ML Project Report: Hotel Value Prediction

Author: Manav Jindal (IMT2023535), Arshbir Singh Dang (IMT2023132), Pushkar Kulkarni (IMT2023087)
Kaggle Team Name: Dall-Eminators
GitHub Link: https://github.com/Arshbir1/ML_Project

Date: October 26, 2025

Table of Contents

1. Task Definition

1.1 Project Objective

The primary objective of this project was to develop a machine learning model capable of accurately predicting the market value (HotelValue) of hotel properties. This involved leveraging a provided dataset containing various property attributes to build a robust regression model. The ultimate goal was to achieve the lowest possible prediction error on unseen data within the context of a Kaggle competition.

1.2 Competition Context

This project was structured as a Kaggle competition. Participants were given a training dataset (train.csv) with known HotelValue outcomes and a test dataset (test.csv) without these values. The task was to train models on the training data and submit predictions for the test data. Submissions were evaluated against a hidden set of true values, and rankings were determined based on the prediction accuracy.

1.3 Performance Metric: RMSE

The official evaluation metric for the competition was the **Root Mean Squared Error (RMSE)**. RMSE measures the average magnitude of the errors between predicted and actual values in dollars. Lower RMSE values indicate better model performance. During development, especially with log-transformed targets, RMSLE (Root Mean Squared Logarithmic Error) or R^2 were often used as proxies during cross-validation.

1.4 Project Constraints

A key constraint for the initial phase (Checkpoint 1) of this project was the limitation on model types. We were restricted to using only the models taught within the course curriculum: Linear Regression (Ridge, Lasso), K-Nearest Neighbors, Decision Trees, and basic Ensemble methods (Bagging, Random Forests, AdaBoost, Gradient Boosting). SVMs and Neural Networks were prohibited.

2. Dataset and Features

2.1 Data Source and Structure

The data was provided as two CSV files: train.csv (1200 rows, 81 columns including target) and test.csv (260 rows, 80 columns). The Id column was used for submission matching but not for training.

2.2 Feature Overview

The 79 features covered location, lot characteristics, property type, quality/condition ratings, size/area, rooms/amenities, basement details, parking details, and sale information. Data types included numerical (continuous and discrete), categorical (nominal and ordinal), and date-related years.

2.3 Initial Data Challenges

Initial analysis revealed:

1. **Missing Data:** Widespread missing values, especially for features like PoolQuality, ServiceLaneType, requiring imputation.
2. **Categorical Features:** Numerous text-based features needing numerical encoding.
3. **Feature Scaling:** Large differences in the scale of numerical features.
4. **Target Variable Skewness:** HotelValue was highly right-skewed.

3. Exploratory Data Analysis (EDA)

EDA was performed to understand data distributions, correlations, and inform preprocessing.

3.1 Target Variable Analysis (HotelValue)

- **Distribution:** A histogram and Q-Q plot confirmed severe right-skewness (skewness ≈ 1.7).
- **Transformation:** A log1p transformation was applied, resulting in a near-normal distribution, essential for linear models.

3.2 Numerical Feature Analysis

- **Correlations:** OverallQuality, UsableArea, TotalSF, GroundFloorArea showed the strongest positive correlations with log(HotelValue). PropertyAge showed a negative correlation.
- **Skewness:** Many numerical predictors were also skewed, suggesting log-transformation might benefit linear models (tested during experiments).

3.3 Categorical Feature Analysis

- **Impact:** Box plots or violin plots showed clear relationships between categories (e.g., OverallQuality) and log(HotelValue).
- **Encoding Need:** Confirmed the necessity of converting features like District, FoundationType, etc., to numbers.

3.4 Missing Data Analysis

- Visualizations confirmed high missing rates for amenity-related features (Pool, Fence, etc.), suggesting 'None' imputation. Moderate missingness in basement/garage features. Sporadic missingness elsewhere (e.g., RoadAccessLength).

