COMP 4433

Data Mining and Data Warehousing

Competition (Team Effort) - Fall'24

House Price Prediction

Team: *HouseKeepers*

Members:

AKTAS Bora (24010776X)

LOENNECKEN Kaja (24010845X),

ALICE Anmaria (21095219D),

TANDIONO Ryan (21104279D)

Github Repository:

https://github.com/boraaktas/COMP4433-Team_Project

Table of Contents:

1 Introduction....3
2 Final Score in Kaggle....3
3 Way to solution....3
3.1 Dataset Preprocessing....3
3.1.1 Missing Values....4
3.1.2 Dataset Analysis and Visualization....7
3.1.3 ARM Between Categorical Features....9
3.1.4 Outlier Analysis....10
3.2 Feature Engineering and Selection....11
3.2.1 Principal Component Analysis (PCA)....11
3.2.2 LASSO....12
3.2.3 Random-Forest....12
3.3 Model Selection and Results....13
3.3.1 Model Selection....13
3.3.2 Results....13
4 Conclusion:....14

1 Introduction

In this report, we present our team's way to solution and findings from participating in Kaggle's House Price Prediction competition. The goal of the competition is to predict house prices based on a dataset of housing attributes, consisting of both numerical and categorical features. Beyond achieving competitive results, our primary objectives included exploring and implementing advanced data mining solutions, and developing practical experience in data preprocessing, feature engineering, and model selection. We began by thoroughly understanding the dataset, identifying its challenges and developing solutions tailored to these challenges. These included handling missing values, detecting outliers, addressing multicollinearity and testing a range of machine learning models. Through this report, we document our methods, insights and the rationale behind key decisions, culminating in a reflection on our performance in the competition.

2 Final Score in Kaggle

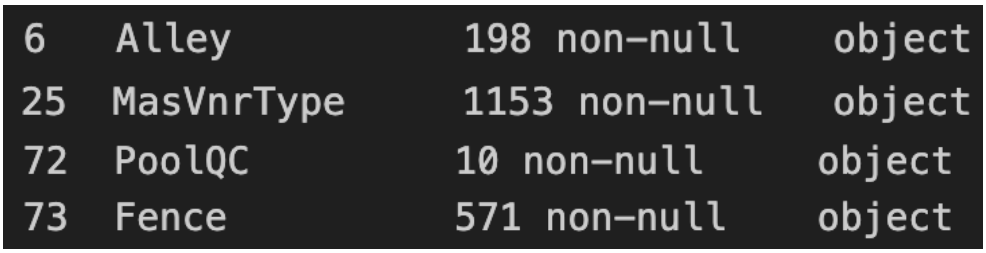
3 Way to solution

3.1 Dataset Preprocessing

Prior to conducting any sort of analysis on the house pricing dataset, we believe it would be wiser for us to analyze and get a high understanding of what this dataset is actually about as well as a guidance for conducting our analysis.

First, we were provided with two datasets; a training set where we will conduct our analysis and apply data mining techniques, as well as a testing set where we will find out if our analysis works well on unseen data. We decided to create a third dataset by combining the two together, resulting in a complete dataset (WHOLE_DATA) that we will be using to conduct analysis in a more efficient manner.

Running the code WHOLE_DATA.info(), we are able to understand all the different features, look for missing values, categorical as well as numerical features.



6	Alley	198 non-null	object
25	MasVnrType	1153 non-null	object
72	PoolQC	10 non-null	object
73	Fence	571 non-null	object

Fig. 1: Count of non-null values in selected features

Fig. 1 shows some of the features that noticeably have plenty of missing values. Therefore, for the first step in cleaning the data, we will deal with missing values in the dataset. Below, we will conduct a deeper analysis on each feature and how their missing values are related to one another.

3.1.1 Missing Values

First of all, from the description of the data, we observed that NaN values in certain columns (PoolQC, GarageType, etc,) indicate the absence of a feature rather than missing information. In these categorical columns, NaN represents a meaningful category (e.g., 'No Pool' or 'No Garage') Since most packages do not accept NaN values, we replaced these NaN values with a string identifier, 'NaN'. For this part we focused on the columns that are not to be allowed to have NaN values.

Since there are various methods to handle missing values, we believe it would be more efficient and accurate for us to separate the features into numerical and categorical types. This distinction allows us to apply appropriate imputation techniques tailored to each data type. For numerical features, common methods include imputing with the mean, median, or mode. However, just blindly replacing these missing values with either the mean, median or mode would be naive and considerably inaccurate.

Therefore, we first aimed to understand the missing values´ patterns by plotting the correlations amongst each feature. The correlation heatmap we generated from the training data provides insights into the relationships between variables and set the foundation for our missing values analysis.

