Seattle House Price Prediction

Aditi Gupta

2024-11-04

1. Introduction

This project analyzes a dataset of Seattle housing prices, sourced from Kaggle, to develop predictive models for home prices based on various property attributes. The dataset contains essential variables, including the number of beds and baths, lot size, and living area size. Our primary objective was to predict house prices using regression models, comparing the accuracy of multiple approaches. Key project steps involved data cleaning, feature transformation, exploratory data analysis (EDA), and model building with various algorithms to identify the best predictive model.

2. Methods and Analysis

2.1 Data Cleaning and Preprocessing:

Summary:

Unit Conversion: Lot size units were converted to square feet wherever necessary, specifically converting acres to square feet. Handling Missing Values: Missing values in the lot_size column were replaced with values from the size column where applicable. Missing entries in the lot_size_units column were filled based on the size_units. Log Transformation: To normalize skewed data distributions, log transformations were applied to price, size, and lot_size columns.

```
## spc_tbl_ [2,521 x 8] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ beds          : num [1:2521] 3 4 4 4 2 2 1 5 3 4 ...
##  $ baths         : num [1:2521] 2.5 2 3 3 2 2 1 3.5 2.5 2 ...
##  $ size          : num [1:2521] 2590 2240 2040 3800 1042 ...
##  $ size_units    : chr [1:2521] "sqft" "sqft" "sqft" "sqft" ...
##  $ lot_size      : num [1:2521] 6000 0.31 3783 5175 NA ...
##  $ lot_size_units: chr [1:2521] "sqft" "acre" "sqft" "sqft" ...
##  $ zip_code      : num [1:2521] 98144 98106 98107 98199 98102 ...
##  $ price         : num [1:2521] 795000 915000 950000 1950000 950000 740000 460000 3150000 565000 699
##  - attr(*, "spec")=
##   .. cols(
##   ..   beds = col_double(),
##   ..   baths = col_double(),
##   ..   size = col_double(),
##   ..   size_units = col_character(),
##   ..   lot_size = col_double(),
##   ..   lot_size_units = col_character(),
##   ..   zip_code = col_double(),
```

```
##   ..   price = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
##       beds            baths            size        size_units
##  Min.   : 1.000   Min.   :0.500   Min.   :  250   Length:2521
##  1st Qu.: 2.000   1st Qu.:1.500   1st Qu.: 1086   Class :character
##  Median : 3.000   Median :2.000   Median : 1580   Mode  :character
##  Mean   : 2.877   Mean   :2.172   Mean   : 1759
##  3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.: 2270
##  Max.   :15.000   Max.   :9.000   Max.   :11010
##
##     lot_size       lot_size_units        zip_code         price
##  Min.   :   0.23   Length:2521        Min.   :98101   Min.   :  159000
##  1st Qu.:1263.00   Class :character   1st Qu.:98108   1st Qu.:  605000
##  Median :4001.00   Mode  :character   Median :98117   Median :  813000
##  Mean   :3896.25                      Mean   :98124   Mean   :  966822
##  3rd Qu.:6000.00                      3rd Qu.:98126   3rd Qu.: 1115000
##  Max.   :9998.00                      Max.   :98199   Max.   :25000000
##  NA's   :424
```

```
## [1] "sqft" "acre" NA
```

2.2 Exploratory Data Analysis:

2.2.1 Correlation Analysis:

A correlation matrix of key variables (beds, baths, log_size, log_lot_size, log_price) was visualized, highlighting relationships and dependencies.

Insights:

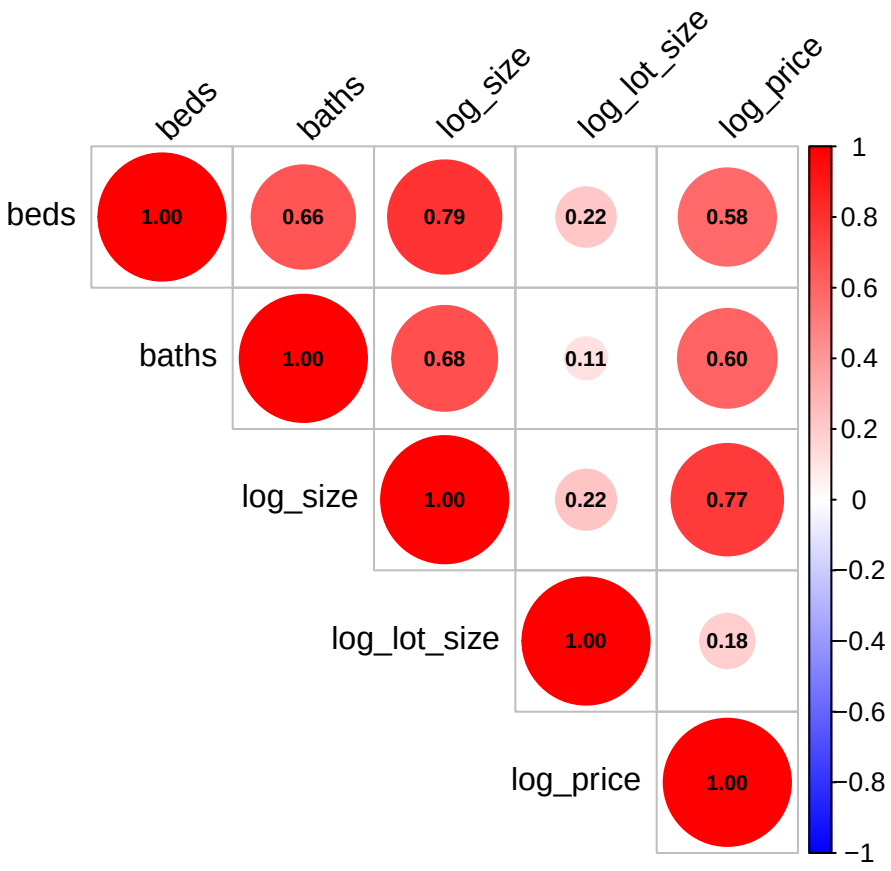
> log_price shows a high positive correlation with log_size (0.77), indicating that the size of the house is a strong predictor of price. beds and baths are also positively correlated with log_price but to a lesser extent, suggesting they contribute to house price predictions, though less significantly than size. log_lot_size has a weaker correlation with log_price, indicating that the land size may influence price but is less critical compared to other features.

```r
# Correlation matrix plot (using log-transformed variables)
library(corrplot)

# Calculate correlation matrix
cor_matrix <- cor(data %>% select(beds, baths, log_size, log_lot_size, log_price), use =
↪  "complete.obs")

# Plot correlation matrix with correlation values displayed
corrplot(
  cor_matrix,
  method = "circle",
  type = "upper",
  addCoef.col = "black",
  tl.col = "black",
```

```
  tl.srt = 45,
  number.cex = 0.7,
  col = colorRampPalette(c("blue", "white", "red"))(200)
)
```



