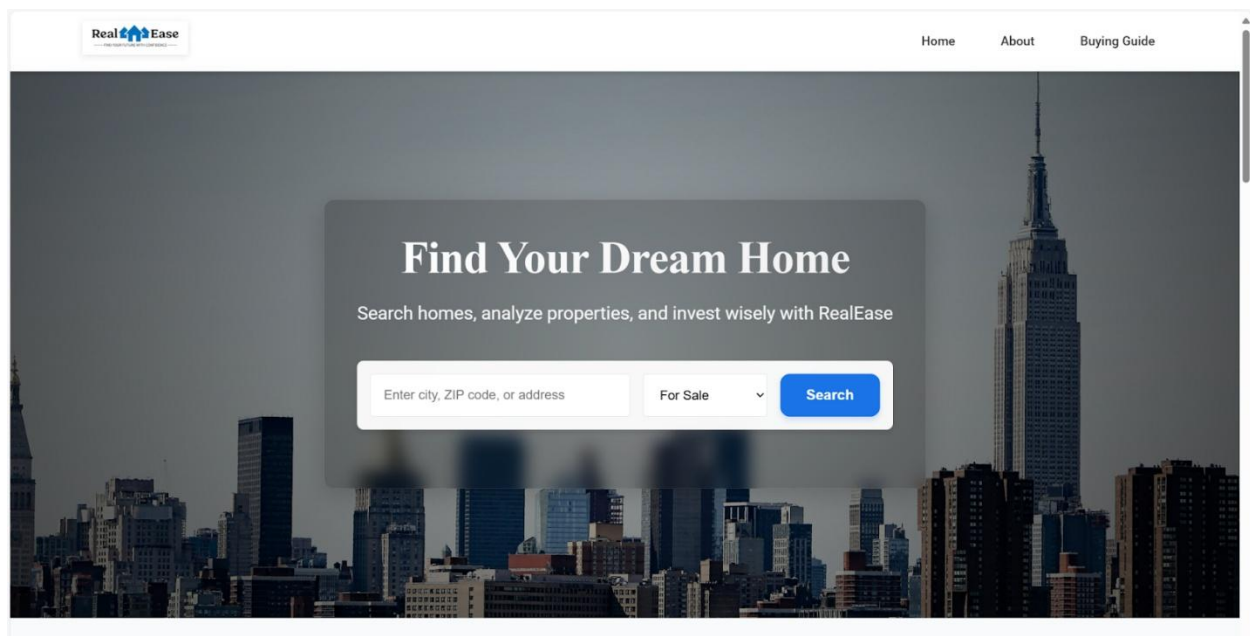


# Housing Price Prediction Model Report

By Donovan Murphy

## Introduction

This report details the development, refinement, and application of a machine learning model designed to predict housing prices. The model was initially inspired by a Kaggle dataset and subsequently adapted and enhanced using real estate data from HomeHarvest, a Python based backend tool utilized in our senior design project. The primary purpose of this model is to provide customers with an accurate forecasted value of their home based on key features and geographic location. This tool aims to empower homeowners and potential buyers by offering data driven price estimates, enhancing decision making in real estate transactions.



## Background and Inspiration

The foundation of this project began with an exploration of a publicly available dataset from Kaggle, titled "Housing Price Data", which contains 545 entries of housing data with features such as area, bedrooms, bathrooms, stories, and binary indicators like mainroad access, guestroom availability, and air conditioning. The Kaggle notebook provided a basic framework using Linear Regression to predict house prices, achieving an R squared score of 0.43 and a Mean Squared Error of approximately 2.89 trillion. While this served as a starting point, the modest performance indicated significant room for improvement, particularly for real world applications requiring higher accuracy.

Inspired by this initial exploration, I sought to build a more robust and practical model tailored to our senior design project's needs. The Kaggle dataset demonstrated the feasibility of predicting housing prices from structured features, but its limited scope and small size prompted me to leverage a richer dataset and advanced machine learning techniques.