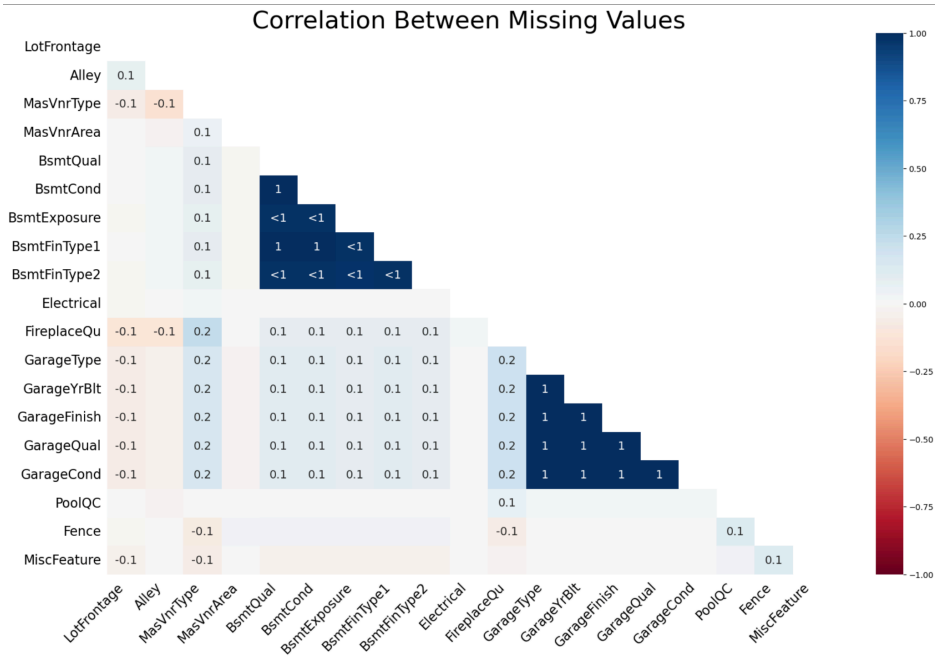


Fig. 2: Correlation-matrix of missing values

As we can see from the heatmap in Fig. 2, multiple features are strongly correlated with one another (dark blue), we can use this information when filling in missing values for those highly correlated features.

For example, the features ‘LotFrontage’ and ‘Electrical’ have little to no correlation with all the other features, therefore we could just impute the missing values with either their mean or mode. For the feature ‘Electrical’, we decided to impute it by using mode because there is just one missing value in the train set for this column. On the other hand, 486 rows (in both train and test set) have missing values for the feature ‘LotFrontage’. We decided to impute these missing values by using more advanced techniques than using means such as KNN imputers.

As for other features such as ‘GarageYrBlt’, it has a perfect correlation with other garage related features. We conduct ARM analysis on garage features in the train set and below are the results.

	Body_Name	Body_Value	Head_Name	Head_Value	Support	Confidence	Lift
0	GarageCond	TA	GarageYrBlt	Exists	0.908219	1.0	1.058738
1	GarageCond	Fa	GarageYrBlt	Exists	0.023973	1.0	1.058738
2	GarageCond	NaN	GarageYrBlt	NaN	0.055479	1.0	18.024691
3	GarageCond	Gd	GarageYrBlt	Exists	0.006164	1.0	1.058738
4	GarageCond	Po	GarageYrBlt	Exists	0.004795	1.0	1.058738
5	GarageCond	Ex	GarageYrBlt	Exists	0.001370	1.0	1.058738

Fig. 3: ARM-analysis for Garage-features

Logically if a house does not have a garage thus all the other garage related features would be NaN as well. We can verify this from the table, whenever the feature ‘GarageCond’ is NaN, then everything else is also NaN because of the confidence rule. If in the test and train set, there are rows where ‘GarageCond’ is NaN and ‘GarageYrBlt’ is also NaN then we can fill the missing values in ‘GarageYrBlt’ as 0. Because otherwise, we would have to impute its value by using the mean because when ‘GarageCond’ is not NaN, ‘GarageYrBlt’ is not NaN.

Next would be the ‘MasVnrArea’ feature, which has a strong correlation ‘MasVnrType’. Once again we start by conducting the ARM on these features. Below are the results.