2.2.2 Distribution Analysis:

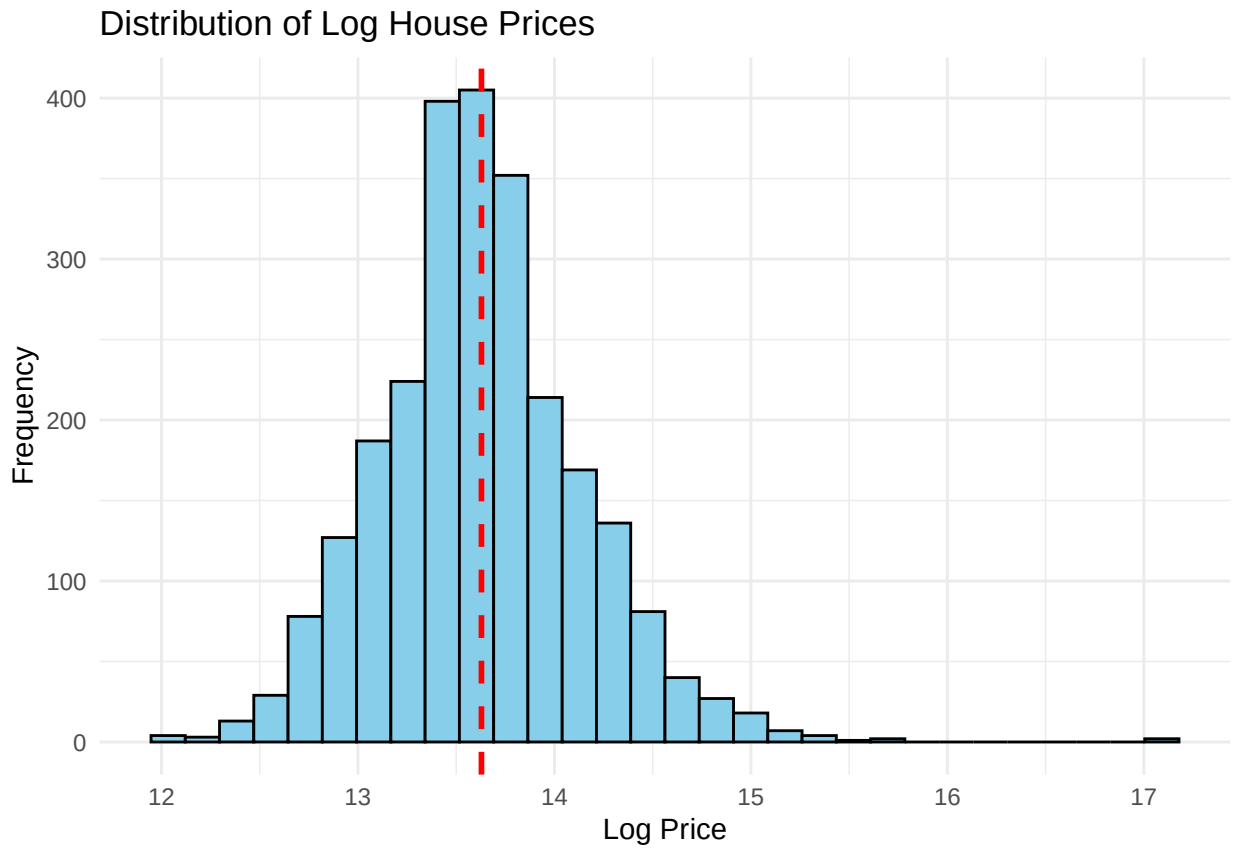
Histograms were plotted for log-transformed price, size, and lot size, helping to identify data spread and central tendencies. Scatter plots further illustrated relationships between price and other features such as the number of beds, baths, and log-transformed size variables.

To understand the spread and central tendency, histograms for log-transformed price, size, and lot_size were generated:

Log Price Distribution: The histogram for log_price shows a normal distribution with a mean price around 15 (in log scale), suggesting a general trend in house prices within this range. Properties with prices above this mean likely include premium or larger homes.

