Report on MLM Project

House Pricing Prediction

Made by:

Gonçalo Correia

September 2025

Index

List of Tables

List of Figures

Report Structure

Each major section ends with a *Summary* subsection highlighting the main findings and decisions.

1 Introduction

The prediction of housing prices is a long-standing problem, both in the real estate market and economics. Given the wide variety of factors that influence property values, predicting that value becomes a good target for a data-driven approach. To this end, we used the Kaggle *House Prices: Advanced Regression Technique* competition, since it provides a dataset with a wide variety of variables for both training and testing of the models. The fact that the true values of the properties used for testing are unknown to us, will assure that the final metric isn't biased in any way.

To tackle this problem we began by preprocessing data, creating new features and carefully treating existing ones to address any inconsistencies. we then trained models to understand which algorithm and approaches are better suited to the task. Finally, the best scoring models were combined into an ensemble to produce a final prediction.

2 Data Understanding

The data provided by the Kaggle competition is composed of 1460 entries, each with 81 variables, including transaction ID and the sale price. Of the available variables , 47 of them are numerical in nature. The other variables are categorical, but, within this set, a distinction must be made between ordinal and nominal categorical variables. Within this subset of categorical variables, only OverallCond and OverallQual are ordinal features, which are ratings on a 0-10 scale. In order to spare some time, the list of variables and their respective descriptions were attached to the report.

2.1 Missing Values

When it comes to missing values, the following results were obtained:

Feature	Missing (%)	Missing Count
PoolQC	99.52	1453
MiscFeature	96.30	1406
Alley	93.77	1369
Fence	80.75	1179
MasVnrType	59.73	872
FireplaceQu	47.26	690
LotFrontage	17.74	259
GarageQual	5.55	81
GarageFinish	5.55	81
GarageType	5.55	81
GarageYrBlt	5.55	81
GarageCond	5.55	81
BsmtFinType2	2.60	38
BsmtExposure	2.60	38
BsmtCond	2.53	37
BsmtQual	2.53	37
BsmtFinType1	2.53	37

Table 1: Missing Values in Dataset

Unlike some other datasets, here the non-existence of a value doesn't imply missing information, but a missing feature in the property. The high count of missing values in `PoolQC` only indicates that most houses do not have a pool, thus, the absence of information here corresponds to the absence of the feature itself.

In the case of our example, we replaced the variable with a new binary feature indicating whether the house has a pool (1) or not (0). The original `PoolQC` variable is then removed. The same process was then applied to the variables `MiscFeature`, `Alley` and `Fence`.

2.2 Skewness

It was also noted that some variables were also subject to skewness, which can influence the results of linear models, since those models assume a normal distribution in residuals, which might not happen with the skewed variables. A solution to this problem will be discussed further on the feature engineering section.

2.3 Data Visualization

Further exploring the features, relationships between them were noted. For example, by plotting the average sale price for each neighborhood, we obtained the results in Figure 1.

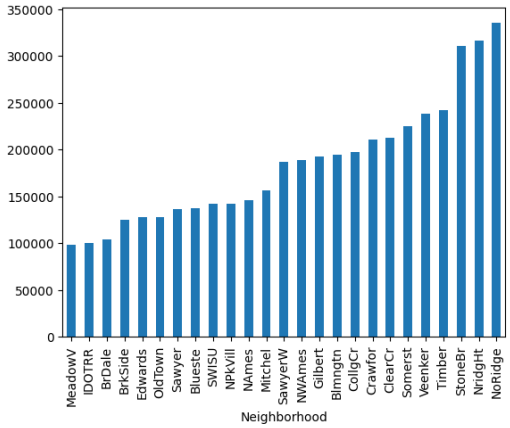


Figure 1: Neighborhood's average sale price

The non-existence of uniformity indicates that the neighborhood might influence the sale price. This figure more or less, mirrors the trend obtained by plotting the average house's area in each neighborhood, as can be seen in figure 2.

