Ames Housing Data Analysis: Property Features and Sale Prices

Bryan Marquez
Data Science 101
Stanford University

2025-02-28

Updated version of an assignment

The Study

This analysis utilizes housing data in Ames, Iowa from the Ames Assessor's Office, covering properties sold between 2006 and 2010. I plan on analyzing how varying conditions of features on the properties sold affect the overall quality and vice versa. The goal is to examine the relationship between property quality/conditions and sale price. By comparing these variables, we aim to identify marketable features versus those that negatively impact sale price. We can also determine materials and condition thresholds that change quality labels. For example, we could check if increasing garage area increases sale price, increases quality/condition by n-levels, or correlates with the amount of fireplaces a place has and also by how much. For comparing our continuous variables, linear regression and various diagnostic plots are used. For comparing nominal variables to discrete or continuous variables, we can use a log regression tree by turning a specific nominal variable into a binary response, which allows for multiple nominal comparisons to each other. This study will test the hypothesis that various property features, such as garage area, fireplaces, overall condition, etc… significantly affect sale price. The analysis will primarily utilize linear regression and log regression trees for continuous and categorical (which we can also view as discrete) variables respectively, with confidence intervals and ANOVA tests used to measure model uncertainty.

The Data

The data consists of 80 variables– 23 nominal, 23 ordinal, 14 discrete, and 20 continuous– given by the number of columns, and 2000 observations for each of the 80 variables, given by the number of rows in the data set. The data is split into a training set (1400) and a testing set (600) testing, a 70-30. Then, models are created with sale price (continuous) as our main response on the training set and use the testing set to validate the models. The overall condition (ordinal) and quality (ordinal) will be our main predictors; overall condition and quality are given by a score from 1, the poorest, to 10, the best. When sorting the data, missing values (NAs) are imputed for discrete and continuous variables by replacing them with the mean after outliers were removed. Additionally, low variance features are removed, meaning that a majority of the data fell into one bucket. Low variance features also caused a problem during splitting because some levels were so rare that when I split the data, both levels were not present, so I removed the levels that are most probable for this to happen. However, as a limitation, this is still possible (if the seed is not set) because an entire level can still be sampled randomly into one of our sets. Furthermore, I analyze specific categorical variables such as fireplace number (discrete), basement quality (ordinal) given by levels "Poor", "Fair", "Typical/Average", "Good", "Excellent" in ascending order, and garage area (continuous) given by square feet to provide a case example of sub-analysis. Moreover, few houses in the data have more than 1 fireplace, so I convert this data into the amount of houses with fireplaces and those without. Using the multi-correlation plot, redundant variables are noticeable as they were highly correlated (>.8), and the redundant variables were removed. For example, the garage car variable was removed since the continuous version, garage area, provided more information in the analysis. To analyze the spread and other behaviors of my predictors, I make a table of their summary statistics and plot their density graphs, using integrated histograms and ggplot. In the data cleaning process, some categorical variables (factors) rarely occurred, and these cause issues during data splitting and model evaluation, so levels with >10 occurrences were removed in a factor for loop. This is to prevent overfitting to rare categories, which may not provide meaningful information. Additionally, this ensures that both the training and testing sets contain consistent factor levels, avoiding potential problems where certain levels may only appear in one part of the data. By removing these infrequent levels, the model is able to generalize better and provide more reliable predictions.

The Models

In order to capture the diversity of the data, I included as many features as possible without the limitations listed in other sections. For the modeling process, the data was first split 70-30 into training and testing sets. Using the training set, a linear model predicted the sale price of the houses based on all available features. The summary of the model's performance on the training set shows each variable's contribution to the prediction of sale price. To evaluate the model, we run the predict function on the test set. Because housing sale price is in hundreds of thousands, the root mean squared error (RMSE) was calculated to assess the accuracy of these predictions. Confidence intervals also provided insight into the uncertainty surrounding the impact of each individual predictor. In addition to the linear model, I implemented a bagging model using the random forest algorithm to improve prediction accuracy. Using the same training data set as the linear model, the random forest model utilizes 50 trees and uses the sqrt of the total features number to consider each split, hopefully reducing the variance of the prediction. To evaluate both models, predictions based on the models were generated on the test set, and a scatter plot visualized the predicted sale prices against the actual sale prices, with a reference line indicating accurate predictions. Because we deal with larger sale price numbers, the root mean squared error (RMSE) was calculated along with the error percentage from the actual mean.

Finally, I coded ANOVA tests for each categorical variable in our dataset to determine their potential impact on the sale price. The list categoricals combines ordinal and nominal variables, representing discretized features of the housing data, such as garage type, sale condition, or heating. The code iterates through each variable, constructing a formula of the form SalePrice ~ variable, and runs an ANOVA test to assess the statistical significance ($p<.05$) of the relationship between each categorical feature and the target variable, SalePrice. The results of each ANOVA test are stored in a list AL, where the variable names act as keys and the ANOVA summaries as values. This helps in identifying which categorical variables have a statistically significant influence on the sale price, which is crucial for deciding which features to include in the final predictive model.

The Results

With both the correlation table in the beginning of the code the the ANOVA loop in the end, we get statistical significance results for every continuous and categorical feature respectively. As an example of data provided, consider the correlation table first: the basement quality (materials) and overall quality are positively correlated (.6), meaning the basement had a significant weight on overall quality. In the analysis of variance (ANOVA) conducted on the Ames housing dataset, the F-value in this case explains impact of categorical predictors on the sale price of homes. The larger F-value of 336.5 suggests that the group means are significantly different, indicating a significant portion of the variability in sale prices; this is also supported by the associated p-value (< 2e-16).

Moreover, the associated bar plots of the overall quality and condition are centered around the center of 5 or 6; however, conditions mostly fell into 5 whereas quality was more gaussian around 5.5. The density plot of the sale price also behaved normal around 175k, and with the summary function, we get all main statistical data from every feature. For example, the majority of the Heating units fall under GasA (1881) with the second most as GasW (19), which is why the ANOVA resulted in a low F value (0.192) and is not statistically significant (0.661).

