Predicting House Prices

Mauricio Razon

2023-09-17

Introduction

The purpose of the project is to build a model that will predict house prices in the USA. I will be embedding multiple machine learning methods to identify which model type produces the best accuracy for this regression problem.

Inspiration and Motivation:

In my entire life, I have never lived or owned a house. Due to extreme circumstances, my family never had the right amount of money to purchase a house. The state of California has such ridiculous high house prices due to the limited availability of land and high demand of houses that makes it difficult for families to purchase a house. Of course, renting a house is an alternative approach but even so, many are burdened with the high prices of rent. Therefore, I wanted to find out what variables play a key role in determining the price a house and can we further use this information to produce an model that accurately predicts the price of a house.

Project Outline:

First, I will explore the data set further and fix any inconsistency with the units for the variables. Afterwards, I will investigate the amount of missing values in the data set and if any are present, I will delete them. Then, I will extract the variables that won't be useful for the EDA portion. I will arrive to this decision by the variable's definition and units. After selecting the appropriate variables, I will then conduct an EDA on the data set. From conducting an EDA on the specified variables, I hope to gain a understanding of the relationships between the variables. Specifically, what variables has a greater on influence the response variable and their distributions. Afterwards, I will preform a training and testing data split on the data. Then, I will create the recipe and set folds for the 10-fold cross validation that will be implemented. The following model types will be embedded on the training: KNN, Ridge Regression, Lasso Regression, Linear Regression, and Elastic Net Linear Regression. Finally, we will extract the RMSE value of each model and used whatever model has the lowest RMSE, we will apply it to our testing data to see how well our model does at predicting house prices.

Lets get started!

##Loading Packages: Let's first load the necessary packages:

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.3     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ ggplot2   3.4.3     ✓ tibble    3.2.1
## ✓ lubridate 1.9.2     ✓ tidyr     1.3.0
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
e errors
```

```
library(dplyr)
library(tidymodels)
```

```
## ── Attaching packages ─────────────────────────────────────── tidymodels 1.1.1 ──
## ✓ broom        1.0.5     ✓ rsample      1.2.0
## ✓ dials        1.2.0     ✓ tune         1.1.2
## ✓ infer        1.0.5     ✓ workflows    1.1.3
## ✓ modeldata    1.2.0     ✓ workflowsets 1.0.1
## ✓ parsnip      1.1.1     ✓ yardstick    1.2.0
## ✓ recipes      1.0.8
## ── Conflicts ────────────────────────────────────────── tidymodels_conflicts() ──
## ✖ scales::discard() masks purrr::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ recipes::fixed()  masks stringr::fixed()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ yardstick::spec() masks readr::spec()
## ✖ recipes::step()   masks stats::step()
## • Use suppressPackageStartupMessages() to eliminate package startup messages
```

```
library(readr)
library(kknn)
library(janitor)
```

```
##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(ISLR)
library(discrim)
```

```
##
## Attaching package: 'discrim'
##
## The following object is masked from 'package:dials':
##
##     smoothness
```

```
library(corrr)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(tidytext)
library(ggplot2)
library(finalfit)
library(purrr)
```

Loading the Data Set:

Now, lets load the data set and take a look at the first few rows of observations:

```
house <- read.csv(file = "Realtors.csv")
head(house)
```

```
##     status bed bath acre_lot          city       state zip_code house_size
## 1 for_sale   3    2     0.12      Adjuntas Puerto Rico      601        920
## 2 for_sale   4    2     0.08      Adjuntas Puerto Rico      601       1527
## 3 for_sale   2    1     0.15    Juana Diaz Puerto Rico      795        748
## 4 for_sale   4    2     0.10         Ponce Puerto Rico      731       1800
## 5 for_sale   6    2     0.05      Mayaguez Puerto Rico      680         NA
## 6 for_sale   4    3     0.46 San Sebastian Puerto Rico      612       2520
##   prev_sold_date  price
## 1                105000
## 2                 80000
## 3                 67000
## 4                145000
## 5                 65000
## 6                179000
```

The data is from the Kaggle data set, “USA Real Estate Data Set”. Kaggle User Ahmed Shahriar Sakib uploaded the data set. It can accessed through the following link: https://www.kaggle.com/datasets/ahmedshahriarsakib/usa-real-estate-dataset (https://www.kaggle.com/datasets/ahmedshahriarsakib/usa-real-estate-dataset). The data was collected from the real estate listing website, realtor. Realtor happens to be the 2nd most used real estate listing website as of 2021. It can be accessed through the following link: https://www.realtor.com/ (https://www.realtor.com/). Also, The website is operated by the News Corp subsidiary Move, Inc.

Note: The data set gets updated every know and then since it can accessed by every user. I have included the most latest Data set.

Exploring and Tidying the Data:

Let's begin by inspecting our data.

