Final Report

Hamza Al Bustanji

Introduction

1. Problem Statement:

With the available data from the Ames Housing dataset, can we build a price prediction model to use when assessing the price of a house? And, given that the model is available, can we build an app that allows everyone, regardless of technical background, to access the model?

2. Approach:

The dataset was provided in this Kaggle competition:

(https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/dat)

And is described as follows:

“The Ames Housing dataset was compiled by Dean De Cock for use in data science education. It’s an incredible alternative for data scientists looking for a modernized and expanded version of the often cited Boston Housing Dataset.”

We will develop the app using the Streamlit framework for python. It’s a very accessible way for data scientists to build web applications. For a detailed look into Streamlit check out their launch blog post:

(https://towardsdatascience.com/coding-ml-tools-like-you-code-ml-models-ddba3357eace)

Data Wrangling

We began our project by wrangling our data. We displayed the loaded data frame and suspected the data types. From the first glance at our data frame, we could tell that this is going to be a lengthy process, the data frame contains 81 features including our target feature. Since our goal in this project is to build an accessible app to predict prices, it becomes clear that we have to reduce the number of features we’re going to include. This might come at the cost of a more accurate model, which we’ll have to keep in mind since this is going to be an act of balancing accuracy and accessibility.

We displayed the feature names along with a brief description of each feature and then moved on to inspect missing values. We wrote a function that produced the number, and percentage of missing values in each column. It was clear that we had to handle the-

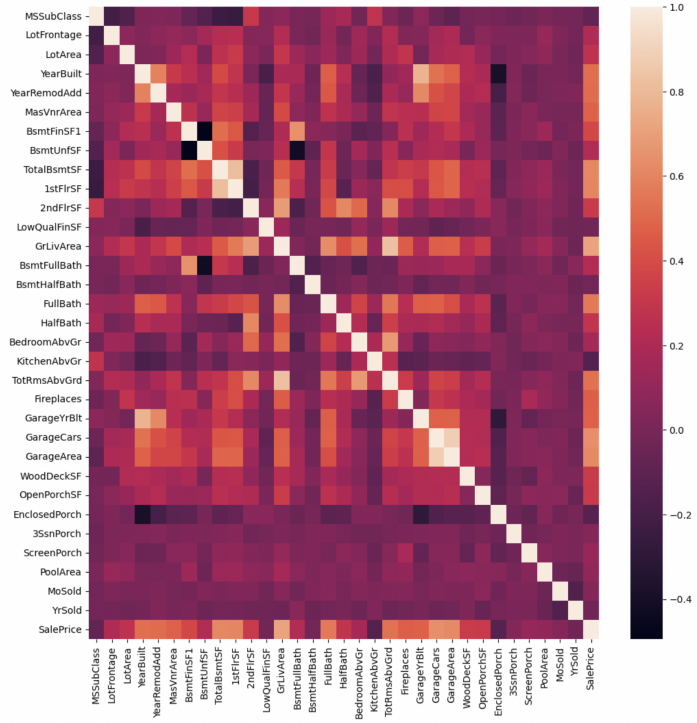
missing values before we can move on. We dropped the columns that were missing

A large percentage of values (above 40%) and we inspected the rest to see what would be the most appropriate way of dealing with the missing values. It became clear after referencing the data description file that most of the missing values in the columns were due to the fact the houses lack those particular features, for example, there were a lot of missing values in the 'Fence' column but the reason the values were missing was that certain houses didn't have fences. This was the logic in dealing with most of the missing values.

