House Price Forecasting with Supervised Learning Models in R

Hariram Gnanachandran

April 2025

1 Project Overview

1.1 Objective

The primary objective of this project is to build predictive models that estimate the sale price of houses using various property-related features. By leveraging different machine learning algorithms, the goal is to evaluate which model most accurately captures the underlying patterns in housing data and can be used for reliable price prediction.

1.2 Dataset

The dataset used in this project is from the Kaggle competition **House Prices: Advanced Regression Techniques**. This dataset provides both training and testing sets for the task of predicting house prices in Ames, Iowa.

- **Size:** The training dataset contains 1,460 different houses (rows) and 81 features (columns).
- **Test Set:** The test dataset contains 1,459 rows with the same features but without the target variable (SalePrice).
- **Source:** Kaggle (House Prices: Advanced Regression Techniques Dataset)

1.3 Target Variable

The target variable for this project is **SalePrice**, which represents the final sale price of the house in US dollars.

1.4 Features

A variety of features were used for prediction after preprocessing and feature selection. These include, but not limited to:

- **GrLivArea:** Above-ground living area (in square feet).
- **OverallQual:** Overall material and finish quality (ordinal, scale of 1–10)
- **YearBuilt:** Year the house was built
- **GarageCars:** Number of cars that fit in the garage
- **TotalBsmtSF:** Total square footage of the basement
- **Neighborhood:** Physical locations within Ames city (categorical)
- **FullBath:** Number of full bathrooms
- **LotArea:** Lot size in square feet
- **KitchenQual:** Kitchen quality (ordinal)

Additional features were engineered or one-hot encoded during preprocessing to enhance model performance.

2 Data Collection & Preprocessing

2.1 Data Collection

The dataset used in this project was sourced from the **Kaggle House Prices - Advanced Regression Techniques competition**. This dataset contains information about homes sold in Ames, Iowa, with various features that describe the properties and their sale prices. It includes both numerical and categorical variables, such as the size of the house, number of rooms, neighborhood, and year built.

2.2 Data Inspection

Before starting any pre-processing or cleaning, an initial inspection of the dataset was conducted to understand its structure. This was done by:

1. **Previewing the Data:** Used functions like str() in R to view the structure of the dataset. This helped identify the variables and their types (e.g., numeric, categorical).
2. **Summary Statistics:** Generated summary statistics (e.g., summary()) to understand the distribution of numerical features and check for any immediate issues, such as extreme values or skewed distributions.

2.3 Data Cleaning & Preprocessing

The data preprocessing stage focused on handling outliers and NA values.

- **Handling Outliers:** The cleanData() function was used to identify and remove outliers from the dataset based on the SalePrice column. This involved:
    1. **Initial Visualisation:** A boxplot of the SalePrice distribution was generated to visually inspect outliers. This helped highlight extreme values that could skew the model. Figure 1 shows the generated boxplot.

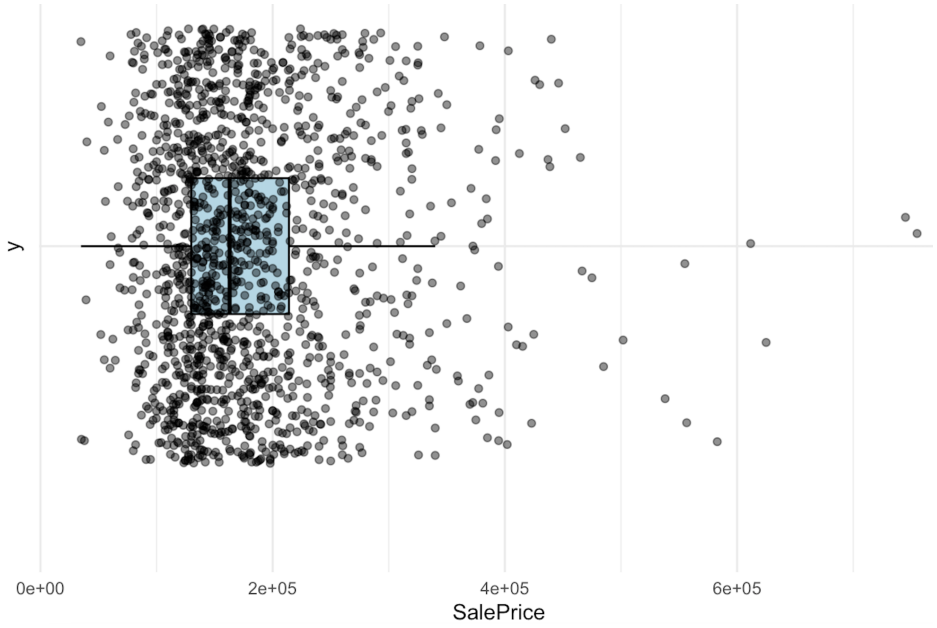


Figure 1: Box Plot of Sale Price ($) with Outliers

    2. **Outlier Detection:** The 1st and 99th percentiles of SalePrice were calculated using the quantile() function, these were used as thresholds to define "acceptable" data points. The assumption was that values below the 1st percentile and above the 99th percentile were potential outliers.
    3. **Filtering Outliers:** The dataset was filtered to retain only rows where SalePrice was within the 1st–99th percentile range.This helped to reduce the influence of extreme values during model training.

4. **Post-Cleaning Visualisation:** Another boxplot was generated using the cleaned data to verify that the extreme outliers had been successfully removed and the distribution looked more reasonable. Figure 2 shows the generated boxplot.

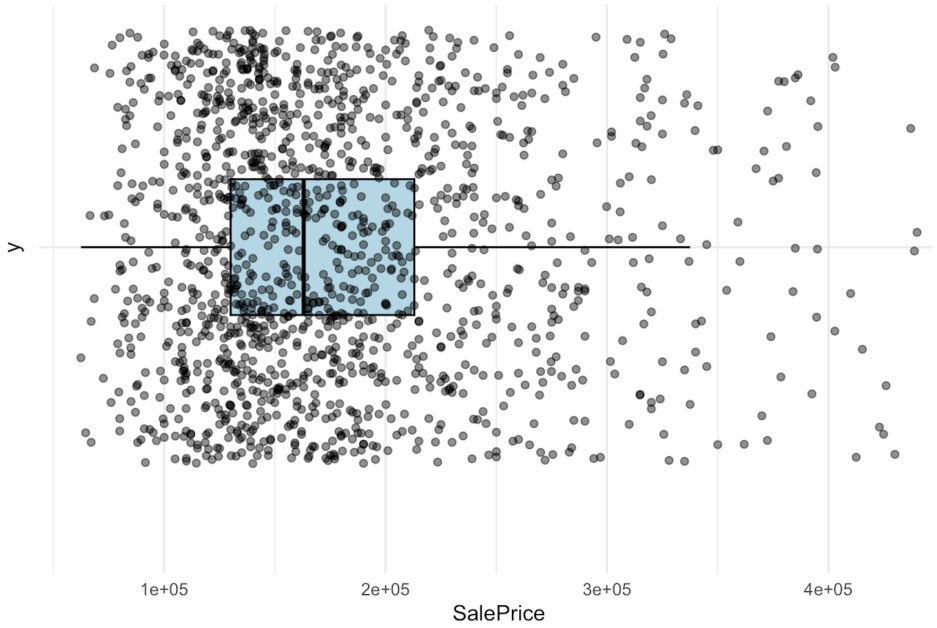


Figure 2: Box Plot of Sale Price ($) with No Outliers

- **Handling NA Values:** After outlier removal, the cleaned dataset was passed into the changeNA() function to deal with missing values.

  1. **Iterating Through Columns:** The function iterates through each column in the dataset to check for missing (NA) values.
  2. **Numeric Columns:** If a column is numeric, all NA values were replaced with 0. This was based on the assumption that a missing numerical value likely meant the absence or non-existence of that feature (e.g., missing garage area could mean no garage).
  3. **Categorical Columns:** If a column is categorical, all NA values were replaced with the string "none". This explicitly marks that the feature was not present (e.g., a missing fireplace quality could mean no fireplace).

  This strategy ensured that:

  – No rows were lost due to missing data.
  – Models could still use those features after imputation.
  – Downstream transformations (like one-hot encoding) wouldn't break due to NAs.

3 Exploratory Data Analysis

To better understand the structure and distribution of the data, various visualisations were created.

3.1 Histogram - Distribution of House Prices

Figure 3 displays the generated histogram for the distribution of sales price.

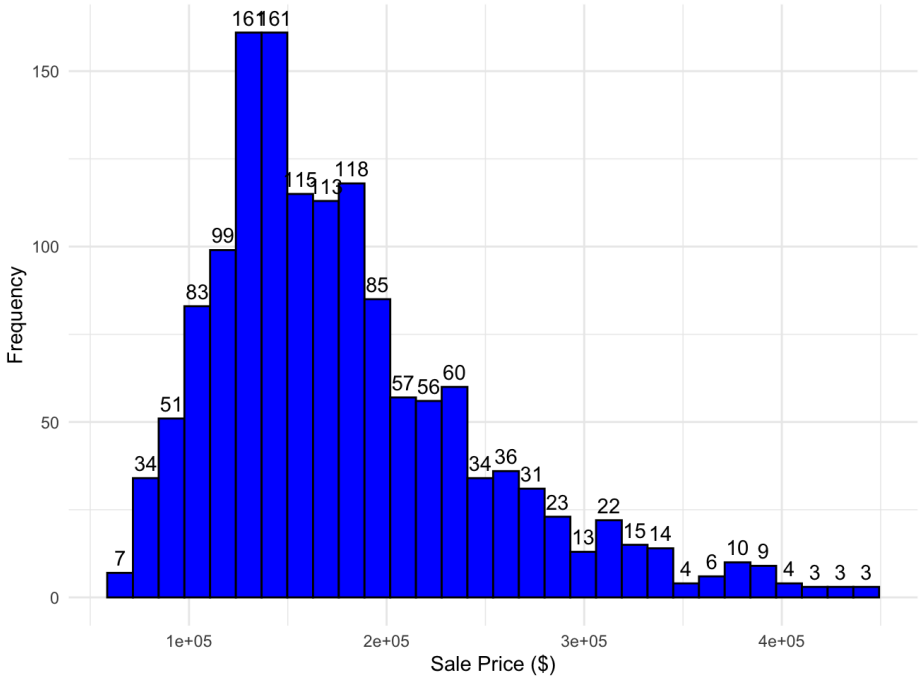


Figure 3: Histogram of the Distribution of Sale Prices ($)

The distribution is **right-skewed**, indicating that most houses are moderately priced, while a few high-end properties pull the tail of the distribution. This skewness may negatively affect certain regression models, so a log transformation was considered in later stages.

3.2 Numerical Variables: Correlation Heatmap

To understand relationships between the numerical variables, a correlation matrix was computed using Pearson correlation. This matrix quantifies the linear relationship between pairs of variables, ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation). Values close to 0 suggest no linear correlation. A heatmap was then used to visualise this matrix, with warmer colors indicating stronger positive correlations and cooler colors representing negative ones. This allowed for quick identification of highly correlated features that may contribute significantly to predicting house prices. Figure 4 show the heatmap created of the linear correlation between numerical variables.

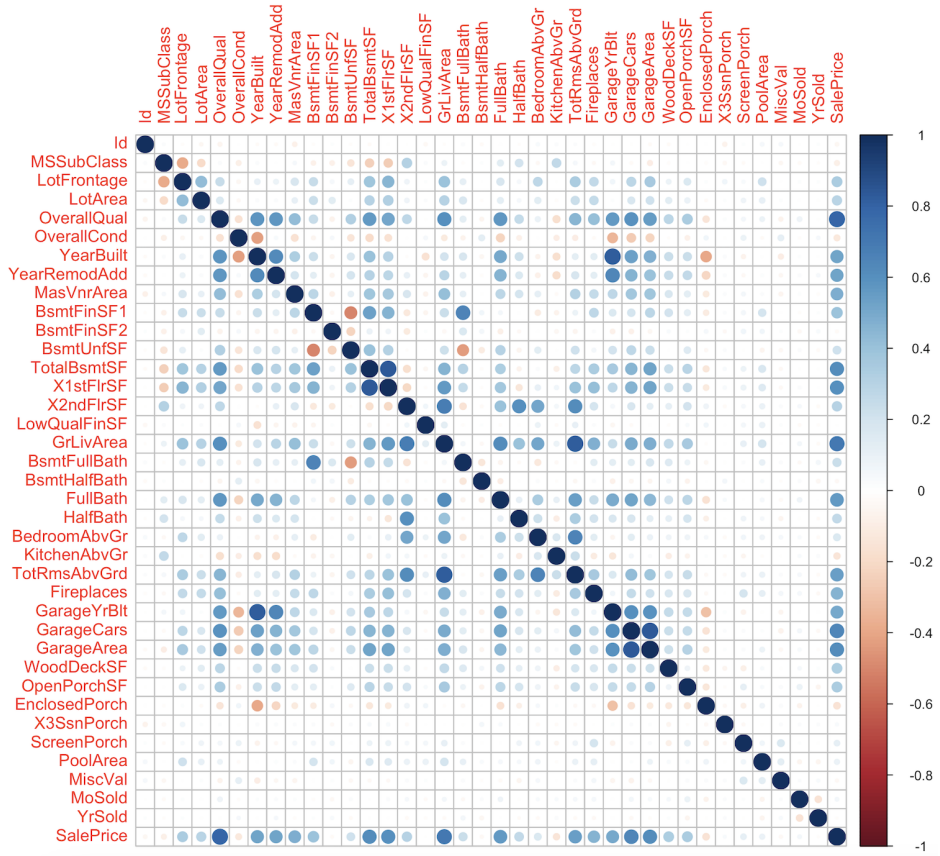


Figure 4: Heat Map of the Correlation between Numerical Variables

3.3 Numerical Variables: Trends and Key Insights

After performing Exploratory Data Analysis (EDA) using various visualisations and correlation metrics, several important trends and relationships were identified in the dataset:

- **OverallQual (Overall Material and Finish Quality)** showed a **strong positive correlation** with SalePrice, making it one of the most predictive features.

  To further illustrate this relationship, I plotted a violin plot (figure 5) to show the distribution of SalePrice across different levels of OverallQual. As seen in the plot, the SalePrice distribution significantly increases as the OverallQual rating improves. Higher-quality homes are clearly associated with higher prices, with the median price rising as the quality rating increases.