	Body_Name	Body_Value	Head_Name	Head_Value	Support	Confidence	Lift
0	MasVnrType	BrkFace	MasVnrArea	Exists	0.304110	0.997753	2.464838
1	MasVnrType	BrkFace	MasVnrArea	Exists as 0	0.000685	0.002247	0.003811
2	MasVnrType	NaN	MasVnrArea	Exists	0.003425	0.005734	0.014165
3	MasVnrType	NaN	MasVnrArea	Exists as 0	0.588356	0.985092	1.670423
4	MasVnrType	NaN	MasVnrArea	NaN	0.005479	0.009174	1.674312
5	MasVnrType	Stone	MasVnrArea	Exists	0.086986	0.992188	2.451089
6	MasVnrType	Stone	MasVnrArea	Exists as 0	0.000685	0.007812	0.013248
7	MasVnrType	BrkCmn	MasVnrArea	Exists	0.010274	1.000000	2.470389

Fig. 4: ARM-analysis for MasVnr

Similarly, when ‘MasVnrType’ is NaN, then the area should logically be 0 as well. (confidence rule is almost 1). After gathering all the relevant rules, we first impute missing values that follow these rules. But after that, we still have some missing values in both numerical and categorical features. So we decided to impute these missing values (that is not following strong rules we found with ARM) by using KNN imputer.

3.1.2 Dataset Analysis and Visualization

Following our goal in understanding the dataset in detail, we decided to visualize the distribution of our target variable ‘SalesPrice’ as well as the log of ‘SalesPrice’. Below are the plots together with some statistics about the distribution.

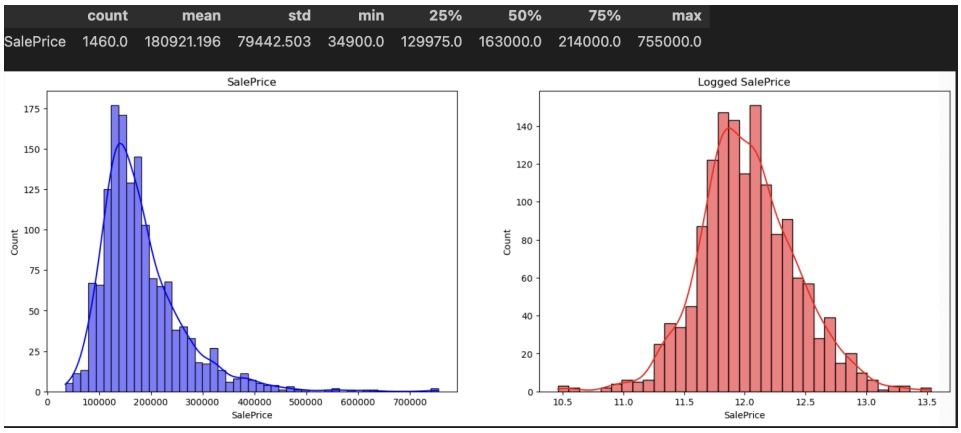


Fig. 5: Distribution Plot of ‘SalePrice’ and log of ‘SalePrice’

We notice a positively skewed distribution with a high concentration of housing prices around the $180,000 level. However, when we logged the variable and plotted its distribution, we noticed something similar to the well-known normal distribution. So we decided to use log-based SalePrice for training.

Fig. 6 provides some examples of the distribution of categorical features (we did not attach all 44 columns for readability purposes, refer to the jupyter notebook for the remaining columns).

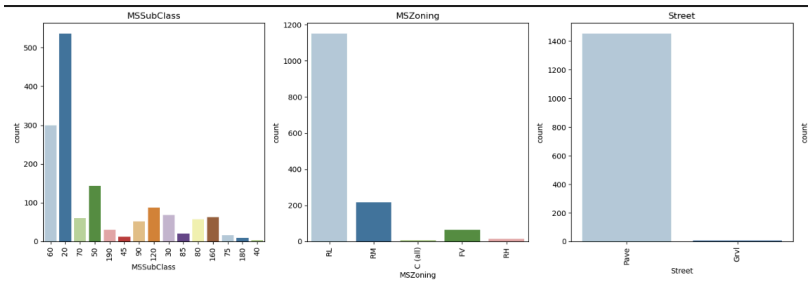


Fig. 6: Distribution of categorical features

Next would be the plot of the number of unique values per categorical features which will help with further analysis on each categorical feature below.

