

Project Report: Advanced House Price Prediction

Subject: Capstone Project – Ames Housing Price Regression Analysis

Author: Madhumitha KA

GitHub [repository link](#)

1. Executive Summary

This report details the development and evaluation of a machine learning pipeline designed to predict residential housing prices in Ames, Iowa. The objective was to build a robust regression model capable of accurately estimating property values based on diverse features such as square footage, property age, and quality indicators.

The final solution utilizes an **Advanced Stacking Ensemble**, combining linear models (Ridge, Lasso) with tree-based methods (Random Forest, XGBoost, LightGBM). The model achieved a **Cross-Validation R^2 of 0.8798** and a **RMSE of 0.1317** (on log-transformed prices). While the model demonstrates strong predictive power, analysis reveals signs of overfitting (Train $R^2 \sim 0.99$) and specific failure modes related to luxury properties and non-standard layouts.

2. Problem Statement & Data

Objective:

To minimize the error between predicted and actual sale prices of homes, enabling precise valuation for real estate stakeholders.

Dataset:

The project utilizes the Ames Housing dataset (via `sklearn.fetch_openml`).

- **Target Variable:** `SalePrice`. The target was log-transformed (`np.log1p`) to correct for the inherent right-skewness of monetary data.
- **Features:** A mix of numerical (e.g., `YearBuilt`, `TotalBsmtSF`) and categorical (e.g., `GarageType`, `PoolQC`) variables.

3. Methodology

3.1 Feature Engineering

The raw data was transformed to capture domain-specific insights:

- **Temporal Features:** Calculated `HouseAge` (Year Sold - Year Built) and `RemodelAge`.
- **Aggregation:** Created `TotalSF` by summing basement, 1st floor, and 2nd floor areas.
- **Interaction Terms:** Introduced `TotalSF_OverallQual` to capture the multiplicative effect of size and quality.
- **Binary Flags:** Converted sparse features like Pools and Fireplaces into binary flags.

3.2 Preprocessing Pipeline

To prevent data leakage, a strict pipeline was implemented:

- **Numerical Data:** Imputed with the **Median** and applied **StandardScaler**.
- **Categorical Data:** Imputed with the **Most Frequent** value and applied **One-Hot Encoding**.

3.3 Model Architecture Overview

A **Stacking Regressor** was chosen to combine the bias-reduction of boosting with the variance-reduction of bagging.

3.4 Deep Dive: Working of the Stacking Ensemble Model

The core of this solution is a **Stacking Ensemble**, a sophisticated technique that combines multiple regression models to improve overall predictive performance. Instead of relying on a single algorithm, stacking allows different models to act as "experts" on different parts of the data, with a final "manager" model learning how to best combine their opinions.

Phase 1: Input Transformation (The Foundation)

Before any learning occurs, the raw housing data flows through the preprocessing pipeline (Section 3.2). Crucially, the target variable (`SalePrice`) is Log-Transformed.

- **Why?** Real estate prices follow a "power law" (many cheap homes, few ultra-expensive ones). This skewness confuses models. Log-transformation forces the prices into a "Normal Distribution" (Bell Curve), making patterns easier to learn.

Phase 2: The Base Learners (Level 0)

The preprocessed data is fed simultaneously into five diverse "Base Models." Diversity is critical here; if all models make the same mistakes, stacking fails.

1. **RidgeCV & LassoCV (Linear Models):** These capture simple, linear trends (e.g., "adding 100 sq ft adds \$5k"). They are stable and prevent the ensemble from going wild on outliers.
2. **Random Forest (Bagging):** This builds hundreds of independent decision trees. It is excellent at handling non-linear data and does not assume a straight-line relationship between price and features.

3. **XGBoost & LightGBM (Gradient Boosting):** These are the heavy hitters. They build trees sequentially, where each new tree specifically tries to fix the errors of the previous one. They capture highly complex interactions (e.g., "A pool is valuable *only if* the lot size is large").