```
dim(house)
```

```
## [1] 904966     10
```

The dimensions of the data set are 904966 and 10, meaning we have 904966 observations (houses) and 10 variables. However, as you can see from the data set not all variables will be necessary for our analysis.

Let's take a further look at the data to decide which of the variables we can exclude from our data set.

```
head(house)
```

```
##     status bed bath acre_lot          city       state zip_code house_size
## 1 for_sale   3    2     0.12      Adjuntas Puerto Rico      601        920
## 2 for_sale   4    2     0.08      Adjuntas Puerto Rico      601       1527
## 3 for_sale   2    1     0.15    Juana Diaz Puerto Rico      795        748
## 4 for_sale   4    2     0.10         Ponce Puerto Rico      731       1800
## 5 for_sale   6    2     0.05      Mayaguez Puerto Rico      680         NA
## 6 for_sale   4    3     0.46 San Sebastian Puerto Rico      612       2520
##   prev_sold_date  price
## 1                105000
## 2                 80000
## 3                 67000
## 4                145000
## 5                 65000
## 6                179000
```

The variable `prev_sold_date` can be dropped from the data set, as it does not contain useful information for our analysis or models. A house's previous sold date will not affect the price of a house nor inform us about its correlation it may have with the other variables. It is pretty much useless so we can drop this variable from the data set.

The same can be said about the `status`. A variable which does not provide much useful information for the purpose of the project. We can drop this variable, as well.

The variables `city` and `zip_code` will not be needed. Since we have `state`, we do not need `city` as we make better generalizations and inferences of the data using only `state`. Also including `city` can complicate our models and analysis. `zip_code` essentially provides information for postal services and delivery purposes. It will not be useful to include this variable.

Excluding the variables mentioned above, our modified data set turns out to be as follows:

```
#Dropping Unwanted Variables
house <- subset(house, select = -c(status,zip_code,city,prev_sold_date))
#Modified Data Set:
head(house)
```

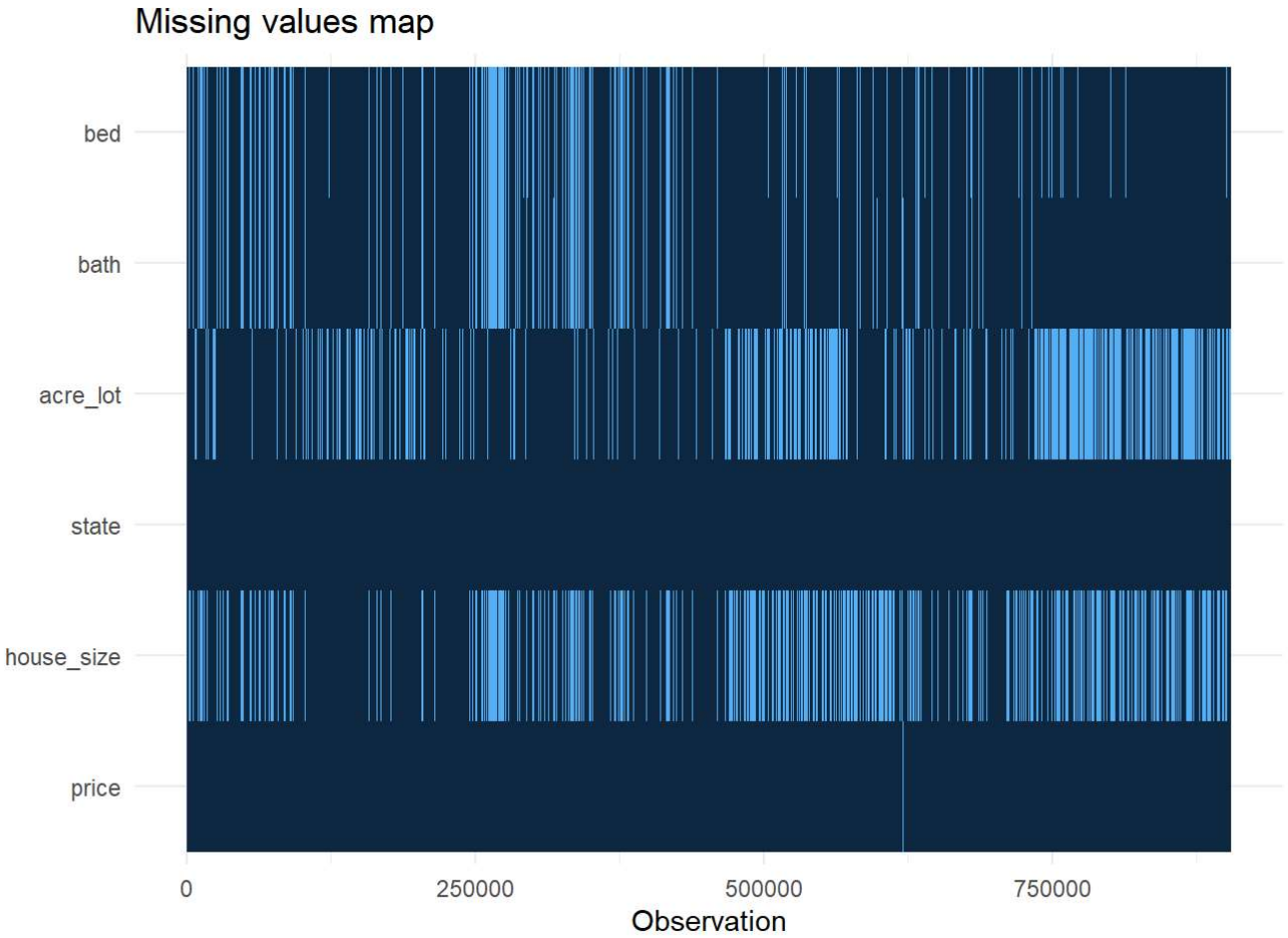
```
##   bed bath acre_lot       state house_size  price
## 1   3    2     0.12 Puerto Rico        920 105000
## 2   4    2     0.08 Puerto Rico       1527  80000
## 3   2    1     0.15 Puerto Rico        748  67000
## 4   4    2     0.10 Puerto Rico       1800 145000
## 5   6    2     0.05 Puerto Rico         NA  65000
## 6   4    3     0.46 Puerto Rico       2520 179000
```

Awesome! Our new data set contains useful variables for our analysis. Let's do some further inspection and tidying.

Missing Data

Let's see if our data has any missing values.

