When Less is More: Evaluating Simplicity vs. Complexity in House Price Prediction

Adam Shaw Michael Kim Anish Kushalapa
University of Southern California
{adshaw,mkim2763,kushalap}@usc.edu

Abstract

Predicting house prices accurately is essential for buyers and sellers alike. This research explores the effectiveness of various machine learning algorithms—including multiple linear regression, support vector regression, ridge and lasso regularization, and XGBoost—in predicting house prices based on property features. Our analysis utilized a dataset from Kaggle, which was made up of 545 houses from the Boston metropolitan area, each described by 13 unique features. To address the high feature-to-sample-size ratio and reduce model variability observed in initial trials, we relied on a 5-fold cross-validation throughout the study, providing a more consistent and reliable estimate of model performance. Despite experimenting with complex models, our results show that standard linear regression demonstrates comparable performance with an R^2 of approximately 0.68, making it a robust and interpretable choice for this dataset.

1. Introduction

Accurately predicting house prices using property features can provide crucial information to homebuyers, real estate agents, and policymakers. In this paper, we aim to extract valuable information from our dataset to create a model that accurately predicts price.

To train our model, we used the Housing Prices Dataset from Kaggle [3], which consisted of 545 houses, each with 13 features, which are laid out in Table 1. In our early planning phase, we ruled out some models due to their incompatibility with our dataset. For example, we considered using neural networks; however, due to the limited sample size, we concluded that it would not be effective. Clustering algorithms such as k-nearest neighbors were also ruled out, as the dataset's high proportion of categorical features could compromise performance. After eliminating these models, we narrowed it down to approaches better suited to the characteristics of our data.

2. Data Preprocessing

We began by converting qualitative variables into a quantitative format—such as converting "yes"/"no" into binary values, and the `furnishingstatus` feature (which consisted of "unfurnished", "semi-furnished", and "furnished") into 0, 1, and 2, respectively—to ensure that they were compatible with our model.

In order to understand what models to attempt, we decided to view the underlying relationships between the features. Since `area` was the only continuous variable along with `price`, we initially explored the `price`–`area` relationship through a scatterplot (Figure 1).

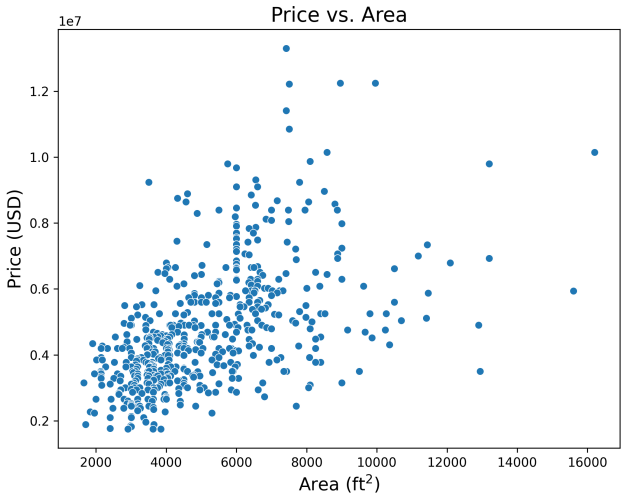

Figure 1. Scatterplot of `price` vs. `area`

The original data appears to have a large concentration in the lower left quadrant of the scatterplot (Figure 1). To linearize the relationship between `price` and `area`, we transformed both of these features using logarithms, creating `log_price` and `log_area`, respectively. Post-transformation, the data demonstrated a clear linear trend, as shown in Figure 2.

Before attempting linear regression, we normalized the

Feature Name	Data Type	Description
price	int	Price of the house (USD)
area	int	Area of the house (ft^2)
bedrooms	int	Number of bedrooms
bathrooms	int	Number of bathrooms
stories	int	Number of stories
mainroad	bool	Whether the house is connected to the main road (“yes”/“no”)
guestroom	bool	Whether the house has a guest room (“yes”/“no”)
basement	bool	Whether the house has a basement (“yes”/“no”)
hotwaterheating	bool	Whether the house has hot water heating (“yes”/“no”)
airconditioning	bool	Whether the house has air conditioning (“yes”/“no”)
parking	int	Number of parking spaces
prefarea	bool	Is the house in a preferred area (“yes”/“no”)
furnishingstatus	string	Furnishing status of the house (“furnished”, “semi-furnished”, “unfurnished”)