The linear regression model gave a Multiple R-squared value of 0.9394, suggesting that approximately 93.94% of the variance in the dependent variable (SalePrice) is explained by the model. The Adjusted R-squared value of 0.927 suggests the removal of redundant variables amplified our model. Also, the associated p-value (< 2.2e-16) indicates that the model is statistically significant. The RMSE for this model was 17,318.65, which represents the average deviation of the predicted sale prices from the actual sale prices. When compared to the average sale price in the data, this RMSE corresponds to an error percentage of approximately 10%. On the other hand, the bagging model produced an RSME of 19,664.56, which represents an error percentage of 11%. However, this may be due to the low data given for each split in the randomForest algorithm described in the limitations section. All in all, both models provided good indicators for sale price with the linear model working better for more generalized or single category houses.

Limitations

Certain variables of the data contained far too many NAs which may have amplified its already low variance. For example, most houses did not contain fireplaces; as a result, the firehouse condition feature contained mostly zeroes (the mode), amplifying its low variance.

Furthermore, the relatively small size of the data set adversely affected the performance of the bagging model; complex models like bagging require substantial data to generalize effectively. The individual decision trees can become highly sensitive to the specific samples drawn, leading to increased variance and the potential for overfitting. Consequently, the bagging model may struggle to learn the underlying patterns, resulting in less stable predictions when evaluated on unseen data. With a small data set, I removed only the outliers for the SalePrice feature. With sufficient data, lopping through every feature and removing outliers accordingly can be done as shown right below. However, we may want to use data sets of this size for varying cities, so the model is up to discretion.

Appendix:

```r
par(mfrow = c(1, 1))

require(dplyr)
```

```
## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
require(randomForest)
```

```
## Loading required package: randomForest

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
require(ggplot2)
```

```
## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.3.3

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin
```

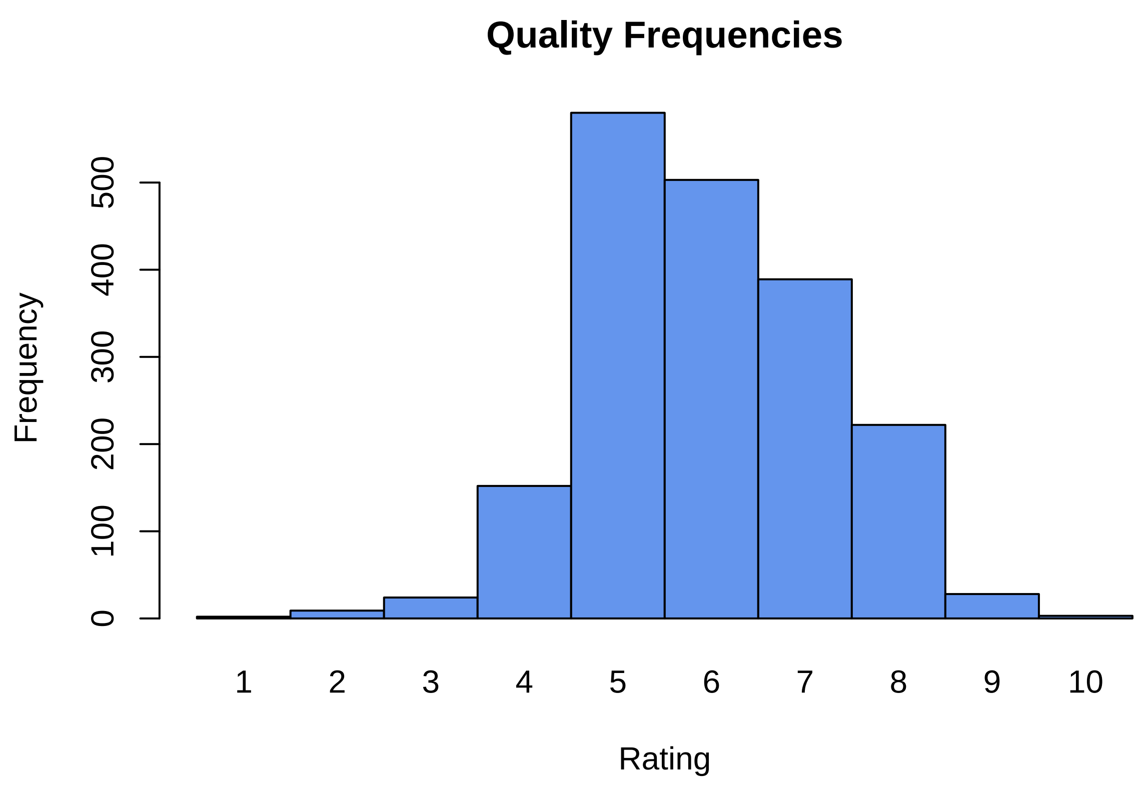
```r
set.seed(6)
Ames_data = read.csv("ames2000_NAfix.csv")

#Remove the features from selection down below
Ames_data = Ames_data %>%select(-Utilities, -BsmtFin.SF.2, -Bsmt.Unf.SF, -Garage.Yr.Blt, -Garage.Cars,

#Setting up the levels for the ordinal RVs:
state_levels = c("None","Po","Fa","TA","Gd","Ex")
fence = c("None","MnWw","GdWo","MnPrv","GdPrv")
lot_shapes = c("IR3","IR2","IR1","Reg")
exposure_levels = c("None","No", "Mn","Av","Gd")
basement_fin_levels = c("None","Unf","LwQ","Rec","BLQ","ALQ","GLQ")
electrical_levels = c("Mix", "FuseP", "FuseF", "FuseA", "SBrkr")
functionality_levels = c("Sal","Sev","Maj2","Maj1","Mod","Min2","Min1","Typ")
garage_finish = c("None","Unf","RFn","Fin")
paved_levels = c("N","P","Y")
Ames_data$Fireplace.Qu = as.numeric(factor(Ames_data$Fireplace.Qu,state_levels)) - 1
Ames_data$Exter.Qual = as.numeric(factor(Ames_data$Exter.Qual,state_levels)) - 1
Ames_data$Exter.Cond = as.numeric(factor(Ames_data$Exter.Cond,state_levels)) - 1
Ames_data$Lot.Shape = as.numeric(factor(Ames_data$Lot.Shape, lot_shapes))
Ames_data$Bsmt.Qual = as.numeric(factor(Ames_data$Bsmt.Qual, state_levels)) - 1
Ames_data$Bsmt.Cond = as.numeric(factor(Ames_data$Bsmt.Cond, state_levels)) - 1
```

```
Ames_data$Garage.Cond = as.numeric(factor(Ames_data$Garage.Cond, state_levels)) - 1
Ames_data$Bsmt.Exposure = as.numeric(factor(Ames_data$Bsmt.Exposure, exposure_levels)) - 1
Ames_data$BsmtFin.Type.1 = as.numeric(factor(Ames_data$BsmtFin.Type.1,basement_fin_levels)) - 1
Ames_data$BsmtFin.Type.2 = as.numeric(factor(Ames_data$BsmtFin.Type.2,basement_fin_levels)) - 1
Ames_data$Heating.QC = as.numeric(factor(Ames_data$Heating.QC,state_levels))
Ames_data$Electrical = as.numeric(factor(Ames_data$Electrical, electrical_levels))
Ames_data$Kitchen.Qual = as.numeric(factor(Ames_data$Kitchen.Qual,state_levels)) - 1
Ames_data$Functional = as.numeric(factor(Ames_data$Functional, functionality_levels))
Ames_data$Garage.Finish = as.numeric(factor(Ames_data$Garage.Finish, garage_finish)) - 1
Ames_data$Garage.Qual = as.numeric(factor(Ames_data$Garage.Qual,state_levels)) - 1
Ames_data$Paved.Drive = as.numeric(factor(Ames_data$Paved.Drive, paved_levels))
Ames_data$Fence = as.numeric(factor(Ames_data$Fence, fence)) - 1
Ames_data$Fireplaces = as.numeric(Ames_data$Fireplaces > 0)

#Grouping the variables
contRVs = c("Lot.Frontage", "Lot.Area", "BsmtFin.SF.1", "X1st.Flr.SF" ,"X2nd.Flr.SF" , "Gr.Liv.Area", "
ordinals = c("Fireplace.Qu", "Exter.Qual", "Exter.Cond", "Lot.Shape", "Bsmt.Qual", "Bsmt.Cond", "Bsmt.E
quantitatives = c(ordinals, contRVs)
nominals = setdiff(names(Ames_data), quantitatives)

#Cleaning NAs after removing outliers
Ames_data = Ames_data %>%mutate(across(all_of(quantitatives), ~ as.numeric(.)))
```

