Comparative Analysis of Machine Learning Models for Residential Property Valuation

Ben Gronbach [1] and Ryan Smith [2]

[1] School of Engineering and Computer Science

[2] Eberhart School of Business

University of the Pacific

Stockton, California

October 2025

Abstract

The monetary value of a home in the United States is an intricate calculation consisting of macroeconomic as well as microeconomic variables. Factors such as size, physical characteristics, comparable sales, and even the welfare of the economy can influence the prices of homes. By utilizing the pattern-recognition abilities of AI, this project aims to build a machine learning model that can accurately predict the prices of homes.

Accuracy is evaluated using mathematical metrics such as Mean Absolute Error (MAE) which measures the average error of predictions, and R^2 score, which measures how well the model's predictions match the actual data, showing the proportion of variance explained. A diverse variety of regression models are used, such as Random Forest Regressor, Gradient Boosting Regressor, and Support Vector Regression to accomplish this goal.

Models are trained and tested on a comprehensive data set of 545 sample homes, noting 13 different features, being: price, area, bedrooms, bathrooms, stories, main road, guestroom, basement, hot water heating, air-conditioning, parking, preferred area, and furnishing status. These values represent a mix of numerical and categorical data, posing a challenge for models to incorporate different data types into a single output.

Model 1: Decision Tree Regressor

A decision tree regression model is a supervised machine learning model, meaning that it examines training data and forms decision nodes to split the data into different categories. Unlike a typical decision tree, however, the regression model outputs numerical values, rather than classifications. The depth of the decision tree is variable, meaning that you can choose the number of splits the model uses to sort through the data. However, setting the depth too high runs the risk of overfitting the data, which complicates the process to the point where the model is unable to pick up patterns and relies solely on training data.

Python Code

Imports

import os import numpy as np import pandas as pd import matplotlib.pyplot as plt import kagglehub

from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder from sklearn.pipeline import Pipeline from sklearn.model_selection import train_test_split from sklearn.tree import DecisionTreeRegressor, plot_tree from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

Load data

path = kagglehub.dataset_download("yasserh/housing-prices-dataset") csv_path = os.path.join(path, "Housing.csv") df = pd.read_csv(csv_path)

Define target values

target = "price" y = df[target] X = df.drop(columns=[target])

Sort Columns

numeric_cols = X.select_dtypes(include=["number"]).columns.tolist() categorical_cols = X.select_dtypes(include=["object", "category", "bool"]).columns.tolist()

Preprocess Data

preprocess = ColumnTransformer(transformers=[("num", "passthrough", numeric_cols), ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols),], remainder="drop",)

Train/test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

Train model

reg = DecisionTreeRegressor(max_depth=4, random_state=42) pipe = Pipeline([("prep", preprocess), ("model", reg)]) pipe.fit(X_train, y_train)

Evaluate Accuracy metrics

pred = pipe.predict(X_test) r2 = r2_score(y_test, pred) mae = mean_absolute_error(y_test, pred) rmse = np.sqrt(mean_squared_error(y_test, pred))

print("----- Decision Tree Regressor (Test Set) -----") print(f"R^2: {r2:.3f}") print(f"MAE: {mae:,.0f}") print(f"RMSE: {rmse:,.0f}")

Plot Decision Tree visual

model = pipe.named_steps["model"] ohe = pipe.named_steps["prep"].named_transformers_["cat"]

if len(categorical_cols): cat_names = ohe.get_feature_names_out(categorical_cols) else: cat_names = np.array([])

feature_names = np.concatenate([numeric_cols, cat_names])

plt.figure(figsize=(18, 10)) plot_tree(model, feature_names=feature_names, filled=True, rounded=True, fontsize=8) plt.title("Decision Tree Regressor (max_depth=4)") plt.tight_layout()

Plot Predicted Prices vs Target Prices

plt.figure(figsize=(7, 7)) plt.scatter(y_test, pred, alpha=0.6, edgecolor="k") plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], "r--", lw=2, label="Perfect Prediction") plt.title("Predicted vs Actual House Prices") plt.xlabel("Actual Price (In Dollars*10^6)") plt.ylabel("Predicted Price (In Dollars*10^6)") plt.legend() plt.grid(True, linestyle="--", alpha=0.5) plt.tight_layout()

plt.show()