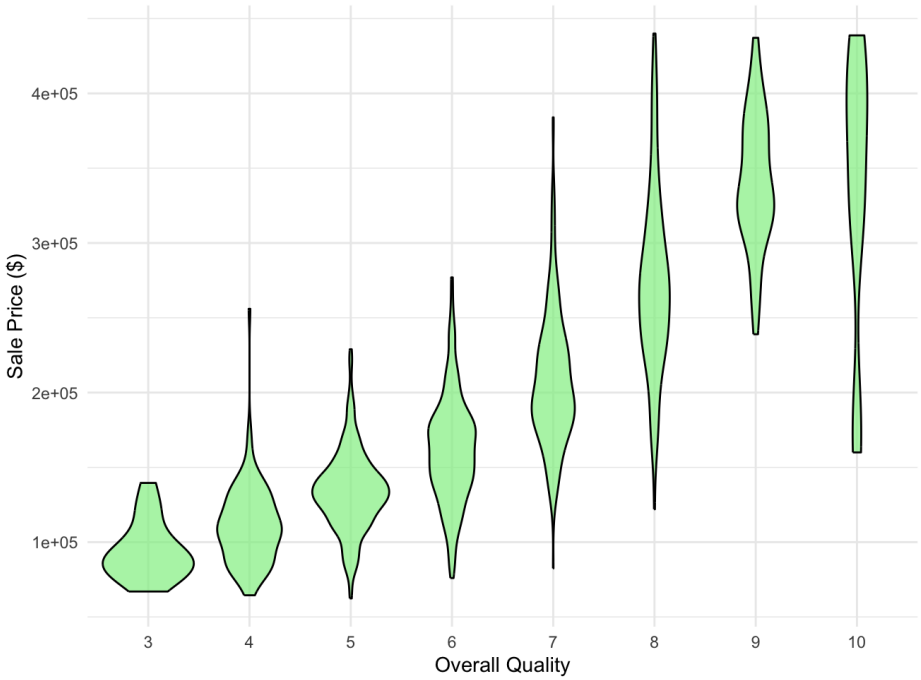


Figure 5: Violin Plot of the Overall Quality against Sales Price($)

- **BsmtFinSF1** and **X1stFlrSF** showed a **moderately strong positive correlation** with sales price, whilst **X2ndFlrSF** seemed to show a **weak positive correlation**. To visualise these relationships side by side, I used a facet plot, plotting SalePrice against each floor area variable.

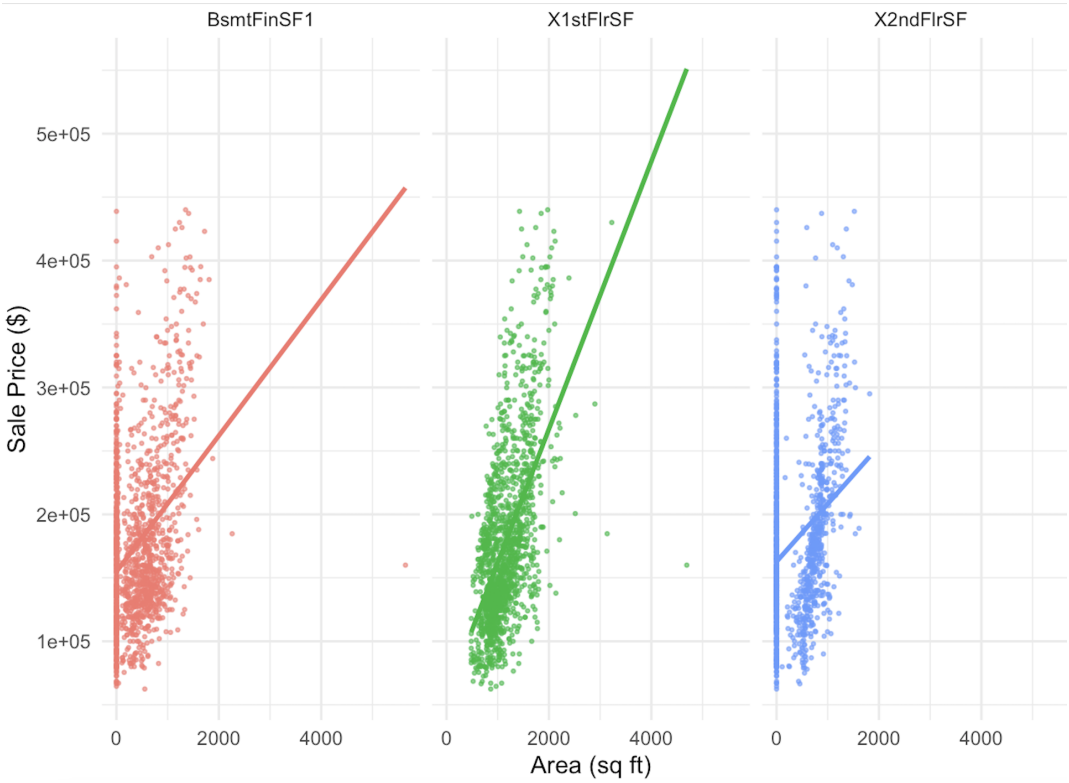


Figure 6: Violin Plot of the Overall Quality against Sales Price($)

The facet plot below displays scatter plots of SalePrice versus each floor area variable, enabling a direct comparison:

- **X1stFlrSF** shows a strong positive trend. Homes with larger first floors tend to have higher sale prices. This relationship is consistent and linear, making it a strong predictor..
- **BsmtFinSF1** displays a more scattered pattern. A large number of homes have zero basement area, creating a visible spike at zero. For homes that do have basements, there is a moderate positive trend, but variability increases with basement size—suggesting that just having a basement may be more important than its size alone.
- **X2ndFlrSF** shows a similar pattern to basements: many homes have no second floor. Among those that do, larger second floors are generally associated with higher prices.

- **GrLivArea** has a **strong positive correlation** with SalePrice, confirming its importance in predicting house prices.

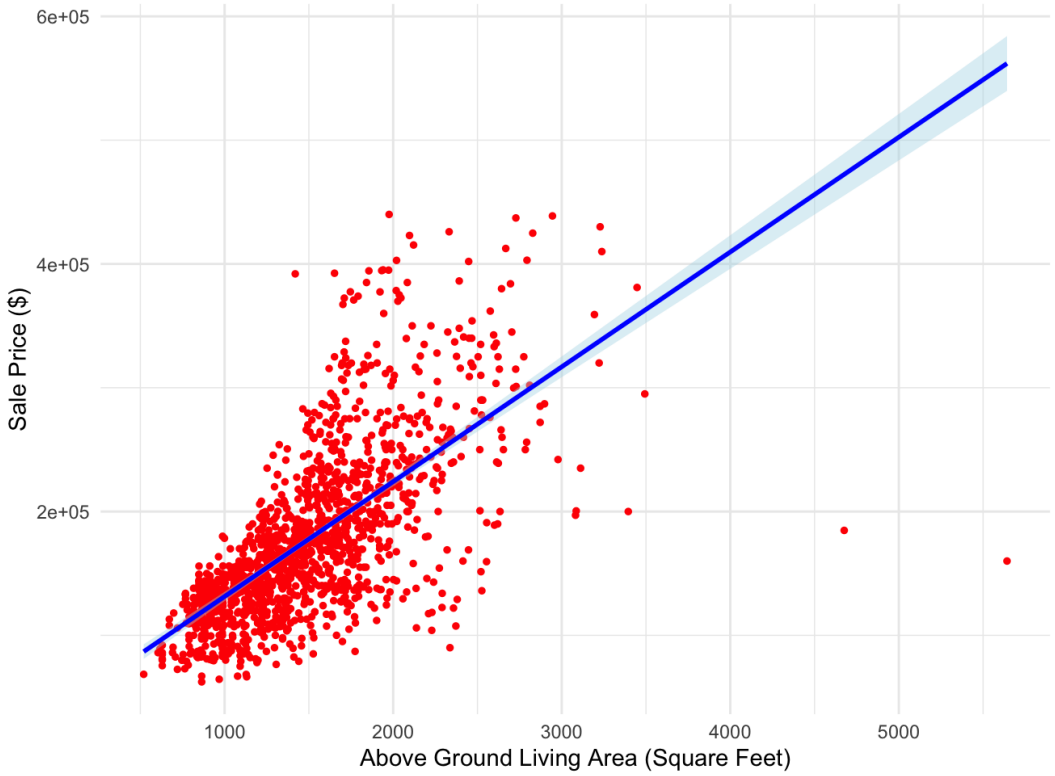


Figure 7: Linear Graph of Ground Living Area against Sale Price($)

For most data points, there is a linear relationship, especially in the mid-range of GrLivArea (1000–1500 sq ft). Some homes with large ground living areas (GrLivArea) may still have lower sale prices due to factors such as poor overall condition, outdated finishes, or being located in less desirable neighborhoods. Conversely, smaller homes may sell for higher prices if they are situated in premium locations, have high-quality construction, or include luxury features not directly reflected by size. These discrepancies highlight the importance of considering multiple variables — not just size — when assessing house value.

- Both **YearBuilt** and **YearRemodAdd** have a **strong positive correlation** with SalePrice, confirming their importance in predicting house prices. To investigate their relationships further, I used a single linear plot to show the distribution of year built and remodelled against sale price and to investigate how remodelling affects sales price, this can be seen in figure 8.

Figure 8: Linear Graphs of Year Made and Remodelled against Sale Price($)

While both YearBuilt and YearRemodAdd show a positive correlation with SalePrice, YearRemodAdd appears to have a slightly stronger relationship. This suggests that renovation work plays a significant role in determining a home's market value, sometimes even more than the original construction date.

Homes that were built several decades ago but have been remodelled in recent years often have sale prices comparable to new builds. This highlights how modern upgrades and improvements can enhance a property's appeal and value, even if the underlying structure is older.

On the other hand, homes with older construction and no recent remodelling tend to have lower sale prices, likely due to outdated features, wear and tear, or lower buyer interest.

- **GarageCars** and **GarageArea** have a moderate to to strong positive correlation with SalePrice, indicating that the size and capacity of a garage can influence the value of a home. These variables are also highly correlated to one another which makes sense as a larger garage will be able to fit more cars.

  A 3D interactive plot was used to visualise how these two features interact together. While a 2D plot might suggest similar insights, the 3D plot made these patterns easier to spot and allowed for more intuitive exploration of the relationships.

  The interactive 3D plot revealed some notable patterns:

  - SalePrice tends to increase with both Garage Area and Garage Cars, but the relationship is not strictly linear. There are cases where additional garage capacity or area does not lead to a proportionate increase in price, suggesting diminishing marginal returns.
  - Clusters of data points with high Garage Area but low car capacity indicate properties that may use the space for other purposes (e.g., storage or workshops), or have inefficient layouts.
  - Conversely, some homes accommodate more cars with relatively smaller garage areas, pointing to more space-efficient garage designs.

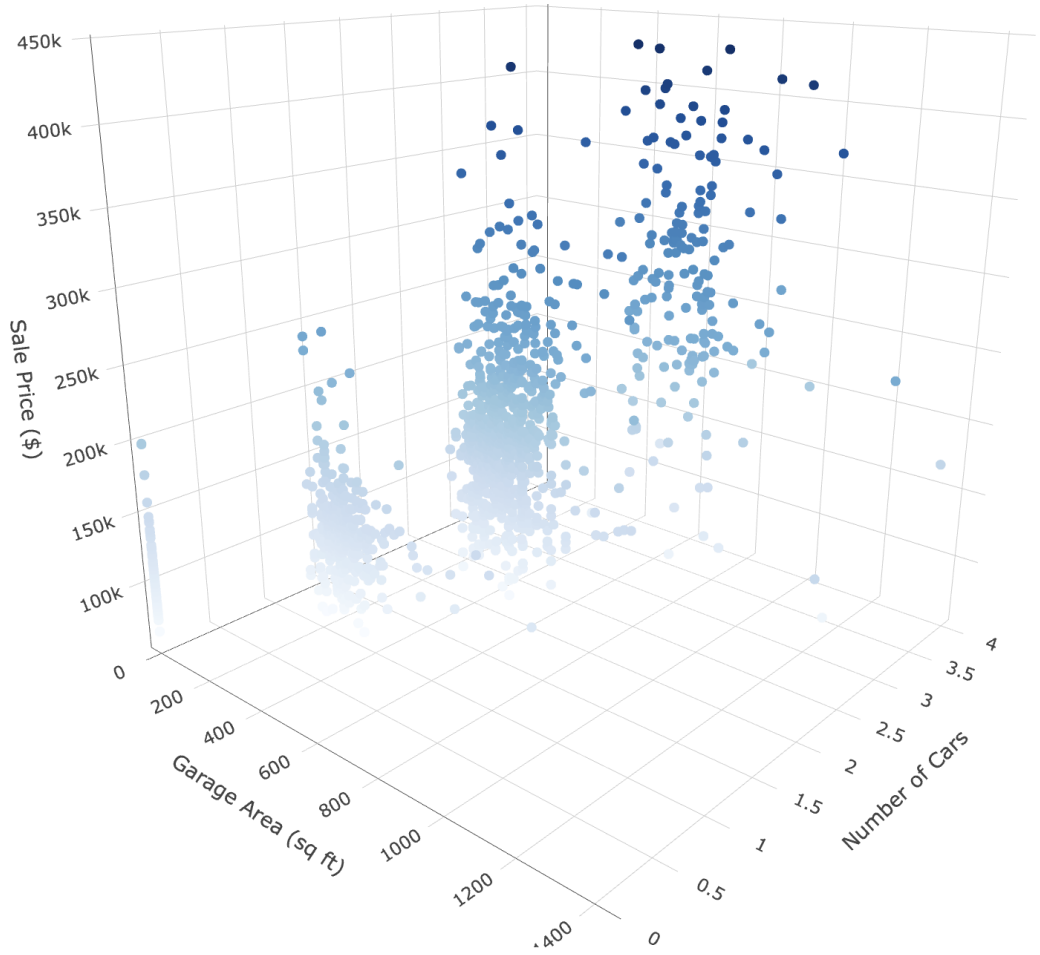


Figure 9: 3D Plot of Garage Area and Number of Cars against Sale Price($)

3.4 Multicollinearity

Multicollinearity refers to the presence of high correlations among independent variables in a regression model. When predictor variables are highly correlated, it becomes difficult to determine the individual effect of each variable on the response variable, leading to unstable coefficient estimates and challenges in model interpretation. This section looks into the correlations with the highest indication of multicollinearity:

– A particularly strong positive correlation was observed between **TotRmsAbvGrd** and **GrLivArea**. This relationship is intuitive, as homes with a larger living area typically accommodate more rooms. To investigate this relationship further, a violin plot was used to show this distribution in Figure 10.