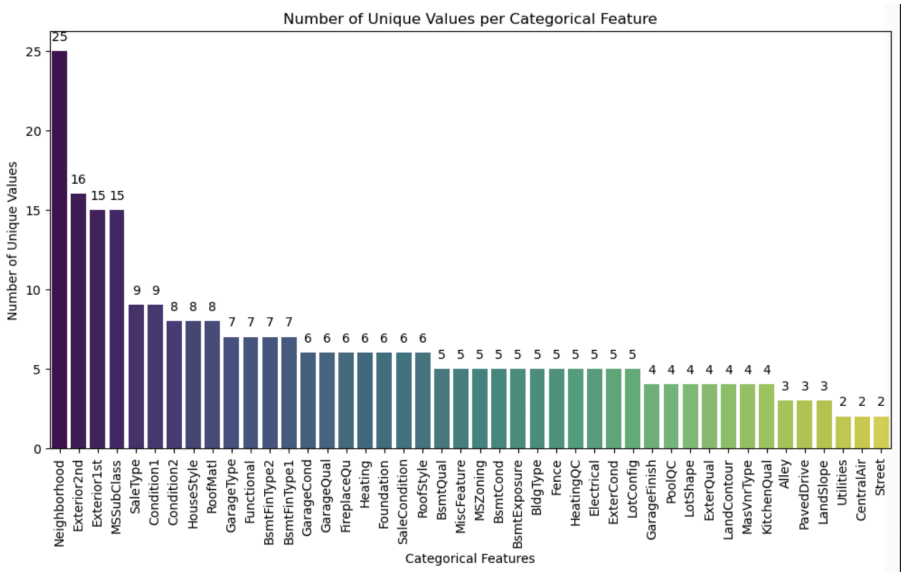


Fig. 7: Plot of number of unique values per categorical feature

Using the 2 plots above, we go through the distribution plots for all 44 categorical features, and observe that many features have a skewed distribution, where certain categories often appear while others are either absent or rarely occur. The absence of certain types in the categorical features could be problematic when encoding categorical features in the training set as the test set could have different columns. To avoid this problem, we further investigate the data description and identify which values that are not present in the train-set. The following are features that have value types not present in the training set:

- Exterior1st: Precast, Other
- Exterior2st: Precast
- MSSubClass: 150
- MsZonig: A, I, RP
- Utilities: NoSewr, ELO
- Condition2: RRNe
- MasVnrType: CBlock
- ExterQual: Po
- BsmtCond: Ex
- KitchenQual: Po
- Functional: Sal
- PoolQC: TA
- MiscFeature: Elev
- SaleType: VWD

As we already have a strong understanding regarding the categorical features of our datasets, we would lastly want to analyze the correlation between numerical features. As the plot itself is quite hectic and would reduce readability, we decided not to attach the plot here. However, it is obvious that there are multiple features that are highly correlated, which could indicate multicollinearity. To address this, we considered applying Principal Component Analysis (PCA) later in the modeling process to mitigate multicollinearity while preserving the most important information in the dataset.

3.1.3 ARM Between Categorical Features

Using a support and confidence of 80%, we looked for the ARM association rules between all 44 categorical features and plotted the top 50 rules. Below are the results.

