REAL ESTATE PRICE PREDICTON PROJECT

MLG 382 CYO Project

Group H

Bianca Grobler 600537
Adolph Jacobus van Coller 601005
Cayden Frank 578131
Renaldo Jardim 601333

Contents

Executive summary

This project presents a machine learning solution for predicting real estate prices using a comprehensive dataset that includes property characteristics, geographic location, and market trends. By leveraging advanced data analysis techniques and predictive modelling, the project aims to support informed decision-making for stakeholders such as property buyers, investors, and realtors.

Multiple regression-based algorithms were tested, with XGBoost yielding the best results in terms of prediction accuracy. The trained model is integrated into a user-friendly web application developed with

DASH and deployed on a live server via Render. The application enables users to input property features and receive instant price predictions.

The project showcases a full ML pipeline—data collection, preprocessing, model development, web app creation, and deployment. It also highlights lessons learned and potential improvements, setting the stage for future enhancements such as dynamic data integration and feature importance visualization.

Links

GitHub Repository

https://github.com/CL-Frank/MLG382_CYOProject

Render App

https://real-estate-price-predicter.onrender.com

Project Objectives

- To identify a suitable dataset for real estate price prediction.
- To perform thorough exploratory data analysis (EDA).
- To preprocess the data for modelling.
- To apply and compare multiple machine learning algorithms.
- To fine-tune and evaluate the best-performing model.
- To create an interactive web application using DASH.
- To deploy the application on a live server.
- To document the entire process and present key insights.

Dataset Research

Publicly available datasets from platforms like Kaggle and government repositories were reviewed. The selected dataset includes property features such as location, number of rooms, area in square feet, year built, proximity to amenities, and market trends.

Description of variables

Variable	**Description**
date	The date the house was listed or sold. Format: YYYY-MM-DD.
price	The sale price of the house (in USD). This is the target variable for prediction.
bedrooms	Number of bedrooms in the house.

bathrooms	Number of bathrooms (includes partial baths as decimal values).
sqft_living	Total interior living space in square feet.
sqft_lot	Total size of the lot in square feet.
floors	Number of floors in the house (includes half-floors as decimal values).
waterfront	Binary indicator if the property has a waterfront view (1 = yes, 0 = no).
view	Quality of the view from the house (0 to 4 scale, higher is better).
condition	Overall condition of the house (1 to 5 scale, higher is better).
sqft_above	Square footage of the house **excluding** the basement.
sqft_basement	Square footage of the **basement** (if any).
yr_built	Year the house was originally built.
yr_renovated	Year the house was last renovated. A value of 0 means it was never renovated.
street	The street address of the property.
city	City in which the property is located.
statezip	State and ZIP code in a combined format (e.g., WA 98103).
country	Country where the property is located (all are USA in this dataset).

System Setup and Data Loading

Environment Setup

Python 3.x environment was used within a Jupyter Notebook. Common machine learning and data science libraries were employed, including Pandas, NumPy, Scikit-learn, XGBoost, TensorFlow/Keras, and Dash.

Package Installation

Installed and imported packages:

- pandas
- numpy
- matplotlib
- seaborn

- sklearn
- xgboost
- keras (from tensorflow.keras)
- dash

Data Import

The dataset Student_performance_data.csv was loaded into a pandas DataFrame using pd.read_csv().

Cleaning up data

Outliers were removed using the inter quartile range for the price, as well as houses without any price.