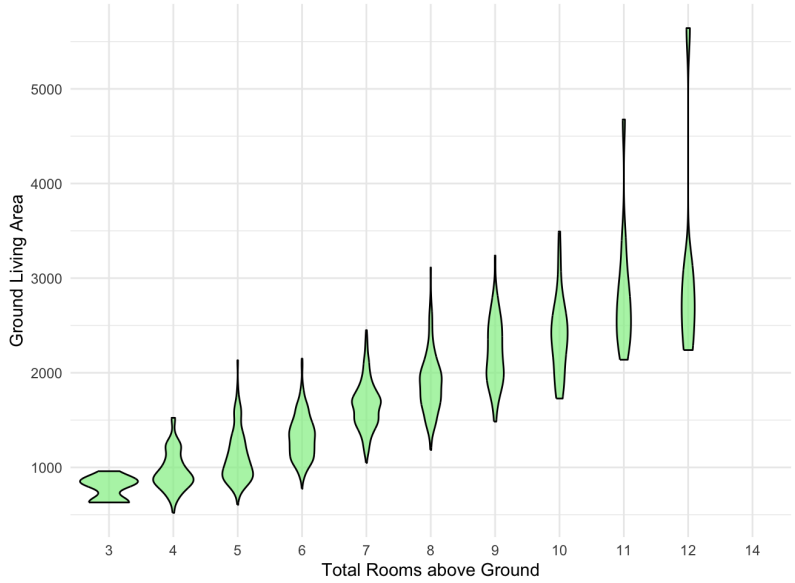


Figure 10: Violin Plot of Total Rooms above Ground against Ground Living Area (Square Feet)

– Another notable case of multicollinearity was found between **YearBuilt** and **GarageYrBlt**. In the majority of records, the garage was constructed in the same year as the house, resulting in a strong correlation between these two variables. In some instances, the garage was built at a later date—likely due to renovations or additions—which introduces slight variation. To visualise this relationship, a linear plot was plotted as seen in Figure 11 below.

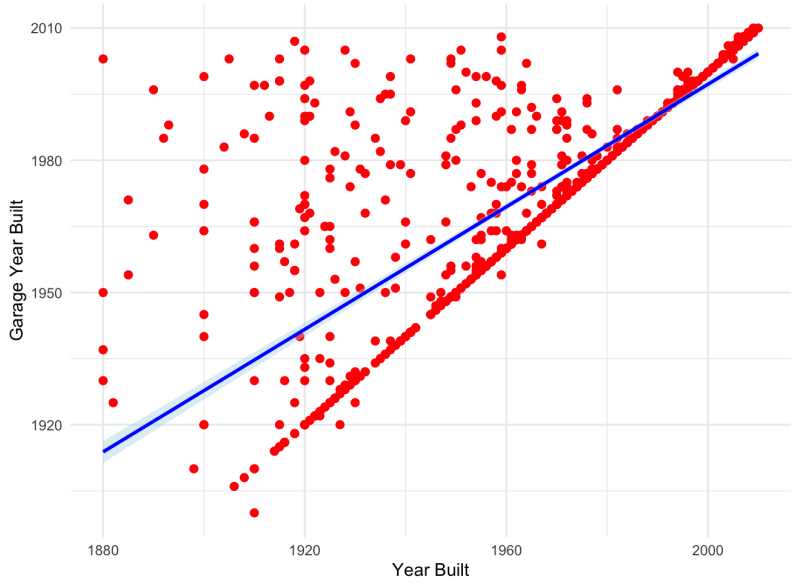


Figure 11: Linear Plot of Year Built against Garage Year Built

– A strong linear relationship was also identified between **TotalBsmtSF** and **X1stFlrSF**. In the majority of observations, these two values were nearly identical, reflecting homes with a full basement directly beneath the first floor. There were a few exceptions where the first floor was larger—likely due to additions or non-basement spaces—and a very small number of homes with a basement area exceeding the first floor, possibly due to architectural anomalies like split-level designs. This relation can be seen in Figure 12 below.

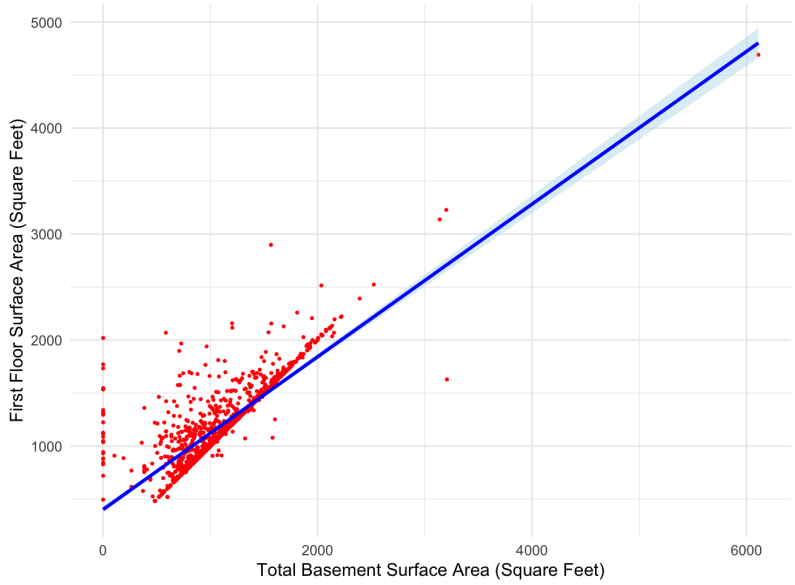


Figure 12: Linear Plot of Basement Area against First Floor Area

– When examining the relationship between **FullBath** and **GrLivArea**, a general upward trend was observed: larger homes tend to have more full bathrooms, with counts increasing up to three. Interestingly, however, homes with zero full bathrooms exhibited higher average living areas than those with just one. This counterintuitive result could reflect specific architectural or usage patterns—for example, properties classified as having no full bathrooms might still include multiple half baths, or they may be custom-designed homes where bathroom count doesn't follow standard expectations. Despite this anomaly, the overall positive correlation beyond one bathroom supports the assumption that as home size increases, so does the likelihood of having additional full bathrooms. To explore this trend further from the heatmap, the following violin plot was created:

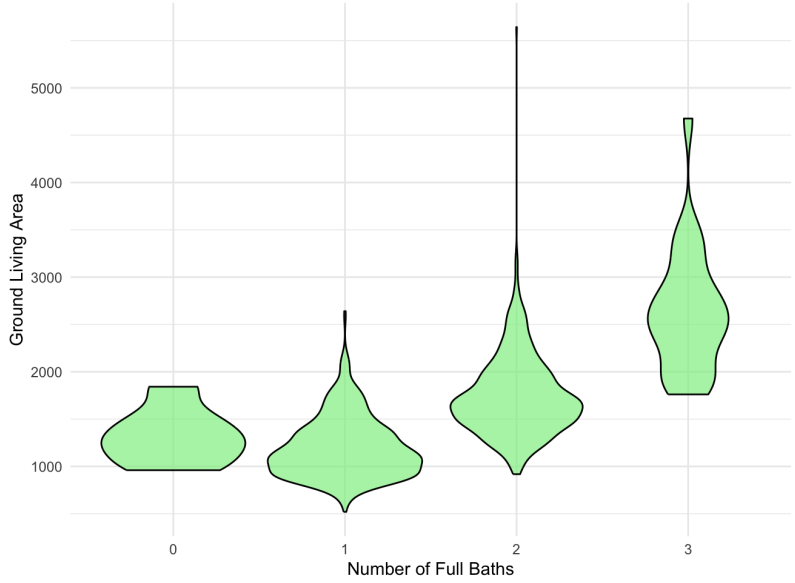


Figure 13: Violin Plot Full Bath against Ground Living Area

4 Feature Engineering

4.1 Categorical Variables: One-Hot Encoding

Many of the features in the dataset are categorical in nature (e.g., Neighborhood, HouseStyle, Exterior1st). In order to incorporate them into predictive models, these variables needed to be converted into numerical format. This was done using one-hot encoding, which creates binary columns for each category within a feature.

The key steps involved in this one-hot encoding process are as follow:

1. **Factor Conversion:** All character-type columns are converted to factors. This is necessary because dummyVars() from the caret package only handles factor levels during dummy variable creation.
2. **Dummy Variable Creation:** The dummyVars() function is used to create a blueprint for encoding, with the parameter fullRank = TRUE to avoid the dummy variable trap (i.e., one category is dropped to prevent multicollinearity).
3. **Application of Transformation:** The blueprint is applied to the original dataset using predict(), creating a new dataset where each level of a categorical variable is now a separate binary column.
4. **Return of Encoded Data:** The encoded matrix is converted back to a data frame and returned for use in modeling.

Due to the large number of one-hot encoded features, the full correlation matrix became too extensive to include in the main body of the report. Therefore, it is provided in **Appendix A** for reference. The key findings from the correlation analysis, particularly those relevant to predicting SalePrice, are summarized and visualized in the following sections.

4.2 Creation of New Variables

Feature engineering involves the creation of new variables derived from existing features to better capture relationships and patterns to improve the model performance.

Below is a breakdown of the engineered features and the purpose of their creation:

1. **Garage Efficiency (GarageEff)**
   - **Definition:** Ratio of GarageArea to GarageCars.
   - **Purpose:** Determines how efficiently space is used. A poor ratio may indicate poor design, which may influence the price.
2. **Overall House Score (OverallScore)**
   - **Definition:** Product of overall condition (OverallCond) and overall quality (OverallQual).
   - **Purpose:** A combined metric that considers both condition and quality, giving a more holistic view of the build.
3. **Total Porch Area (totalPorch)**
   - **Definition:** Sum of OpenPorchSF, EnclosedPorch, ScreenPorch, and 3SsnPorch.
   - **Purpose:** Reflects the total outdoor living space, which could be a strong selling point
4. **House Age (HouseAge)**
   - **Definition:** 2025 - YearBuilt
   - **Purpose:** Measures how old the property is, which may negatively impact the sale price due to aging design and infrastructure
5. **Years Since Remodel (YearSinceRemodel)**
   - **Definition:** 2025 - YearRemodAdd
   - **Purpose:** Captures how recently the house was updated. More recent remodels often suggest better interior standards or up-to-date features.
6. **Season Sold (SeasonSold)**

- **Definition:** Categorical feature based on MoSold, mapping months into seasons:
  - Winter: Dec–Feb
  - Spring: Mar–May
  - Winter: Summer: Jun–Aug
  - Autumn: Sep–Nov
- **Purpose:** Seasonal market fluctuations may impact house prices due to demand variation across the year.

7. **Lot Area per Room (LotPerRoom)**
   - **Definition:** LotArea / TotRmsAbvGrd
   - **Purpose:** Measures average lot space per room, potentially reflecting spaciousness and layout efficiency.
8. **Sale Price per Area (SalePricePerArea)**
   - **Definition:** SalePrice / LotArea
   - **Purpose:** A normalised price metric helping compare pricing relative to lot size.

To understand the relationships between these new features and the target variable (SalePrice):

- The categorical variables (e.g., SeasonSold) were one-hot encoded using the oneHotEncode() function.
- A correlation matrix was generated using the createCorrMat() function to identify which engineered features showed the strongest relationships with house prices.

Figure 14 displays the generated heat map of the corrlation between the newly created variables.

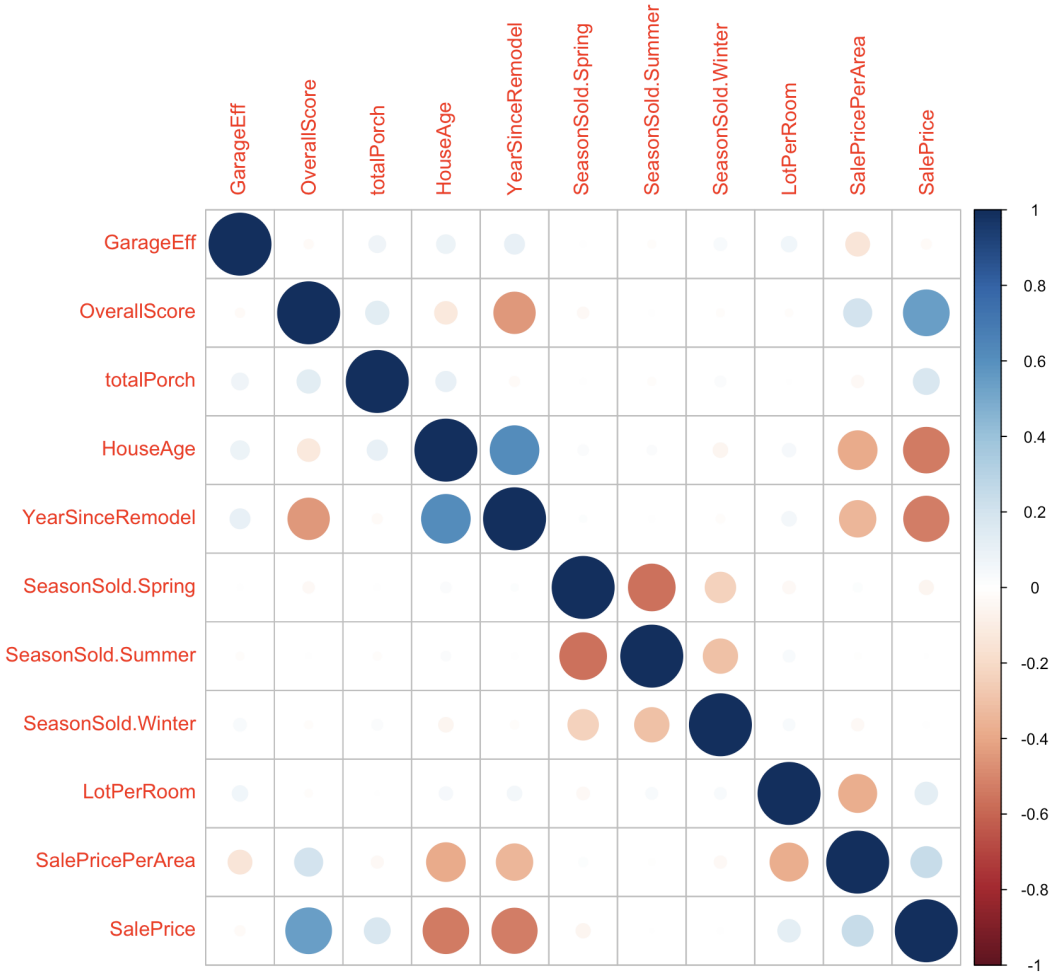


Figure 14: Heat Map of the Correlation between Newly Created Variables

4.3 Analysis of Engineered Features

Key observations from the engineered features and their relationship with sale price include:

- **GarageEff** showed unexpected no correlation with sale price indicating that space efficiency isn't as inherently important to sale price as compared with the overall size of the garage
- **OverallScore** as expected showed a moderately strong correlation with sale price
- **totalPorch** surprisingly had a weak positive correlation rather than a much stronger one, suggesting the porch area may not be as significant as initially expected
- **YearSinceRemodel** and **HouseAge** showed expected trends, with newer or more recently remodelled homes generally valued higher.
- **seasonSold**, after one-hot encoding, showed no seasonal variation in pricing.
- **LotPerRoom** and **SalePricePerArea** both showed mild-low correlation with sale price. Although the size of the house may boost the price slightly, it does not consider more important aspects, like location, neighborhood amenities, or interior quality.

