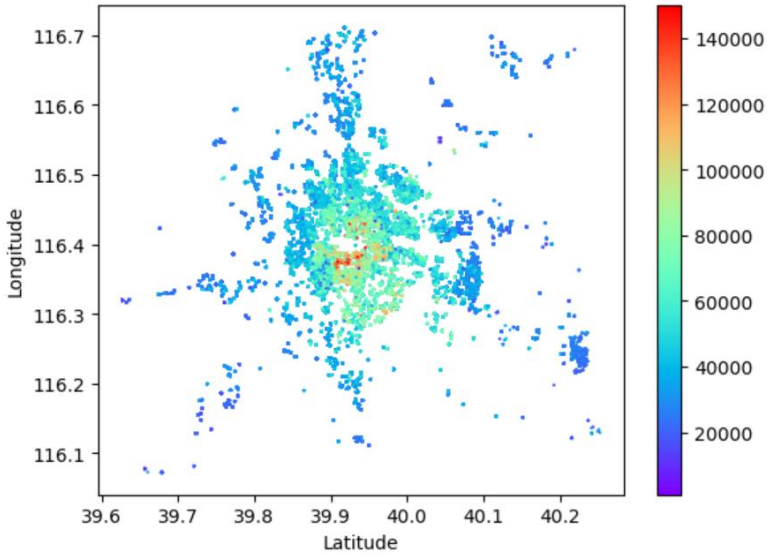


Fig 8. Price Distribution in Beijing

6.3. *Outliers*

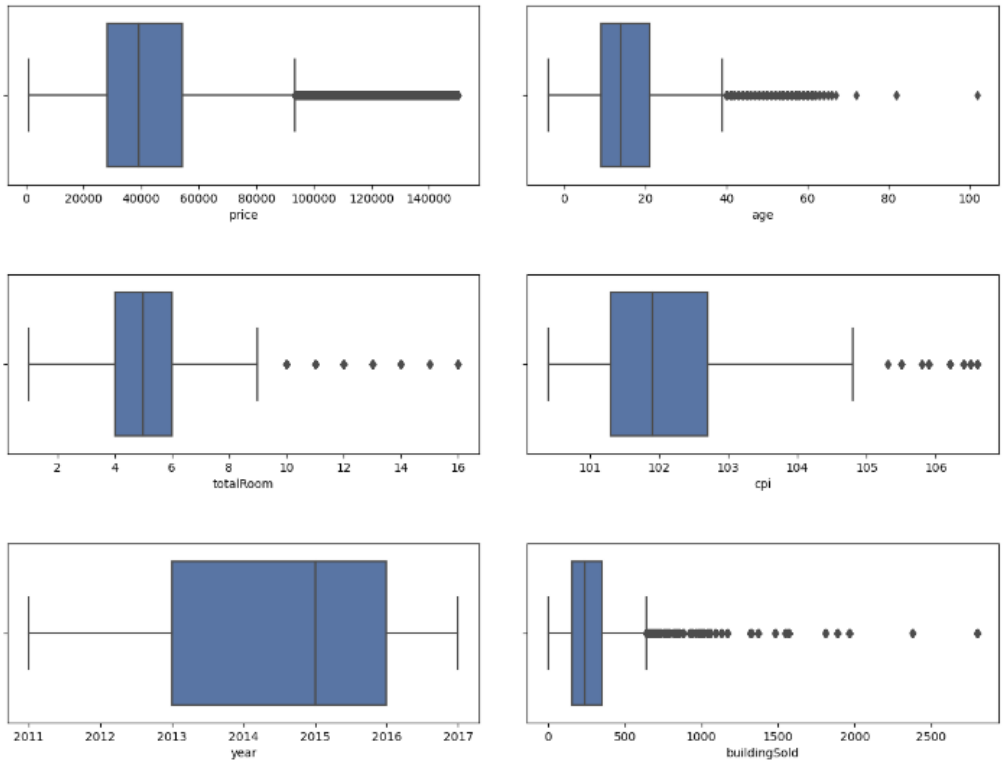


Fig 9. Outliers of Continuous Variables

6.4. Significant Insights

[Fig 1, Fig 9] All continuous variables, except the year variable, have right-skewed distributions with outliers at the large values. However, outliers are not removed in the final dataset for some reasons. First, outliers in housing prices may represent valid observations but not incorrectly measured data. For instance, an outlier may be a luxury property that has desirable features that make it stand out from other houses. Second, the data may be representative of a wide range of properties. Housing price data may cover a wide range of properties, from small apartments to large mansions. In this case, outliers may represent the extreme ends of the distribution, and removing them may not be appropriate.

[Fig 2] The nominal variables in the dataset are imbalanced. Specifically, houses with rough renovation conditions, bungalow building types, and located in Western districts (Fangshan and Mentouhou) or Eastern districts (Tongzhou and Shunyi) are rare compared to other categories. This suggests that these kinds of properties may be less common in the housing market in Beijing.

[Fig 4, 5] The year and age variables in the dataset display a strong positive correlation with housing prices, indicating that newer transactions and older houses tend to have higher prices. Conversely, the cpi variable has a strong negative correlation with housing prices which may suggest that housing prices tend to increase as cpi decreases. However, the strong negative correlation between the cpi and year variables implies that the cpi of Beijing persistently decreased between 2011 and 2017, which may suggest that the impact of cpi or year on housing prices may vary in other periods of time.

[Fig 7, 8] The high housing prices in the dataset are strongly associated with the bungalow building type, as well as the Xicheng, Dongcheng, and Haidian districts. This suggests that properties with the bungalow building type and located in center districts tend to have higher prices in the housing market.

Table 5: Multiple Linear Regressions on Internal, External, and All Features

	Dependent variable: Housing Prices		
	(1)	(2)	(3)
Intercept	1.573***	-0.637***	1.162***
	(0.091)	(0.003)	(0.061)
age	0.385***		0.103***
	(0.002)		(0.001)
totalRoom	-0.058***		-0.011***
	(0.002)		(0.001)
elevator	0.490***		0.205***
	(0.005)		(0.004)
renovationCondition[T.refined]	0.591***		-0.113***
	(0.004)		(0.003)
renovationCondition[T.rough]	0.368***		-0.260***
	(0.013)		(0.009)
renovationCondition[T.simple]	0.492***		-0.217***
	(0.004)		(0.004)
buildingType[T.plate]	-2.176***		-1.712***
	(0.091)		(0.061)
buildingType[T.plate_tower]	-2.125***		-1.773***
	(0.091)		(0.061)
buildingType[T.tower]	-2.300***		-1.897***
	(0.091)		(0.061)
cpi		0.165***	0.145***
		(0.002)	(0.002)
year		0.656***	0.678***
		(0.002)	(0.002)
fiveYearsProperty		-0.100***	-0.094***
		(0.002)	(0.002)
buildingSold		-0.024***	-0.019***
		(0.001)	(0.001)
district[T.Chaoyang]		0.694***	0.631***
		(0.004)	(0.004)
district[T.Daxing]		0.062***	0.052***
		(0.006)	(0.006)
district[T.Dongcheng]		1.526***	1.436***
		(0.006)	(0.006)
district[T.Fangshan]		-0.488***	-0.500***
		(0.012)	(0.012)
district[T.Fengtai]		0.462***	0.432***
		(0.005)	(0.005)
district[T.Haidian]		1.262***	1.202***
		(0.004)	(0.005)
district[T.Mentougou]		-0.145***	-0.129***
		(0.016)	(0.016)
district[T.Pinggu]		0.047***	0.046***
		(0.006)	(0.006)
district[T.Shijingshan]		0.367***	0.293***
		(0.007)	(0.007)
district[T.Shunyi]		-0.131***	-0.149***
		(0.007)	(0.007)
district[T.Tongzhou]		-0.004	-0.009
		(0.013)	(0.013)
district[T.Xicheng]		1.845***	1.754***
		(0.005)	(0.005)
Observations	297,225	297,225	297,225
R^2	0.221	0.643	0.656
Adjusted R^2	0.221	0.643	0.656
Residual Std. Error	0.883(df = 297215)	0.598(df = 297208)	0.586(df = 297199)
F Statistic	9372.745*** (df = 9.0; 297215.0)	33436.556*** (df = 16.0; 297208.0)	22691.533*** (df = 25.0; 297199.0)
Note:			*p<0.1; **p<0.05; ***p<0.01

Table 6: Multiple Linear Regressions on All Features and Interaction & Polynomial Terms

	Dependent variable: Housing Prices	
	(1)	(2)
Intercept	1.162***	0.693***
	(0.061)	(0.056)
age	0.103***	0.050***
	(0.001)	(0.001)
totalRoom	-0.011***	-0.045***
	(0.001)	(0.001)
elevator	0.205***	0.200***
	(0.004)	(0.003)
renovationCondition[T.refined]	-0.113***	-0.102***
	(0.003)	(0.004)
renovationCondition[T.rough]	-0.260***	-0.207***
	(0.009)	(0.010)
renovationCondition[T.simple]	-0.217***	-0.204***
	(0.004)	(0.004)
buildingType[T.plate]	-1.712***	-1.488***
	(0.061)	(0.056)
buildingType[T.plate_tower]	-1.773***	-1.547***
	(0.061)	(0.056)
buildingType[T.tower]	-1.897***	-1.664***
	(0.061)	(0.056)
cpi	0.145***	0.029***
	(0.002)	(0.003)
year	0.678***	0.631***
	(0.002)	(0.013)
fiveYearsProperty	-0.094***	-0.029***
	(0.002)	(0.002)
buildingSold	-0.019***	0.000
	(0.001)	(0.001)
district[T.Chaoyang]	0.631***	0.630***
	(0.004)	(0.004)
district[T.Daxing]	0.052***	0.042***
	(0.006)	(0.005)
district[T.Dongcheng]	1.436***	1.447***
	(0.006)	(0.005)
district[T.Fangshan]	-0.500***	-0.486***
	(0.012)	(0.011)
district[T.Fengtai]	0.432***	0.424***
	(0.005)	(0.004)
district[T.Haidian]	1.202***	1.168***
	(0.005)	(0.004)
district[T.Mentougou]	-0.129***	-0.067***
	(0.016)	(0.015)
district[T.Pinggu]	0.046***	0.017***
	(0.006)	(0.006)
district[T.Shijingshan]	0.293***	0.296***
	(0.007)	(0.006)
district[T.Shunyi]	-0.149***	-0.160***
	(0.007)	(0.006)
district[T.Tongzhou]	-0.009	0.003
	(0.013)	(0.012)
district[T.Xicheng]	1.754***	1.756***
	(0.005)	(0.007)
cpi:district[T.Xicheng]		0.068***
		(0.005)
age:district[T.Haidian]		0.232***
		(0.003)
age:year		0.009***
		(0.001)
age:totalRoom		-0.070***
		(0.001)
year^2		0.111***
		(0.002)
year:buildingSold		-0.028***
		(0.001)
year:district[T.Dongcheng]		0.367***
		(0.005)
year:district[T.Haidian]		0.171***
		(0.003)
year:district[T.Xicheng]		0.555***
		(0.005)
year:district[T.Changping]		-0.119***
		(0.003)
renovationCondition[T.simple]:year		0.042***
		(0.014)
renovationCondition[T.refined]:year		0.090***
		(0.014)
renovationCondition[T.refined]:cpi		-0.011**
		(0.004)
renovationCondition[T.other]:cpi		0.027***
		(0.004)
renovationCondition[T.other]:year		-0.087***
		(0.014)
renovationCondition[T.other]:district[T.Xicheng]		0.086***
		(0.011)
renovationCondition[T.simple]:district[T.Xicheng]		0.042***
		(0.008)
Observations	297,225	297,225
R^2	0.656	0.714
Adjusted R^2	0.656	0.714
Residual Std. Error	0.586(df = 297199)	0.535(df = 297182)
F Statistic	22691.533*** (df = 25.0; 297199.0)	17691.471*** (df = 42.0; 297182.0)
Note:		*p<0.1; **p<0.05; ***p<0.01

7. Results

7.1. Explanatory Analysis

7.1.1. Multiple Linear Regression of Internal, External, and All Featrues

Table 5 shows the results of three multiple regression analyses with standardized coefficients.

The first regression analysis was conducted using five internal features as independent variables. All internal variables were found to be statistically significant at the 0.01 level. However, the results for age (β=0.385) and total rooms (β=-0.385) were inconsistent with the hypothesis, while the presence of at least one elevator was found to increase housing prices by 0.49 standard deviations on average, which supports the initial hypothesis. Additionally, the effect of renovationCondition[T.refined] (β=0.591), renovationCondition[T.rough] (β=0.368), and renovationCondition[T.simple] (β=0.492) on housing prices are all positive compared to renovationCondition[T.other], which served as the reference group. Finally, buildingType was found to have the greatest impact on housing prices in the first model. Compared to buildingType[T.bungalow], buildingType[T.plate] (β=-2.176), buildingType[T.plate_tower] (β=-2.125), and buildingType[T.tower] (β=-2.300) were all negatively related to housing prices, indicating that bungalows were transacted at the highest prices.

In the second regression analysis, external features were examined as independent variables. All variables, with the exception of the district[T.TongZhou], were found to be statistically significant at the 0.01 level. Although the positive association between the year of the transaction and housing prices with a coefficient estimate of 0.656 supports the initial hypothesis, the effects of the CPI (β=0.165), totalBuilding (β=-0.024), and fiveYearsProperty (β=-0.1) were estimated to be inconsistent with the hypotheses. Among the external features, district[T.Xicheng] (β=1.845), district[T.Dongcheng] (β=1.526), and district[T.Haidian] (β=1.262) were found to have the greatest positive impact on housing prices, with district[T.changqing] serving as the reference group. The fact that Xicheng, Dongcheng, and Haidian districts are all located at the center of Beijing emphasizes the importance of regional location as a determinant of housing prices.

The third regression which included all independent variables had the highest R-squared score of 0.656, while the R-squared scores of the first and second regressions were 0.221 and

0.643, respectively. As in the second regression, all variables in the final model, except for district[T.TongZhou], were found to be statistically significant at the 0.01 level. Interestingly, the signs of the coefficients for renovationCondition were found to be negative, opposite to the results of the first regression. Specifically, renovationCondition[T.refined] (β=-0.113), renovationCondition[T.rough] (β=-0.260), and renovationCondition[T.simple] (β=-0.217) were all found to have negative impacts on housing prices, which represents that houses with renovationCondition[T.other] are estimated to have the highest prices in this model.

7.1.2. Multiple Linear Regression with Interaction and Polynomial Terms

Interaction and second-degree polynomials are added to the third multiple linear regression model, which has the highest R-squared score in the previous analysis, in order to find the set of variables that can most effectively explain the variance in the dependent variable. The dataset contains 434 possible interaction and polynomial terms. To determine which terms can increase the model fitting, each one of the 434 terms are combined with the initial regression model, and statistics for each model are recorded. Instead of R-squared, adjusted R squared is used as the evaluation statistic because it penalizes the addition of extraneous variables that do not enhance the fit of the model by adjusting the R-squared in accordance with the number of independent variables and the sample size.

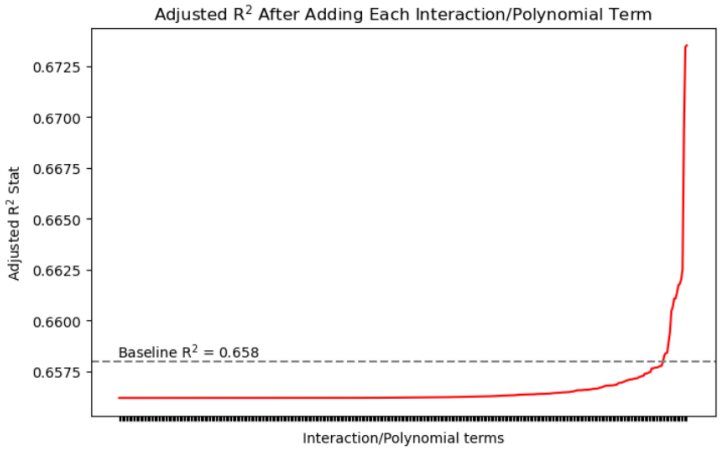


Fig 10. Adjusted R Squared and Interaction/Polynomial terms

Fig 10 illustrates that the adjusted R-squared score decreases rapidly until it reaches 0.658, after which it levels off at lower values. Considering the adjusted R-squared score of the regression model without interaction/polynomial terms is 0.656, one polynomial and 16

interaction terms that yield an adjusted R-squared score above 0.658 are added together to the initial model, which increased the R-squared score of the final regression model to 0.714.

Table 6 presents the results of multiple regression models before and after the inclusion of 17 interaction and polynomial terms. After adding these terms, the most notable change in the initial variables is that the coefficient of totalBuilding changed from -0.019 standard deviation to zero, indicating that it was no longer significantly associated with housing prices. Among the 17 newly added terms, the variables that appeared most frequently are year, district, and renovationCondition, with 10, 8, and 7 occurrences, respectively. Specifically, three interaction terms, year:district[T.Xicheng] (β=0.555), year:district[T.Dongcheng] (β=0.367) and age:district[T.Haidian] (β=0.232), have the largest positive impact on housing prices. These results suggest that the interaction between year, age, and district variables has a larger positive impact on housing prices in the central districts of Beijing.

7.2. Predictive Analysis

7.2.1. Five predictive Models

The performances of linear regression, lasso regression, decision tree, random forest, and XGBoost are compared using RMSE and R2. The dataset is split into two parts: 80% in the training set and 20% in the test set. Since the main purpose of this section is to select the best model that can best predict the housing prices, only default settings are utilized, except the alpha value set as 0.001 in lasso regression.

Table 7. RMSE and R2 scores of Predictive Models

	Metrics	Internal Features	External Features	All Features
Linear Regression	RMSE	0.77929	0.35415	0.34120
	R2	0.21540	0.64344	0.65647
Lasso Regression	RMSE	0.78078	0.35462	0.34277
	R2	0.21389	0.64296	0.65489
Decision Tree	RMSE	0.75867	0.27752	0.37901
	R2	0.23615	0.72058	0.61840
Random Forest	RMSE	0.75334	0.26814	0.20839
	R2	0.24152	0.73003	0.79019
XGBoost	RMSE	0.74313	0.24342	0.19589
	R2	0.25180	0.75492	0.80278

Table 7 shows the RMSE and R square scores of five models. Overall, models of all features have the lowest RMSE and highest R-squared scores, while models of internal features have the highest RMSE and lowest R-squared scores. This is the only exception for the Decision Tree model whose best score comes from external features. Additionally, while the difference in performance between models with external features and models with all features is small, models with internal features performs 2 to 4 times worse in terms of RMSE and R squared compared to the other two models. Overall, since XGBoost with all features performs best, this is selected as the final model that would be further developed.

7.2.2. XGBoost with Interaction and Polynomial Terms

Since there are only a limited number of variables in the dataset, interaction and polynomial terms are added to XGBoost to enhance its prediction performance. In total, 434 interaction and second-degree polynomial terms are included with all initial features in the dataset. As seen in Table 8, RMSE increased by 0.00116 while R-squared decreased by 0.00116.

Table 8. RMSE and R2 of XGBoost After Adding Interaction and Polynomial Terms

	Metrics	**All + Interaction + Polynomial Features**
XGBoost	RMSE	0.19473
	R2	0.80394

7.2.3. Feature selection

After the addition of interaction and polynomial terms, the dimension of the dataset increased to 462. To overcome the issue of the curse of dimensionality, two feature selection methods are conducted on this dataset.

7.2.3.1. PCA

Fig 11 shows the cumulative proportion of explained variance as the number of components increases. Since 123 components can explain 99% of the variance in the dataset, 123 are selected as the number of components in PCA. However, the R-squared score decreased by 0.04245 while RMSE increased by 0.04216 as seen in table 9.

PCA algorithm has some limitations when using it in a predictive modeling context. First, since PCA aims to capture the maximum variance in the data, it treats features with

large variance as important, instead of considering the target variable into account. Thus, it may lose some of the information that is important for predicting the target variable, which can impact the accuracy of the predictions. Additionally, PCA is sensitive to outliers in the data. But the dataset used in this paper has outliers, which can distort the principal components and lead to incorrect predictions.

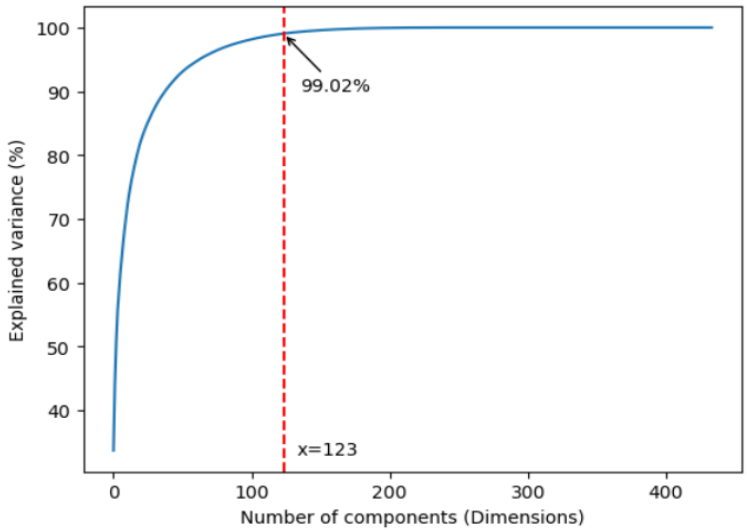


Fig 11. Cumulative Scree Plot of PCA

Table 9. RMSE and R2 of XGBoost with 123 Principle Components

	Metrics	**All + Interaction + Polynomial Features**
XGBoost	RMSE	0.23689
	R2	0.76149

7.2.3.2. Feature Importance

The feature importance scores in XGBoost are calculated based on how often each feature is used to split the data across all trees in the ensemble, and weighted by how much each split reduces the error in the model. Features with higher importance scores are considered more important for predicting the target variable, and prioritizing these variables can potentially improve the model's accuracy. Fig 12 displays the top 30 variables with the highest feature importance on the XGBoost model.

