Machine Learning Model for Real Estate Price Prediction in Houston: Comparing and Optimizing to Find a Higher Accuracy

Ricardo Gabriel Mendez Pinero

Received June 22, 2025
Accepted October 22, 2025
Electronic access November 30, 2025

The house price prediction for real estate analysis is one of the most extensively explored areas within the machine learning field. By combining features and training different models, researchers have achieved high accuracy in this task across multiple studies. However, these models are generally trained with data from broader areas such as states or entire countries, which generalizes better but may perform worse for certain regions. For that reason, the focus of my study lies in finding a model to predict house prices in Houston, particularly in the Houston metropolitan area, which is the fifth-largest metro area in the US and includes the City of Houston, which is the fourth largest city in the US by population and has been growing in the housing market in recent years. To do so, I used the Houston housing market 2024 dataset to train and compare 5 models–Ridge, Random Forest Regressor, Artificial Neural Network, XGBoost, and LightGBM– using mean absolute error, mean squared error, root mean square error and r-squared as error metrics. After testing, I determined the LightGBM model had the best performance, achieving an r-squared value of 81%, and a mean absolute error of 46k –12% of the mean price. Therefore, my study provides a model to predict house prices in Houston effectively.

Keywords: Machine Learning, Houston, Real Estate, LightGBM, Housing, Artificial Intelligence.

Introduction

Machine learning models have been widely applied for real estate data analysis. Over time, the models have evolved in both accuracy and complexity –from linear regression models to algorithms capable of capturing nonlinear relationships. S. Rosen introduced the hedonic pricing model, in which the price of the house is determined by a combination of internal characteristics (such as size or appearance) and external factors (such as crime rates or proximity to schools). He proposed that the price could be modeled as a function of these attributes, which could be estimated using regression techniques [1]. However, the linear regression models struggle to capture nonlinear patterns within the data. For that reason, recent advances in machine learning (ML) have introduced new possibilities for real estate price prediction. Unlike traditional models, ML methods such as Random Forests, Gradient Boosting Machines, and Neural Networks can model complex, nonlinear interactions between variables without strong assumptions about data distribution. In fact, studies have demonstrated that other ML models typically outperform hedonic models models based on Rosens hedonic pricing model in predictive accuracy. In their study, E. Antipov and E. Pokryshevskaya compared several machine learning models like KNN and Artificial Neural Networks to Random Forest to determine whether it was possible to use Random Forest for mass appraisal, which is valuing a group of properties with common characteristics using statistical methods. Besides, they used metrics like coefficient of determination and mean average percentage error for evaluating the models. In fact, they proved their hypothesis of applying random forest as a method for mass appraisal to be valid [2].

Another example of the application of machine learning for predicting prices in the real estate field is Zillow's Zestimate, which is a widely known automated valuation model (AVM) that applies machine learning techniques to estimate home prices across the U.S. According to Zillow, the Zestimate has a nationwide median error rate of 1.83% for on-market homes, and 7.01% for off-market homes. Nonetheless, its accuracy depends on the availability of data in a home's area [3].

In 2017, Zillow launched the Zillow Prize, a $1 million public competition aimed at improving the Zestimate algorithm. The competition attracted more than 3,000 teams from over 90 countries. Participants used cutting-edge ML techniques including ensembles of LightGBM, XGBoost, and neural networks, along with feature engineering. The winning model reduced the Zestimate's error by 13%, demonstrating the value of community-driven innovation and advanced machine learning. While the Zestimate had been calculating the price based on characteristics like square footage, location, listing price, description, and tax assessments, the winning team highlighted the use of rental rates, commute times and contextual information like road noise to reduce the error of their model. The use

of these features shows that features not necessarily associated with the building's characteristics but with the broader context in which it fits, like the time it takes the owner to travel from the property to his place of work, provided useful information to improve the model. Likewise, they emphasized the use of neural networks and the outliers' removal[4].

The competition showed how combining different models and tuning them with approaches like randomized search can yield better accuracy than any single algorithm. In 2019, Zillow announced a major update involving a single neural network model trained on a national dataset, which improved the median error from 6.9% to 4.5%[5]. The new model leverages a wide range of data sources, including geospatial data, tax records, and even image data from property listings[6]. In addition, other studies have proved the efficacy of the gradient boost models for predicting house values. L. John et al. used LightGBM to predict property values with 90% accuracy. They decided to use LightGBM instead of other gradient boosting models like XGBoost for being faster and adapting better when the size of the data was large. They compared logistic regression, random forest regressor, and LightGBM to see which performed the task of house prices better. In fact, they found LightGBM to perform better[7]. Nonetheless, the primary location of their study is Bangalore, which is a different housing market from the Houston housing market. Besides, S. Bushuyev et al. found that the predictions of house prices could be improved by using textual descriptions of the properties. In their study, they used LightGBM in combination with the tokenization of keywords in the properties' textual descriptions to increase the accuracy of the predictions[8]. As a result, they reduced the mean square error by 13.4%, which shows the possibility of improving models using this approach.

Also, J. Ruan portrays Houston's real estate landscape and uses machine learning models to predict house prices. First, he introduces data about the growth of house prices in Houston over the last few years. Then, he addresses and explains the trend in Houston's expansion from 2017 to 2023 using k-means clustering, and panel regression to explain the evolution in prices over time, which shows a relationship between the highways and transport infrastructure with the expansion pattern of houses. Finally, he used machine learning models –random forest and XGBoost– to predict house prices. Also, he found that the models were underpredicting the prices when they were above 300k[9]. Even though many studies have addressed real estate analysis using machine learning models, they tend to focus on markets of broader areas, which may generalize better for predicting in a bigger market but might not be very effective for predicting in a specific area. Therefore, if a model was trained only with data from that area, it would predict better on that data.

For my project, I decided to focus on the Houston metropolitan area because it is one of the metro areas with the largest population in the country, a growing housing market and not many studies focusing on predicting house values there. According to the U.S. Census Bureau, the Houston metropolitan area is the fifth largest by population in the country, with nearly 7.8 million residents[10]. Also, it was the second fastest growing metro area in the U.S. by numeric growth between July 1, 2023, and July 1, 2024, adding 198,171 people[11]. Besides, it includes the City of Houston, which is the 4th largest city in the US by population with a population of 2.3 million people. According to Redfin, while the number of homes sold in the U.S. has decreased a 1.3% year over year for July of 2025, the City of Houston experienced a 10.7% increase[12]. Similarly, the Houston Association of Realtors' August 2025 Housing Market Update reports that the total properties sales in the Houston metropolitan area rose 9.2% compared to August 2024[13]. Together, these reports suggest that the Houston housing market has been more active in terms of home sales than the national market. Therefore, the goal of my study lies in training a model to predict house prices in the Houston market. As a result, the project provides a practical tool for buyers, sellers, realtors, and real estate analysts in Houston. Besides, it could also be useful as a tool for further research into machine learning models applied to the Houston housing market.