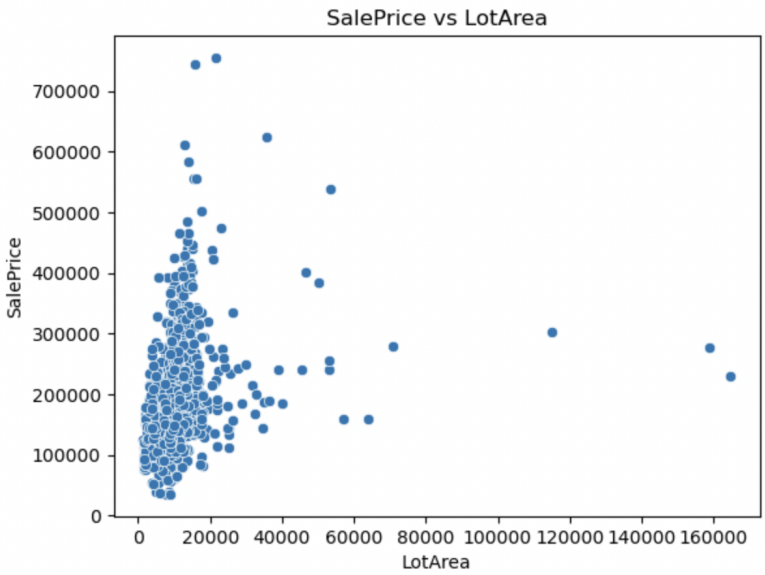
	Null Count	Null Percentage %	DataTypes
PoolQC	1453	99.520548	object
MiscFeature	1406	96.301370	object
Alley	1369	93.767123	object
Fence	1179	80.753425	object
FireplaceQu	690	47.260274	object

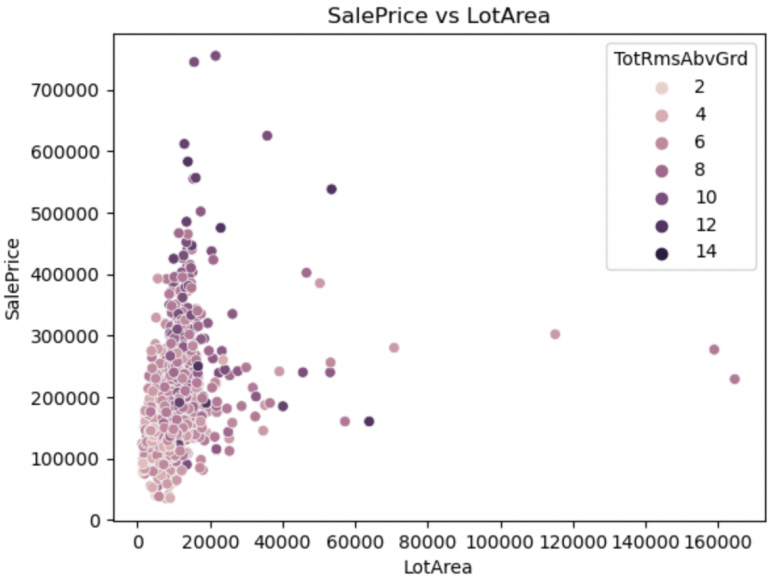
Exploratory Data Analysis

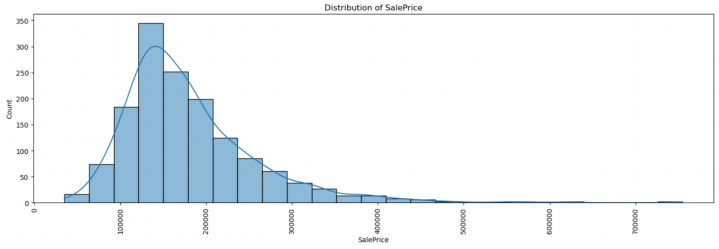
In our Exploratory Data Analysis notebook, we began by producing bar charts for all of the categorical features, which helped us visualize the type of values in each column, and also helped us identify columns with almost no variation in values, meaning columns where almost all the values were the same. We dropped those columns given that they do not provide much information for our goal of trying to predict the house price. Then we moved to inspect the numeric features. We produced summary statistics for the numeric columns and a heat map that displays the correlation between the numeric features. After analyzing the numeric features, we began looking into feature selection for