3.5 EDA Conclusions

EDA confirmed the need for: Log-transforming the target, robust imputation, feature engineering (e.g., TotalSF, PropertyAge), categorical encoding, and feature scaling (especially for linear models).

4. Preprocessing and Feature Engineering (Final Strategy)

This section describes the preprocessing pipeline used in the **winning strategy**. Experiments with more aggressive feature engineering are discussed in Section 5.

4.1 Rationale and Strategy

The goal was to create a clean, numerical dataset suitable for regularized linear models (Lasso, Ridge) while preserving key predictive signals identified in EDA. Simplicity and robustness were prioritized based on experimental results.

4.2 Target Variable Transformation

HotelValue was transformed using np.log1p.
y = np.log1p(train[target_col])
train = train.drop(columns=[target_col])

4.3 Missing Value Imputation

A straightforward approach proved most effective:

1. Specific categorical features indicating absence were filled with 'None' (though the final

script used mode/median for simplicity).
2. All remaining categorical NaNs were filled with the **mode**.
3. All remaining numerical NaNs were filled with the **median**.

```
all_data = pd.concat([train, test], ignore_index=True)
for col in all_data.columns:
    if all_data[col].dtype == 'object':
        all_data[col] = all_data[col].fillna(all_data[col].mode()[0])
    else:
        all_data[col] = all_data[col].fillna(all_data[col].median())
```

4.4 Feature Engineering

Simple, high-impact features were created:

```
all_data['HotelAge'] = all_data['YearSold'] - all_data['ConstructionYear']
all_data['YearsSinceRenovation'] = all_data['YearSold'] - all_data['RenovationYear']
all_data['TotalSF'] = all_data['BasementTotalSF'] + all_data['GroundFloorArea'] +
all_data['UpperFloorArea']
all_data['TotalBathrooms'] = (
    all_data['FullBaths'] + 0.5 * all_data['HalfBaths'] +
    all_data['BasementFullBaths'] + 0.5 * all_data['BasementHalfBaths']
)
```

4.5 Categorical Encoding

One-Hot Encoding (pd.get_dummies) was applied to all object-type columns.

```
all_data = pd.get_dummies(all_data, drop_first=True)
```

4.6 Feature Scaling

StandardScaler was applied after splitting the data back into training and test sets. This step was crucial for Lasso and Ridge.

```
X = all_data[:len(train)]
X_test = all_data[len(train):]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_test_scaled = scaler.transform(X_test)
```

4.7 Final Preprocessed Data Structure

The final training data (X_scaled) was a NumPy array containing only scaled numerical features (over 200 after one-hot encoding), ready for linear model training.

5. Model Selection and Experiments

5.1 Initial Model Screening (01_initial_model_screening.py)

- **Objective:** Evaluate all permitted models using a consistent preprocessing pipeline (scaling, one-hot encoding) and 5-fold cross-validation (RMSLE scoring).
- **Models:** LinearRegression, Ridge, Lasso, KNeighborsRegressor, DecisionTreeRegressor, BaggingRegressor, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor.
- **Findings:**
  - Linear models (Lasso, Ridge) showed strong, stable performance (RMSLE ~0.13-0.14).
  - Tree ensembles (GradientBoosting, RandomForest) achieved slightly better CV scores (RMSLE ~0.12-0.13).
  - KNN, DecisionTree performed poorly.
- **Conclusion:** Linear models and tree ensembles were identified as the primary candidates. The script also included GridSearchCV for GradientBoostingRegressor and generated an initial submission based on it.

5.2 Experiment 1: Advanced Tree Ensembles

Based on promising CV scores, extensive experiments were conducted to optimize tree-based models.