4.4 Data Normalisation

To ensure consistent scaling across all numeric features, a z-score standardisation process was applied to the cleaned dataset. This involved:

1. **Identify Numeric Columns:** Each column is checked to determine whether it is numeric. Only numeric variables are considered for normalisation, apart from Sale Price.
2. **Apply Z-Score Standardisation:** For each of the numeric varaibles, values were normalised using the formula:

$$Z = \frac{X - \mu}{\sigma} \tag{1}$$

   where:
   - Z is the Normalised Value
   - X is the Observed Value
   - μ is the Mean of the Feature
   - σ is the Standard Deviation of the Feature

This step helps prevent features with larger numeric ranges from disproportionately influencing model training, particularly in algorithms sensitive to feature magnitude.

4.5 Data Splitting

To evaluate model performance effectively, the dataset was split into training and testing subsets. This ensures that the models are trained on one portion of the data and evaluated on unseen data to mimic real-world performance. This involved:

1. **Define Split Ratio:** An 80/20 split was used, where 80% of the data was allocated for training and 20% for testing. This provides a good balance between giving models enough data to learn and keeping enough data to evaluate generalisation.
2. **Random Sampling:** The dataset was randomly shuffled before splitting to prevent any ordering bias (e.g., by year, size, etc.) that could affect performance.
3. **Create Training and Test Sets:** The data was partioned into a training dataset, used for training all models, and a testing dataset, reserved exclusively for final evaluation.

5 Model Building & Tuning

5.1 The Baseline Model: Intercept-Only Model

The intercept-only model is a statistical model in which only the intercept is included, without any predictor variables. In regression analysis, this model predicts the mean of the dependent variable across all observations, assuming no influence from independent variables [1].

In this scenario, it will be used as the baseline model to assess the performance of more complex models. Essentially acting as the null hypothesis, suggesting that none of the predictors have a significant effect on the outcome of sale price.

The mathematical representation of this model [1]:

$$y_i = \beta_0 + \varepsilon_i \tag{2}$$

Where:

- y_i = Predicted Value
- β_0 = Intercept Value (Mean of all Observed House Sale Prices)
- ε_i = Error term for Observation i

5.2 Multiple Linear Regression Model

The multiple linear regression model is a statistical model and a supervised learning algorithm used to model the relationship between a continuous target variable and multiple independent variables. It extends simple linear regression through incorporating more than one predictor variables [2].

The mathematical representation of this model [2]:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i \tag{3}$$

Where:

- y_i = Predicted Value
- β_0 = Intercept Value (Mean of all Observed House Sale Prices)
- β_1, β_2, ..., β_p = Coefficients representing the Effects of each Independent Variable
- x_{i1}, x_{i2}, ..., x_{ip} = Values of the Independent Variables for Observation i
- ε_i = Error term for Observation i

Assumptions made by the model [2]:

- Linearity between the predictors and the outcome
- Independence between residuals
- No multicollinearity between predictor variables

5.3 Ridge Regression

Ridge regression is a regularisation technique used to prevent overfitting in linear regression through adding a penalty to the size of coefficients. Ridge regression shrinks the coefficients evenly across all predictors but doesn't eliminate any predictors completely [3].

It's typically used with models that suffer from high multicollinearity or higher number of predictor variables than observations.

5.3.1 Tuning Hyperparameters

For Ridge regression, the key hyperparameter is *lambda* (often referred to as the regularisation strength), which controls the degree of penalty applied to the model's coefficients. The goal of Ridge regression is to find the optimal value of *lambda* that strikes the right balance between fitting the model well and keeping the coefficients small to avoid overfitting.

The optimal value of *lambda* can be chosen via cross-validation or through techniques like Grid Search to balance bias and variance effectively. Cross-validation is a technique where the data is split into multiple subsets, and the model is trained and tested on different combinations of these subsets. This process helps identify the lambda value that minimises prediction error and prevents overfitting. On the other hand, Grid Search is a brute-force method where a range of hyperparameter values is tested to determine the best combination for the model's performance.

Figure 15 shows a plot of the cross-validation results for the Ridge regression model, showing the relationship between the regularisation parameter (lambda) and the mean cross-validation error. The curve represents the mean error across all folds for different values of lambda. The optimal lambda value is selected as the one that results in the smallest mean error, indicated by the lowest point on the curve.

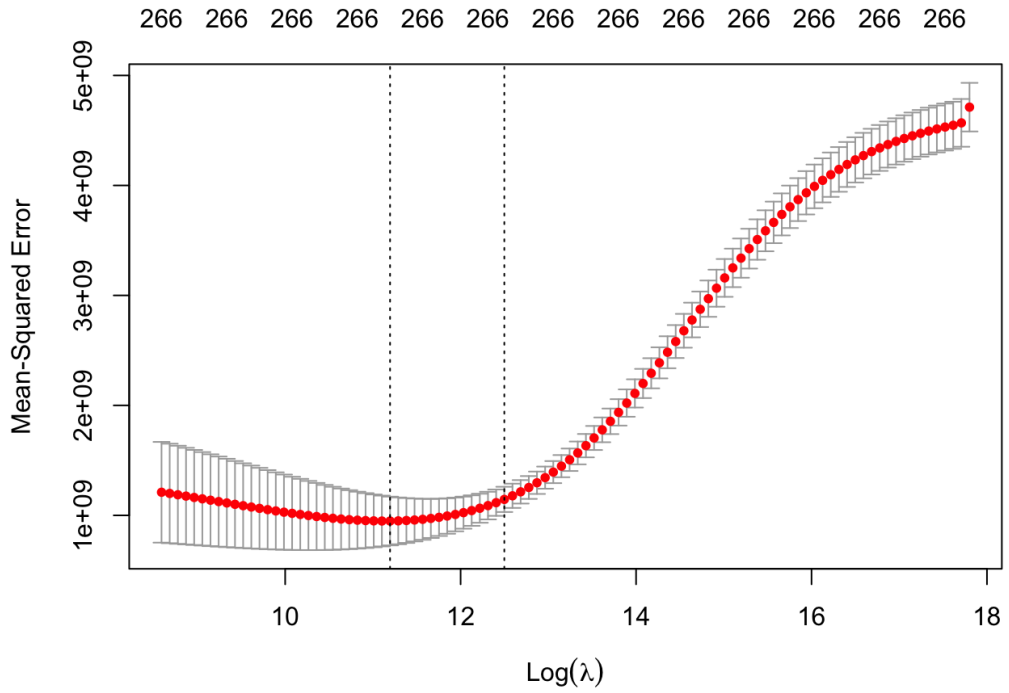


Figure 15: Cross-Validation Results for the Ridge Regression Model

5.4 Lasso Regression

Lasso regression is a regularisation technique used to prevent overfitting and perform feature selection in linear regression through shrinking some coefficients to zero [3].

This form of regularisation is useful with a dataset with a large number of predictor variables and want to perform automatic feature selection.

5.4.1 Tuning Hyperparameters

The process for tuning the Lasso Regression model follows a similar approach to that of the Lasso Regression model. The key hyperparameter that controls the regularisation strength in Lasso Regression is *lambda*

(lambda), which determines the extent of shrinkage applied to the model coefficients. A larger *lambda* applies a higher penalty, leading to more shrinkage of the coefficients, while a smaller *lambda* leads to a model that resembles a standard linear regression.

Figure 16 shows a plot of the cross-validation results for the Lasso regression model, showing the relationship between the regularisation parameter (lambda) and the mean cross-validation error.

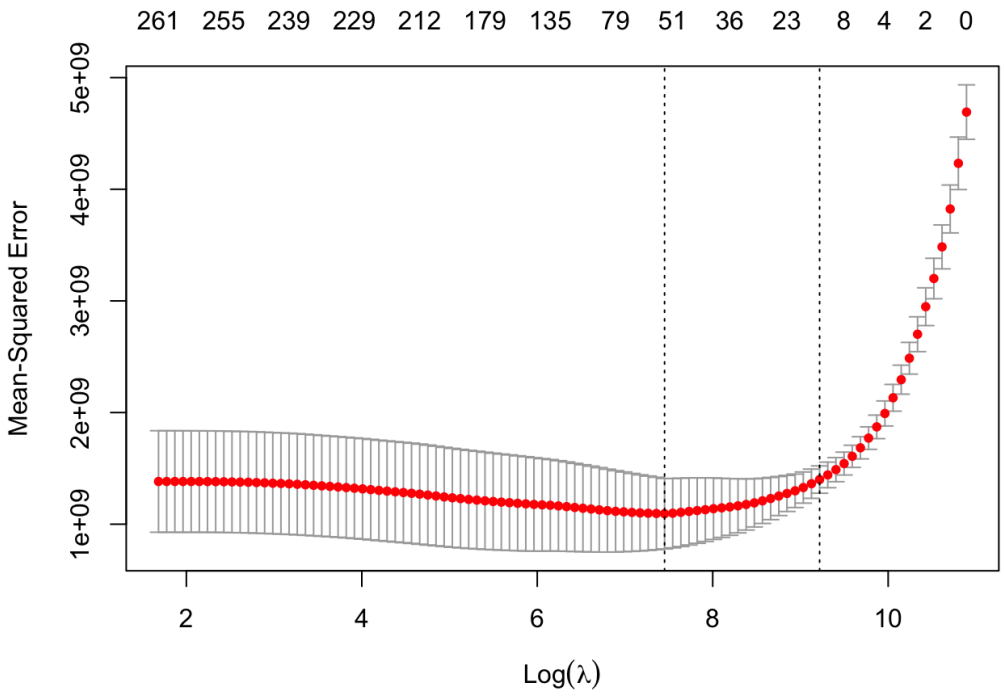


Figure 16: Cross-Validation Results for the Lasso Regression Model

5.5 Decision Tree Regressor Model

The decision tree regressor model is a non-linear machine learning algorithm that works by recursively splitting the data based on feature values to create branches and then make predictions at the leaf node. This involves [4]:

1. **Splitting the Data:** A decision tree starts with the entire dataset and splits into subsets based on the value of features. The goal of splitting is to reduce the impurity in the dataset.
2. **Selecting the Best Feature:** At each step, the algorithm evaluates all available features and selects the one that minimised impurity. This process is repeated recursively for each subset until a stopping condition is met (e.g., a maximum depth, a minimum number of samples at a node), or when further splitting does not improve the model.
3. **Tree Structure:** The result of this process is a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents a predicted value
4. **Prediction:** To predict the value for new data, the model follows the splits in the tree based on the feature values of the input data until it reaches a leaf node. The value at the leaf node is then the predicted output.

5.5.1 Tuning Hyperparameters

One of the key hyperparameters for tuning decision trees is the **complexity parameter (cp)**, which controls the minimum improvement required to create a new split. A high cp value results in a smaller tree (more

pruning), while a lower value may lead to a more complex tree that overfits the training data.

In this section, a decision tree model was trained using both grid search and 10-fold cross-validation. Grid search was used to explore a range of values for the complexity parameter (cp), which controls the tree's complexity. For each value of cp, 10-fold cross-validation was performed to evaluate the model's performance on unseen data. This method helps identify the optimal cp value that minimises the model's error and prevents overfitting. The cp value that resulted in the best cross-validation performance was selected as the final model parameter.

Figure 17 shows a plot of the cross-validation results for the decision tree regressor model, showing the relationship between the complexity parameter (lambda) and the cross-validation error.

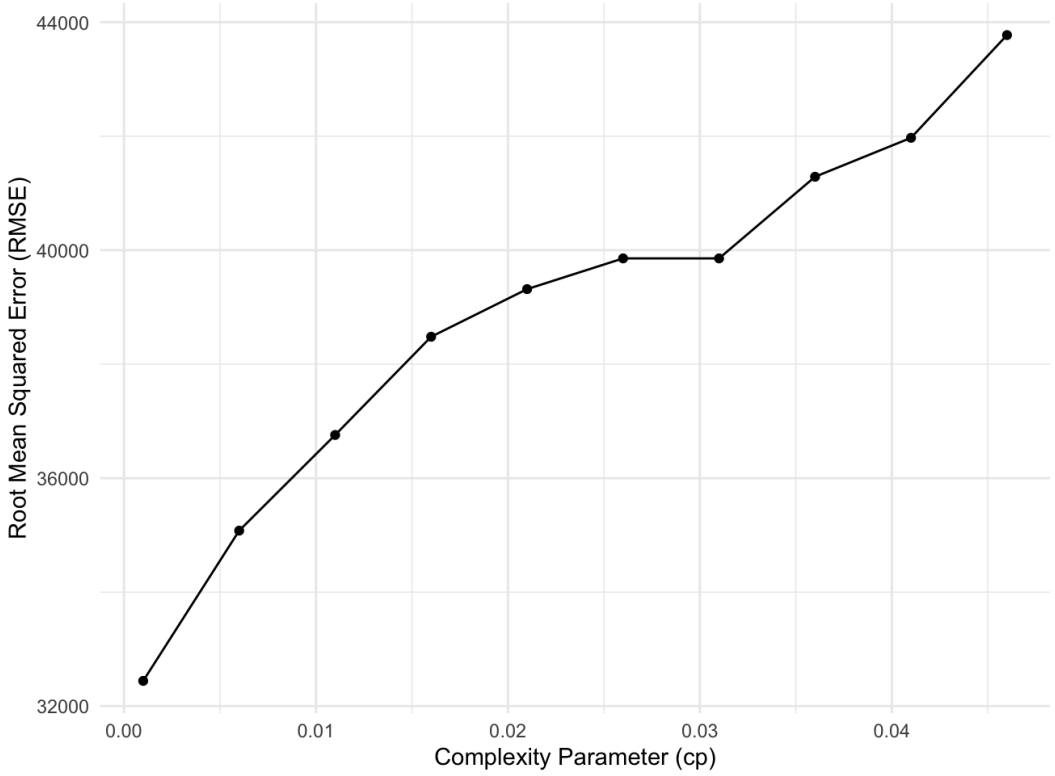


Figure 17: Cross-Validation Results for the Decision Tree Regressor Model

5.6 Random Forest Regressor Model

The Random Forest Regressor is an ensemble learning method that builds multiple decision trees and combines their outputs to make more accurate and stable predictions. Unlike a single decision tree, which can be prone to overfitting, Random Forest uses a technique called bagging (Bootstrap Aggregating) to reduce variance and improve generalisation. This involves [5]:

1. **Bootstrapping:** The algorithm creates multiple subsets of the training data by randomly sampling (with replacement). Each subset is used to train a separate decision tree.
2. **Random Feature Selection:** When building each tree, the algorithm randomly selects a subset of features at each split, rather than considering all features. This increases diversity among the trees and reduces the correlation between them.
3. **Ensemble Prediction:** For regression tasks, the final prediction is the average of the outputs from all individual trees. This ensemble approach tends to produce more robust and accurate predictions than a single tree.