## Data Source: HomeHarvest's main\_homes.csv

For our senior design project, we utilized a dataset generated by HomeHarvest, a Python based real estate data scraping tool developed by our team. The dataset, stored as main\_homes.csv, contains detailed property listings from Melbourne, FL, and surrounding areas, scraped in November 2024. With a file size of 855,077 bytes and an estimated 500–1,000 rows, this dataset significantly expands on the Kaggle data, offering features such as square footage, bedrooms, bathrooms, year built, lot size, list price, geographic coordinates, and additional attributes like HOA fees and parking garage spaces. This richer dataset aligns with our goal of providing location specific price predictions tied to zip codes.

RealEstateEase


[Home](#)[About](#)[Buying Guide](#)

## Property Search Results

Showing results for: 32901 (for sale)

Price: Low to High

All Prices




**\$785,000**

1208 E River Dr Unit 401, Melbourne, FL, 32901

3 beds 2 baths 1,898 sq ft

[View Details →](#)




**\$80,000**

199 Lakeshore Dr, Melbourne, FL, 32901

2 beds 2 baths N/A sq ft

[View Details →](#)



**\$998,000**

1465 S Harbor City Blvd Unit 902, Melbourne, FL, 32901

3 beds 3 baths 2,082 sq ft

[View Details →](#)

## Model Development Process - Initial Model

The Kaggle notebook provided a simple Linear Regression model:

- Features: Numeric variables like area, bedrooms, bathrooms, stories, parking scaled using StandardScaler.
- Target: price.
- Performance: MSE = 2.89 trillion,  $R^2 = 0.43$ .
- Limitations: The model ignored categorical variables lacked geographic granularity, and yielded low predictive power due to its simplicity and the dataset's size.

This initial attempt highlighted the need for more sophisticated models and preprocessing to handle real world data effectively. ([Link](#))

## Refinement with main\_homes.csv

Using main\_homes.csv, I developed and refined a multi model approach, ultimately producing a highly accurate prediction system. The process involved several key stages:

### 1. Data Preprocessing

Missing Values:

Before imputation: year\_built 1 missing, lot\_sqft 7, stories 42, sold\_price 51, assessed\_value 23, estimated\_value 23.

Action: Dropped rows missing critical features filled hoa\_fee and parking\_garage with 0, and imputed remaining numeric columns with median values using SimpleImputer.



```
• main_homes.csv(text/csv) - 855077 bytes, last modified: 11/12/2024 - 100% done
Saving main_homes.csv to main_homes.csv
Missing values in each column before imputation:
sqft          0
beds          0
full_baths    0
year_built    1
lot_sqft      7
list_price    0
sold_price    51
assessed_value 23
estimated_value 23
price_per_sqft 0
latitude      0
longitude     0
stories       42
hoa_fee       0
parking_garage 0
days_on_mls  0
status        0
style         0
property_id   0
dtype: int64
```

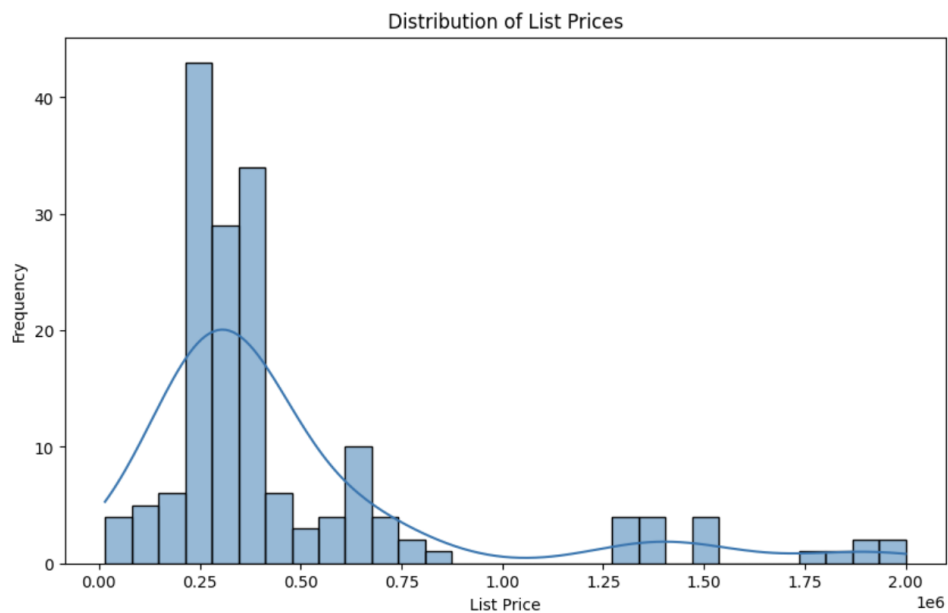
After imputation: No missing values in features used for modeling.