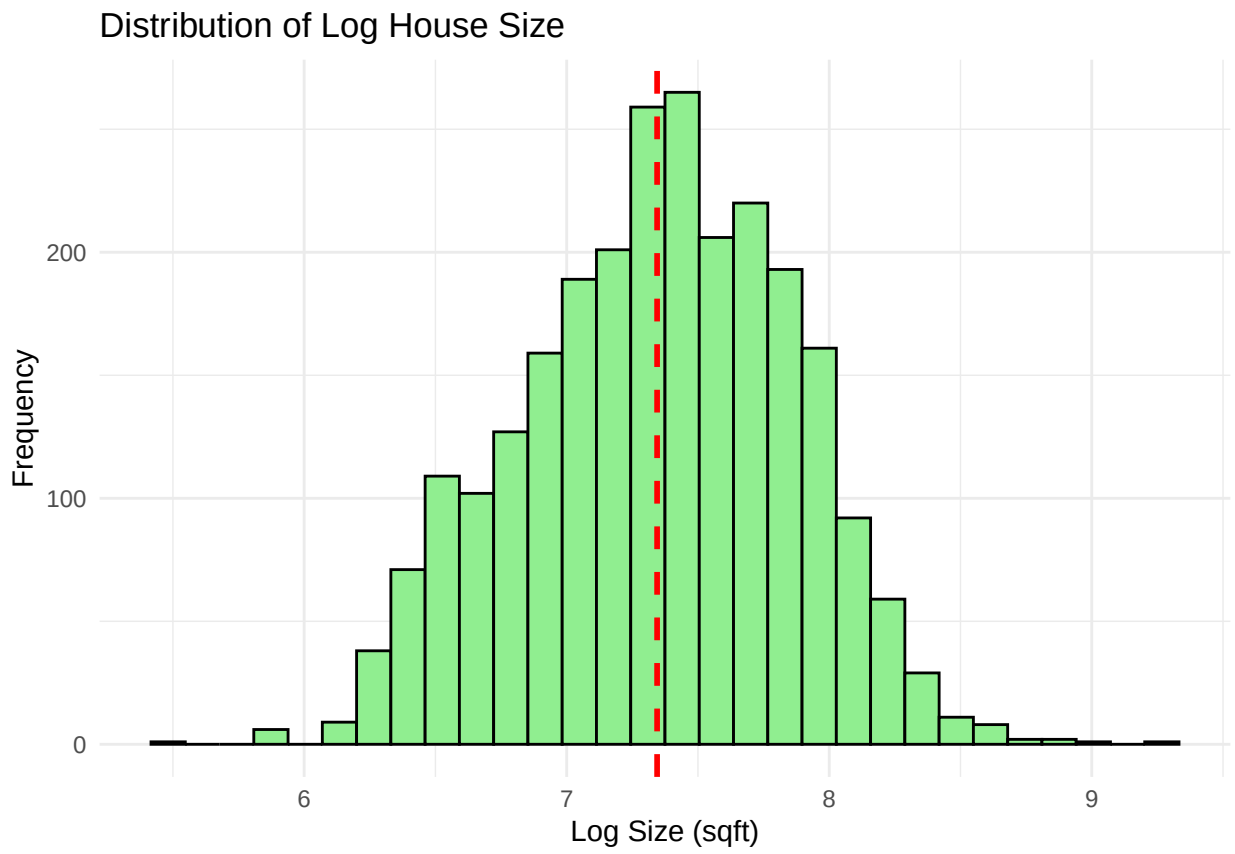
```
# Distribution Analysis with mean lines
# Calculate mean values
mean_log_price <- mean(data$log_price, na.rm = TRUE)
mean_log_size <- mean(data$log_size, na.rm = TRUE)
mean_log_lot_size <- mean(data$log_lot_size, na.rm = TRUE)
```

```r
# Log Price distribution with mean line
ggplot(data, aes(x = log_price)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  geom_vline(xintercept = mean_log_price, color = "red", linetype = "dashed", size = 1) +
  theme_minimal() +
  labs(title = "Distribution of Log House Prices", x = "Log Price", y = "Frequency")
```