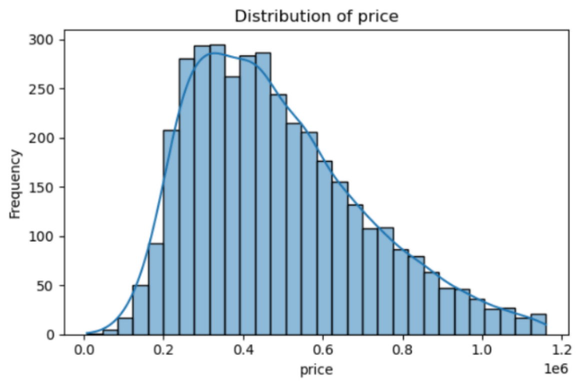
Exploratory Data Analysis

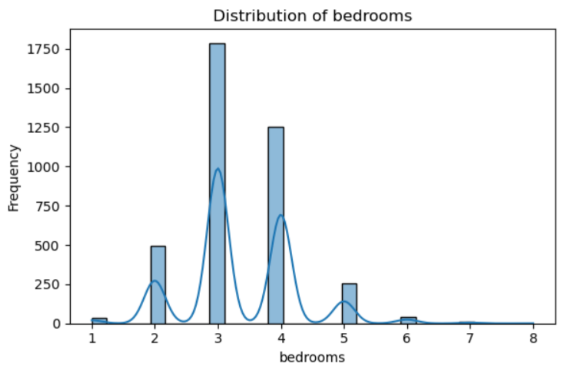
Getting information on the dataset

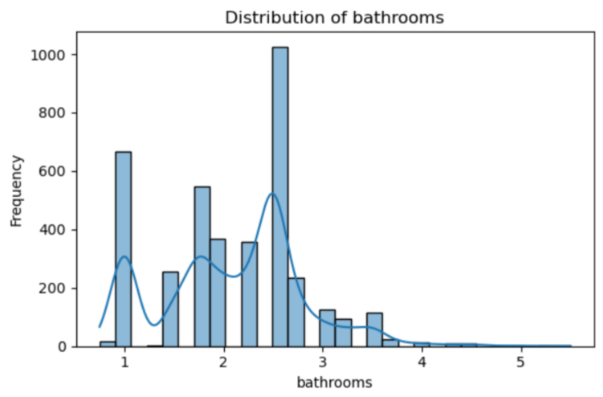
Information on the dataset was acquired from methods such as df.describe() and df.info().

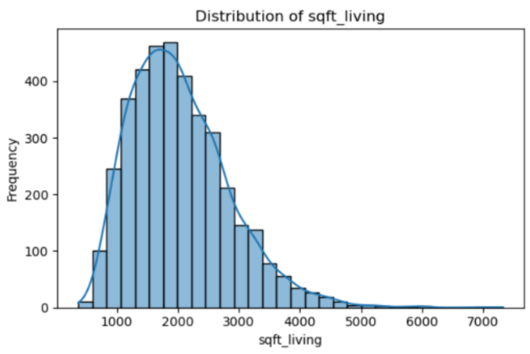
Univariate Analysis

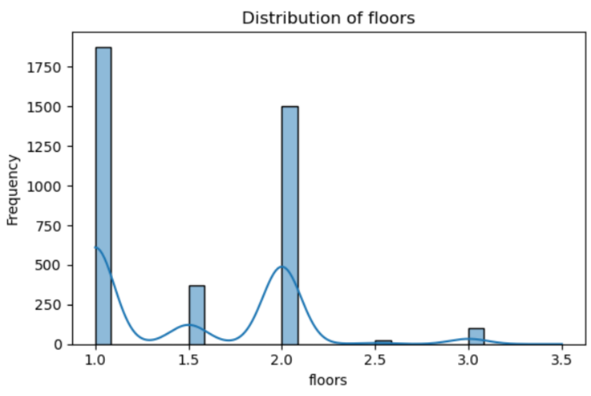
Analysis included graphs of each variable's distribution