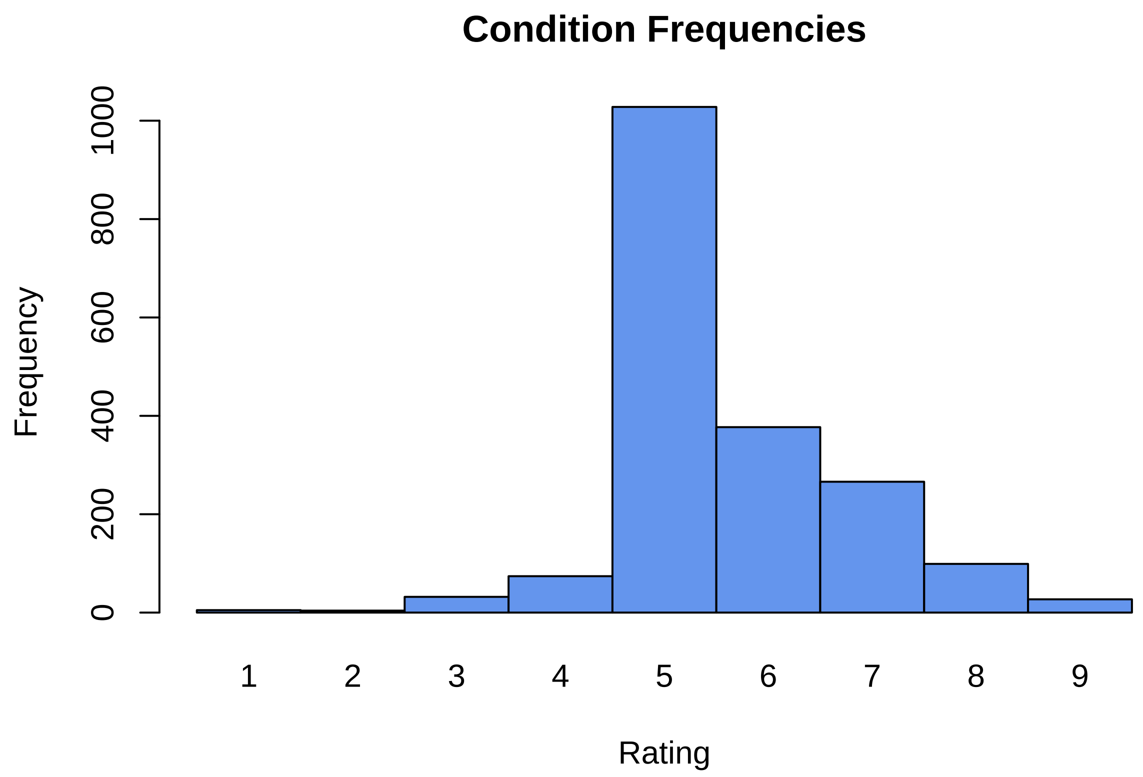
```
## Warning: There were 6 warnings in ‘mutate()‘.
## The first warning was:
## i In argument: ‘across(all_of(quantitatives), ~as.numeric(.))‘.
## Caused by warning:
## ! NAs introduced by coercion
## i Run ‘dplyr::last_dplyr_warnings()‘ to see the 5 remaining warnings.
```

```
#IQR:
IQR = quantile(Ames_data$SalePrice, 0.75) - quantile(Ames_data$SalePrice, 0.25)

#Define the lower and upper bounds for outliers
lower_bound = quantile(Ames_data$SalePrice, 0.25) - 1.5 * IQR
upper_bound = quantile(Ames_data$SalePrice, 0.75) + 1.5 * IQR

#Remove outliers based on sale price and replace NAs
Ames_data = Ames_data[lower_bound <= Ames_data$SalePrice & Ames_data$SalePrice <= upper_bound, ]
Ames_data = Ames_data %>%mutate(across(any_of(quantitatives), ~ ifelse(is.na(.), mean(., na.rm = TRUE),
Ames_data = Ames_data %>%mutate(across(all_of(ordinals), ~ round(.)))
Ames_data = Ames_data %>%mutate(across(any_of(nominals), ~ ifelse(is.na(.), mode(., na.rm = TRUE),.)))


#With a larger data set, remove outliers from more features with this
#numeric_cols = names(Ames_data)[sapply(Ames_data, is.numeric)]  # Select numeric columns
#for (col in numeric_cols) {
  # Calculate IQR for the current column
#  IQR_value = IQR(Ames_data[[col]], na.rm = TRUE)
  # Calculate the lower and upper bounds for outliers
#  lower_bound = quantile(Ames_data[[col]], 0.25, na.rm = TRUE) - 1.5 * IQR_value
#  upper_bound = quantile(Ames_data[[col]], 0.75, na.rm = TRUE) + 1.5 * IQR_value
  # Remove rows that contain outliers in this column
```

```r
#  Ames_data = Ames_data[Ames_data[[col]] >= lower_bound & Ames_data[[col]] <= upper_bound, ]}


#Double check for NAs
NAs = sapply(Ames_data, function(x) any(is.na(x)))

#Correlation table
corrs = cor(Ames_data%>%select_if(is.numeric))

#Factor the nominals and ordinals to prepare data for feature graphing and the models
Ames_data = Ames_data %>%mutate(across(all_of(c(nominals, ordinals)), as.factor))

#Plotting Some features
qualcount = table(Ames_data$Overall.Qual) #Roughly Gauss
condcount = table(Ames_data$Overall.Cond) #Indicates low variance and left skew.

bpqual = barplot(qualcount, space = 0, main = "Quality Frequencies", xlab = "Rating", ylab = "Frequency
```