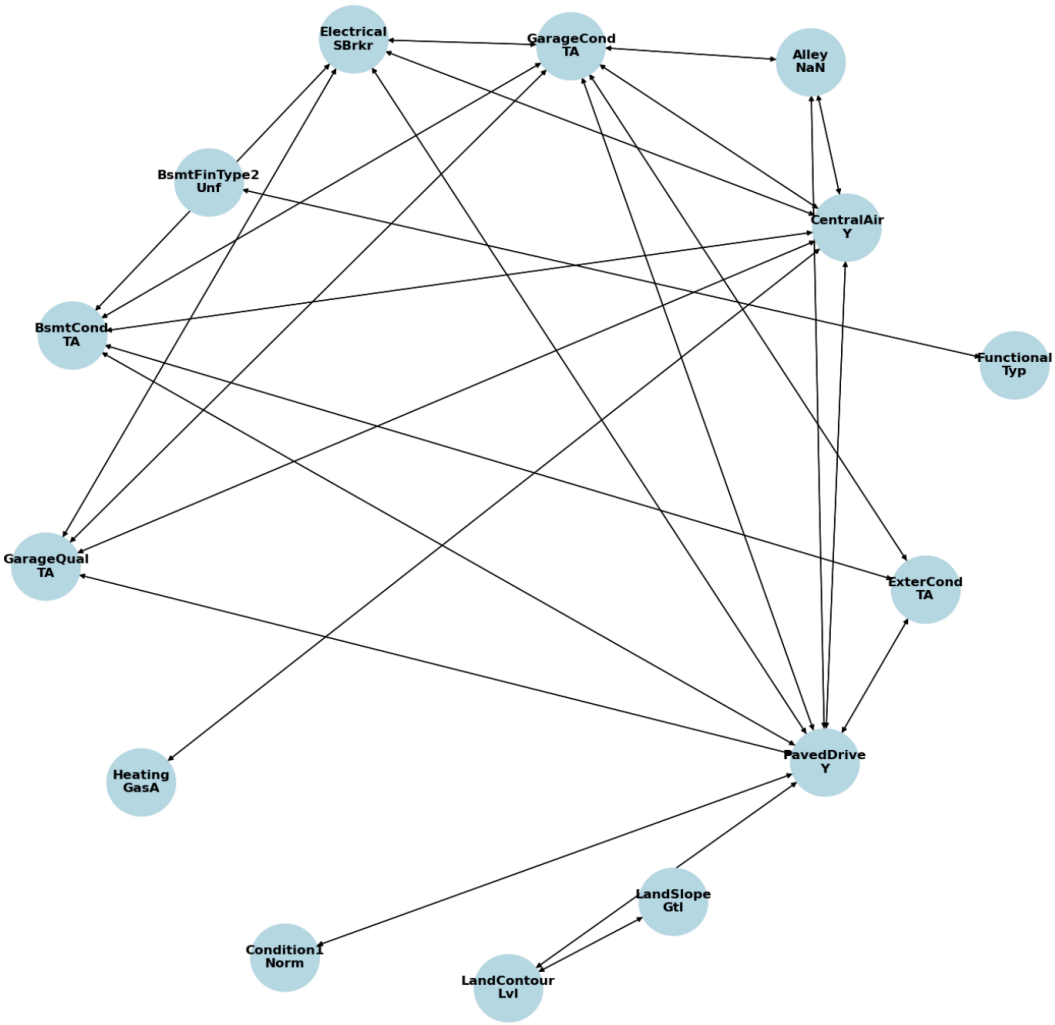


Fig. 8: ARM-analysis graph

3.1.4 Outlier Analysis

Next in our data preprocessing step, we want to get rid of outliers as they will reduce the accuracy of our model. We want to look for outliers for features with >50 unique values, since that is where it is more likely for outliers to exist. Once again, to improve readability, we do not attach the plots here. But after observing the plots from our code, we identify several numerical features that have outliers. The following features are:

- LotFrontage
- LotArea
- GrLivArea
- BsmtFinSF1
- TotalBsmtSF
- 1stFlrSF
- MasVnrArea

We conduct a deep dive into each feature and plot each datapoint to identify if certain points should be removed. Let us take a deep dive into LotArea as one example.

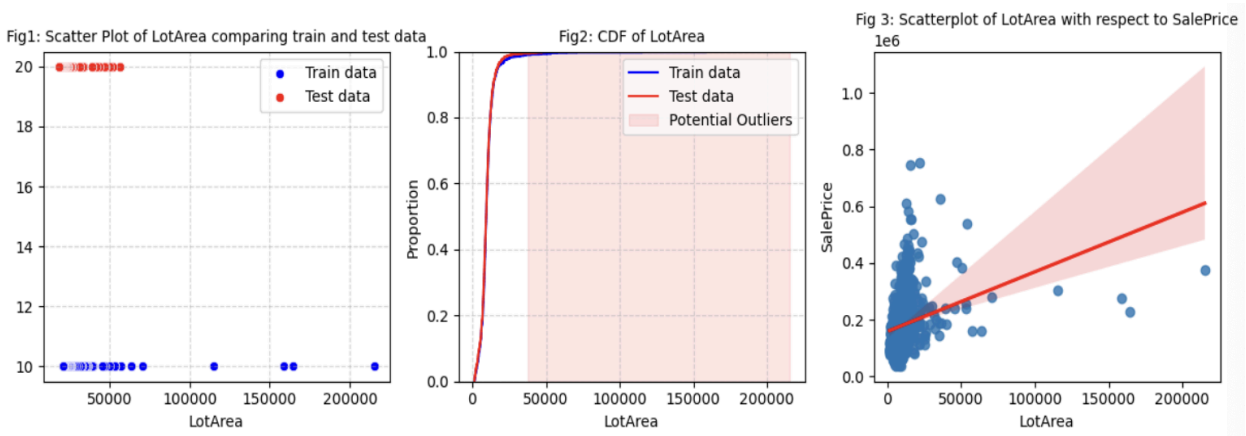


Fig. 9. Plots for outlier analysis

From the figures, we observe that there are 4 values in LotArea exceeding 100,000. Notably, such high values are absent in the test set, further indicating that they are outliers and should be removed. Therefore we drop values of LotArea > 100,000. We conduct a similar approach for all the remaining columns.

3.2 Feature Engineering and Selection

As some features are redundant/similar, we decided to create several new features that we think are more efficient. For example, instead of having both ‘OverallQual’ and ‘OverallCond’, we created a new feature called ‘overall-cond-qual’. Another example would be ‘TotalBaths’, which are the total number of bathrooms in the house (‘BsmtBats’ + ‘AboveBaths’).

Another interesting feature we added were the inflation rates between 1850 and 2024. We thought inflation data could provide a better estimate into housing prices as high inflation rates would lead to more expensive housing. We also calculated the cumulative inflation rate between YrBuild and YrSold.

Since we have a large dataset with over 300 columns (after one-hot encoding categorical features), we decided that a LASSO regression model would be useful, in hopes of reducing our dataset dimensionality while still preserving relevant information. As noted above, we would also like to reduce multicollinearity by conducting Principal Component Analysis (PCA) and compare it with our model if no PCA was conducted.