```
house %>% missing_plot()
```



Wow!

We have a lot of missing values for the following variables: `bed` , `bath` , `acre_lot` , `house_size` , and `price`

Let's see how much exactly:

```
colSums(is.na(house))
```

```
##        bed       bath   acre_lot      state house_size      price
##     129840     113884     266642          0     292886         71
```

Now, let's see the size of the data set if we were to exclude the missing values. This would help improve the model's performance and robustness.

```
house <- na.omit(house)
nrow(house)
```

```
## [1] 413163
```

Our modified data set is 413163 if we were to exclude the missing values. One of the major downsides of excluding the `NA` values is that the number of observations decreases to a value of half the original size. This is a huge marginal difference. However, our data set without these missing values is still relatively large. Also, excluding missing values increases the model's performance and robustness. The pros exceed the cons, so I will be removing the null values from the data set.

Checking the outcome type:

We have no more missing observations for `price` . Lets now check that `price` is indeed numeric. It is always important that our response variable is in the type appropriate for the regression problem. In this case, our outcome, `price` , must be continous and numerical. Let's check if our data set does label `price` as numeric.

```
is.numeric(house$price)
```

```
## [1] TRUE
```

Yes, our response variable, `price` is in the desired form, numeric. We are able to continue.

Tidying the variables:

The variable `acre_lot` is in acres while the variable `house_size` is in sq feet. This is a sign of inconsistency. I will convert `acre_lot` to sq feet. This will help us understand the data better.

```
#Conversion
house$sq_lot <- house$acre_lot
house$sq_lot <- house$sq_lot*43560
head(house$sq_lot)
```

```
## [1]  5227.2  3484.8  6534.0  4356.0 20037.6  8712.0
```

```
#Lets remove acre_lot
house <- subset(house, select = -c(acre_lot))
```

Our variable `acre_lot` has been renamed to `sq_lot` to reflect the conversion we had to do for consistency purposes.

Also I will need to convert state into factors based on the unique values of `state` :

In fact, what states do we even have in our data set?

```
unique(house$state)
```

```
##  [1] "Puerto Rico"    "Virgin Islands" "Massachusetts"  "Connecticut"
##  [5] "New Jersey"     "New York"       "New Hampshire"  "Vermont"
##  [9] "Rhode Island"   "Wyoming"        "Maine"          "Pennsylvania"
## [13] "West Virginia"  "Delaware"
```

Okay, so we have 14 unique values for `state`. It will be difficult to make generalizations with these values and can over complicate the models and analysis.

How about I grouped them up based on the divisions they reside in.

Puerto Rico and Virgin Islands are part of The Caribbean.

New Jersey, New York, Pennsylvania are in the Mid-Atlantic division.

Rhode Island, Massachusetts, Connecticut, New Hampshire, Vermont, and Maine are all in the New England division.

Delaware, Georgia, South Carolina, West Virginia are part of the South Atlantic division.

That leaves Wyoming in the Mountain division and Tennessee in the East South Central Division.

In total, we have 6 divisions which is much better than having 14 different states.

Now, we can assign the observations to their respected division in the `state` and make a new variable called `division`. We can then delete the `state` from the data set.