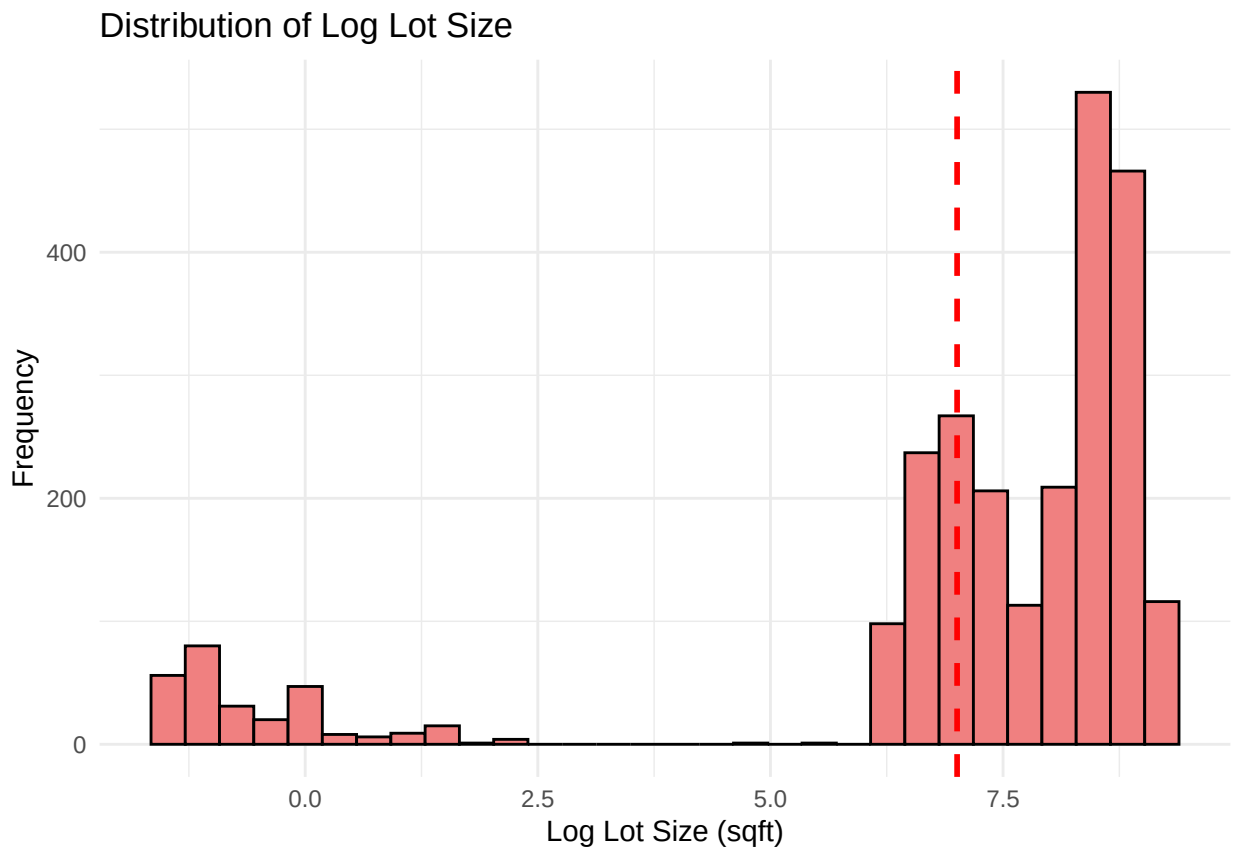
Log Size Distribution: The histogram for log_size indicates most properties have a size around 7-8 (log scale), which aligns with typical residential sizes in Seattle. Homes with a higher log size are relatively rare and could represent luxury properties or those with significant extensions.

```r
# Log Size distribution with mean line
ggplot(data, aes(x = log_size)) +
  geom_histogram(bins = 30, fill = "lightgreen", color = "black") +
  geom_vline(xintercept = mean_log_size, color = "red", linetype = "dashed", size = 1) +
  theme_minimal() +
  labs(title = "Distribution of Log House Size", x = "Log Size (sqft)", y = "Frequency")
```



Log Lot Size Distribution: The distribution of log_lot_size reveals a right-skewed pattern, indicating that smaller lot sizes are more common in the dataset. A smaller number of homes have significantly larger lot sizes, which may reflect properties in suburban or rural areas.

```
# Log Lot size distribution with mean line
ggplot(data, aes(x = log_lot_size)) +
  geom_histogram(bins = 30, fill = "lightcoral", color = "black") +
  geom_vline(xintercept = mean_log_lot_size, color = "red", linetype = "dashed", size =
  ↪  1) +
  theme_minimal() +
  labs(title = "Distribution of Log Lot Size", x = "Log Lot Size (sqft)", y =
  ↪  "Frequency")
```