Results

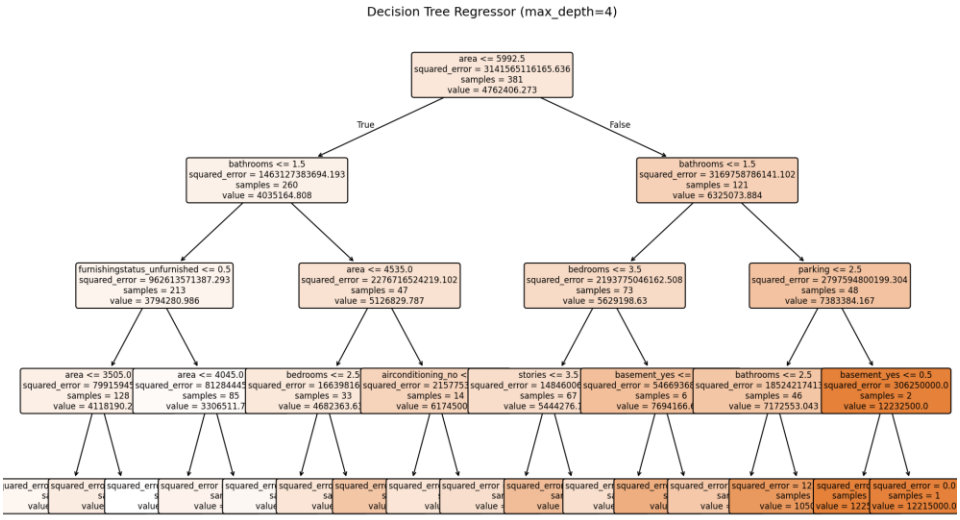

Figure 1. Decision Tree Visual Category Split

```
----- Decision Tree Regressor (Test Set) -----
R²:   0.316
MAE:  1,212,650
RMSE: 1,715,941
```

Figure 2. Decision Tree Accuracy Metrics

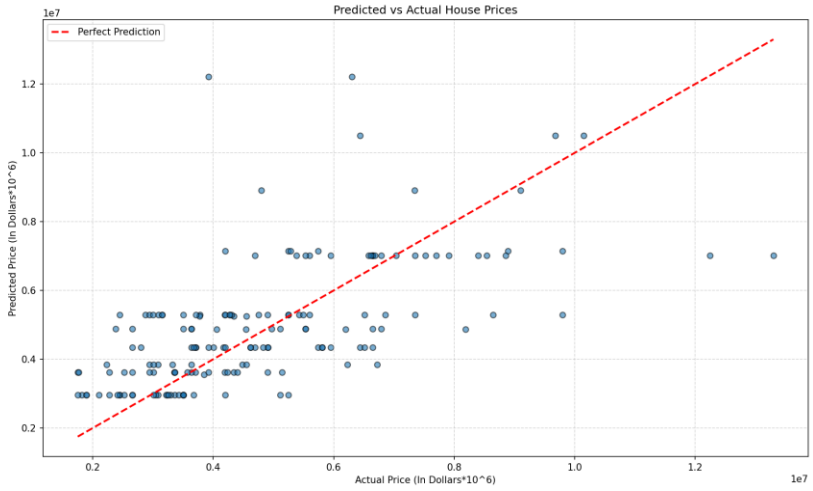

Figure 3. Predicted Prices vs Target Prices (max depth =4)

Max Depth Value	R^2	MAE	RMSE
3	0.257	1,287,592	1,788,461
4	0.316	1,212,650	1,715,941
5	0.306	1,202,231	1,729,355

Figure 4. Accuracy Metrics of Different Depth Decision Trees

Discussion

While the decision tree provides a clear and interpretable visual (see figure 1), the categorical nature of the algorithm results in inaccurate predictions. The model's predictions resulted in a low R^2 value of 0.316, indicating that the predicted values show a high degree of variance to the target values. Additionally, the model displayed a high mean absolute error of 1,212,650, meaning that the model was, on average, ~1.2 million dollars off the target value. In the context of estimating housing prices, this error is massive, rendering this specific model inviable in a real-world scenario. Furthermore, the high root mean squared value of 1,715,941 also implies that there are a large number of outliers in the predictions with very high variance to the target data.