Table 1. Features of the Housing Prices Dataset

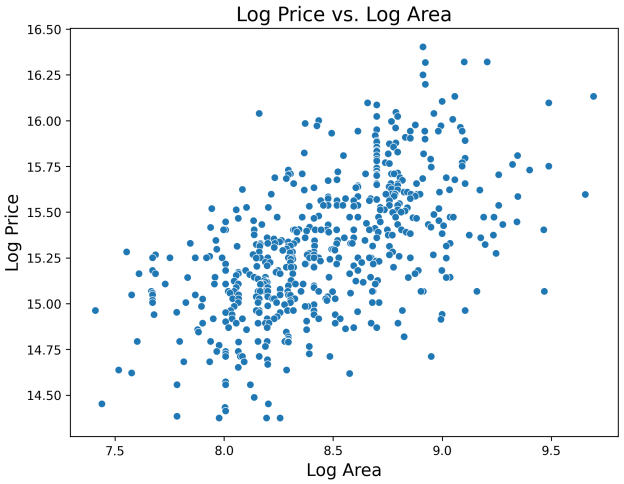

Figure 2. log_price and log_area have a clear linear relationship.

features using standard scaling (StandardScaler from the scikit-learn library [6]), ensuring that they all have a mean of 0 and a standard deviation of 1. This helps mitigate bias by ensuring that a variable’s magnitude doesn’t disproportionately influence the model. After scaling and fitting our data, we utilized a 5-fold cross-validation from the KFold class to evaluate the model’s performance. We used cross-validation on every attempt to evaluate performance due to our high feature-to-sample-size ratio. In the initial stages of this project, we realized that there was high variability in our model’s performance, and changing the random_state of our train_test_split resulted in significant changes in the model’s performance (*e.g.*, when performing linear regression, random_state=42 resulted in an R^2 of 0.62, and random_state=17 resulted in an R^2 of 0.83). The cross-validation method of training on four folds and validating on the fifth (and repeating four more times) provided a more robust and unbiased estimate of accuracy, reducing dependence on any single train-test split.

3. Multicollinearity

3.1. Correlation Matrix

The next step we took to better understand our data was seeing how the features were correlated. We suspected high degrees of multicollinearity in our data; for example, bedrooms and bathrooms are likely correlated, as the more people you have living in a house, the more bathrooms you need. We constructed a heatmap to visualize the correlation between the features (Figure 3) and included a table of the top-ten most-correlated features for easy reference (Table 2).

Feature 1	Feature 2	Correlation ($\lvert \cdot \rvert$)
bedrooms	stories	0.41
bedrooms	bathrooms	0.37
guestroom	basement	0.37
parking	log_area	0.36
mainroad	log_area	0.33
bathrooms	stories	0.33
stories	airconditioning	0.29
airconditioning	log_area	0.26
basement	prefarea	0.23
prefarea	log_area	0.22

Table 2. Top-ten most-correlated features (absolute value)

As suspected, the bedrooms and bathrooms features are correlated, but not as strongly as we assumed. Interestingly, bedrooms and stories have the highest corre-

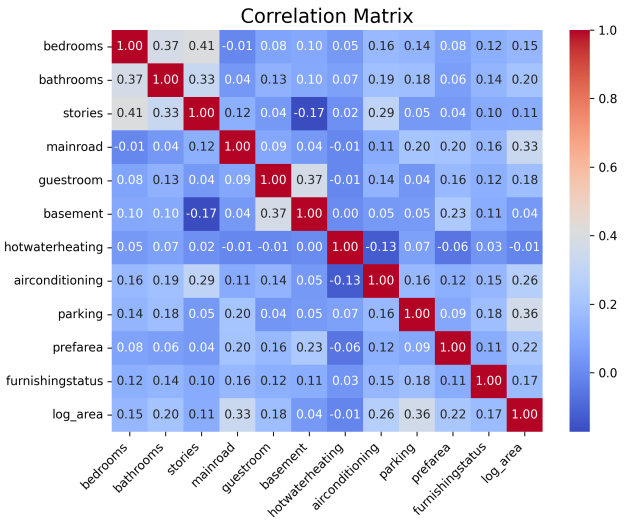

Figure 3. Feature correlation heatmap

lation, a result we didn't anticipate, but ultimately makes sense: most homes are either one or two stories, and houses with two stories are likely to have more bedrooms on average than those with one story. One drawback to this approach is that this matrix only compares features pairwise—it could be the case that features are dependent on a combination of many other features, which this heatmap does not capture. To address this possible shortcoming, we move on to the Variance Inflation Factor.

