MMA 867 – Management of Data

Master of Management Analytics
Professor Jue Wang

Team Assignment: Kaggle Competition – Housing Prices
April 26, 2025

Student Name	Student Number
Jillian Gallays	20498531
Lavanya Ramachandran	20442213
Omar Mutleq	20503733
Rabab Handa	20500392
Sudip Roy	20497347
Thorn Leung	20533947
Yunying (Lillian) Liang	20547333

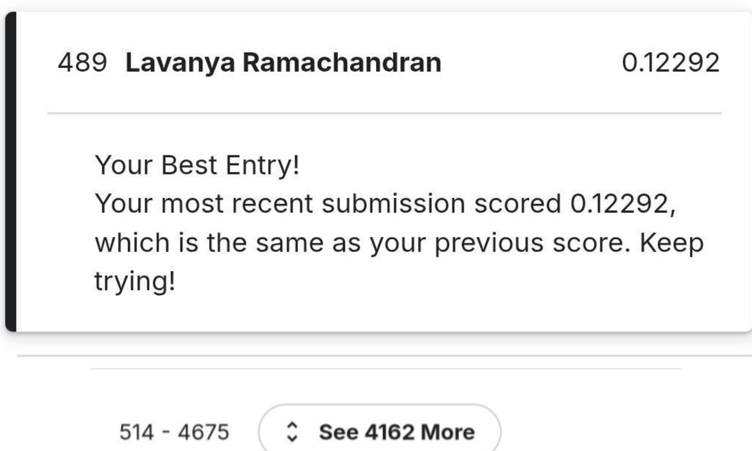
Table of Contents

Executive Summary

Given today's competitive housing market, buyers need clear insights to make smart home buying decisions. This analysis uses linear regression modeling to predict housing prices, helping buyers understand what types of homes or features they can afford within their budget.

Our team focuses on predicting housing prices using a dataset of residential properties, leveraging data cleaning, exploratory data analysis (EDA), feature engineering, and predictive modeling techniques. Our model identifies the relationship between these features and housing prices. Empowering buyers to make data-driven decisions by helping them understand home costs that align with their budget and preferences.

The analysis achieves a cross-validated Root Mean Square Error (RMSE) of 0.12315 on the log-transformed sale prices, indicating robust predictive performance. Key features influencing prices include total square footage, number of bathrooms, and quality-related attributes. Our team's model performed exceptionally well, ranking 489 out of 4,675 entries in the competition, placing us in the **top 11% of participants**. The resulting model offers a reliable tool for estimating property values, helping stakeholders navigate the housing market with confidence.



Techniques

Data Cleaning & EDA

The data cleaning process ensured the dataset was reliable for modeling by addressing missing values, ensuring consistency, and removing potential sources of error. The exploratory data analysis (EDA) then uncovered key patterns and relationships influencing housing prices, guiding feature selection and model development.

- **Null Value Handling:** Among the 80 features in the dataset, several categorical (e.g., *PoolQC*, *Alley*) and numerical (e.g., *LotFrontage*, *GarageYrBlt*) columns had missing values. Categorical features were imputed using the mode, while numerical ones were filled with the median to preserve data integrity and minimize bias.
- **Data Type Consistency:** Categorical variables were transformed into numeric format using LabelEncoder, ensuring compatibility with modeling tools. No inconsistent data types were found.
- **Duplicates and Empty Columns:** The dataset had no duplicate rows or empty columns, supporting a clean and structured foundation for analysis.
- **EDA Findings:** The dataset comprises 1,460 training rows and 1,459 test rows, with *SalePrice* as the target variable. EDA highlighted that size and quality-related features are the primary drivers of housing prices. A log transformation of *SalePrice* corrected right-skewness, improving model performance. Additionally, the presence of outliers indicates the need for caution in predicting prices for atypical homes.

Managerial Insight: The data cleaning process resolved missingness and ensured consistency, laying a solid foundation for accurate predictions. For stakeholders, this reliability minimizes errors in price estimation. EDA insights suggest that larger, better-quality homes in desirable locations fetch higher prices. Buyers can use this information to prioritize features within their budget, while sellers may consider upgrades to enhance property value. Future data collection should focus on reducing missing values to limit the need for imputation.