For my study, I used the "Houston housing market 2024" dataset". However, given the computational limitations of my machine, I decided to use the light version of the dataset, which contains fewer features and thus might make the models predict worse. To conduct my study, the first step was cleaning the dataset from values that may make the predictions worse by preprocessing. Then, the dataset was split into three subsets for training, development, and testing. Then, I trained the models with the training set to compare their results in the development set to find the one with the highest accuracy in most of the metrics. Finally, I tuned the hyperparameters of the best-performing model to increase the model's accuracy and retrained it to test it in the test dataset to see how it performs in front of unknown data.

Methods

The study uses a cross-sectional, observational research design. It evaluates real estate data collected in 2024 to model the relationship between housing features and their market prices. Since the data reflects a single period and no experimental manipulation was done, the study is observational rather than experimental or longitudinal. To conduct the study, I used a Kaggle notebook with Python. Moreover, some of the libraries frequently used were the following: NumPy[14], for mathematical operations; and pandas[15], for data manipulation. The steps taken were the following:

- Download the dataset

- Clean the dataset from null values, outliers, and highly correlated features by using data preprocessing techniques
- Split the dataset into three subsets: training, development, and testing
- Train the five models–Ridge, XGBoost, LightGBM, Artificial Neural Network, Random Forest Regressor– with the training subset
- Test the five models in the development subset using the error metrics MAE, MSE, RMSE, and R-squared to choose the model with the lowest error in most of the metrics. Also, the cross-validation was used over the training set to determine R-squared across several data folds
- Use the method randomized search with the data of the training set to find the best hyperparameters combination for the model that performed best in the development subset. Then, train the model again with the new params in the training subset
- Test the model in the test subset using the same four metrics after tuning its hyperparameters

Dataset

For this project, I used the "Houston housing market 2024" dataset uploaded by Natasha Lekh on Kaggle[16], which contains data from Zillow. The dataset contains two versions: The heavy version, 6.72 GB; and the light version, 143.57 MB. While both have the same number of listings, the heavier version contains approximately 17 additional predictive features such as schools near the property, nearby homes, tax history, and the year the property was built. Nonetheless, some of the processes made when using machine learning models and large amounts of data can be very computationally expensive for the resources needed to execute them. For that reason, due to the computational limitations of my machine like a lack of a lot of RAM or a stronger processor, I used the light version. Given the fact that the light version contains less features, the models might not capture some of the relationships that involve the features that are not present in the light version, which could be improved if the heavier version of the dataset were used. However, it is also possible that those features are noisy or lead to overfitting, in which case their absence could improve generalization. Also, it might be the case that by using more features the number of rows with null values increases, which implies a reduction of total samples that might negatively affect the models because there is less data available for the models to learn. Nonetheless, it would be necessary to experiment with the heavier version to know the real effect on the model's generalization and performance. The light version of the dataset contains 25948 rows and 58 columns. Table 1 shows the original features of the dataset. Nonetheless, given it is a JSON file many other features are inside some columns. For example, columns such as latitude and home type were inside a column called "hdpData", so I had to extract them and remove the original column. Likewise, I dropped some other columns that were redundant, like the column called "hdpData" that had the same values that the columns I extracted from it, or related to showing the property in a website, like photos or street view, rather than representing a feature of the property. Consequently, Table 2 presents the features after extracting and dropping columns. Similarly, the column addressStreet was removed because it was the exact address of the property. Hence, given it was a unique identifier it did not have predictive power.

Data Preprocessing

Given the presence of thousands of entries with null values that could negatively affect the predictions, they were dropped. It reduced the number of rows to 16824. Also, when I visualized numerical and categorical features using the libraries matplotlib[17] and seaborn[18], it turned out to be a gap in the distribution of the zip codes, as shown in Figure 1. Besides, there were entries of other cities besides Houston in the dataset, which are the cities adjacent to the City of Houston that belong to the Houston metropolitan area and may have caused this disparity in the zip code distribution. Given the scope of the project was the Houston metropolitan area, the other cities were kept. Nonetheless, the results of the initial comparison of the models using only the samples from the City of Houston, which are 6109 samples in total after dropping rows with null values and removing outliers, can be seen in Table 5. To improve the accuracy of the models and prevent extreme values from affecting the results, I applied the Interquartile Range (IQR) method, also known as the quartile rule, to remove the outliers in each feature distribution. To use the Interquartile Range, we need to compute the first quartile, the value below which 25% of the data falls;

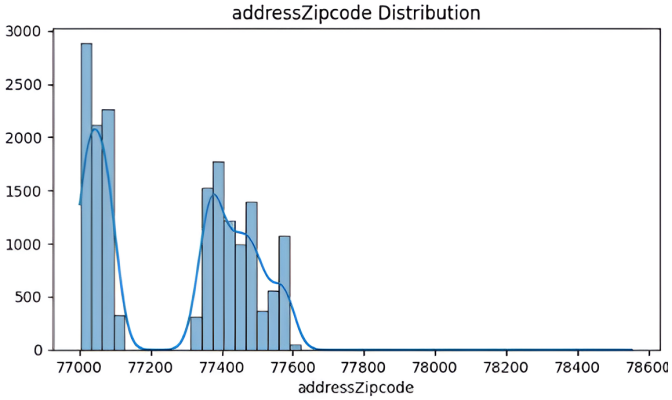

Fig. 1 Distribution of zip codes in the dataset.

Table 1 Original Shape of the Dataset.