3.2. Variance Inflation Factor (VIF)

Variance Inflation Factor (VIF) [4] is a metric used to detect multicollinearity in a regression model. It is a metric of a single feature, representing how strongly that feature is correlated with the other features in the data. It is calculated as follows:

Variance Inflation Factor Calculation

Step 1: Regress X_i on all other predictors.
Run an ordinary least squares regression where X_i is the response variable and all other X variables are predictors. For example, if $i = 1$:

$$X_1 = \alpha_0 + \alpha_2 X_2 + \alpha_3 X_3 + \cdots + \alpha_k X_k + \varepsilon$$

Step 2: Compute the Variance Inflation Factor.
Use the formula:

$$\text{VIF}_i = \frac{1}{1 - R_i^2} \tag{1}$$

where R_i^2 is the coefficient of determination from the regression in Step 1.

A VIF of 1 means there is no correlation with other features, while a VIF greater than 1 suggests some degree of correlation. Generally, a VIF exceeding 5 or 10 (depending on the context) is considered a sign of high multicollinearity [1].

The motivation to study VIF was inspired by prior research that applied VIF to similar projects, such as the work by Jully Ye on factors affecting house prices using a multiple linear regression model [8]. In her research, Ye employed VIF to analyze the collinearities between the features in her own house price dataset. We calculated the VIFs of our data using the `variance_inflation_factor` function from the `statsmodels` library [7], the results of which can be found in Table 3.

Feature	VIF
log_area	27.47
bedrooms	23.37
bathrooms	9.71
mainroad	8.31
stories	7.85
furnishingstatus	2.73
basement	2.02
parking	1.85
airconditioning	1.73
guestroom	1.46
prefarea	1.46
hotwaterheating	1.09

Table 3. Variance Inflation Factors (VIF) for Each Feature

The `log_area` feature had the highest VIF at 27.47, signaling its strong correlation with other features. The `bedrooms` feature also had a very high VIF at 23.37. One possible explanation for this is that the number of bedrooms and bathrooms in a house likely grow at a similar rate—a suspicion confirmed by the correlation matrix (Figure 3, Table 2).

We initially attempted to mitigate the effects of multicollinearity by outright removing the `log_area` feature, as it had the highest VIF. Theoretically, if `log_area` is just a linear combination of some set of other features, then the linear regression model should do just fine without it. The model without the `log_area` feature had an R^2 of 0.606, a good amount lower than that of the standard linear regression model ($R^2 = 0.681$), which is seen in section 4.1. Removing both `log_area` and `bedrooms` had a similar effect ($R^2 = 0.602$), but removing only `bedrooms` did not ($R^2 = 0.681$). Although this did not improve our model, the fact that it stayed the same likely implies that the feature is not that important to the model. However, we kept the feature in as it may have been helpful later.

4. Model Selection

4.1. Linear Regression

For our first hypothesis, we attempted multiple linear regression by fitting all variables, omitting `log_price` to be used as the target variable. To evaluate the data, we reversed the standardization using the `inverse_transform` function, which brought our data back into units of USD. However, when we evaluated our model using mean squared error (MSE), we were surprised to get values on the order of 10^9. This initially seemed very high, but we found a good explanation: an absolute error in predicted house price of only $100,000 would lead to an squared error of 10 billion. Therefore, to increase interpretability, we decided to evaluate model performance using normalized root mean squared error (RMSE) instead. This was done by dividing the absolute RMSE by the mean of the target variable (`log_price`). Our linear regression model produced an R^2 score of 0.681 and a scaled RMSE of 0.210, establishing a strong baseline.

