University of Warsaw

Faculty of Economic Sciences

OLS vs ML: which one is better?

Social Networks in Economic Geography and Spatial ML

Giacomo Fantato · **K-16958**

g.fantato@student.uw.edu.pl

Contents

Introduction

General description

This is a task for passing the 'Spatial Machine Learning' class. Below you have data and model framework – the goal is to check if predictions from spatial ML are better than predictions from the OLS model. Please submit R codes, outputs and nice visualizations that solve this challenge. Please consider typical (and/or fancy) machine learning models and their diagnostics (variable importance, partial plots, quality of predictions etc.)

We use a classical dataset on housing valuation in Toledo (U.S.). This raw dataset is available in package varycoef. In the **link** you get a pre-processed dataset – a few spatial variables were added. The goal of the exercise is to run models on data up to 1997 and test them on the year 1998. The micro-geography model means that it uses information from the local neighborhood (here, for k=50 nearest neighbors).

OLS and variables description

Specification of the model (maximum, you can eliminate some variables if needed): The logarithm of the price of real estate depends on its characteristics, location, and characteristics in the neighborhoods.

$$\begin{aligned} \text{eq} = \log(\text{price}) \sim & \log(\text{age}) + \log(\text{lotsize}) \\ & + \log(\text{livearea}) + \text{Story2more} + \text{wall} \\ & + \text{beds} + \text{baths} + \text{dGarage} + \text{dToledo} \\ & + \text{MeanPrice} + \text{XWX2M} + \text{XWX3M} + \text{XWX4M} \\ & + \text{XWX6M} + \text{XWX7M} + \text{XWX8M} + \text{SAFE} \end{aligned} \tag{1}$$

Here a full table with the description of the working environment variables

name	**explanation**	**→ function of variable**
log_price	logarithm of price of real estate (house)	→ y variable
log_age	logarithm of the age of real estate	→ x - internal features
log_lotsize	logarithm of the size of land (parcel)	
log_livearea	logarithm of the surface (floor) of real estate	
Story2more	dummy to express if there are more than two floors	
wall	material of construction (partbrk, metlvnyl, stucdrvt, wood, brick, stone, ccbtile)	
beds	number of bedrooms	
baths	number of bathrooms	
dGarage	dummy if there is a garage	
dToledo	distance to the city centre of Toledo	→ x - spatial variables
MeanPrice	average of log_price in the neighbourhoods (k=50 nearest neighbours, knn=50)	
XWX2M	difference between X and average X of knn=50 for log_age	
XWX3M	difference between X and average X of knn=50 for log_lot_size	
XWX4M	difference between X and average X of knn=50 for log_livearea	
XWX6M	difference between X and average X of knn=50 for log_bathrooms	
XWX7M	difference between X and average X of knn=50 for Story2more	
XWX8M	difference between X and average X of knn=50 for dToledo	
SAFE	factor to express in which mini-district the house is located	

OLS regression

Summary over OLS regression

Performing a summary over the OLS regression we could see some insights:

- The model has a **Multiple R-squared of 0.8259** and an **Adjusted R-squared of 0.8247**, indicating that approximately **82.5% of the variance** in the dependent variable (*log_price*) is explained by the predictors.
- The overall significance of the model is confirmed by a **high F-statistic (686.2, p-value $<$ 2.2e-16)**, suggesting that at least one predictor has a significant effect on the response variable.
- The **residual standard error (0.3173)** represents the typical deviation of observed values from the regression line, indicating that the model provides reasonably precise predictions.
- The **distribution of residuals** appears relatively symmetric, with a median close to zero. However, the range from **-2.337 to 1.829** suggests the presence of some extreme residuals, warranting further investigation into potential outliers or model specification issues.

Quality of regression: MAE & RMSE

MAE (Mean Absolute Error) is a metric used to measure the average magnitude of errors in a set of predictions, without considering their direction (i.e., it treats overestimates and underestimates equally). It is the average of the absolute differences between the

observed actual outcomes and the predicted values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2}$$

The MAE gives us an idea of how far off our predictions are from the true values on average. A lower MAE *indicates a better model performance*. Since it uses absolute differences, it does not penalize large errors as much as RMSE.
In our OLS **the value of the MAE** is: **0.2151571**; the predictions are off by approximately 0.215 units. That's a good metrix, considering that the value of population is around 20k units.

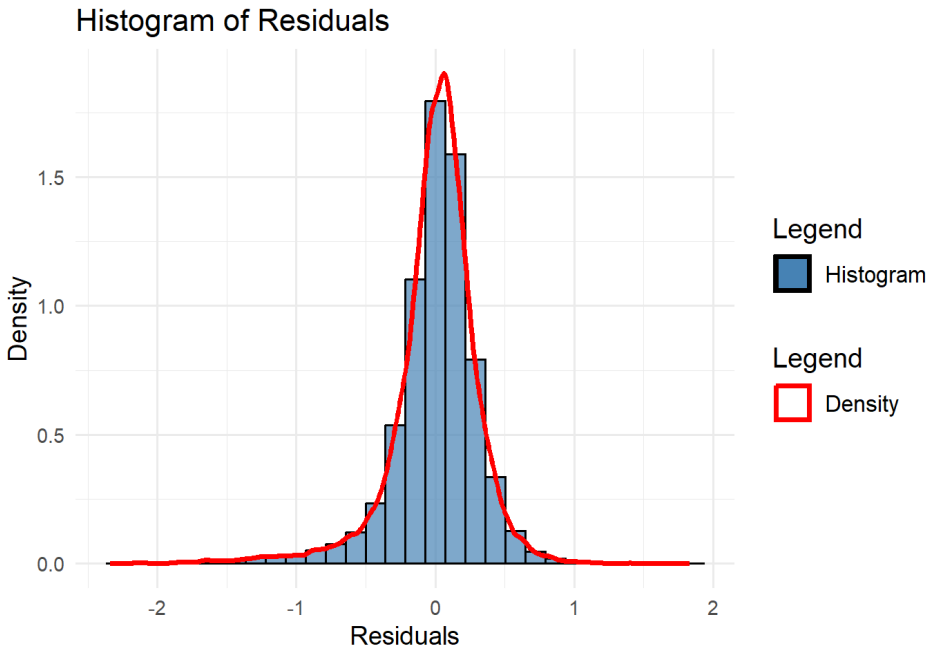
RMSE (Root Mean Squared Error) is a metric that measures the square root of the average of the squared differences between the predicted values and the actual values. It is sensitive to large errors and gives more weight to larger discrepancies between observed and predicted values, making it more sensitive to outliers than MAE.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