```
#Caribbean Div
house$state[house$state == 'Puerto Rico'] <- 'Caribbean'
house$state[house$state == 'Virgin Islands'] <- 'Caribbean'

# Mid-Atlantic Div
house$state[house$state == 'New Jersey'] <- 'Mid-Atlantic'
house$state[house$state == 'New York'] <- 'Mid-Atlantic'
house$state[house$state == 'Pennsylvania'] <- 'Mid-Atlantic'

# New England Div
house$state[house$state == 'Rhode Island'] <- 'New England'
house$state[house$state == 'Massachusetts'] <- 'New England'
house$state[house$state == 'Connecticut'] <- 'New England'
house$state[house$state == 'New Hampshire'] <- 'New England'
house$state[house$state == 'Vermont'] <- 'New England'
house$state[house$state == 'Maine'] <- 'New England'

#South Atlantic
house$state[house$state == 'Delaware'] <- 'South Atlantic'
house$state[house$state == 'West Virginia'] <- 'South Atlantic'
house$state[house$state == 'Georgia'] <- 'South Atlantic'
house$state[house$state == 'South Carolina'] <- 'South Atlantic'
house$state[house$state == 'Virginia'] <- 'South Atlantic'

#Mountain
house$state[house$state == 'Wyoming'] <- 'Mountain'

#East South Central
house$state[house$state == 'Tennessee'] <- 'East South Central'

#Create the division column
house$division <- house$state

#Delete the state column
house <- subset(house, select= -c(state))


#Assign division to factor
house$division <- as.factor(house$division)
head(house)
```

```
##   bed bath house_size  price  sq_lot  division
## 1   3    2        920 105000  5227.2 Caribbean
## 2   4    2       1527  80000  3484.8 Caribbean
## 3   2    1        748  67000  6534.0 Caribbean
## 4   4    2       1800 145000  4356.0 Caribbean
## 6   4    3       2520 179000 20037.6 Caribbean
## 7   3    1       2040  50000  8712.0 Caribbean
```

Now our data looks a lot organized and easy to understand. Lets continue.

Describing the Variables:

Lets display the first few rows of our modified data set.

```
head(house)
```

```
##   bed bath house_size  price  sq_lot  division
## 1   3    2        920 105000  5227.2 Caribbean
## 2   4    2       1527  80000  3484.8 Caribbean
## 3   2    1        748  67000  6534.0 Caribbean
## 4   4    2       1800 145000  4356.0 Caribbean
## 6   4    3       2520 179000 20037.6 Caribbean
## 7   3    1       2040  50000  8712.0 Caribbean
```

After removing the unnecessary variables, Our modified data set includes the variables that will provide important information in the analysis portion. As seen above, the data set displays the following variables: `bed` , `bath` , `sq_lot` , `division` , `house_size` , and `price` .

Let's define the variables mentioned previously: * `bed` : The number of beds * `bath` : The number of baths * `division` : The division where the house is located. * `sq_lot` : The property/land size in sq feet * `house_size` : The size of the house in sq feet * `price` : The price of the house. This is our outcome variable aka the variable we are trying to predict.

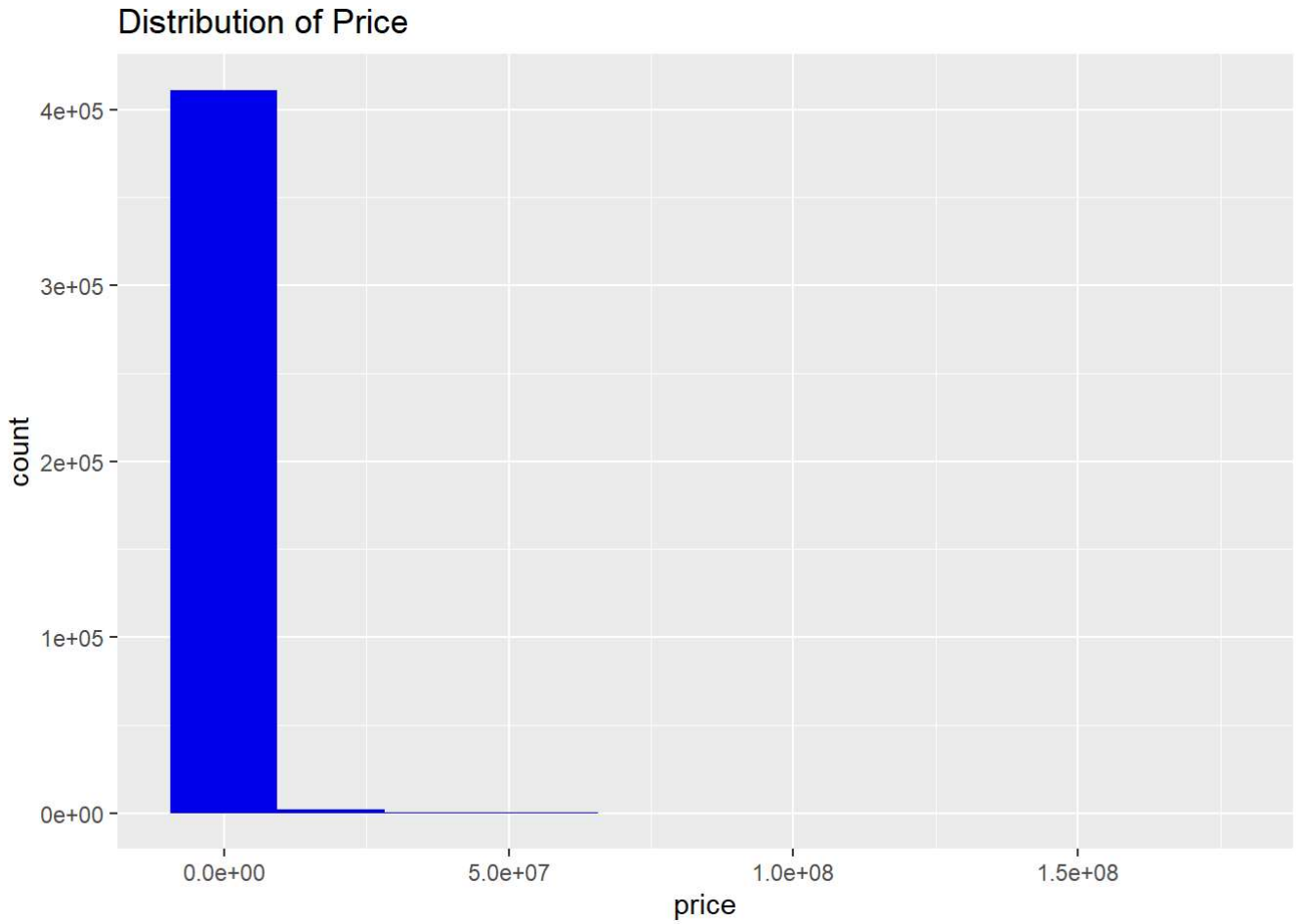
Exploratory Data Analysis:

We will now conduct EDA to see the relationships the selected variables have with the outcome variable and each with each other.

Statistics and Distribution of Outcome:

Let's see the distribution of our outcome variable, `price` :