Feature engineering

To enhance the predictive accuracy of our regression models, we implemented transformations that addressed optimized model interpretability and aligned with practical considerations in real estate valuation.

- **Categorical Encoding:**

- Ordinal variables (e.g., *ExterQual*, *KitchenQual*) were label-encoded to retain their natural rank order.
- Nominal variables (e.g., *Neighborhood*, *SaleCondition*) were one-hot encoded to eliminate artificial hierarchies and ensure accurate representation.

- **Composite Feature Creation:**
  - *TotalSF* was introduced as the combined square footage across basement, first, and second floors.
  - *TotalBath* aggregated full and half bathrooms, weighted appropriately.
  - *YrBltAndRemod* served as a proxy for effective property age, combining year built and year remodeled. These features often demonstrated stronger correlation with SalePrice, capturing how buyers evaluate homes holistically.
- **Skewness Reduction:**
  - Logarithmic transformations were applied to highly skewed variables, including *GrLivArea* and *SalePrice*.
  - This step improved the model's adherence to linear regression assumptions and reflected diminishing marginal utility of added space.
- **Feature Scaling:**
  - Numeric features were standardized (zero mean, unit variance) for regularized models such as Ridge and Lasso.
  - This ensured no feature dominated due to scale, resulting in balanced coefficient estimates and improved model generalizability.

Managerial Insight: Feature engineering improved model accuracy and reflected real-world home valuation by contextually imputing missing values and encoding variables, enabling our model to handle diverse, incomplete housing datasets. Composite features like TotalSF and YrBltAndRemod revealed that larger, newer homes fetch higher prices, while scaling and transformation ensured fair pricing across varied properties, fostering transparent, data-driven decisions for buyers and sellers.

Predictive modelling

We implemented a stacked regression model that combines multiple regression techniques to improve prediction accuracy. This ensemble method leverages the strengths of different models to better capture the complex dynamics of housing data and included:

- **Lasso regression**: For feature selection — it removes less important variables and keeps the model interpretable.
- **Ridge regression**: For robustness — it shrinks all coefficients to reduce overfitting without eliminating predictors.

- **XGBoost (XGB) and LightGBM (LGB):** Tree-based models that capture complex, non-linear relationships between variables. These models create networks of decision trees, allowing them to model interactions that linear models may miss.
- **Model Stacking:** The stacked model layers and combines the outputs from each of these models, combining their individual strengths. This helps the model generalize better to new data, where accuracy depends on subtle combinations of features like neighborhood, quality, and square footage.

Model	**Strengths**
Lasso	Acts as a feature selector, dropping irrelevant ones
Ridge	Handles multicollinearity well
XGBoost	Excellent at capturing non-linear interactions and complex feature relationships
LGBM	Great for large datasets with high-dimensional features and handles categorical variables natively

Managerial Insight: Stacked regression provides real estate stakeholders with a reliable pricing tool, blending linear and non-linear insights, much like homebuyers weigh objective (e.g., square footage) and subjective (e.g., neighborhood appeal) factors. This ensemble approach mirrors real-world decision-making, delivering balanced, trustworthy estimates. Its flexibility ensures effectiveness across diverse properties, supporting consistent investment and purchase decisions.

Model Evaluation and Submission

To assess model performance, we used 10-fold cross-validation on the training data and calculated Root Mean Squared Error (RMSE) on the log-transformed *SalePrice*. This approach captured accuracy while accounting for price skewness and ensured a fair comparison across a wide range of home values.

- **Evaluation Metric:** RMSE on the log scale was chosen because it penalizes large errors and stabilizes variance, improving model reliability for both high- and low-priced properties.
- **Cross-Validation**: The RMSE of 0.12315 on log-transformed prices indicates consistent performance across data splits.
- **Prediction and Output:** After training on the full dataset, test set predictions were transformed back to the original scale using np.expm1(). A final CSV file was generated with only *Id* and *SalePrice* columns for the test dataset and submitted to Kaggle for evaluation.