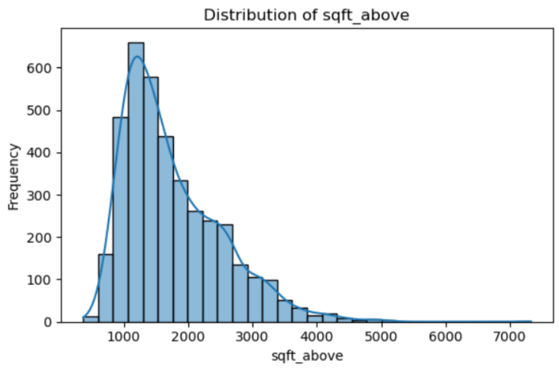


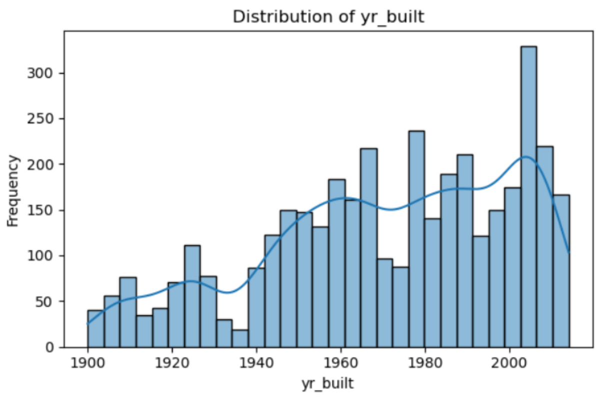






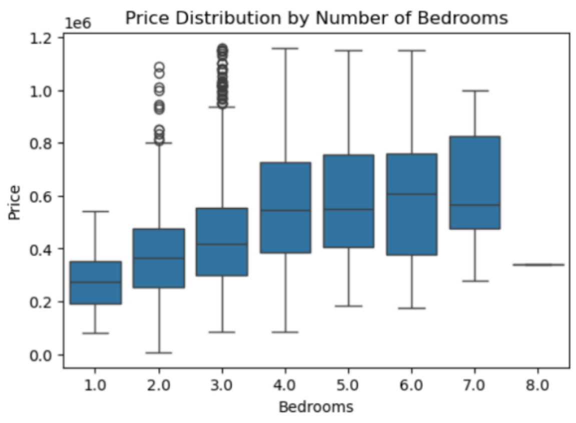


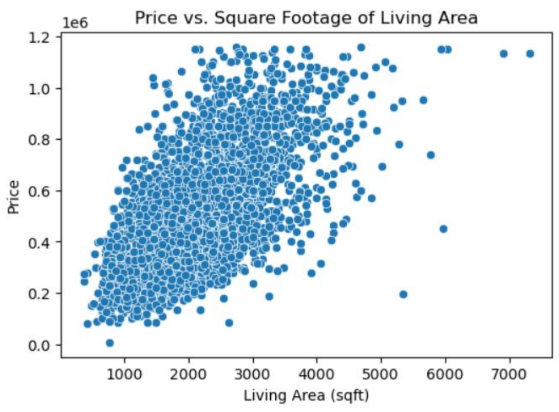


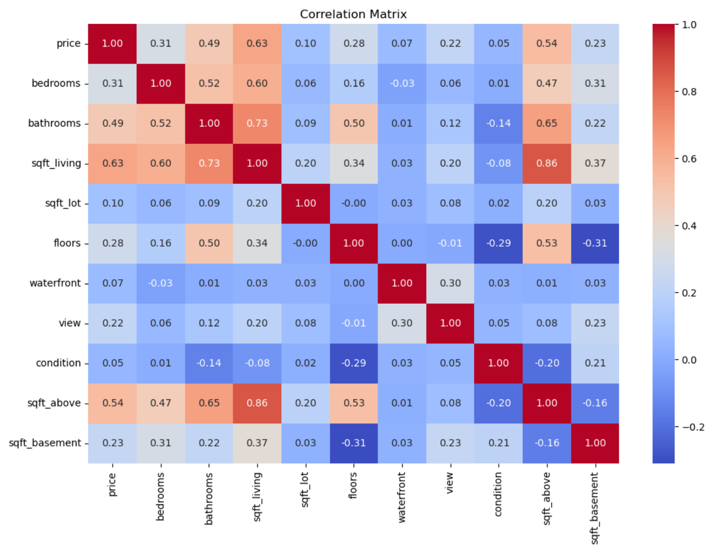


Bivariate analysis

Bivariate analysis mainly compared the price of the house to the different features of the house, as well as a correlation matrix.







Data preprocessing

Null values and outliers were already removed in the beginning. Unneeded columns were dropped such as 'date', 'street', 'country', 'sqft_lot', and 'sqft_above' as these fields were unnecessary as other fields already represented them and since all the houses in the dataset were from USA.

Features were scaled and variables such as city and zip code were encoded.

Feature engineering

To enhance the predictive power of the dataset and improve model performance, several new features were engineered from the existing variables:

- **house_age**: A new feature representing the age of the house was created by subtracting the year built (yr_built) from 2014, which is the year of the dataset. This provides a normalized way to assess how old a property is.
- **is_renovated**: A binary indicator was derived from yr_renovated to show whether a house had undergone renovation. A value of 1 indicates the house was renovated (i.e., yr_renovated > 0), and 0 otherwise. This simplifies renovation status for modelling purposes.
- **sqft_living_bathrooms**: An interaction term was introduced by multiplying the total living area (sqft_living) with the number of bathrooms (bathrooms). This feature aims to capture the combined effect of space and amenity availability on house value.
- **zipcode**: The ZIP code was extracted as a numeric variable from the combined statezip field using regular expressions. This allows for geographical segmentation and analysis.