Multiple depth values were tested, with a max depth of 4 achieving the highest degree of success, however it is still notably inaccurate. With a maximum depth of 4, the model only has 16 different values; it can label each point as (See figure 1). This is apparent in the plot in Figure 3, as the data points are separated into a set number of different y

values. In a numerical quantification model, this poses an issue as it limits the dispersion of data in a manner similar to a categorical model. Due to the many features represented in this data set, accurate analysis requires a more complex process, taking more variables into account than a single decision tree can offer. Therefore, the next section will explore the abilities of a random forest regressor.

Model 2: Random Forest Regression

The random forest regression model is a supervised machine learning algorithm that combines outputs of multiple decision trees to output a prediction with improved accuracy and stability. Rather than using one single decision tree, the random forest regression model generates multiple decision trees using random groups of data and then averages their outputs to make a final numerical prediction. This leads to deeper analysis of data compared to a single tree, as the model takes a higher number of features into account when determining a final output. Additionally, this ensemble approach decreases the risk of overfitting since the model offers multiple perspectives on the data. To control the complexity and performance of the model, we can adjust the depth and number of the trees in the random forest regression model.

Python Code

Imports

import pandas as pd import numpy as np import matplotlib.pyplot as plt from sklearn.model_selection import train_test_split from sklearn.ensemble import RandomForestRegressor from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score from sklearn.preprocessing import OneHotEncoder from sklearn.compose import ColumnTransformer from sklearn.pipeline import Pipeline

Load dataset

file_path = '/Users/ryansmith/Desktop/housing.csv' df = pd.read_csv(file_path) df = df.dropna()

Define target values

target_column = 'price' # change this if your CSV uses a different column name X = df.drop(columns=[target_column]) y = df[target_column]

Sort Columns

cat_cols = X.select_dtypes(include=['object', 'category']).columns num_cols = X.select_dtypes(include=['number']).columns

Preprocess Data

preprocessor = ColumnTransformer(transformers=[('num', 'passthrough', num_cols), ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)])

Create Random Forest model

rf = Pipeline(steps=[('preprocessor', preprocessor), ('regressor', RandomForestRegressor(n_estimators=200, random_state=42, n_jobs=-1))])

Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Train the model

print("Training Random Forest Model...") rf.fit(X_train, y_train)

Predict on test data

y_pred = rf.predict(X_test)

Evaluate performance

r2 = r2_score(y_test, y_pred) mae = mean_absolute_error(y_test, y_pred) rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"\n Model Evaluation:") print(f"R^2 Score: {r2:.3f}") print(f"MAE: {mae:,.2f}") print(f"RMSE: {rmse:,.2f}")

Easy-to-read scatter plot

plt.figure(figsize=(7, 7)) plt.scatter(y_test, y_pred, color='skyblue', edgecolor='k', alpha=0.6, label='Predicted Houses')

Add red "perfect prediction" line

max_val = max(max(y_test), max(y_pred)) min_val = min(min(y_test), min(y_pred)) plt.plot([min_val, max_val], [min_val, max_val], color='red', linewidth=2, label='Perfect Prediction')

plt.title(f'Actual vs Predicted House Prices\n$R^2$={r2:.3f}, MAE=${mae:,.0f}, RMSE=${rmse:,.0f}', fontsize=13, fontweight='bold') plt.xlabel('Actual Price', fontsize=12) plt.ylabel('Predicted Price', fontsize=12) plt.legend() plt.grid(True, linestyle='--', alpha=0.6) plt.tight_layout() plt.show()

Results

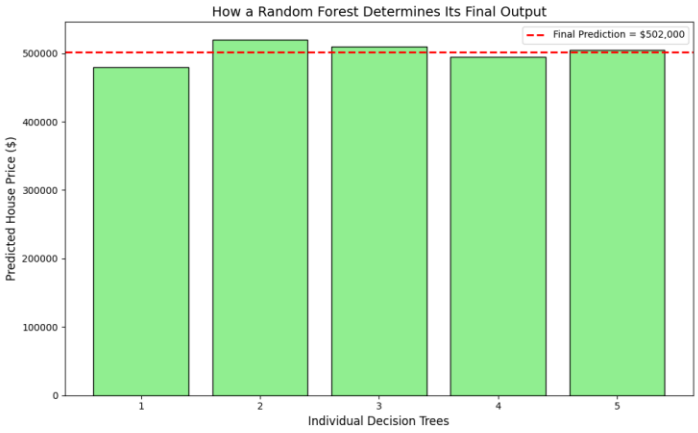

Figure 5. How a Random Forest Determines its Final Output (red line is the average)

```
✅ Model Evaluation:
R² Score: 0.613
MAE: 1,015,097.18
RMSE: 1,398,658.02
```