Number	Column
0	zpid
1	id
2	rawHomeStatusCd
3	marketingStatusSimplifiedCd
4	imgSrc
5	hasImage
6	detailUrl
7	statusType
8	statusText
9	countryCurrency
10	price
11	unformattedPrice
12	address
13	addressStreet
14	addressCity
15	addressState
16	addressZipcode
17	isUndisclosedAddress
18	beds
19	baths
20	area
21	latLong
22	isZillowOwned
23	variableData
24	hdpData
25	isSaved
26	isUserClaimingOwner
27	isUserConfirmedClaim
28	pgapt
29	sgapt
30	zestimate
31	shouldShowZestimateAsPrice
32	has3DModel
33	hasVideo
34	isHomeRec
35	hasAdditionalAttributions
36	isFeaturedListing
37	isShowcaseListing
38	list
39	relaxed
40	brokerName
41	carouselPhotos
42	hasOpenHouse
43	openHouseStartDate
44	openHouseEndDate
45	openHouseDescription
46	lotAreaString
47	providerListingId
48	builderName
49	streetViewURL
50	streetViewMetadataURL
51	isPropertyResultCDP
52	flexFieldText
53	flexFieldType
54	info3String
55	info6String
56	info2String
57	availabilityDate

and the third quartile, the value below which 75% of the data falls. Then, we calculate the IQR with the following formula:

$$\text{IQR} = Q_3 - Q_1. \tag{1}$$

Next, we use it to evaluate all data points: Any data point below $Q1 - 1.5 \times$ IQR or above $Q3 + 1.5 \times$ IQR is considered an outlier and is removed. This method is widely used because it is non-parametric, meaning it makes no assumptions about the distribution of the data and is particularly effective in real estate

Table 2 Shape of the dataset after extracting and dropping columns.

Number	Column
0	marketingStatusSimplifiedCd
1	unformattedPrice
2	addressCity
3	addressZipcode
4	isUndisclosedAddress
5	beds
6	baths
7	area
8	latitude
9	longitude
10	homeType
11	lotAreaValue

datasets where extreme prices or sizes can negatively affect the model. Also, there are other common methods for outliers' removal that can be effective as well. For example, the Z-score is a method that calculates how many standard deviations a value is from the mean. For each point x, the Z-score is calculated with the following formula:

$$z = \frac{x - \mu}{\sigma} \tag{2}$$

After calculating the z score, a threshold like $z > 3$ is used to categorize the point as an outlier. This method is useful for data following a gaussian distribution because it uses the mean as a measure of how far a point is from the others.

Additionally, another method used to detect outliers is the Isolation Forest. It is a tree-based algorithm, which is explained in the subsection 2.4 Models, that categorizes outliers as the points that require a lower number of splits to reach than the other points. This algorithm is useful because it does not make assumptions about data distribution, which means that its performance is not affected by the data distribution, and it can detect relationships among different variables, like a house with a large price but with a small number of bedrooms, which is something that neither the IQR nor the Z-score can do, because they just evaluate the outliers of only one variable; if you had several features, you would have to detect the outliers independently for each feature.

However, these methods have some disadvantages that resulted in the selection of the IQR over them for this project. Unlike the IQR, the Z-score needs a gaussian distribution to work properly, but in this case many of the features do not

follow a gaussian distribution, so this method would not be appropriate. Also, even if the Isolation Forest does not make assumptions and captures multivariable relationships, it requires manual tuning of hyperparameters, which can introduce bias if not implemented correctly; however, given the Isolation Forest is a machine learning algorithm, it would require a separate training process to implement it properly. Therefore, even though the IQR cannot detect relationships among different variables, it was chosen over the other methods because of its simplicity to implement and the fact that it does not make assumptions about data distribution.

Figure 2 displays the data distribution of each numerical feature after handling outliers. After using the IQR method

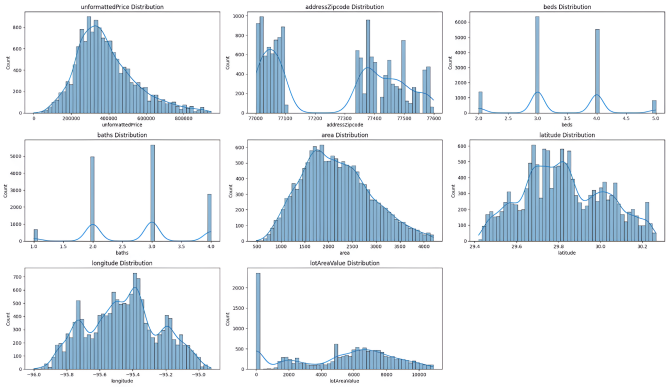

Fig. 2 Numerical feature distributions.

in my project to remove outliers, the number of entries was reduced to 14097. Also, the properties had a median price of $359,500, a mean price of $386,332 and a mode of $350,000, which can provide useful context when discussing the results of the models and their impact.

Besides, Table 6 shows how the performance is drastically affected by the presence of the outliers.

I also used a correlation heatmap to see the correlation among features and the correlation between each feature and the unformatted price, the value that is going to be predicted. This method is used to prevent multicollinearity, which occurs when independent variables are highly intercorrelated, potentially leading to a reduction in the performance of the models. To do so, I was going to delete one of the features if it had a correlation of more than 0.70 or lower than -0.70 with another feature. In fact, the only pair of features that crossed the boundary was the baths-area pair, but when I removed it during the testing phase the accuracy of the models slightly changed; particularly, the performance of the best model slightly decreased, as can be seen in Table 7, so I decided to keep it. Therefore, no features were removed. Figure 3 presents the resulting correlation heatmap used to visualize the correlation among features.

The categorical variables 'homeType', 'marketingStatusSimplifiedCd', and 'addressCity' were transformed using one-hot

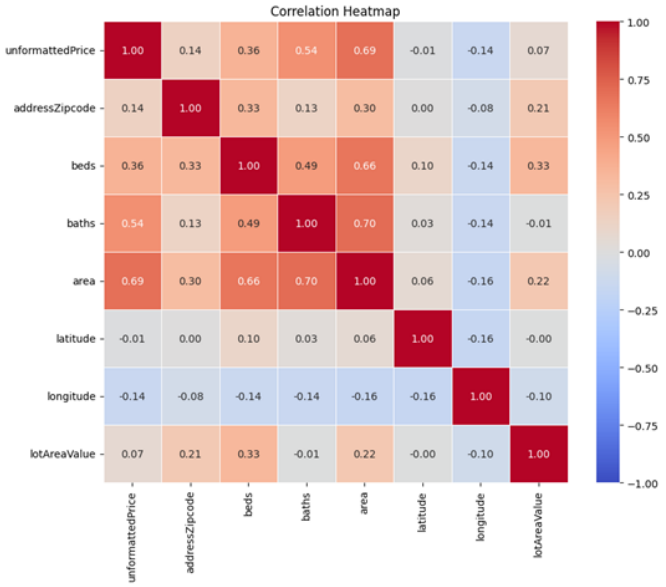

Fig. 3 Correlation matrix(heatmap).

encoding to allow their use in machine learning models. This was implemented using the get_dummies() function from the pandas [15] library in Python.