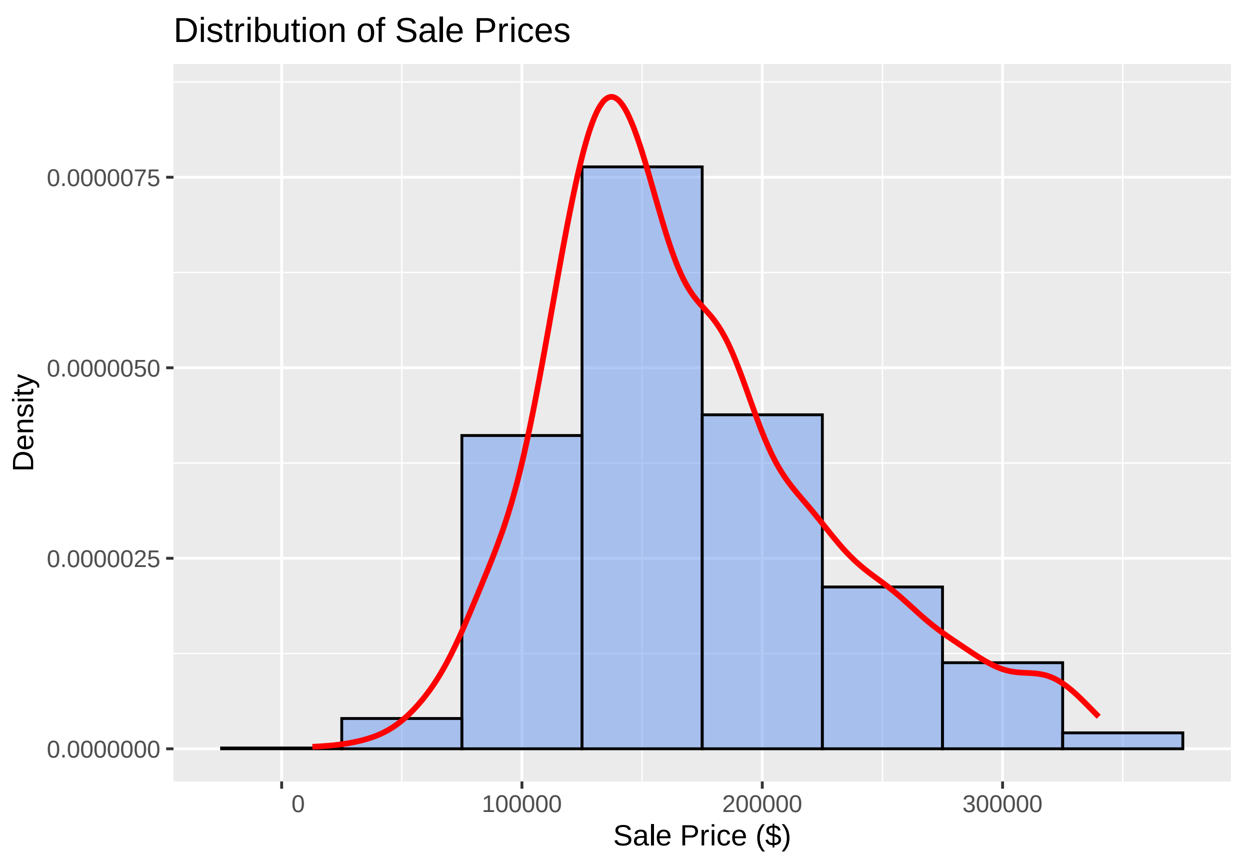


```r
bpcond = barplot(condcount, space = 0, main = "Condition Frequencies", xlab = "Rating", ylab = "Frequen
```



```
#ggplot(Ames_data, aes(x = SalePrice)) + geom_histogram(aes(y = ..density..), binwidth = 50000, fill =
#  labs(title = "Distribution of Sale Prices",
#       x = "Sale Price ($)",
#       y = "Density") +
#    theme_minimal()                          ### Simpler plot

ggplot(Ames_data, aes(x = SalePrice)) + geom_histogram(aes(y = ..density..), binwidth = 50000, fill = "
  scale_y_continuous(labels = function(x) format(x, scientific = FALSE))+  # Prevent scientific notatio
  scale_x_continuous(labels = function(x) format(x, scientific = FALSE))
```