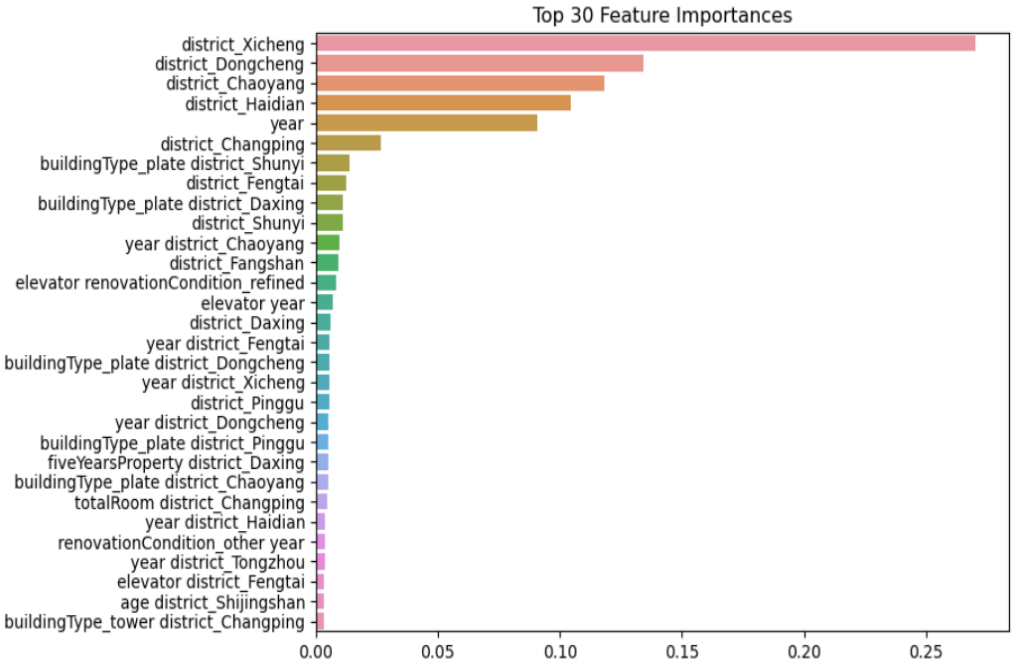


Fig 12. Top 30 with Highest Feature Importance

To identify the optimal number of variables, the R-squared and RMSE scores are calculated for each model by increasing the number of features by five in each iteration. As a result, the model with the top 95 variables, selected based on their feature importance scores, achieved the highest R-squared and lowest RMSE scores, as illustrated in Fig 13. Table 10 shows a decrease in RMSE by 0.00256 and an increase in R-squared by 0.00257 compared to the previous model without feature selection, indicating improved predictive performance. Therefore, these 95 features are adopted for use in the final model.

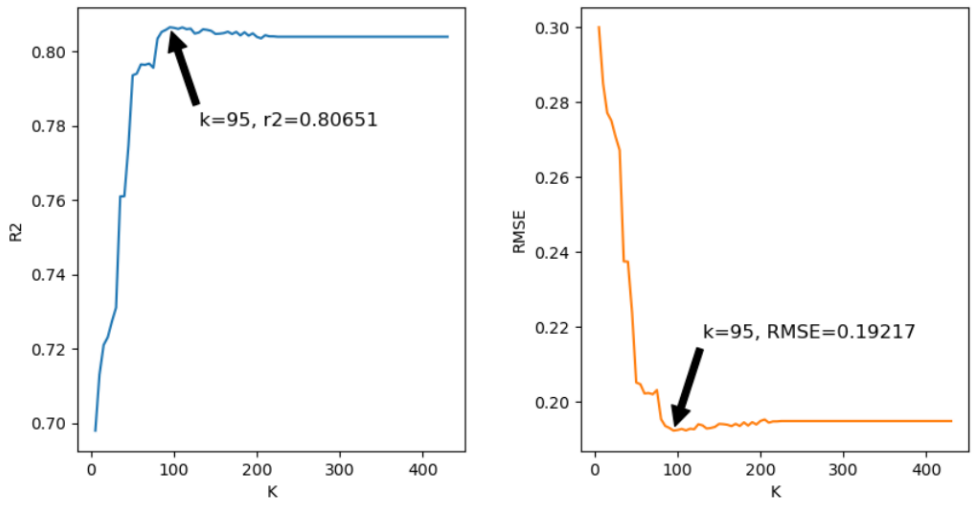


Fig 13. R2 and RMSE of K Features with Highest Feature Importance

Table 10. RMSE and R2 Top 95 Variables with Highest Feature Importance

	Metrics	All + Interaction + Polynomial Features
XGBoost	RMSE	0.19217
	R2	0. 80651

7.2.4. Hyperparameter Tuning

The XGBoost model, with 95 selected features, was optimized using the GridSearchCV function from the scikit-learn library in Python. The optimization process focused on three key hyperparameters: 'n_estimators', 'max_depth', and 'learning_rate'. The 'n_estimators' determines the number of decision trees to be created during training, while 'max_depth' specifies the maximum depth of each decision tree. Although increasing these two parameters can lead to better model performance, it also increases the risk of overfitting. 'learning_rate' controls the step size at each iteration while updating the weights of the decision trees. A smaller learning rate can improve the model's generalization performance by preventing overfitting, but it may also slow down the training process and cause the model to converge to a local minimum. Therefore, finding the optimal values for these hyperparameters is essential for achieving a good balance between the model's performance and overfitting risk.

The following parameter settings are applied in 5-fold cross validation:

- ‘n_estimators’: [100, 200, 300, 400]
- ‘max_depth’: [6, 7, 9]
- ‘learning_rate’ :[0.0001, 0.001, 0.01, 0.1]

Consequently, the optimal hyperparameters combination for XGBoost is identified as follows: 'learning_rate' of 0.1, 'max_depth' of 9, and 'n_estimators' of 400. Applying this combination increases R-squared by 0.00711 and decreases RMSE by 0.00717 as evident by table11.

Table 11. RMSE and R2 After Hyperparameter Tuning

	Metrics	All + Interaction + Polynomial Features
XGBoost	RMSE	0.185067
	R2	0.81368

7.2.5. Predictive Performance of Final Model

In Fig 14, the points represent the observed actual values in the test set and the predicted values generated by the final XGBoost model, while the fitted regression line represents the relationship between these two values. The tight clustering of the points around the regression line and the proximity of the regression line to the diagonal line suggest that the model is producing highly accurate predictions.

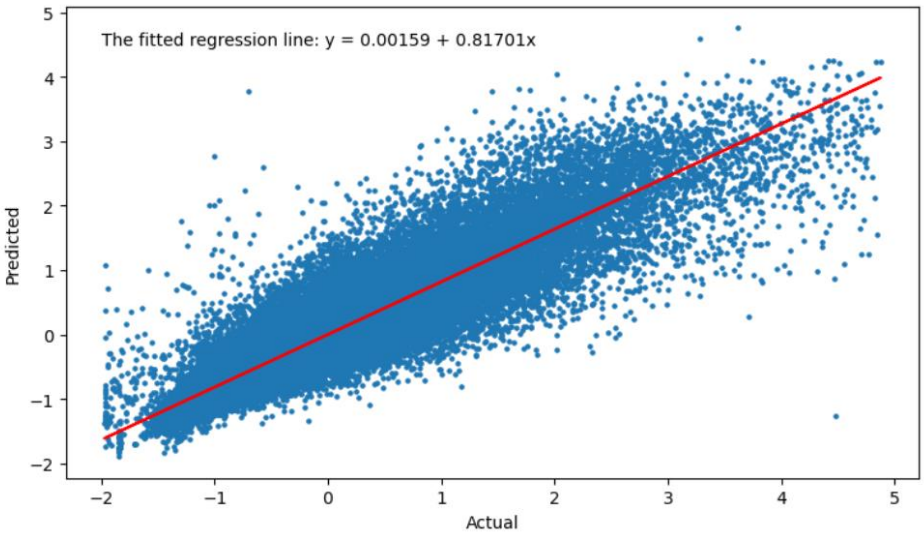


Fig 14. Scatter Plot of Predicted Results and Actual Test Set with the Line of Best Fit

Fig 15 displays the performance of the XGBoost model across the first 300 observations of the test set. The line plot shows the actual and predicted values, which follow a similar pattern and are closely aligned. This close alignment suggests that the model is generating highly accurate predictions.

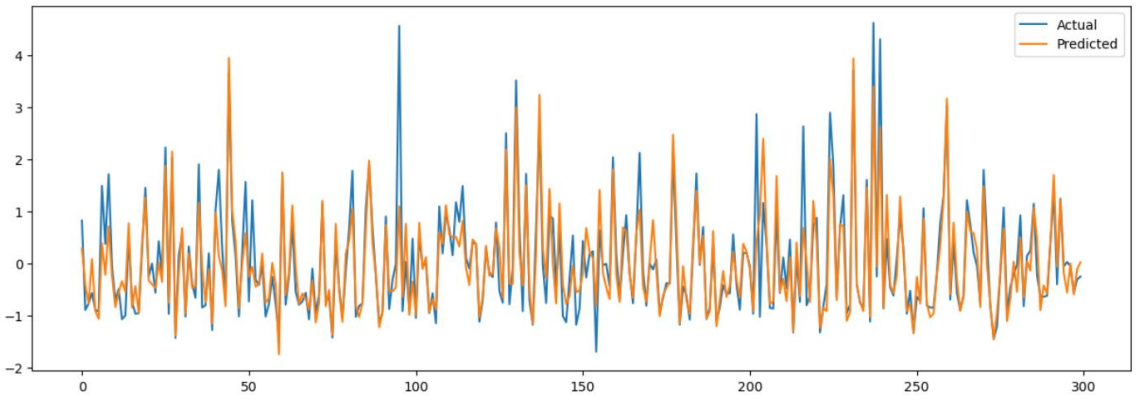


Fig 15. A line plot of Predicted Results and Actual Test Set on the first 300 observations

8. Conclusions & Discussion

The study examined the impacts of internal and external features on housing prices in Beijing through multiple linear regression models and employed five different machine learning methods, including linear regression, lasso regression, decision tree, random forest, and XGBoost, to develop an optimized forecasting model for housing prices.

There are five main findings. First, external independent variables were found to explain approximately three times more variance in housing prices compared to internal independent variables. In other words, the impact of external features on housing prices was found to be significantly greater than that of internal features in this study. Second, among internal features, the type of building is a critical determinant, with bungalows having the most substantial impact on housing prices. Third, the location of the houses has a crucial role in determining their value, with properties located in the central areas of Beijing having higher prices. Fourth, contrary to the initial hypotheses, the study found that ownership of property for less than five years, the number of rooms, and the number of buildings sold on the day of the transaction had a negative impact while the Consumer Price Index of Beijing and the age of the building had a positive impact on housing prices. Last, among the five predictive models analyzed, the XGBoost with tuned hyperparameters and top 95 variables based on feature importance generates the highest R2 and RSME.

However, it is important to acknowledge that the present study has certain limitations. Firstly, the analysis is based on the dataset that only covers transactions that occurred between 2011 and 2017, which may limit the generalizability of the findings to the current housing market. Secondly, the data of CPI and the number of buildings sold in Beijing are

merged with the transaction dataset in accordance with the transaction date, which may not account for any time delay effect. Lastly, housing prices could be influenced by various factors beyond those considered in this study. Therefore, future research can benefit from incorporating additional features to obtain a more comprehensive understanding of the determinants of housing prices and enhance the accuracy of predictive models.

References

Breiman, L. (2001). Random Forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/a:1010933404324

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Li, Y., Zhu, D., Zhao, J., Zheng, X., & Zhang, L. (2020). Effect of the housing purchase restriction policy on the real estate market: Evidence from a typical suburb of Beijing, China. *Land Use Policy*, *94*, 104528. https://doi.org/10.1016/j.landusepol.2020.104528

Lu, Z., Zhang, S., & Hong, J. (2021). Examining the impact of home purchase restrictions on China's housing market. *China Economic Review*, *67*, 101620. https://doi.org/10.1016/j.chieco.2021.101620

Man, J. Y. (2011). *China's housing reform and outcomes*. Lincoln Institute of Land Policy.

Xu, X., & Zhang, Y. (2021). House price forecasting with Neural Networks. *Intelligent Systems with Applications*, *12*, 200052. https://doi.org/10.1016/j.iswa.2021.200052

Yu, L., Jiao, C., Xin, H., Wang, Y., & Wang, K. (2018). Prediction on housing price based on deep learning. *International Journal of Computer and Information Engineering*, 12(2), 90-99.

Appendices

Appendix A: Journal

WEEK	Tasks Achieved	Contact with Advisor
1	Review my previous proposal	
2	More literature review and re-check research validity	yes
3	Improve the proposal	yes
4	Find additional data based on improved proposal	
5	Data-preprocessing	
6	Start the coding part (descriptive analysis)	
7	Continue the coding part (running models)	yes
8	Prepare Mid-term progress Report	yes
9	Finish the coding part	
10	Prepare the final presentation	
11	Write the final paper	
12	Write the final paper	

Appendix B: code

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as mno
import scipy.stats as stats

from sklearn.preprocessing import StandardScaler, OneHotEncoder, PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import PCA
import xgboost as xgb

import statsmodels.api as sm
import statsmodels.formula.api as smf
from stargazer.stargazer import Stargazer

from pylab import *
from scipy.stats import pointbiserialr
```

In [20]:
```
data = pd.read_csv("beijing.csv",  encoding="utf-8")
```

Additional Datasets

In [21]:
```
building_exist= pd.read_csv("new/Building_Existing.csv")
building_advance = pd.read_csv("new/Building_Advance.csv")
cpi = pd.read_csv("new/cpi.csv")
```

In [22]:
```
building_exist = building_exist.set_index('date')

new_dates = pd.date_range("2011-01-01", "2017-12-31")

building_exist.index = pd.DatetimeIndex(building_exist.index)
building_exist = building_exist.reindex(new_dates, fill_value=np.NaN)

building_exist = building_exist.reset_index()

building_exist['year'] = building_exist['index'].dt.year
building_exist['month'] = building_exist['index'].dt.month
building_exist['day'] = building_exist['index'].dt.day
```

In [23]:
```
building_advance = building_advance.set_index('date')

new_dates = pd.date_range("2011-01-01", "2017-12-31")

building_advance.index = pd.DatetimeIndex(building_advance.index)
building_advance = building_advance.reindex(new_dates, fill_value=np.NaN)

building_advance = building_advance.reset_index()

building_advance['year'] = building_advance['index'].dt.year
```

```
building_advance['month'] = building_advance['index'].dt.month
building_advance['day'] = building_advance['index'].dt.day
```

In [24]:
```
cpi['date'] = pd.DatetimeIndex(cpi['date'])

cpi['year'] = cpi['date'].dt.year
cpi['month'] = cpi['date'].dt.month

cpi.drop(columns=['date'], inplace=True)
```

In [25]:
```
# check missing values

print(sum(building_exist['value'].isnull()))
print(sum(building_advance['value'].isnull()))
```

```
442
441
```

In [26]:
```
building_exist = building_exist.set_index('index')
building_advance = building_advance.set_index('index')

building_exist= building_exist.interpolate(method='spline', limit_direction='backwar
building_advance= building_advance.interpolate(method='spline', limit_direction='bac
```

In [27]:
```
# sum existing + advance

building = pd.merge(building_exist, building_advance, on=['year', 'month', 'day'], h
building['buildingSold'] = building['value_x'] + building['value_y']
building.drop(columns=['value_x', 'value_y'], inplace=True)
```

Data Pre-Processing

In [28]:
```
# correct the wrong column names
data.rename(columns = {'livingRoom':'bedRoom', 'drawingRoom':'livingRoom'}, inplace
```

In [29]:
```
# extract 'year', 'month', and 'day' from tradeTime
data['year'] = pd.DatetimeIndex(data['tradeTime']).year
data['month'] = pd.DatetimeIndex(data['tradeTime']).month
data['day'] = pd.DatetimeIndex(data['tradeTime']).day
```

In [30]:
```
# convert 未知 to nan
data['constructionTime'] = data['constructionTime'].replace('未知', np.nan)
```

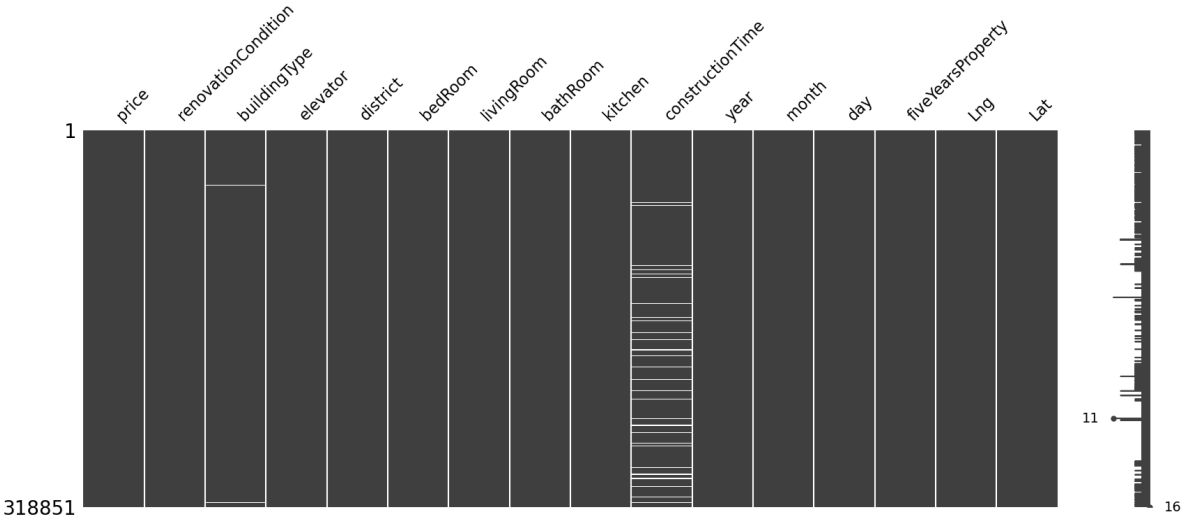
In [31]:
```
data_selected = data[['price',
                      'renovationCondition', 'buildingType', 'elevator', 'district',
                      'bedRoom', 'livingRoom', 'bathRoom', 'kitchen',
                      'constructionTime',
                      'year', 'month', 'day','fiveYearsProperty',
                      'Lng', 'Lat']].copy( )
data_selected.shape
```

Out[31]:
```
(318851, 16)
```

In [32]:
```
# Handling Missing Values

mno.matrix(data_selected, figsize = (20, 7))
missing_values_count = data_selected.isnull().sum().sort_values(ascending=False)
missing_values_prop = missing_values_count / data_selected.shape[0]
```

```
missing_values = pd.concat([missing_values_count, missing_values_prop], axis=1, keys
print(missing_values)
```