Splitting the Data

To train the models, we need to split the data into several sets for training and testing, where each pair in each of the subsets contains all the columns but the unformatted price, our output, as inputs, and the unformatted price as our output. I used the approach of splitting the dataset into three subsets: 60% for training all the models; 20% for development; here I will compare the results of each model to choose the one that best performs, also the results of using cross-validation with R-squared as the error metric from the training set, and make adjustments to the models to improve their performance on this dataset; and 20% for testing, here I will test the model that performed better in the development dataset after hyper tunning it with data from the training dataset to have a realistic view of its performance in the new dataset.

The approach of using a development set instead of a simple train-test split is used to avoid biased results. The reason for that is that by using only two sets, training and testing set, we will not have a dataset to see how the models perform in front of unknown data, because we will use the results gotten from the testing set to adjust the model to improve these results, so we are manually telling the models what is the right configuration for the models in that particular dataset, so the predictions will be more accurate there than if we did not give them the additional information that is not available when the model is facing new data, which is the realistic condition. Therefore, by implement-

ing a development set, we can use its results to improve the accuracy there and reserve the testing set to see how the models perform in unknown conditions.

The total number of rows of the dataset used to train the models after preprocessing was 14097 8457 for training, 2820 for development, and 2820 for testing. To split the data, I used the method train_test_split from the library scikit-learn [19]. Then, I scaled each subset with the method StandardScaler from scikit-learn [19].

Models

Ridge Regression:
Ridge regression is a variant of linear regression that adds L2 regularization to the cost function. This regularization penalizes large coefficients to reduce overfitting, handle multicollinearity, and lower the model's variance.

In a linear regression model, we have an ideal hypothesis function that is the sum of every input feature times a weight plus an initial weight that represents the bias. Therefore, the goal of the model is to update the weights to make the actual hypothesis as close to the ideal hypothesis as possible. To do so, the model uses gradient descent to iteratively update the weights in the direction that minimizes the cost function.

Cost function:

$$J(\theta) = \sum_{i=1}^{n} \left(y_i - \theta^\top \mathbf{x}_i \right)^2 + \lambda \sum_{j=1}^{p} \theta_j^2. \tag{3}$$

Where:

- n: number of training samples
- p: number of features (excluding bias if not regularized)
- y_i: true output for the ith sample
- $\mathbf{x}_i$: input feature vector for the ith sample
- θ: weight vector (parameters)
- $\theta^\top \mathbf{x}_i$: prediction for ith sample
- λ: regularization strength (hyperparameter)

The cost function is the sum of all the differences between the real output and the hypothesis for each sample in the dataset. However, in Ridge regression the cost function also includes the regularization term to reduce overfitting.

To use gradient descent, we use the following formula.

Gradient:

$$\nabla_\theta J = -X^\top (y - X\theta) + \lambda \theta \tag{4}$$

This expression combines the gradient of the mean squared error loss with the gradient of the L2 regularization term. The first term moves the parameters in the direction that reduces the prediction error, while the second penalizes large weights to prevent overfitting.

Update Rule:

$$\theta := \theta + \alpha X^\top (y - X\theta) - \alpha \lambda \theta \tag{5}$$

Where α is the learning rate. This rule adjusts the weights in the direction that minimizes the cost function while using the regularization term. Figure 4 reproduces Nicoguaro's illustration to show the effect of regularization [20], where the x-axis represents a feature and the y-axis represents the prediction. Also, the blue line is the hypothesis without regularization, and the green line is the hypothesis with the regularization term.

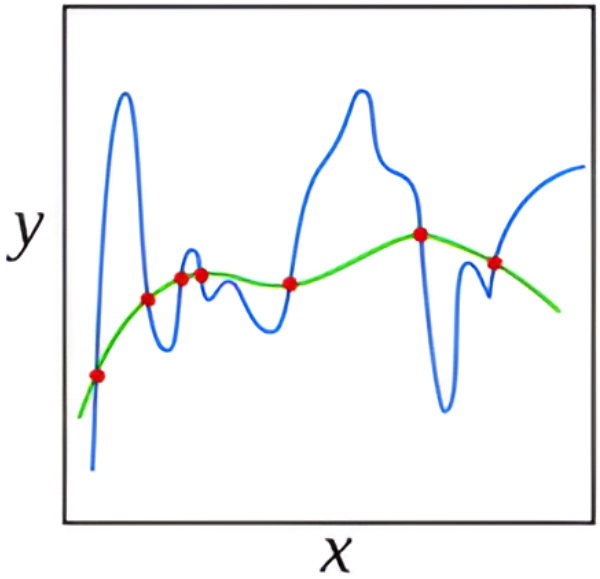

Fig. 4 Effect of regularization.

To reduce the error as much as possible, the model will try to fit the line to predict correctly all the output values. However, in doing so, the model might create a very specific hypothesis that will not generalize well when facing new dataoverfitting. In this case, the blue line passes through all the datapoints; nonetheless, given it has several specific curves that might not be found in another dataset, the model will predict inaccurately for values that do not follow the same specific path, even if they are close to the line. Therefore, by inserting the regularization term, the model creates a simpler line, the green line, that avoids specific curves or shapes such that if the line is used in a new dataset, the model might still predict with a similar accuracy. For this project, I used the Ridge model from scikit-learn [19].

XGBoost:

XGBoost, which stands for Extreme Gradient Boosting, is a machine learning algorithm based on the gradient boosting

framework[21]. The model uses the principle of gradient boosting by creating several trees and training them to reduce the error from the previous trees. However, the model parallelizes the internal process when building each tree, which makes the model faster. Besides, the model uses a level-wise strategy, which is to try to split all the nodes at the same level before going deeper into the next level. Additionally, the model calculates the first and second derivative of the loss function to take decisions about the quality of the split at each split.

The first derivative tells the model the direction and magnitude of the gradient to adjust predictions and reduce the loss, and the second derivative tells the model how sensitive the gradient is to a change in the predictions, which indicates how much the gradient changes depending on the curvature of the loss. By using both, the model avoids drastic changes that would occur if the model used a gradient that suggests a large update at a point with a high curvature in the graph, so the updates are more stable and less likely to give large inaccurate predictions.

The objective function minimized by XGBoost at each iteration includes both a loss function that measures how well the model fits the data, and a regularization term that penalizes model complexity to reduce overfitting:

$$\text{Obj}(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{6}$$

Where:

- n: number of samples
- $l(y_i, \hat{y}_i)$: loss function comparing true label y_i and predicted label $\hat{y}_i$
- K: number of trees (or models)
- $\Omega(f_k)$: regularization term for the k-th tree/function f_k

For my model, I set the hyperparameter alpha equal to 1.0 to apply L1 regularization.