5.6.1 Tuning Hyperparameters

The process for tuning the hyperparameters for the random forest regressor model followed a similar approach to that of the decision tree regressor model, where grid search with a 10-fold cross-validation was used to find the combination that gave the best performance.

The key hyperparameters tuned to improve the model's performance were:

- **ntree:** The number of trees in the forest. More trees can improve performance but increase computational cost.
- **mtry:** The number of features to consider when looking for the best split. A smaller mtry increases randomness and diversity among trees.

Figure 18 shows a plot of the cross-validation results for the random forest regressor model, showing the relationship between the number of trees (ntree) and the number of features to consider (mtry) with the cross-validation error

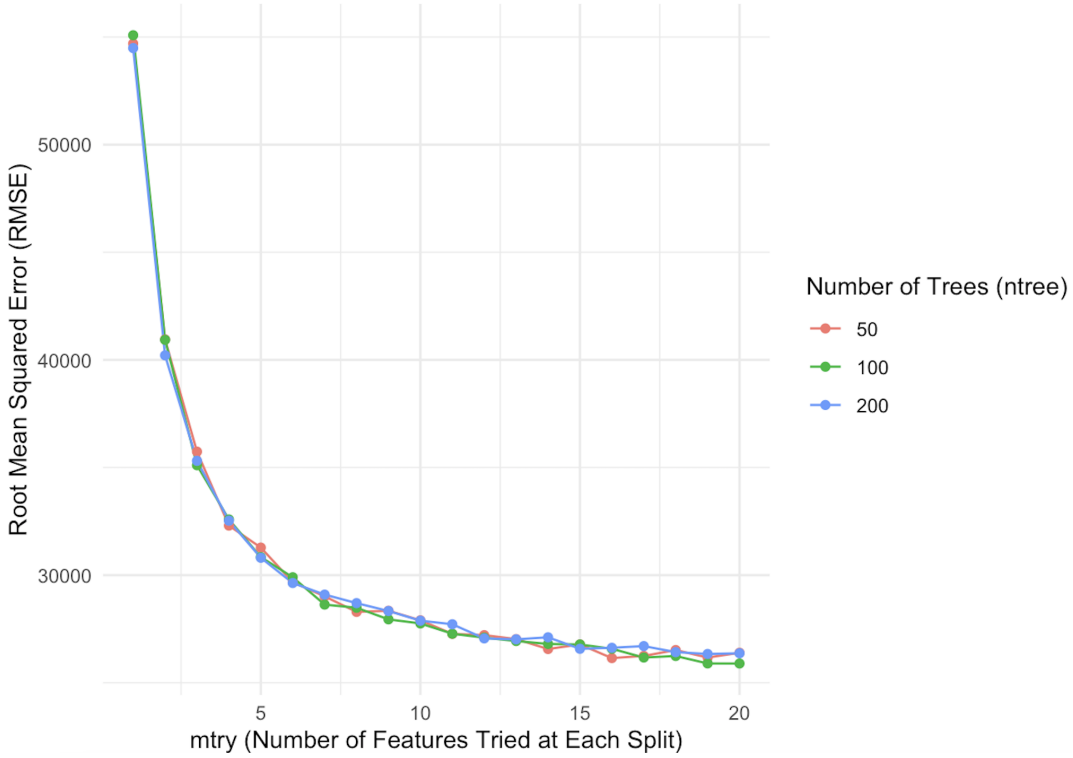


Figure 18: Cross-Validation Results for the Random Forest Regressor Model

5.7 Gradient Boosting (GBR) Model

The Gradient Boosting model is an ensemble learning method that builds a strong predictive model by combining multiple weaker models, usually decision trees. This involves [6]:

1. **Base Model:** The model begins by making a naive prediction, usually using the mean value of the target variable.
2. **Calculate Errors:** Determines the residuals, the difference between the actual values and the predicted ones.
3. **Fit New Tree to the Residuals:** A decision tree (a weak learner) is trained to predict these results. Essentially the tree is learning where the current tree is doing poorly.
4. **Update the Model:** The predictions of the new tree are added to the model, scaled by the learning rate, to adjust how much the new tree influences the final prediction.

5. **Repeat:** Steps 2-4 are repeated for a pre-defined number of iterations. At each step the model becomes better at predicting the target through reducing the residuals.

5.7.1 Tuning Hyperparameters

To optimise the performance of the Gradient Boosting model, several key hyperparameters were tuned using grid search combined with 5-fold cross-validation. The hyperparameters included:

- **n.tree:** number of boosting iterations (i.e., the number of trees), which controls how many weak learners are added sequentially. Higher values can lead to better performance but increase computation and risk of overfitting.
- **interaction.depth:** defines the maximum depth of each individual tree. Deeper trees can capture more complex interactions but may also lead to overfitting.
- **shrinkage:** controls how much each tree contributes to the overall model. Smaller values lead to slower learning and typically better generalisation, at the cost of needing more trees.
- **n.minobsinnode:** controls the minimum number of observations in a terminal node. Larger values result in more conservative trees, reducing overfitting and increasing model stability.

Each combination of these hyperparameters was evaluated using cross-validation, which helps ensure that the chosen model generalises well to unseen data. The combination that achieved the lowest validation error was selected as the optimal configuration for the final model.

Figure 19 shows a plot of the cross-validation results for the gradient boosting model, showing how the tuning the different hyperparameters affect the cross-validation error.

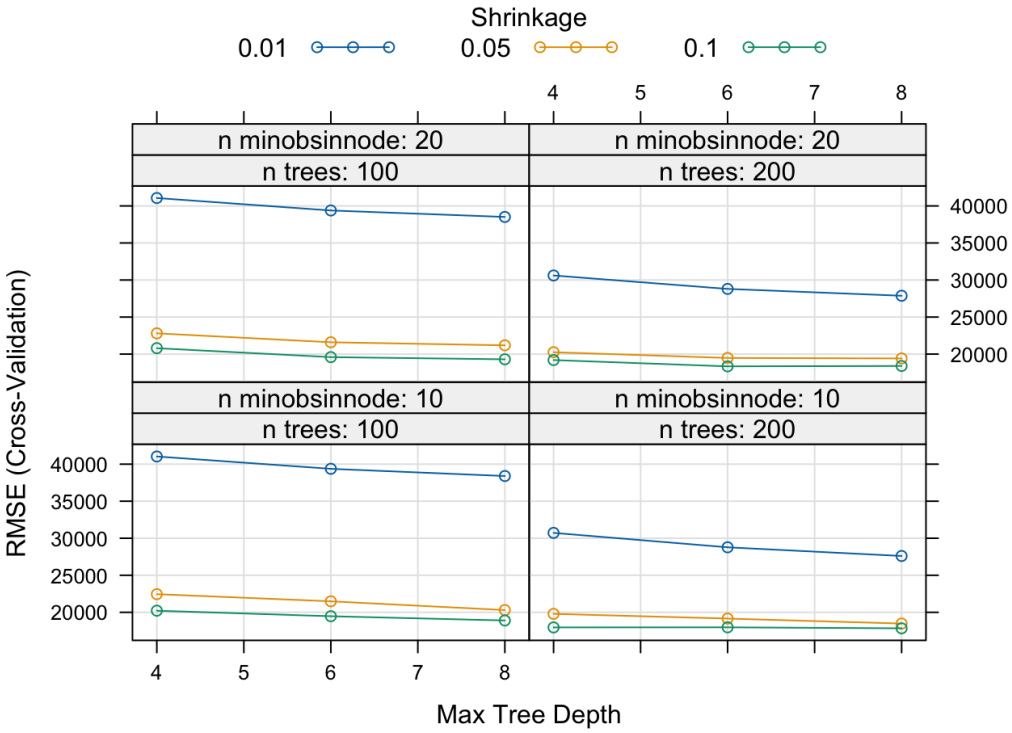


Figure 19: Cross-Validation Results for the Gradient Boosting Model

5.8 XGBoost Model

The XGBoost model is an advanced ensemble machine learning algorithm that builds on the Gradient Boosting model through applied regularisation. This involves [7]:

1. **Base Model:** The model begins by making a naive prediction, usually using the mean value of the target variable.
2. **Calculate Errors:** Determines the residuals, the difference between the actual values and the predicted ones.
3. **Fit New Tree to the Residuals:** A decision tree (a weak learner) is trained to predict these results, rather than the actual target values. Essentially the tree is learning where the current tree is doing poorly.
4. **Update the Model:** The predictions from the tree are scaled by the learning rate (called shrinkage), then added to the previous predictions to improve accuracy.
5. **Repeat:** Steps 2-4 are repeated for a pre-defined number of iterations. At each step the model becomes better at predicting the target through reducing the residuals.
6. **Apply Regularisation:** XGBoost applies penalties to tree complexity using L1 (alpha) and L2 (lambda) regularisation. This helps overfitting, especially when the data is noisy or high-dimensional.

5.8.1 Tuning Hyperparameters

To optimise the performance of the Gradient Boosting model, several key hyperparameters were tuned using grid search combined with 5-fold cross-validation. The hyperparameters included:

- **nrounds:** Number of boosting iterations (i.e., how many trees will be added). Higher values can increase model complexity and accuracy, but at the cost of overfitting if too high.
- **max_depth:** defines the maximum depth of each individual tree. Deeper trees can capture more complex interactions but may also lead to overfitting.
- **eta:** controls how much each tree contributes to the overall model. Smaller values lead to slower learning and typically better generalisation, at the cost of needing more trees.
- **gamma:** specifies the minimum loss reduction required to make a further partition on a leaf node. It acts as a regularisation parameter to control overfitting, where larger values make the algorithm more conservative.
- **colsample_bytree:** determines the proportion of features randomly sampled for each tree. Lower values introduce more randomness and can help prevent overfitting, while higher values allow the model to use more information per tree.
- **min_child_weight:** sets the minimum sum of instance weights needed in a child node. Larger values make the model more conservative by discouraging splits that yield small leaf nodes.
- **subsample:** defines the proportion of the training data to sample for each boosting round. Lower values increase randomness and help prevent overfitting, especially on noisy data.

Each combination of these hyperparameters was evaluated using cross-validation, which helps ensure that the chosen model generalises well to unseen data. The combination that achieved the lowest validation error was selected as the optimal configuration for the final model.

The results of the cross-validation for the XGBoost model can be found in Appendix B.

6 Model Evaluations

6.1 Comparisom of Model Evaluation Metrics

Table 1 displays the values of the performance metrics used to predict the multiple linear regression model's predictive accuracy.

Regression Model	RMSE	MAE	R^2
Intercept-Only (Baseline)	69032.0777039673	52807.4230830358	$-2.22044604925031 \times 10^{-}16$
Multiple Linear Regression	29131.229747369	14796.3773189224	0.82149598421745
Ridge Regression	25582.5523040611	16788.2636963242	0.862663539049991
Lasso Regression	27042.3191488428	16054.4115267468	0.846543256348492
Decision Tree Regressor	30849.5631233761	21259.0908419937	0.800291685362668
Random Forest Regressor	22382.4285055133	14705.2394214026	0.894873423360399
Gradient Boosting Model	14450.2854430239	9770.2694220187	0.956182160291553
XGBoost Model	16731.2760820432	10967.0328070367	0.941256983864958

Table 1: Performance of the Different Regression Model

Figure 20 shows a bar chart comparing the errors across the different models through their MAE and RMSE values.

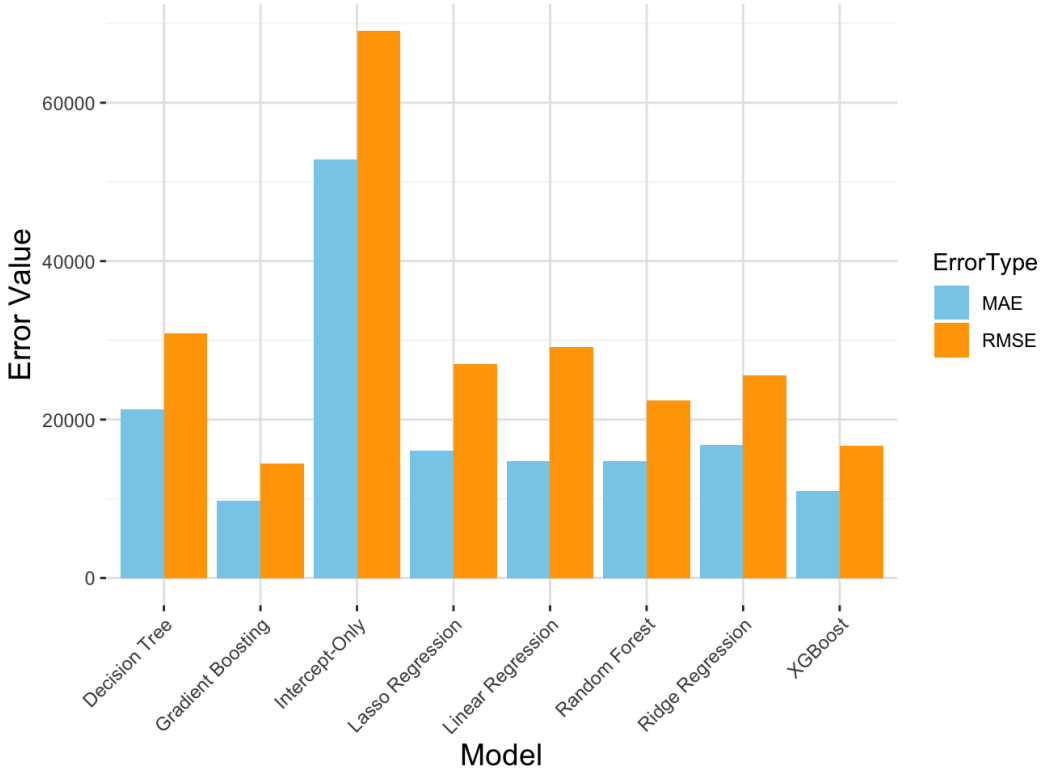


Figure 20: Comparing the Errors across the Different Machine Learning Models

6.2 Comparing the Predicted and Actual Values of the Best Model

To visually assess the performance of the Gradient Boosting model, a plot was created comparing the actual versus predicted house prices, shown in figure 21 below. As seen in the plot, the Gradient Boosting model demonstrates strong predictive power with minimal deviation from the actual sales prices, further confirming its effectiveness in this task.

