Advanced Regression for House Price Prediction

Project Report and MLOps Deployment

Vaibhav and Vismita

November 18, 2025

Outline

Project Goal & Dataset

The Goal

Our primary goal was to explore and implement a sophisticated, end-to-end machine learning workflow to accurately predict the final sale price of residential homes for the Kaggle competition "House Prices: Advanced Regression Techniques," using a complex dataset of 79 explanatory variables describing homes in Ames, Iowa.

Evaluation Metric: RMSLE

- The competition is judged on the **Root Mean Squared Logarithmic Error (RMSLE)**.
- This metric is sensitive to *relative* errors, not absolute values. It also heavily penalizes under-predictions more than over-predictions.

Evaluation Metric: RMSLE

- The competition is judged on the **Root Mean Squared Logarithmic Error (RMSLE)**.
- This metric is sensitive to *relative* errors, not absolute values. It also heavily penalizes under-predictions more than over-predictions.

RMSLE Formula

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

Evaluation Metric: RMSLE

- The competition is judged on the **Root Mean Squared Logarithmic Error (RMSLE)**.
- This metric is sensitive to *relative* errors, not absolute values. It also heavily penalizes under-predictions more than over-predictions.

RMSLE Formula

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

- **Our Optimization Strategy:**
  - We log-transform the target variable: $y = \log(\text{SalePrice} + 1)$.
  - We then train our models to optimize the standard **Root Mean Squared Error (RMSE)** on this transformed data.
  - This is mathematically equivalent to optimizing for RMSLE on the original data.

Target Variable: SalePrice

- The original `SalePrice` variable was highly right-skewed.
- This violates the assumptions of normality required by many linear regression models and can negatively impact tree-based models.
- Applying a $\log(1+x)$ transformation normalized the distribution, stabilizing the variance and making it far more suitable for modeling.

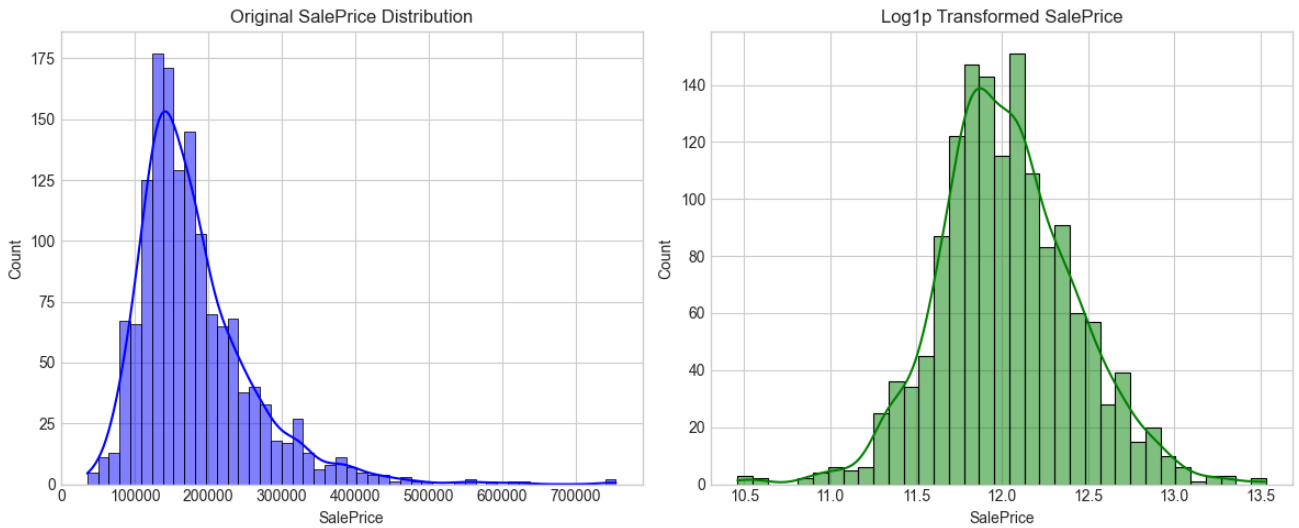

Figure: Distribution of SalePrice before and after log transformation.