```
                     Count  Proportion
constructionTime     19283    0.060477
buildingType          2021    0.006338
elevator                32    0.000100
bedRoom                 32    0.000100
livingRoom              32    0.000100
fiveYearsProperty       32    0.000100
bathRoom                 2    0.000006
price                    0    0.000000
renovationCondition      0    0.000000
district                 0    0.000000
kitchen                  0    0.000000
year                     0    0.000000
month                    0    0.000000
day                      0    0.000000
Lng                      0    0.000000
Lat                      0    0.000000
```



In [33]:
```
data_selected.dropna(inplace=True)
data_selected.shape
```

Out[33]:
```
(297990, 16)
```

In [34]:
```
# create new feature: 'age', 'totalRoom'
data_selected['age'] = data_selected['year'].astype(int) - data_selected['constructi
data_selected['totalRoom'] = data_selected['bedRoom'] + data_selected['livingRoom']
```

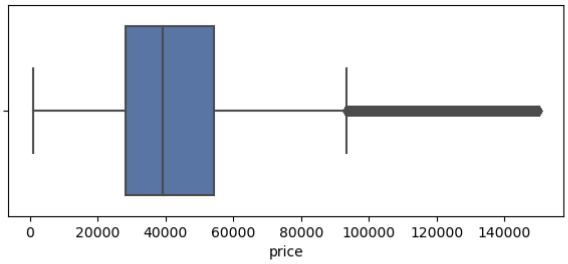
In [35]:
```
data_selected['renovationCondition'] = data_selected['renovationCondition'].map({1:

data_selected['buildingType'] = data_selected['buildingType'].map({1: 'tower', 2: 'b

data_selected['district'] = data_selected['district'].map({1: 'Dongcheng', 2: 'Fengt
                                                           6: 'Changping', 7: 'Chaoy
                                                           11: 'Pinggu', 12: 'Mentou
```

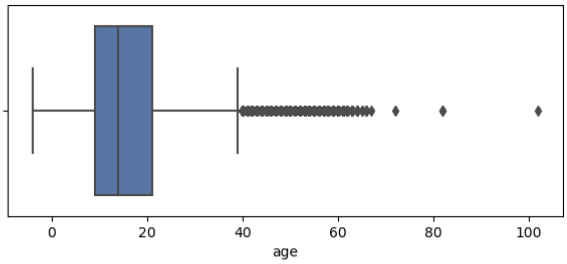
In [36]:
```
# set minimun and maximun values
data_selected = data_selected[(data_selected['year'] >= 2011) & (data_selected['year
                              & (data_selected['price'] >= 1000)]
```

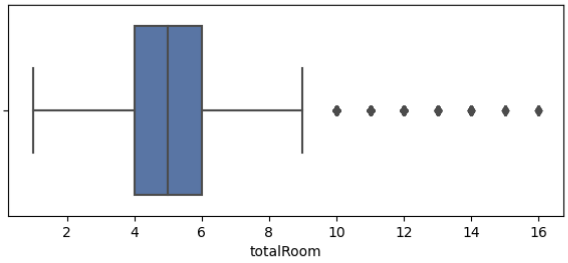
In [ ]:
```
# merge three datasets

merged = pd.merge(data_selected, building, on=['year', 'month', 'day'], how='left')
merged = pd.merge(merged, cpi, on=['year', 'month'], how='left')
```

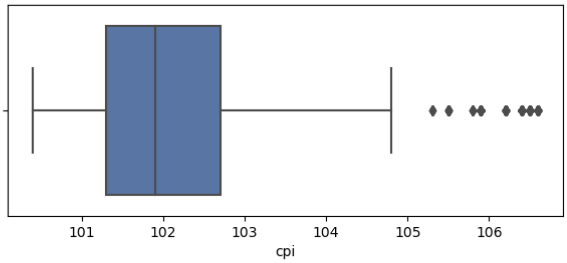
In [38]:
```
beijing = merged[['price', 'age', 'totalRoom', 'renovationCondition', 'buildingType'
                  'fiveYearsProperty', 'cpi', 'year', 'buildingSold', 'Lng', 'Lat']]
```

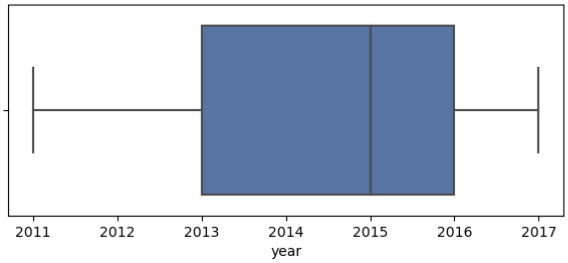
In [39]:
```
# check outliers in numberical values

continuous = beijing[['price', 'age', 'totalRoom', 'cpi', 'year', 'buildingSold']]
c_list=list(continuous.columns)
plt.figure(figsize=[15,15])
plt.subplots_adjust(wspace=0.1, hspace =0.5)

for i in range(len(c_list)):
    plt.subplot(4,2,i+1)
    sns.boxplot(x=beijing[c_list[i]], color="#4c72b0")
```

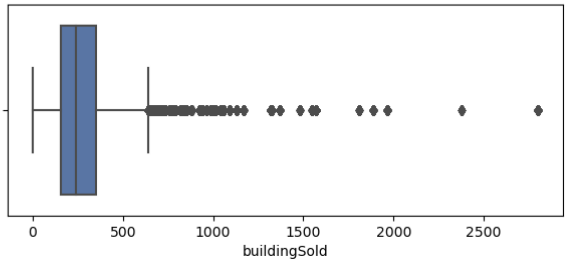












In [40]:
```
Q1 = beijing[['price', 'age', 'totalRoom', 'cpi', 'year', 'buildingSold']].quantile(
Q3 = beijing[['price', 'age', 'totalRoom', 'cpi', 'year', 'buildingSold']].quantile(
IQR = Q3 - Q1

lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)
```

In [42]:
```
outlier = ((beijing[['price', 'age', 'totalRoom', 'cpi', 'year', 'buildingSold']] <
outlier.sum()
```

Out[42]:
```
price            10812
age               2329
totalRoom         2527
cpi               2562
year                 0
buildingSold     12253
dtype: int64
```

Descriptive Statistics

In [23]:
```
# descriptive summary of continous variables

beijing.describe()
```

Out[23]:

	price	age	totalRoom	elevator	fiveYearsProperty	
count	297225.000000	297225.000000	297225.000000	297225.000000	297225.00000	297225.0000
mean	43889.193116	15.621859	5.358812	0.570662	0.65224	102.0422
std	21761.154135	8.866045	1.422028	0.494983	0.47626	0.9932
min	1000.000000	-4.000000	1.000000	0.000000	0.00000	100.4000
25%	28274.000000	9.000000	4.000000	0.000000	0.00000	101.3000
50%	39063.000000	14.000000	5.000000	1.000000	1.00000	101.9000
75%	54278.000000	21.000000	6.000000	1.000000	1.00000	102.7000
max	150000.000000	102.000000	16.000000	1.000000	1.00000	106.6000

In [25]:
```
# descriptive summary of categorical variables

summary = pd.DataFrame(columns=['Variable', 'Count', 'Unique', 'Most frequent', 'Fre

for col in beijing.columns:
    if beijing[col].dtype == 'object' or beijing[col].nunique() == 2:
        # count the non-missing values and number of unique categories
        count = beijing[col].count()
        unique = beijing[col].nunique()

        # get the frequency of each category
        freq = beijing[col].value_counts(normalize=True)

        # get the name and frequency of the most frequent and least frequent categor
        most_freq_name = freq.index[0]
        most_freq_freq = freq.values[0]
        least_freq_name = freq.index[-1]
        least_freq_freq = freq.values[-1]

        # add the summary statistics to the summary dataframe
        row = {
            'Variable': col,
            'Count': count,
            'Unique': unique,
            'Most frequent': most_freq_name,
            'Frequency of most frequent': most_freq_freq,
            'Least frequent': least_freq_name,
            'Frequency of least frequent': least_freq_freq
        }
```

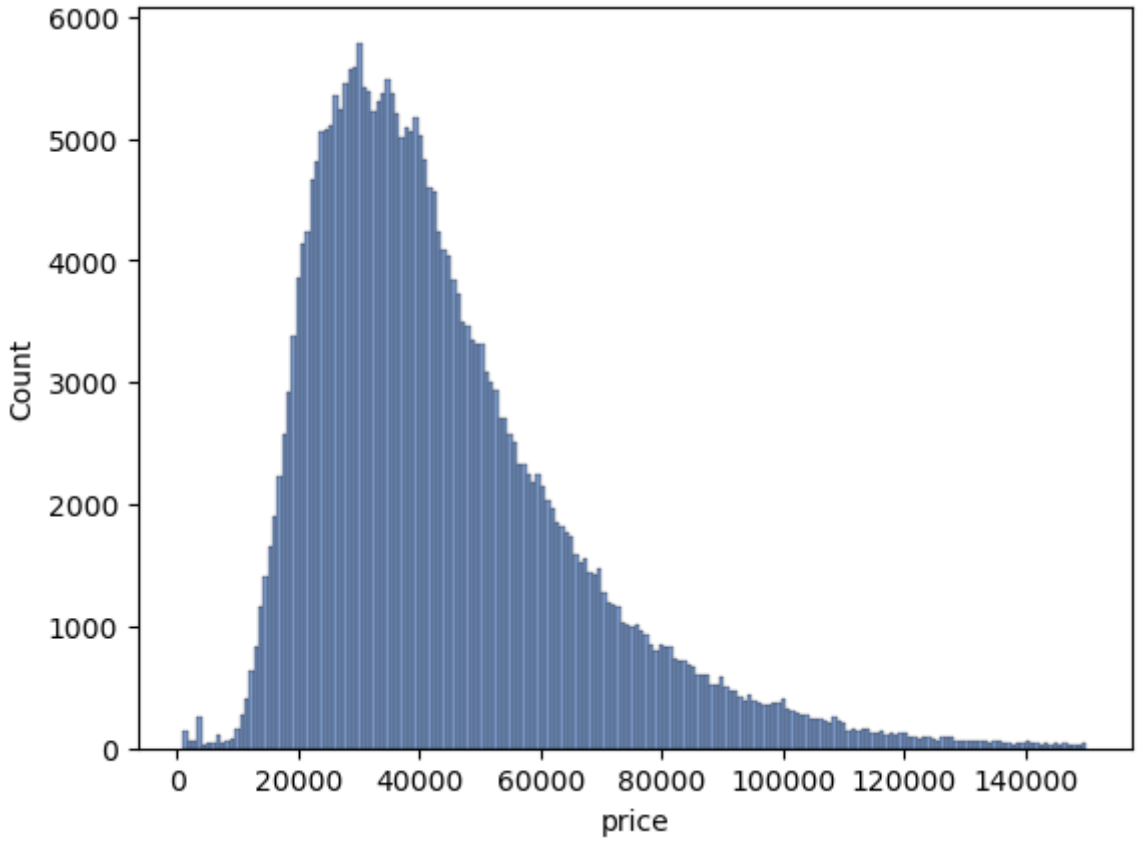
```
        summary = pd.concat([summary, pd.DataFrame(row, index=[0])], ignore_index=T
summary
```

Out[25]:

	Variable	Count	Unique	Most frequent	Frequency of most frequent	Least frequent	Frequency of least frequent
0	renovationCondition	297225	4	refined	0.368266	rough	0.016425
1	buildingType	297225	4	plate	0.553073	bungalow	0.000316
2	elevator	297225	2	1.0	0.570662	0.0	0.429338
3	district	297225	13	Chaoyang	0.340052	Mentougou	0.004855
4	fiveYearsProperty	297225	2	1.0	0.652240	0.0	0.347760

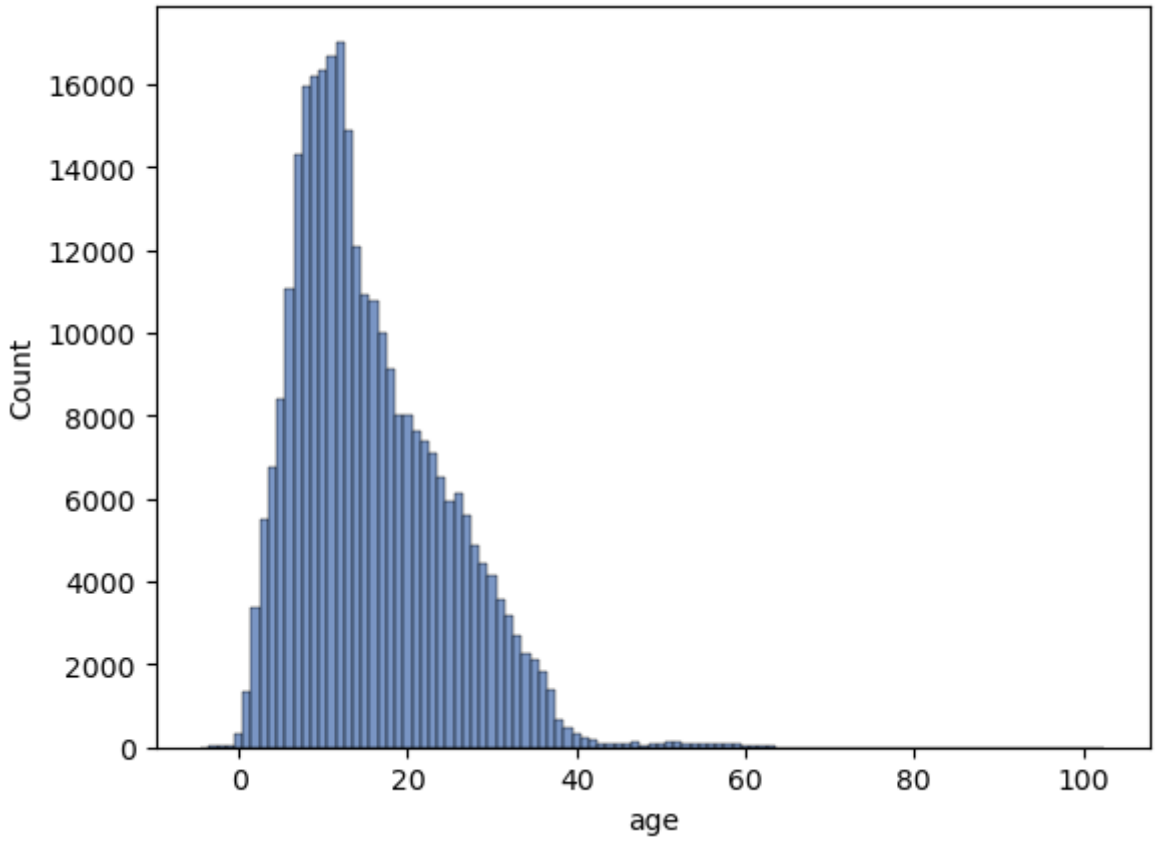
In [26]:
```
sns.histplot(beijing['price'], color="#4c72b0")
```

Out[26]:
```
<AxesSubplot:xlabel='price', ylabel='Count'>
```



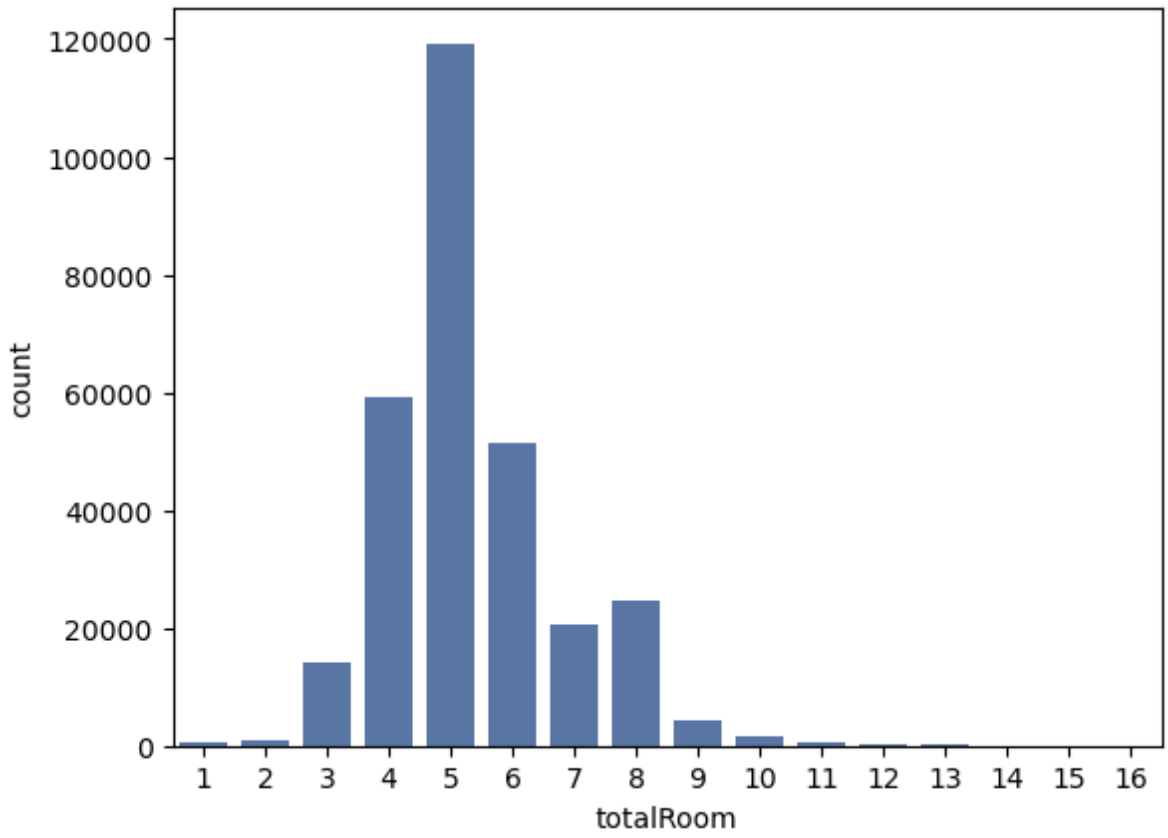
In [27]:
```
sns.histplot(beijing['age'], discrete = True, color="#4c72b0")
```

Out[27]:
```
<AxesSubplot:xlabel='age', ylabel='Count'>
```



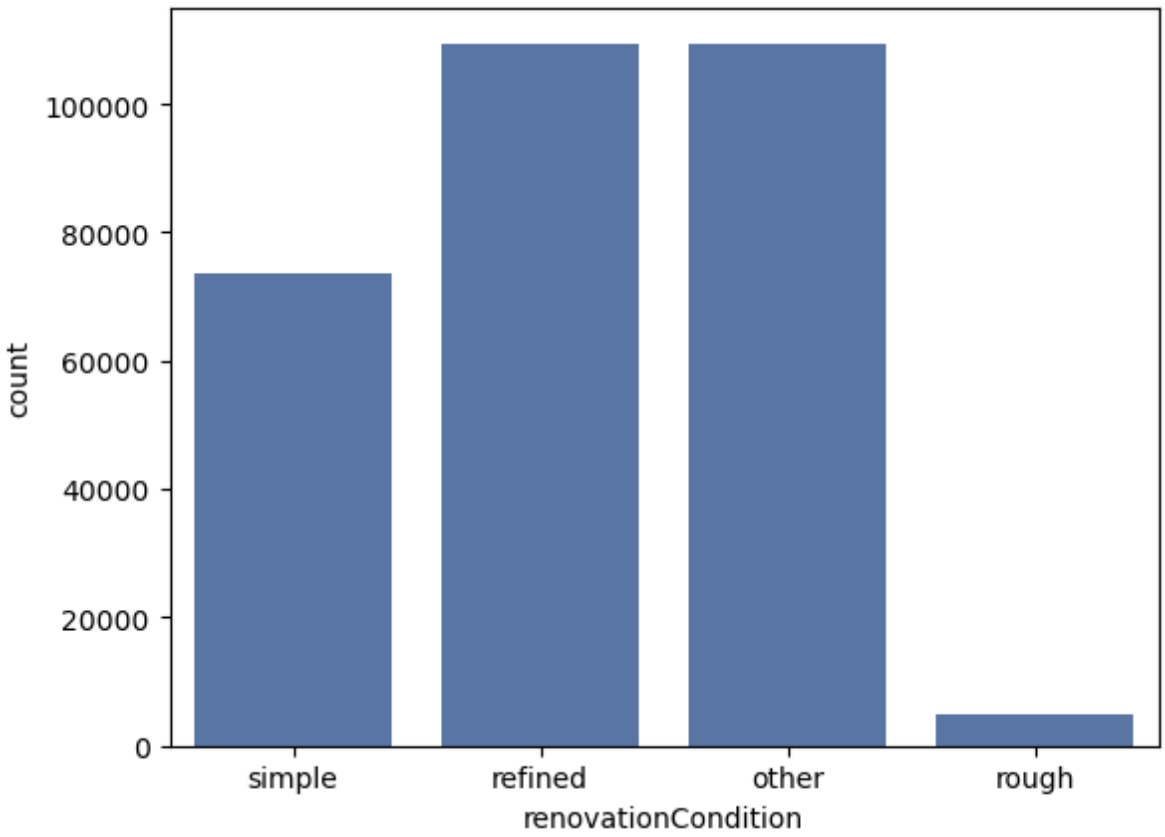
In [28]:
```
beijing['totalRoom'] = beijing['totalRoom'].round(decimals=0).astype(int)