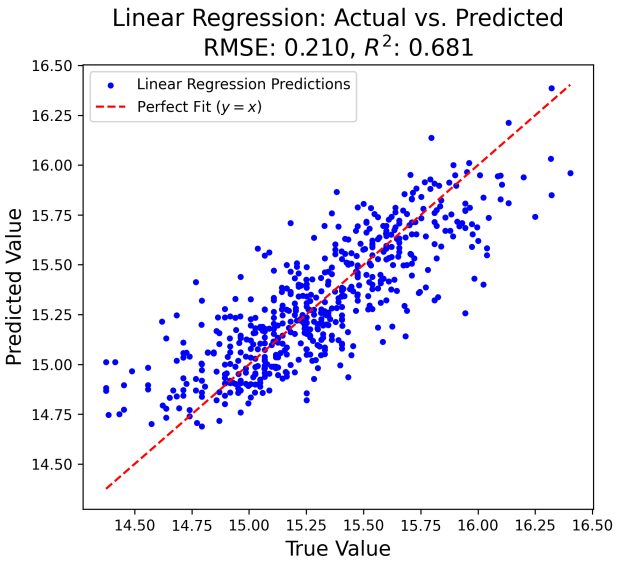

Figure 4. Our baseline linear regression model had strong performance.

4.2. Support Vector Regression (SVR)

As an alternative to linear regression, we also tried applying support vector regression (SVR). SVR requires careful tuning of the C, ε, and `kernel` parameters, which we addressed using grid search via `GridSearchCV` along with our previously defined 5-fold cross-validation. After finding the optimal parameters, we evaluated the model, which had an RMSE of 0.209 and an R^2 of 0.684, slightly outperforming the linear regression model.

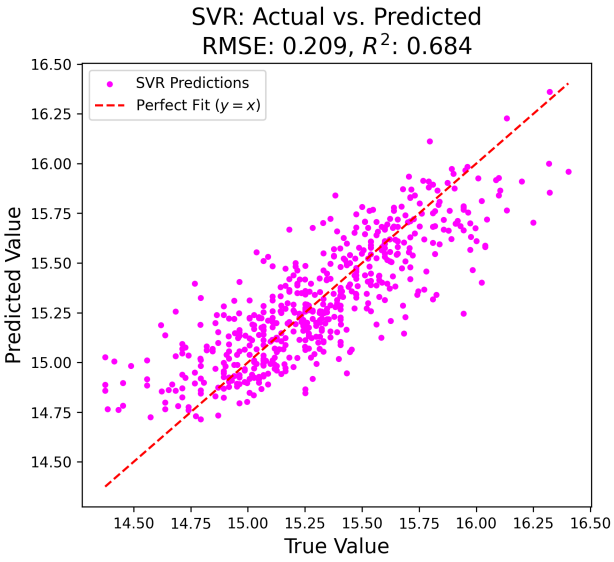

Figure 5. Optimized SVR model performance

5. Feature Importance

To further inform our understanding of what influences the price of homes, we investigated the importance of certain features. Not only would this give us insight into which features were the most important in predicting house prices, but it could also help with feature selection and general tuning of our models later on.

5.1. Random Forest

To determine the most influential features in our model, we used a random forest, an ensemble method that combines many decision trees. The model computes feature importance by averaging the information gained across all decision trees. By setting `log_price` as the target y variable and the rest of the features as the x, we fit a random forest using `RandomForestRegressor`. Through this, we obtained a ranking of the features based on their importance to `log_price` (Figure 6).

As one might expect, `log_area` was the most important feature. Surprisingly, features such as the number of bedrooms (`bedrooms`) or whether the house was in a preferred area (`prefarea`) were ranked lower. We suspected the low ranking for `bedrooms` as possibly due to collinearity among multiple features (*e.g.*, `log_area`, `bathrooms`, and `bedrooms`). What may have happened is, since the features are highly correlated, the decision tree may have chosen to split on either one arbitrarily, distributing the importance between them.

We also used random forest to perform regression as a comparison to linear regression. Using this forest, we achieved an RMSE of 0.223 and an R^2 of 0.641. This was