Preprocessing & Feature Engineering

A robust pipeline was essential for model performance:

Data Cleaning

- Removed 2 significant outliers (`GrLivArea` > 4000 sq. ft. with unusually low price).
- **Smart Imputation:** Based on data description.
  - 'NA' $\rightarrow$ 'None' (e.g., `PoolQC` where 'NA' means no pool).
  - 'NA' $\rightarrow$ 0 (e.g., `GarageArea` where 'NA' means no garage).
  - Median by `Neighborhood` (for `LotFrontage`).

Feature Engineering

- **New Features:**
  - `TotalSF` (Sum of basement, 1st, and 2nd floor sqft).
  - `TotalBath` (Sum of all full and half baths).
- **Transformations:**
  - Corrected skew for numerical features (Box-Cox).
  - Label Encoding for categorical features used in tree models.

Phase 1: Baseline Model Comparison

- A 5-Fold Cross-Validation strategy was used to get a reliable estimate of each model's generalization performance.
- **Key Finding:** Gradient Boosting models (CatBoost, LGBM, XGBoost) and Regularized Linear Models (Ridge, Lasso) were the most promising.
- **Validation:** Our internal CV scores were strongly validated by the Kaggle public leaderboard scores, giving us high confidence in our setup.

Phase 1: Model Performance

Model	Mean CV RMSLE	Std Dev
CatBoost	0.1239	0.0191
Lasso	0.1312	0.0233
Ridge (baseline)	0.1319	0.0197
LightGBM	0.1346	0.0174
XGBoost	0.1493	0.0167
RandomForest	0.1446	0.0178
WideDeepNN	0.1509	0.0159
EmbeddingNN	0.1766	0.0222
SVR	0.2119	0.0231

Table: Untuned base model comparison (CV).