sns.countplot(x=beijing['totalRoom'], color="#4c72b0")
```

Out[28]:
```
<AxesSubplot:xlabel='totalRoom', ylabel='count'>
```



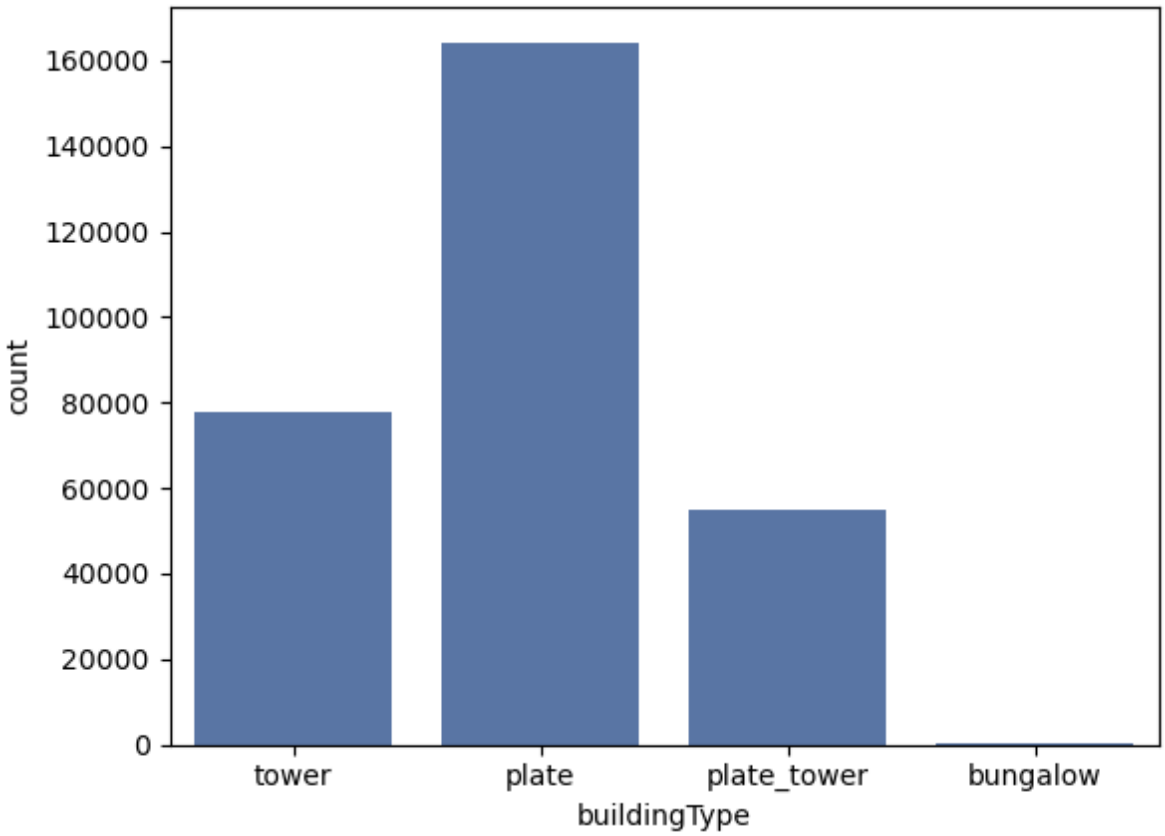
In [29]:
```
sns.countplot(x=beijing['renovationCondition'], color="#4c72b0")
```

Out[29]: <AxesSubplot:xlabel='renovationCondition', ylabel='count'>



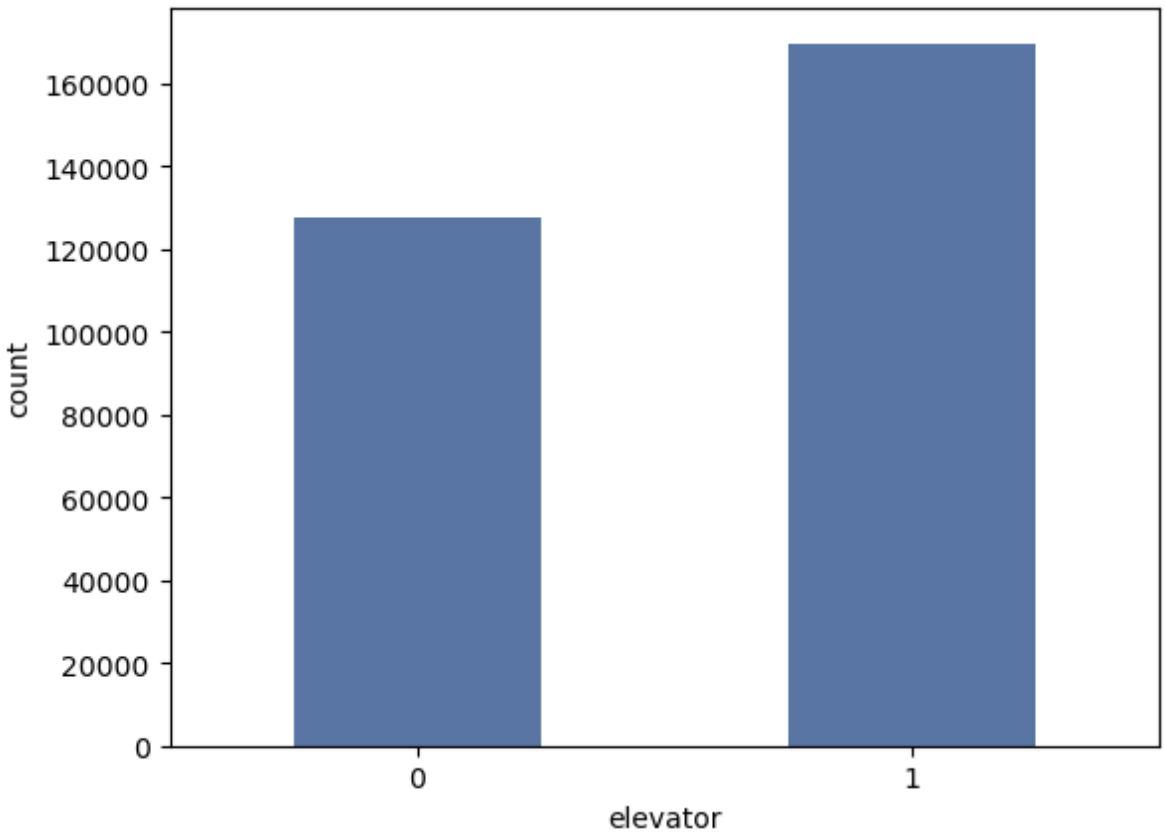
In [30]:
```
sns.countplot(x=beijing['buildingType'], color="#4c72b0")
```

Out[30]: <AxesSubplot:xlabel='buildingType', ylabel='count'>



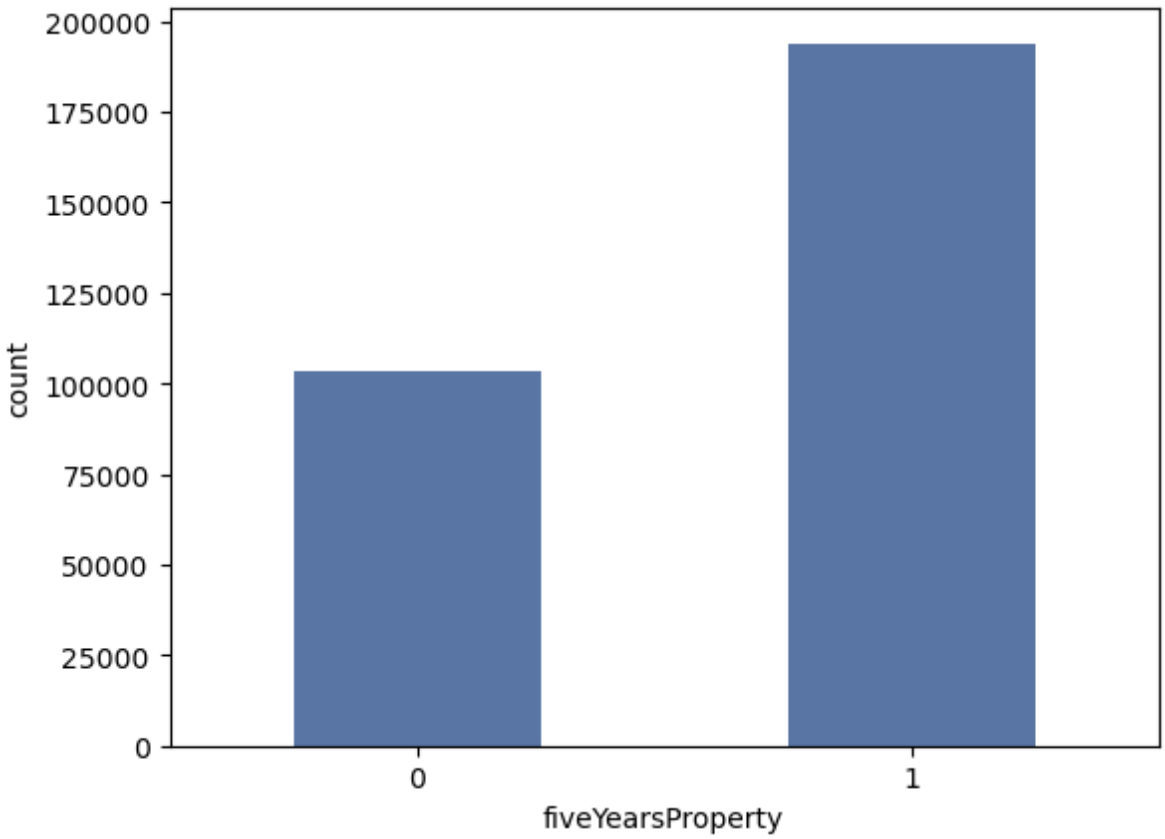
In [31]:
```
beijing['elevator'] = beijing['elevator'].round(decimals=0).astype(int)

sns.countplot(x=beijing['elevator'], color="#4c72b0", width=0.5)
```

Out[31]: <AxesSubplot:xlabel='elevator', ylabel='count'>



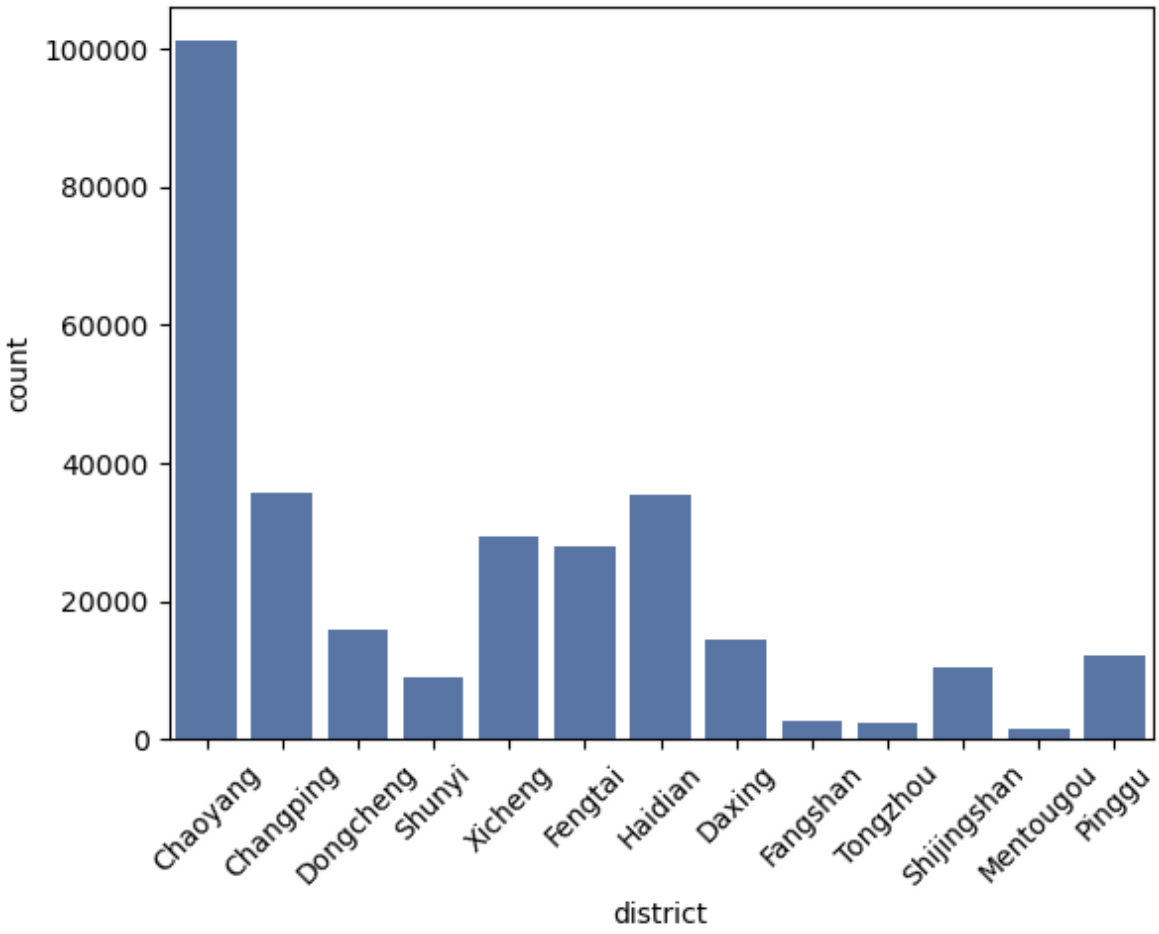
In [32]:
```
beijing['fiveYearsProperty'] = beijing['fiveYearsProperty'].round(decimals=0).astype