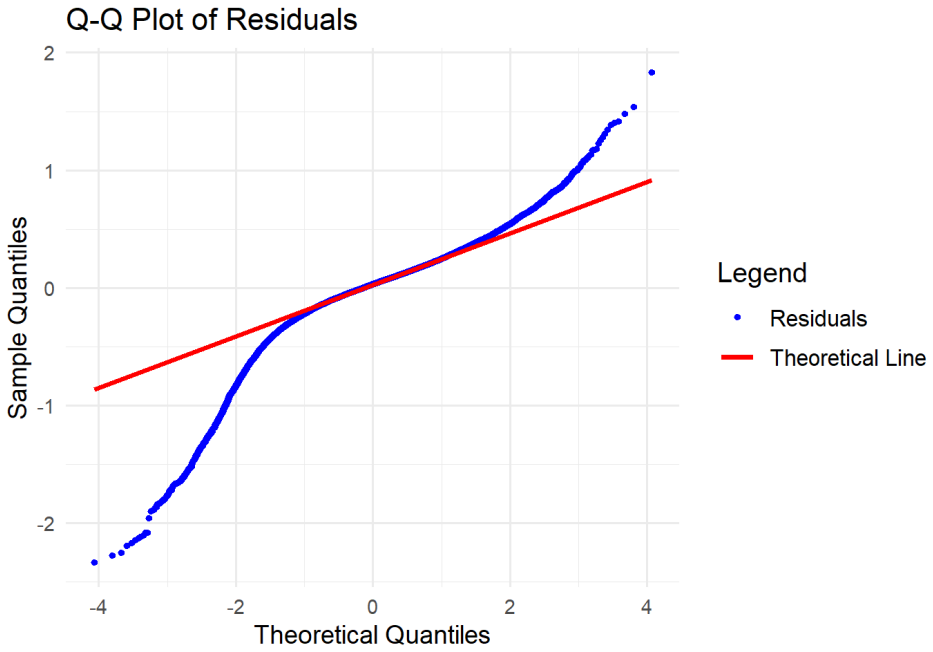
The RMSE penalizes larger errors more than MAE due to the squaring of differences. A lower RMSE indicates better model accuracy, but because RMSE is sensitive to outliers, it can sometimes give a higher value even if most of the predictions are accurate.
In our OLS the **value of the RMSE** is: **0.3002817**; means that, on average, the predictions are off by 0.30 units.
Since RMSE is sensitive to larger errors, it suggests that while most of ours predictions are close to the actual values (based on the MAE), there might be a few larger errors that are influencing the RMSE more.

Residuals analysis

First we need to understand the distribution of residuals in our linear regression model. To do this, we use a histogram plot to check the distribution of the residuals (whether it follows a normal distribution or not).



From this histogram we can see that the tails are not perfectly symmetrical, although it would appear that the residuals follow a normal distribution.
Finally, to verify graphically the I.I.D. property we use a Q-Q plot; in this case to have a normal distribution, the residuals should be close to the bisector.

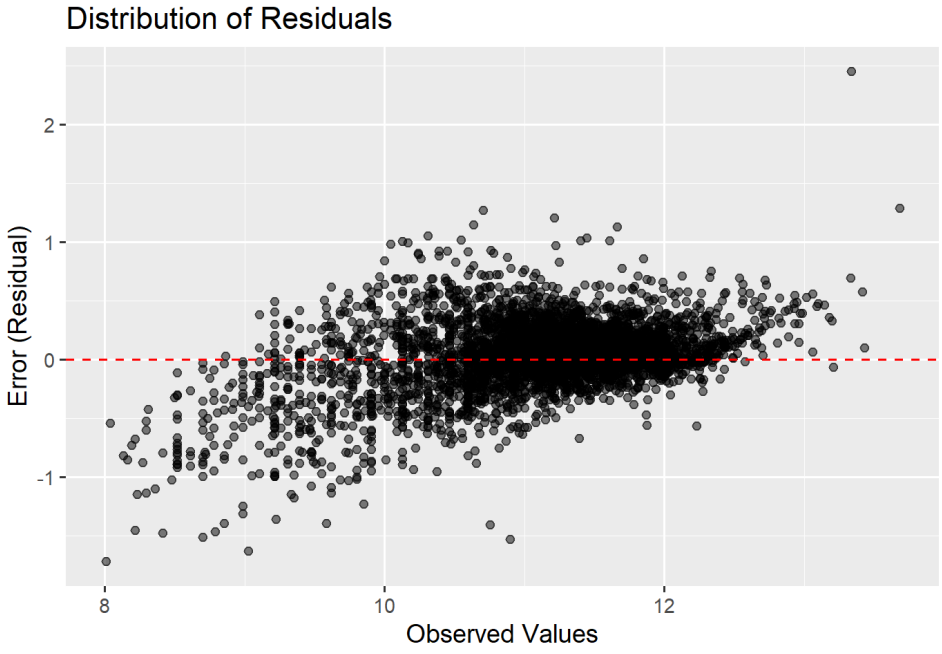


With this graph we can see how the tails actually do not have normal behavior, concluding that they still fall within the typical errors in these data.

After, we plot the distribution of residuals in relation to the observed values. The residuals (the errors between observed and predicted values) are plotted against the actual observed values. The horizontal red dashed line at y = 0 serves as a baseline, showing where residuals are exactly zero (meaning perfect predictions).

```
ggplot(data = data.pred.OLS, aes(x = obs, y = obs - pred)) +
  geom_point(alpha = 0.5) +
  geom_hline(yintercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Distribution of Residuals", x = "Observed Values",
  y = "Error (Residual)") + theme_minimal()
```