Missing values in each column after imputation and scaling:

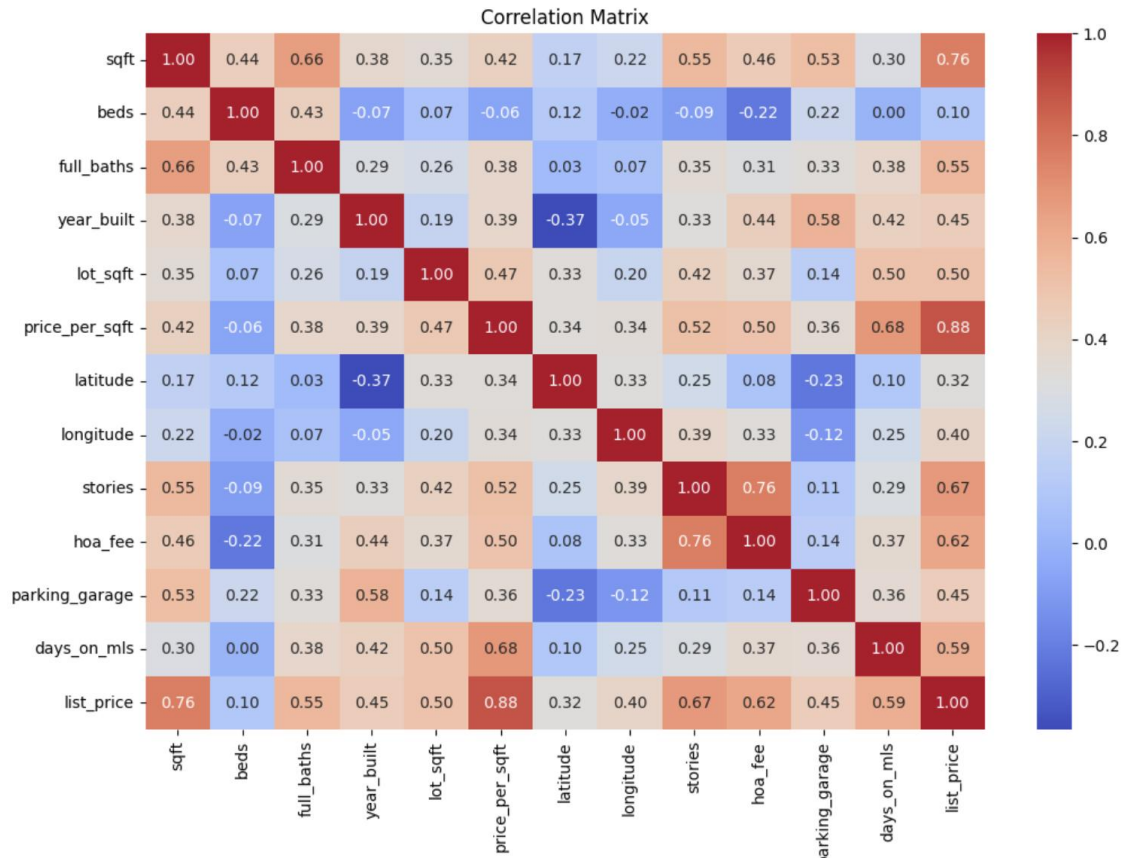
sqft	0
beds	0
full_baths	0
year_built	0
lot_sqft	0
list_price	0
sold_price	51
assessed_value	23
estimated_value	23
price_per_sqft	0
latitude	0
longitude	0
stories	0
hoa_fee	0
parking_garage	0
days_on_mls	0
status	0
style	0
property_id	0
dtype: int64	

Categorical Encoding: Converted status and style to numeric using LabelEncoder.