sns.countplot(x=beijing['fiveYearsProperty'], color="#4c72b0", width=0.5)
```

Out[32]: <AxesSubplot:xlabel='fiveYearsProperty', ylabel='count'>



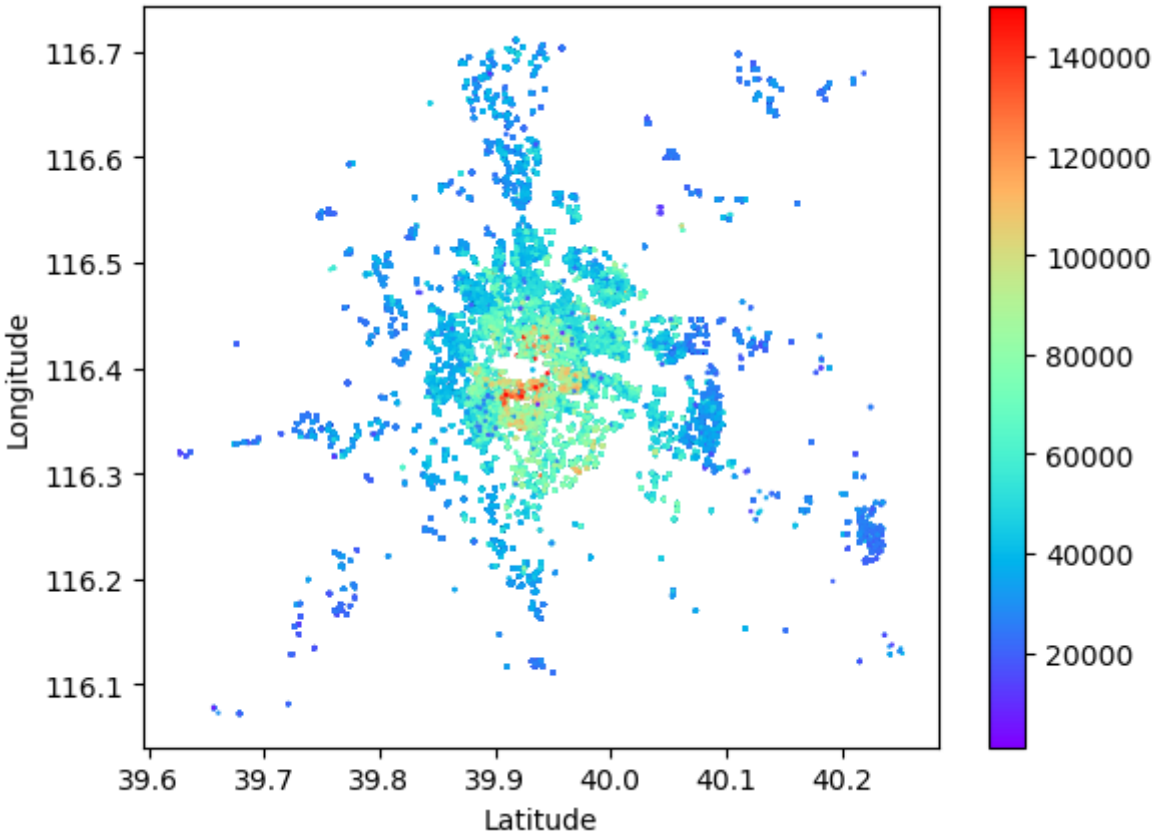
In [33]:
```
district_Plot=sns.countplot(x=beijing['district'], color="#4c72b0")
district_Plot.set_xticklabels(district_Plot.get_xticklabels(), rotation=45)
```

Out[33]:
```
[Text(0, 0, 'Chaoyang'),
 Text(1, 0, 'Changping'),
 Text(2, 0, 'Dongcheng'),
 Text(3, 0, 'Shunyi'),
 Text(4, 0, 'Xicheng'),
 Text(5, 0, 'Fengtai'),
 Text(6, 0, 'Haidian'),
 Text(7, 0, 'Daxing'),
 Text(8, 0, 'Fangshan'),
 Text(9, 0, 'Tongzhou'),
 Text(10, 0, 'Shijingshan'),
 Text(11, 0, 'Mentougou'),
 Text(12, 0, 'Pinggu')]
```



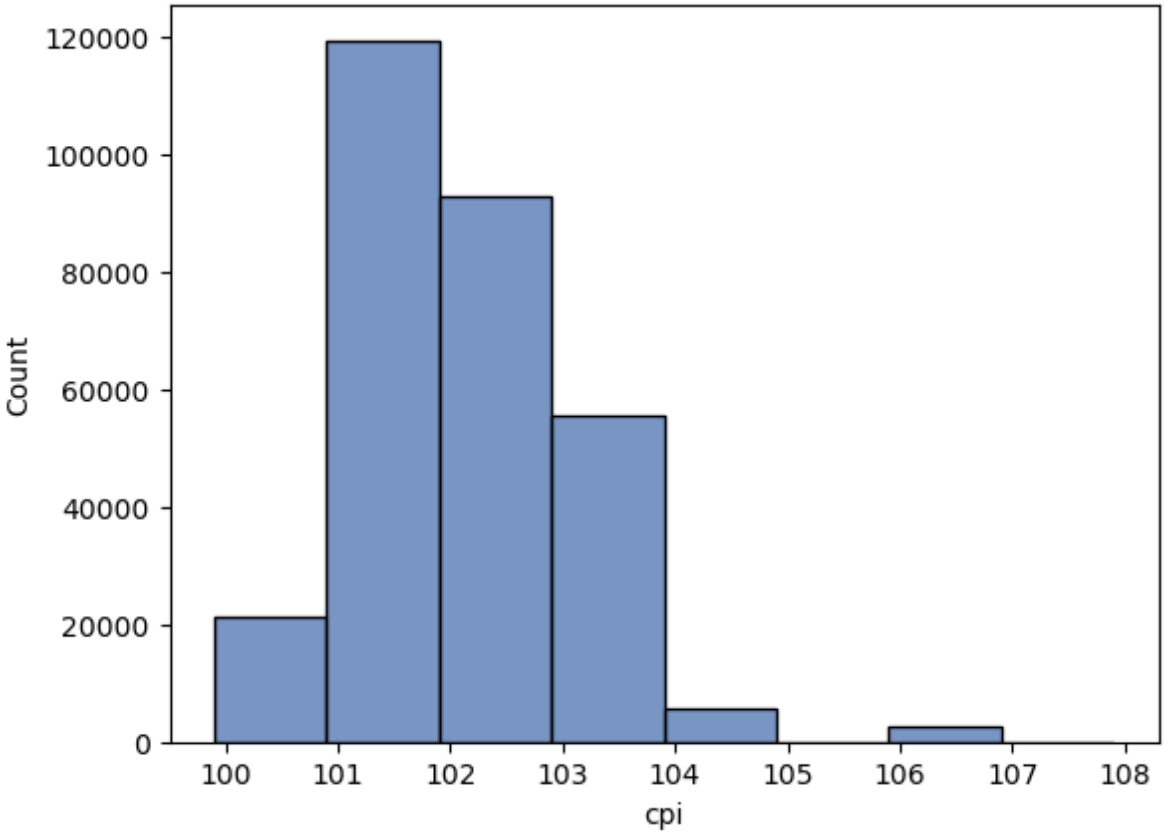
In [34]:
```
plt.scatter(beijing["Lat"], beijing["Lng"],c = beijing["price"], cmap = 'rainbow', s
cbar = plt.colorbar()
plt.xlabel('Latitude')
plt.ylabel('Longitude')
```

Out[34]:
```
Text(0, 0.5, 'Longitude')
```



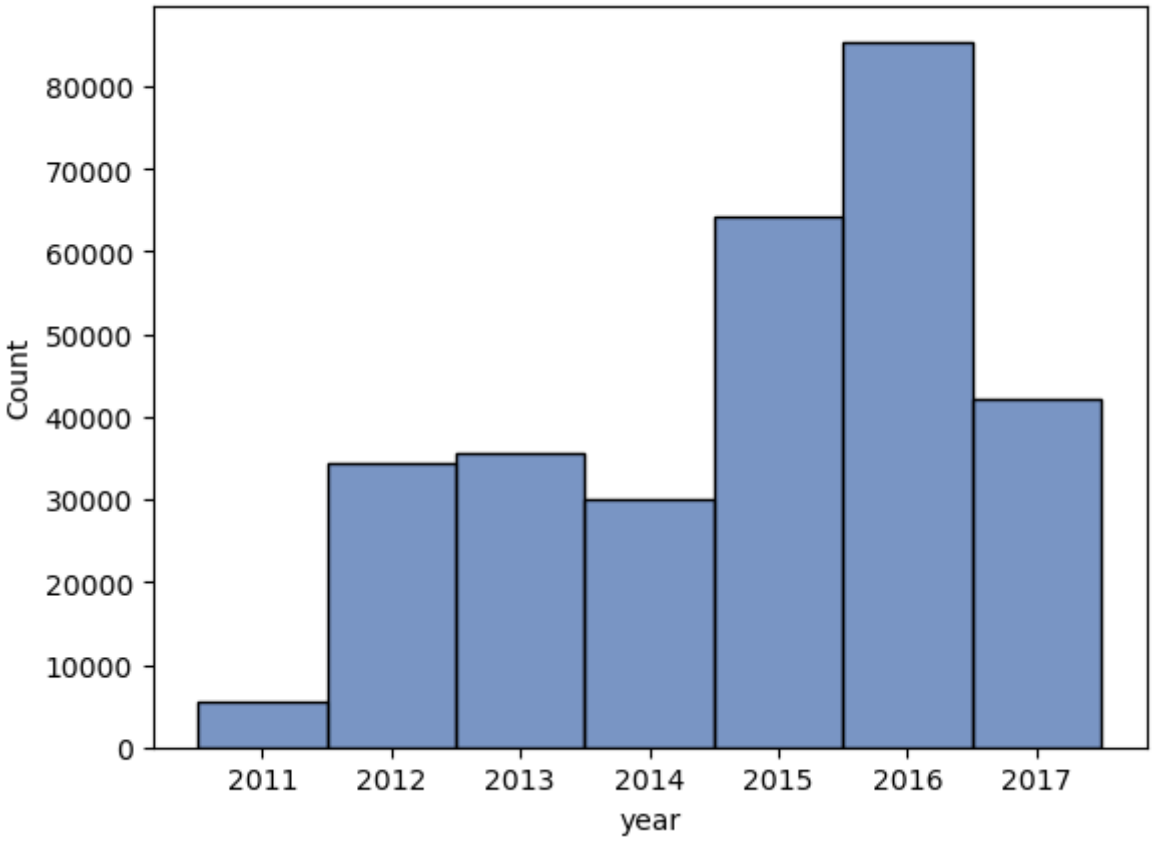
In [35]:
```
sns.histplot(beijing['cpi'], discrete=True, color="#4c72b0")
```

Out[35]:
```
<AxesSubplot:xlabel='cpi', ylabel='Count'>
```



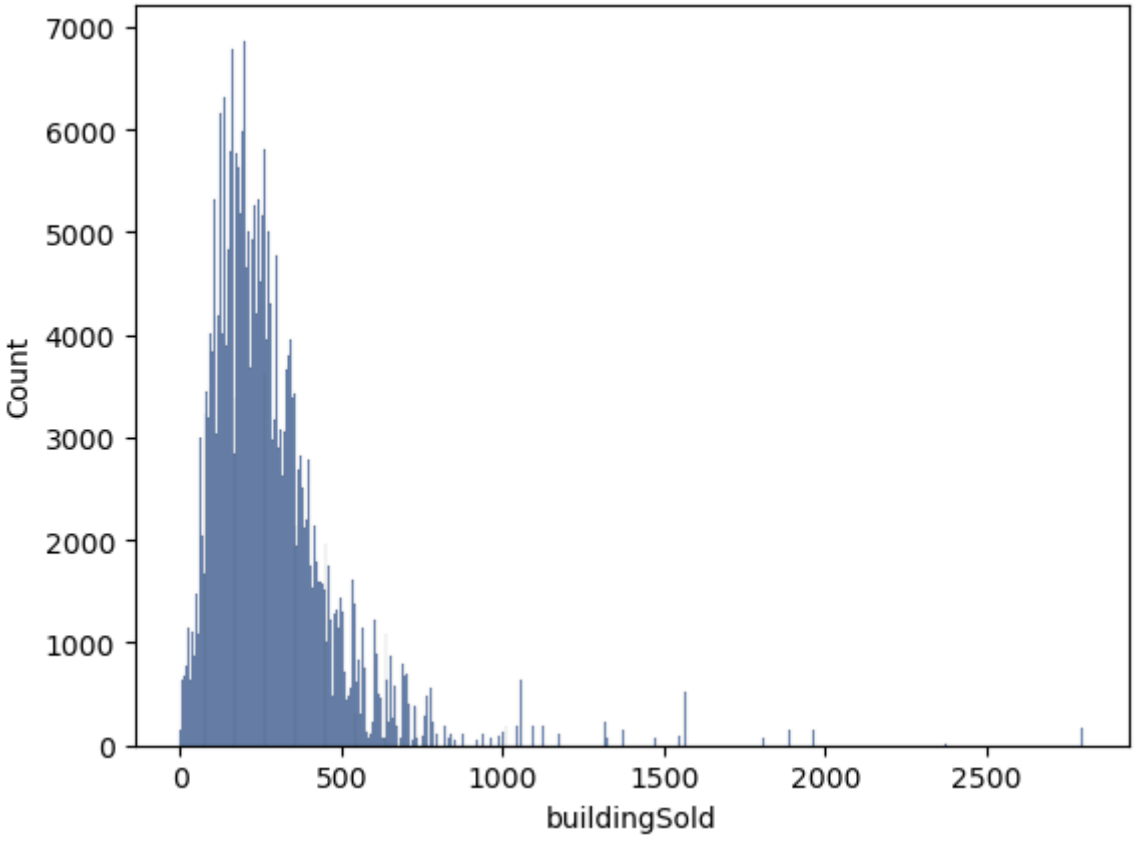
In [36]:
```
sns.histplot(beijing['year'], discrete=True ,color="#4c72b0")
```

Out[36]:
```
<AxesSubplot:xlabel='year', ylabel='Count'>
```



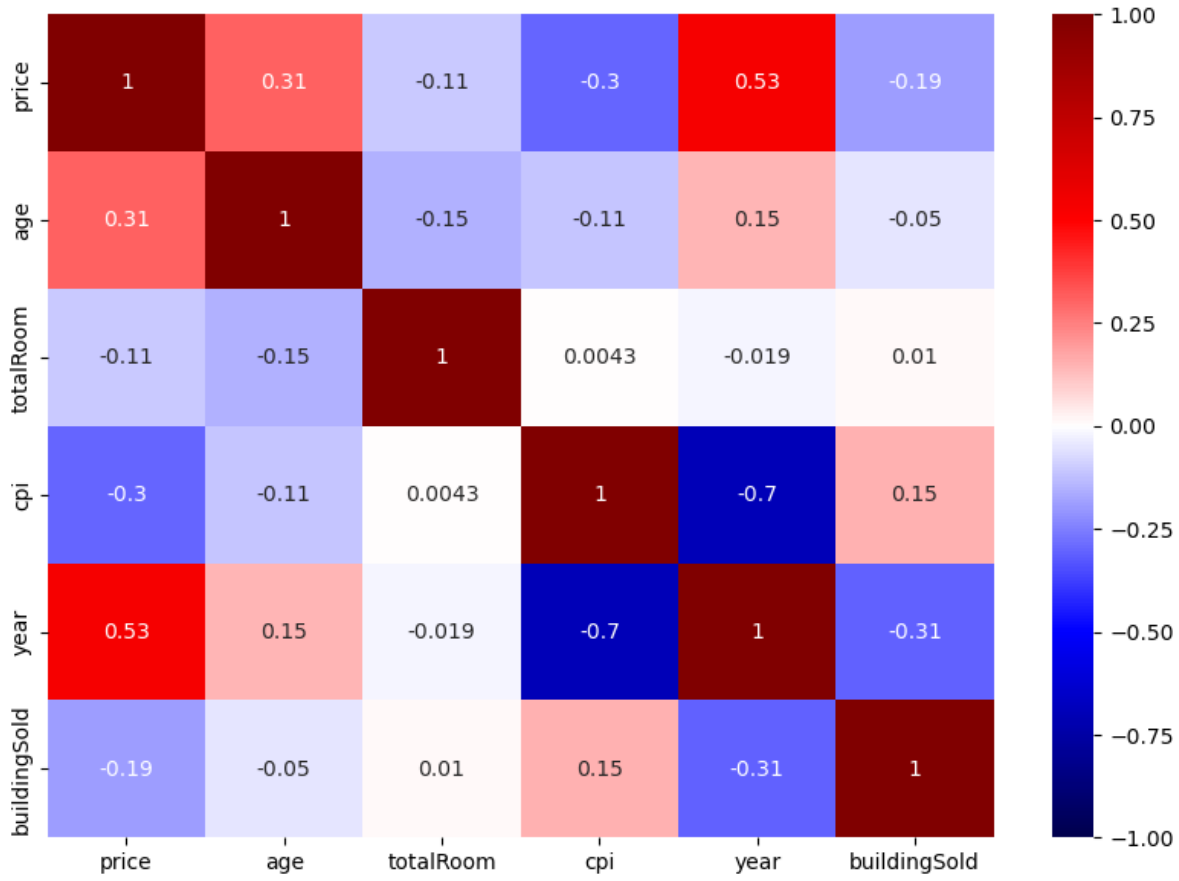
In [37]:
```
sns.histplot(beijing['buildingSold'], color="#4c72b0")
```

Out[37]:
```
<AxesSubplot:xlabel='buildingSold', ylabel='Count'>
```



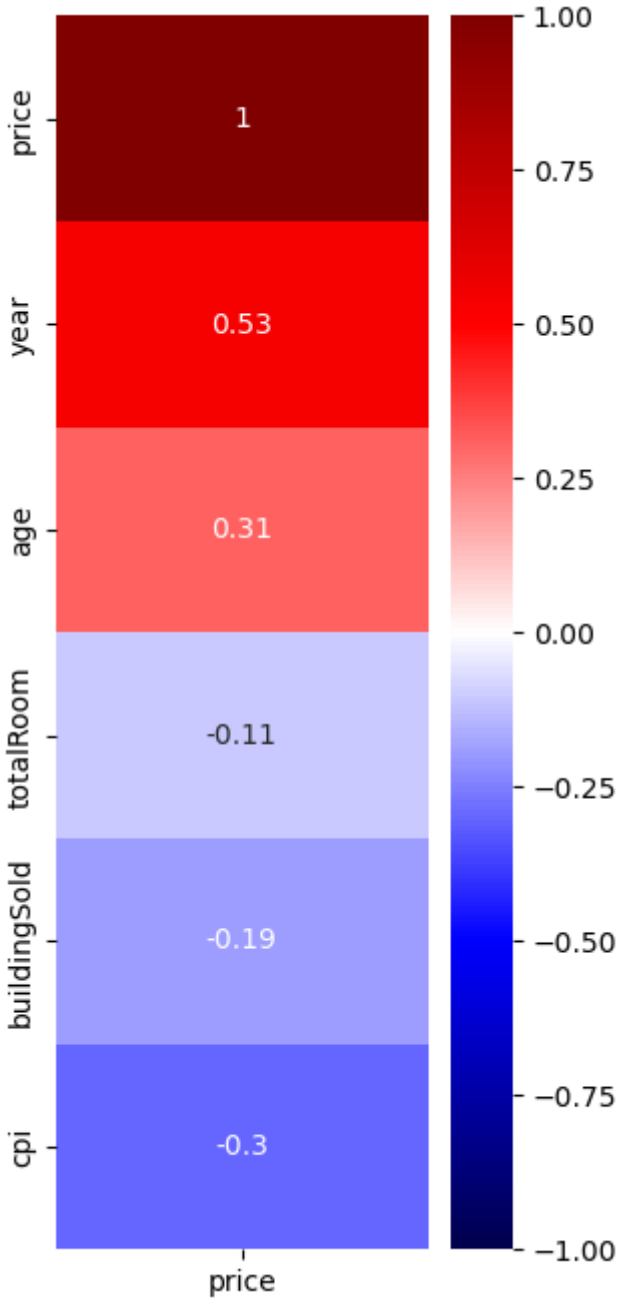
In [38]:
```
continuous = beijing[['price', 'age', 'totalRoom', 'cpi', 'year', 'buildingSold']]
corr = continuous.corr()

plt.figure(figsize=(10,7))
sns.heatmap(corr, cmap='seismic', annot=True, vmax= 1, vmin=-1)
```

Out[38]: <AxesSubplot:>

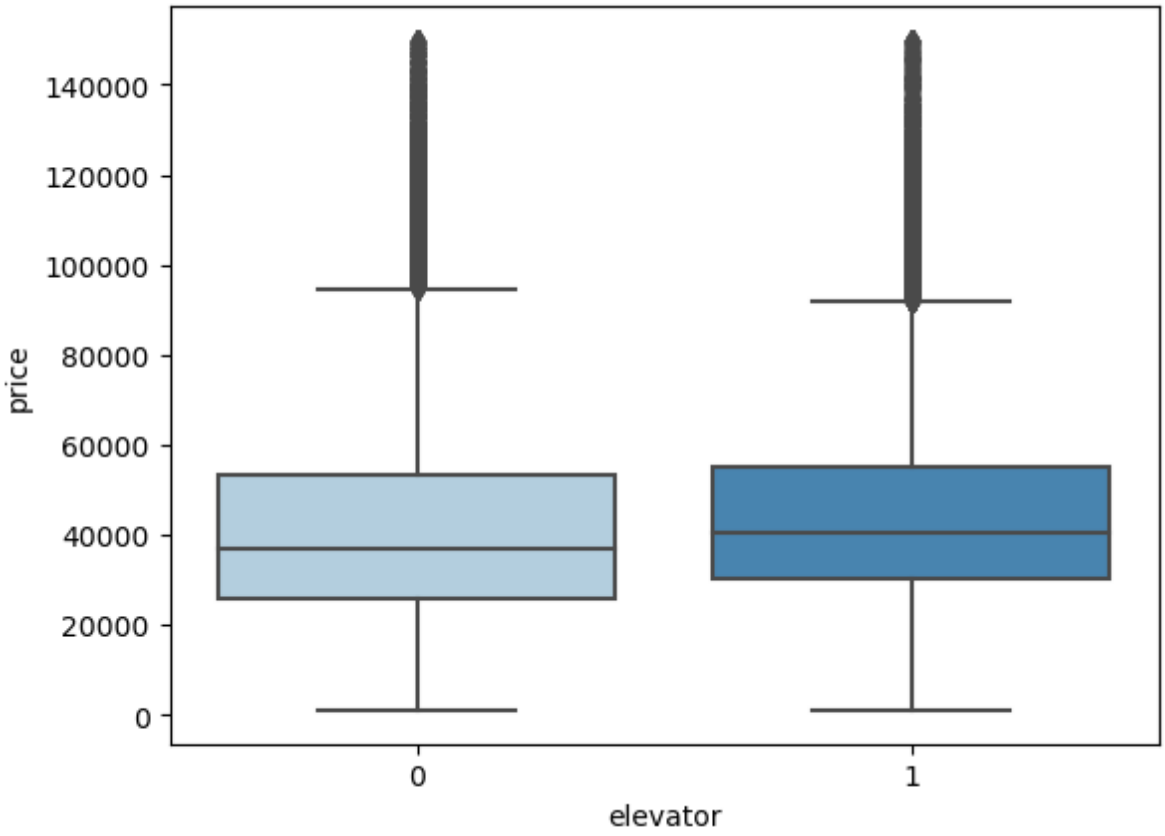


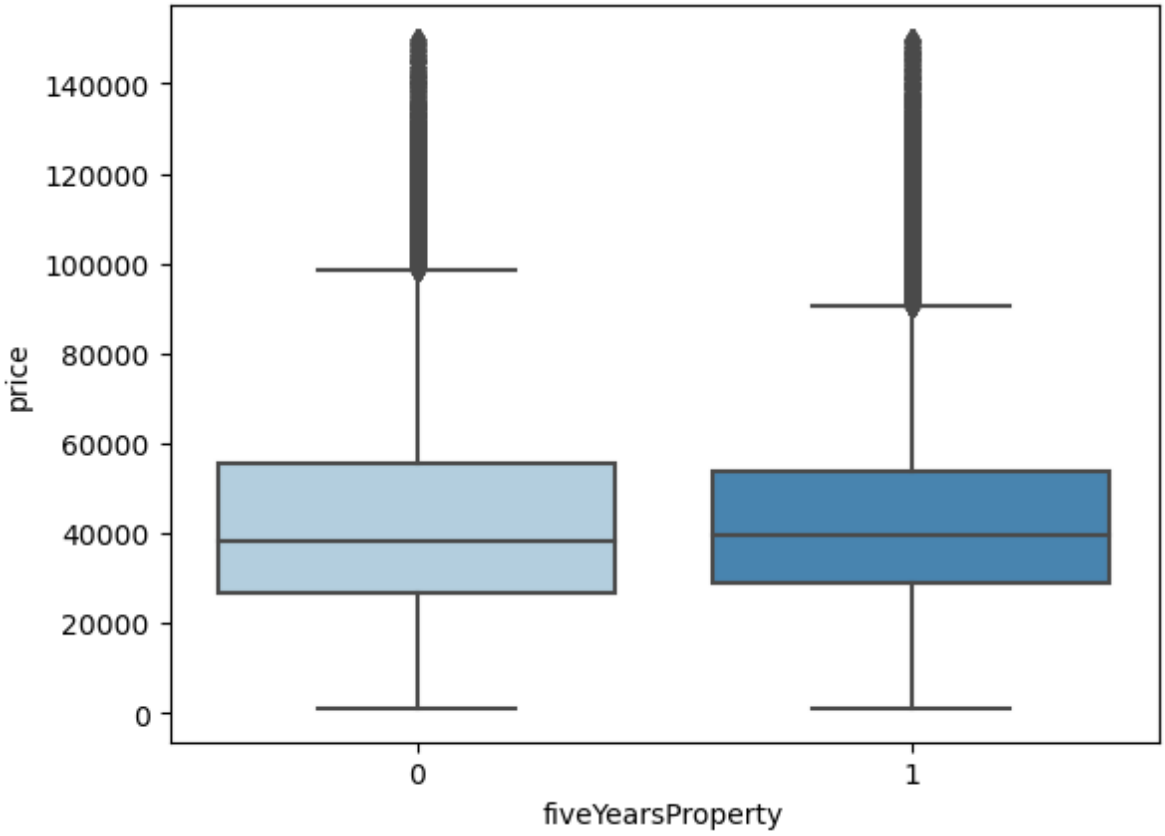
In [39]:
```
price_corr = corr[['price']].sort_values(by=['price'],ascending=False)
plt.figure(figsize=(3,8))
sns.heatmap(price_corr, cmap='seismic', vmax= 1, vmin=-1, annot=True)
```

Out[39]: <AxesSubplot:>



In [40]:
```
sns.boxplot(x='elevator', y='price', data=beijing, palette="Blues")
plt.show()

sns.boxplot(x='fiveYearsProperty', y='price', data=beijing, palette="Blues")
plt.show()
```





In [41]:
```
# calculate the point-biserial correlation coefficient of binary variables (elevator
r_pb, p_value = pointbiserialr(beijing['price'], beijing['elevator'])

print('Point-biserial correlation coefficient:', r_pb)
```

```
Point-biserial correlation coefficient: 0.052691446910817
```

In [42]:
```
# calculate the point-biserial correlation coefficient binary variables (fiveYearsPr
r_pb, p_value = pointbiserialr(beijing['price'], beijing['fiveYearsProperty'])
```