3.2.1 Principal Component Analysis (PCA)

We perform PCA to help us visualize our high-dimensional dataset by projecting it into a lower-dimensional space. By doing this, we could simplify our data interpretation and exploration because PCA can remove noise or redundant information from data by focusing on the principal components that capture the underlying patterns. In other words, it could also help us to tackle the multicollinearity problem in our dataset. After performing PCA, we reduced our initial features which are more than 300 by half that is only 156 components. Moreover, below are the performance, best parameters of our PCA model and best CV MSE.

```
PCA Model Performance:
Validation MSE: 443415785.25110847
Validation RMSE: 21057.440140033843
Validation MAPE: 0.08827284684956639
Validation R^2 Score: 0.913107524406162
```

Fig. 10. Performance of PCA model

3.2.2 LASSO

The purpose of conducting a LASSO regression is to penalize and reduce data dimensionality by filtering for the most important features. After conducting LASSO, we received 132 non-zero important features that we will use for our model. This was a great reduction from over 300 features. Moreover, below are the performance, best parameters of our LASSO model and best CV MSE.

```
Lasso Model Performance:
Validation MSE: 415978000.0454675
Validation RMSE: 20395.538728983538
Validation MAPE: 0.08555617430062416
Validation R^2 Score: 0.918484277243186
```

Fig. 11. Performance of LASSO model

3.2.3 Random-Forest

Instead of only relying on 1 model (LASSO), we decided to experiment and try use a Random-Forest algorithm to filter out for the most important features as well. Instead of 132 features, we get 167 features from our Random-Forest algorithm. Below is the plot for the top 20 important features using the RF algorithm.

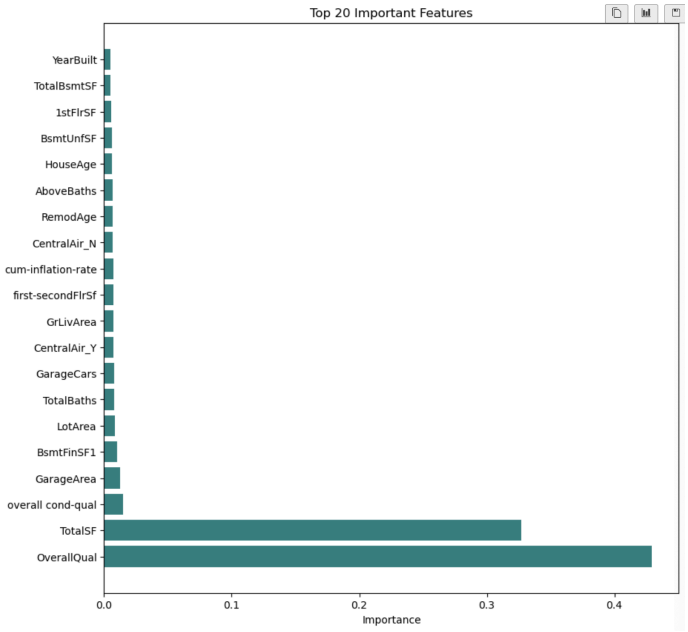


Fig. 12. Most important features using RF

3.3 Model Selection and Results

3.3.1 Model Selection

In order to develop a robust predictive model, a comprehensive model selection framework was implemented. We combined the above preprocessing methods together with a diverse set of machine learning and regression models to explore which would be the best-performer. As a recap, our preprocessing methods include:

- Imputing missing values (for both numerical and categorical features)
- Eliminating outliers
- Creating 4 types of data:
    - Raw Data (without applying PCA, or doing feature selection)
    - PCA Applied Data
    - PCA Applied + LASSO based feature selection
    - Random Forest based feature selection

For the modeling phase, we experimented with a wide array of algorithms, ranging from basic models such as linear regression to higher level, more advanced methods such as Gradient Boosting, XGBoost, CatBoostRegressor, Light GBM Regressor, etc. In order to optimize their hyperparameters, we assign the models with specifically tailored hyperparameter grids and fine-tuned using Bayesian optimization with a 5-fold cross-validation.

In order to properly select the best method for our model, we deploy various performance metrics identical to the metrics used for LASSO and RF above. These metrics provide a holistic perspective for each model's predictive accuracy.

After we trained all 24 models (6 algorithm x 4 types of data), we selected the best model that has the minimum mean absolute percentage error (MAPE) on the validation set. We also give ranks to all models according to their performance on the validation set.