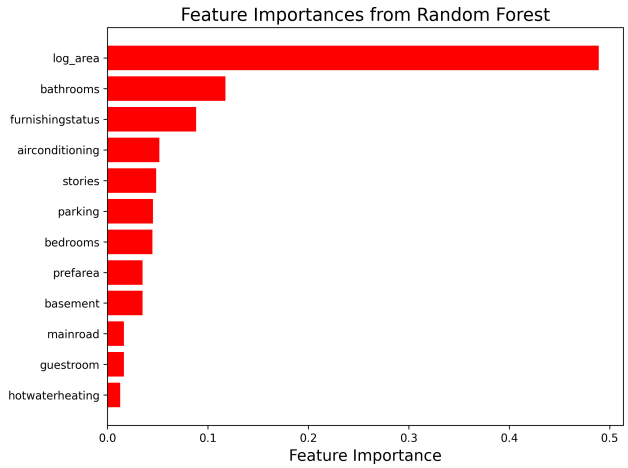

Figure 6. Feature importances extracted from the random forest model

the worst of the models so far and the worst performing model we tried (all of which are shown in Figure 9). We suspect that Random Forest was overfitting to certain features, resulting in higher error. While we could have spent time refining this model more with techniques such as changing the tree depth, we use XGBoost as an alternative in section 6.4.

6. Regularization

Regularization adds a penalty term to the model's loss function that discourages overly complex models. In other words, it shrinks the model coefficients, especially for less important features or features that are dependent on others. As stated before, we noticed some codependencies in our data, especially among area and bedrooms. Regularization was our answer to possibly combat this issue in hopes of achieving better performance. The regularization models we chose were ridge regression, lasso regression, and XGBoost.

6.1. Ridge Regression

The loss function for ridge regression using the sum of squared errors as a base is given as

$$\text{Loss}_{\text{Ridge}} = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} X_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} {\beta_j}^2 \quad (2)$$

where X is the feature matrix (shape $n \times p$), y is the target vector (length n), $\boldsymbol{\beta}$ is the coefficient vector (length p), and λ is the penalty term coefficient (a real number), which is called α in `scikit-learn`.

The penalty term here (λ) helps to reduce the influence of less important features but ultimately doesn't discard any from the model. We used a `logspace` function to generate 400 values between 0.01 and 100, and used cross-validation to determine the optimal λ. With a cross-validated λ of 24.46, we achieved a scaled RMSE of 0.210 and an R^2 score of 0.682. Comparing this with our linear regression result, we do not see a significant improvement.

6.2. Lasso Regression

The loss function for lasso regression using the sum of squared errors as a base is given as

$$\text{Loss}_{\text{Lasso}} = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} X_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \quad (3)$$

The penalty term here not only helps reduce the influence of less important features, it can also send some of them to zero. We obtained λ here in a similar fashion to ridge, except we set the `logspace` range from 10^{-5} to 1 (instead of 0.01 to 100). Despite lasso regression's feature-selection ability, it gave us a scaled RMSE of 0.210 and an R^2 of 0.681—exactly the same as our standard linear regression result. This makes sense once we consider how the optimal λ value was 10^{-5}. Notice how λ was the lower endpoint of the interval 10^{-5} to 1. In fact, any lower boundary we picked was always selected as the optimal λ, no matter how low we made it. With λ being near zero, the model was essentially just performing standard linear regression. This suggests that the penalty term isn't significant, confirming linear regression's suitability.

While lasso regression did not shrink any features to zero, we can force it to perform feature selection by setting the λ high enough. In this case, we can gain some insight into what lasso views as important features by noting which features zero out first, as seen in Figure 7.

This figure aligned mostly with the results from the random forest in terms of important features; in particular, `log_area` and `bathrooms` were still the most important features. Another interesting discovery is that all the features remained present for a long time, and then suddenly zeroed out around 10^{-1}, with the less important features zeroing out first. This lack of spread could indicate that all the features were useful in predicting the price, and would explain why λ was picked to be so small.

6.3. Differences in Feature Importance Between Models

If we compare the important features between Linear Regression, Ridge, and Lasso (Figure 8), we see just how much regularization affected each feature.

Recall that the optimal λ for ridge regression was 24.46. In Figure 8, we can see how much this affected each feature. While there were minor adjustments to the coefficients, nothing stands out, and this helps explain our finding that ridge performed very similarly to linear regression. We