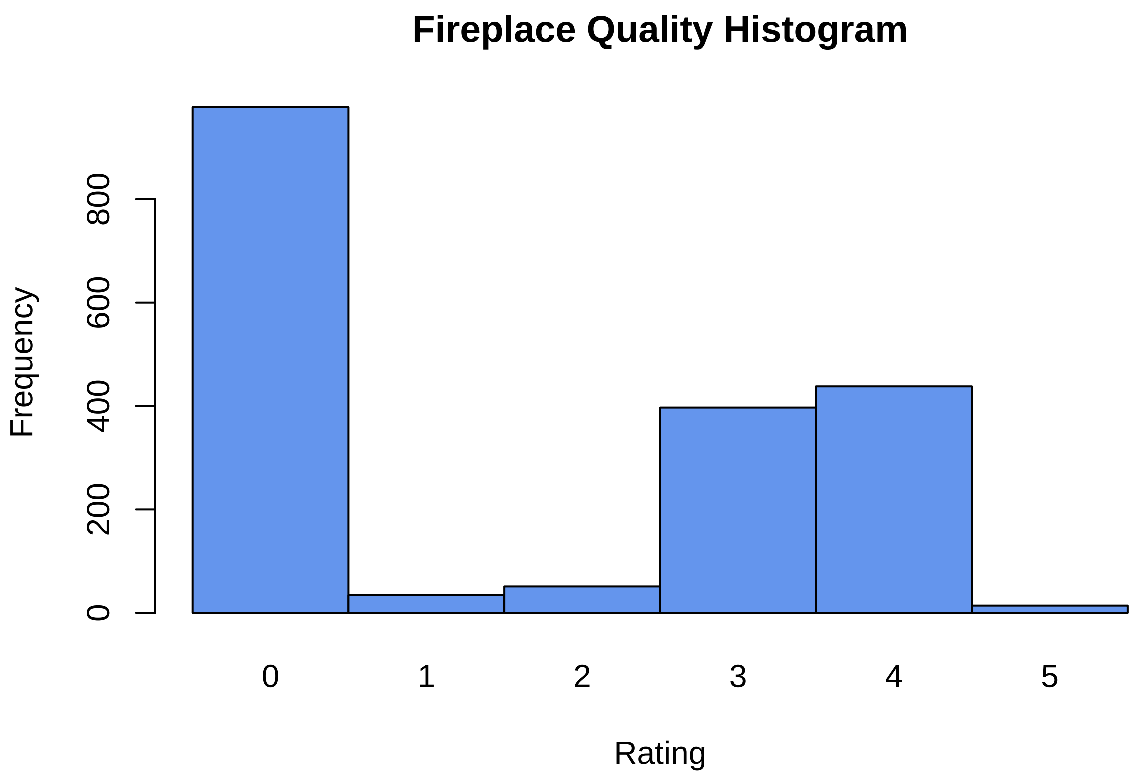
```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```r
  theme_minimal()              ### Sale Price Density plot
```

```
## List of 136
##  $ line                            :List of 6
##   ..$ colour       : chr "black"
##   ..$ linewidth    : num 0.5
##   ..$ linetype     : num 1
##   ..$ lineend      : chr "butt"
##   ..$ arrow        : logi FALSE
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_line" "element"
##  $ rect                            :List of 5
##   ..$ fill         : chr "white"
##   ..$ colour       : chr "black"
##   ..$ linewidth    : num 0.5
##   ..$ linetype     : num 1
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_rect" "element"
##  $ text                            :List of 11
##   ..$ family       : chr ""
##   ..$ face         : chr "plain"
##   ..$ colour       : chr "black"
##   ..$ size         : num 11
##   ..$ hjust        : num 0.5
##   ..$ vjust        : num 0.5
##   ..$ angle        : num 0
##   ..$ lineheight   : num 0.9
##   ..$ margin       : 'margin' num [1:4] 0points 0points 0points 0points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : logi FALSE
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ title                           : NULL
##  $ aspect.ratio                    : NULL
##  $ axis.title                      : NULL
##  $ axis.title.x                    :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : NULL
##   ..$ vjust        : num 1
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 2.75points 0points 0points 0points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.title.x.top                :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : NULL
##   ..$ vjust        : num 0
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 0points 0points 2.75points 0points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.title.x.bottom             : NULL
##  $ axis.title.y                    :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : NULL
##   ..$ vjust        : num 1
##   ..$ angle        : num 90
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 0points 2.75points 0points 0points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.title.y.left               : NULL
##  $ axis.title.y.right              :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : NULL
##   ..$ vjust        : num 1
##   ..$ angle        : num -90
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 0points 0points 0points 2.75points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.text                       :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : chr "grey30"
##   ..$ size         : 'rel' num 0.8
##   ..$ hjust        : NULL
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : NULL
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.text.x                     :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : NULL
##   ..$ vjust        : num 1
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 2.2points 0points 0points 0points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.text.x.top                 :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : NULL
##   ..$ vjust        : num 0
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 0points 0points 2.2points 0points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.text.x.bottom              : NULL
##  $ axis.text.y                     :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : num 1
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 0points 2.2points 0points 0points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.text.y.left                : NULL
##  $ axis.text.y.right               :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : num 0
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 0points 0points 0points 2.2points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.text.theta                 : NULL
##  $ axis.text.r                     :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : num 0.5
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : 'margin' num [1:4] 0points 2.2points 0points 2.2points
##   .. ..- attr(*, "unit")= int 8
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.ticks                      : list()
##   ..- attr(*, "class")= chr [1:2] "element_blank" "element"
##  $ axis.ticks.x                    : NULL
##  $ axis.ticks.x.top                : NULL
##  $ axis.ticks.x.bottom             : NULL
##  $ axis.ticks.y                    : NULL
##  $ axis.ticks.y.left               : NULL
##  $ axis.ticks.y.right              : NULL
##  $ axis.ticks.theta                : NULL
##  $ axis.ticks.r                    : NULL
##  $ axis.minor.ticks.x.top          : NULL
##  $ axis.minor.ticks.x.bottom       : NULL
##  $ axis.minor.ticks.y.left         : NULL
##  $ axis.minor.ticks.y.right        : NULL
##  $ axis.minor.ticks.theta          : NULL
##  $ axis.minor.ticks.r              : NULL
##  $ axis.ticks.length               : 'simpleUnit' num 2.75points
##   ..- attr(*, "unit")= int 8
##  $ axis.ticks.length.x             : NULL
##  $ axis.ticks.length.x.top         : NULL
##  $ axis.ticks.length.x.bottom      : NULL
##  $ axis.ticks.length.y             : NULL
##  $ axis.ticks.length.y.left        : NULL
##  $ axis.ticks.length.y.right       : NULL
##  $ axis.ticks.length.theta         : NULL
##  $ axis.ticks.length.r             : NULL
##  $ axis.minor.ticks.length         : 'rel' num 0.75
##  $ axis.minor.ticks.length.x       : NULL
##  $ axis.minor.ticks.length.x.top   : NULL
##  $ axis.minor.ticks.length.x.bottom: NULL
##  $ axis.minor.ticks.length.y       : NULL
##  $ axis.minor.ticks.length.y.left  : NULL
##  $ axis.minor.ticks.length.y.right : NULL
##  $ axis.minor.ticks.length.theta   : NULL
##  $ axis.minor.ticks.length.r       : NULL
##  $ axis.line                       : list()
##   ..- attr(*, "class")= chr [1:2] "element_blank" "element"
##  $ axis.line.x                     : NULL
##  $ axis.line.x.top                 : NULL
##  $ axis.line.x.bottom              : NULL
##  $ axis.line.y                     : NULL
##  $ axis.line.y.left                : NULL
##  $ axis.line.y.right               : NULL
##  $ axis.line.theta                 : NULL
##  $ axis.line.r                     : NULL
##  $ legend.background               : list()
##   ..- attr(*, "class")= chr [1:2] "element_blank" "element"
##  $ legend.margin                   : 'margin' num [1:4] 5.5points 5.5points 5.5points 5.5points
##   ..- attr(*, "unit")= int 8
##  $ legend.spacing                  : 'simpleUnit' num 11points
##   ..- attr(*, "unit")= int 8
##  $ legend.spacing.x                : NULL
##  $ legend.spacing.y                : NULL
##  $ legend.key                      : list()
##   ..- attr(*, "class")= chr [1:2] "element_blank" "element"
##  $ legend.key.size                 : 'simpleUnit' num 1.2lines
##   ..- attr(*, "unit")= int 3
##  $ legend.key.height               : NULL
##  $ legend.key.width                : NULL
##  $ legend.key.spacing              : 'simpleUnit' num 5.5points
##   ..- attr(*, "unit")= int 8
##  $ legend.key.spacing.x            : NULL
##  $ legend.key.spacing.y            : NULL
##  $ legend.frame                    : NULL
##  $ legend.ticks                    : NULL
##  $ legend.ticks.length             : 'rel' num 0.2
##  $ legend.axis.line                : NULL
##  $ legend.text                     :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : 'rel' num 0.8
##   ..$ hjust        : NULL
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : NULL
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ legend.text.position            : NULL
##  $ legend.title                    :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : num 0
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : NULL
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi TRUE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ legend.title.position           : NULL
##  $ legend.position                 : chr "right"
##  $ legend.position.inside          : NULL
##  $ legend.direction                : NULL
##  $ legend.byrow                    : NULL
##  $ legend.justification            : chr "center"
##  $ legend.justification.top        : NULL
##  $ legend.justification.bottom     : NULL
##  $ legend.justification.left       : NULL
##  $ legend.justification.right      : NULL
##  $ legend.justification.inside     : NULL
##  $ legend.location                 : NULL
##  $ legend.box                      : NULL
##  $ legend.box.just                 : NULL
##  $ legend.box.margin               : 'margin' num [1:4] 0cm 0cm 0cm 0cm
##   ..- attr(*, "unit")= int 1
##  $ legend.box.background           : list()
##   ..- attr(*, "class")= chr [1:2] "element_blank" "element"
##  $ legend.box.spacing              : 'simpleUnit' num 11points
##   ..- attr(*, "unit")= int 8
##   [list output truncated]
##  - attr(*, "class")= chr [1:2] "theme" "gg"
##  - attr(*, "complete")= logi TRUE
##  - attr(*, "validate")= logi TRUE
```

```
FPquality = table(Ames_data$Fireplace.Qu)
barplot(FPquality, space = 0, main = "Fireplace Quality Histogram", xlab = "Rating", ylab = "Frequency"
```