- **5.2.1 Gradient Boosting Tuning (04_experiment_gbr_tuned.py):** Focused on GradientBoostingRegressor, using GridSearchCV with a wider parameter grid (including subsample, max_features, min_samples_leaf) and Huber loss. Preprocessing was tailored (less aggressive, no scaling).
- **5.2.2 XGBoost Tuning & Comparison (05_experiment_xgb_native_api.py, 06_experiment_xgboost_vs_ridge.py):** Explored XGBoost, using its native API (xgb.cv, xgb.train) for efficient tuning (05...py). Also compared tuned XGBoost directly against Ridge and attempted blending (06...py).
- **5.2.3 LightGBM with K-Fold Averaging (03_experiment_lightgbm_kfold.py):** Implemented LightGBM with preprocessing optimized for it (native categorical handling). Trained 10 models on K-Folds and averaged their predictions for robustness.
- **5.2.4 Stacking Ensemble (02_experiment_stacking_ensemble.py):** Built a StackingRegressor using GBR, RandomForest, ExtraTrees, AdaBoost as base models and Ridge as the meta-model, combined with more complex feature engineering.
- **5.2.5 Findings: Overfitting Issues:** Despite achieving excellent CV scores (RMSLE often ~0.12), **all tree-based ensemble approaches performed poorly on the Kaggle leaderboard** (RMSE ~21,000 - 26,000). This indicated significant overfitting to the training data.

5.3 Experiment 2: Aggressive Feature Engineering with Linear Models

Since linear models showed better generalization, experiments focused on enhancing them with more features.

- **5.3.1 Aggressive Features without Scaling (08_linear_model_aggressive_blend.py):** Created numerous polynomial and interaction features, plus target encoding. Crucially, **omitted scaling**. Used LassoCV, RidgeCV and blended the results. *Result:* Performance degradation (RMSE ~21,000), proving scaling is essential.
- **5.3.2 Aggressive Features with Optimal Blending (09_linear_model_optimized_blend.py):** Used the same aggressive features but **no scaling**. Trained LassoCV, RidgeCV, ElasticNetCV using K-Fold averaging and found optimal blending weights using scipy.optimize. *Result:* Better than script 08, but still worse than the baseline (RMSE ~19,100).
- **5.3.3 Findings: Diminishing Returns and Instability:** Adding excessive features, especially without scaling, hurt performance or provided only marginal gains while increasing complexity. The core features seemed most important.

5.4 The Winning Strategy: Scaled Linear Models

The failures of complex models and aggressive features led back to the simple, robust strategy.

- **5.4.1 Baseline Linear Tuning (07_linear_model_tuning_baseline.py, 10_final_linear_model_selection.py):** These scripts represent the core logic of Draft5.py. They used simple features, standard imputation, one-hot encoding, **StandardScaler**, and loops with cross_val_score (R^2 scoring) to find the best alpha for Lasso and Ridge, selecting the single best for submission. *Result:* Achieved the best leaderboard scores (RMSE ~18,600-18,900).
- **5.4.2 Final Model: Polished Blend (Based on Winning Logic):** The ultimate script combined the winning preprocessing from 07/10 with more efficient tuning (LassoCV, RidgeCV) and blended the predictions of the best Lasso and Ridge models (50/50).

5.5 Model Performance Summary

Model / Strategy (Script Ref.)	**Key Preprocessing**	**Best CV Score (RMSLE↓)**	**Best Kaggle Score (RMSE↓)**	**Notes**
Lasso + Ridge Blend (Polished 07/10 logic)	Simple Features, Scaler, Blend	~0.135	**~18,600**	**Winning Strategy:** Best generalization.
Lasso/Ridge Single Best (07, 10)	Simple Features, Scaler	~0.135	~18,900 - 19,000	Core winning logic, slightly less stable than blend.
Aggressive Linear Blend, Optimal Weights, No Scaler (09)	Max Features, No Scaler, Optimal Blend	~0.125	~19,100	Proved scaling is essential.