```
ggplot(aes(x = price), data = house) +
  geom_histogram(fill='blue2', bins = 10) +
  labs(
    title = "Distribution of Price"
  )
```



We clearly are not getting much visualization from this graph. There must be outliers present in our data that affects the distribution. That explains why it's hard to get a visualization of the outcome.

Extracting the outliers:

The issue with keeping these outliers is that it skews our data and increases the bias, which is not something we want. Also, it violates the assumptions of linear regression model, which will be embedded later. We can limit these values but keep in mind removing these observations decreases our sample size. I will conduct the process of limiting outliers on our response variable, `price`. I won't decide to do the same for the other variables as it makes it complicated to make generalizations since majority of the observations from the original data set are missing. Also, it makes it difficult to make inferences if we remove all of the outliers.

We will run a for loop to detect the outliers. Once detected, we will replace the outliers with `NA`. Then, we will drop the `NA`, thus removing or at least limiting the outliers.

```
#Function for outliers
for (i in 'price')
{
  value = house[,i][house[,i] %in% boxplot.stats(house[,i])$out]
  house[,i][house[,i] %in% value] = NA
}

#Amt. of outliers present for the price variable variable
sum(is.na(house$price))
```

```
## [1] 36947
```

```
#Extraction
house <- house[!is.na(house$price),]