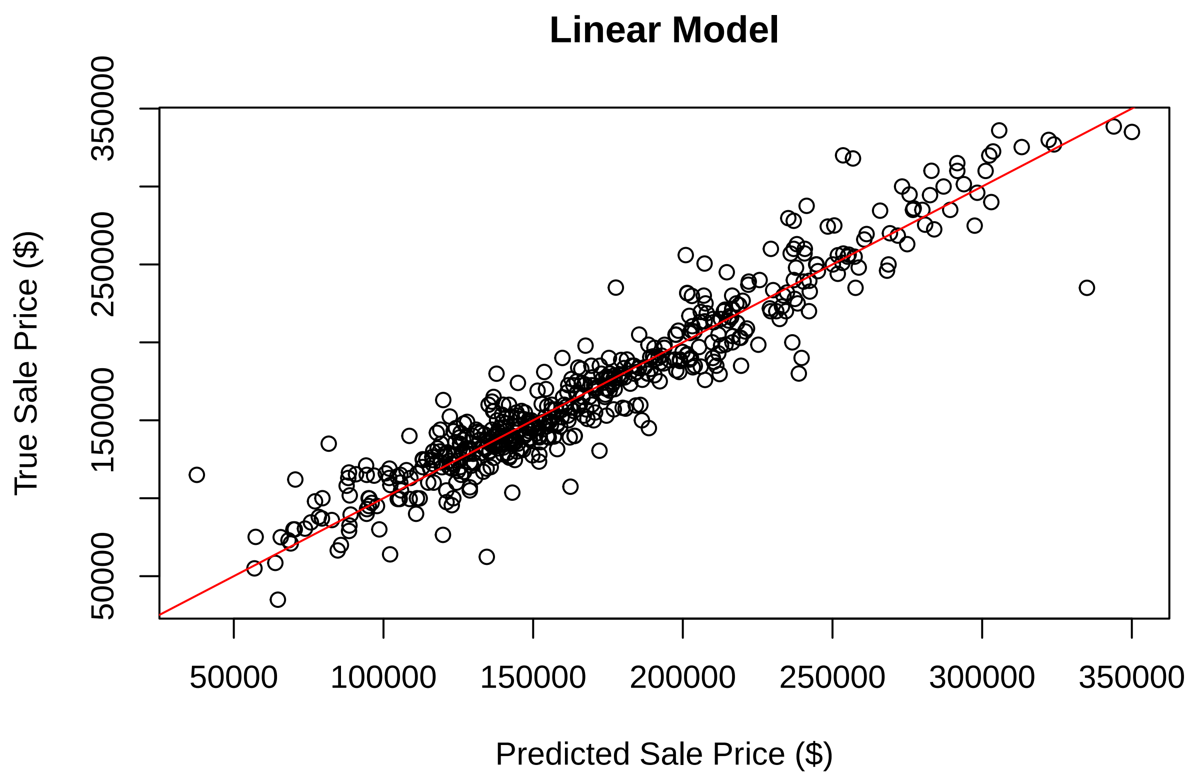


```
#Using the summary to help with variance of the nominal features for next step
summary_data = summary(Ames_data)

#Loop through the factors and remove rows that arent present more than 10 times to help with the data s
factors = names(Ames_data)[sapply(Ames_data, is.factor)]
for (factor in factors) {
  # Calculate level counts with the table
  level_counts = table(Ames_data[[factor]])

  # Get levels with counts greater than or equal to 10
  valid_levels = names(level_counts[level_counts >= 10])

  # Filter data frame based on the levels with sufficient data
  Ames_data = Ames_data %>% filter(.data[[factor]] %in% valid_levels)
}

#Data Splitting indices

Ames_train = sample(1:nrow(Ames_data), size = 0.7 * nrow(Ames_data)) # 70% for training
```

```r
Ames_test = setdiff(1:nrow(Ames_data), Ames_train)  # Remaining 30% for testing

#Linear model:
m1 = lm(SalePrice ~., data  = Ames_data[Ames_train,])
summary = summary(m1)

#Evaluating for the RMSE
predictions = predict(m1, newdata = Ames_data[Ames_test, ])
```