Aggressive Linear Blend, 50/50, No Scaler (08)	Max Features, No Scaler, 50/50 Blend	~0.130	~21,000	Proved scaling is essential.
LightGBM K-Fold Average (03)	Tree Preprocessing, K-Fold	~0.121	~21,000 - 26,000	Overfitting.
Tuned Gradient Boosting (04)	Tree Preprocessing	~0.124	~22,000 - 25,000	Overfitting.
Tuned XGBoost (05, 06)	Tree Preprocessing	~0.125	~22,000 - 25,000	Overfitting.
Stacking Ensemble (02)	Complex Features, Scaler	~0.128	~24,000	Overfitting / Complexity issue.
Initial Screening Models (01)	Simple Features, Scaler		~26000	Identified Linear/Ensembles as candidates.

Note: CV Scores are RMSLE estimates from training. Kaggle Scores are RMSE based on leaderboard feedback.

6. Conclusion

6.1 Summary of Findings

This project demonstrated that for this hotel value prediction task, a **blended ensemble of well-tuned, regularized linear models (Lasso, Ridge) applied to appropriately scaled data with simple feature engineering provided the best generalization performance.** Despite promising cross-validation scores, more complex tree-based ensembles (GBR, XGBoost, LightGBM, Stacking) and aggressive feature engineering strategies consistently resulted in overfitting and poorer performance on the hidden test set.

6.2 Key Learnings

- **Trust the Leaderboard (Generalization):** Cross-validation scores can be misleading. Performance on unseen data (Kaggle leaderboard) is the ultimate test.
- **Preprocessing is Model-Specific:** Linear models require scaling and careful handling of categoricals (one-hot encoding worked best here). Tree models need different preprocessing (native categorical handling, no scaling).
- **Complexity vs. Performance:** More complex models or features do not guarantee better results. Overfitting is a significant risk. Simplicity and robustness often win.
- **Linear Models are Powerful:** With proper regularization and preprocessing, linear models can be highly effective, especially on high-dimensional data created by one-hot encoding.

6.3 Future Work

- **Refined Blending Weights:** Systematically optimize the blending weights for the final Lasso/Ridge models (beyond 50/50).
- **Feature Selection:** Apply explicit feature selection techniques before training the linear models to potentially remove noise.
- **Alternative Linear Models:** Explore other regularized linear models like Bayesian Ridge.
- **Careful Tree Ensemble Tuning:** Revisit tree ensembles with much stronger regularization parameters or different preprocessing to mitigate overfitting, though linear models seem superior here.

7. Appendix

7.1 Full Feature List

- PropertyClass: Identifies the type of dwelling involved in the sale.
- ZoningCategory: Identifies the general zoning classification of the sale.
- RoadAccessLength: Linear feet of street connected to property
- LandArea: Lot size in square feet
- RoadType: Type of road access
- ServiceLaneType: Type of alley access
- PlotShape: General shape of property
- LandElevation: Flatness of the property
- UtilityAccess: Type of utilities available
- PlotConfiguration: Lot configuration
- LandSlope: Slope of property
- District: Physical locations within Ames city limits
- NearbyTransport1: Proximity to main road or railroad
- NearbyTransport2: Proximity to main road or railroad (if a second is present)
- PropertyType: Type of dwelling
- HotelStyle: Style of dwelling
- OverallQuality: Overall material and finish quality
- OverallCondition: Overall condition rating
- ConstructionYear: Original construction date
- RenovationYear: Remodel date
- RoofDesign: Type of roof
- RoofMaterial: Roof material
- ExteriorPrimary: Exterior covering on house
- ExteriorSecondary: Exterior covering on house (if more than one material)
- FacadeType: Masonry veneer type
- FacadeArea: Masonry veneer area in square feet
- ExteriorQuality: Exterior material quality