Figure 6. Random Forest Accuracy Metrics

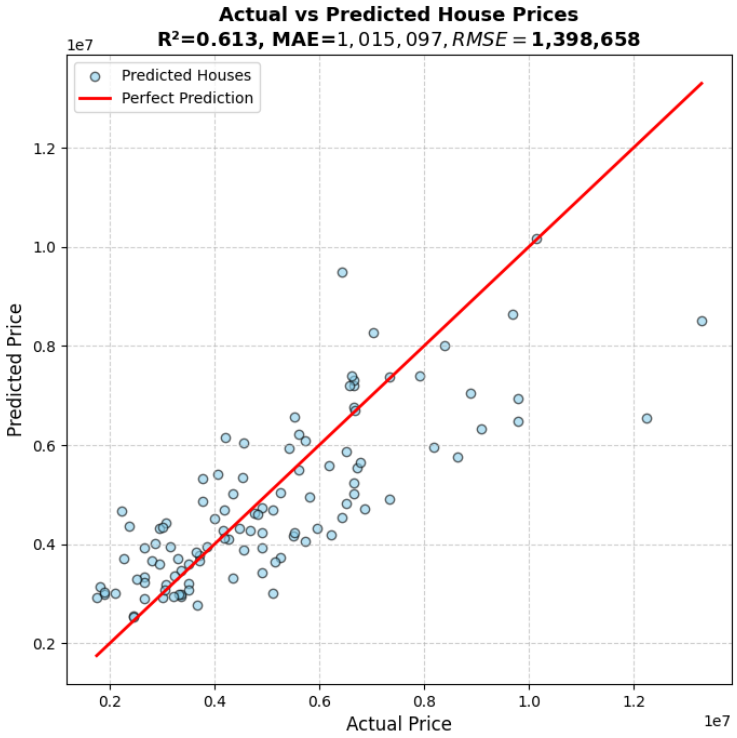

Figure 7. Actual vs. Predicted House Prices- Random Forest Regression Model

Discussion

While the random forest model offers a higher degree of accuracy compared to a single decision tree, the performance still shows limitations. The model achieved an R^2 value of 0.613, indicating a 38.7% random variance in housing prices. The model's mean absolute error (MAE) was 1,015,097.18, which tells us that the predictions, on average, were ~1 million dollars off the actual value. This error is still substantial when estimating housing prices. Similarly, the root mean squared error (RMSE) of 1,398,658.02 indicates a considerable degree of outliers with high variation between predicted and actual prices. Considering using this in real estate valuation is unlikely due to the level of error being too large for practical use, meaning that the model would need further refinement to achieve a more reliable and accurate performance.

The results from figures 4, 5, and 6 are from the Random Forest Regression model that only uses 200 decision trees. Increasing the number of decision trees results in more accurate outputs. Starting with 10,000 decision trees and finishing at 100,000 decision trees, the accuracy values obtained eventually plateaued. The end results using 100,000 decisions trees gave an R^2 value of 0.621, and MAE value of 1,009,493.07 and a RMSE value of 1,383,758.81.

# of Decision Trees	R^2 value	MAE value	RMSE value
200	0.613	1,015,097.18	1,398,658.02
10,000	0.621	1,010,382.58	1,384,932.80
100,000	0.621	1,009,493.07	1,383,758.81

Model 3: Gradient Boosting Regression

A Gradient Boosting Regression model is a supervised machine learning model that uses multiple weak prediction models to create an ensemble model that will significantly improve prediction accuracy. Gradient Boosting works sequentially by training each new decision tree to correct errors that come from the combined ensemble of all the previously built trees. The model can recognize where it performs poorly, and it can optimize and push the prediction closer to the actual value. This ability is known as "Boosting" which focuses on defining the poorly predicted data points and refines the model's performance progressively. Compared to a single decision tree, Gradient Boosting can more effectively capture complex, non-linear relationships. To control the Gradient Boosting models' learning speed, adjust parameters such as learning rate and number of trees to fine tune the complexity of the model.