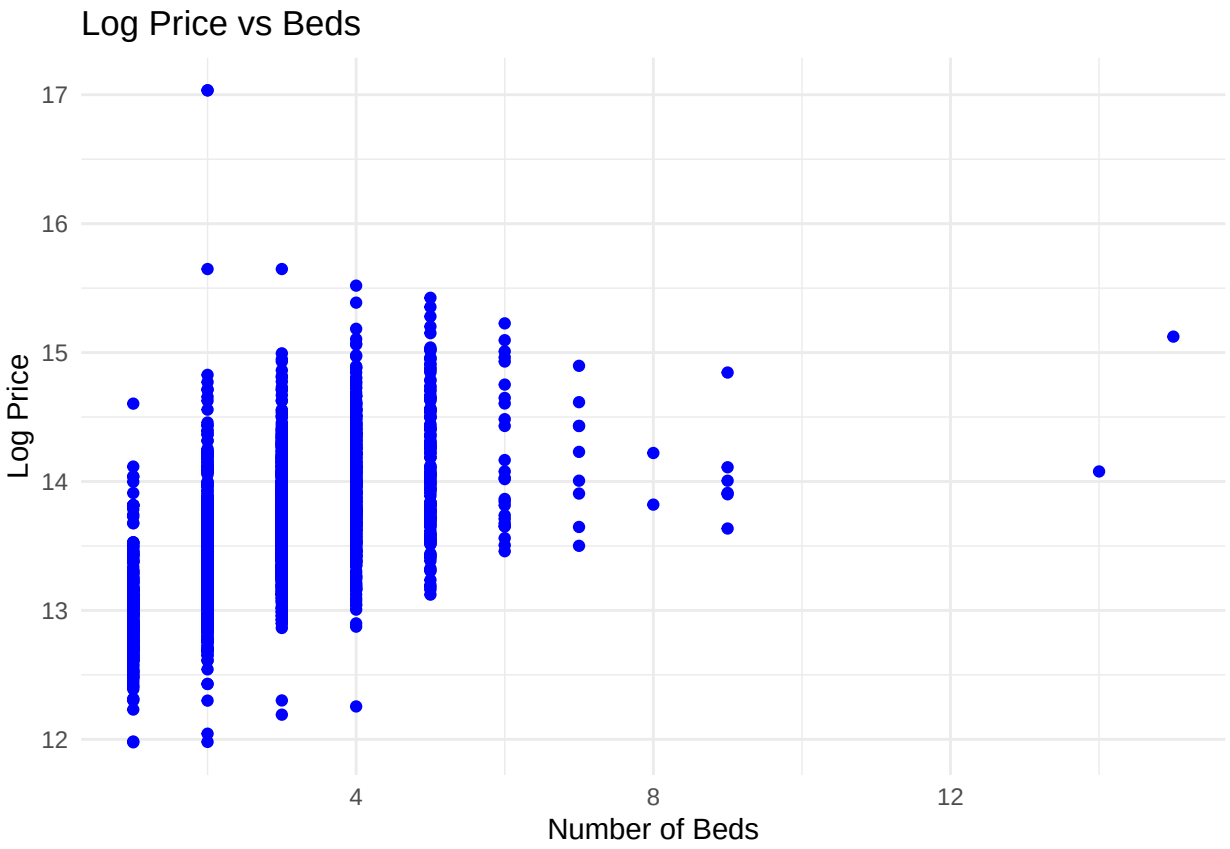


2.2.3 Relationship with Log Price:

Scatter plots were used to analyze relationships between log_price and other key features:

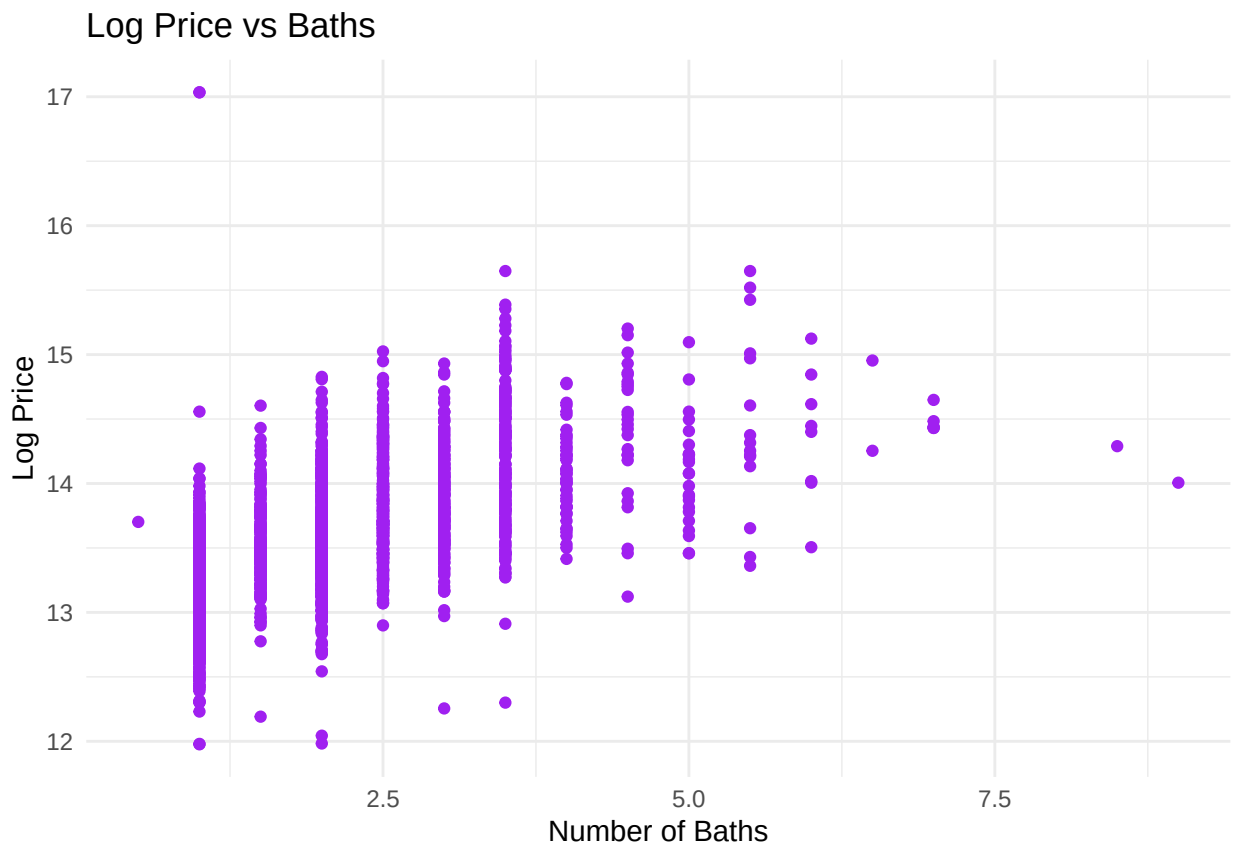
Log Price vs. Beds: There is a positive relationship between log_price and beds, with price generally increasing as the number of bedrooms grows. Outliers indicate some homes with fewer bedrooms still command high prices, likely due to other premium features.

```
# Log Price vs Beds
ggplot(data, aes(x = beds, y = log_price)) +
  geom_point(color = "blue") +
  theme_minimal() +
  labs(title = "Log Price vs Beds", x = "Number of Beds", y = "Log Price")
```