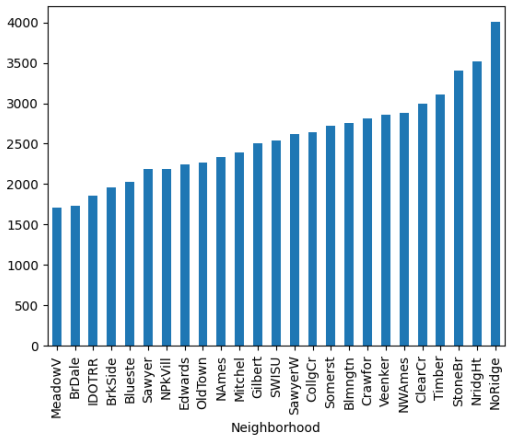
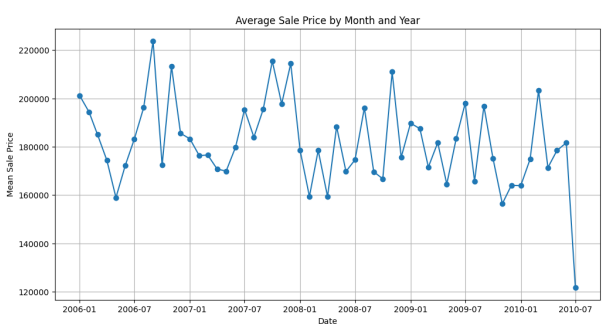
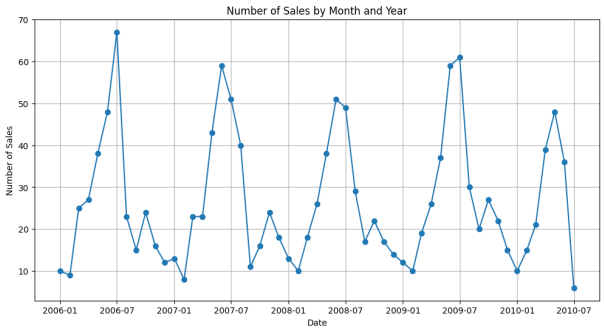


Figure 2: Average house size by neighborhood

In Figure 3 we can see some seasonality in both plots. In 3a, it is notable that the average sale price tends to decline during the first semester of the year and then follows it up with an increase during the second semester. The behaviour of 3b is more repetitive than the one observed on the average sale price, with the number of sales increasing during the first semester, and declining during the second half of the year.



(a) Price over Time



(b) Number of sales over Time

Figure 3: Comparison of trends over time.

2.4 Summary

In this section, a distinction was made between different types of features (numerical, categorical, and ordinal). For variables with a high proportion of missing values, we replaced them with binary indicators to reflect the presence or absence of the corresponding feature. We also observed considerable skewness in several numerical features, which will be addressed later (see section 3.2).

Exploratory analysis revealed a strong relationship between neighborhood and house prices, as well as clear seasonal patterns in the number of sales, with activity peaking during the summer months.

3 Feature Engineering

3.1 Creating new features

As is standard in data science, we then went on to create new features out of the ones that we already had, in hopes of helping the models understand certain relations that, otherwise, might be difficult to catch. In Table 2, we can see which features were created and their respective meanings.

Variable	**Meaning**
Grade	Product of `OverallQual` and `OverallCond`
Total_Bathrooms	Total number of bathrooms
LotAreaRatio	Proportion of the lot's area to the lot's frontage
Bed_Bath_Ratio	Number of bathrooms per bedroom
TimeSinceRemodel	Years passed since last remodel
HouseAge	House's age
TotalSF	Total house area in square feet

Table 2: Engineered Variables and Their Meaning

To further help some of the linear models that we plan to use in the future, we created the variables `AreaQual`, `GarageLux` and `SpaceQual`. These features do not correspond directly to tangible house characteristics, but instead combine existing information in a way that helps linear models uncover patterns that may otherwise be missed. The variable `LuxuryFlag` was also created, which indicates whether a house is 'luxurious' or not. This feature is particularly helpful for linear models, which tend to struggle with extremely high-value houses. As shown in the code, these variables were used only for the linear models and kernel methods since that is where their contribution is most relevant.