3.3.2 Results

Finally, after deploying all the steps above, we arrive at our best performing model with its respective metrics. As it can be seen in the figure below, the best model is the LGBMRegressor Algorithm trained with the data that is not applied PCA or filtering by any selection method. Its MAPE in the validation set is 0.082 which is the best one among all models.

```
# sort the models according to the validation MAPE
PERFORMANCE_DF = PERFORMANCE_DF.sort_values(by='Val_MAPE', ascending=True)
PERFORMANCE_DF
✓ [59] 37ms
```

24 rows | 24 rows × 12 columns | Static Output

	Regressor	Data	Selected Parameters	CV_Score	Train_MSE	Train_RMSE
16	LGBMRegressor	No Selection and No PCA	{'learning_rate': 0.0831156389521627, 'max_dep.…	0.015117	1.473899e+08	1.214042e+04
12	XGBRegressor	No Selection and No PCA	{'learning_rate': 0.0831156389521627, 'max_dep.…	0.015374	3.455429e+07	5.878290e+03
20	CatBoostRegressor	No Selection and No PCA	{'depth': 5, 'iterations': 467, 'l2_leaf_reg':.…	0.013565	1.052080e+08	1.025710e+04
8	GradientBoostingRegressor	No Selection and No PCA	{'learning_rate': 0.0831156389521627, 'max_dep.…	0.015354	2.786782e+07	5.278998e+03
23	CatBoostRegressor	Random Forest Feature Selection Applied	{'depth': 6, 'iterations': 476, 'l2_leaf_reg':.…	0.015902	2.394555e+08	1.547435e+04
1	LinearRegression	PCA Applied	{'fit_intercept': True}	0.013095	2.895416e+08	1.701592e+04
14	XGBRegressor	Lasso Feature Selection and PCA Applied	{'learning_rate': 0.0831156389521627, 'max_dep.…	0.017900	3.879232e+06	1.969576e+03
2	LinearRegression	Lasso Feature Selection and PCA Applied	{'fit_intercept': True}	0.013173	2.925588e+08	1.710435e+04
22	CatBoostRegressor	Lasso Feature Selection and PCA Applied	{'depth': 5, 'iterations': 410, 'l2_leaf_reg':.…	0.016016	1.718736e+07	4.145764e+03
13	XGBRegressor	PCA Applied	{'learning_rate': 0.0831156389521627, 'max_dep.…	0.017485	3.090783e+06	1.758062e+03
18	LGBMRegressor	Lasso Feature Selection and PCA Applied	{'learning_rate': 0.0831156389521627, 'max_dep.…	0.017398	4.786422e+07	6.918397e+03

Fig. 13 Model Performances

The following graphs illustrate the prediction and the error for training and validation data sets. As we can see, the errors of the selected model are approximately normally distributed, so we can be sure that we are not underestimating or overestimating.