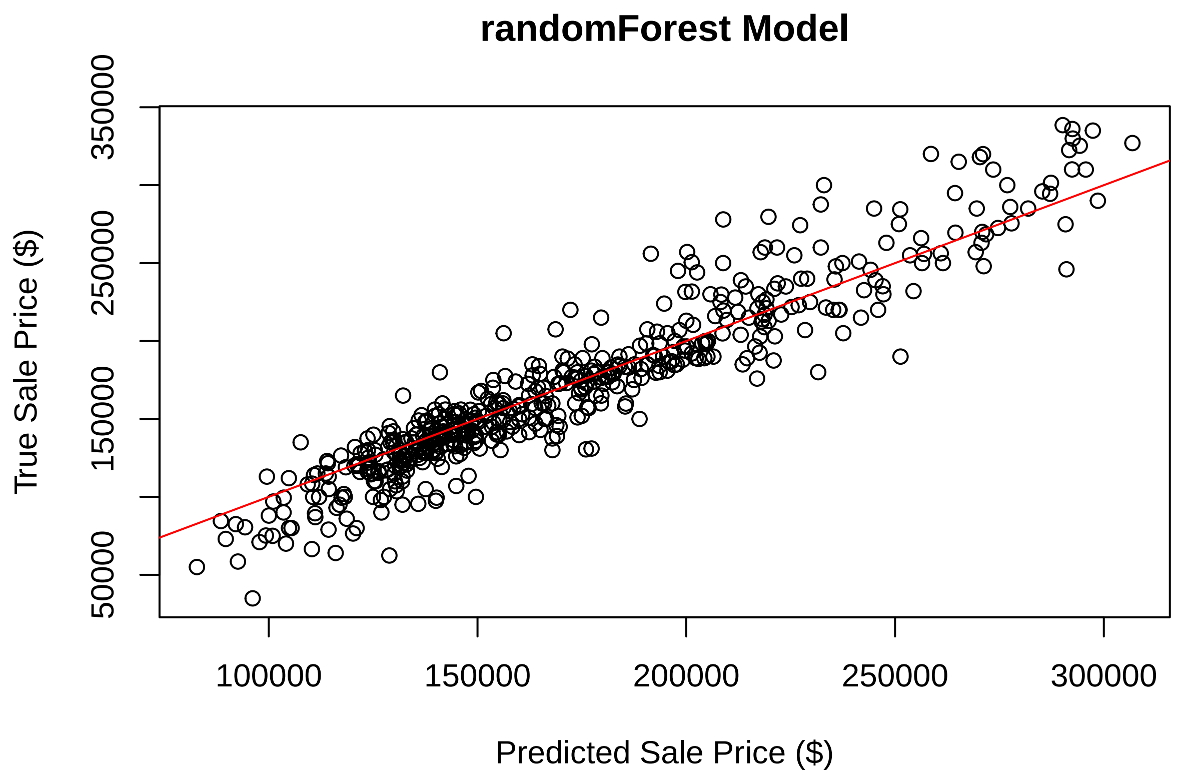
```
## Warning in predict.lm(m1, newdata = Ames_data[Ames_test, ]): prediction from
## rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```r
actual_values = Ames_data$SalePrice[Ames_test]
amean = mean(actual_values)
RMSE1 = sqrt(mean((predictions - actual_values)^2))
coeff_intervals = confint(m1)

plot(predictions, actual_values, main = "Linear Model", xlab = "Predicted Sale Price ($)", ylab = "True
abline(0, 1, col = "red")
```