```
print('Point-biserial correlation coefficient:', r_pb)
```

```
Point-biserial correlation coefficient: 0.004287668660858269
```

Continuous (Z-Normalization)

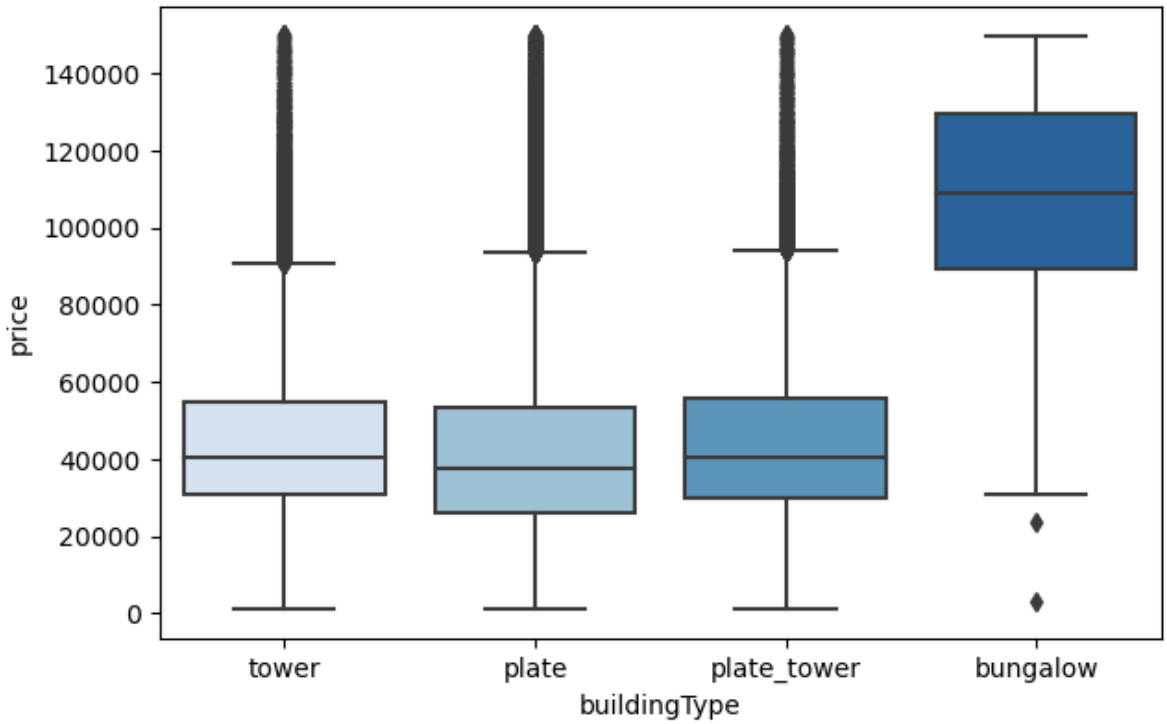
In [44]:
```
continuous = beijing[['price', 'age', 'totalRoom', 'cpi', 'year', 'buildingSold']]

sns.pairplot(data=continuous, diag_kind='kde', diag_kws={"color": "#4c72b0"}, plot_k
```

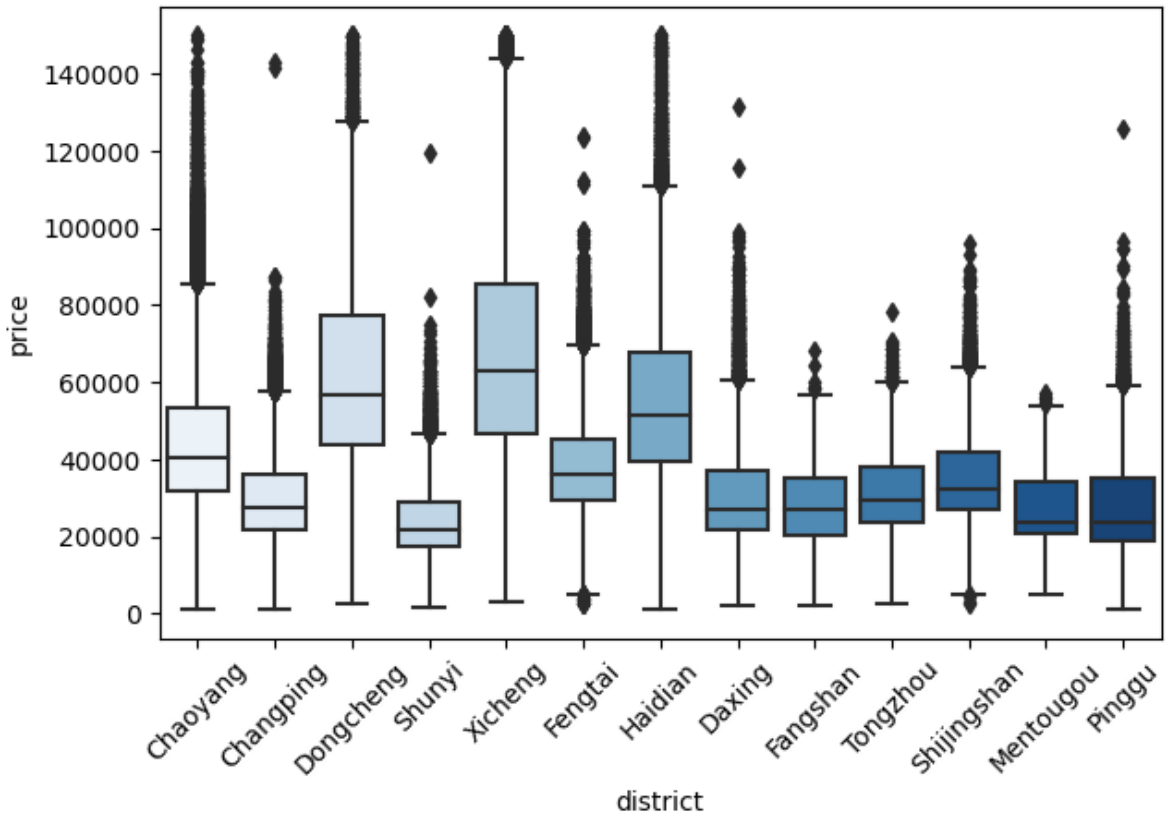
Out[44]:
```
<seaborn.axisgrid.PairGrid at 0x1db3441b430>
```

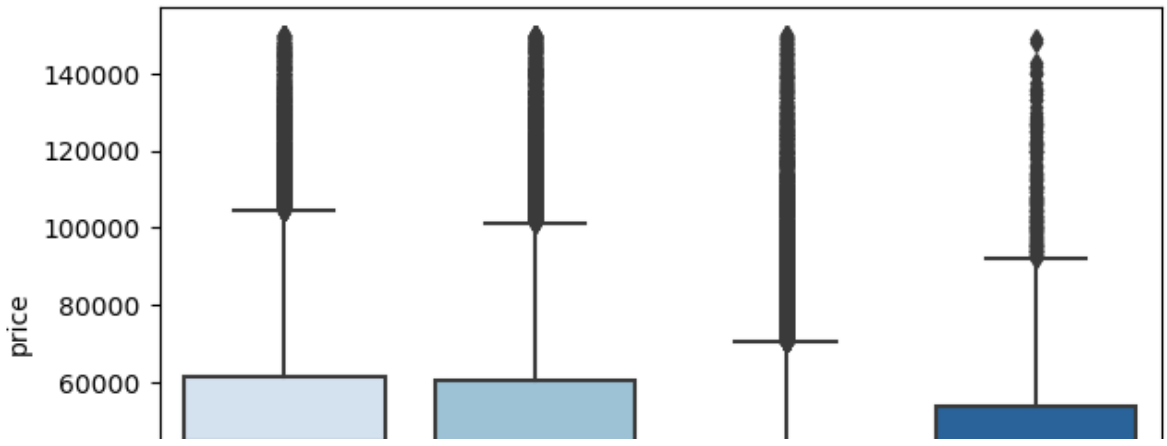
In [45]:
```
# z-score normalization

continuous = beijing[['price', 'age', 'totalRoom','cpi', 'year', 'buildingSold']]
scaler = StandardScaler()
scaled_df = pd.DataFrame(scaler.fit_transform(continuous), index=continuous.index, c
```

Categorical (One-hot encoding)

In [46]:
```
categorical = beijing[['buildingType', 'district', 'renovationCondition']]
c_list=list(categorical.columns)
plt.figure(figsize=[7,16])
plt.subplots_adjust(wspace=0.1, hspace =0.3)

for i in range(len(c_list)):
    plt.subplot(3,1,i+1)
    ax=sns.boxplot(data=beijing,x=beijing[c_list[i]],y='price', palette="Blues")
    if c_list[i] == 'district':
        ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```


price
140000
120000
100000
80000
60000
40000
20000
0
tower
plate
plate_tower
bungalow
buildingType


price
140000
120000
100000
80000
60000
40000
20000
0
Chaoyang
Changping
Dongcheng
Shunyi
Xicheng
Fengtai
Haidian
Daxing
Fangshan
Tongzhou
Shijingshan
Mentougou
Pinggu
district


price
140000
120000
100000
80000
60000

In [47]:
```
# one-hot encoding

categorical = beijing[['buildingType', 'district', 'renovationCondition']]
enc = OneHotEncoder()
encoded = enc.fit_transform(categorical)
```

In [48]:
```
column_names = ['buildingType[T.bungalow]', 'buildingType[T.plate]', 'buildingType[T
                'district[T.Changping]', 'district[T.Chaoyang]', 'district[T.Daxing]
                'renovationCondition[T.other]', 'renovationCondition[T.refined]', 'r

encoded_frame =  pd.DataFrame(encoded.todense(), columns=column_names)
```

Final Dataset

In [49]:
```
# merge all selected columns
beijing_pre = pd.concat([scaled_df, beijing[['elevator', 'fiveYearsProperty','buildi
```

In [50]:
```
y = beijing_pre['price']
X = beijing_pre[['age', 'totalRoom', 'elevator',
                 'renovationCondition[T.other]', 'renovationCondition[T.refined]', '
                 'buildingType[T.bungalow]', 'buildingType[T.plate]', 'buildingType[
                 'fiveYearsProperty', 'cpi', 'year', 'buildingSold',
                 'district[T.Changping]', 'district[T.Chaoyang]', 'district[T.Daxing


x_in = beijing_pre[['age', 'totalRoom', 'elevator', 'renovationCondition[T.other]',
x_out =beijing_pre[['fiveYearsProperty', 'cpi', 'year', 'buildingSold',
                    'district[T.Changping]', 'district[T.Chaoyang]', 'district[T.Da
```

Explanatory Analysis (MLR)

In [54]:
```
formula="price ~ age + totalRoom + elevator + renovationCondition + buildingType"
model_in = smf.ols(formula, data=beijing_pre).fit()

print(model_in.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.221
Model:                            OLS   Adj. R-squared:                  0.221
Method:                 Least Squares   F-statistic:                     9373.
Date:                Tue, 02 May 2023   Prob (F-statistic):               0.00
Time:                        22:20:30   Log-Likelihood:            -3.8461e+05
No. Observations:              297225   AIC:                         7.692e+05
Df Residuals:                  297215   BIC:                         7.694e+05
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
==============
                                   coef    std err          t      P>|t|      [0.0
25      0.975]
------------------------------------------------------------------------------
--------------
Intercept                          1.5730      0.091     17.246      0.000       1.3
94       1.752
renovationCondition[T.refined]     0.5912      0.004    155.626      0.000       0.5
84       0.599
renovationCondition[T.rough]       0.3684      0.013     28.480      0.000       0.3
43       0.394
renovationCondition[T.simple]      0.4920      0.004    113.672      0.000       0.4
83       0.500
buildingType[T.plate]             -2.1756      0.091    -23.851      0.000      -2.3
54      -1.997
buildingType[T.plate_tower]       -2.1255      0.091    -23.267      0.000      -2.3
05      -1.946
buildingType[T.tower]             -2.2995      0.091    -25.186      0.000      -2.4
78      -2.121
age                                0.3853      0.002    199.766      0.000       0.3
82       0.389
totalRoom                         -0.0578      0.002    -34.742      0.000      -0.0
61      -0.055
elevator                           0.4898      0.005     91.854      0.000       0.4
79       0.500
==============================================================================
Omnibus:                    70846.544   Durbin-Watson:                   1.193
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           177745.767
Skew:                           1.310   Prob(JB):                         0.00
Kurtosis:                       5.736   Cond. No.                         158.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```

In [55]:

```
formula="price ~ fiveYearsProperty + cpi + year + buildingSold + district"
model_out = smf.ols(formula, data=beijing_pre).fit()

print(model_out.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.643
Model:	OLS	Adj. R-squared:	0.643
Method:	Least Squares	F-statistic:	3.344e+04
Date:	Tue, 02 May 2023	Prob (F-statistic):	0.00
Time:	22:20:33	Log-Likelihood:	-2.6873e+05
No. Observations:	297225	AIC:	5.375e+05
Df Residuals:	297208	BIC:	5.377e+05
Df Model:	16		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.6367	0.003	-183.512	0.000	-0.643	-0.630
district[T.Chaoyang]	0.6941	0.004	188.370	0.000	0.687	0.701
district[T.Daxing]	0.0615	0.006	10.418	0.000	0.050	0.073
district[T.Dongcheng]	1.5259	0.006	267.815	0.000	1.515	1.537
district[T.Fangshan]	-0.4876	0.012	-40.119	0.000	-0.511	-0.464
district[T.Fengtai]	0.4616	0.005	96.505	0.000	0.452	0.471
district[T.Haidian]	1.2625	0.004	281.179	0.000	1.254	1.271
district[T.Mentougou]	-0.1451	0.016	-9.036	0.000	-0.177	-0.114
district[T.Pinggu]	0.0473	0.006	7.521	0.000	0.035	0.060
district[T.Shijingshan]	0.3667	0.007	55.037	0.000	0.354	0.380
district[T.Shunyi]	-0.1308	0.007	-18.438	0.000	-0.145	-0.117
district[T.Tongzhou]	-0.0043	0.013	-0.331	0.741	-0.030	0.021
district[T.Xicheng]	1.8451	0.005	390.617	0.000	1.836	1.854
fiveYearsProperty	-0.1004	0.002	-43.083	0.000	-0.105	-0.096
cpi	0.1646	0.002	106.150	0.000	0.162	0.168
year	0.6556	0.002	407.396	0.000	0.652	0.659
buildingSold	-0.0241	0.001	-20.782	0.000	-0.026	-0.022

Omnibus:	40567.575	Durbin-Watson:	1.557
Prob(Omnibus):	0.000	Jarque-Bera (JB):	111771.747
Skew:	0.751	Prob(JB):	0.00
Kurtosis:	5.602	Cond. No.	20.3

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [56]:
```
formula= "price ~ age + totalRoom + elevator + renovationCondition + buildingType +
model_all = smf.ols(formula, data=beijing_pre).fit()
```

```
print(model_all.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.656
Model:	OLS	Adj. R-squared:	0.656
Method:	Least Squares	F-statistic:	2.269e+04
Date:	Tue, 02 May 2023	Prob (F-statistic):	0.00
Time:	22:20:40	Log-Likelihood:	-2.6307e+05
No. Observations:	297225	AIC:	5.262e+05
Df Residuals:	297199	BIC:	5.265e+05
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.1616	0.061	19.135	0.000	1.043	1.281
renovationCondition[T.refined]	-0.1126	0.003	-32.584	0.000	-0.119	-0.106
renovationCondition[T.rough]	-0.2604	0.009	-29.322	0.000	-0.278	-0.243
renovationCondition[T.simple]	-0.2168	0.004	-58.313	0.000	-0.224	-0.209
buildingType[T.plate]	-1.7116	0.061	-28.221	0.000	-1.831	-1.593
buildingType[T.plate_tower]	-1.7728	0.061	-29.187	0.000	-1.892	-1.654
buildingType[T.tower]	-1.8965	0.061	-31.235	0.000	-2.016	-1.778
district[T.Chaoyang]	0.6307	0.004	162.873	0.000	0.623	0.638
district[T.Daxing]	0.0523	0.006	8.993	0.000	0.041	0.064
district[T.Dongcheng]	1.4363	0.006	246.364	0.000	1.425	1.448
district[T.Fangshan]	-0.5003	0.012	-41.764	0.000	-0.524	-0.477
district[T.Fengtai]	0.4319	0.005	88.562	0.000	0.422	0.441
district[T.Haidian]	1.2025	0.005	261.625	0.000	1.193	1.211
district[T.Mentougou]	-0.1290	0.016	-8.158	0.000	-0.160	-0.098
district[T.Pinggu]	0.0455	0.006	7.345	0.000	0.033	0.058
district[T.Shijingshan]	0.2930	0.007	43.729	0.000	0.280	0.306
district[T.Shunyi]	-0.1491	0.007	-21.374	0.000	-0.163	-0.135
district[T.Tongzhou]	-0.0091	0.013	-0.709	0.478	-0.034	0.016
district[T.Xicheng]	1.7543	0.005	350.568	0.000	1.745	1.764
age	0.1029	0.001	71.566	0.000	0.100	0.106
totalRoom	-0.0111	0.001	-9.940	0.000	-0.013	-0.009
elevator	0.2050	0.004	56.354	0.000	0.198	0.212
fiveYearsProperty	-0.0942	0.002	-39.578	0.000	-0.099	-0.090
cpi	0.1453	0.002	92.448	0.000	0.142	0.148

```
42       0.148
year                            0.6777      0.002    376.086      0.000       0.6
74       0.681
buildingSold                    -0.0192      0.001    -16.853      0.000      -0.0
21      -0.017
==============================================================================
Omnibus:                    41426.882   Durbin-Watson:                   1.572
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           121412.334
Skew:                           0.747   Prob(JB):                         0.00
Kurtosis:                       5.752   Cond. No.                         182.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```

In [57]:
```
stargazer = Stargazer([model_in, model_out, model_all])
stargazer.covariate_order(['Intercept', 'age', 'totalRoom', 'elevator',
                           'renovationCondition[T.refined]', 'renovationCondition[T.
                           'buildingType[T.plate]', 'buildingType[T.plate_tower]' ,
                           'cpi', 'year', 'fiveYearsProperty', 'buildingSold',
                           'district[T.Chaoyang]', 'district[T.Daxing]', 'district[T

print(stargazer.render_latex())
```

```
Wbegin{table}[!htbp] Wcentering
Wbegin{tabular}{@{Wextracolsep{5pt}}lccc}
WW[-1.8ex]Whline
Whline WW[-1.8ex]
& Wmulticolumn{3}{c}{Wtextit{Dependent variable:}} W
Wcr Wcline{3-4}
WW[-1.8ex] & (1) & (2) & (3) WW
Whline WW[-1.8ex]
 Intercept & 1.573$^{***}$ & -0.637$^{***}$ & 1.162$^{***}$ WW
  & (0.091) & (0.003) & (0.061) WW
 age & 0.385$^{***}$ & & 0.103$^{***}$ WW
  & (0.002) & & (0.001) WW
 totalRoom & -0.058$^{***}$ & & -0.011$^{***}$ WW
  & (0.002) & & (0.001) WW
 elevator & 0.490$^{***}$ & & 0.205$^{***}$ WW
  & (0.005) & & (0.004) WW
 renovationCondition[T.refined] & 0.591$^{***}$ & & -0.113$^{***}$ WW
  & (0.004) & & (0.003) WW
 renovationCondition[T.rough] & 0.368$^{***}$ & & -0.260$^{***}$ WW
  & (0.013) & & (0.009) WW
 renovationCondition[T.simple] & 0.492$^{***}$ & & -0.217$^{***}$ WW
  & (0.004) & & (0.004) WW
 buildingType[T.plate] & -2.176$^{***}$ & & -1.712$^{***}$ WW
  & (0.091) & & (0.061) WW
 buildingType[T.plate_tower] & -2.125$^{***}$ & & -1.773$^{***}$ WW
  & (0.091) & & (0.061) WW
 buildingType[T.tower] & -2.300$^{***}$ & & -1.897$^{***}$ WW
  & (0.091) & & (0.061) WW
 cpi & & 0.165$^{***}$ & 0.145$^{***}$ WW
  & & (0.002) & (0.002) WW
 year & & 0.656$^{***}$ & 0.678$^{***}$ WW
  & & (0.002) & (0.002) WW
 fiveYearsProperty & & -0.100$^{***}$ & -0.094$^{***}$ WW
  & & (0.002) & (0.002) WW
 buildingSold & & -0.024$^{***}$ & -0.019$^{***}$ WW
  & & (0.001) & (0.001) WW
 district[T.Chaoyang] & & 0.694$^{***}$ & 0.631$^{***}$ WW
  & & (0.004) & (0.004) WW
 district[T.Daxing] & & 0.062$^{***}$ & 0.052$^{***}$ WW
  & & (0.006) & (0.006) WW
 district[T.Dongcheng] & & 1.526$^{***}$ & 1.436$^{***}$ WW
  & & (0.006) & (0.006) WW
 district[T.Fangshan] & & -0.488$^{***}$ & -0.500$^{***}$ WW
  & & (0.012) & (0.012) WW
 district[T.Fengtai] & & 0.462$^{***}$ & 0.432$^{***}$ WW
  & & (0.005) & (0.005) WW
 district[T.Haidian] & & 1.262$^{***}$ & 1.202$^{***}$ WW
  & & (0.004) & (0.005) WW
 district[T.Mentougou] & & -0.145$^{***}$ & -0.129$^{***}$ WW
  & & (0.016) & (0.016) WW
 district[T.Pinggu] & & 0.047$^{***}$ & 0.046$^{***}$ WW
  & & (0.006) & (0.006) WW
 district[T.Shijingshan] & & 0.367$^{***}$ & 0.293$^{***}$ WW
  & & (0.007) & (0.007) WW
 district[T.Shunyi] & & -0.131$^{***}$ & -0.149$^{***}$ WW
  & & (0.007) & (0.007) WW
 district[T.Tongzhou] & & -0.004$^{}$ & -0.009$^{}$ WW
  & & (0.013) & (0.013) WW
 district[T.Xicheng] & & 1.845$^{***}$ & 1.754$^{***}$ WW
  & & (0.005) & (0.005) WW
Whline WW[-1.8ex]
 Observations & 297,225 & 297,225 & 297,225 WW
 $R^2$ & 0.221 & 0.643 & 0.656 WW
 Adjusted $R^2$ & 0.221 & 0.643 & 0.656 WW
```

```
 Residual Std. Error & 0.883(df = 297215) & 0.598(df = 297208) & 0.586(df = 297199)
WW
 F Statistic & 9372.745$^{***}$ (df = 9.0; 297215.0) & 33436.556$^{***}$ (df = 16.0;
297208.0) & 22691.533$^{***}$ (df = 25.0; 297199.0) WW
Whline
Whline WW[-1.8ex]
Wtextit{Note:} & Wmulticolumn{3}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.0
1} WW
Wend{tabular}
Wend{table}
```

Interaction/Ploynominal Terms For MLR

In [59]:
```
interaction = PolynomialFeatures(degree=2, include_bias=False, interaction_only=Fa
poly_features = interaction.fit_transform(X)
X_inter = pd.DataFrame(interaction.fit_transform(X), columns=interaction.get_feature
```

In [61]:
```
print('The number of interaction and second degree polynomials:', len(interaction.ge
```