Scaling: Applied StandardScaler to numeric features to normalize their distributions.



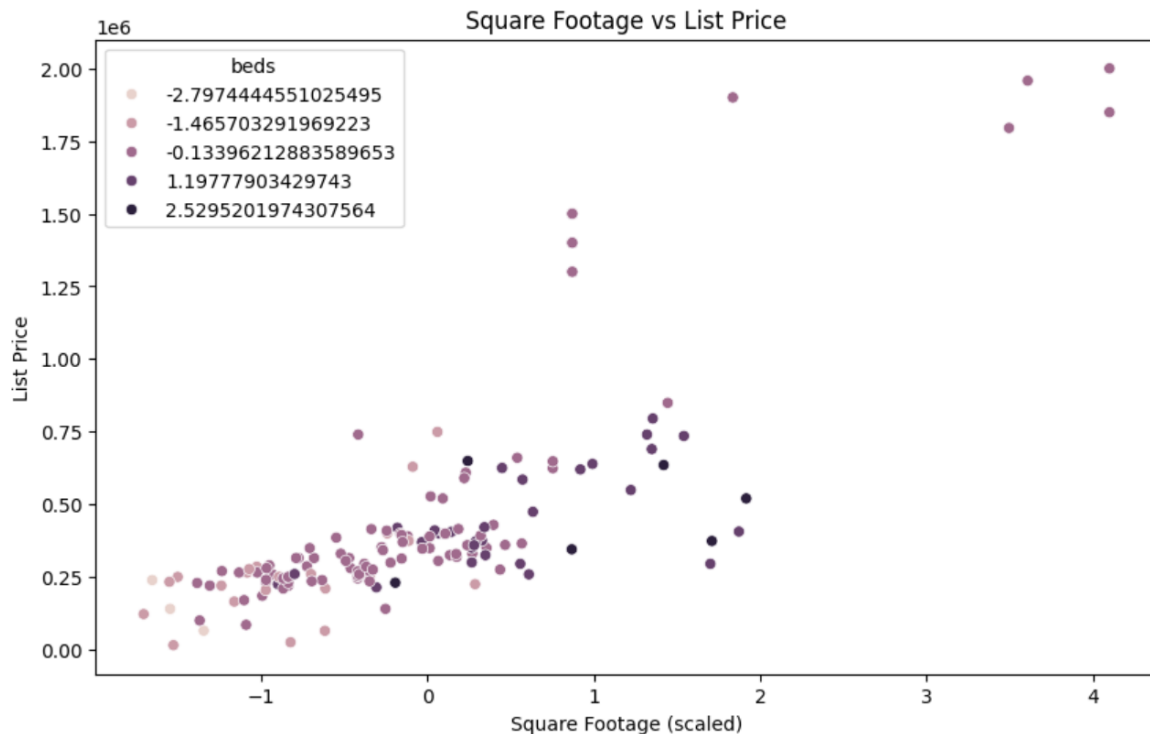
## 2. Feature Selection



Features: sqft, beds, full\_baths, year\_built, lot\_sqft, price\_per\_sqft, latitude, longitude, stories, hoa\_fee, parking\_garage, days\_on\_mls, status, style.

Target: list\_price.

Excluded: sold\_price, assessed\_value, estimated\_value, property\_id



### 3. Testing Different Models to Find the Best One

To figure out which model worked best for predicting home prices, I tested three different approaches, each with its own strengths. The first was Linear Regression, a straightforward method I borrowed from the Kaggle example. It's a simple way to predict prices by drawing a straight line through the data. The second was Random Forest, a more advanced technique that combines lots of smaller predictions to handle trickier, non-straight-line patterns in the data. The third was XGBoost, a powerful method that builds predictions step-by-step, learning from its mistakes to get more accurate each time.