Random Forest:

Random Forest is an ensemble model that creates several decision trees to predict in regression and classification tasks. It is based on the idea that many trees can predict better than just one to avoid overfitting. The model starts by creating subsets of the dataset where some samples can be repeated more than once and others are ignored, which is known as bootstrap sampling. Then, for each bootstrap sample or subset a decision tree is created and trained using splits. At each split, it randomly selects some of the features and picks the best split among those features. Once all the trees are trained, each tree is used to make a prediction. Ultimately, if it is a classification task, the final prediction is the class that most of the trees predicted. On the other hand, if it is a regression task, the final prediction is the average of the predictions from all the individual trees:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} f_t(x) \tag{7}$$

Where:

- $\hat{y}$: the final prediction
- T: the total number of models (e.g., trees in an ensemble)
- $f_t(x)$: the prediction from the t-th model

The model reduces overfitting because of the use of many trees trained with different bootstrap samples and a random subset of features when splitting. For the project, I used the scikit-learn random forest model[19]. Also, I used the following combination of hyperparameters to get a balance between computational cost and accuracy.

Params= {
n_estimators=100,
max_depth=10,
min_samples_leaf=5,
max_features=0.8,
min_samples_split=10,
n_jobs=-1}

Where:

n_estimators: the number of trees that there will be in the forest; each of them is trained to make predictions, which will be averaged to produce the final prediction of the model max_depth: the longest distance from the root to a leaf; it is measured by the largest number of splits that it takes to reach one of the leaves. A node will not be split once this depth is reached min_samples_leaf: the minimum number of samples that there must be in each leaf; it prevents the trees from creating leaves with fewer samples than this number max_features: the percentage of the total features that will be used at every split to find which of them reduces the error the most and thus gives the best split. In each split, the features are selected randomly, but the number of features is determined by this parameter min_samples_split: the minimum number of samples that a node must have to be considered for splitting n_jobs: the number of CPU cores(threads) to use when fitting the model or predicting; each tree can be grown on a different CPU core to increase the speed of the processes

Artificial Neural Network:

Artificial Neural Networks are models that build structures of interconnected neurons, based on the neural networks of human brains, to make predictions. Especially, they are useful to capture non-linear relationships within the data for the multiple neurons involved. The neuron is the basic node or unit in the network. It has weights for each of its inputs plus an additional

bias. In order to predict, a neuron multiplies each weight with its corresponding input and adds the additional bias. Within an artificial neural network, we have layers of several neurons where the predictions of each of the neurons of a layer are the inputs of each of the neurons of the following layer. As a result, we have a dense network of connected neurons that are ultimately connected to a final neuron that predicts (in the case of regression). The output of each neuron is computed as:

$$z = \mathbf{w}^\top \mathbf{x} + b, \quad a = \phi(z) \tag{8}$$

Where:

- $\mathbf{w}$: weight vector
- $\mathbf{x}$: input vector
- b: bias term
- $\phi(z)$: activation function (e.g., ReLU, sigmoid, or tanh)

The model is trained using gradient descent to minimize a loss function. Specifically, the model uses an algorithm called backpropagation to compute the gradients of the loss with respect to the model's weights and update the weights of the neurons. For this project, I imported the library Tensorflow to implement the neural network[22]. For its architecture, I used 5 dense layers. The first one is the input layer with 32 neurons, the second and third had 16 neurons each, the fourth had 8 neurons, and the last one, the output layer, had 1 neuron. The first fourth layers had the activation function relu, while the output layer had a linear activation function. After each layer of the first fourth, I used dropout 0.2 to turn off some neurons during training and avoid overfitting. Also, I used the Adam optimizer. Figure 5 is a graphic representation of the neural network's architecture, where each of the large boxes represents a layer. Besides, the input and output shapes indicate the number of values entering and leaving each layer per sample, which corresponds to the number of neurons in the layer.

LightGBM (Light Gradient Boosting Machine):

LightGBM is a gradient boosting framework developed by Microsoft[23]. Like XGBoost, it builds an ensemble of decision trees sequentially to correct the errors of the previous trees using gradient boosting. However, the model uses leaf-wise tree growth, which is to go for the split that reduces the loss function the most even if it makes the tree deeper before completing the splits of the previous level. Also, the model prioritizes data points with larger gradients to improve training speed. Additionally, the model handles features that might have few values by combining them with other features to reduce dimensions, which makes it less computationally expensive. Likewise, instead of evaluating all the split points, it buckets continuous feature values into discrete bins, which also makes it more efficient. The objective function is like other gradient boosting

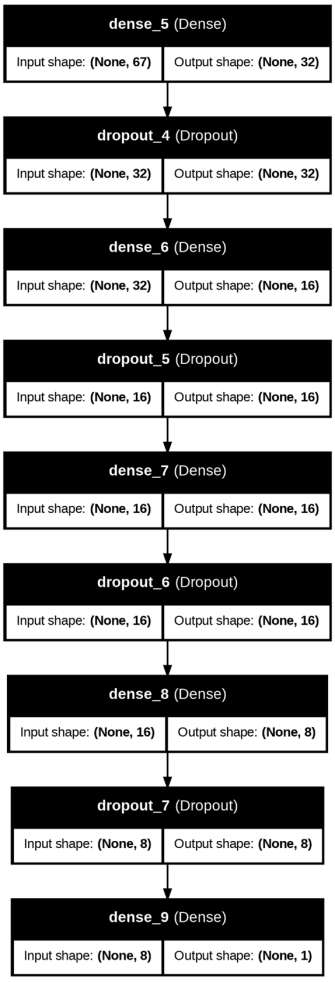

Fig. 5 Architecture of the Neural Network.

methods, minimizing a loss function with a regularization term:

$$\text{Obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \Omega(f) \tag{9}$$

Where:

- $l(y_i, \hat{y}_i)$: loss function comparing true label y_i and predicted label $\hat{y}_i$
- $\Omega(f)$: regularization term to penalize model complexity (e.g., number of leaves in trees)

Comparison of Models:

To determine which was the best-performing model to predict the price, the models were trained in the training set, 60% of the dataset, and tested in the development set, 20% of the dataset, under the metrics mean absolute error(MAE), mean squared error (MSE), root mean square error (RMSE), and r-squared (R2).