Managerial Insight: This evaluation ensures model reliability through rigorous validation, aligning with industry standards. RMSE minimizes pricing errors, crucial for real estate decisions. Our Kaggle submission scored 0.12315 RMSE, ranking us in the top 11% (#489 of 4,675), validating our pipeline's robustness for real-world property valuation.

Conclusion

This project delivers a robust housing price prediction model, achieving a cross-validated RMSE of 0.12315 on log-transformed prices. Key price drivers include total square footage, bathrooms, and quality features, providing clear insights for buyers and sellers. The stacked ensemble model balances accuracy and robustness, though its complexity suggests future exploration of simpler alternatives or additional regularization. Stakeholders can leverage this tool to estimate property values confidently, with the caveat that outlier properties may require specialized handling. Future work could incorporate external data (e.g., market trends) to further enhance accuracy.

Modelling Approach & Strategy

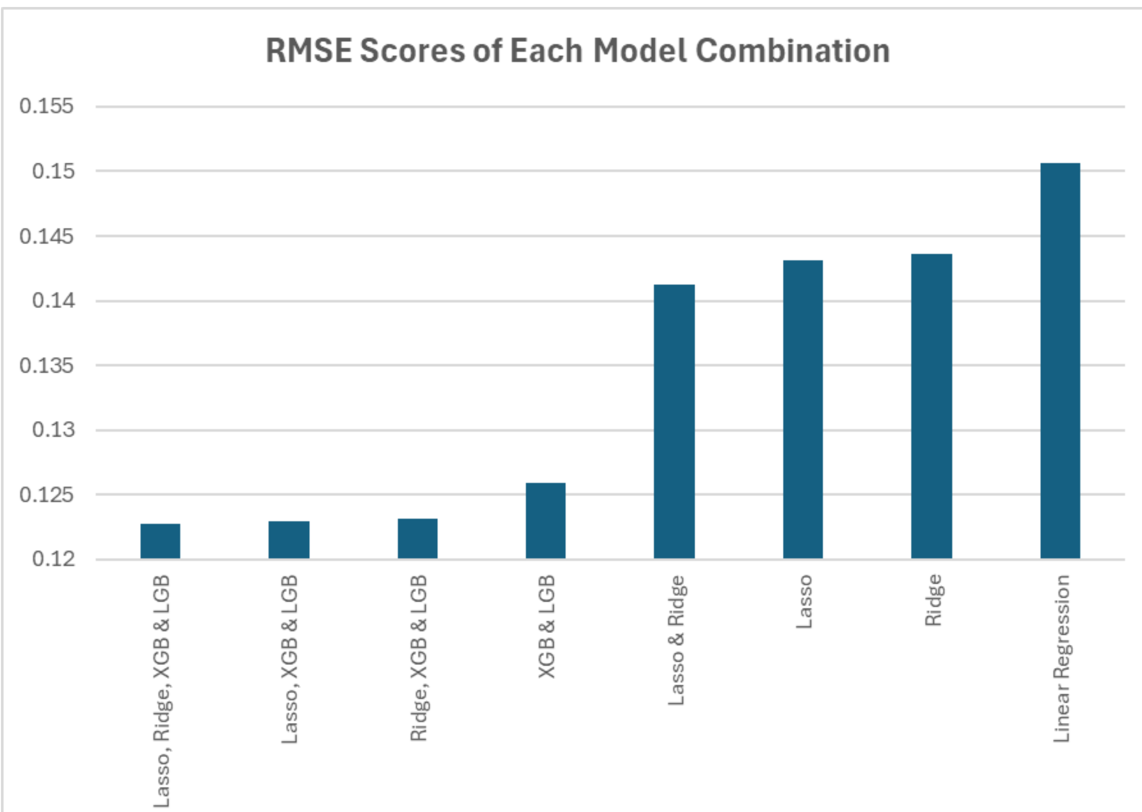
We started our modelling with Linear Regression (as learned in the previous course MMA 860) to set a baseline. This resulted in an RMSE of 0.15064, which we used as a reference point to evaluate more advanced models. We understood that Linear Regression assumes a linear relationship between features and the target (log-transformed SalePrice), but housing price data often involves non-linear relationships, interactions, and multicollinearity. The relatively high RMSE confirmed these limitations, so we decided to explore other regression techniques and their combinations to improve performance. We observed the following RMSE scores for each technique:

Model	RMSE
Linear Regression	submission.csv Complete · now 0.15064
Lasso	submission_lasso.csv Complete · now 0.14309
Ridge	submission_ridge.csv Complete · now 0.14360
Lasso & Ridge	submission_ridgelasso.csv Complete · now 0.14128
XGB & LGB	submission_lgbxgb.csv Complete · now 0.12595
Lasso, XGB & LGB	submission_lasso_lgbxgb.csv Complete · now 0.12297
Ridge, XGB & LGB	submission_ridge_lgbxgb.csv Complete · now 0.12314

Lasso, Ridge, XGB & LGB	submission_lassridge_lgbxgb.csv Complete · now 0.12278

Model Comparison

Our analysis showed that Lasso (0.14309) and Ridge (0.14336) outperformed Linear Regression (0.15064), highlighting regularization's role in reducing overfitting and multicollinearity, with Lasso's edge tied to its feature selection; a tuned Lasso in our modified notebook scored 0.12487, nearly matching XGB & LGB (0.12595), despite its weaker handling of non-linear patterns. Combining Lasso and Ridge (0.14128) slightly improved their individual results, while XGB & LGB excelled at capturing complex relationships. Stacking further boosted performance, with Lasso, XGB & LGB (0.12297), Ridge, XGB & LGB (0.12314), and the full stack (Lasso, Ridge, XGB & LGB: 0.12278) showing the benefits of combining Lasso's feature selection, Ridge's stability, and gradient boosting's non-linear modeling in an effective ensemble.



Therefore, we decided to proceed with the stacked combination of Lasso, Ridge, XGB & LGB as our final model as that helped us achieve the lowest RMSE.

Insights on Lasso and Ridge Regressions

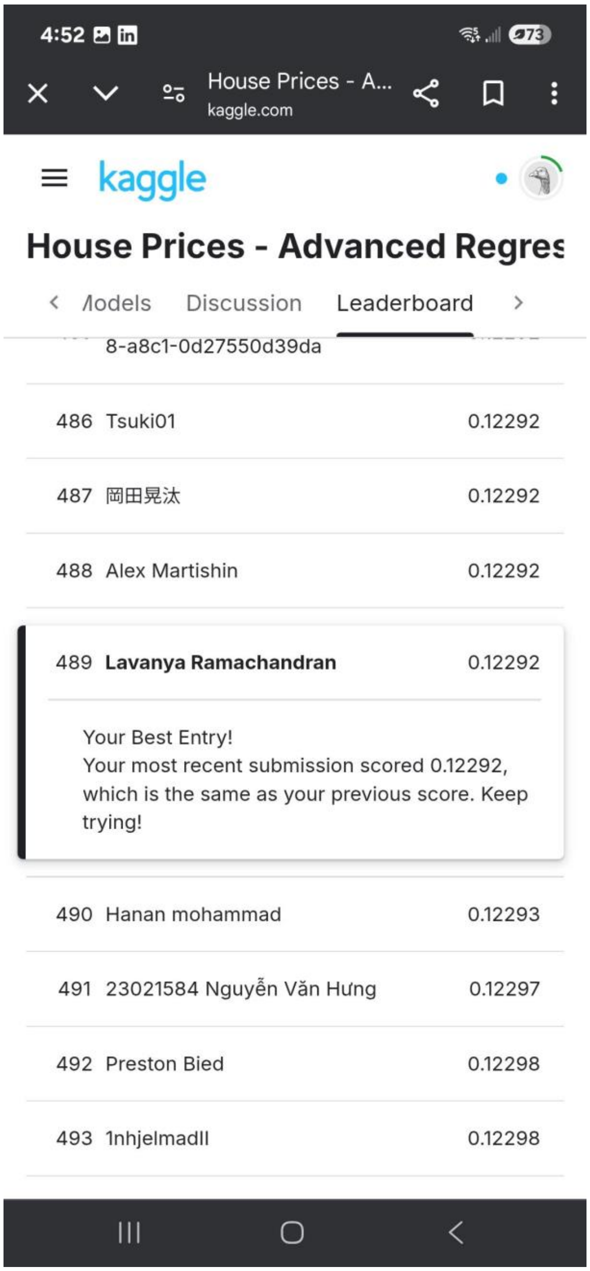
1. **Individual Performance:** Lasso (0.14309) and Ridge (0.14336) both outperformed our baseline Linear Regression (0.15064), showing that regularization mitigates overfitting and multicollinearity. Lasso's slight edge likely stems from its feature selection, shrinking less important coefficients to zero, while Ridge retains all features, potentially keeping some noise.
2. **Stacked Performance:** When added to XGB and LGB, Lasso (0.12297) outperformed Ridge (0.12314), indicating that Lasso's feature selection reduces noise in the ensemble, while Ridge stabilizes predictions by handling multicollinearity but introduces slight noise due to lack of feature elimination.