3.2 Treating Skewness

As explained before, there are some features that suffer from considerable skewness, to address these, we applied a logarithmic function to them. While this approach mostly solves the problem in this context, it can only be applied to variables that are strictly positive.

3.3 Summary

In this section we explored the newly created features and how some of them might help some types of algorithms. It was also briefly explained how the problem of the skewed variables was resolved.

4 Modelling

4.1 Evaluation Metric

The competition uses the Root Mean Squared Error (RMSE) of the logarithm of the predicted sale price. In practice, this means we train our models on `log(SalePrice)` and evaluate predictions according to the following metric:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(\log(\hat{y}_i) - \log(y_i)\big)^2}.$$

This metric in particular allows us to quantify relative error rather than absolute differences, to ensure that the price of the expensive houses do not dominate the results.

4.2 Hyper-parametrization and Pipeline

Before presenting the chosen models, we outline the strategy used for their optimization and evaluation. Since every model has a set number of parameters and they influence how well the model performs, we need to find the set of values that gives us the best model. For this we will be using the *Optuna* package. The algorithm starts by proposing random parameter sets and storing their results. Then, it uses those results to search for better ones. The number of trials is inserted by the user. Although 100-200 trials is usually enough for *Optuna* to find a good enough result, the number of trial used in our case varied according to the results that were obtained.

For each *Optuna* trial we also used cross-validation, which, in simpler terms, divides the training data in equal size chunks, trains on some and test on others; after that it calculates the average performance of each model and feeds it to *Optuna*. Cross-validation allows us to assess the ability of the model, trained with a given set of parameters, to generalize its predictions to unseen data.

To further help our models, we created a pipeline, that target encodes our categorical variables, and standardizes our numeric ones.

4.3 Tree-Based Gradient Boosting Models

Gradient boosting methods are a very powerful family of algorithms for tabular data. They start by building ensembles of decision trees where each new one tries to correct the errors present in the previous ones. For this project we used the most popular implementations: CatBoost, XGBoost, LightGBM and Gradient Boosting Regressor. Each one of these implementations has its own advantages and disadvantages.

4.3.1 CatBoost

CatBoost is a gradient boosting algorithm designed with categorical features in mind. Unlike most boosting libraries, that require a previous encoding of categorical variables, CatBoost does this internally.

Another feature that makes this a good model to use, is its ordered boosting, which, in simple terms, uses only previous rows for target encoding, reducing the risk of target leakage.

4.3.2 XGBoost

One of the most popular gradient boosting algorithms is XGBoost. Its use of regularization parameters such as L_1 and L_2 helps the model generalize better and manage overfitting.

When dealing with missing values, XGBoost is able to deduce the optimal direction, making it a good choice when our dataset has incomplete information.

These characteristics, along with its computational optimizations, enable XGBoost to train faster than conventional gradient boosting techniques without sacrificing much of its predictive power.

4.3.3 LightGBM

LightGBM is a gradient boosting framework that is optimized for speed and memory efficiency. Instead of exploring every possible split point across a continuous range, it uses a histogram-based binning process, where the values are grouped into bins.

Another twist is that this method tends to focus on the most promising leaves of its tree, allowing it to develop deeper trees and higher accuracy, although it can also increase the risk of overfitting.

4.3.4 Gradient Boosting Regressor

This is the most classical approach in this family of algorithms without relying on advanced optimization techniques like its more recent counterparts. Although methods such as LightGBM and XGBoost usually outperform it, our results showed the opposite, which highlights the capabilities of gradient boosting methods in contexts like this.

4.4 Linear Models

To ensure diversity in the final ensemble, it was also tested a set of linear models with regularization. These models assume a linear relationship between the variables and their targets, and, although this might seem a limiting feature, with the help of the regularization techniques from LassoLars and ElasticNet, the models are able to generalize a bit better without making the process slower.