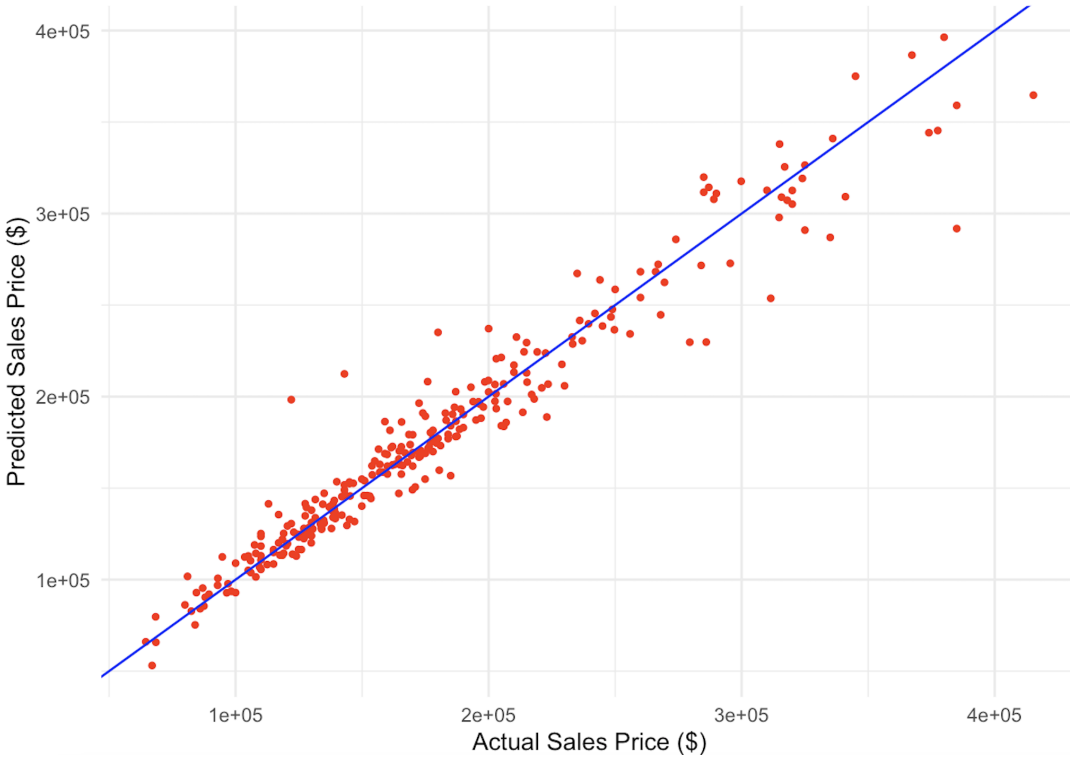


Figure 21: Plot of the Predicted vs Actual Values of the Best Model

6.3 Feature Importance of Best Model

Figure 22 below shows hows the relative importance of each feature, with a longer bar indicating a higher importance score. The importance score is a metric that quantifies how much each feature contributes to the overall predictive power of the model.

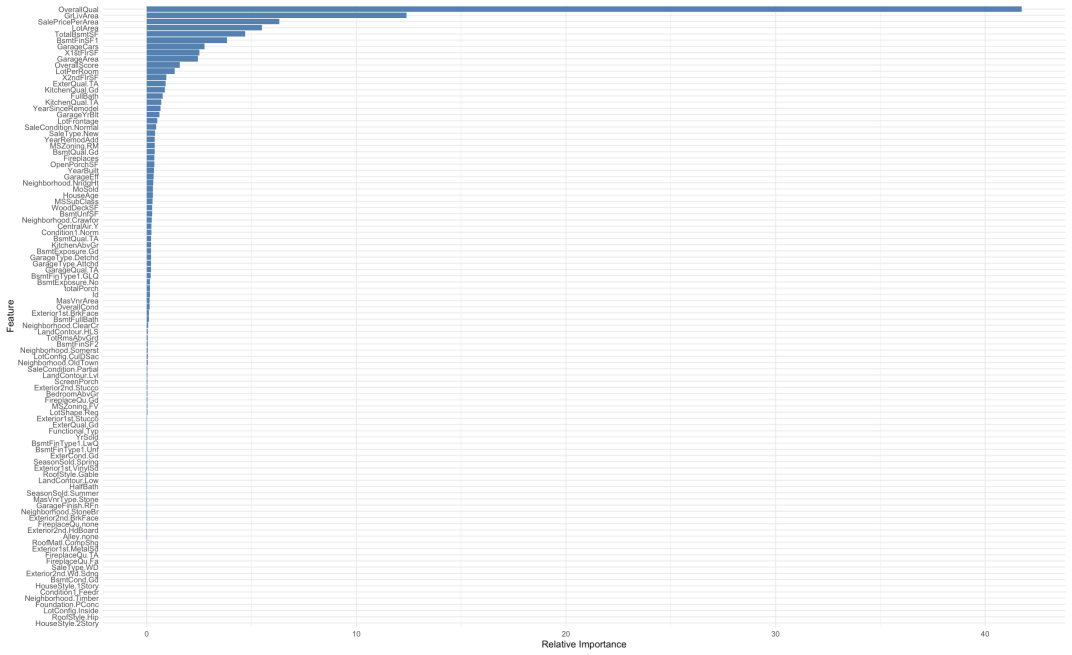


Figure 22: Plot of the Predicted vs Actual Values of the Best Model

Key Observations:

- **OverallQual** (Overall material and finish quality) and **GrLivArea** (Above grade (ground) living area square feet) are by far the primary drivers of the model's house price predictions. This makes intuitive sense, as the size and quality of a house are major determinants of its market value.
- The next tier of important features includes **LotArea** (total lot size in square feet). A larger lot is highly desirable as it provides more outdoor space, potential for future expansion, and a sense of privacy. This is followed closely by **TotalBsmtSF** (total square footage of the basement area), this feature is a direct measure of additional space. Even if the basement is unfinished, it provides significant potential for living space, storage, or a workshop, which makes it a major factor in a home's value. **BsmtFinSF1** (finished square feet of the first type of finished basement area) follows this which indicates that a finished basement with good quality finishings adds a great deal of value by expanding the functional living area of the home.
- **GarageCars** and **GarageArea** are highly important features because they directly measure the size and capacity of a garage. For buyers, a 2-car or 3-car garage is a major selling point, as it not only provides parking but also a place for storage, hobbies, or a workshop.
- After garage related features, the model identifies **1stFlrSF** (1st floor square footage) and **2ndFlrSF** (2nd floor square footage) as the next most important features, with 1stFlrSF being ranked as a more important feature. The first floor often contains the main living areas, such as the kitchen, living room, and dining area, which are central to a home's functionality and are a primary consideration for buyers. While it contributes to the overall living area, it is generally considered less important than the first floor. This is likely because the second floor is typically where bedrooms and guest bathrooms are located, which, while valuable, are not used as frequently as the common areas on the first floor.

7 Conclusion

In this project, several machine learning models were built, tuned and tested including Decision Tree, Random Forest , Gradient Boosting and XGBoost. Through careful hyperparameter tuning and cross-validation, it was observed that the Gradient Boosting model achieved the best performance with an R^2 of 95%, followed closely by XGBoost at 94%.

Typically the XGBoost is regarded as the most powerful model for predicting tabular data, but in this case has been slightly outperformed by the simpler Gradient Boosting model. However this might be the case if certain hyperparameters may not have been optimised well or if the characteristics of the datset may suit the Gradient Boosting model moreso than the XGBoost model. This highlights the importance of proper model selction and effective tuning of hyperparameters based on the characteristic of the dataset. Further exploration into the hyperparameter space and additional models may improve predictive accuracy, but the results demonstrate the effectiveness of boosting techniques in predictive modeling tasks

References

[1] Southekal, P. (2024) Understanding linear regression intercepts in plain language, DATAVERSITY. Available at: https://www.dataversity.net/understanding-linear-regression-intercepts-in-plain-language/ (Accessed: 19 April 2025).

[2] Comment et al. (2025) Linear regression in machine learning, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/ml-linear-regression/ (Accessed: 20 April 2025).

[3] GeeksforGeeks (2025) Ridge regression vs lasso regression, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/ridge-regression-vs-lasso-regression/ (Accessed: 20 April 2025).

[4] Decision tree algorithm, explained (no date) KDnuggets. Available at: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html (Accessed: 23 April 2025).

[5] Baladram, S. (2024) Random Forest, explained: A visual guide with code examples, Medium. Available at: https://medium.com/data-science/random-forest-explained-a-visual-guide-with-code-examples-9f736a6e1b3c#:~:text=A%

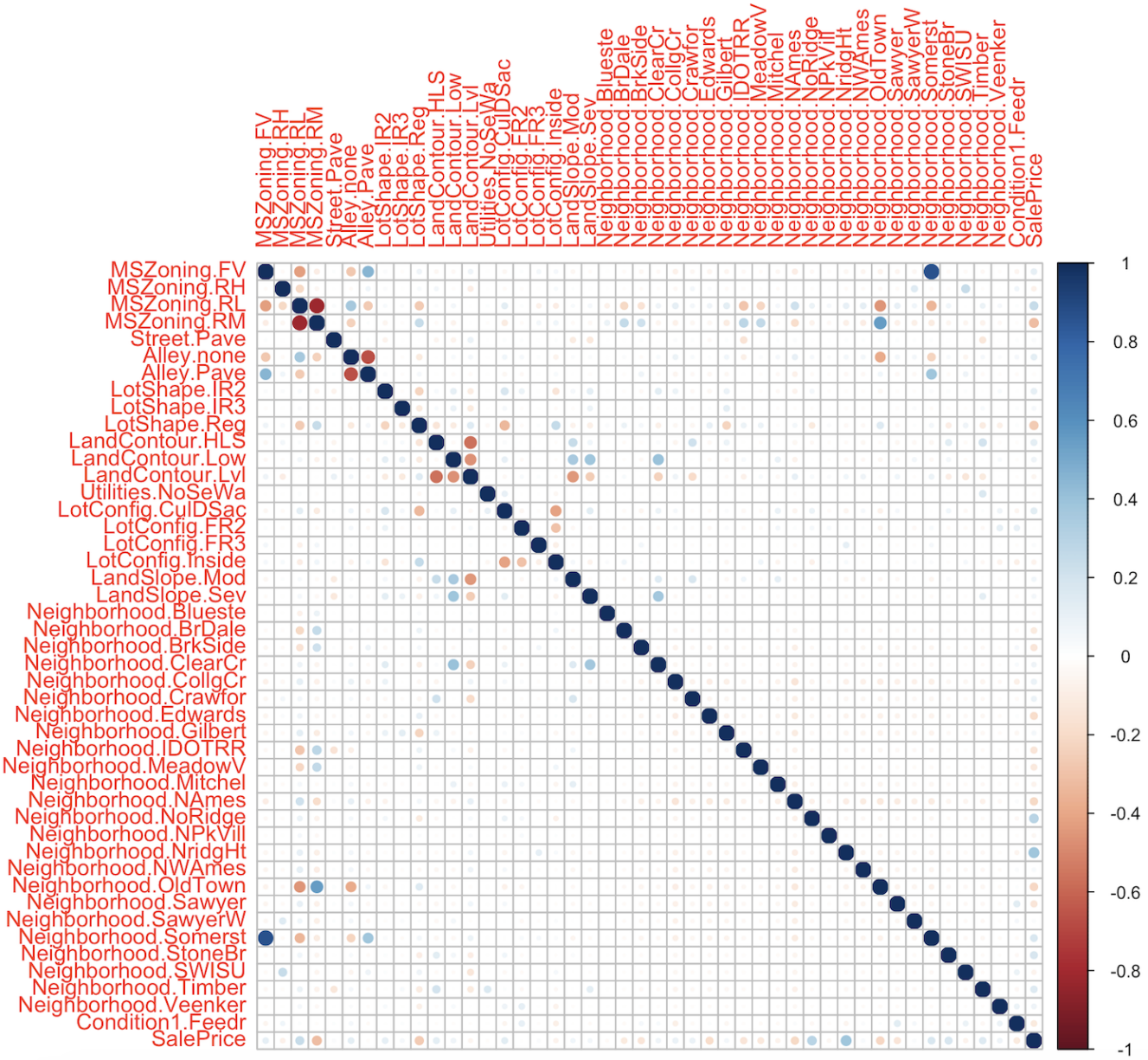
20Random%20Forest%20is%20an,making%20splits%20(feature%20randomization). (Accessed: 23 April 2025).

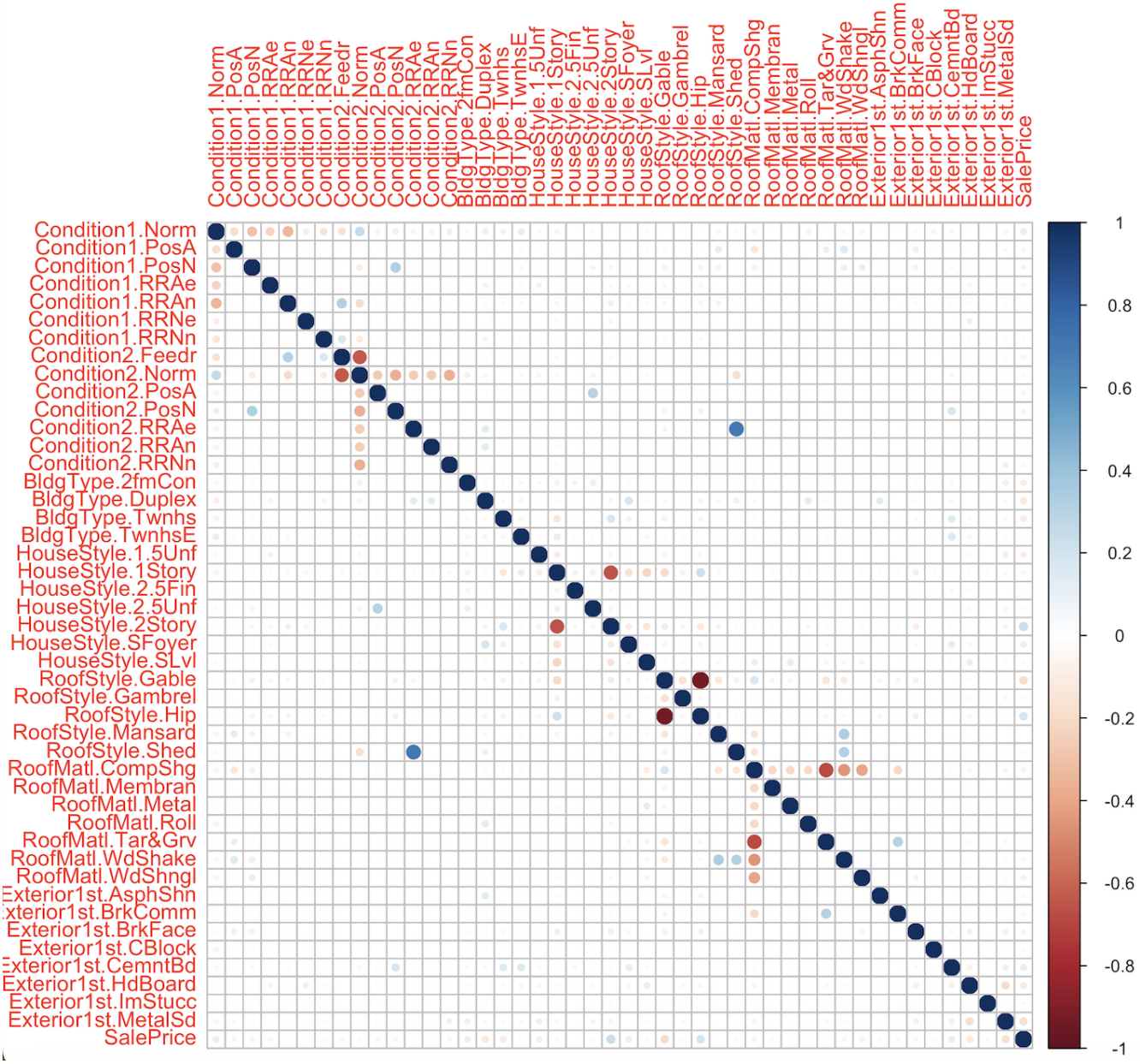
[6] Masui, T. (2024). *All you need to know about gradient boosting algorithm - Part 1: regression. Medium.* Available at: https://medium.com/data-science/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502 (Accessed: 23 April 2025).