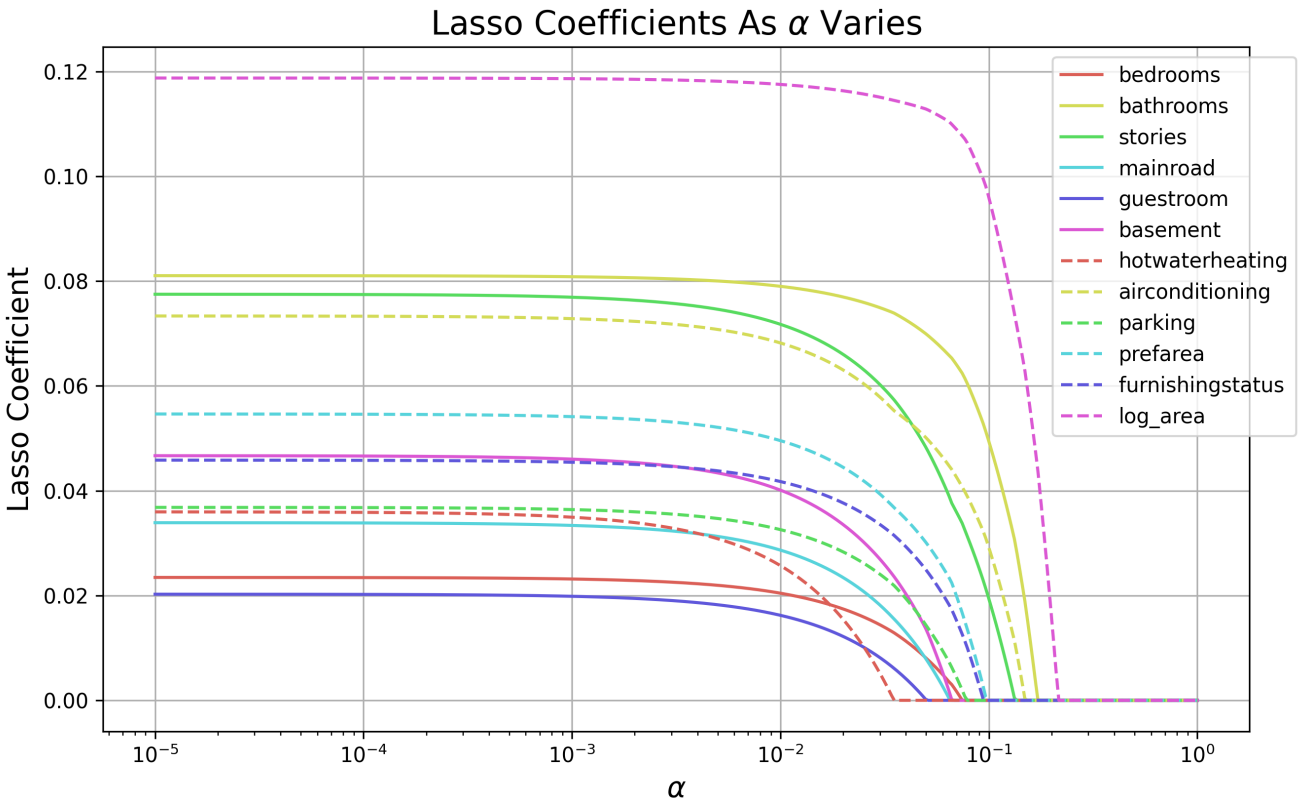

Figure 7. Some coefficients were set to zero earlier than others, meaning they were less important to the lasso model. Note: `scikit-learn` calls the penalty term α.

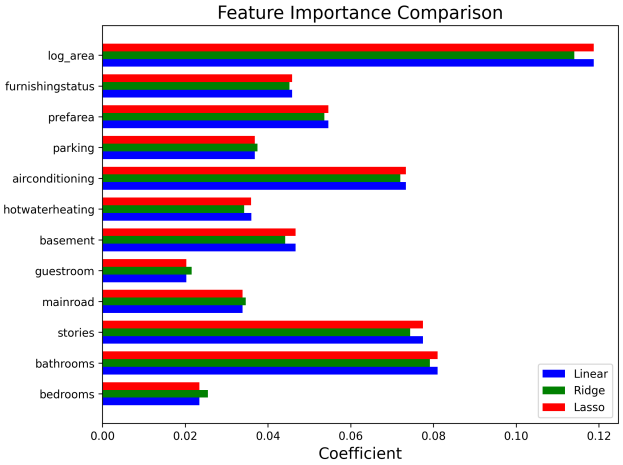

Figure 8. A comparison of regression coefficients between different models

also see that lasso and linear regression's coefficients are the same. This also makes sense given the very small value for λ, as discussed in section 6.2.