#double-check to see if outliers are truly gone
sum(is.na(house$price))
```

```
## [1] 0
```

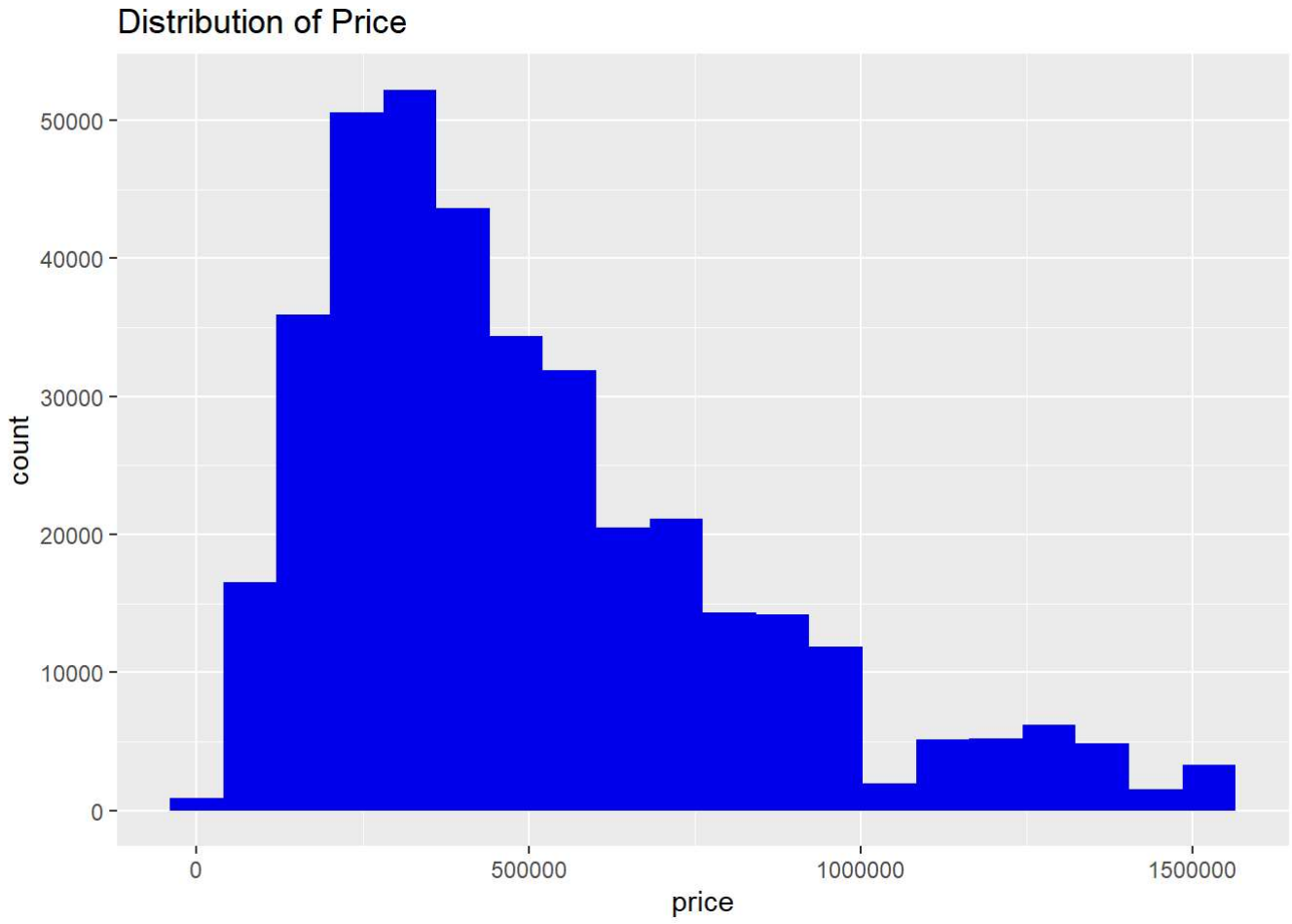
Source: The following function was developed by Digital Ocean Author, Safa Mulani, on the Digtal Ocean Website. The function can be accessed through the following link: https://www.digitalocean.com/community/tutorials/outlier-analysis-in-r (https://www.digitalocean.com/community/tutorials/outlier-analysis-in-r)

After running our function one times, we have limited the amount of outliers. Let's continue. We could extract more observations if we run the We extracted about 36947 observations in the `price` column that were outliers.

Updated Histogram of Price with outliers removed:

Now lets see our histogram with removal of outliers:

```
ggplot(aes(x = price), data = house) +
  geom_histogram(fill='blue2', bins = 20) +
  labs(
    title = "Distribution of Price"
  )
```