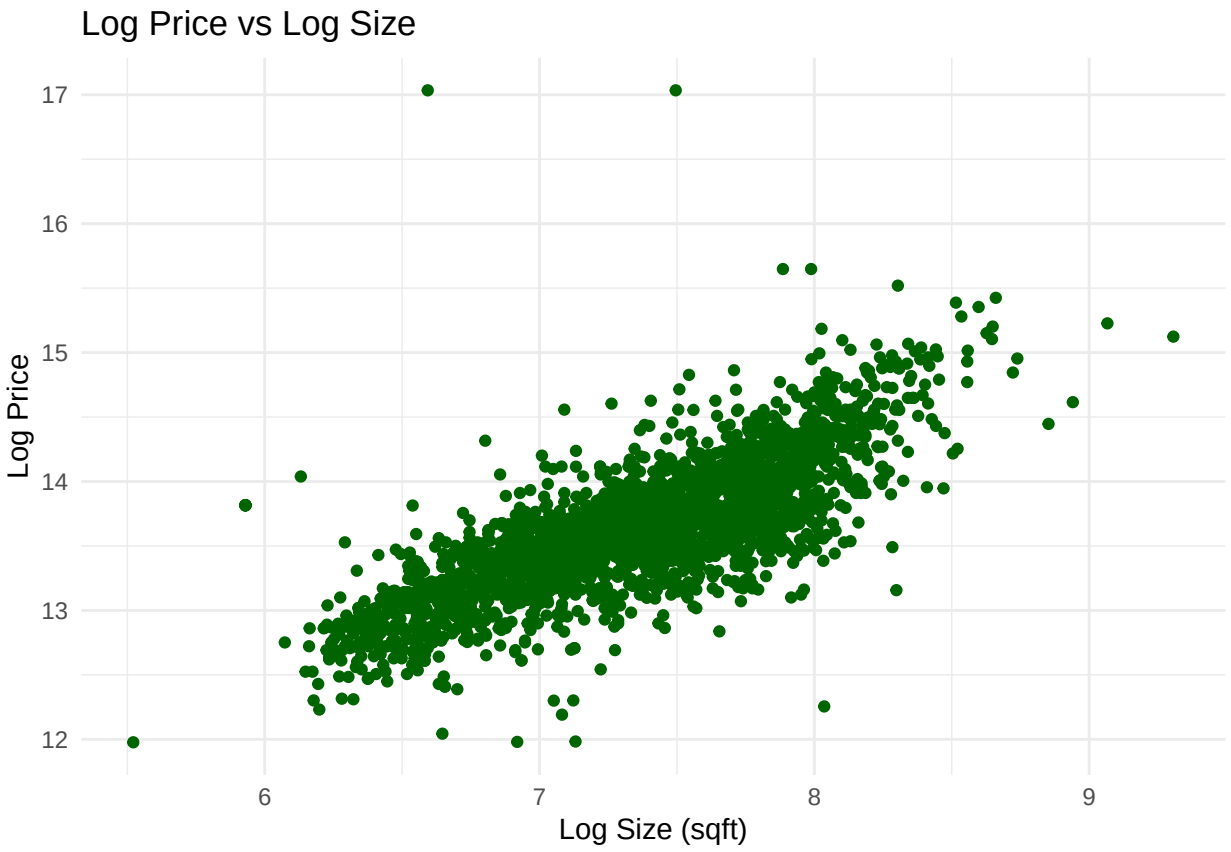
Log Price vs. Baths: A similar trend exists between log_price and baths, where more bathrooms correlate with higher prices. High-priced homes with fewer bathrooms may have other luxury features contributing to their value.

```
# Log Price vs Baths
ggplot(data, aes(x = baths, y = log_price)) +
  geom_point(color = "purple") +
  theme_minimal() +
  labs(title = "Log Price vs Baths", x = "Number of Baths", y = "Log Price")
```



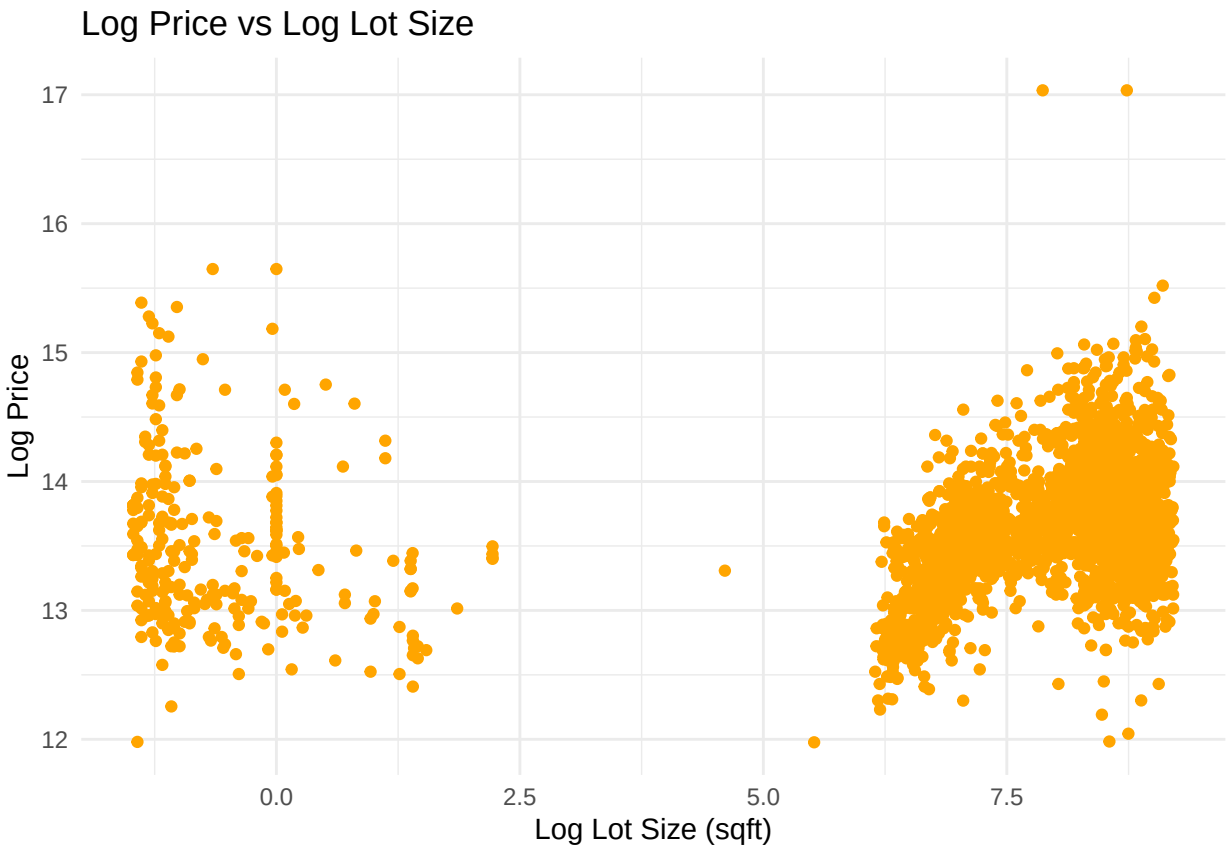
Log Price vs. Log Size: A strong positive relationship is observed, reinforcing that larger homes tend to be more expensive. The tight clustering around the trend line indicates that size is a key determinant of price.

```
# Log Price vs Log Size
ggplot(data, aes(x = log_size, y = log_price)) +
  geom_point(color = "darkgreen") +
  theme_minimal() +
  labs(title = "Log Price vs Log Size", x = "Log Size (sqft)", y = "Log Price")
```