Phase 3: The Stacking Mechanism

The models do not just output a final number. They undergo a process called **Out-of-Fold (OOF) Prediction**:

1. The training data is split into 5 parts (folds).
2. Each base model is trained on 4 parts and predicts the 5th.
3. This creates a new dataset where the "features" are no longer *Square Footage* or *Year Built*, but rather *Prediction from XGBoost*, *Prediction from Ridge*, etc.

Phase 4: The Meta-Learner (Level 1)

The final piece is the Meta-Learner, which in this project is **LassoCV**.

- **Input:** The predictions from the five base models.
- **Task:** It learns weights for each base model. For example, it might learn that "*When the predicted price is high, trust XGBoost; when low, trust Ridge.*"
- **Why Lasso?** Lasso has a regularization property that can shrink coefficients to zero. If the Random Forest model is predicting poorly, Lasso will assign it a weight of zero, effectively firing that "expert" from the committee.

Final Output: The Meta-Learner outputs a log-price, which is then converted back to a dollar amount (`np.exp(m1)`) for the final valuation.

4. Results & Evaluation

The model was evaluated using 5-Fold Cross-Validation to ensure stability.

Metric	Score	Interpretation
Average CV RMSE	0.1317	The average error in log-price prediction is low, roughly translating to ~13-14% error in actual dollar value.
Average CV R^2	0.8798	The model explains approximately 88% of the variance in house prices on unseen data.

Average Train R^2	0.9883	The model explains 99% of variance on training data.
---------------------	--------	--

Performance Analysis:

The significant gap between Training R^2 (0.99) and Validation R^2 (0.88) indicates overfitting. The ensemble is memorizing noise in the training set.

5. Failure Analysis (Critical Review)

Despite high aggregate accuracy, the model exhibits specific failure modes. Analyzing the "Actual vs. Predicted" and "Residual" plots generated in the notebook reveals where the model struggles.

Failure Type 1: The "Luxury Ceiling" Effect (High-Value Underestimation)

- **Observation:** In the regression plot, data points at the far right (highest actual prices) tend to fall *below* the red prediction line.
- **Example Case:** A home selling for \$750,000 might be predicted at \$650,000.
- **Why it failed:** Tree-based models (Random Forest/XGBoost) cannot extrapolate. They can only predict values within the range of the training data. If the training set lacks "super-luxury" homes, the model clips predictions at the highest known value.

Failure Type 2: The "Fixer-Upper" Variance (Low-Value Instability)

- **Observation:** The residual plot shows higher scatter at the lower end of the price spectrum.
- **Example Case:** Two homes with identical small square footage sell for different prices—one is a cottage, the other a foreclosure.
- **Why it failed:** The binary feature `HasFireplace` or `OverallQual` might not capture the *extent* of damage in a low-quality home. The model likely over-predicts the price of damaged homes because it assumes a baseline "livable" condition based on median imputation.

Failure Type 3: The "Large Lot" Outliers

- **Observation:** Homes with massive `TotalSF` but average `SalePrice` create large residuals.
- **Example Case:** A property with 4,000 sq ft situated in a less desirable neighborhood.
- **Why it failed:** The interaction feature `TotalSF * OverallQual` assumes a linear relationship between size and value. However, the market demonstrates diminishing returns. The linear components (Ridge/Lasso) of the stack might interpret the massive size as a signal for a massive price, leading to an over-prediction.

6. Conclusion & Recommendations

The Stacking Ensemble successfully automates the valuation of Ames housing with an R^2 of ~0.88. However, the model complexity has led to overfitting.

Actionable Next Steps:

1. **Reduce Complexity:** Prune the Random Forest depth and increase regularization in Ridge/Lasso.
2. **Handle Outliers:** Manually remove homes with >4000 sq ft from the training set to prevent skewing the linear models.
3. **Feature Selection:** Use Lasso coefficients to drop zero-weight features before feeding them into the XGBoost model.