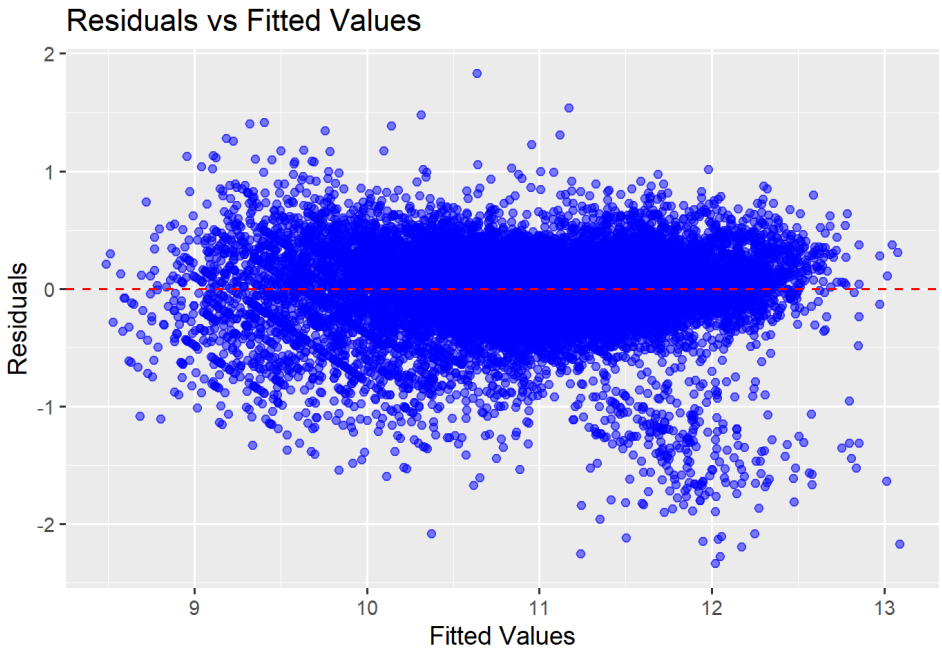
From the generated plot we could see a clustered behavior with a cone tending toward the center-right. That behavior could suggest the presence of heteroskedasticity in the model.



Plotting the graph of residuals in comparison with fitted values, we are going to check the presence of **heteroskedasticity**.

```r
ggplot(data.frame(Fitted=model.lm$fitted.values,
    Residuals=residuals_OLS), aes(x=Fitted, y=Residuals)) +
    geom_point(alpha=0.5, color="blue") +
    geom_hline(yintercept=0, color="red", linetype="dashed") +
    labs(title="Residuals vs Fitted Values", x="Fitted Values",
    y="Residuals") + theme_minimal()
```

The presence of heteroskedasticity is confirmed by the graph below. We can be seen that the variance of the residuals, which is not constant over the range of the adjusted values, leads to any nonlinearity in the relationship between the predictors and the target variable.



To further investigate and confirm not only visually, but also quantitatively, the presence of heteroschedasticity, we need to perform some statistical tests.

To be sure of the data just displayed, we use some statistics. The initial idea was to implement a Saphiro - Wilkinson test as a check for normality.
However, since the observed sample is very large (about 20k population elements), the

test is not executable since it allows a maximum of 5000 observations.
In this case we can use the **Kolmogorov-Smirnov test**, which allows us to allow us to compare the observed cumulative distribution function for a variable with the specified theoretical distribution, which can be *normal*, uniform or Poisson's distribution [1].

```
        Asymptotic one-sample Kolmogorov-Smirnov test

data:  residuals_OLS
D = 0.098074, p-value < 2.2e-16
alternative hypothesis: two-sided
```

The p-value is extremely low, much less than 0.05. This means that we can reject the null hypothesis of normality (the residuals do not follow a normal distribution).

The **Anderson-Darling** (AD) test[2] is a statistical test of whether a data set follows a specific distribution, typically the normal distribution. It is an extension of the Kolmogorov-Smirnov test, but *gives more weight to the tails of the distribution.*

```
        Anderson-Darling normality test

data:  residuals_OLS
A = 416.13, p-value < 2.2e-16
```

The null hypothesis of normality is rejected because of the p-value is extremely low, much less than 0.05.
The residuals do not follow a normal distribution, and this confirms the result of the Kolmogorov-Smirnov test.

Finally, we use 2 statistical tests:

1. **Durbin-Watson test** (to test **autocorrelation**): the Durbin-Watson (DW) test checks for autocorrelation of the residuals in a linear regression. A value of DW =

[1] https://www.ibm.com/docs/it/spss-statistics/saas?topic=tests-one-sample-kolmogorov-smirnov-test

[2] https://www.itl.nist.gov/div898/handbook/eda/section3/eda35e.htm

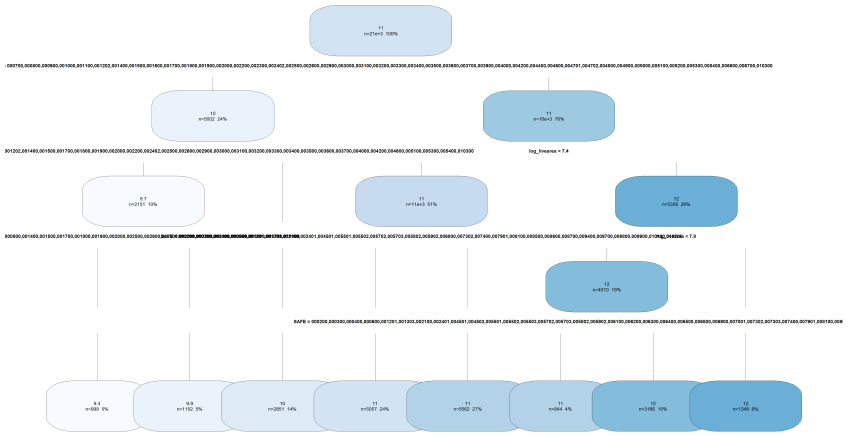
1.9423, indicates a value very close to 2, indicating low or no significant autocorrelation.
The p-value = 7.217e-06 is very low suggests that the autocorrelation might be statistically significant, but the value of DW indicates that it is not strong.
There does not appear to be strong positive autocorrelation in the residuals; however, the low p-value may indicate a slight dependence between the errors.

2. **Breusch-Pagan test** (to test **heteroskedasticity**): the Breusch-Pagan test checks for heteroschedasticity, that is, whether the variance of the residuals is constant (homoschedasticity) or varies systematically (heteroschedasticity).
In uest case we have: BP = 3948.5, a very high value of the test statistic and 144 degrees of freedom of the test. The p-value < 2.2e-16, is extremely low, indicating that the null hypothesis (constant variance of the residuals) must be rejected.
In this case we see the presence of heteroschedasticity in the residuals i.e., the variance of the errors is not constant. As a consequence, OLS estimators are no longer efficient (although they remain unbiased). Standard errors in the estimates may be biased, making significance tests unreliable.