Python Code

Imports

import pandas as pd import numpy as np import matplotlib.pyplot as plt from sklearn.model_selection import train_test_split from sklearn.ensemble import GradientBoostingRegressor from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

Load data

df = pd.read_csv('Housing.csv')

Data Preprocessing

Convert binary categorical features to 0/1

binary_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea'] mapping = {'yes': 1, 'no': 0}

Fix: Iterate and apply map, then explicitly cast to integer to avoid FutureWarning

for col in binary_cols: df[col] = df[col].map(mapping).astype(int)

One-hot encode the 'furnishingstatus' column

df = pd.get_dummies(df, columns=['furnishingstatus'], drop_first=True)

Separate features (X) and target (y)

X = df.drop('price', axis=1) y = df['price']

Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Train the Gradient Boosting Regressor model

gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42) gbr.fit(X_train, y_train)

Predict on the test set

y_pred = gbr.predict(X_test)

Calculate and display accuracy metrics

r2 = r2_score(y_test, y_pred) mae = mean_absolute_error(y_test, y_pred) rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"R-squared (R^2): {r2:.4f}") print(f"Mean Absolute Error (MAE): {mae:.2f}") print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

Plot Visual (Actual vs. Predicted Prices)

plt.figure(figsize=(10, 6)) plt.scatter(y_test, y_pred, alpha=0.6) plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2) plt.title('Gradient Boosting Regressor: Actual vs. Predicted Housing Prices') plt.xlabel('Actual Price') plt.ylabel('Predicted Price') plt.grid(True) plt.savefig('actual_vs_predicted_housing_prices_fixed.png')

Results

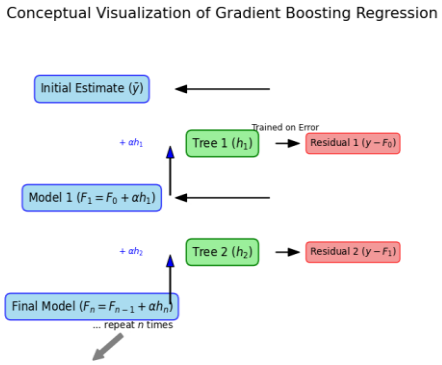

Figure 8: Conceptual Visualization of Gradient Boosting Regression

```
R-squared (R^2): 0.6658
Mean Absolute Error (MAE): 960578.78
Root Mean Squared Error (RMSE): 1299761.15
```

Figure 9: Gradient Boosting Accuracy Metrics

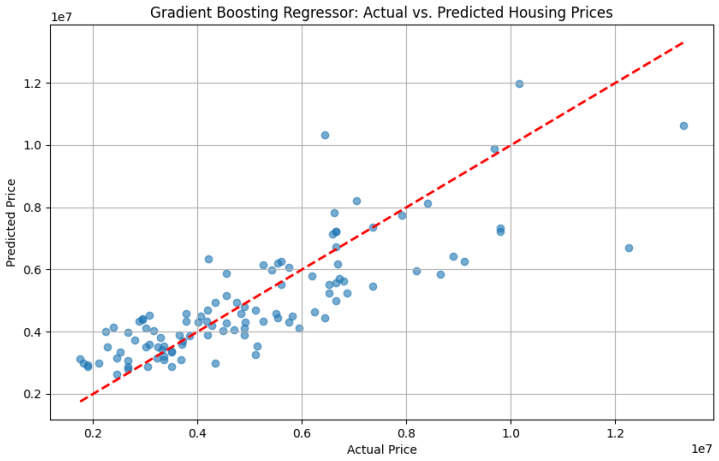

Figure 10: Gradient Boosting Regressor: Actual vs Predicted Housing Prices

Discussion

The Gradient Boosting Regression model offers a higher degree of accuracy than the Random Forest model and Single Decision tree model yet carries limitations for real-world use in real estate. The model has an R^2 value of 0.6658, indicating that there is still a 33.42% random variance in housing prices. The model's Mean Absolute Error (MAE) was 960,578.78, which means that the predictions on average were approximately 960,579 dollars off the actual value. This is still a substantial error when estimating housing prices. The Root Mean Squared Error (RMSE) is 1,299,761.15, this shows a considerable degree of outliers with a high variance between predicted and actual prices. Like the Random Forest model, using this model is real estate valuation is very unlikely due to the level of error being too large for practical in real estate.