6.4. XGBoost

The last model we tested in regularization is XGBoost, specifically with trees. XGBoost is more than just regularization. It is an implementation of gradient boosting, which is a method of building models sequentially, in which each new model tries to correct the errors of the previous ones. Additionally, it has L_1 and L_2 regularization (*i.e.*, lasso and ridge) built into the model. Even though it can be done with a multitude of weak learners, XGBoost with trees is the state-of-the-art application [2]. Applying XGBoost to our dataset, we achieved a scaled RMSE of 0.216 and an R^2 of 0.663, again yielding no improvement over simpler models. In conclusion, regularization did not produce any significant results.

7. Dimensionality Reduction

Since removing features did not increase performance, we decided to try PCA to reduce multicollinearity. We used PCA with ridge regression and performed cross-validation to determine both the optimal number of principal components to extract and the best value for the regularization parameter α. The choice of `n_components` that resulted in

the best performance was 12, which matches the number of original features. This means that actually, all of the features are valuable in predicting the target, and the model does not benefit at all from dimensionality reduction. Using these 12 principal components in our linear regression model gave us an R^2 of 0.682—the same as standard linear regression in section 4.1. We tried the same process using SVR, yielding similar results: `n_components` $= 12$ and $R^2 = 0.684$.

In an effort to obtain meaningful results from PCA, we tried applying it selectively to different subsets of features. Specifically, we performed a brute force on all possible subsets of the features. For each subset, we applied PCA, performed cross-validation to select the optimal number of principal components to extract from this subset, and then combined these principal components with the remaining (untouched) features. This process was repeated for each possible subset, and each resulting feature set was used to train a linear regression model.

The best results came from extracting one principal component from the `bathrooms`, `basement`, and `airconditioning` features, with an R^2 of 0.687, only a marginal improvement over SVR from section 4.2.

8. Conclusion

In this paper, we explore various machine learning models to predict house prices, beginning with standard multiple linear regression as our baseline, and moving on to more advanced techniques such as regularization methods, support vector regression, principal component analysis, and adjustments for multicollinearity using the variance inflation factor. After trying many different techniques, we could not find a model that produced significantly greater results than standard linear regression. One could argue that PCA or SVR produced better results, but we decided that these minor improvements were not practically significant. However, perhaps this is not a bad thing. A simple model like linear regression is more easily interpretable, and if a simple, interpretable model can be used without sacrificing performance, it can be more valuable for us to understand the data itself. Ultimately, this reinforces the idea that complexity is not always necessary—simple, well-understood models can be both effective and insightful.

We now take this time to discuss the limitations and future considerations for this project. In order to see if our models generalize well, we would like to test on other datasets. This would also give us insight into whether our models overfit or not. Since this dataset only contains 545 points, it is possible that the data does not capture enough of the variation or nuance in the very diverse housing market. More data to train and test on can help tune our models or even open the doors to different models that may perform better, like neural networks. We considered polynomial regression in our planning phase, but ultimately decided against it as the data looked linear, so a polynomial model may have a higher risk of overfitting. In terms of how the current models can be improved with the data we are currently using, we can consider interaction terms in linear regression. Since we already researched the multicollinearity of the features, this is a natural next step. We could also consider feature selection or outlier detection. We originally decided to keep all of our features and possible outliers to maximize the use of all our data, but it may be true that certain features or outliers are adding unwanted noise to our models.

Many different models were presented throughout the course of this paper—we use Figure 9 and Table 4 to organize the methods used and their results into one place for the reader's convenience. Additionally, the code we used to achieve these results can be found on GitHub [5].

References

[1] M. O. Akinwande, H. G. Dikko, and A. Samson. Variance inflation factor: As a condition for the inclusion of suppressor variable(s) in regression analysis. *Open Journal of Statistics*, 05(07):754–767, 2015.

[2] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794. ACM, Aug. 2016.

[3] M. Y. H. Housing prices dataset. `https://www.kaggle.com/datasets/yasserh/housing-prices-dataset`, 2020.

[4] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor. *An Introduction to Statistical Learning: with Applications in Python*. Springer Texts in Statistics. Springer, Cham, 2023.