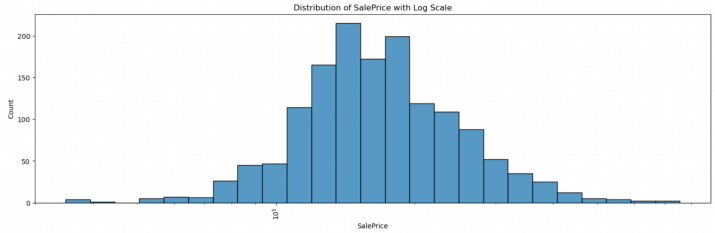


our model. We were particularly interested in how these features relate to the house price, for example, we displayed a scatter plot between the lot area and the house price, which showed that there is a linear relationship between the two features, but that the house price didn't vary as much as the lot area, we suspected that this was because the lot area feature doesn't contain any information about the actual house, for example, how many rooms, how many stories..etc to inspect this we added the number of rooms to the scatter plot as a color dimension, which supported our theory since we could see that the higher the price the more rooms were in a house, and also the houses that had large lot areas but lower prices had fewer rooms. We also engineered new features that we thought might be helpful, such as instead of the quality of a fireplace as a column we had a column that displayed whether a house had a fireplace or not. We also looked at our target feature, the house price, and inspected its distribution. It seemed to be a skewed normal distribution, but after applying a log scale the distribution seemed to follow a normal distribution.









Preprocessing

Data Types:

Firstly, we inspected the data types to make sure that there weren't any miscategorized features, like a categorical feature that is formatted as a numeric feature. This would cause our models to interpret those categorical features as continuous numeric features, which would negatively impact our models' behavior.

Dummy Variables

Next, we created dummy variables for our categorical features. This is an important step because most algorithms can't interpret categorical features as they are, but require creating dummy variables. We used the One-Hot Encoding method and not the Integer Encoding method since we don't want our models to assume an ordinal relationship between the categorical variables.

Train/Test Split

We then created our train/test split, this is essential since we don't want any data leakage and we'd like to confidently validate our models on unseen data.

Scaling

After the train/test split, we proceeded to scale our data. We used a Standard Scaler which we fit on the training predictive features and applied it to both the training and testing predictive features. The reason for only fitting on the training features is that we don't want data leakage.

Modeling

Initial-Not-Even-A-Model

For modeling, we began by creating an initial model that always estimates the mean when predicting values, this would serve as a baseline to judge our models since if they don't perform at least as well as predicting the mean they're useless. The

initial-not-even-a-model had the following results:

Model	R^2	Mean Absolute Error	Root Mean Squared Error
Initial-not-even-a-model	0.0	57266	79533

Linear Models: Elastic Net Model

We trained 3 linear models; a Linear Regression model, a Polynomial model, and an Elastic Net model. Out of the three, the only one that performed relatively well is the Elastic Net Model. The Linear Regression and the Polynomial model produced estimates that were unreasonably off. The reason for this could be one or a combination of the following:

- Linearity: The relationship between the independent variables and the dependent variable is not linear.
- Multicollinearity: The independent variables are highly correlated with each other.
- The presence of outliers.

We attributed the reason for the higher performance of the Elastic Net Model to the fact that it was mostly a Lasso Regression Model, which adds a regularization term that shrinks the coefficients of unnecessary features down to zero.

Support Vector Regression Model

Support Vector Regression (SVR) is a type of Support Vector Machine (SVM) algorithm, which is a supervised learning algorithm that can be used for regression problems. SVR is a linear model that aims to find the hyperplane that maximally separates the data points into two classes, while at the same time minimizing the classification error. In SVR, the goal is to find the hyperplane that maximally separates the data points from the prediction error, while at the same time minimizing the margin of deviation between the predicted value and the true value of the dependent variable. We performed a grid search that arrived at the best combination of hyperparameters for the model. The best combination of hyperparameters was the following:

{'C': 10000, 'epsilon': 10000, 'gamma': 'scale', 'kernel': 'linear'}

Random Forest Regression Model

Random Forest is a tree-based regression model. In our implementation, we performed a grid search to find the best number of decision trees to use in our model. And according to our grid search, the best number of decision trees to use was 183.