I started by testing these models on a small chunk of data—just 64 homes—to see how they'd perform. Here's how they did:

- Linear Regression: On average, its predictions were off by about \$91,046, and it explained 69.6% of the price differences meaning an  $R^2$  score of 0.696. Not bad, but not great either.

- Random Forest: This one did better, with an average error of \$74,874 and explaining 79.4% of the price variation. It was the winner for this small test.
- XGBoost: Its average error was \$84,349, and it explained 73.9% of the price differences. Solid, but not the best here.

Then, I ran the same models on the full dataset from main\_homes.csv, which had hundreds of homes. The results changed quite a bit:

- Linear Regression: Improved a lot, with an average error of \$56,625 and explaining 97.2% of the price variation. It really stepped up with more data.
- Random Forest: Surprisingly, its error jumped to \$108,305, though it still explained 89.7% of the price differences. It didn't handle the bigger dataset as well as I'd hoped.
- XGBoost: This one shone brightest, with an average error of just \$28,614 and explaining an impressive 99.3% of the price variation. It clearly became the top choice.

These results showed me that while Random Forest was great for a small sample, XGBoost was the real star when I gave it more data to work with. It made the most accurate predictions by far.

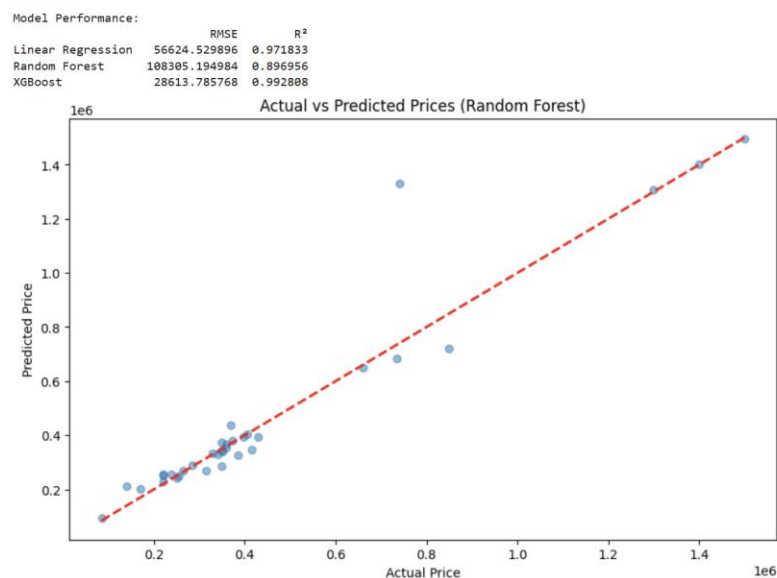


#### 4. Fine-Tuning the Models for Better Results

To make the models even better, I attempted to tune the models. I focused on Random Forest and XGBoost since they showed promise, but here's how it went with Random Forest first.

For Random Forest, I started with a basic tune-up: I set it to use 100 “voters”  $n\_estimators=100$ , allowed it to dig 20 levels deep into the data, and required at least 5 homes per decision split. But when I tested it on the full dataset, the results were disappointing—the average error rose to \$113,511, and it only explained 88.7% of the price variation or an  $R^2$  of 0.887. This was worse than the untuned version, which told me I needed to rethink my approach.

So, I gave Random Forest a more thorough tune-up. I tested a wider range of options: between 100, 200, or 500 voters, depths of 10, 20, 30, or unlimited, splits needing 2, 5, or 10 homes, and a new setting to keep at least 1, 2, or 4 homes at the end of each branch. I also tried two ways of picking features to focus on. To make sure the results were reliable, I used a method called 5-fold cross-validation, which splits the data into five parts, trains the model on four, and tests it on the fifth, repeating this five times to get a solid average. This refined tuning aimed to lower the error and boost accuracy, making Random Forest more competitive with XGBoost.