4.4.1 ElasticNet

ElasticNet combines both L_1 (Lasso) and L_2 (Ridge) regularization, allowing it to both shrink coefficients and perform variable selection. This makes it particularly useful when features are correlated, since it balances sparsity and stability in the solution.

4.4.2 LassoLars

LassoLars is a version of Lasso regression that uses the Least Angle Regression (LARS) algorithm to efficiently compute the full regularization path. This makes it effective in high-dimensional settings,

as it can quickly identify a sparse subset of important features.

4.5 Support Vector Regression

The last model used was the Support Vector Regression (SVR), which belongs in the family of kernel methods, which have a more abstract formulation and are harder to visualize. SVR, one of the most popular Kernel methods, tries to find the function that better replicates the target, within a given margin of tolerance, while trying, at the same time, to keep the function as simple as possible.

In table 3, we can see the best RMSE each model got in Cross-Validation.

Model	**Best RMSE**
CatBoost	0.11759
XGBoost	0.12320
LightGBM	0.12184
Gradient Regressor	0.11839
ElasticNet	0.12730
SVR	0.12950
LassoLars	0.12780

Table 3: Model Performance Comparison

In Table 3, the tree-based models were the clear winners in terms of performance. With our models trained and tested we could now do our final ensemble.

4.6 Summary

In this section we compared three families of models: tree-based gradient boosting methods (CatBoost, XGBoost, LightGBM and Gradient Boosting Regressor), regularized linear models (ElasticNet and LassoLars), and a kernel method (Support Vector Regression). The cross-validation results showed that the linear models and kernel methods were often outperformed by the tree-based gradient boosting methods on this dataset, with CatBoost and Gradient Boosting Regressor obtaining the best results.

While the linear and kernel models obtained less accurate results, they still bring some diversity and contributed to improving the final ensemble.

5 Smearing

Since the target variable is the house's sale price, which spans a wide range of values, we applied a logarithmic transformation to stabilize variance and make the prediction task easier. As a result, the models are trained to predict the logarithm of the sale price.

While it might seem logical to directly apply an exponential function to the predicted value, i.e. use $e^{\hat{y}}$, this is not correct. By Jensen's Inequality, since the exponential function is convex and given that our target isn't constant we have

$$e^{E[Y]} < E[e^Y].$$

This means that directly applying the exponential function to the predicted log-price will underestimate the true expected sale price.

To address this bias, Duan's smearing estimator was applied. The estimator is obtained from the residuals in the training data:

$$\text{Smear factor} = \frac{1}{n}\sum_{i=1}^{n} e^{\varepsilon_i},$$

where $\varepsilon_i = \hat{y}_i - y_i$ are log-space residuals. The final predictions without any bias are obtained as

$$\hat{y}_{\text{final}} = e^{\hat{y}} \times \text{Smear factor}.$$

5.1 Summary

To stabilize variance, the target variable was log-transformed before model training. However, directly exponentiating predictions leads to underestimation due to Jensen's Inequality. To adjust this problem of inequality, we applied Duan's to all predictions. This adjustment ensures unbiased results on the original price scale.

6 Ensembling

For the final ensemble there was the option of a simple average between the models, but, as can be seen in Table 3, the models have drastically different performances, and, thus, giving them the same weight could result in a lackluster result.

We implemented a simple optimization procedure using Optuna to search for the set of ensemble weights that minimized cross-validation error. While this approach does not guarantee a globally optimal solution, it provided a practical and easy way to approximate a good weighting scheme for combining the models. Figure 5 presents Optuna's suggested weights.

	CatBoost	**XGBoost**	**LightGBM**	**GBR**	**ElasticNet**	**SVR**	**LassoLars**
Weight	0.4409	0.00015	0.00018	0.5376	0.02044	0.000402	0.00026

Table 4: Ensemble Weights for Each Model, proposed by Optuna

While the weights proposed by Optuna did not lead to the optimal performance, they provided a good starting point by indicating which models should contribute more to the ensemble (e.g., CatBoost and GBR).