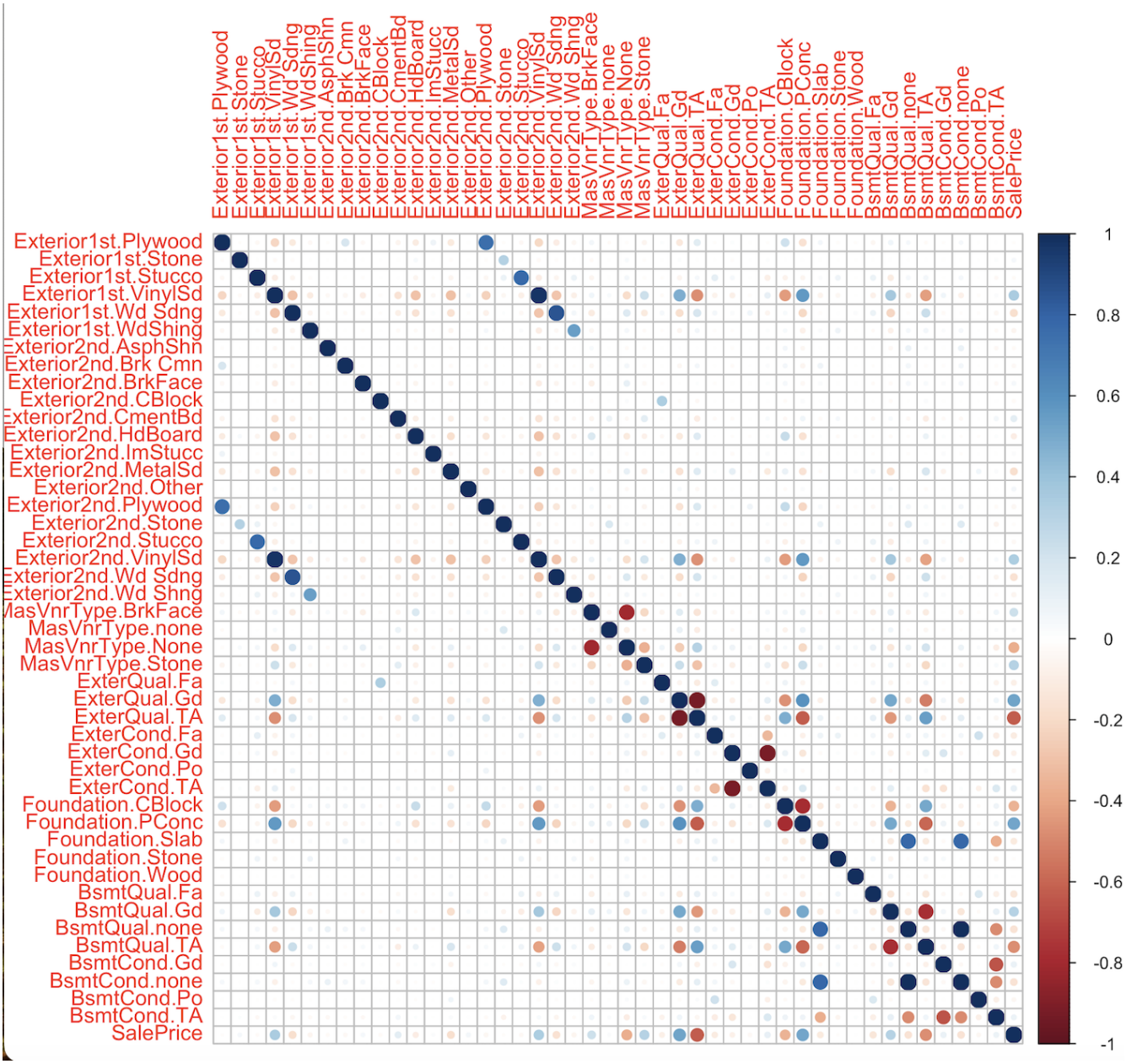
[7] Sonawane, P. (2023) XGBoost-how does this work, Medium. Available at: https://medium.com/@prathameshsonawane/xgboost-how-does-this-work-e1cae7c5b6cb (Accessed: 23 April 2025).

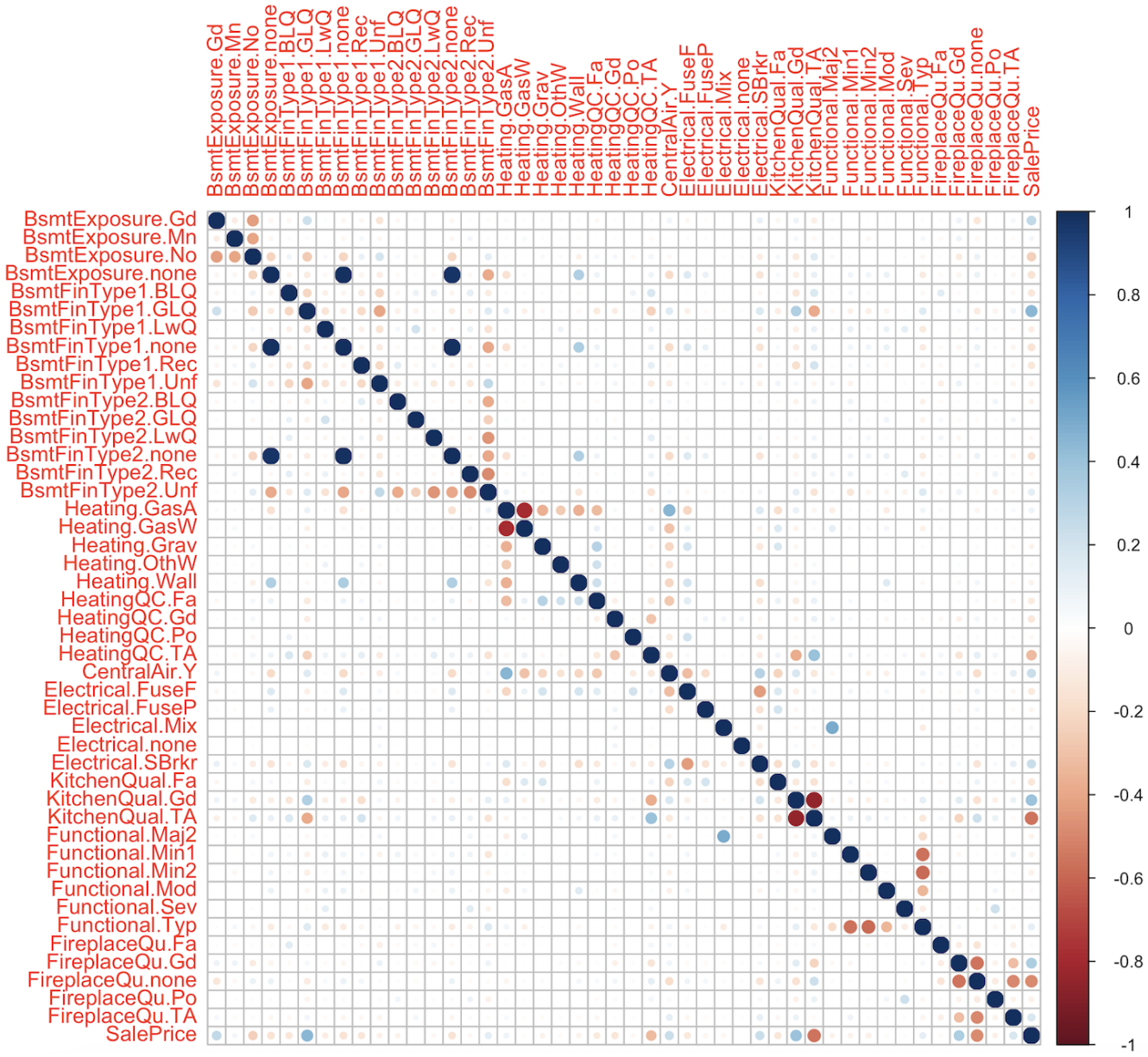
A Correlation Matrix of One-Hot Encoded Categorical Variables

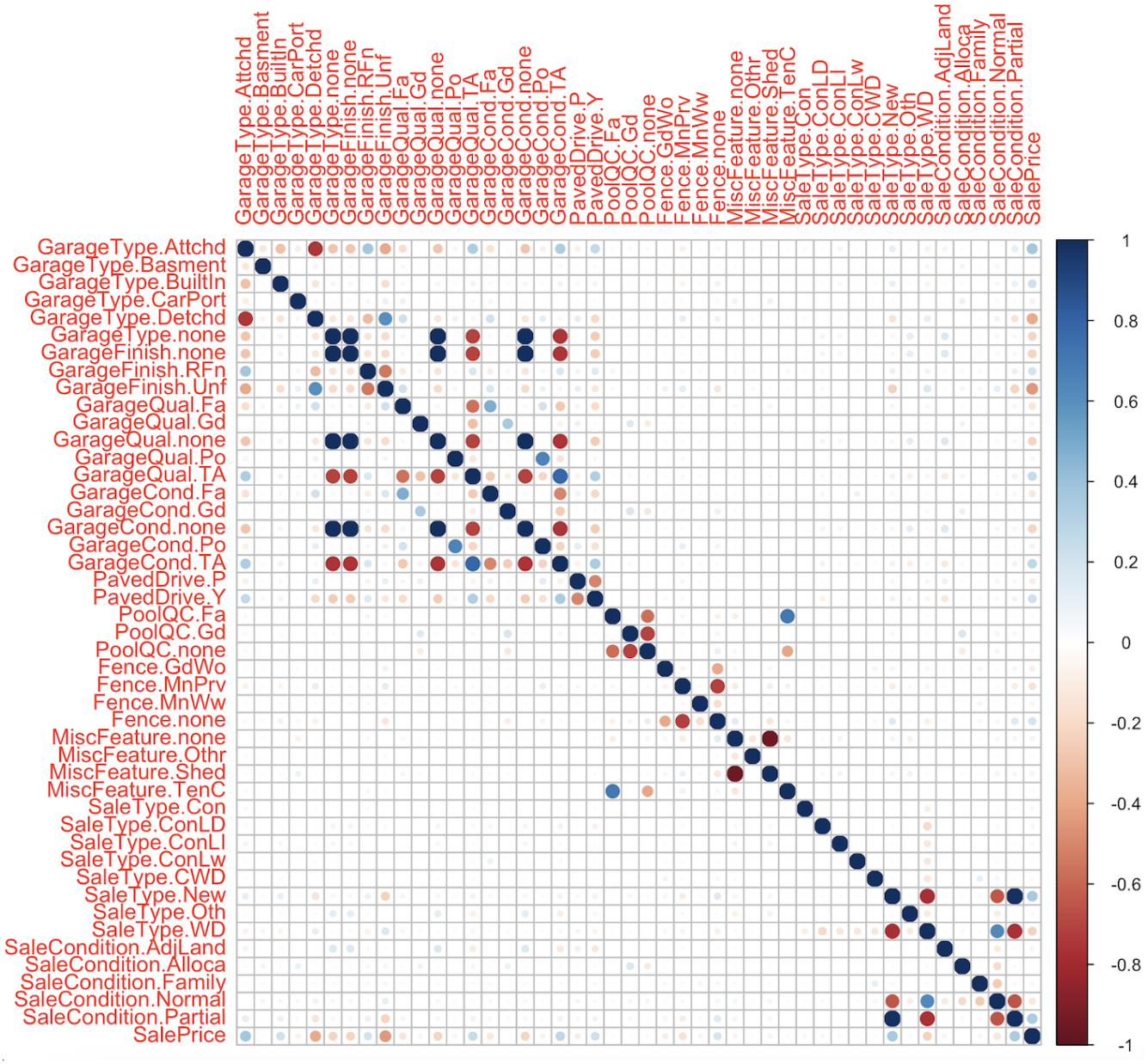
This section presents the correlation heatmaps derived from the one-hot encoded categorical variables. Due to the high dimensionality of the dataset following one-hot encoding, the correlation matrix was large and complex, making it unsuitable for direct inclusion in the main report. However they have been included below to visualise the key relationship between the encoded categorical features.