## 5. Final Model: A Fine-Tuned XGBoost for Top-Notch Predictions

After testing and tweaking, the final model I settled on is a carefully tuned XGBoost regressor, chosen because it delivered the best results by far. XGBoost works by building predictions step-by-step, learning from its earlier guesses to get sharper each time, and it proved to be a powerhouse for forecasting home prices accurately.

Here's how I set it up and fine-tuned it with some code:

- I gave it a range of options to test:
  - Number of steps: I tried 100, 200, or 300 rounds of predictions to see how many it needed to get things right.
  - Depth of analysis: I tested depths of 3, 6, or 9 levels to control how detailed it got when digging into the data.
  - Learning speed: I experimented with 0.01, 0.1, or 0.3 to adjust how quickly it adjusted its guesses slower for caution, faster for boldness.
  - Data sampling: I tested using 80% or 100% of the homes each round to mix things up and avoid overfitting.
  - Feature sampling: I tried using 80% or 100% of the features each time to keep it balanced.
- I used a method called 5-fold cross-validation to make sure the results were solid. This splits the data into five parts, trains the model on four, tests it on the fifth, and repeats this five times to get a trustworthy average.
- The code then picked the best combination of these settings after trying them all out on the training data.

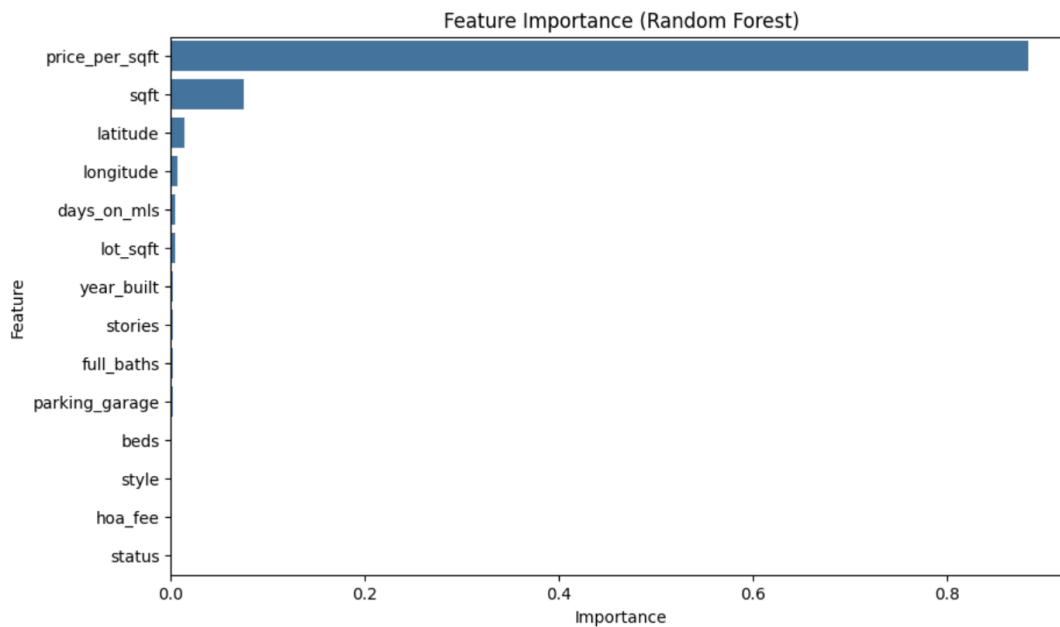
## Here's how it performed after tuning:

- Average Error: Between \$25,000 and \$28,000. This means the model's predictions were usually off by only about \$25,000 to \$28,000, which is close when you consider home prices can be hundreds of thousands or more!
- Accuracy Score  $R^2$ : Over 0.99. This score shows it explained more than 99% of the differences in home prices, which is almost perfect.