After creating these new features, the original columns yr_built, yr_renovated, and statezip were dropped to avoid redundancy and multicollinearity. The transformed dataset was then saved as a CSV file for further analysis and modelling.

Model Building

Model Selection

Model	Type	Reason for Selection
Linear Regression	Linear / Baseline	Simple and interpretable; serves as a baseline to compare against more complex models.
Ridge Regression	Linear (L2 Regularized)	Handles multicollinearity and reduces overfitting by penalizing large coefficients.
Lasso Regression	Linear (L1 Regularized)	Performs feature selection by shrinking some coefficients to zero, aiding interpretability.
Decision Tree	Non-linear / Tree-based	Captures non-linear relationships and interactions; easy to visualize but can overfit.
Random Forest	Ensemble (Bagging)	Combines multiple decision trees to improve accuracy and generalization; robust to overfitting.
Gradient Boosting	Ensemble (Boosting)	Builds models sequentially to reduce error, typically more accurate but sensitive to tuning.
XGBoost	Advanced Boosting	Highly efficient and regularized version of gradient boosting; performs well on structured data.

All selected models were trained using the engineered dataset to predict housing prices. The performance of each model was evaluated on a separate test set using three key metrics:

- **Root Mean Squared Error (RMSE)**: Measures the average magnitude of prediction errors. Lower values indicate better performance.
- **Mean Absolute Error (MAE)**: Represents the average absolute difference between predicted and actual values.
- **R-squared (R^2)**: Indicates the proportion of variance in the target variable explained by the model. Higher values suggest better explanatory power.

The results of the model evaluations (on unscaled data) are summarized below:

Model	RMSE	MAE	R^2 Score
Linear Regression	118,750.21	74,468.33	0.7062
Ridge Regression	115,965.01	74,028.12	0.7198

Lasso Regression	172,013.67	136,388.48	0.3835
Decision Tree	155,157.68	105,830.44	0.4984
Random Forest	124,932.75	83,212.47	0.6748
Gradient Boosting	127,608.19	90,349.22	0.6607
XGBoost	118,341.34	75,529.46	0.7082

Hyperparameter Tuning of Tree-Based Models

Following the initial evaluations, Random Forest and XGBoost demonstrated strong performance, making them prime candidates for further enhancement through hyperparameter tuning. This process aimed to optimize each model's configuration for improved predictive accuracy and generalization.

Hyperparameters control model behavior—like tree depth, number of trees, and learning rate—and are not learned from data. By tuning them systematically using GridSearchCV, we identified the most effective combinations to minimize prediction error.

Random Forest Tuning

We tuned the following parameters:

- n_estimators: [100, 200, 300]
- max_depth: [None, 10, 20]
- min_samples_split: [2, 5]

The search was performed using 5-fold cross-validation with negative mean squared error as the scoring metric.

Optimized Random Forest Performance:

- **RMSE**: 123,337.28
- **MAE**: 81,979.62
- **R²**: 0.6831

XGBoost Tuning

For XGBoost, a broader range of parameters was tuned:

- n_estimators: [100, 200, 300]
- learning_rate: [0.01, 0.1, 0.2]
- max_depth: [3, 5, 7]
- subsample: [0.8, 1.0]
- colsample_bytree: [0.8, 1.0]

Again, GridSearchCV with 5-fold CV was used to identify the best configuration.

Optimized XGBoost Performance:

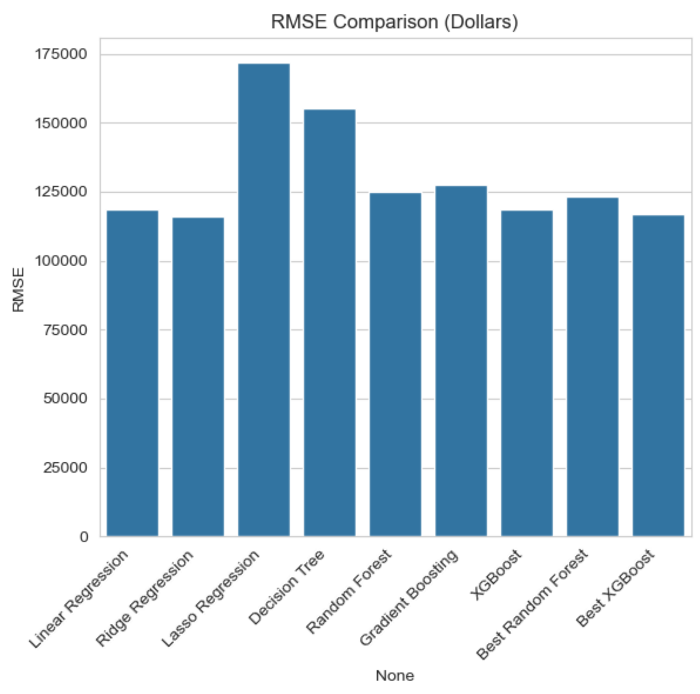
- **RMSE**: 116,772.44
- **MAE**: 73,300.57
- **R²**: 0.7159