In addition, they were trained using k-fold cross-validation with R-squared as the error metric, 10 iterations, over the training set. By taking nine sets for training and one for testing on each iteration, the cross-validation result shows how the model performs when it is trained with slightly different data. Hence, it portrays how well the model generalizes. Given the results of the development set are used to adjust the models to improve the same results and find the one that best performs in this subset, they cannot be trusted as unbiased predictions of real-world performance; despite the bias that those predictions have, the development set is still used to compare and choose the model that appears to perform better than the others over this data to adjust in the same subset later.

Table 3 presents the results of each model over the development set. Besides, the results of using cross validation are also shown.

Table 3 Comparison of results of each model.

Model	MAE	MSE	RMSE	R2	Cross validation
Ridge	69535.42	9.901935e+09	99508.47	0.58	0.57
XGBoost	46683.36	4.931613e+09	70225.44	0.79	0.78
Random Forest	55655.46	6.564925e+09	81024.23	0.72	0.71
Artificial Neural Network	70396.33	1.155054e+10	107473.44	0.51	None
LightGBM	46482.81	4.673367e+09	68362.03	0.80	0.79

It is important to point out that the neural network didn't use cross-validation for computational costs. However, it had a higher error in all the other metrics compared to the other models. On the other hand, the LightGBM model got the best performance in all the metrics.

Given it had the best performance, I used RandomizedSearchCV from scikit-learn[19] to hyper tune the LightGBM model, which is an attempt to find the most optimal combination of hyperparameters to make more accurate predictions by iterating over several combinations of those. The following dictionary defines the parameter distributions used in the randomized search.

param_dist_new = {
'num_leaves': [15,31,63],
'max_depth': [3,5,7,-1],
'learning_rate': [0.01,0.05,0.1],
'n_estimators': [100, 300, 500],
'min_child_samples': [10,20,50],
'subsample': [0.6, 0.8,1.0],
'colsample_bytree':[0.6,0.8,1.0],
'reg_alpha': [0,1,5,10],
'reg_lambda': [0,1,5,10],
}

The following is the combination of the most optimal hyperparameters gotten:

Best hyperparameters: 'subsample': 1.0, 'reg_lambda': 10, 'reg_alpha': 10, 'num_leaves': 31, 'n_estimators': 500, 'min_child_samples': 20, 'max_depth': -1, 'learning_rate': 0.05, 'colsample_bytree': 0.6

After hypertuning the model, I trained it once again in the training set to test it over the test dataset to get a realistic overview of the model facing unknown data. However, to test the model in the test set I used a different approach. Given the fact that the results might be influenced by uncertainty, I used bootstrapping, which is to create several bootstraps (the same process of bootstrap sampling that was mentioned when explaining the Random Forest) with the data of the test set to predict with the hypertuned model in each of them to get the average of each of the evaluation metrics of all the boostraps. In doing so, it is possible to calculate the confidence interval, which is a range of values that is likely to contain the true value of an unknown population parameter with a specified probability (95% in this case). In this context, the confidence intervals indicate the range of values that the different evaluation metrics can have with some certainty, making the results more reliable than point estimates alone. Ultimately, the number of bootstraps used was 1000; each of them with the same number of samples as the test set (2820). Similarly, the confidence intervals were not used for the comparison of models in the development set because those results are biased, as it was mentioned previously, and are not intended to be reported as a final evaluation.

Results

Table 4 presents the results of the evaluation metrics for the hypertuned LightGBM, estimated using 1000 bootstrap samples of the test set. The model achieved a coefficient of determination (R2) of 81%, which indicates that the model explains 81% of the variability in the dataset. Besides, the average absolute error of the model is 46k dollars, which is 13% of the median price and 12% of the mean price of the houses with some predictions off by about $69k, given the root mean square error (RMSE).

Table 4 LightGBM results in the test set after hypertuning.

Metric	Point estimate	95% CI
MAE	45859.82	[43960.82, 47926.82]
MSE	4722163695.32	[4197896264.17, 5278945068.03]
RMSE	68688.18	[64791.17, 72656.35]
R2	0.8135	[0.7939, 0.8310]

Additionally, Figure 6 shows the scatter plot graph of the predicted vs the real price of the properties, where each home has a predicted and real value. The 45-degree line represents the perfect prediction, where the real and predicted values are the same.

The scatterplot shows that while the predictions were accurate under $500k, they tend to be far from the real value as the prices grow over that point.

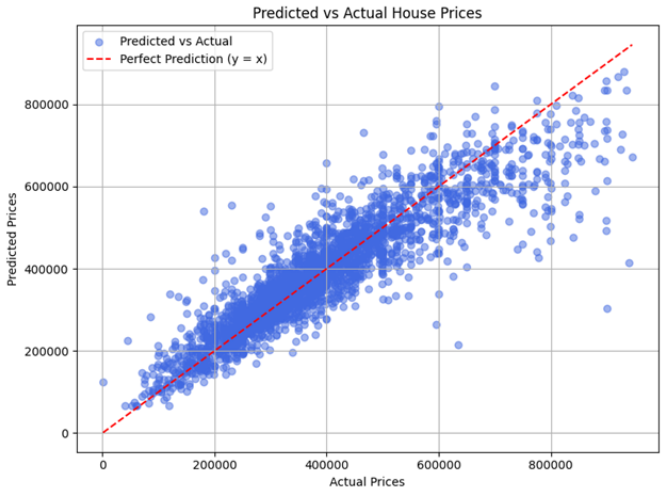

Fig. 6 Scatterplot of predicted vs actual house prices.

In addition, a feature importance analysis, which is used to determine the impact of each feature on the predictions of a model, was used to have a better understanding of the model. To do this, I used the method SHAP, which calculates the change that each feature produces on a prediction and then averages the results, because it works with any machine learning model and ensures that each feature's contribution to the prediction is proportional to its influence, which makes the contributions more accurate. Figure 7 shows the feature importance analysis of the best-performing model. Table 5 shows the comparison of models' performance in the development set when the samples with other cities besides Houston were removed. Likewise,

Table 5 Comparison of results of each model with only samples from Houston.

Model	MAE	MSE	RMSE	R2	Cross validation
Ridge	77685.61	1.174896e+10	108392.63	0.54	0.52
XGBoost	47905.94	5.333597e+09	73031.48	0.79	0.78
Random Forest	51670.28	6.178350e+09	78602.48	0.76	0.75
Artificial Neural Network	81952.02	1.472924e+10	121364.08	0.43	None
LightGBM	45860.94	4.907490e+09	70053.48	0.81	0.80

Table 6 presents the comparison of the models' performance without removing the outliers. Similarly, Table 7 shows the

Table 6 Comparison of results of each model with the presence of outliers in the dataset.