Machine Learning

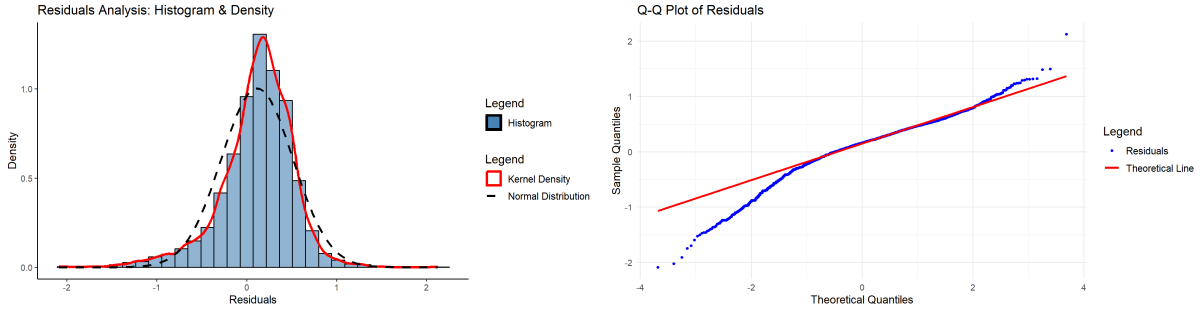
Decision Tree

First of all, we started with the most basic model: **Decision Tree**. I decided to use as my first model the simplest and most basic, a tree structure in which each node represents a condition based on a variable, and each branch leads to a result based on that condition.
As a pro, it does not require much data preparation (can handle categorical and numeric data), can model nonlinear relationships between variables, and works well with small and medium datasets.
The biggest con is overfitting: if the tree is too deep, it may overfit the training data and generalize poorly. Sensitive to noisy data (outliers or outliers can distort decisions). Finally, it is less accurate than more complex models (such as Random Forest or XGBoost) but lighter and allows us to see if a basic ML model can already suffice or not.

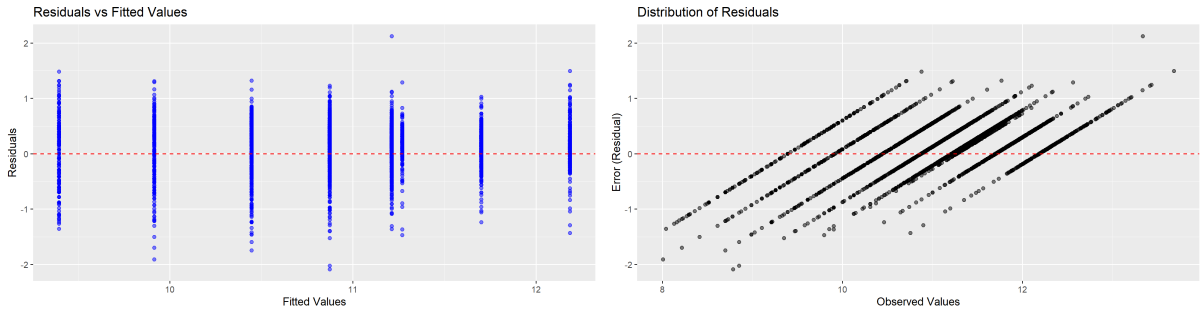


The Decision Tree

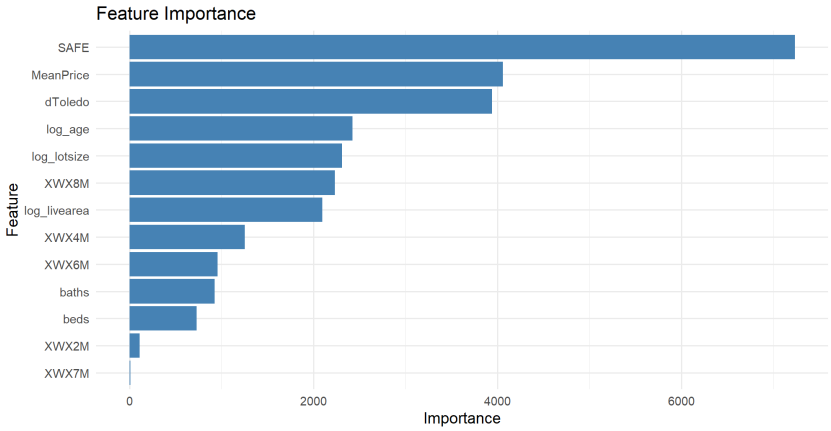
First, we try to use the same comparison metrics as OLS, namely MAE and RMSE. In this case, **the results are worse than the OLS** (respectively: **MAE: 0.3225019, RMSE: 0.4145775**)
The analysis of residuals does not make too much sense; the data are split binary, so the behavior of the residuals will obviously be affected by this choice. Moreover, it is possible to perform this type of analysis because the tree construction method is based on ANOVA.
Compared with OLS we can see from the Q-Q plot an improvement in the trend of residuals.



From the plots of the distribution of residuals we can see heteroskedasticity given by the classic "*columnar*" pattern.



To better understand which features are key or most dominant in the model, we plot feature importance; this is a technique used in machine learning to determine which variables (or features) are most significant in predicting model output. In other words, it indicates what impact each feature has on the model output.
In this case SAFE is the one predominant over all others, subsequently meanPrice and distance to Toledo are critical.

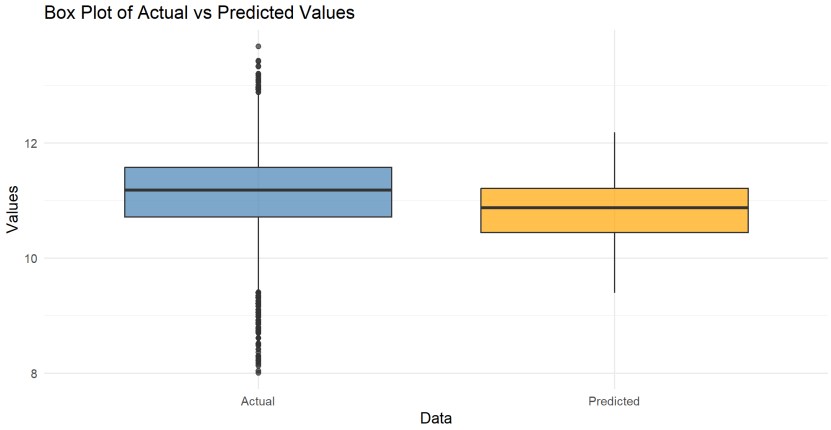


As for statistical tests, we can perform:

- **Breusch-Pagan**: BP = 17.61, df = 1, p-value = 2.712e-05
  The Breusch-Pagan (BP) test result shows a p-value of 2.712e-05, which is less than 0.05. This indicates heteroskedasticity in the model, meaning the variance of the residuals is not constant. Therefore, the assumption of homoskedasticity is violated.