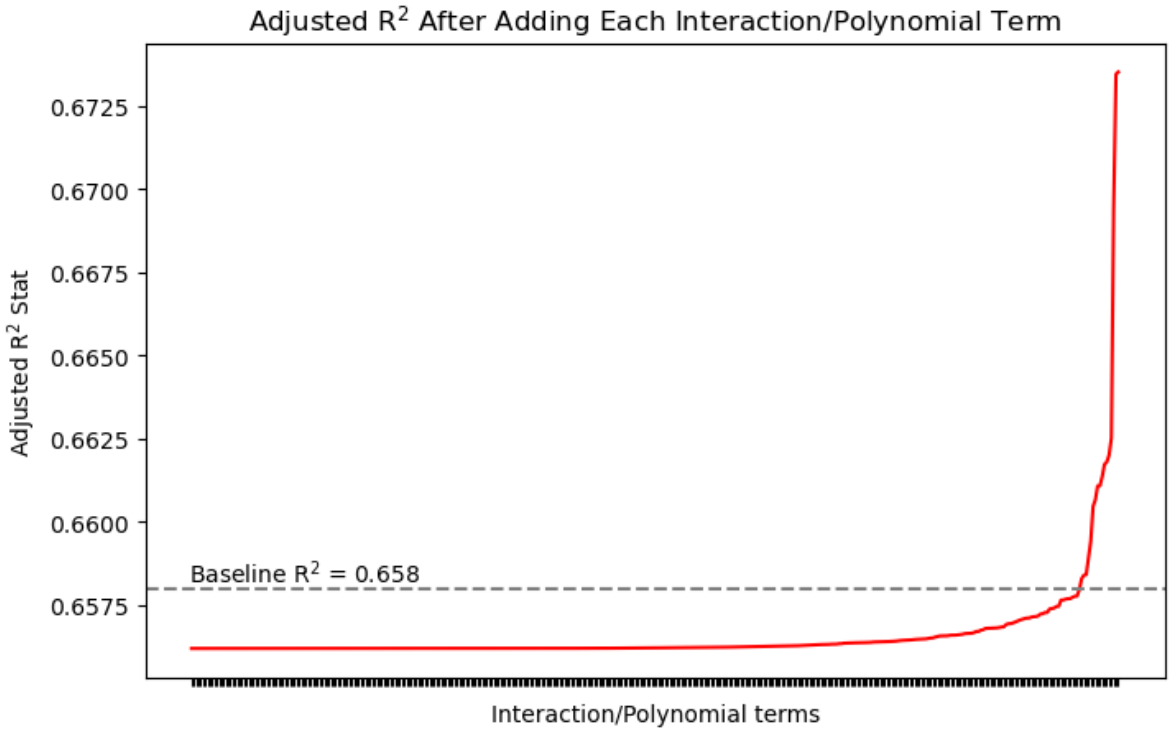
```
The number of interaction and second degree polynomials: 434
```

In [62]:
```
X_inter_alt = X_inter.iloc[:, 0:29]

# 1. Create new datasets where the 29th item changes
# 2. print all adjusted r squared for each model

numbers = []
for i in range(28, 434):
    X_inter_alt_2 = X_inter.iloc[:, np.r_[0:28, i]]

    model = sm.OLS(y, sm.add_constant(X_inter_alt_2)).fit()
    numbers.append(model.rsquared_adj)
```

In [63]:
```
# place the array into a data frame
numbers_df = pd.DataFrame(numbers)

# get names of interative terms
names = X_inter.iloc[: , 28:434]
col_heads = pd.DataFrame(list(names))

# combine adjusted r-squared results & names of interative terms
output = pd.concat([col_heads, numbers_df.reset_index(drop=True)], axis=1)
output.columns = ['features', 'rsquared_adj']

# sort outputs for graph
output_sort = output.sort_values(by=['rsquared_adj'])
```

In [64]:
```
x_plot = output_sort['features']
y_plot = output_sort['rsquared_adj']

fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(x_plot, y_plot, 'r')
ax.set_xlabel('Interaction/Polynomial terms')
ax.set_ylabel('Adjusted R$^2$ Stat')
ax.set_title('Adjusted R$^2$ After Adding Each Interaction/Polynomial Term')

# Remove x-axis labels
ax.set_xticklabels([])

# Add horizontal line at y=0.66
ax.axhline(y=0.658, color='grey', linestyle='--')
```

```
ax.annotate('Baseline R$^2$ = 0.658', xy=(100, 0.658), ha='right', va='bottom')

plt.show()
```



In [65]:
```
best_order = output.sort_values(by=['rsquared_adj'], ascending=False)
best = best_order[best_order['rsquared_adj'] > 0.658]
```

In [66]:
```
new = best['features'].tolist()
orginal = X_inter.iloc[:, 0:28].columns.tolist()
final = new + orginal

# dataframe with original featrues and newly added interaction terms
selected_df = X_inter[final]
selected_df = selected_df.drop(["buildingType[T.bungalow]", "district[T.Changping]",
```

In [67]:
```
selected_df.columns = selected_df.columns.str.replace(' ', ':')
selected_df.columns = selected_df.columns.str.replace('^', '_', regex=True)
```

In [68]:
```
#Fit model with new dataset stored in selected_df

model_2 = sm.OLS(y, sm.add_constant(selected_df)).fit()
print(model_2.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.714
Model:	OLS	Adj. R-squared:	0.714
Method:	Least Squares	F-statistic:	1.769e+04
Date:	Tue, 02 May 2023	Prob (F-statistic):	0.00
Time:	22:29:50	Log-Likelihood:	-2.3556e+05
No. Observations:	297225	AIC:	4.712e+05
Df Residuals:	297182	BIC:	4.717e+05
Df Model:	42		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.6931	0.056	12.363	0.000	0.583	0.803
year:district[T.Xicheng]	0.5551	0.005	106.480	0.000	0.545	0.565
year_2	0.1114	0.002	66.785	0.000	0.108	0.115
renovationCondition[T.other]:year	-0.0873	0.014	-6.357	0.000	-0.114	-0.060
renovationCondition[T.other]:district[T.Xicheng]	0.0862	0.011	8.128	0.000	0.065	0.107
cpi:district[T.Xicheng]	0.0677	0.005	14.262	0.000	0.058	0.077
age:district[T.Haidian]	0.2315	0.003	70.265	0.000	0.225	0.238
renovationCondition[T.refined]:year	0.0899	0.014	6.612	0.000	0.063	0.117
year:district[T.Changping]	-0.1185	0.003	-37.669	0.000	-0.125	-0.112
age:year	0.0088	0.001	8.193	0.000	0.007	0.011
year:buildingSold	-0.0284	0.001	-22.231	0.000	-0.031	-0.026
year:district[T.Dongcheng]	0.3669	0.005	79.569	0.000	0.358	0.376
renovationCondition[T.other]:cpi	0.0274	0.004	6.713	0.000	0.019	0.035
renovationCondition[T.simple]:year	0.0422	0.014	3.073	0.002	0.015	0.069
age:totalRoom	-0.0702	0.001	-49.665	0.000	-0.073	-0.067
year:district[T.Haidian]	0.1710	0.003	54.171	0.000	0.165	0.177
renovationCondition[T.refined]:cpi	-0.0108	0.004	-2.467	0.014	-0.019	-0.002
renovationCondition[T.simple]:district[T.Xicheng]	0.0423	0.008	5.097	0.000	0.026	0.059
age	0.0495	0.001	35.114	0.000	0.047	0.052
totalRoom	-0.0448	0.001	-37.339	0.000	-0.047	-0.042
elevator	0.1996	0.003	59.871	0.000	0.193	0.206
renovationCondition[T.refined]	-0.1024	0.004	-26.929	0.000	-0.110	-0.095
renovationCondition[T.rough]	-0.2067	0.010	-20.192	0.000	-0.227	-0.187
renovationCondition[T.simple]	-0.2043	0.004	-47.615			

```
0.000      -0.213      -0.196
buildingType[T.plate]                                -1.4884      0.056    -26.594
0.000      -1.598      -1.379
buildingType[T.plate_tower]                          -1.5470      0.056    -27.583
0.000      -1.657      -1.437
buildingType[T.tower]                                -1.6639      0.056    -29.659
0.000      -1.774      -1.554
fiveYearsProperty                                    -0.0288      0.002    -13.075
0.000      -0.033      -0.025
cpi                                                   0.0292      0.003      8.634
0.000       0.023       0.036
year                                                  0.6315      0.013     47.194
0.000       0.605       0.658
buildingSold                                          0.0001      0.001      0.114
0.909      -0.002       0.002
district[T.Chaoyang]                                  0.6300      0.004    177.470
0.000       0.623       0.637
district[T.Daxing]                                    0.0415      0.005      7.819
0.000       0.031       0.052
district[T.Dongcheng]                                 1.4467      0.005    271.049
0.000       1.436       1.457
district[T.Fangshan]                                 -0.4860      0.011    -44.231
0.000      -0.508      -0.464
district[T.Fengtai]                                   0.4237      0.004     95.052
0.000       0.415       0.432
district[T.Haidian]                                   1.1678      0.004    276.884
0.000       1.159       1.176
district[T.Mentougou]                                -0.0669      0.015     -4.593
0.000      -0.096      -0.038
district[T.Pinggu]                                    0.0165      0.006      2.924
0.003       0.005       0.028
district[T.Shijingshan]                               0.2958      0.006     48.266
0.000       0.284       0.308
district[T.Shunyi]                                   -0.1598      0.006    -25.110
0.000      -0.172      -0.147
district[T.Tongzhou]                                  0.0027      0.012      0.232
0.817      -0.020       0.026
district[T.Xicheng]                                   1.7557      0.007    247.412
0.000       1.742       1.770
==============================================================================
Omnibus:                    35614.295   Durbin-Watson:                   1.695
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           124416.484
Skew:                           0.595   Prob(JB):                         0.00
Kurtosis:                       5.937   Cond. No.                         255.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```

In [70]:

```
stargazer = Stargazer([model_all,model_2])
stargazer.covariate_order(['Intercept', 'const', 'age', 'totalRoom', 'elevator',
                           'renovationCondition[T.refined]', 'renovationCondition[T.
                           'buildingType[T.plate]', 'buildingType[T.plate_tower]' ,
                           'cpi', 'year', 'fiveYearsProperty', 'buildingSold',
                           'district[T.Chaoyang]', 'district[T.Daxing]', 'district[T
                           'cpi:district[T.Xicheng]', 'age:district[T.Haidian]', 'ag
stargazer.rename_covariates({'const': 'Intercept'})
print(stargazer.render_latex())
```

```
Wbegin{table}[!htbp] Wcentering
Wbegin{tabular}{@{Wextracolsep{5pt}}lcc}
WW[-1.8ex]Whline
Whline WW[-1.8ex]
& Wmulticolumn{2}{c}{Wtextit{Dependent variable:}} W
Wcr Wcline{2-3}
WW[-1.8ex] & (1) & (2) WW
Whline WW[-1.8ex]
 Intercept & 1.162$^{***}$ & WW
  & (0.061) & WW
 Intercept & & 0.693$^{***}$ WW
  & & (0.056) WW
 age & 0.103$^{***}$ & 0.050$^{***}$ WW
  & (0.001) & (0.001) WW
 totalRoom & -0.011$^{***}$ & -0.045$^{***}$ WW
  & (0.001) & (0.001) WW
 elevator & 0.205$^{***}$ & 0.200$^{***}$ WW
  & (0.004) & (0.003) WW
 renovationCondition[T.refined] & -0.113$^{***}$ & -0.102$^{***}$ WW
  & (0.003) & (0.004) WW
 renovationCondition[T.rough] & -0.260$^{***}$ & -0.207$^{***}$ WW
  & (0.009) & (0.010) WW
 renovationCondition[T.simple] & -0.217$^{***}$ & -0.204$^{***}$ WW
  & (0.004) & (0.004) WW
 buildingType[T.plate] & -1.712$^{***}$ & -1.488$^{***}$ WW
  & (0.061) & (0.056) WW
 buildingType[T.plate_tower] & -1.773$^{***}$ & -1.547$^{***}$ WW
  & (0.061) & (0.056) WW
 buildingType[T.tower] & -1.897$^{***}$ & -1.664$^{***}$ WW
  & (0.061) & (0.056) WW
 cpi & 0.145$^{***}$ & 0.029$^{***}$ WW
  & (0.002) & (0.003) WW
 year & 0.678$^{***}$ & 0.631$^{***}$ WW
  & (0.002) & (0.013) WW
 fiveYearsProperty & -0.094$^{***}$ & -0.029$^{***}$ WW
  & (0.002) & (0.002) WW
 buildingSold & -0.019$^{***}$ & 0.000$^{}$ WW
  & (0.001) & (0.001) WW
 district[T.Chaoyang] & 0.631$^{***}$ & 0.630$^{***}$ WW
  & (0.004) & (0.004) WW
 district[T.Daxing] & 0.052$^{***}$ & 0.042$^{***}$ WW
  & (0.006) & (0.005) WW
 district[T.Dongcheng] & 1.436$^{***}$ & 1.447$^{***}$ WW
  & (0.006) & (0.005) WW
 district[T.Fangshan] & -0.500$^{***}$ & -0.486$^{***}$ WW
  & (0.012) & (0.011) WW
 district[T.Fengtai] & 0.432$^{***}$ & 0.424$^{***}$ WW
  & (0.005) & (0.004) WW
 district[T.Haidian] & 1.202$^{***}$ & 1.168$^{***}$ WW
  & (0.005) & (0.004) WW
 district[T.Mentougou] & -0.129$^{***}$ & -0.067$^{***}$ WW
  & (0.016) & (0.015) WW
 district[T.Pinggu] & 0.046$^{***}$ & 0.017$^{***}$ WW
  & (0.006) & (0.006) WW
 district[T.Shijingshan] & 0.293$^{***}$ & 0.296$^{***}$ WW
  & (0.007) & (0.006) WW
 district[T.Shunyi] & -0.149$^{***}$ & -0.160$^{***}$ WW
  & (0.007) & (0.006) WW
 district[T.Tongzhou] & -0.009$^{}$ & 0.003$^{}$ WW
  & (0.013) & (0.012) WW
 district[T.Xicheng] & 1.754$^{***}$ & 1.756$^{***}$ WW
  & (0.005) & (0.007) WW
 cpi:district[T.Xicheng] & & 0.068$^{***}$ WW
  & & (0.005) WW
```

```
 age:district[T.Haidian] & & 0.232$^{***}$ WW
  & & (0.003) WW
 age:year & & 0.009$^{***}$ WW
  & & (0.001) WW
 age:totalRoom & & -0.070$^{***}$ WW
  & & (0.001) WW
 year_2 & & 0.111$^{***}$ WW
  & & (0.002) WW
 year:buildingSold & & -0.028$^{***}$ WW
  & & (0.001) WW
 year:district[T.Dongcheng] & & 0.367$^{***}$ WW
  & & (0.005) WW
 year:district[T.Haidian] & & 0.171$^{***}$ WW
  & & (0.003) WW
 year:district[T.Xicheng] & & 0.555$^{***}$ WW
  & & (0.005) WW
 year:district[T.Changping] & & -0.119$^{***}$ WW
  & & (0.003) WW
 renovationCondition[T.simple]:year & & 0.042$^{***}$ WW
  & & (0.014) WW
 renovationCondition[T.refined]:year & & 0.090$^{***}$ WW
  & & (0.014) WW
 renovationCondition[T.refined]:cpi & & -0.011$^{**}$ WW
  & & (0.004) WW
 renovationCondition[T.other]:cpi & & 0.027$^{***}$ WW
  & & (0.004) WW
 renovationCondition[T.other]:year & & -0.087$^{***}$ WW
  & & (0.014) WW
 renovationCondition[T.other]:district[T.Xicheng] & & 0.086$^{***}$ WW
  & & (0.011) WW
 renovationCondition[T.simple]:district[T.Xicheng] & & 0.042$^{***}$ WW
  & & (0.008) WW
Whline WW[-1.8ex]
 Observations & 297,225 & 297,225 WW
 $R^2$ & 0.656 & 0.714 WW
 Adjusted $R^2$ & 0.656 & 0.714 WW
 Residual Std. Error & 0.586(df = 297199) & 0.535(df = 297182)  WW
 F Statistic & 22691.533$^{***}$ (df = 25.0; 297199.0) & 17691.471$^{***}$ (df = 42.
0; 297182.0) WW
Whline
Whline WW[-1.8ex]
Wtextit{Note:} & Wmulticolumn{2}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.0
1} WW
Wend{tabular}
Wend{table}
```

Predictive Analysis

In [51]:
```
# xgBoost may not contain [, ] or <
# Replace the characters '[T.' with '_' and ']' with '' in the column names

X.columns = X.columns.str.replace('W[TW.', '_', regex=True).str.replace('W]', '', 
x_in.columns = x_in.columns.str.replace('W[TW.', '_', regex=True).str.replace('W]'
x_out.columns = x_out.columns.str.replace('W[TW.', '_', regex=True).str.replace('W
```

In [53]:
```
lr = LinearRegression()
lasso = Lasso(alpha = 0.001, random_state=375)
dt = DecisionTreeRegressor(random_state=375)
rf = RandomForestRegressor(random_state=375)
xgboost = xgb.XGBRegressor(random_state = 375)
```

```
classifiers = {
    'Linear_Regression' : lr,
    'Lasso_Regression' : lasso,
    'Decision_Tree' : dt,
    'Random_Forest': rf,
    'XGBoost_Regressor': xgboost,
    }
```

Split dataset

In [96]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2, random_state=3

X_train_in, X_test_in, y_train_in, y_test_in = train_test_split(x_in, y, test_size=0
X_train_out, X_test_out, y_train_out, y_test_out = train_test_split(x_out, y, test_s
```

Comparison of Five Models

In [97]:
```
def compare(classifier):
    classifier_name = (list(classifiers.keys())[list(classifiers.values()).index(cla

    classifier.fit(X_train_in, y_train_in)
    ypred_in = classifier.predict(X_test_in)
    r2_in = round(r2_score(y_test_in, ypred_in),5)
    rmse_in = round(mean_squared_error(y_test_in, ypred_in, squared = True),5)


    classifier.fit(X_train_out, y_train_out)
    ypred_out = classifier.predict(X_test_out)
    r2_out = round(r2_score(y_test_out, ypred_out),5)
    rmse_out = round(mean_squared_error(y_test_out, ypred_out, squared = True),5)

    classifier.fit(X_train, y_train)
    ypred = classifier.predict(X_test)
    r2_all = round(r2_score(y_test, ypred),5)
    rmse_all = round(mean_squared_error(y_test, ypred, squared = True),5)

    test_results = [['Inner Features', r2_in, rmse_in],['Outter Features', r2_out, r
    test_results_pd = pd.DataFrame(test_results, columns = [classifier_name, 'R_squa
    return test_results_pd
```

In [98]:
```
compare(lr)
```

Out[98]:

	Linear_Regression	R_squared	RMSE
0	Inner Features	0.21540	0.77929
1	Outter Features	0.64344	0.35415
2	All Features	0.65647	0.34120

In [99]:
```
compare(lasso)
```

Out[99]:

	Lasso_Regression	R_squared	RMSE
0	Inner Features	0.21389	0.78078
1	Outter Features	0.64296	0.35462
2	All Features	0.65489	0.34277

In [100…
```
compare(dt)
```

Out[100]:

	Decision_Tree	R_squared	RMSE
0	Inner Features	0.23615	0.75867
1	Outter Features	0.72058	0.27752
2	All Features	0.61840	0.37901

In [101…
```
compare(rf)
```

Out[101]:

	Random_Forest	R_squared	RMSE
0	Inner Features	0.24152	0.75334
1	Outter Features	0.73003	0.26814
2	All Features	0.79019	0.20839

In [102…
```
compare(xgboost)
```

Out[102]:

	XGBoost_Regressor	R_squared	RMSE
0	Inner Features	0.25180	0.74313
1	Outter Features	0.75492	0.24342
2	All Features	0.80278	0.19589

Interaction/Ploynominal Terms For XGBoost

In [54]:
```
interaction = PolynomialFeatures(degree=2, include_bias=False, interaction_only=Fa
poly_features = interaction.fit_transform(X)
print("The number of features: ", len(interaction.get_feature_names_out()))

X_inter_3 = pd.DataFrame(interaction.fit_transform(X), columns=interaction.get_featu
X_train, X_test, y_train, y_test = train_test_split(X_inter_3,y, test_size=0.2, rand
```

```
The number of features:  434
```

In [104…
```
xgboost.fit(X_train, y_train)
ypred = xgboost.predict(X_test)
r2 = round(r2_score(y_test, ypred),5)
rmse = round(mean_squared_error(y_test, ypred, squared = True),5)

r2_result = [r2]
rmse_result = [rmse]

dic = {'R2':r2_result,'RMSE':rmse_result}
test_results_pd = pd.DataFrame(dic, columns=['R2','RMSE'])
test_results_pd
```

Out[104]:

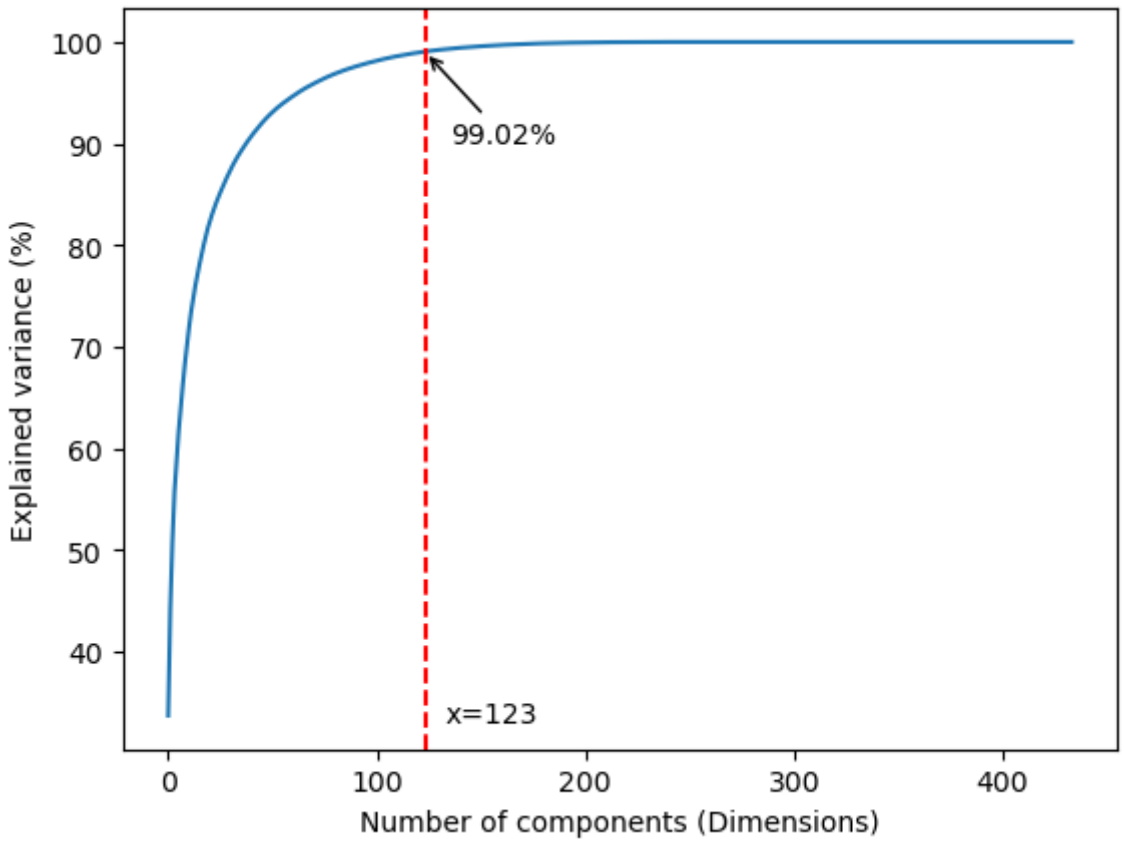
	R2	RMSE
0	0.80394	0.19473

Feature Selection

PCA

In [61]:
```
pca = PCA(n_components=None, random_state=375)
pca.fit(X_train)