Using these results, we manually explored alternative weight configurations. Through iterative testing, we arrived at the set of weights shown in Table 5, which offered a more conservative balance between models and achieved stronger results.

	CatBoost	XGBoost	LightGBM	GBR	ElasticNet	SVR	LassoLars
Weight	0.25	0.10	0.20	0.35	0.00	0.00	0.10

Table 5: Ensemble Weights for Each Model

7 Results

In Table 6 we can see the result from each individual model, and also the ensemble (using simple means, Optuna's weights, and our weights).

Model	Result
CatBoost	0.12400
XGBoost	0.12235
LightGBM	0.12248
Gradient Regressor	0.12228
ElasticNet	0.14492
SVR	0.13096
LassoLars	0.14360
Simple Average	0.12140
Using Optuna's weights	0.11878
Weighted Prediction	0.11832

Table 6: Model Results and Ensemble Performance

From these results we can conclude that by using the weights proposed in the previous section we can get a better model than the ones obtained through any of the other options. The proximity of these result to the obtained with Optuna's weights, shows some of the strengths of that approach that, if refined, could perhaps achieve even better results.

On Kaggle's public leaderboard, our final ensemble achieved an RMSE of 0.11832, placing 77 out of 4032 participants (top 2%, as of 11/09/2025).

8 Possible improvements

From where we stand one of the paths that could possibly improve our results would be to make better use of the `neighborhood` variable. While it already provides information on location, additional spatial features could be constructed, such as the distance of each property to key points of interest (e.g., schools, hospitals, or shopping centres).

Another potential improvement lies in our ensemble weighting strategy. In this project they were assigned heuristically. A more systematic approach could possibly give us a significant improvement in our results.

9 Conclusion

This project provided the opportunity to work on a regression problem, rather than the binary classification problems I've become used to. The structure of the dataset led me to learn about new

models such as CatBoost, LassoLars, SVR, and ElasticNet.

Another important subject that I learned while doing this project is the different techniques for ensembling, and, although I didn't explore very advanced techniques, experimenting with weighted averaging and optimization through Optuna gave me the foundation to build better ensembles in future projects.

The most valuable takeaway was the use and importance of the smearing factor, something that fascinated me. This small but substantial detail can affect even the best models. Overall, this project not only delivered a competitive result but also expanded my understanding of how mathematical theory and practice work together to solve real-world prediction problems.

10 Used Libraries

Package	Description
Pandas [7]	Data manipulation and analysis library providing high-performance data structures such as `DataFrame` for handling tabular data.
NumPy [3]	Fundamental library for numerical computing in Python, supporting fast multi-dimensional arrays, linear algebra, and mathematical operations.
Matplotlib [4]	Plotting library for creating static, interactive, and animated visualizations in Python. Widely used for data exploration and presentation.
XGBoost [2]	Gradient boosting framework optimized for performance and scalability. Includes regularization, efficient handling of missing values, and parallel training.
LightGBM [5]	Gradient boosting framework developed by Microsoft that uses histogram-based learning and leaf-wise tree growth for efficiency and accuracy.
scikit-learn [8]	Machine learning library offering tools for preprocessing, model selection, training, and evaluation across a wide range of algorithms.
CatBoost [9]	Gradient boosting library from Yandex designed to handle categorical features efficiently and reduce overfitting via ordered boosting.
Optuna [1]	Hyperparameter optimization framework that uses efficient search strategies and pruning to automate tuning.
Category Encoders [6]	Library providing a wide range of encoding techniques to transform categorical variables into numerical form.

Table 7: Summary of main packages used in the project.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794. ACM, August 2016.

[3] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[4] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qi Ye, T.-Y. Liu, et al. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, pages 3149–3157, 2017.

[6] Will McGinnis. *Category Encoders*, 2018.

[7] Wes McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51–56, 2010.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Liudmila Prokhorenkova, Gleb Gusev, Alexey Vorobev, Anna V. Dorogush, and Andrey Gulin. Catboost: Unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 6638–6648, 2018.