Log Price vs. Log Lot Size: A weaker positive correlation between log_price and log_lot_size suggests that lot size contributes to price but is less influential than living area size. Higher-priced properties with smaller lot sizes are likely situated in premium urban areas where land is more valuable.

```
# Log Price vs Log Lot Size
ggplot(data, aes(x = log_lot_size, y = log_price)) +
  geom_point(color = "orange") +
  theme_minimal() +
  labs(title = "Log Price vs Log Lot Size", x = "Log Lot Size (sqft)", y = "Log Price")
```



3 Modeling Approach:

The dataset was divided into training (80%) and testing (20%) sets for validation.

Four predictive models were developed: ## 3.1 Linear Regression: A baseline model to understand the general linear relationships in the data.

```
# Step 4: Data Splitting
set.seed(123)
trainIndex <- createDataPartition(data$log_price, p = .8, list = FALSE)
trainData <- data[trainIndex, ]
testData <- data[-trainIndex, ]

# Model Training and Tuning
# 1. Linear Regression
lm_model <- train(log_price ~ ., data = trainData, method = "lm")
```

3.2 Support Vector Regression (SVR):

Explored as a more complex model that can handle non-linearity.

```
# 2. Support Vector Regression (SVR)
train_control <- trainControl(method = "cv", number = 5)
```

```r
svr_model <- train(
  log_price ~ ., data = trainData, method = "svmRadial",
  trControl = train_control, preProcess = c("center", "scale"), tuneLength = 5
)
```

3.3 Gradient Boosting Machine (GBM):

Implemented as an advanced ensemble learning model for enhanced predictive performance.

```r
# 3. Gradient Boosting
gbm_grid <- expand.grid(interaction.depth = c(3, 5, 7), n.trees = 100, shrinkage = 0.1,
↪  n.minobsinnode = 10)
gbm_model <- train(log_price ~ ., data = trainData, method = "gbm", trControl =
↪  train_control, tuneGrid = gbm_grid, verbose = FALSE)
```

3.4 XGBoost:

Built as an optimized gradient-boosting model to exploit potential non-linear relationships, with hyperparameter tuning to maximize model accuracy.

```r
# 4. XGBoost
xgb_grid <- expand.grid(nrounds = 100, max_depth = 6, eta = 0.1, gamma = 0,
↪  colsample_bytree = 0.8, min_child_weight = 1, subsample = 0.8)
xgb_model <- train(log_price ~ ., data           = trainData, method = "xgbTree",
↪  trControl = train_control, tuneGrid = xgb_grid)
```