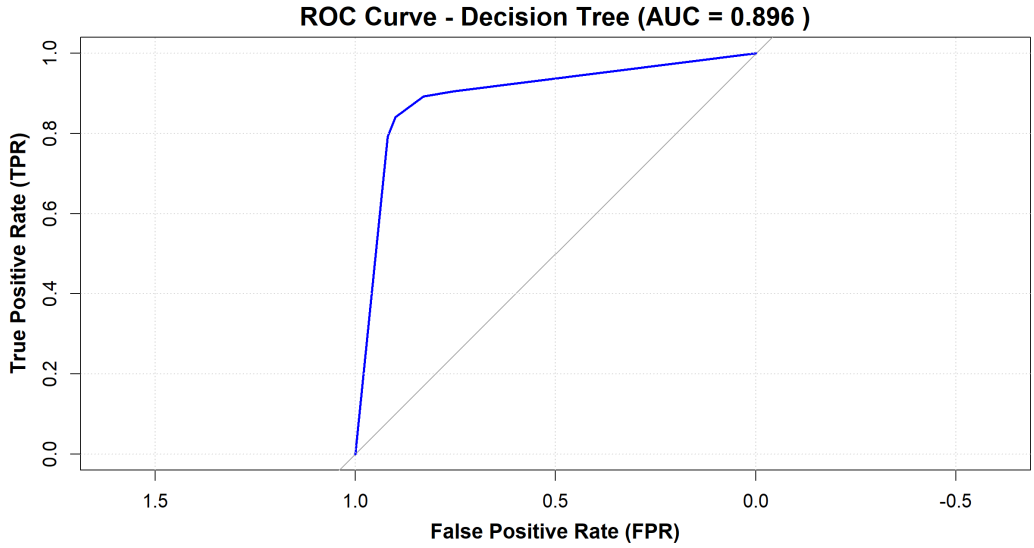
- **Durbin-Watson**: DW = 1.928, p-value = 0.008502
  The Durbin-Watson (DW) test result shows a p-value of 0.008502, which is less than 0.05. This suggests the presence of autocorrelation in the residuals. Specifically, there is evidence of positive autocorrelation.

The boxplot compares the distribution of observed (Actual) and predicted (Predicted) values from the model. It shows the median, quartiles and outliers for each group, helping to visualize the differences between the actual and predicted data.



As a final comparison parameter, we have the ROC calculation, which is **AUC: 0.8955686**. In this case, an AUC of 0.895 suggests that the model has an excellent performance, with an ability to distinguish between classes of almost 90%.

In addition, the ROC curve helps visualize how the model performs in terms of the balance between accuracy and sensitivity, with the area under the curve (AUC) indicating the overall quality of the model.



Running the confusion matrix we have the following output:

```
Confusion Matrix and Statistics
          Reference
Prediction    0    1
         0 1973  348
         1  219 1838
               Accuracy : 0.8705
                 95% CI : (0.8602, 0.8803)
    No Information Rate : 0.5007
    P-Value [Acc > NIR] : < 2.2e-16
                  Kappa : 0.741
 Mcnemar's Test P-Value : 7.637e-08
```

```
            Sensitivity : 0.9001
            Specificity : 0.8408
         Pos Pred Value : 0.8501
         Neg Pred Value : 0.8935
             Prevalence : 0.5007
         Detection Rate : 0.4507
   Detection Prevalence : 0.5302
      Balanced Accuracy : 0.8704
       'Positive' Class : 0
```

Based on the confusion matrix, the model exhibits the following performance metrics:

- **Accuracy**: 87.05% (95% CI: 86.02% to 88.03%), this indicates that the model correctly classifies 87.05% of all observations.
- **Kappa**: 0.741, the Kappa statistic indicates a substantial agreement between predicted and actual values.
- **Sensitivity** (recall for class 0): 90.01%; the model correctly identifies 90.01% of the actual '0' class instances.
- **Specificity**: 84.08%; the model correctly identifies 84.08% of the actual '1' class instances.
- **Positive Predictive Value** (precision for class 0): 85.01%; the model's positive predictions for class '0' are 85.01% correct.
- **Negative Predictive Value**: 89.35%; the model's negative predictions for class '1' are 89.35% correct.

The McNemar's Test p-value is extremely small (7.637e-08), suggesting a significant difference between the misclassifications of the two classes. Given the p-value and the accuracy metrics, the model performs significantly better than a random classifier, and the model's predictions are reliable.

eXtreme Gradient Boosting - XGBoost

Next, we moved on to a more advanced model: XGBoost. I chose to use XGBoost as my second approach due to its advantages in handling complex machine learning problems. This algorithm, based on the boosting technique, combines multiple decision trees to enhance predictive performance. Unlike a single decision tree, XGBoost applies sequential learning, where each tree corrects the errors of the previous ones, leading to a more refined and accurate model.
One of its main strengths is its ability to mitigate overfitting through built-in regularization techniques, making it particularly suitable for datasets with a high number of features. Additionally, its computational efficiency and parallelization capabilities make it ideal for handling large datasets while maintaining an optimal balance between accuracy and execution time. Compared to a simple Decision Tree, XGBoost provides a more robust and reliable approach, making it a powerful choice for improving model performance.

After several attempts and parameter tuning, we arrived at this training configuration to balance model performance and generalization, preventing overfitting while maintaining accuracy.

```
params <- list(
  objective = "reg:squarederror",
  eval_metric = "rmse",
  eta = 0.05,
  max_depth = 4,
  subsample = 0.6,
  colsample_bytree = 0.6,
  lambda = 1.0,
  alpha = 0.5
)
```

Following we have the explanation of very choices:

- **Objective**: "reg:squarederror": the model is performing regression using squared error loss, minimizing the difference between predicted and actual values.

- **Evaluation Metric**: "rmse" (Root Mean Squared Error): measures how far predictions deviate from actual values. Lower RMSE indicates better accuracy.
- **Learning Rate** (eta): 0.0: controls the step size of updates during training. A lower value ensures a more gradual learning process, improving stability and preventing overfitting.
- **Max Depth**: 4; limits the depth of each decision tree to control complexity. Prevents overfitting by ensuring the model does not memorize training data too precisely.
- **Subsample**: 0.6; uses only 60% of the training data for each tree. Introduces randomness, improving generalization to new data.
- **Colsample_bytree**: 0.6; uses only 60% of the available features for each tree. Reduces feature correlation, making the model more robust.
- **Lambda**: 1.0 (L2 Regularization): helps prevent overfitting by penalizing large coefficients, encouraging smaller weights.
- **Alpha**: 0.5 (L1 Regularization): encourages sparsity in the model by reducing the impact of less important features.

This combination of parameters ensures a good balance between learning capacity and generalization, preventing the model from being too complex while maintaining strong predictive power. Through an *iterative tuning process*, these specific values proved to deliver the best results in terms of accuracy and model stability.