Phase 1: Model Performance

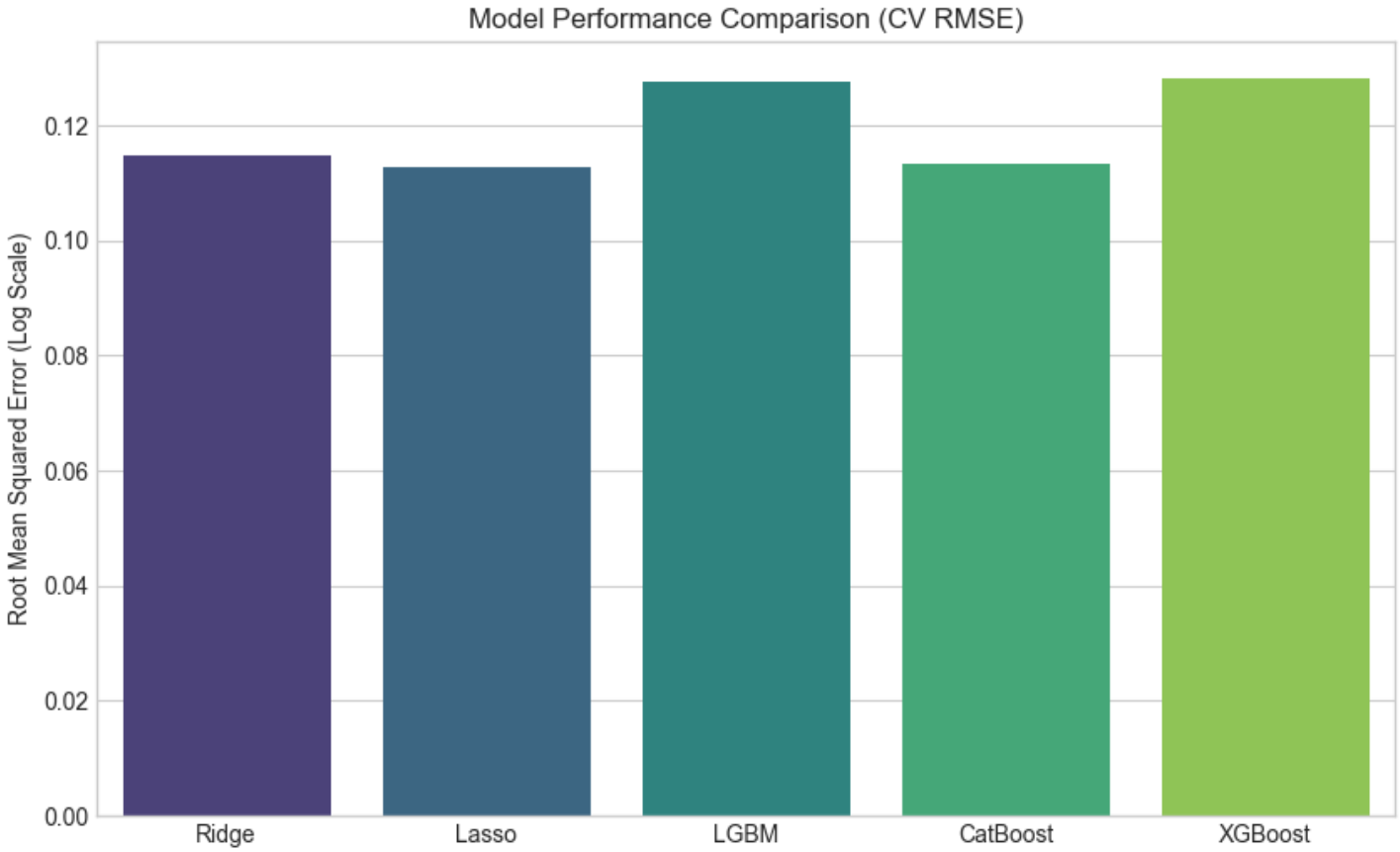

Figure: Mean CV RMSE for baseline models (lower is better).

Phase 2: Hyperparameter Tuning (Optuna)

- We systematically tuned the top models using **Optuna**, an automated hyperparameter optimization framework.
- This step was critical and led to **massive** performance improvements, slashing CV error rates by 35-45% for the boosting models.

Table: Impact of Hyperparameter Tuning (CV Score)

Model	**Untuned CV**	**Tuned CV**	**Improve (%)**
LightGBM	0.13463	0.07929	41.1%
XGBoost	0.14933	0.07729	48.2%
CatBoost	0.12393	0.07462	39.8%
Ridge	0.13190	0.10762	18.4%

Phase 3: Final Model & Ensemble

- A **stacking ensemble** was created to combine the strengths of the top models.
  - *Level 1 (Base Models):* Tuned CatBoost, LightGBM, Ridge, and Lasso.
  - *Level 2 (Meta-Model):* A RidgeCV model was trained on the out-of-fold predictions of the base models.

- **Result:** A single, deeply-optimized **LightGBM model** achieved the best score (0.03679). The multiple stacking of three models had a score of 0.12301.

Key Takeaway

For this problem, deep optimization of a single superior model (LGBM) proved more effective than ensembling.

Global Feature Importance (SHAP)

- We used SHAP (SHapley Additive exPlanations) to understand *how* our best model makes predictions.
- SHAP values quantify the impact of each feature on the model's output, aligning with real-world intuition.
- **Top Features:** `OverallQual`, `GrLivArea`, and `TotalSF` were confirmed to have the largest impact on price predictions.