Condition1.Norm
Condition1.PosA
Condition1.PosN
Condition1.RRAe
Condition1.RRAn
Condition1.RRNe
Condition1.RRNn
Condition2.Feedr
Condition2.Norm
Condition2.PosA
Condition2.PosN
Condition2.RRAe
Condition2.RRAn
Condition2.RRNn
BldgType.2fmCon
BldgType.Duplex
BldgType.Twnhs
BldgType.TwnhsE
HouseStyle.1.5Unf
HouseStyle.1Story
HouseStyle.2.5Fin
HouseStyle.2.5Unf
HouseStyle.2Story
HouseStyle.SFoyer
HouseStyle.SLvl
RoofStyle.Gable
RoofStyle.Gambrel
RoofStyle.Hip
RoofStyle.Mansard
RoofStyle.Shed
RoofMatl.CompShg
RoofMatl.Membran
RoofMatl.Metal
RoofMatl.Roll
RoofMatl.Tar&Grv
RoofMatl.WdShake
RoofMatl.WdShngl
Exterior1st.AsphShn
Exterior1st.BrkComm
Exterior1st.BrkFace
Exterior1st.CBlock
Exterior1st.CemntBd
Exterior1st.HdBoard
Exterior1st.ImStucc
Exterior1st.MetalSd
SalePrice
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

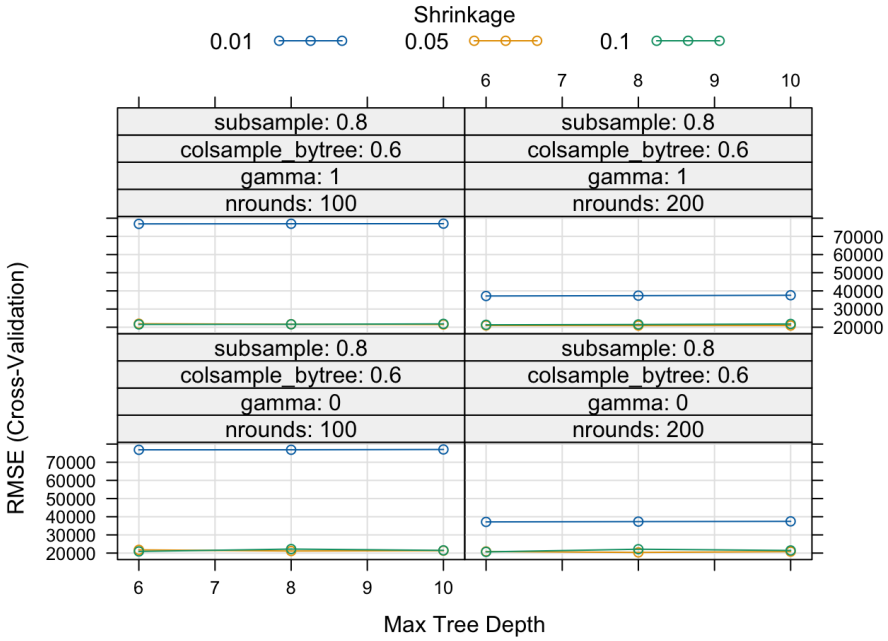

Exterior1st.Plywood
Exterior1st.Stone
Exterior1st.Stucco
Exterior1st.VinylSd
Exterior1st.Wd Sdng
Exterior1st.WdShing
Exterior2nd.AsphShn
Exterior2nd.Brk Cmn
Exterior2nd.BrkFace
Exterior2nd.CBlock
Exterior2nd.CmentBd
Exterior2nd.HdBoard
Exterior2nd.ImStucc
Exterior2nd.MetalSd
Exterior2nd.Other
Exterior2nd.Plywood
Exterior2nd.Stone
Exterior2nd.Stucco
Exterior2nd.VinylSd
Exterior2nd.Wd Sdng
Exterior2nd.Wd Shng
MasVnrType.BrkFace
MasVnrType.none
MasVnrType.None
MasVnrType.Stone
ExterQual.Fa
ExterQual.Gd
ExterQual.TA
ExterCond.Fa
ExterCond.Gd
ExterCond.Po
ExterCond.TA
Foundation.CBlock
Foundation.PConc
Foundation.Slab
Foundation.Stone
Foundation.Wood
BsmtQual.Fa
BsmtQual.Gd
BsmtQual.none
BsmtQual.TA
BsmtCond.Gd
BsmtCond.none
BsmtCond.Po
BsmtCond.TA
SalePrice
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

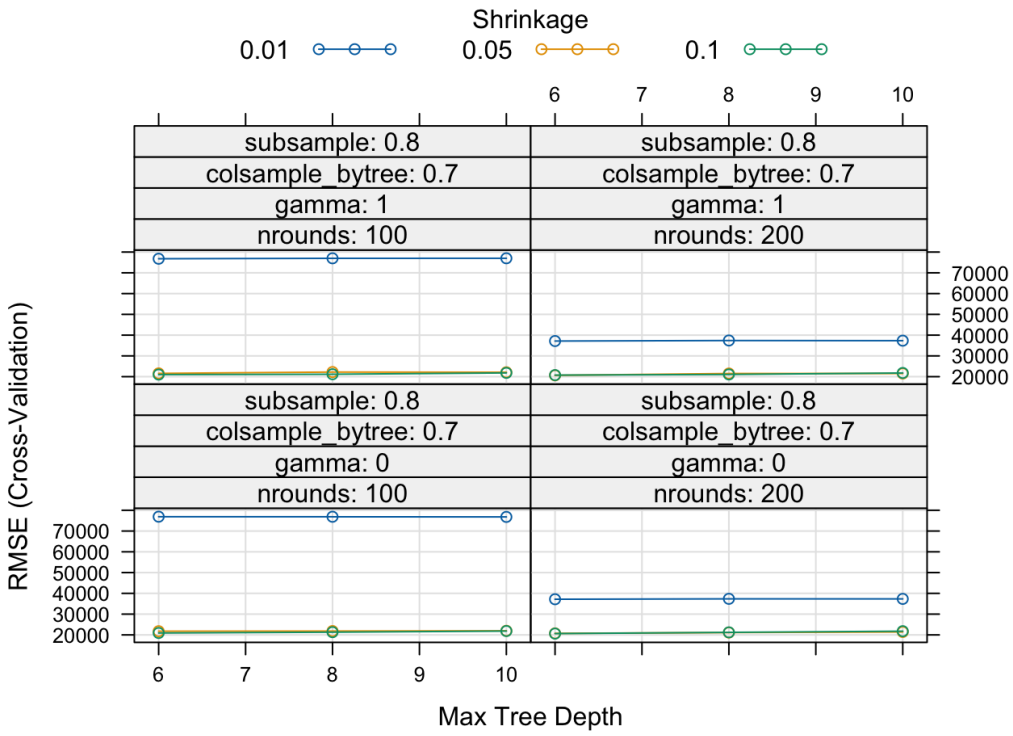

BsmtExposure.Gd
BsmtExposure.Mn
BsmtExposure.No
BsmtExposure.none
BsmtFinType1.BLQ
BsmtFinType1.GLQ
BsmtFinType1.LwQ
BsmtFinType1.none
BsmtFinType1.Rec
BsmtFinType1.Unf
BsmtFinType2.BLQ
BsmtFinType2.GLQ
BsmtFinType2.LwQ
BsmtFinType2.none
BsmtFinType2.Rec
BsmtFinType2.Unf
Heating.GasA
Heating.GasW
Heating.Grav
Heating.OthW
Heating.Wall
HeatingQC.Fa
HeatingQC.Gd
HeatingQC.Po
HeatingQC.TA
CentralAir.Y
Electrical.FuseF
Electrical.FuseP
Electrical.Mix
Electrical.none
Electrical.SBrkr
KitchenQual.Fa
KitchenQual.Gd
KitchenQual.TA
Functional.Maj2
Functional.Min1
Functional.Min2
Functional.Mod
Functional.Sev
Functional.Typ
FireplaceQu.Fa
FireplaceQu.Gd
FireplaceQu.none
FireplaceQu.Po
FireplaceQu.TA
SalePrice
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

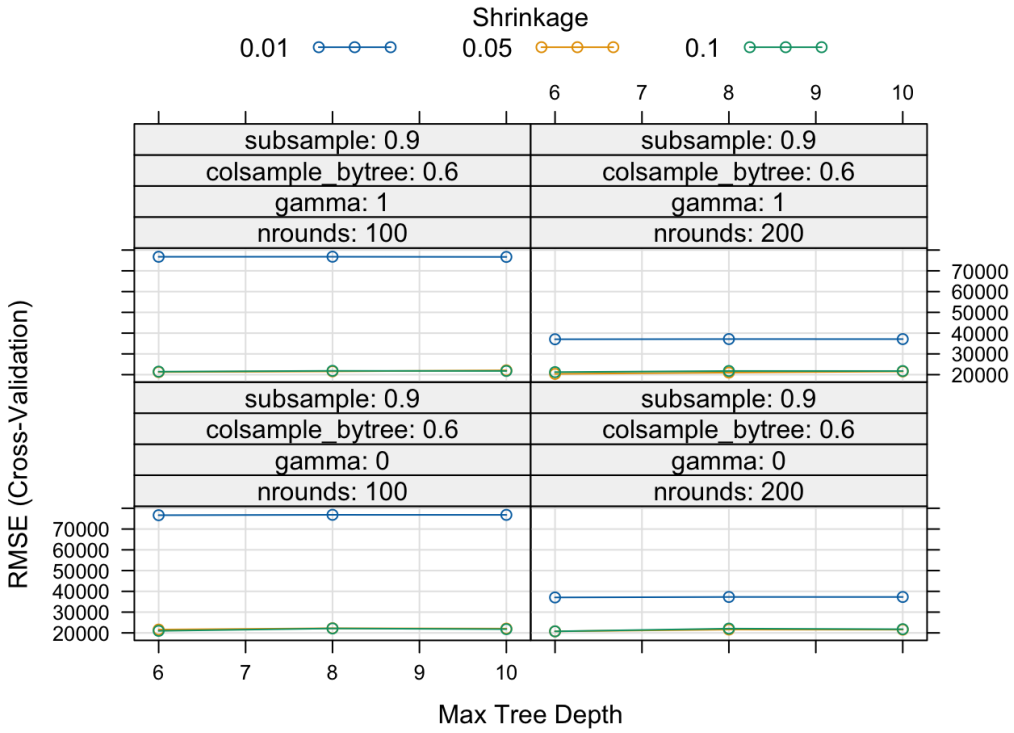

GarageType.Attchd
GarageType.Basment
GarageType.BuiltIn
GarageType.CarPort
GarageType.Detchd
GarageType.none
GarageFinish.none
GarageFinish.RFn
GarageFinish.Unf
GarageQual.Fa
GarageQual.Gd
GarageQual.none
GarageQual.Po
GarageQual.TA
GarageCond.Fa
GarageCond.Gd
GarageCond.none
GarageCond.Po
GarageCond.TA
PavedDrive.P
PavedDrive.Y
PoolQC.Fa
PoolQC.Gd
PoolQC.none
Fence.GdWo
Fence.MnPrv
Fence.MnWw
Fence.none
MiscFeature.none
MiscFeature.Othr
MiscFeature.Shed
MiscFeature.TenC
SaleType.Con
SaleType.ConLD
SaleType.ConLI
SaleType.ConLw
SaleType.CWD
SaleType.New
SaleType.Oth
SaleType.WD
SaleCondition.AdjLand
SaleCondition.Alloca
SaleCondition.Family
SaleCondition.Normal
SaleCondition.Partial
SalePrice
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

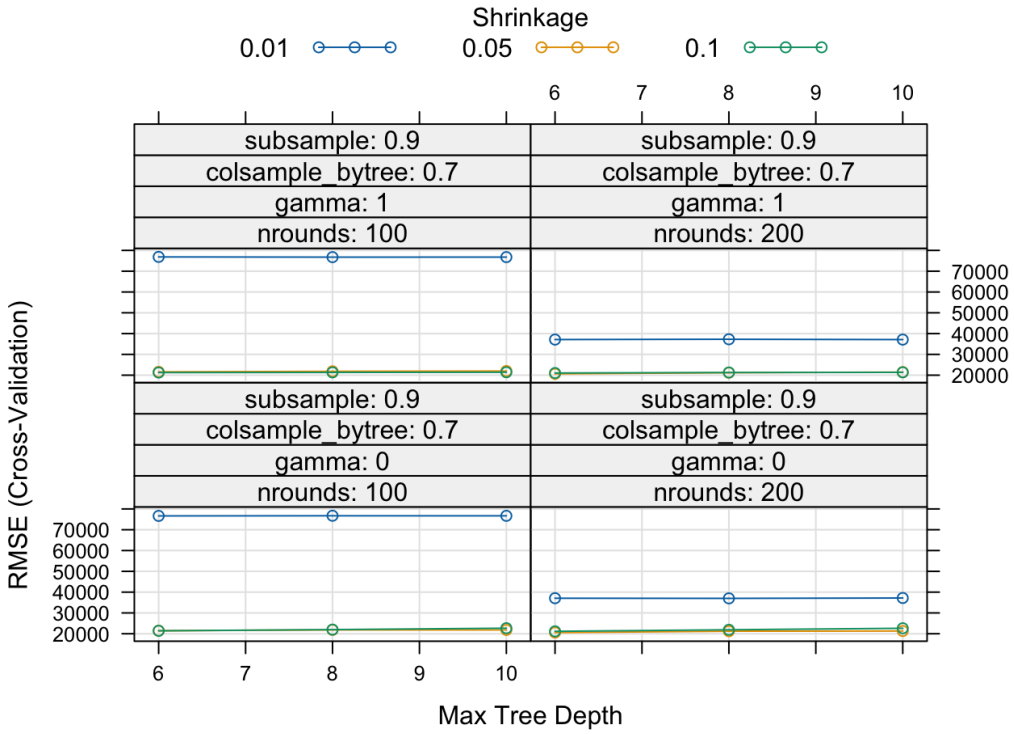
B XGBoost Hyperparameter Tuning Results

To optimise the performance of the XGBoost model, a comprehensive grid search was conducted over a defined range of hyperparameters using 5-fold cross-validation. The process evaluated combinations of key parameters such as the number of boosting rounds (nrounds), tree depth (max_depth), learning rate (eta), column sampling rate (colsample_bytree), and subsampling ratio (subsample). The plots presented in this appendix display how each parameter affects the cross-validation error.






Shrinkage
0.01
0.05
0.1
subsample: 0.9
colsample_bytree: 0.6
gamma: 1
nrounds: 100
subsample: 0.9
colsample_bytree: 0.6
gamma: 1
nrounds: 200
subsample: 0.9
colsample_bytree: 0.6
gamma: 0
nrounds: 100
subsample: 0.9
colsample_bytree: 0.6
gamma: 0
nrounds: 200
RMSE (Cross-Validation)
Max Tree Depth


Shrinkage
0.01
0.05
0.1
subsample: 0.9
colsample_bytree: 0.7
gamma: 1
nrounds: 100
subsample: 0.9
colsample_bytree: 0.7
gamma: 1
nrounds: 200
subsample: 0.9
colsample_bytree: 0.7
gamma: 0
nrounds: 100
subsample: 0.9
colsample_bytree: 0.7
gamma: 0
nrounds: 200
RMSE (Cross-Validation)
Max Tree Depth