First, we run a k-fold cross-validation with 5 folds to evaluate the model's performance. The results are as follows:

```
Average RMSE - Training Set: 0.2309422
Average RMSE - Test Set: 0.2690682
Overfitting Ratio (Test RMSE / Train RMSE): 1.165089
```

These results indicate that the model maintains a good balance between training and test performance. The test RMSE is only slightly higher than the training RMSE, with an

overfitting ratio close to 1. This suggests that the model does not suffer from significant overfitting and generalizes well to unseen data. Here are the evaluation metrics for the XGBoost model:

- **MAE (XGBoost)**: 0.2053327
- **RMSE (XGBoost)**: 0.2747155

The MAE value indicates that, on average, the model's predictions are off by approximately 0.205, while the RMSE reflects a slightly higher error, with the model's predictions deviating by around 0.275. These metrics suggest that the XGBoost model performs reasonably well, with a balance between accuracy (MAE) and error magnitude (RMSE).

For a statistical test we can perform:

- Asymptotic One-Sample **Kolmogorov-Smirnov test**: D statistic: 0.077503, p-value: $<$ 2.2e-16.
  Given that the p-value is extremely small (much smaller than the typical significance level of 0.05), we can reject the null hypothesis that the residuals follow a uniform distribution. This suggests that the residuals do not follow a normal distribution, which could imply that the model's errors are not perfectly random and may exhibit some structure.

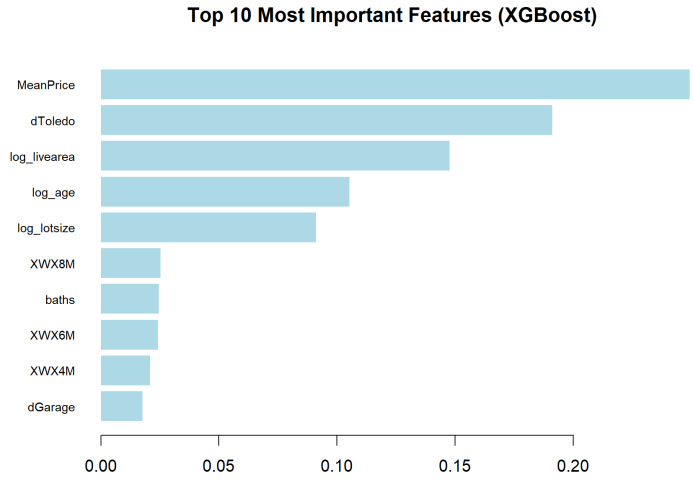
- **Anderson-Darling normality test**: A statistic: 49.835, p-value: $<$ 2.2e-16
  Given that the p-value is extremely small (much smaller than 0.05), we can reject the null hypothesis that the residuals follow a normal distribution. This result reinforces the finding from the Kolmogorov-Smirnov test, indicating that the residuals of the XGBoost model are not normally distributed. This may suggest that the model could benefit from further refinement or that additional modeling techniques may be required to address non-normality in the residuals.

- **Durbin-Watson test for autocorrelation**: DW statistic: 1.9709, p-value: 0.1668.
  Since the DW statistic is close to 2 (1.9709), it suggests there is no significant autocorrelation in the residuals. The p-value of 0.1668 is greater than the typical significance level of 0.05, meaning we fail to reject the null hypothesis that there is no autocorrelation in the residuals. This implies that the residuals are likely independent of each other, indicating that the model does not exhibit any significant autocorrelation.

- **Breusch-Pagan test for heteroscedasticity**: BP statistic: 243.72, Degrees of freedom (df): 1
  Since the p-value is extremely small (much smaller than the typical significance level of 0.05), we reject the null hypothesis that the residuals have constant variance (i.e., homoscedasticity). This indicates that there is significant evidence of heteroscedasticity in the residuals, meaning the variance of the residuals is not constant across the range of predicted values. This suggests that the model might benefit from additional transformations or adjustments to account for heteroscedasticity, such as using weighted least squares or transforming the target variable.

We can analyze the feature importance to understand which features are most significant in the XGBoost model. The table of results for feature importance is as follows:

	Feature	Gain	Cover	Frequency	Importance
	<char>	<num>	<num>	<num>	<num>
1:	MeanPrice	0.26581486	0.079692125	0.071323960	0.26581486
2:	dToledo	0.17622663	0.037179260	0.064440539	0.17622663
3:	log_livearea	0.14824563	0.051488197	0.077035735	0.14824563
4:	log_age	0.09702748	0.078165709	0.084504979	0.09702748
5:	log_lotsize	0.09536195	0.043354856	0.061218512	0.09536195
6:	XWX6M	0.03844572	0.025140522	0.036174575	0.03844572
7:	XWX8M	0.02732161	0.039765356	0.050673697	0.02732161
8:	baths	0.02563349	0.005750314	0.005565319	0.02563349
9:	dGarage	0.02041795	0.011952385	0.010837727	0.02041795
10:	latitude	0.01343608	0.033883748	0.073520797	0.01343608



In conclusion, MeanPrice is the most important feature, with the highest Gain value of 0.2658, indicating its strong influence on the model's predictions. dToledo and log_livearea also contribute significantly, with Gain values of 0.1762 and 0.1482. In contrast, features like baths, dGarage, and latitude have lower importance, with Gain values under 0.03. The Gain metric highlights each feature's contribution to reducing the model's loss function, while Cover and Frequency show how often these features are used across the trees.

Confusion matrix analysis:

```
Confusion Matrix and Statistics
          Reference
Prediction    0    1
         0 2066  425
         1  110 1776
               Accuracy : 0.8778
                 95% CI : (0.8677, 0.8873)
    No Information Rate : 0.5029
    P-Value [Acc > NIR] : < 2.2e-16
                  Kappa : 0.7557
 Mcnemar's Test P-Value : < 2.2e-16
            Sensitivity : 0.9494
```

```
            Specificity : 0.8069
         Pos Pred Value : 0.8294
         Neg Pred Value : 0.9417
             Prevalence : 0.4971
         Detection Rate : 0.4720
   Detection Prevalence : 0.5691
      Balanced Accuracy : 0.8782
       'Positive' Class : 0
```