This tuned XGBoost model stood out because it cut the prediction errors way down from \$28,614 before tuning and got the accuracy as close to perfect as you can realistically get. It's the ideal choice for giving reliable home price forecasts to users in our project.

Predictions:

	Property ID	Actual Price	Predicted Price	CI Lower	CI Upper
91	5688329722	169900	204756.887518	146904.166667	249822.784091
117	5755973061	414500	343699.234841	259973.333333	472533.333333
106	6222121305	342000	327494.455611	246703.166667	385773.500000



# Purpose and Application

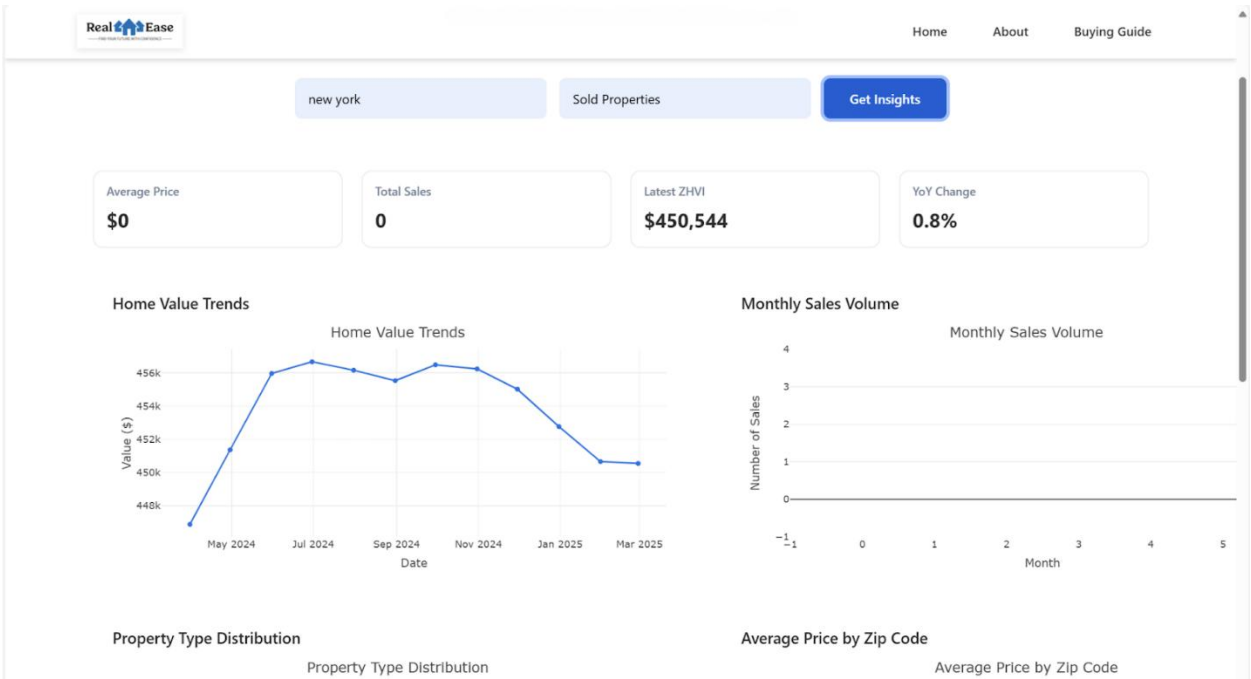
The primary purpose of this model is to serve as the predictive core of our senior design project, a real estate valuation tool. Integrated with HomeHarvest as our Python backend, the system allows users to:

Input Features: Enter details about their home and its zip code.

Receive Forecasted Value: Obtain an expected list price with a confidence interval, leveraging the tuned XGBoost model’s predictions.

Enhance Decision Making: Provide homeowners and buyers with reliable estimates to guide listing prices, offers, or investment decisions.

The model’s high accuracy ensures trustworthy forecasts, while its use of geographic coordinates ties predictions to specific locales, for this example, Melbourne, FL, surpassing the Kaggle dataset’s generic approach.