Lasso consistently outperformed Ridge in both individual and ensemble settings, leveraging its feature selection to reduce noise and enhance accuracy, while Ridge's strength in handling multicollinearity was less impactful for this dataset. Thus, Lasso is the better choice for our housing price prediction task where noise reduction through feature selection is key, though both methods add value in ensembles.

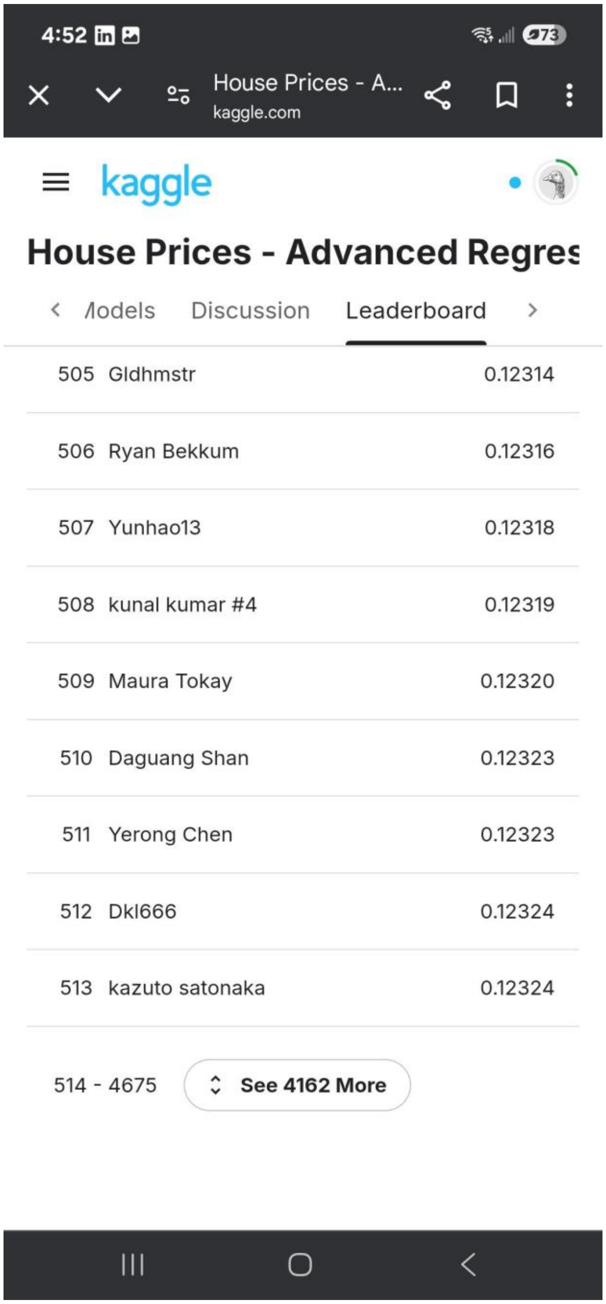
Appendix

Ranking

Individual Ranking - 489



Total Submissions – 4675



Code

Set-up & Library Imports

```
# %pip install openpyxl
# %matplotlib inline
# %pip install statsmodels
```

```
# %pip install scikit-learn seaborn
# %pip install jupyter_contrib_nbextensions
# %pip install --upgrade scikit-learn
# %pip install lightgbm xgboost scikit-learn --quiet
# %pip install missingno
import os

import pandas as pd
import numpy as np

import missingno as msno

import statsmodels.imputation.mice as mice
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.diagnostic import het_breuschpagan

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import Ridge, Lasso
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline
from sklearn import linear_model
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.impute import SimpleImputer
from sklearn.ensemble import StackingRegressor

from patsy import dmatrices
import scipy.stats as stats

import matplotlib.pyplot as plt
import seaborn as sns

from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
#Converting data source to dataframes
file_path_test  = "test.csv"
file_path_train = "train.csv"

test  = pd.read_csv(file_path_test)
```

```
train = pd.read_csv(file_path_train)

data = pd.concat([train, test], sort=False)
```

Data Cleaning & EDA

```
# Fill categorical with mode
cat_cols = data.select_dtypes(include='object').columns
for col in cat_cols:
    data[col] = data[col].fillna(data[col].mode()[0])
    lbl = LabelEncoder()
    data[col] = lbl.fit_transform(data[col].astype(str))

# Fill numerical with median
num_cols = data.select_dtypes(include=['int64', 'float64']).columns
for col in num_cols:
    data[col] = data[col].fillna(data[col].median())
```

Feature Engineering

```
#Combining features for Bathrooms to reduce dimensions
data['TotalSF'] = data['TotalBsmtSF'] + data['1stFlrSF'] + data['2ndFlrSF']
data['TotalBath'] = (data['FullBath'] + data['HalfBath'] * 0.5 +