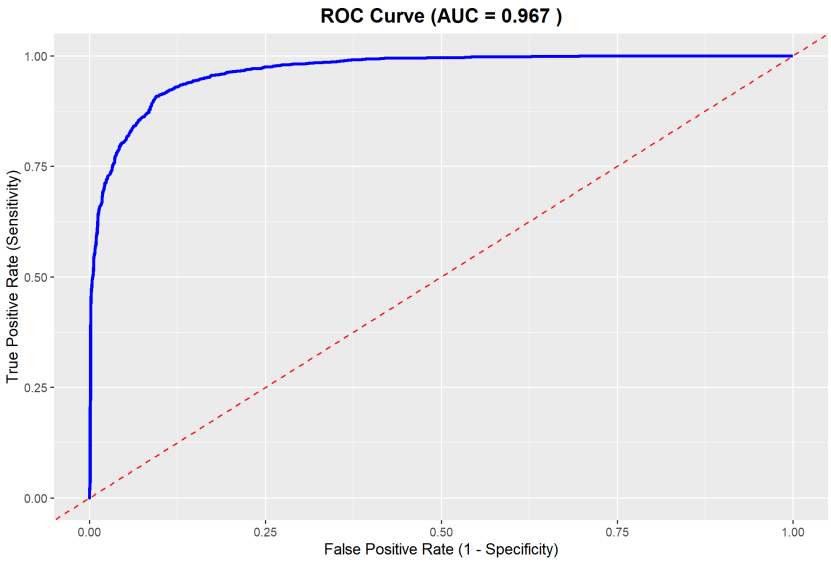
The model performs very well overall, with an accuracy of 87.78%, meaning it correctly predicts the outcome 87.78% of the time. The 95% confidence interval for accuracy is between 86.77% and 88.73%, indicating that this result is consistent across different samples of data. The no information rate, which represents the accuracy you would get by always predicting the most frequent class (class 0), is 50.29%. The p-value for accuracy being better than the baseline is extremely small, confirming that the model is significantly better than simply guessing the majority class.

The Kappa statistic of 0.7557 indicates substantial agreement between the model's predictions and actual outcomes, after accounting for chance. The McNemar's test p-value being extremely small suggests that there are significant differences between false positives and false negatives, which may indicate some biases in the model's predictions.

The sensitivity (recall) of 94.94% shows that the model correctly identifies 94.94% of the actual class 0 cases, while the specificity of 80.69% indicates that it correctly identifies 80.69% of the actual class 1 cases. The positive predictive value (precision) of 82.94% means that when the model predicts class 0, it is correct 82.94% of the time, and the negative predictive value is 94.17%, meaning that when the model predicts class 1, it is correct 94.17% of the time.

The prevalence of class 0 in the dataset is 49.71%, showing that the classes are fairly balanced. The detection rate of 47.20% indicates that the model detects 47.20% of all actual class 0 cases, while the detection prevalence of 56.91% shows that the model predicts class 0 56.91% of the time. Finally, the balanced accuracy is 87.82%, which combines sensitivity and specificity to give a more holistic measure of the model's performance.

Overall, the model performs well in identifying both classes, with strong sensitivity for class 0, though there is still room to refine the model further, especially in minimizing false positives.



Finally, we have plotted the ROC curve with an **AUC of 0.9665304**. This result indicates that the model has excellent classification performance, correctly distinguishing between classes in 96.65% of cases. With an AUC close to 1, the model demonstrates strong predictive power, meaning it effectively balances sensitivity and specificity. Such a high AUC suggests that the model generalizes well and makes reliable predictions, confirming its robustness for the given dataset.

Conclusions

Are predictions from ML models better than those from OLS?

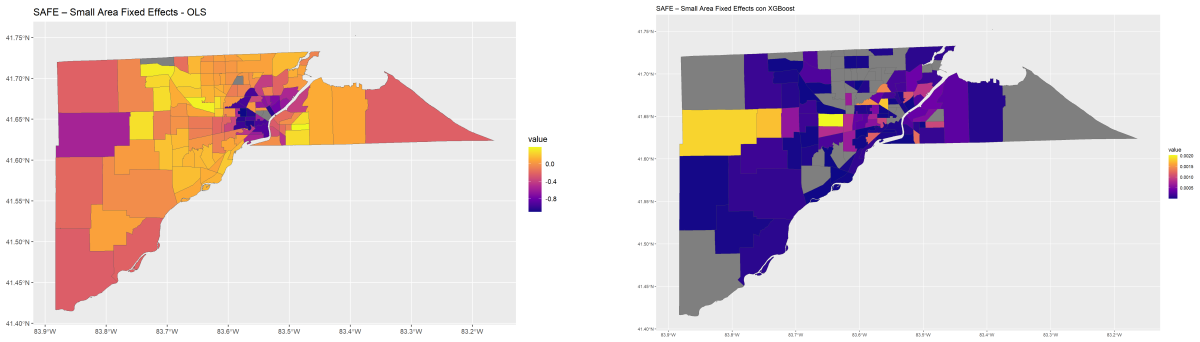
In comparing the predictions from Machine Learning (ML) models and Ordinary Least Squares (OLS), the results indicate that the ML models, particularly XGBoost, provide more accurate and reliable predictions than OLS. The MAE for OLS is 0.2151571, and the RMSE is 0.3002817, while the XGBoost model achieves better performance with an MAE of 0.2053327 and an RMSE of 0.2747155. The ML model, particularly XGBoost, exhibits a much higher AUC (0.9665304) compared to the decision tree (0.8955686), indicating its superior classification ability.

Additionally, the residuals of the OLS model show significant deviations from normality, as seen in the results of the Anderson-Darling test (A = 416.13, p-value < 2.2e-16) and the Kolmogorov-Smirnov test (D = 0.098074, p-value < 2.2e-16), suggesting that the assumptions of linearity and normality may not hold well for OLS. In contrast, the XGBoost model's residuals, while still significant, are less problematic based on the test results.

The XGBoost model also outperforms the decision tree in terms of predictive power, as evidenced by its higher AUC and better generalization performance, with an AUC of 0.9665304 compared to the decision tree's AUC of 0.8955686. The Kappa score for the decision tree (0.741) shows strong agreement between the predicted and actual values, but the XGBoost model's performance across multiple metrics demonstrates that it provides more reliable predictions.

In conclusion, *based on the MAE, RMSE, and AUC, the ML models, particularly XGBoost, deliver superior predictive accuracy compared to OLS*, which shows limitations in terms of normality and residual analysis. Therefore, predictions from ML models, espe-

cially XGBoost, are generally better than those from OLS in this case. When evaluat-



Comparison between OLS and XGBoost outputs.

ing the trade-offs between predictive performance and computational complexity among Ordinary Least Squares (OLS), Decision Trees, and XGBoost models, it's essential to consider both the accuracy metrics and the resources required for training and prediction.