Feature Importance Plot

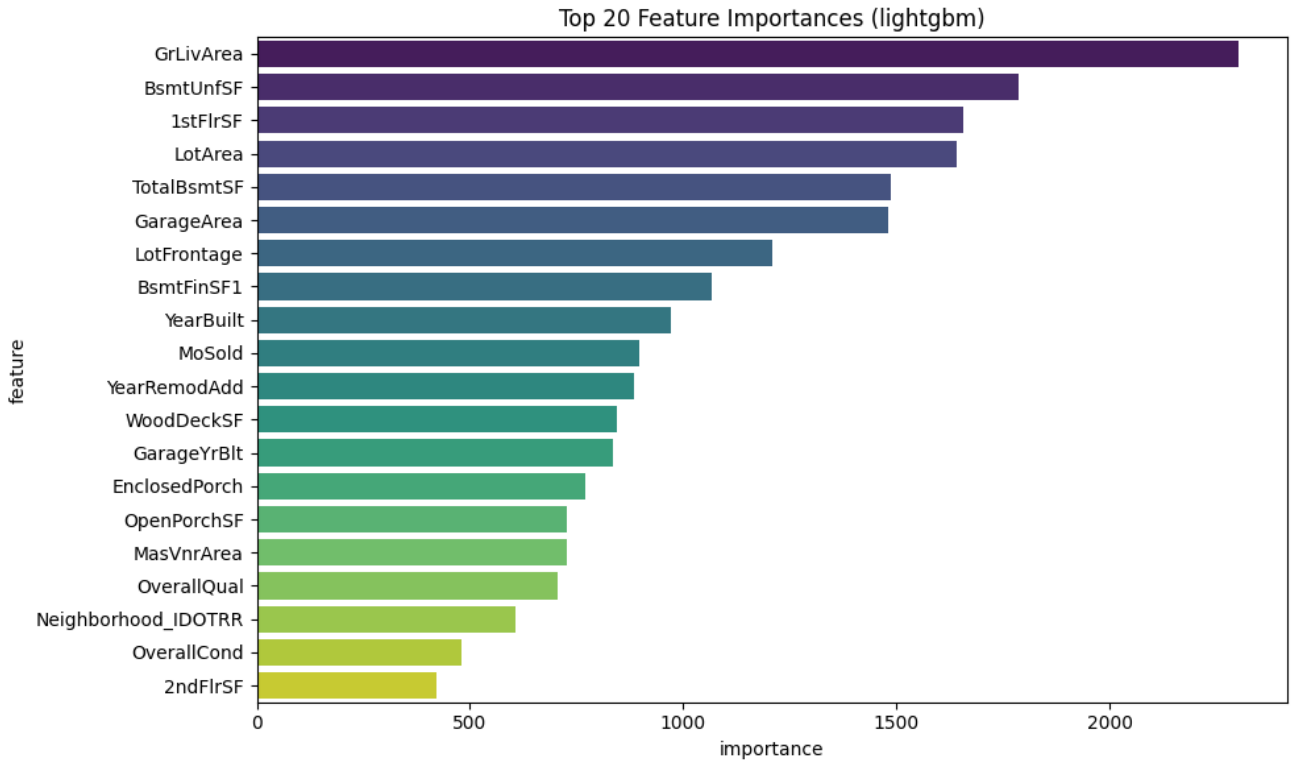

Figure: Top 20 Features (Generated via MLflow Pipeline).

SHAP Summary Analysis

- The SHAP summary plot combines feature importance with feature effects.
- **Interpretation:**
  - High values of `OverallQual` (red) stretch far to the right, indicating a strong positive impact on price.
  - Low values (blue) stretch to the left, lowering the predicted price.