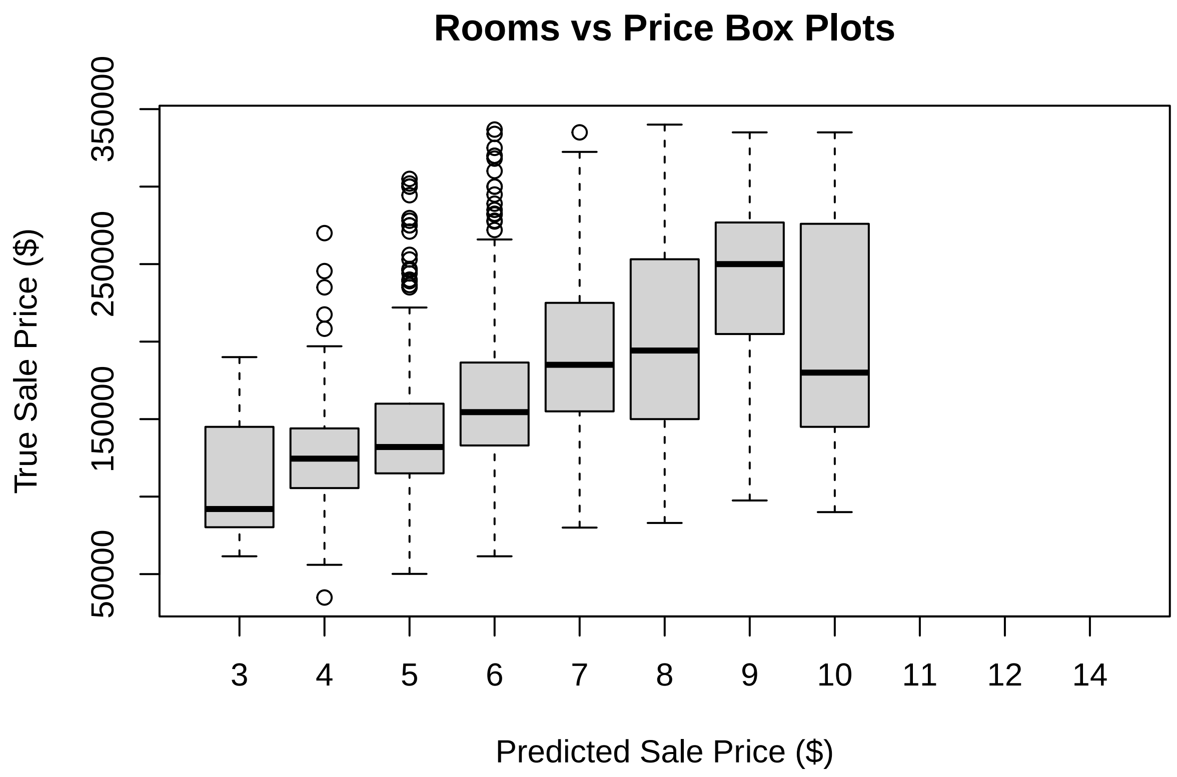
```r
#Error Percentage
Error1 = RMSE1/amean

#Bagging:
```

```r
num_features = ncol(Ames_data) - 1
bag_Ames = randomForest(SalePrice ~ ., data = Ames_data, subset = Ames_train, ntree = 50, mtry = sqrt(nu
#Evaluating for the RMSE and plot comparison line
yhat.bag = predict(bag_Ames, newdata = Ames_data[Ames_test, ])
RMSE2 = sqrt(mean((yhat.bag - Ames_data[Ames_test,"SalePrice"])^2))

plot(yhat.bag, actual_values, main = "randomForest Model", xlab = "Predicted Sale Price ($)", ylab = "Tr
abline(0, 1, col = "red")
```



```r
#Error Percentage
Error2 = RMSE2/amean

#ANOVA tests for categorical vars

categoricals = c(ordinals, nominals)
AL = list()

for (var in categoricals){

  formula = as.formula(paste("SalePrice ~", var))
  anova_result = aov(formula, data = Ames_data)
  AL[[var]] = summary(anova_result)

}
```

```r
plot(Ames_data$TotRms.AbvGrd, Ames_data$SalePrice, main = "Rooms vs Price Box Plots", xlab = "Predicted
```



References

Dean De Cock. Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project. Journal of Statistical Education, 19(3), 2011. Predicting Prices of Boston Housing Values. Boston Housing Project, nsamrao.github.io/Boston_Housing/. Accessed 12 Aug. 2023.