Comparison

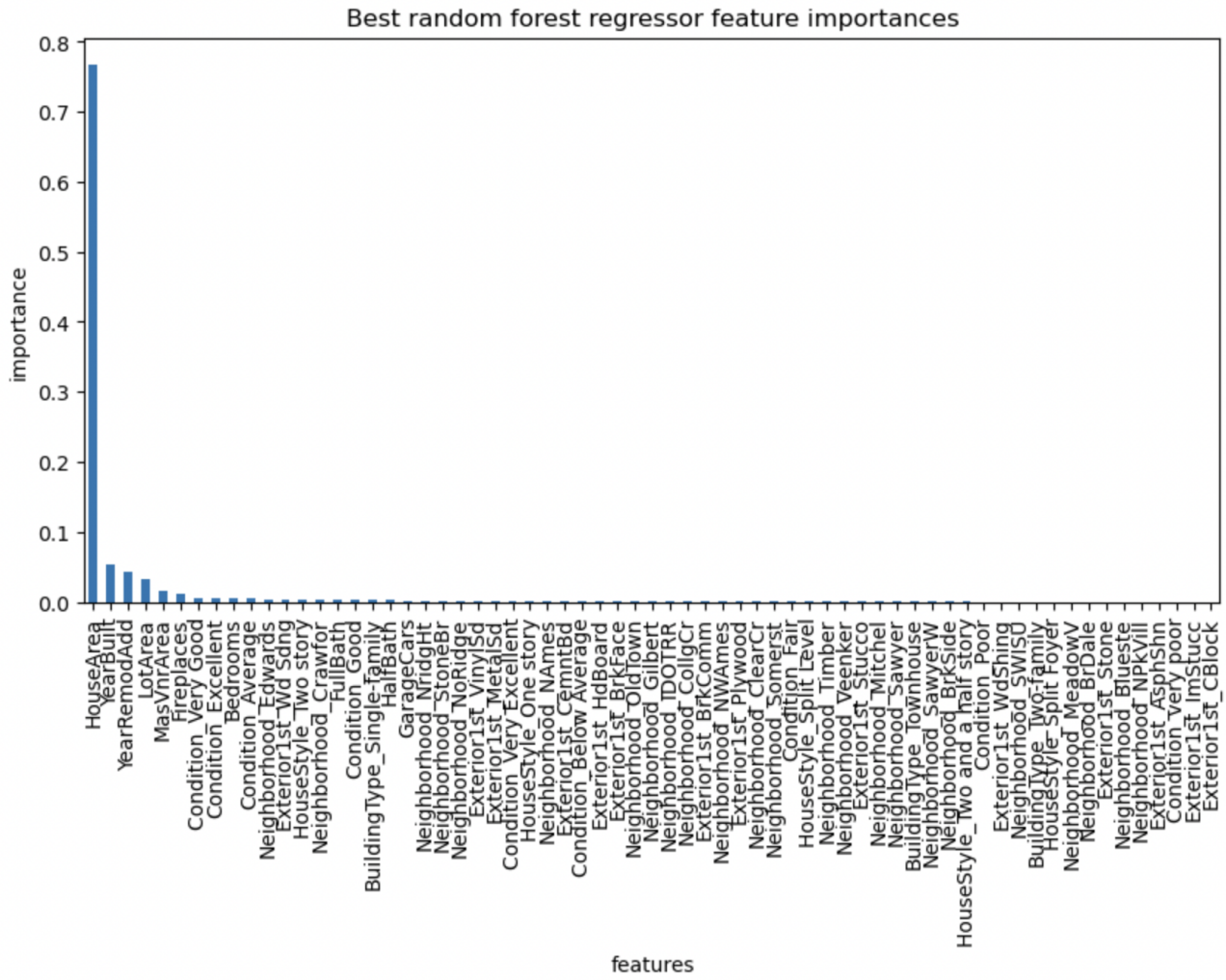
Now that we have our models evaluated, it's time to compare them using our evaluation metrics: R^2, MAE, and RMSE:

Model	R^2	MAE	RMSE
Elastic Net	0.699	20971	42735
Support Vector Regression	0.686	20621	43630
Random Forrest	0.808	20757	34105

We can see that overall the best performing model was the Random Forest model, which we chose to move forward with.

Random Forest Model Exploration

We then looked a bit deeper into the Random Forest Model and displayed its feature importance: A



App

After saving our Random Forest model we began working on our Streamlit app. The process was straightforward, we just needed to ask the user for the house properties which we then processed and fed to our model then displayed the predicted price back to our user. For a detailed look at the Python code please visit the app folder.