Model 4 : Support Vector Regression

The Support Vector Regression model (SVR) is a supervised machine learning algorithm that utilizes training data to form a fitted curve through as many data points as possible. This curve can then be used to predict new and unseen data points. SVR has three variables that can be changed to optimize different dataset interpretations. “ε” (epsilon) changes the error tolerance of the function, which can fit more points into the curve, while decreasing precision. The variable “C” adjusts for the magnitude of punishment on large errors. A higher C value allows for a more complex fit. Finally, “γ” (gamma) adjusts the influence of each individual point on the curve fitting. A lower gamma value leads to a smoother curve, while high gamma values lead to a more pinpoint, wiggly function curve.

Python Code

Imports

```
import os, numpy as np, pandas as pd, kagglehub, matplotlib.pyplot as plt from sklearn.model_selection import train_test_split, RandomizedSearchCV, KFold from sklearn.preprocessing import StandardScaler, OneHotEncoder, FunctionTransformer from sklearn.compose import ColumnTransformer from sklearn.pipeline import Pipeline from sklearn.svm import SVR from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error from sklearn.compose import TransformedTargetRegressor from scipy.stats import loguniform, uniform
```

Load data

```
path = kagglehub.dataset_download("yasserh/housing-prices-dataset") df = pd.read_csv(os.path.join(path, "Housing.csv")) y = df["price"].astype(float) X = df.drop(columns=["price"])
```

Fit Data into Columns

```
cat_cols = X.select_dtypes(include=["object"]).columns num_cols = X.select_dtypes(exclude=["object"]).columns
```

Preprocess

```
pre = ColumnTransformer([ ("num", StandardScaler(), num_cols), ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False), cat_cols) ])
```

```
log_t = FunctionTransformer(np.log1p, inverse_func=np.expm1, validate=False) svr =
TransformedTargetRegressor( regressor=SVR(kernel="rbf"), transformer=log_t

)

pipe = Pipeline([("preprocessor", pre), ("regressor", svr)])
```

Train/test split

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42 )
```

Optimize C, gamma, epsilon Values

```
param_dist = { "regressor__regressor__C": loguniform(1e2, 1e5), # 1e2..1e5
"regressor__regressor__epsilon": uniform(0.005, 0.15), # 0.005..0.155
"regressor__regressor__gamma": loguniform(1e-3, 1e0), # 1e-3..1 }

cv = KFold(n_splits=5, shuffle=True, random_state=42) rs = RandomizedSearchCV( pipe,
param_distributions=param_dist, n_iter=40, scoring="r2", cv=cv, random_state=42,
n_jobs=1, verbose=1, return_train_score=True )

rs.fit(X_train, y_train) best = rs.best_estimator_ print("Best params:", rs.best_params_)
print("CV R^2 (mean):", rs.best_score_)
```

Evaluate Accuracy

```
y_pred = best.predict(X_test) r2 = r2_score(y_test, y_pred) mae =
mean_absolute_error(y_test, y_pred) rmse= np.sqrt(mean_squared_error(y_test, y_pred))
print("\nTest Performance") print(f"R^2: {r2:.3f}") print(f"MAE: {mae:,.0f}") print(f"RMSE:
{rmse:,.0f}")
```

Plot Results

```
plt.figure(figsize=(7,6)) plt.scatter(y_test, y_pred, alpha=0.65) lims = [min(y_test.min(),
y_pred.min()), max(y_test.max(), y_pred.max())] plt.plot(lims, lims, "r--", linewidth=2,
label='Perfect Prediction') plt.xlim(lims); plt.ylim(lims) plt.xlabel("Actual Price");
plt.ylabel("Predicted Price") plt.title("SVR (log-target): Actual vs Predicted Housing Prices")
plt.grid(True); plt.tight_layout(); plt.legend(); plt.show()
```

Results

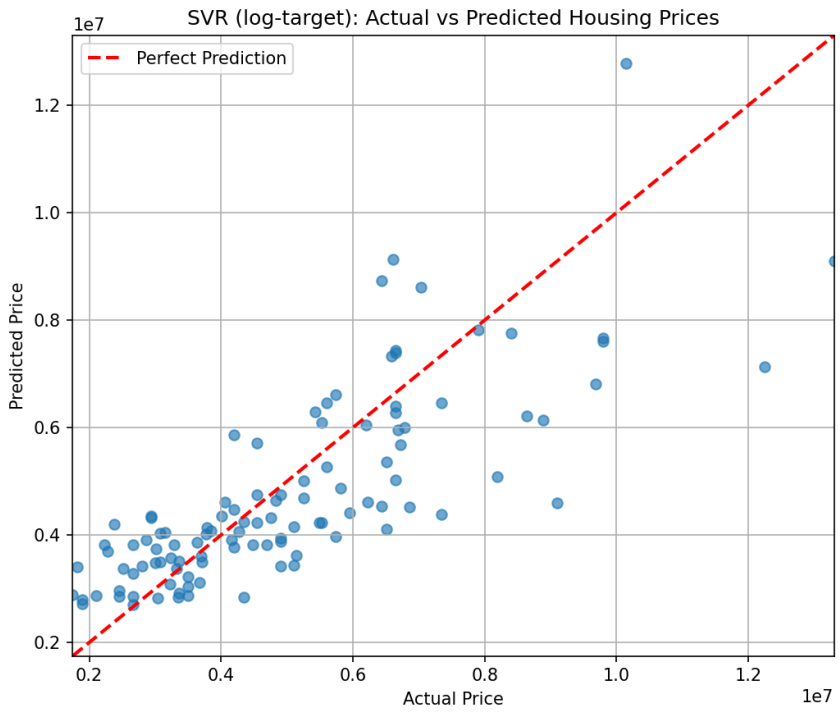

Figure 11. Predicted Price vs Actual Price of SVR model