Model	MAE	MSE	RMSE	R2	Cross validation
Ridge	246620.81	6.247609e+11	790418.21	0.53	0.62
XGBoost	131475.45	4.894772e+11	699626.51	0.63	0.68
Random Forest	144847.45	4.759499e+11	689891.22	0.64	0.75
Artificial Neural Network	214922.89	6.689455e+11	817890.86	0.50	None
LightGBM	132194.55	5.130908e+11	716303.57	0.62	0.78

comparison of the models' performance when the column baths was removed.

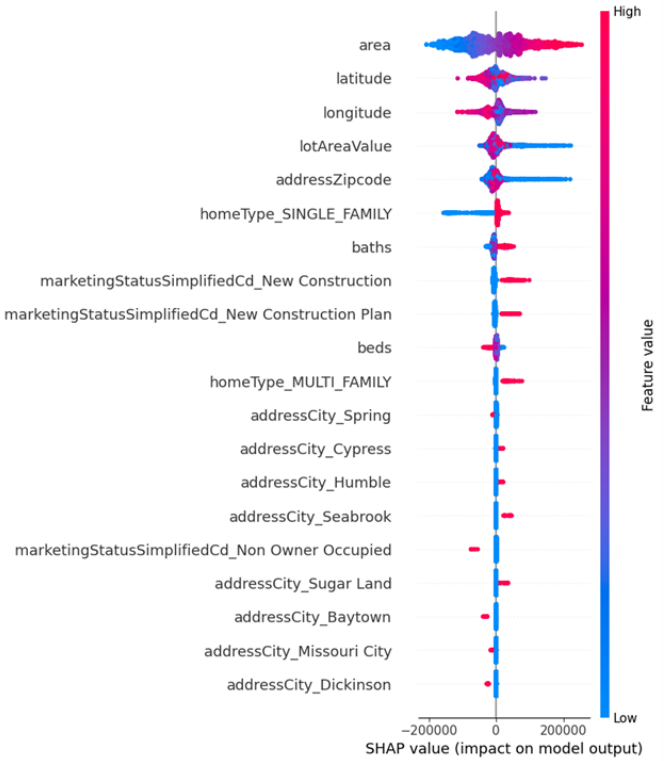

Fig. 7 Feature importance analysis of the best performing model using SHAP values.

Table 7 Comparison of results of each model after removing the column "baths."

Model	MAE	MSE	RMSE	R2	Cross validation
Ridge	69784.42	9.953352e+09	99766.49	0.58	0.57
XGBoost	47027.32	4.995603e+09	70679.58	0.79	0.78
Random Forest	55354.65	6.497342e+09	80606.09	0.73	0.71
Artificial Neural Network	72206.18	1.187671e+10	108980.33	0.50	None
LightGBM	46904.70	4.784892e+09	69172.91	0.80	0.78

Discussion

After comparing the five models, the LightGBM proved to be the best-performing for predicting house prices in the dataset, achieving a coefficient of determination of 81% after hypertuning.

In addition, Table 5 shows the effect of using samples only from the City of Houston on the performance of the models. The Random Forest Regressor had a better performance across all the metrics, reducing the MAE, MSE and RMSE and increasing the coefficient of determination (R^2) from 0.72 to 0.76. Likewise, the LightGBM reduced the MAE and increased the coefficient of determination from .80 to .81 but also increased the MSE and RMSE; hence, there was a reduction in performance for some of its metrics but an increase in others. Also, even though the XGBoost had an increase in MAE, MSE, and RMSE -a higher error- the coefficient of determination remained the same.

Nonetheless, the ridge and neural network faced a significant decrease in performance across all metrics. Those results could indicate that the ensemble tree models -LightGBM, XGBoost and Random Forest Regressor- were not very negatively affected by the change in dimensionality of features, given the presence of other cities increased the dimensionality of features from 15 to 67 (every city was encoded as a feature). This might be because of the nature of the learning process of the ensemble tree models: If there is a feature that does not have a large effect or predictive power, like it might be the case for a specific city, they just ignore it. In contrast, the ridge and neural network models must assign a specific weight to each feature, so they are more affected by a change in dimensionality. Also, the change in the results might be the consequence of reducing the number of samples in the dataset.

Similarly, those results show that the ensemble tree models achieved a lower error and a higher coefficient of determination than the neural network and the ridge model. Given the ensemble models outperformed the others, these results support the findings of E. Antipov and E. Pokryshevskaya's study, which demonstrated the effectiveness of Random Forest, which is an ensemble tree model, for house price predictions[2]. Also, the fact that the LightGBM was the best-performing model is consistent with the results of the study of L. John et al. because their study also found LightGBM as the model that outperformed others[7], although they compared the LightGBM with logistic regression and Random Forest instead of the models that I used.

Likewise, the results from Table 6 show a significant decrease in performance due to the presence of outliers. In fact, the mean absolute error (MAE), mean squared error (MSE) and root mean square error (RMSE) are larger for all the models. This happens because the presence of outliers affects the hypotheses of the models by making them try to learn patterns for uncommon cases, which makes them generalize worse and reduces their performance.

The results from Table 7 show a slight reduction in performance across all the models in MAE, MSE and RMSE except for the Random Forest Regressor, which had a slight increase in performance. The reduction in performance across most of the models shows that the feature baths still has predictive power and carries unique information despite its correlation with the column area.

The results from Figure 7 show that the predictions of the best-performing model were mostly driven by the area and location of the property. In fact, the most influential feature was area, because it was the one that pushed the predicted price up or down the most. Additionally, lotAreaValue had a high influence on the predictions, showing the weight of the property's dimensions on its price. Also, the baths feature seems to have more influence than beds. Similarly, the features addressZipcode, latitude, and longitude had a high influence on the predictions, showing the importance of the location of the property. Furthermore, homeType and marketingStatusSimplifiedCd, which are related to the type of dwellingapartment, house, duplex, etc.and the state of the buildingwhether it is under construction or already builthad some influence on the predictions. On the other hand, features like isUndisclosedAddress, which could provide some information about luxury properties, and addressCity did not have much influence. This might be because there were not many samples per city and many individual cities, so the model did not capture significant differences among them, or that most of the information related to the location of the property was already captured by other features like latitude and zipcode. These results show the possibility of using machine learning models to accurately predict house prices based on features like area, address, longitude, and altitude coordinates in Houston. Furthermore, the model serves as a useful tool for people involved in the real estate industry. For instance, the model could be used by buyers to have an idea of what the property they are interested in could cost, or for realtors to approximately know how much other properties similar to the one they are selling cost. Besides, the model could be improved by adding features or used for comparisons with other models in further research.