# Get the eigenvalues
# print("Eigenvalues:")
# print(pca.explained_variance_)
# print()

# Get explained variances
# print("Variances (Percentage):")
# print(pca.explained_variance_ratio_ * 100)
# print()

# Make the scree plot

plt.plot(np.cumsum(pca.explained_variance_ratio_ * 100))
plt.xlabel("Number of components (Dimensions)")
plt.ylabel("Explained variance (%)")
plt.axvline(x=123, color='r', linestyle='--')
plt.text(133, 33, 'x=123', fontsize=10)

plt.annotate(str(round(np.cumsum(pca.explained_variance_ratio_ * 100)[122], 2)) + "
```

Out[61]:
```
Text(135, 90, '99.02%')
```



In [60]:
```
pca_99 = PCA(n_components=0.99, random_state=375)
X_train_pca = pca_99.fit_transform(X_train)
X_test_pca = pca_99.transform(X_test)

print('The number of selected components' ,pca_99.n_components_)

xgboost.fit(X_train_pca, y_train)

y_pred = xgboost.predict(X_test_pca)

r2 = round(r2_score(y_test, y_pred),5)
rmse = round(mean_squared_error(y_test, y_pred, squared = True),5)

r2_result = [r2]
rmse_result = [rmse]

dic = {'R2':r2_result,'RMSE':rmse_result}
test_results_pd = pd.DataFrame(dic, columns=['R2','RMSE'])
test_results_pd
```

```
The number of selected components 123
```

Out[60]:

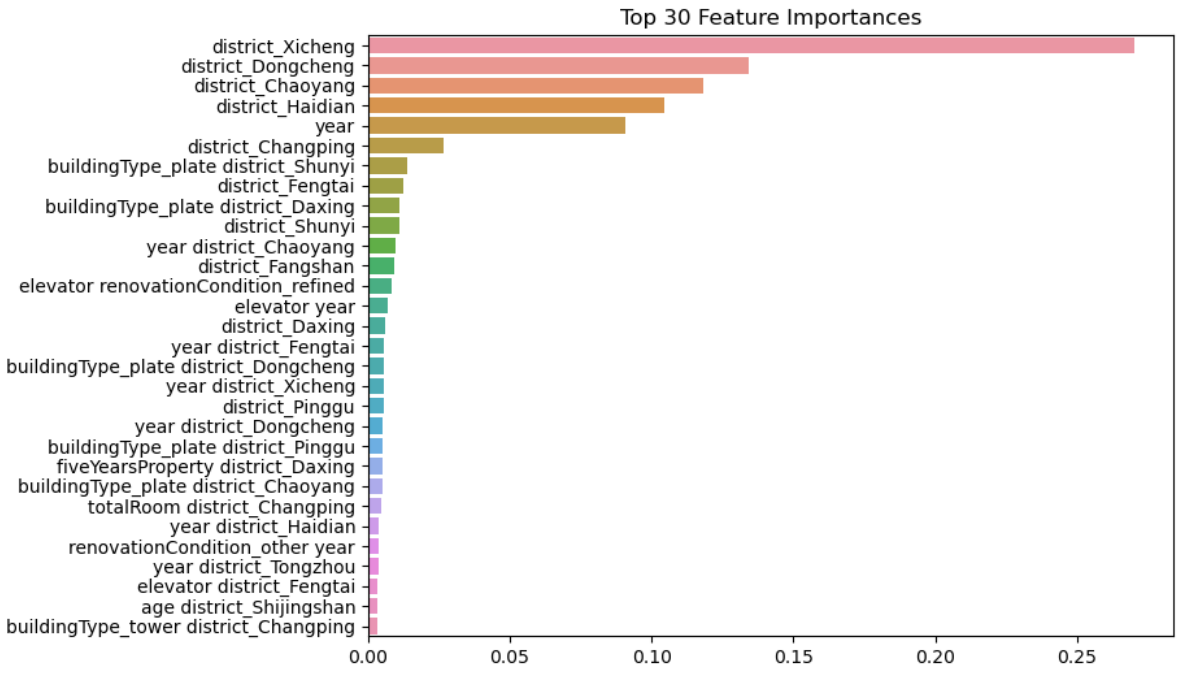
	R2	RMSE
0	0.76149	0.23689

Feature Importance

In [55]:
```
xgboost.fit(X_train, y_train)
```

Out[55]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=375, ...)
```

In [56]:
```
%matplotlib inline

importances_values = xgboost.feature_importances_
importances = pd.Series(importances_values, index = X_train.columns)
top30 = importances.sort_values(ascending=False)[:30]

plt.figure(figsize=(8,6))
plt.title('Top 30 Feature Importances')
sns.barplot(x=top30, y=top30.index)
plt.show()
```



In [57]:
```
k = list(range(5,434,5))
length = len(k)

r2_result = []
rmse_result = []

for i in range(length):
    print("k: %d" % (k[i]), end="Wt")
    top = importances.sort_values(ascending=False)[:k[i]]
    feature_names= (top.index)
    X_train_best = X_train[feature_names]
    X_test_best = X_test[feature_names]

    xgboost.fit(X_train_best, y_train)
    y_pred = xgboost.predict(X_test_best)

    r2 = round(r2_score(y_test, y_pred),5)
    rmse = round(mean_squared_error(y_test, y_pred, squared = True),5)
```

```
    r2_result.append(r2)
    rmse_result.append(rmse)
    print("r2: %.5f,₩trmse: %.5f" % (r2, rmse))
```

```
k: 5    r2: 0.69796,    rmse: 0.29999
k: 10   r2: 0.71307,    rmse: 0.28499
k: 15   r2: 0.72101,    rmse: 0.27710
k: 20   r2: 0.72310,    rmse: 0.27503
k: 25   r2: 0.72734,    rmse: 0.27081
k: 30   r2: 0.73105,    rmse: 0.26713
k: 35   r2: 0.76098,    rmse: 0.23740
k: 40   r2: 0.76104,    rmse: 0.23734
k: 45   r2: 0.77474,    rmse: 0.22373
k: 50   r2: 0.79357,    rmse: 0.20503
k: 55   r2: 0.79403,    rmse: 0.20457
k: 60   r2: 0.79651,    rmse: 0.20211
k: 65   r2: 0.79639,    rmse: 0.20223
k: 70   r2: 0.79673,    rmse: 0.20189
k: 75   r2: 0.79560,    rmse: 0.20302
k: 80   r2: 0.80344,    rmse: 0.19523
k: 85   r2: 0.80526,    rmse: 0.19342
k: 90   r2: 0.80580,    rmse: 0.19289
k: 95   r2: 0.80651,    rmse: 0.19217
k: 100  r2: 0.80634,    rmse: 0.19235
k: 105  r2: 0.80606,    rmse: 0.19263
k: 110  r2: 0.80647,    rmse: 0.19222
k: 115  r2: 0.80600,    rmse: 0.19268
k: 120  r2: 0.80614,    rmse: 0.19255
k: 125  r2: 0.80481,    rmse: 0.19387
k: 130  r2: 0.80510,    rmse: 0.19358
k: 135  r2: 0.80596,    rmse: 0.19273
k: 140  r2: 0.80580,    rmse: 0.19288
k: 145  r2: 0.80552,    rmse: 0.19316
k: 150  r2: 0.80471,    rmse: 0.19397
k: 155  r2: 0.80478,    rmse: 0.19390
k: 160  r2: 0.80494,    rmse: 0.19374
k: 165  r2: 0.80531,    rmse: 0.19337
k: 170  r2: 0.80469,    rmse: 0.19398
k: 175  r2: 0.80527,    rmse: 0.19341
k: 180  r2: 0.80431,    rmse: 0.19437
k: 185  r2: 0.80516,    rmse: 0.19352
k: 190  r2: 0.80425,    rmse: 0.19443
k: 195  r2: 0.80487,    rmse: 0.19381
k: 200  r2: 0.80391,    rmse: 0.19476
k: 205  r2: 0.80352,    rmse: 0.19515
k: 210  r2: 0.80437,    rmse: 0.19430
k: 215  r2: 0.80406,    rmse: 0.19462
k: 220  r2: 0.80405,    rmse: 0.19463
k: 225  r2: 0.80395,    rmse: 0.19473
k: 230  r2: 0.80395,    rmse: 0.19473
k: 235  r2: 0.80395,    rmse: 0.19473
k: 240  r2: 0.80395,    rmse: 0.19473
k: 245  r2: 0.80395,    rmse: 0.19473
k: 250  r2: 0.80395,    rmse: 0.19473
k: 255  r2: 0.80395,    rmse: 0.19473
k: 260  r2: 0.80395,    rmse: 0.19473
k: 265  r2: 0.80395,    rmse: 0.19473
k: 270  r2: 0.80395,    rmse: 0.19473
k: 275  r2: 0.80395,    rmse: 0.19473
k: 280  r2: 0.80395,    rmse: 0.19473
k: 285  r2: 0.80395,    rmse: 0.19473
k: 290  r2: 0.80395,    rmse: 0.19473
k: 295  r2: 0.80395,    rmse: 0.19473
k: 300  r2: 0.80395,    rmse: 0.19473
k: 305  r2: 0.80395,    rmse: 0.19473
k: 310  r2: 0.80395,    rmse: 0.19473
k: 315  r2: 0.80395,    rmse: 0.19473
k: 320  r2: 0.80395,    rmse: 0.19473
```

```
k: 325  r2: 0.80395,    rmse: 0.19473
k: 330  r2: 0.80395,    rmse: 0.19473
k: 335  r2: 0.80395,    rmse: 0.19473
k: 340  r2: 0.80395,    rmse: 0.19473
k: 345  r2: 0.80395,    rmse: 0.19473
k: 350  r2: 0.80395,    rmse: 0.19473
k: 355  r2: 0.80395,    rmse: 0.19473
k: 360  r2: 0.80395,    rmse: 0.19473
k: 365  r2: 0.80395,    rmse: 0.19473
k: 370  r2: 0.80395,    rmse: 0.19473
k: 375  r2: 0.80395,    rmse: 0.19473
k: 380  r2: 0.80395,    rmse: 0.19473
k: 385  r2: 0.80395,    rmse: 0.19473
k: 390  r2: 0.80395,    rmse: 0.19473
k: 395  r2: 0.80395,    rmse: 0.19473
k: 400  r2: 0.80395,    rmse: 0.19473
k: 405  r2: 0.80395,    rmse: 0.19473
k: 410  r2: 0.80395,    rmse: 0.19473
k: 415  r2: 0.80395,    rmse: 0.19473
k: 420  r2: 0.80395,    rmse: 0.19473
k: 425  r2: 0.80395,    rmse: 0.19473
k: 430  r2: 0.80395,    rmse: 0.19473
```

In [58]:
```
dic = {'K':k, 'R2':r2_result,'RMSE':rmse_result}
test_results_pd = pd.DataFrame(dic, columns=['K','R2','RMSE'])
test_results_pd
```

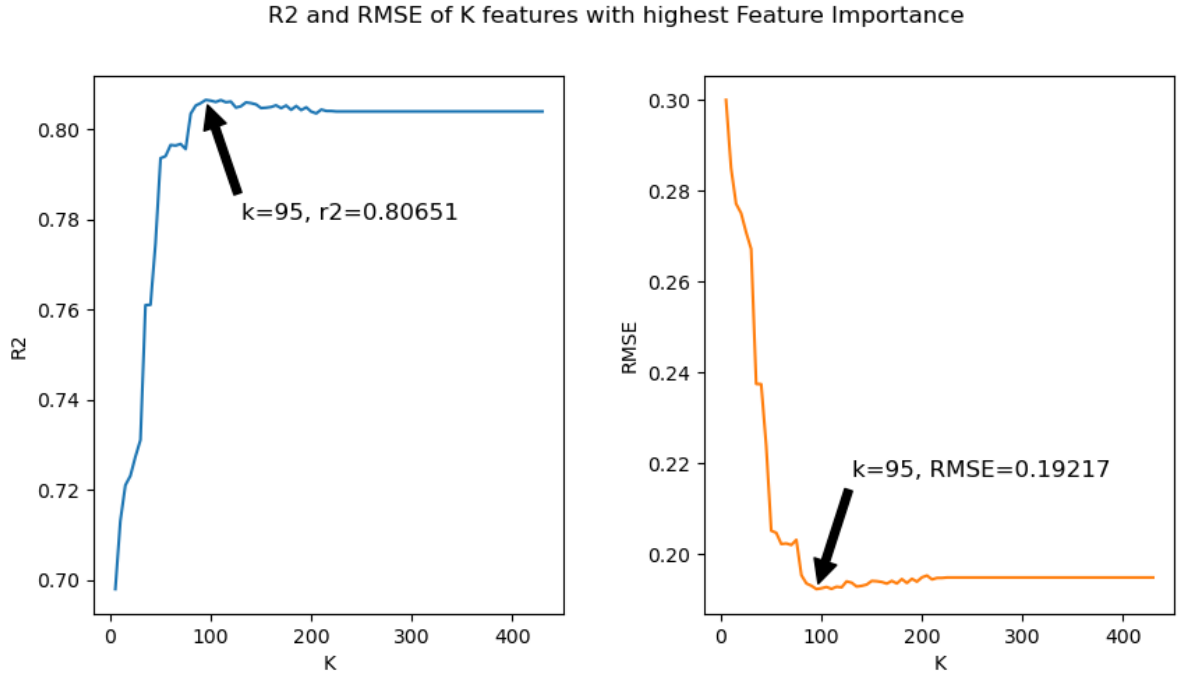
Out[58]:

	K	R2	RMSE
0	5	0.69796	0.29999
1	10	0.71307	0.28499
2	15	0.72101	0.27710
3	20	0.72310	0.27503
4	25	0.72734	0.27081
...	...	...	...
81	410	0.80395	0.19473
82	415	0.80395	0.19473
83	420	0.80395	0.19473
84	425	0.80395	0.19473
85	430	0.80395	0.19473

86 rows × 3 columns

In [59]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))
fig.suptitle('R2 and RMSE of K features with highest Feature Importance')

ax1.plot(k, r2_result, label = "R2")
ax1.annotate('k=95, r2=0.80651', xy=(95, 0.80651), xytext=(130, 0.780),
             arrowprops=dict(facecolor='black', shrink=0.05), fontsize = 12)
ax1.set_xlabel('K')
ax1.set_ylabel('R2')

ax2.plot(k, rmse_result,'tab:orange', label = "RMSE")
ax2.annotate('k=95, RMSE=0.19217', xy=(95, 0.19217), xytext=(130, 0.217),
             arrowprops=dict(facecolor='black', shrink=0.05), fontsize = 12)
```

```
ax2.set_xlabel('K')
ax2.set_ylabel('RMSE')

fig.subplots_adjust(wspace=0.3)
plt.show()
```



Hyperparameter Tunning

In [147…
```
xgboost.fit(X_train, y_train)
importances_values = xgboost.feature_importances_
importances = pd.Series(importances_values, index = X_train.columns)

top = importances.sort_values(ascending=False)[:95]
feature_names= (top.index)
X_train_best = X_train[feature_names]
X_test_best = X_test[feature_names]
```

In [148…
```
param_grid = {'n_estimators': [100, 200, 300, 400],
              'max_depth': [6, 7, 9],
              'learning_rate' :[0.0001, 0.001, 0.01, 0.1]
             }

xgboost_search = xgb.XGBRegressor(random_state = 375)
xgboost_search.fit(X_train_best, y_train)

grid = GridSearchCV(xgboost_search, param_grid)
grid_result = grid.fit(X_train_best, y_train)

best_params =  grid_result.best_params_
print(best_params)

#using best params to create and fit model
ypred = grid.predict(X_test_best)

#evaluate metrics
r2 = round(r2_score(y_test, ypred),5)
rmse = round(mean_squared_error(y_test, ypred, squared = True),5)

r2_result = [r2]
```

```
rmse_result = [rmse]

dic = {'R2':r2_result,'RMSE':rmse_result}
test_results_pd = pd.DataFrame(dic, columns=['R2','RMSE'])
test_results_pd
```

```
{'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 400}
```

Out[148]:

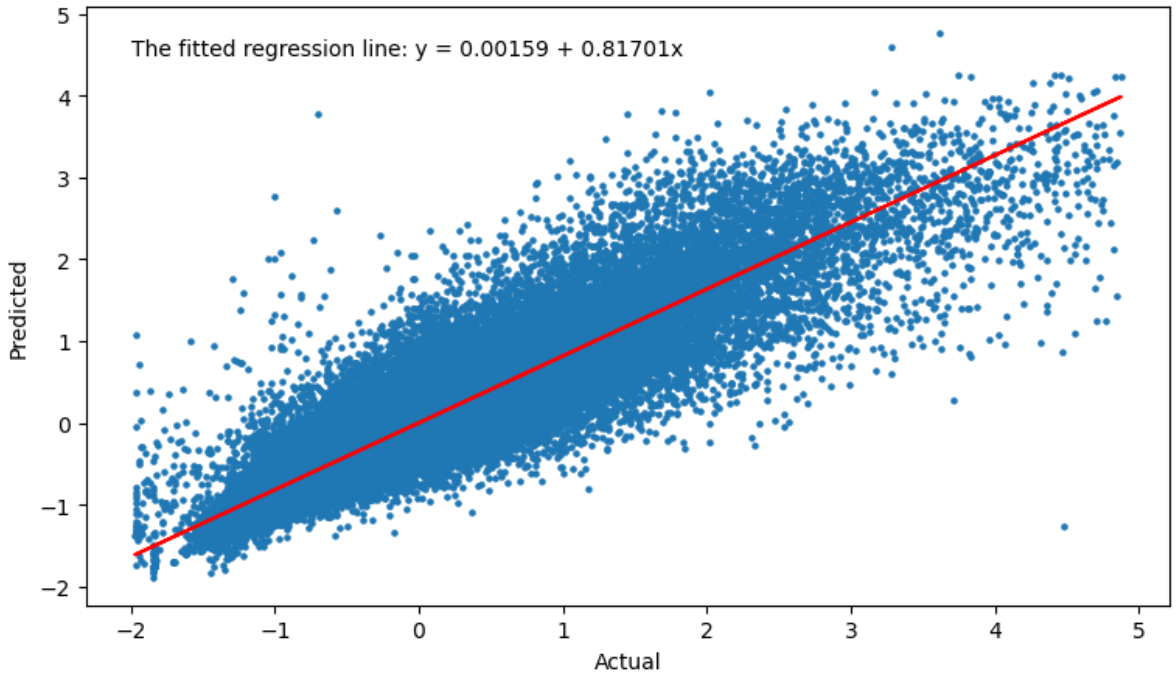
	R2	RMSE
0	0.81368	0.18506

In [149…

```
ypred_df = pd.DataFrame(ypred, columns = ['Predicted'])

y_test_df = y_test.reset_index()
y_test_df = y_test_df.drop(['index'], axis=1)

compare_df = pd.concat([ypred_df, y_test_df], axis=1)
```

In [150…

```
#find line of best fit
a, b = np.polyfit(compare_df['price'], compare_df['Predicted'], 1)

plt.figure(figsize=(9,5))

#add points to plot
plt.scatter(compare_df['price'], compare_df['Predicted'], s=5)

#add line of best fit to plot
plt.plot(compare_df['price'], a*compare_df['price']+b, color='red')

plt.text(-2, 4.5, 'The fitted regression line: ' + 'y = ' + '{:.5f}'.format(b) + ' +

plt.xlabel("Actual")
plt.ylabel("Predicted")

plt.show()
```



In [151…

```
plt.figure(figsize=(15,5))
plt.plot(compare_df['price'].head(300), label='Actual')
plt.plot(compare_df['Predicted'].head(300), label='Predicted')
```

```
plt.legend()
plt.show()
```