- ExteriorCondition: Present condition of the material on the exterior
- FoundationType: Type of foundation
- BasementHeight: Height of the basement
- BasementCondition: General condition of the basement
- BasementExposure: Walkout or garden level basement walls
- BasementFacilityType1: Quality of basement finished area
- BasementFacilitySF1: Type 1 finished square feet
- BasementFacilityType2: Quality of second finished area (if present)
- BasementFacilitySF2: Type 2 finished square feet
- BasementUnfinishedSF: Unfinished square feet of basement area
- BasementTotalSF: Total square feet of basement area
- HeatingType: Type of heating
- HeatingQuality: Heating quality and condition
- CentralAC: Central air conditioning
- ElectricalSystem: Electrical system
- GroundFloorArea: First Floor square feet
- UpperFloorArea: Second floor square feet
- LowQualityArea: Low quality finished square feet (all floors)
- UsableArea: Above grade (ground) living area square feet
- BasementFullBaths: Basement full bathrooms
- BasementHalfBaths: Basement half bathrooms
- FullBaths: Full bathrooms above grade
- HalfBaths: Half baths above grade
- GuestRooms: Number of bedrooms above basement level
- Kitchens: Number of kitchens
- KitchenQuality: Kitchen quality
- TotalRooms: Total rooms above grade (does not include bathrooms)
- PropertyFunctionality: Home functionality rating
- Lounges: Number of fireplaces
- LoungeQuality: Fireplace quality
- ParkingType: Garage location
- ParkingConstructionYear: Year garage was built
- ParkingFinish: Interior finish of the garage
- ParkingCapacity: Size of garage in car capacity
- ParkingArea: Size of garage in square feet
- ParkingQuality: Garage quality
- ParkingCondition: Garage condition
- DrivewayType: Paved drive
- TerraceArea: Wood deck area in square feet
- OpenVerandaArea: Open porch area in square feet
- EnclosedVerandaArea: Enclosed porch area in square feet
- SeasonalPorchArea: Three season porch area in square feet
- ScreenPorchArea: Screen porch area in square feet

- SwimmingPoolArea: Pool area in square feet
- PoolQuality: Pool quality
- BoundaryFence: Fence quality
- ExtraFacility: Miscellaneous feature not covered in other categories
- ExtraFacilityValue: Value of miscellaneous feature
- MonthSold: Month Sold
- YearSold: Year Sold
- DealType: Type of sale
- DealCondition: Condition of sale
-