                     data['BsmtFullBath'] + data['BsmtHalfBath'] * 0.5)

#Combining features for porches to reduce dimensions
data['TotalPorchSF'] = (data['OpenPorchSF'] + data['EnclosedPorch'] +
                        data['3SsnPorch'] + data['ScreenPorch'])

#Converting numerical features to Categorical (binary features make
presence/effect of features more explicit)
data['HasPool'] = (data['PoolArea'] > 0).astype(int)
data['HasGarage'] = (data['GarageArea'] > 0).astype(int)
data['HasFireplace'] = (data['Fireplaces'] > 0).astype(int)
```

Regression Modelling

```
#Split data into train & test sets
train_clean = data[:len(train)].copy()
test_clean = data[len(train):].copy()
train_clean['SalePrice'] = train['SalePrice']

#Log transforming target to reduce skewness in data
```

```
y = np.log1p(train_clean['SalePrice'])

X = train_clean.drop(['Id', 'SalePrice'], axis=1)
X_test = test_clean.drop(['Id', 'SalePrice'], axis=1)

#Defining base models
#Using pipeline method to chain steps & uses features in similar scaling
ridge = make_pipeline(RobustScaler(), Ridge(alpha=15))
lasso = make_pipeline(RobustScaler(), Lasso(alpha=0.0005))

#Usring XBG & LGB models to handle complex feature and target relationships
xgb = XGBRegressor(n_estimators=1000, learning_rate=0.05, max_depth=3,
                   subsample=0.7, colsample_bytree=0.7, random_state=42)
lgbm = LGBMRegressor(objective='regression', num_leaves=5, learning_rate=0.05,
                     n_estimators=1000, random_state=42)

#Stacking all models
stacked_model = StackingRegressor(
    estimators=[('ridge', ridge), ('lasso', lasso), ('xgb', xgb), ('lgbm',
lgbm)],
    final_estimator=Ridge(alpha=10)
)
```

Model Evaluation

```
#Calculating Root Mean Square Error
def rmse_cv(model):
    kf = KFold(n_splits=10, shuffle=True, random_state=42)
    rmse = -cross_val_score(model, X, y, scoring="neg_root_mean_squared_error",
cv=kf)
    return rmse.mean()

print(f"Stacked Model CV RMSE: {rmse_cv(stacked_model):.5f}/n")
```

Predict Values for Competition & Submission

```
#Fit and predict
stacked_model.fit(X, y)
final_preds = np.expm1(stacked_model.predict(X_test))

#Export csv with predictions for comptetion submission
submission = pd.DataFrame({
    'Id': test['Id'],
    'SalePrice': final_preds
```

```
})
submission.to_csv('submission.csv', index=False)
print("Submission file saved: submission.csv")
```