The goal of my study was to find a model to predict house prices in Houston with high accuracy, and the LightGBM model successfully fulfilled this objective. However, given the computational limitations, I didn't use the heavier version of the dataset, which contains additional features like the year built and the schools near the property that could enhance the model's performance by making the hypothesis of the model a more complex function. Besides, given the light version of the dataset didn't include descriptions of the properties, they couldn't be used to enhance the performance of the models. Also, the distributions of the latitude and longitude showed that most of the properties were mostly concentrated around latitude 29.7 and 95.4, which is the Houston Downtown area, and that 40% of the samples were from the City of Houston; it indicates that an important part of the samples are from or near the City of Houston. Also, most of the properties were single-family homes with an area of 2500 square feet and 3 or 4 baths, which are properties usually found in suburban areas due to the presence of larger lots available. Therefore, given the sample bias of having properties that are mostly suburban homes, the predictions of the model might have a larger error for properties with a more reduced space such as townhouse properties or condominiums that are usually in denser urban areas. Furthermore, the absence of temporal features, such as the year in which the properties were built, it is not possible to determine possible biases toward properties of certain age in the dataset, which could influence the price of the predictions as well.

Also, the model tends to predict worse when the prices grow over $500k. Nonetheless, it might be explained by the probability density function of the unformatted prices of the properties, which can be seen in Figure 2. The probability density function

for the unformatted prices is right-skewed, with a median price of $359,500 and a mode of $350,000, indicating that most of the houses in the dataset have a price near these values, and there are more houses with prices below the median than above. Consequently, as the prices of the houses get larger, in this case over 500k, there are less samples available for the model to learn, so the predictions for those prices will be further from the real value. Additionally, this does not seem to happen to predictions for prices below the median because there are more samples below the median than above, so there is more data to learn and avoid a reduction in performance.

While the models can reflect to some extent the reality of the housing market, they can also raise some ethical concerns. In fact, the models can sometimes predict far from the real value of the property, which can be misleading to buyers. Also, the models could unintentionally reinforce existing inequalities by reflecting lower prices in certain neighborhoods, which might influence perceptions of their desirability or quality. Therefore, the predictions of the models should be evaluated as probabilistic estimations subject to bias and noise rather than absolute truths when making property-related decisions.

With further research, the models could be modified by adding additional features like the year when the property was built, schools nearby, crime data of the zone, and by analyzing textual descriptions and hyper tuning. In fact, when people consider buying or renting a property, some of the aspects that influence their decision are not characteristics of the property itself, but of the broader area and how it might impact the lifestyle. For example, the year when the property was built and the textual descriptions could provide information about the property's condition and architecture, which influences the price and the type of buyers that it will attract. Similarly, the schools and crime data provide context of the broader area in which the property is located; some schools might be associated with more expensive neighborhoods, while some zones might be less expensive because of a higher criminality. Therefore, given that these features can influence the price of the properties, their addition to the model would mean an increase in its hypothesis' dimensionality, resulting in a more complex hypothesis that might reduce the error of the predictions.

Acknowledgements

I would like to thank my mentor Cassondra Hayes for her guidance in writing the paper and to my teacher Kopatic for helping me with statistical concepts related to the project.

References

1 S. Rosen, *Journal of Political Economy*, 1974, **82**, 34–55.

2 E. A. Antipov and E. B. Pokryshevskaya, *Expert Systems with Applications*, 2012, **39**, 1772–1778.

3 *What is a Zestimate? Zillow's Zestimate accuracy*, https://www.zillow.com/z/zestimate, 2025.

4 Zillow, *Zillow awards $1 million to team that built a better Zestimate*, https://zillow.mediaroom.com/2019-01-30-Zillow-Awards-1-Million-to-Team-that-Built-a-Better-Zestimate, 2019.

5 K. Johnson, *Zillow taps AI to improve its home value estimates*, https://www.wired.com/story/zillow-taps-ai-improve-home-value-estimates/, 2019.

6 S. Humphries, *Introducing a new and improved Zestimate algorithm*, https://www.zillow.com/tech/introducing-a-new-and-improved-zestimate-algorithm/, 2019.

7 L. M. John, R. Shinde, S. Shaikh and D. Ashar, *SSRN*, 2022.

8 S. Bushuyev, D. Bushuiev, D. Kravtsov, N. Poletaev and M. Malaksiano, CEUR Workshop Proceedings, 2024, pp. 20–29.

9 J. Ruan, *Understanding Houston's growth: Real estate new development data analysis*, https://repository.rice.edu/server/api/core/bitstreams/5858cdaa-9d8b-400c-b5ec-53a5f8ee1bd4/content, 2024.

10 U.S. Census Bureau, *Metropolitan and micropolitan statistical areas population totals: 2020-2024*, https://www.census.gov/data/tables/time-series/demo/popest/2020s-total-metro-and-micro-statistical-areas.html, 2025.

11 U.S. Census Bureau, *Some metro areas' population gains reversed COVID-19 era trends*, https://www.census.gov/library/stories/2025/04/metro-area-trends.html, 2025.

12 Redfin, *United States housing market*, https://www.redfin.com/us-housing-market, 2025.

13 Houston Association of Realtors, *Monthly housing update*, https://www.har.com/content/department/mls, 2025.

14 C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Grénard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, *Nature*, 2020, **585**, 357–362.

15 W. McKinney, Proceedings of the 9th Python in Science Conference (SciPy 2010), 2010, pp. 51–56.

16 N. Lekh, *Houston housing market 2024*, https://www.kaggle.com/datasets/datadetective08/houston-housing-market-2024, 2024.

17 J. D. Hunter, *Computing in Science and Engineering*, 2007, **9**, 90–95.

18 M. L. Waskom, *Journal of Open Source Software*, 2021, **6**, year.

19 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and É. Duchesnay, *Journal of Machine Learning Research*, 2011, **12**, 2825–2830.

20 Nicoguaro, *Regularization.svg*, https://commons.wikimedia.org/wiki/File:Regularization.svg, 2016.

21 T. Chen and C. Guestrin, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.

22 M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, `https://arxiv.org/abs/1603.04467`, 2016.

23 G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye and T.-Y. Liu, Advances in Neural Information Processing Systems, 2017, pp. 3146–3154.