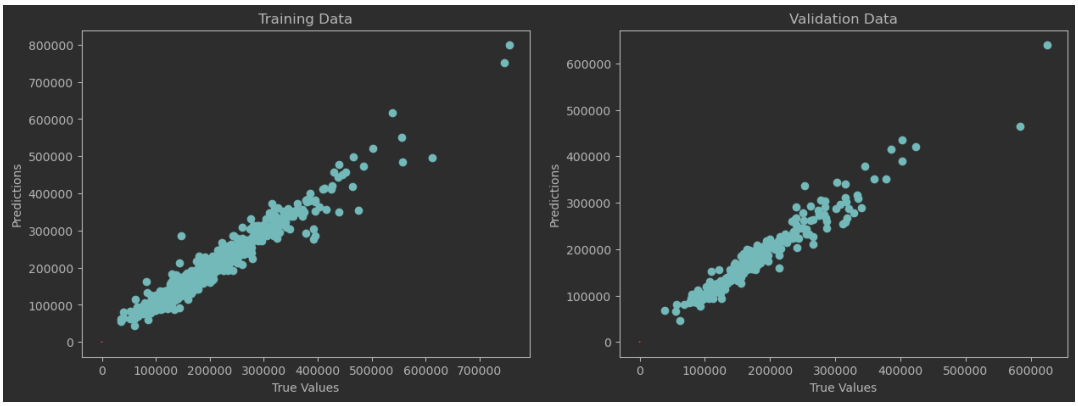


Fig. 14. Plots for Best Model's Predictions

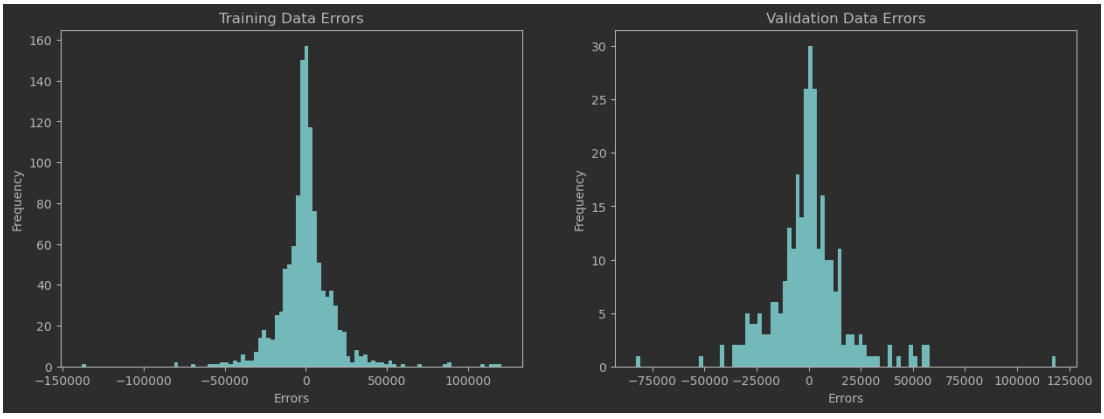


Fig. 15: Error for training and validation sets

For the prediction of the test set, we first trained this selected model and all other models with the whole train data, then we used these retrained models to predict test data. When we submit our best models, as we expected the best model we chose has the best score among all of them. However, in order to improve our score, we decided to combine all of the predictions from all models by taking the mean of them. This approach was better than some single model predictions but it did not improve the best solution we got. Finally, instead of uniformly combining all the predictions we decided to combine them by assigning different weights according to their ranks. As a result of this weighted combination of predictions of all models we achieved our best score as it can be seen below

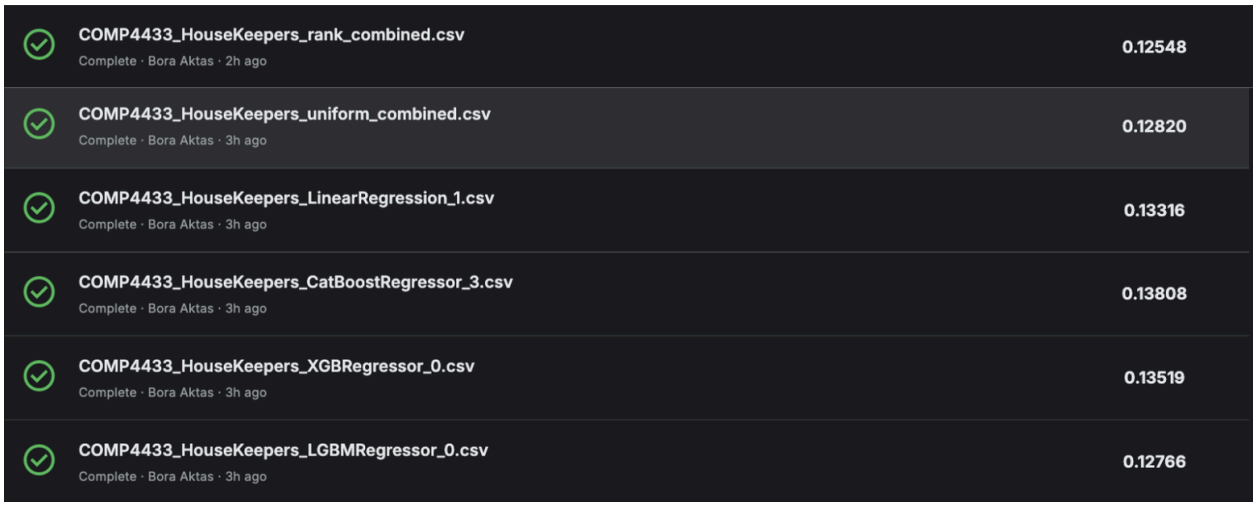


Fig. 16: Final score

Submitting our final model to Kaggle, we received a score of 0.12548, which is ranked 796 out of almost 6000 submissions.

4 Conclusion

The most impactful future engineering effort was creating our own features. This allowed us to integrate domain knowledge and construct useful attributes like overall-cond-qual and TotalBaths. Such features encapsulated information that individual variables could now fully represent, and contributed to the model's predictiveness.

In contrast, e.g. LASSO was not as beneficial as initially anticipated. While it did reduce dimensionality by selecting 132 non-zero features from over 300, the selected features did not provide substantial performance gains compared to the models trained with unfiltered datasets.

The most significant improvement occurred when we combined all model predictions using a weighted approach based on their validation ranks, instead of uniformly combining them.

The uniformly combined prediction models and the weighted combined prediction models saw an increase from 0.1282 to 0.12548.

The reason we decided to assign weights to different models stem from the fact that our uniformly combined models performed worse than just the LGBM Regression model alone, therefore logically we should assign a higher weight to the LGBM Regression model when combining everything together.