```
#Statistics of Outcome:
summary(house$price)
```

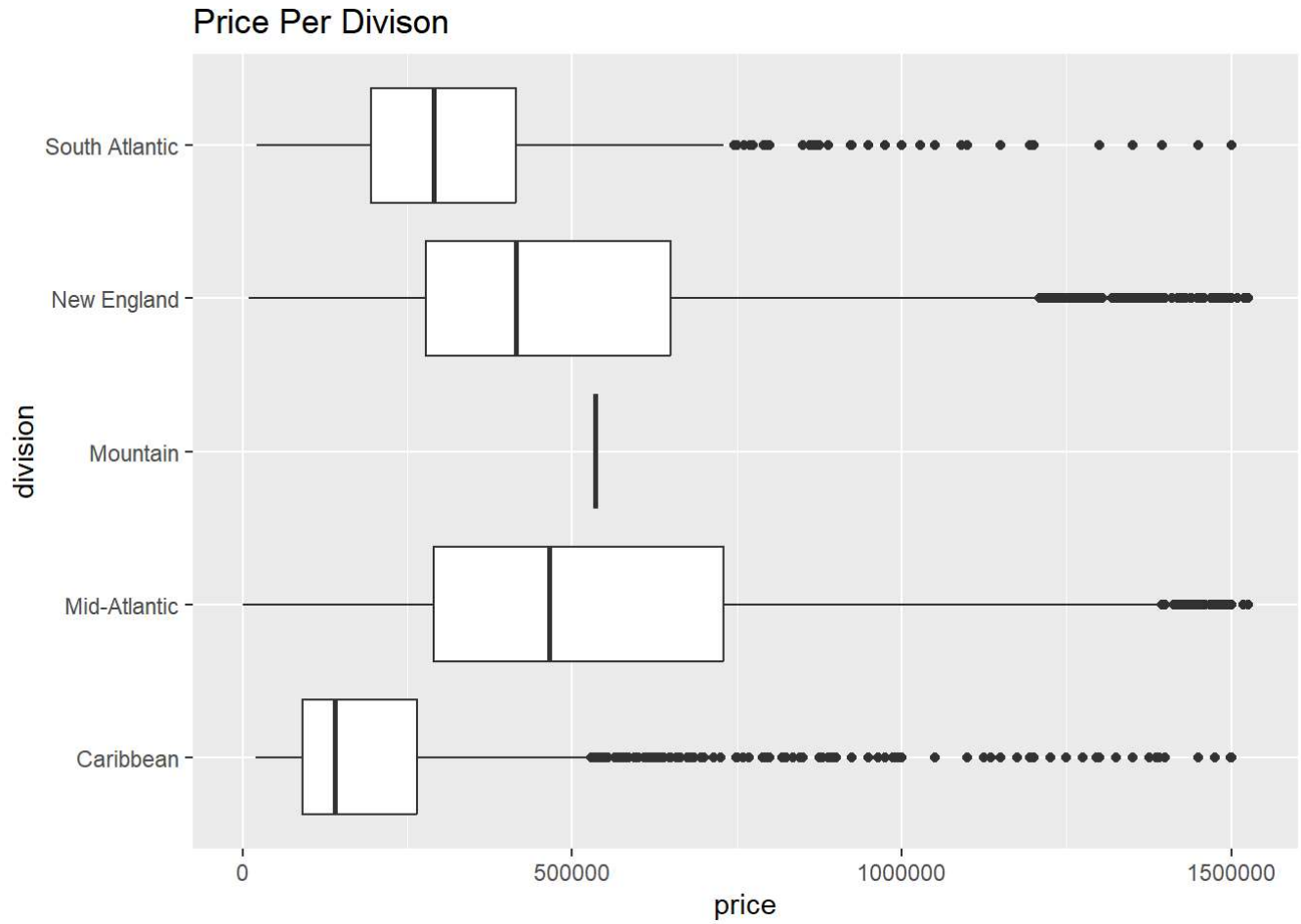
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     500  269900  419900  500977  669000 1525000
```

Awesome! We can confirm see the outcome variable is skewed left. The maximum value of `price` is 15250000 and the mean falls around 500977. The minimum value is 500.

Boxplot of House Price per division:

Let's create boxplots for the distribution of house prices per division. I want to see how the distribution of prices compare for each division.

```
library(ggplot2)
ggplot(aes(y = division, x = price), data = house) +
  geom_boxplot() +labs(title = "Price Per Divison")
```

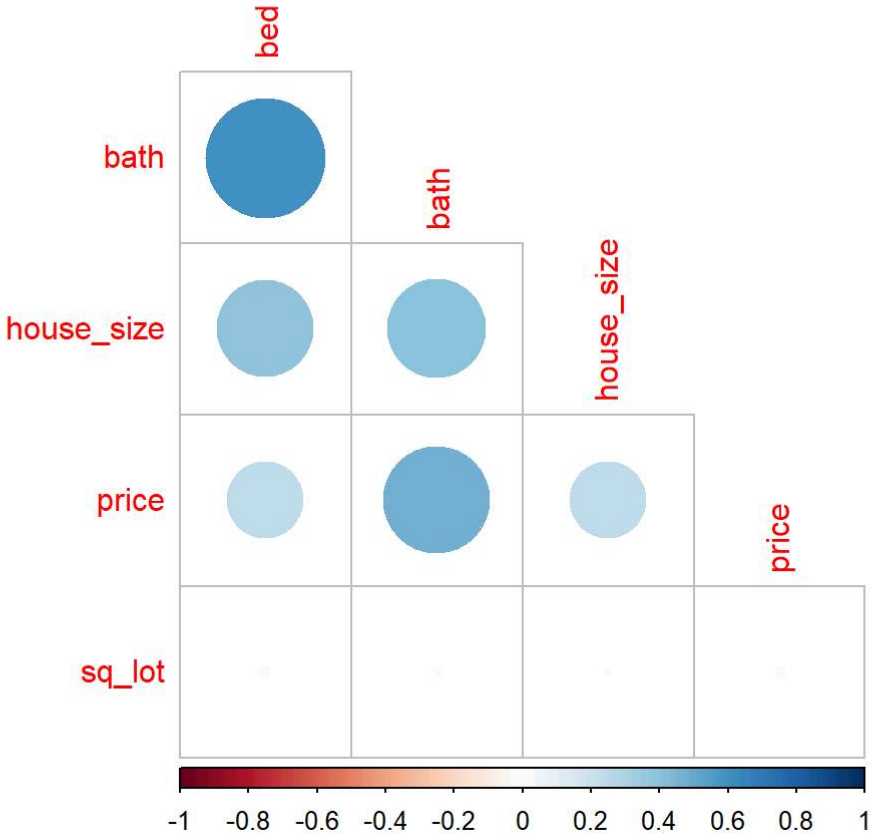


We can see from the graph that all the medians of `price` are different from one another based on the division. The Mid-Atlantic division appears to have the highest median, followed by the New England Division. It is interesting to see that the Caribbean has the smallest median. However, the division does have a lot of outliers that skews the division's observations.