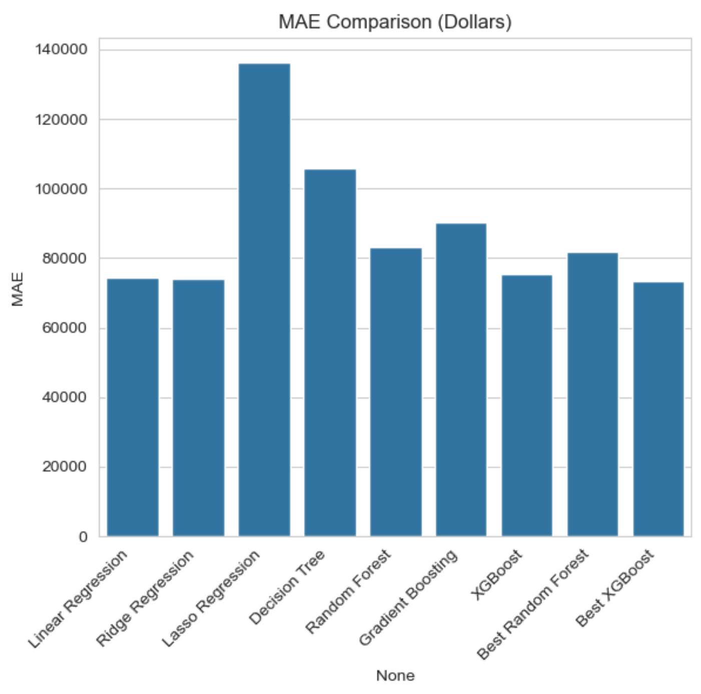
Model Comparison

After training and evaluating a range of regression models—both linear and tree-based—we compared their performance using three key metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R²). These metrics provide insight into how accurate and reliable each model is in predicting house prices.

Model Comparison Summary

Model	**RMSE**	**MAE**	**R²**
Ridge Regression	**115,965.01**	74,028.12	**0.7198**
Best XGBoost	116,772.44	**73,300.57**	0.7159
XGBoost	118,341.34	75,529.46	0.7082
Linear Regression	118,750.21	74,468.33	0.7062
Best Random Forest	123,337.28	81,979.62	0.6831
Random Forest	124,932.75	83,212.47	0.6748
Gradient Boosting	127,608.19	90,349.22	0.6607
Decision Tree	155,157.68	105,830.44	0.4984
Lasso Regression	172,013.67	136,388.48	0.3835





Why Best XGBoost Was Chosen

While Ridge Regression achieved the lowest RMSE and highest R^2 score by a very small margin, Best XGBoost was ultimately selected as the final model for the following reasons:

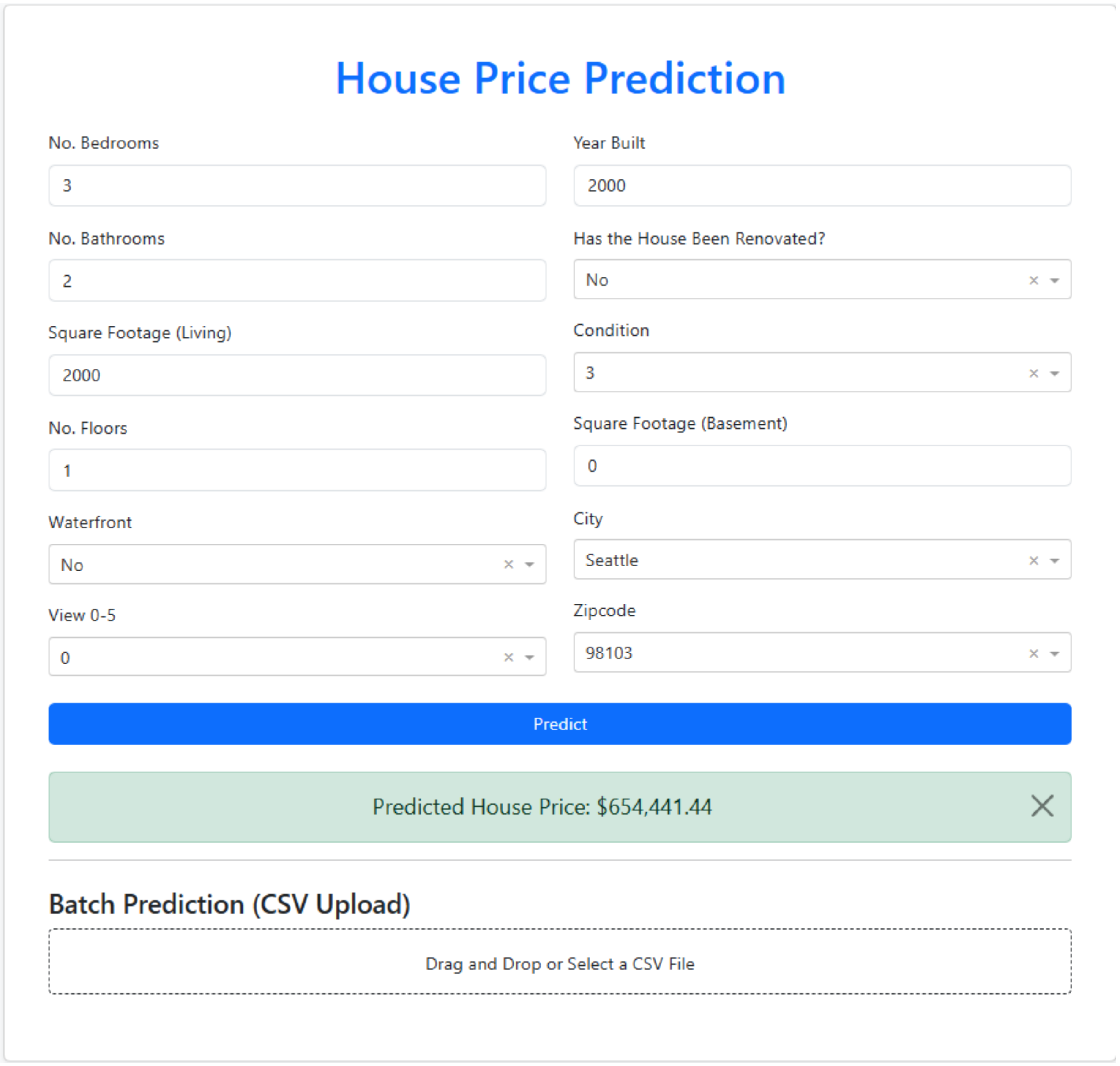
- **Consistent High Performance**: It ranked at or near the top across all metrics, especially MAE, where it outperformed Ridge Regression.
- **Nonlinear Modeling Capabilities**: XGBoost is a gradient boosting algorithm capable of capturing complex, nonlinear relationships in the data—something linear models may miss.
- **Tuning Flexibility**: The hyperparameter tuning process further enhanced XGBoost's performance, ensuring the model is both optimized and generalizable.
- **Robustness**: XGBoost is known for its regularization techniques and resistance to overfitting, making it a reliable choice for predictive tasks.

As a result, the best XGBoost model was selected as the best-performing and most suitable model for predicting house prices in this study.

Web Application Development with DASH

A DASH-based web app was created to serve the trained model.

Features: User-friendly interface, form input for features, dynamic prediction display and CSV upload for bulk predicting.