7.2 Final Winning Code

```
# hotel_prediction.py
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge, Lasso
import warnings

warnings.filterwarnings('ignore')

# --- LOAD DATA ---
try:
   train = pd.read_csv('train.csv')
   test = pd.read_csv('test.csv')
   print("Datasets loaded successfully!")
except FileNotFoundError as e:
   print(f"{e}")
   exit()

# --- ID column detection ---
id_col = 'Id' if 'Id' in train.columns else 'id'
print(f" Using '{id_col}' as ID column")

train_ids = train[id_col]
test_ids = test[id_col]
train = train.drop(id_col, axis=1)
test = test.drop(id_col, axis=1)

# --- TARGET COLUMN ---
target_col = 'HotelValue'
if target_col not in train.columns:
```

```
    raise ValueError(" 'HotelValue' column not found in train.csv!")

# --- FIX NEGATIVE VALUES ---
# Any negative numeric values that don't make sense are replaced with 0
num_cols = train.select_dtypes(include=[np.number]).columns
for col in num_cols:
    neg_count = (train[col] < 0).sum()
    if neg_count > 0:
        print(f" Found {neg_count} negative values in '{col}', replacing with 0")
        train[col] = np.where(train[col] < 0, 0, train[col])

# --- DELIVERY / ORDER FIX ---
# If both columns exist, fix delivery < order problem
if 'DeliveryDay' in train.columns and 'OrderDay' in train.columns:
    invalid_rows = train[train['DeliveryDay'] < train['OrderDay']].shape[0]
    print(f" Found {invalid_rows} rows where DeliveryDay < OrderDay")
    train.loc[train['DeliveryDay'] < train['OrderDay'], ['DeliveryDay', 'OrderDay']] = np.nan

# --- DROP TARGET ---
y = np.log1p(train[target_col])  # log-transform target
train = train.drop(columns=[target_col])

# --- COMBINE FOR CLEAN PREPROCESSING ---
all_data = pd.concat([train, test], ignore_index=True)

# --- HANDLE MISSING VALUES ---
for col in ['PoolQuality', 'ExtraFacility', 'FacadeType', 'BoundaryFence', 'LoungeQuality',
            'ParkingType', 'ParkingFinish', 'ParkingQuality', 'ParkingCondition',
            'BasementHeight', 'BasementCondition', 'BasementExposure',
            'BasementFacilityType1', 'BasementFacilityType2']:
    if col in all_data.columns:
        all_data[col] = all_data[col].fillna('None')

for col in all_data.columns:
    if all_data[col].dtype == 'object':
        all_data[col] = all_data[col].fillna(all_data[col].mode()[0])
    else:
        all_data[col] = all_data[col].fillna(all_data[col].median())

# --- FEATURE ENGINEERING ---
if 'YearSold' in all_data.columns and 'ConstructionYear' in all_data.columns:
    all_data['HotelAge'] = all_data['YearSold'] - all_data['ConstructionYear']
if 'RenovationYear' in all_data.columns and 'YearSold' in all_data.columns:
```

```
    all_data['YearsSinceRenovation'] = all_data['YearSold'] - all_data['RenovationYear']
if all(x in all_data.columns for x in ['BasementTotalSF', 'GroundFloorArea', 'UpperFloorArea']):
    all_data['TotalSF'] = all_data['BasementTotalSF'] + all_data['GroundFloorArea'] +
all_data['UpperFloorArea']
if all(x in all_data.columns for x in ['FullBaths', 'HalfBaths', 'BasementFullBaths',
'BasementHalfBaths']):
    all_data['TotalBathrooms'] = (
        all_data['FullBaths'] +
        0.5 * all_data['HalfBaths'] +
        all_data['BasementFullBaths'] +
        0.5 * all_data['BasementHalfBaths']
    )

# --- ENCODING ---
all_data = pd.get_dummies(all_data, drop_first=True)

# --- SPLIT BACK ---
X = all_data[:len(train)]
X_test = all_data[len(train):]

# --- SCALING ---
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_test_scaled = scaler.transform(X_test)

# --- RIDGE & LASSO MODELS ---
alphas_ridge = [14.5, 14.6, 14.7, 14.8, 14.9, 15.0, 15.1]
alphas_lasso = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005]

best_model, best_score, best_alpha = None, -999, None

print("\n--- Cross-Validation Results ---")
for alpha in alphas_ridge:
    model = Ridge(alpha=alpha)
    score = np.mean(cross_val_score(model, X_scaled, y, cv=10, scoring='r2'))
    print(f"Ridge (α={alpha}): {score:.4f}")
    if score > best_score:
        best_model, best_score, best_alpha = ('ridge', score, alpha)

for alpha in alphas_lasso:
    model = Lasso(alpha=alpha, max_iter=10000)
    score = np.mean(cross_val_score(model, X_scaled, y, cv=10, scoring='r2'))
    print(f"Lasso (α={alpha}): {score:.4f}")
```

```
    if score > best_score:
        best_model, best_score, best_alpha = ('lasso', score, alpha)

print(f"\n Best Model: {best_model.upper()} (α={best_alpha}) with CV R² = {best_score:.4f}")

# --- FINAL MODEL TRAINING ---
if best_model == 'ridge':
    final_model = Ridge(alpha=best_alpha)
else:
    final_model = Lasso(alpha=best_alpha, max_iter=10000)

final_model.fit(X_scaled, y)
preds_log = final_model.predict(X_test_scaled)
preds = np.expm1(preds_log)

submission = pd.DataFrame({id_col: test_ids, target_col: preds})
submission.to_csv('submission.csv', index=False)

print("\n 'submission.csv' created successfully!")
```