SHAP Summary Plot

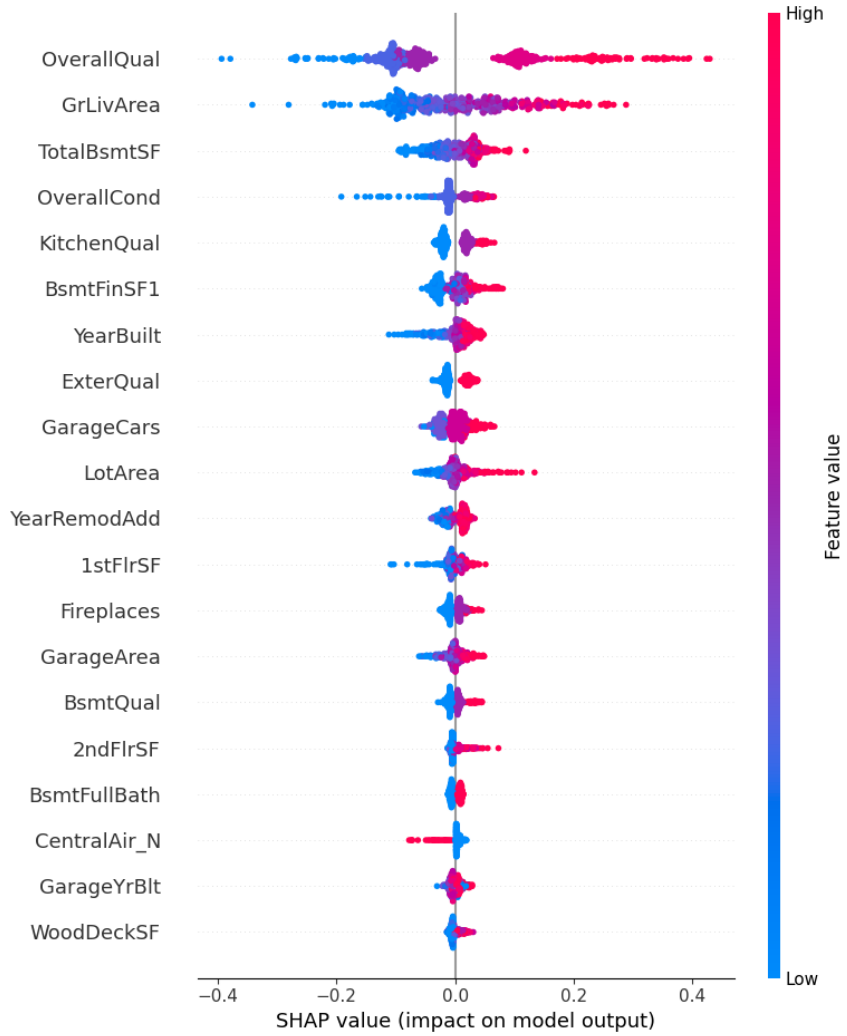

Figure: SHAP Summary Plot (Impact on Model Output).

Phase 4: Model Deployment (Flask API)

- **Core Service:** A Flask API was developed to operationalize the model.
- **Architecture:** We utilized a **Single Artifact Approach**.
  - Instead of saving models and scalers separately, we serialized the entire Scikit-Learn `Pipeline` into one file: `model_pipeline.pkl`.
- **Robust Pipeline:** This ensures training/serving consistency:
  - The pipeline handles **Imputation**, **Scaling**, and **One-Hot Encoding** internally.
  - This eliminates "Train/Serve Skew" by automatically handling missing columns or unseen categories.

Phase 5: Containerization (Docker)

To ensure the API runs reliably on any machine, we implemented a **Docker Deployment Pipeline**.

- **The Dockerfile Strategy:**
  - **Base Image:** `python:3.9-slim` for a lightweight footprint.
  - **System Dependencies:** Explicitly installed `libgomp1` to support the LightGBM engine.
  - **Port Mapping:** Exposed Port 5000 to allow external requests.
- **Result:** A portable, production-ready container. We verified functionality by mapping local port 5000 to container port 5000 and successfully receiving predictions via JSON `POST` requests.

Key Project Takeaways

- **Preprocessing is Fundamental:** Rigorous preprocessing (contextual 'NA' imputation, skew correction) built the foundation for all model success.
- **Tuning is Critical:** Systematic hyperparameter tuning was the single most impactful activity, reducing error by nearly 50%.
- **MLOps Maturity:** Moving beyond a notebook to a fully containerized API with MLflow tracking ensures the project is reproducible and deployable in the real world.
- **Optimization > Ensembling:** A single, well-tuned model (LGBM) outperformed a complex stack, demonstrating the power of deep optimization.

Project Contributions I

Vaibhav's Contributions:

- Cleaned dataset and built preprocessing pipeline.
- Engineered new features and established baseline model performance.
- Trained ensemble models and performed comparative analysis.
- Implemented stacking strategies.
- Implemented SHAP and LIME frameworks to execute comprehensive global feature analysis and local prediction interpretability and drafted project reports.
- (MLOps & Deployment):
  - Integrated MLflow for experiment tracking.
  - Developed Flask API and Docker Deployment Pipeline(With Frontend).
- Finalized project report and Documented codebase.

Project Contributions II

Vismita's Contributions:

- Authored the Exploratory Data Analysis (EDA) report.
- Produced the cleaned dataset and the data quality report.
- Implemented and trained advanced ensemble models and the optional neural network.
- Performed hyperparameter tuning to optimize model performance.
- Led the model evaluation and selection process.
- Developed the REST API for model deployment.
- Co-authored the final project report and presentation.

Thank You