- **Ordinary Least Squares (OLS):** the computational complexity of training OLS involves calculating the regression coefficients using the formula $\beta = (X^T X)^{-1} X^T y$, which has a time complexity of $O(nm^2 + m^3)$, where n is the number of samples and m is the number of features. This complexity arises from computing the matrix product $X^T X$ and its inversion. Prediction with OLS is relatively efficient, with a time complexity of $O(m)$ per sample [3] [4].

- **Decision Trees:** the computational complexity of training a Decision Tree typically has a time complexity of $O(mn \log(n))$. This involves sorting the data and evaluating potential splits at each node. Prediction time is efficient, with a complexity of $O(\log(n))$ per sample, corresponding to the depth of the tree [3].

- **XGBoost:** the computational complexity to training an XGBoost model involves constructing multiple decision trees sequentially, with each tree aiming to correct errors from the previous ones. The time complexity is $O(kmn \log(n))$, where k is the number of boosting rounds or trees. This process is more computationally intensive due to the iterative nature of boosting and the need to update weights

[3] https://www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/
[4] https://marcovirgolin.github.io/extras/details_time_complexity_machine_learning_algorithms/?utm_source=chatgpt.com

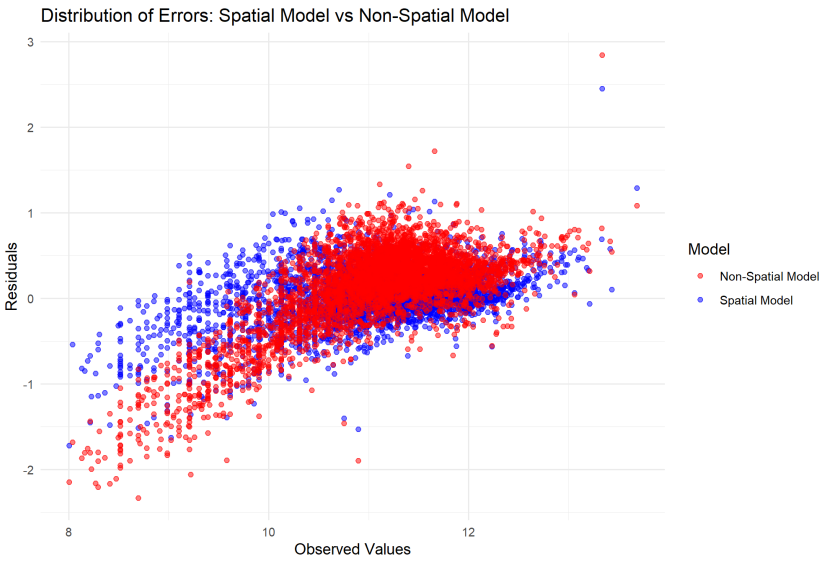
based on previous models. Prediction complexity is $O(k \log(n))$ per sample, as it aggregates the outputs from all trees in the ensemble [3].

The decision to use a more complex model like XGBoost should be based on a careful assessment of the available computational resources and the necessity for higher predictive accuracy. If computational resources are limited or rapid predictions are required, OLS or Decision Trees might be more appropriate choices. In contrast, if achieving the best possible predictive performance is critical and resources permit, the additional computational complexity of XGBoost is justified.

Are 'spatial variables' an important explanatory factor?

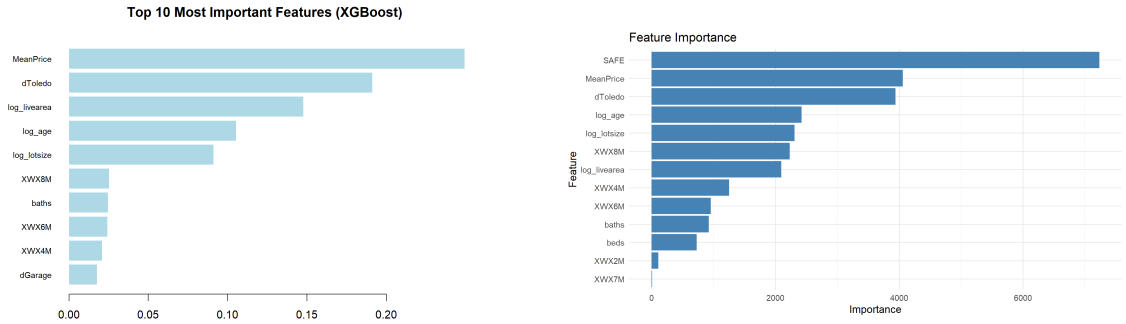
Running a modified version of OLS, without the spatial variables we obtain the following output of MAE & RMSE:

```
MAE (NO spatial): 0.3905381
RMSE (NO spatial): 0.4976525
MAE (Spatial model): 0.2151571
RMSE (Spatial model): 0.3002817
```



The spatial model has significantly lower MAE and RMSE compared to the non-spatial model. Specifically, the MAE for the non-spatial model is 0.3905, and the RMSE is 0.4977. For the spatial model, the MAE is 0.2152, and the RMSE is 0.3003. These results suggest that the spatial model performs better in terms of predictive accuracy. MAE measures the average size of errors in predictions, regardless of direction, while RMSE gives more weight to larger errors. A smaller MAE and RMSE indicate better model performance. Since the spatial model has both lower MAE and RMSE values, it shows that including spatial variables significantly improves the model's ability to make accurate predictions. This suggests that spatial variables are indeed important explanatory factors, helping to capture spatial dependencies or patterns that the non-spatial model does not account for. Therefore, spatial variables appear to have a meaningful impact on the model and improve its performance.

For the ML feature selection we have got the following outputs that confirm that spatial variables are fundamentals: In conclusion, it can be observed that when spatial vari-



Comparison between XGBoost (left) and Decision Tree (right) top 10 features in the model.

ables are removed from the OLS model, the performance of the model deteriorates. This suggests that spatial factors play a crucial role in explaining the variation in the data. Furthermore, during the feature selection process for both machine learning models, spatial variables consistently ranked at the top, significantly outpacing other features. This reinforces the conclusion that spatial variables are an important explanatory factor across all models.

Technological choices and code

The project was totally done in Visual Studio Code (VSC); given the modest computational complexity and size of the code, I did not decide to implement code engineering policies via CI-CD or Agile development methodologies.
The code is made in single classes, developed in a single "ready to execute" script with no external classes written in S3, S4 and R6. The code therefore is a script and does not follow OOP rules.

Deployment of the code was done via GitHub (the major Version Control System) linked to Visual Studio, with a push to the main branch for each significative update of the code. A version of the code is available at the public repository on GitHub shown below.

Warning!
To download the file, go to the top right and press "code" and then "Download Zip "which also includes the dataset. Also the current working directory should be changed in the code.

Download from GitHub