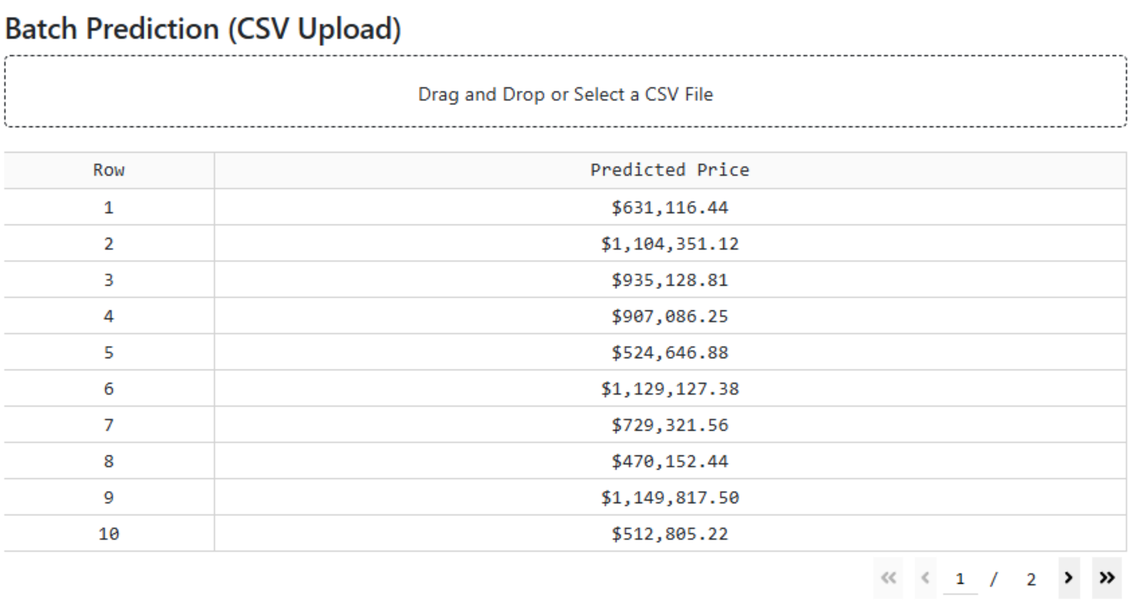


House Price Prediction

No. Bedrooms	3
Year Built	2000
No. Bathrooms	2
Has the House Been Renovated?	No
Square Footage (Living)	2000
Condition	3
No. Floors	1
Square Footage (Basement)	0
Waterfront	No
City	Seattle
View 0-5	0
Zipcode	98103

Predicted House Price: $654,441.44

Batch Prediction (CSV Upload)



Batch Prediction (CSV Upload)

Row	Predicted Price
1	$631,116.44
2	$1,104,351.12
3	$935,128.81
4	$907,086.25
5	$524,646.88
6	$1,129,127.38
7	$729,321.56
8	$470,152.44
9	$1,149,817.50
10	$512,805.22

Model Deployment

The Dash app is deployed on **Render**, a cloud platform suitable for hosting web apps with server-side logic. Deployment involved the following steps:

1. **Project Structure Setup**:
   - App files (e.g., app.py) are placed in the src/ folder.
   - Model artifacts (e.g., .pkl, .h5) are stored in an artifacts/ directory.
   - A requirements.txt file lists all dependencies like Dash, scikit-learn, TensorFlow, and XGBoost.
2. **Model Path Handling**:
   - Relative file paths are used with os.path to locate model files during server runtime.
   - Render's file system is case-sensitive and sandboxed, so exact directory matching is essential.
3. **Execution Flow**:
   - The app uses app.run() with use_reloader=False to prevent issues with double-loading models during deployment.
   - Environment setup is automated via the requirements.txt.
4. **Live Access**:
   - Once deployed, the app is accessible via a public Render URL, making it easy to demonstrate predictions to stakeholders or testers.

Reflection and Future Work

Valuable experience gained in EDA, model development, and deployment.

Future enhancements could include a feature importance dashboard and integration with real-time APIs for dynamic market data.

Conclusion

This project successfully delivered a comprehensive machine learning pipeline for real estate price prediction, from data collection and preprocessing to model selection, evaluation, and deployment. Among the tested models, **XGBoost** emerged as the most suitable choice due to its balance of performance, flexibility, and robustness. While Ridge

Regression showed slightly better R^2 and RMSE metrics, XGBoost offered superior capabilities in capturing complex, nonlinear relationships and exhibited excellent generalization through hyperparameter tuning.

The deployment of the trained model via a DASH web application on Render demonstrates the practical application of machine learning in real-world scenarios. Users can interact with the model to obtain real-time housing price predictions, making this tool valuable for potential buyers, investors, and real estate professionals.