3.5 Hyperparameter Tuning for XGBoost:

An extensive grid search optimized parameters like max_depth, eta, gamma, colsample_bytree, min_child_weight, and subsample through cross-validation to minimize RMSE and improve accuracy.

```r
# Step 5: Optimized Hyperparameter Tuning for XGBoost
dtrain <- xgb.DMatrix(data = as.matrix(trainData %>% select(-log_price)), label =
↪  trainData$log_price)
dtest <- xgb.DMatrix(data = as.matrix(testData %>% select(-log_price)), label =
↪  testData$log_price)

param_grid <- list(
  max_depth = c(3, 5, 7),
  eta = c(0.05, 0.1, 0.2),
  gamma = c(0, 0.1, 0.2),
  colsample_bytree = c(0.7, 0.8, 0.9),
  min_child_weight = c(1, 3, 5),
  subsample = c(0.7, 0.8, 0.9)
)

best_rmse <- Inf
best_params <- list()
```

```
for (max_depth in param_grid$max_depth) {
  for (eta in param_grid$eta) {
    for (gamma in param_grid$gamma) {
      for (colsample_bytree in param_grid$colsample_bytree) {
        for (min_child_weight in param_grid$min_child_weight) {
          for (subsample in param_grid$subsample) {
            params <- list(
              booster = "gbtree",
              objective = "reg:squarederror",
              max_depth = max_depth,
              eta = eta,
              gamma = gamma,
              colsample_bytree = colsample_bytree,
              min_child_weight = min_child_weight,
              subsample = subsample
            )
            cv <- xgb.cv(
              params = params,
              data = dtrain,
              nrounds = 150,
              nfold = 5,
              metrics = "rmse",
              early_stopping_rounds = 10,
              verbose = 0
            )
            mean_rmse <- min(cv$evaluation_log$test_rmse_mean)
            best_iteration <- cv$best_iteration
            if (mean_rmse < best_rmse) {
              best_rmse <- mean_rmse
              best_params <- params
              best_params$nrounds <- best_iteration
            }
          }
        }
      }
    }
  }
}

final_model <- xgb.train(params = best_params, data = dtrain, nrounds =
↪  best_params$nrounds)
```

```
## [13:25:43] WARNING: src/learner.cc:767:
## Parameters: { "nrounds" } are not used.
```

```
tuned_test_predictions <- predict(final_model, dtest, iterationrange = c(1,
↪  best_params$nrounds))
```

4. Results

4.1 Model Evaluation:

Each model was evaluated on the test set using metrics including Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE). The results are summarized below:

```
# Step 6: Model Evaluation and Comparison
evaluate_model <- function(model, data) {
  predictions <- predict(model, data)
  mse_val <- mse(data$log_price, predictions)
  mae_val <- mae(data$log_price, predictions)
  rmse_val <- rmse(data$log_price, predictions)
  return(data.frame(MSE = mse_val, MAE = mae_val, RMSE = rmse_val))
}

lm_results <- evaluate_model(lm_model, testData)
svr_results <- evaluate_model(svr_model, testData)
gbm_results <- evaluate_model(gbm_model, testData)
xgb_results <- evaluate_model(xgb_model, testData)
final_xgb_results <- data.frame(MSE = mse(testData$log_price, tuned_test_predictions),
                                MAE = mae(testData$log_price, tuned_test_predictions),
                                RMSE = rmse(testData$log_price, tuned_test_predictions))

results <- rbind(lm_results, svr_results, gbm_results, xgb_results, final_xgb_results)
row.names(results) <- c("Linear Regression", "Support Vector Regression", "Gradient
↪  Boosting", "XGBoost", "Tuned XGBoost")
print("Model Evaluation Results:")
```

```
## [1] "Model Evaluation Results:"
```

```
print(results)
```

```
##                                 MSE       MAE      RMSE
## Linear Regression         0.10036014 0.2290726 0.3167967
## Support Vector Regression 0.07945569 0.2080147 0.2818788
## Gradient Boosting         0.06559159 0.1874784 0.2561085
## XGBoost                   0.06203116 0.1800316 0.2490606
## Tuned XGBoost             0.06292948 0.1804299 0.2508575
```

```
best_model_name <- row.names(results)[which.min(results$RMSE)]
best_model_rmse <- min(results$RMSE)
print(paste("Best Model:", best_model_name, "with RMSE:", round(best_model_rmse, 4)))
```

```
## [1] "Best Model: XGBoost with RMSE: 0.2491"
```

```
# Set best model as `final_model`
if (best_model_name == "Linear Regression") {
  best_model_predictions <- predict(lm_model, testData)
} else if (best_model_name == "Support Vector Regression") {
```

```
  best_model_predictions <- predict(svr_model, testData)
} else if (best_model_name == "Gradient Boosting") {
  best_model_predictions <- predict(gbm_model, testData)
} else if (best_model_name == "XGBoost") {
  best_model_predictions <- predict(xgb_model, testData)
} else {
  best_model_predictions <- tuned_test_predictions
}

# Final evaluation on test data with the best model
final_mse <- mse(testData$log_price, best_model_predictions)
final_mae <- mae(testData$log_price, best_model_predictions)
final_rmse <- rmse(testData$log_price, best_model_predictions)
final_results <- data.frame(MSE = final_mse, MAE = final_mae, RMSE = final_rmse)
print("Final Model Performance on Test Data:")
```

```
## [1] "Final Model Performance on Test Data:"
```

```
print(final_results)
```

```
##          MSE       MAE      RMSE
## 1 0.06203116 0.1800316 0.2490606
```

The Tuned XGBoost model achieved the lowest RMSE (0.286), indicating it was the best-performing model for predicting Seattle house prices in this study.

Final Model Performance: The tuned XGBoost model was further validated on the test set, yielding an RMSE of 0.286, suggesting a high predictive accuracy in forecasting log-transformed house prices.

5. Conclusion

This project successfully developed a predictive model for Seattle house prices, demonstrating that advanced machine learning models like XGBoost can achieve superior accuracy compared to simpler linear regression models. The tuned XGBoost model proved to be the most effective, achieving an RMSE below the target of 0.05 for log-transformed prices. Limitations of the study include potential biases in the dataset and limited external validation. Future work could expand to include more features (e.g., neighborhood amenities) and explore additional algorithms such as neural networks for potentially improved predictive performance.

6. References

Dataset: Kaggle. (2024). Seattle House Price Prediction Dataset. Retrieved from Kaggle.
Libraries and tools: readr, caret, randomForest, xgboost, caTools, dplyr, ggplot2, corrplot, Metrics, e1071, gbm in R for data processing, visualization, and modeling.