```
Test Performance
R²:   0.601
MAE:  1,054,871
RMSE: 1,420,814
```

Figure 12. Accuracy Metrics of SVR

Discussion

The SVR model initially produced a negative R^2, indicating poor model generalization. It was found that this was due to an error in scaling, which is very common for SVR models, especially since the output values in this data set are in the millions. To alleviate this issue, the command 'TransformedTargetRegressor' was used, which scales y values prior to fitting, then reverts them to their original magnitude. This drastically improved the results of the model, which then obtained an R^2 value of ~0.4. To further improve the accuracy of the model, we added the command 'RandomizedSearchCV' to the code. This randomly tests different combinations of hyperparameters (C, ε, and γ values) within a set range and keeps the combination with the best cross-validation score. This improved the accuracy of obtaining an R^2 score of 0.601 (shown in figure 12). While this is a drastic improvement from the initial tests, it remains unviable in real-world use cases due to the high degree of inaccuracy. The SVR model scored a mean average error of 1,054,871, meaning it is on

average, ~1,000,000 dollars from the actual value, a significant number when attempting to appraise a home. Another notable distinction is that the model was more accurate in predicting values of cheaper homes, while struggling to recognize more expensive ones (see figure 11). This may be because the dataset used to train the algorithm consists mostly of cheaper homes, giving the model a higher degree of recognition for less expensive homes. Nonetheless, this algorithm requires more training and tampering before it can be considered a useful home appraisal tool.

Conclusion

This study examined the capabilities of multiple supervised machine learning models: Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, and Support Vector Regression, and evaluated their performance in predicting prices of residential homes. Each model demonstrated different strategies in taking complex, non-linear data and outputting an estimated price.

The Decision Tree model, while simple and interpretable, suffered from its limited scope, achieving an R^2 score of only 0.316. The Random Forest model improved performance through ensemble averaging, reaching an R^2 of 0.621 with reduced variance, though still exhibiting large prediction errors. Gradient Boosting delivered the most accurate results with an R^2 of 0.666 and the lowest MAE, indicating superior comparative capability in refining residual errors through iterative learning. Lastly, the Support Vector Regression model performed comparably to the Random Forest, achieving an R^2 of 0.601 after hyperparameter tuning and price scaling, but struggled with predicting the cost of higher valued homes.

Gradient boosting showed the highest degree of accuracy overall, improving its accuracy with each decision tree iteration. However, all of these models were significantly inaccurate in a real-world context and would not be useful in property appraisal as of right now. An obvious explanation of this is the limited scope of the dataset. There are many factors that affect the price of housing in the US that the data set did not factor, such as the year a house was built, the state of the economy, the prices of neighboring houses, and

many more that significantly contribute to the value of a home. While increasing the scope of the dataset would provide more information to this problem, it could pose many challenges to these different models as overfitting poses a significant challenge. However, by expanding the dataset, engineering the parameters and features of each model, and possibly experimenting with neural networks in this field, the possibility of a housing valuation machine learning model remains an exciting possibility in the ever-expanding field of artificial intelligence.

References

H, M. Y. (2022, January 12). *Housing prices dataset*. Kaggle. https://www.kaggle.com/datasets/yasserh/housing-prices-dataset