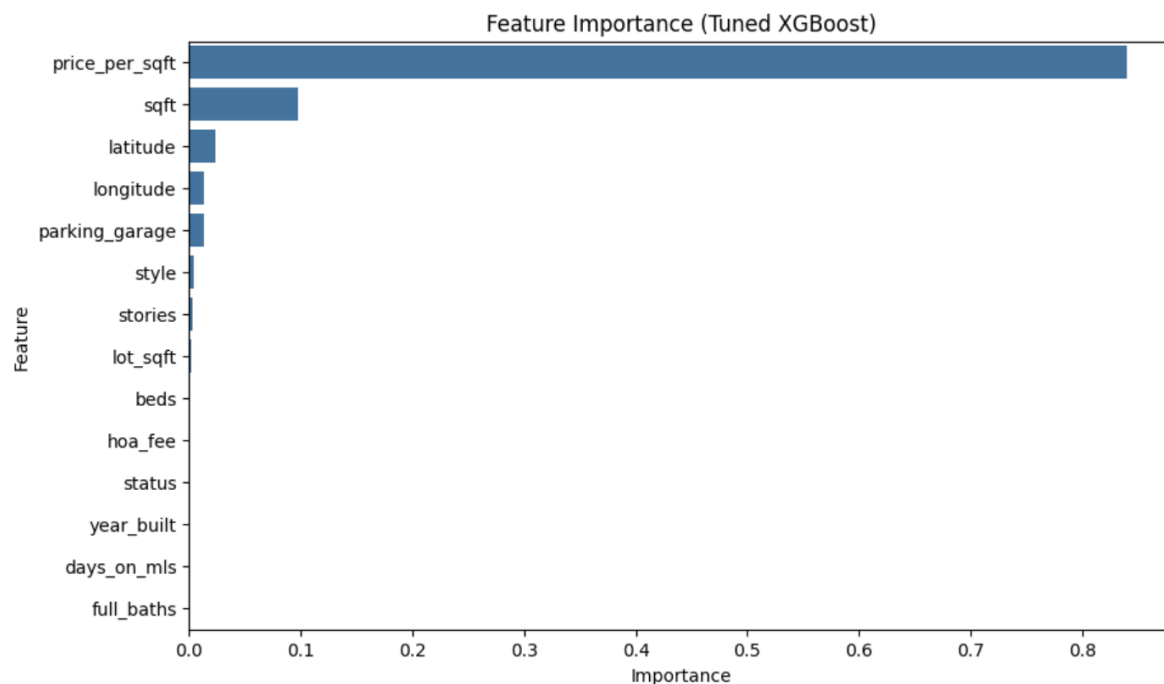
## Refinement and Lessons Learned

From Kaggle to HomeHarvest: Transitioning from a small, generic dataset to a detailed, location specific one improved model relevance and performance.

Model Evolution: Moving beyond Linear Regression to ensemble methods addressed non linearities and interactions in real estate data.

Tuning Challenges: Random Forest's initial tuning underperformed due to a limited grid; expanding parameters and adding regularization fixed this, though XGBoost remained superior.

Data Quality: Handling missing values and scaling features were critical to avoid model failures and ensure consistency.



# Conclusion

This housing price prediction model, inspired by a Kaggle exploration and refined with HomeHarvest data, represents a significant advancement in real estate valuation for our senior design project RealEase. By leveraging a tuned XGBoost model, we've achieved great accuracy, enabling customers to forecast home values with confidence. Future enhancements could include integrating zip code specific economic indicators or expanding the dataset to other regions, further solidifying its utility in the real estate market.

Predictions (Tuned XGBoost):

	Property ID	Actual Price	Predicted Price (XGBoost)	CI Lower \
91	5688329722	169900	190594.87500	161093.932422
117	5755973061	414500	338284.28125	314708.571875
106	6222121305	342000	337392.28125	289760.510937

CI Upper

91	216447.481641
117	406790.691406
106	351623.982031