[5] M. Kim, A. Kushalapa, and A. Shaw. Housing Price Prediction. `https://github.com/ashaw1270/housing-prices`, May 2025.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[7] S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[8] J. Ye. Research on the factors affecting house price using multiple linear regression model. *Science and Technology of Engineering, Chemistry and Environmental Protection*, 1(9), Oct. 2024.

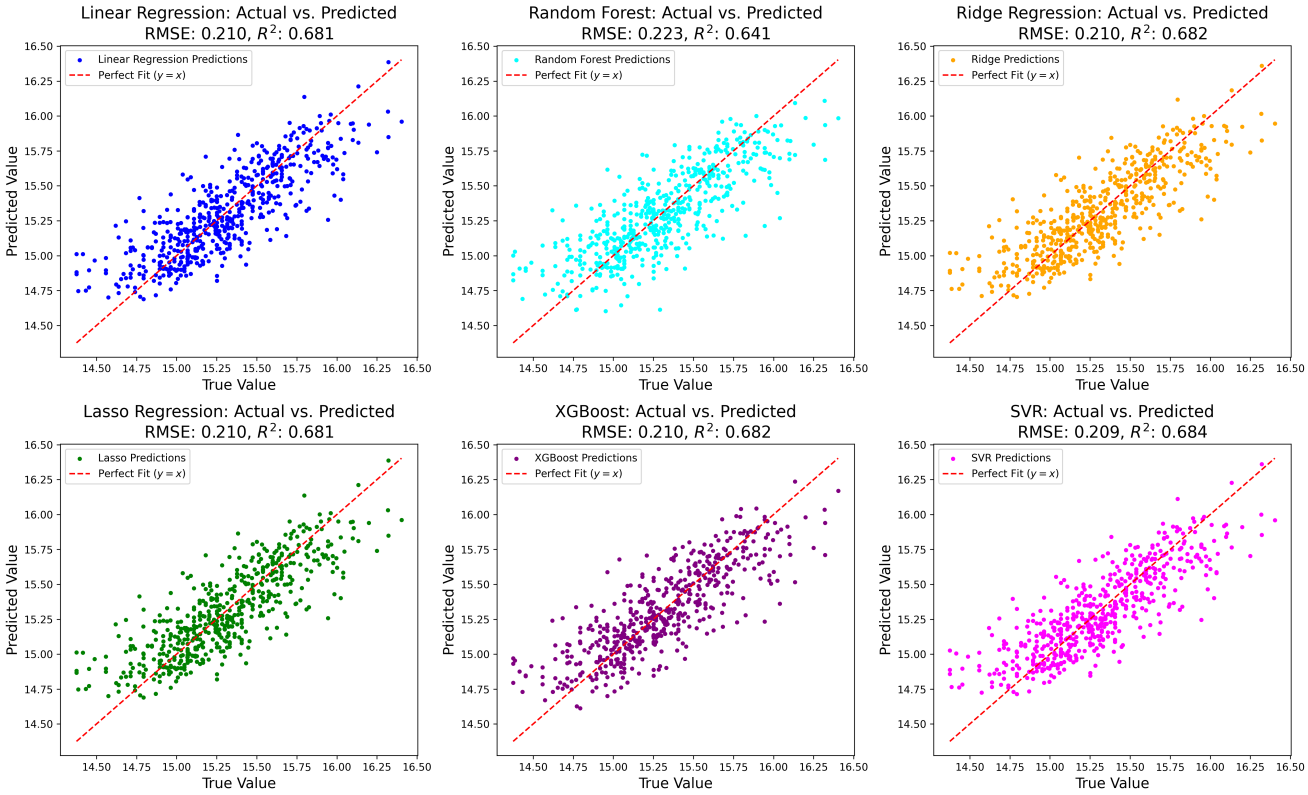

Figure 9. Actual vs. predicted values for all models used

	RMSE	R^2	MSE	MAE	Adjusted R^2	MAPE (%)
Linear Regression	0.210	0.681	0.044	0.163	0.674	1.069
Random Forest	0.223	0.641	0.050	0.170	0.633	1.109
Ridge Regression	0.210	0.682	0.044	0.162	0.674	1.063
Lasso Regression	0.210	0.681	0.044	0.163	0.674	1.069
XGBoost	0.210	0.682	0.044	0.160	0.675	1.046
SVR	0.209	0.684	0.044	0.162	0.676	1.062

Table 4. Performance metrics for all models used