Correlation Plot:

Let's now develop a Correlation Plot to view the correlation of our variables to the outcome

```
library(corrplot)
library(dplyr)
house %>%
  select(bed, bath, house_size, price, sq_lot) %>% # selecting numeric columns
  cor() %>%
  corrplot(type = "lower", diag = F)
```



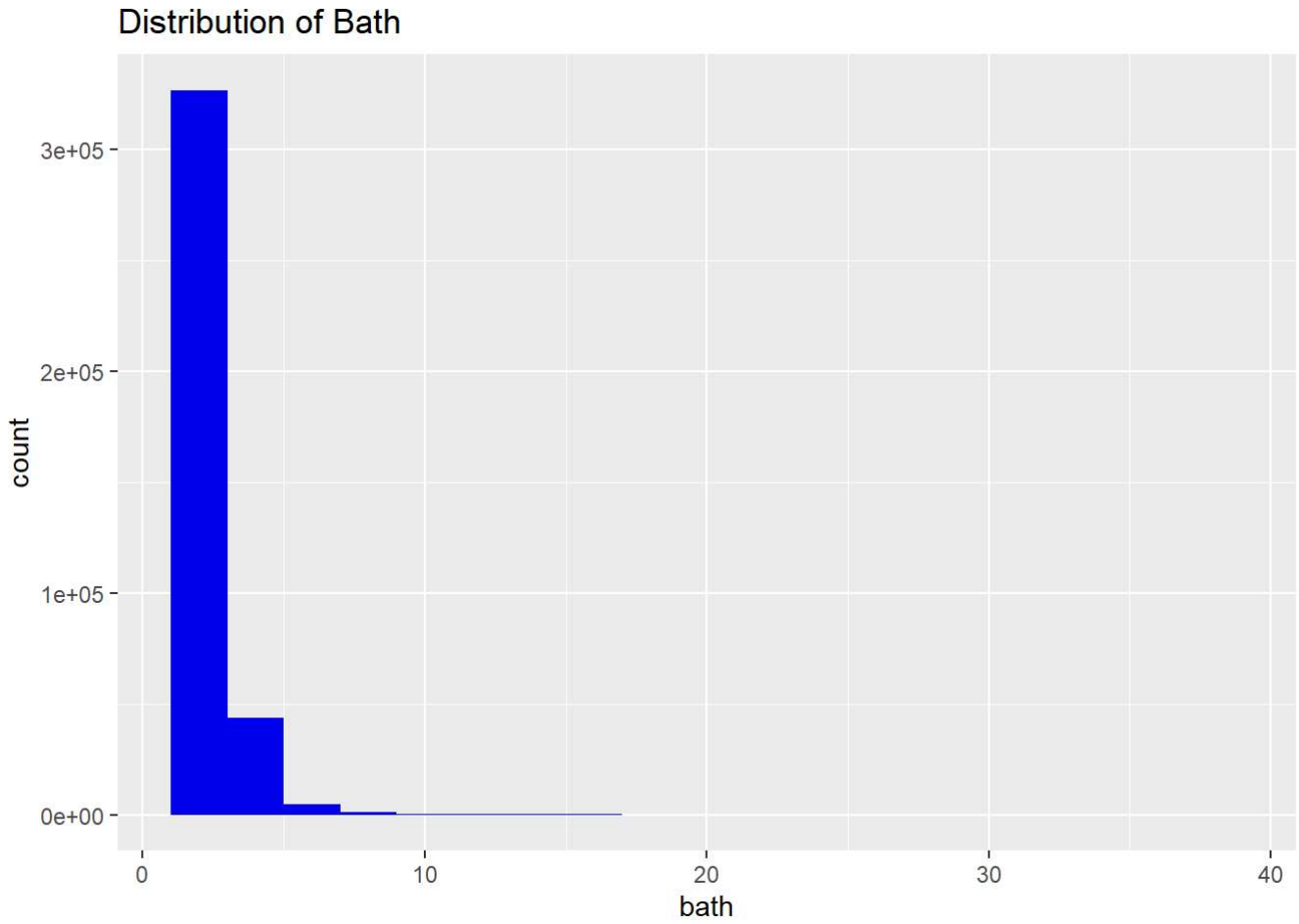
From the correlation plot, we see that there is a positive linear correlations between `bed`, `bath`, `house_size`, and `price`. There is a positive linear correlation between `bath`, `house_size`, and `price`. `price` appears to have a strong positive linear correlation with `bath`. `bed` and `bath` have a strong positive linear correlation. This does make sense as we would expect the number of baths to increase with the number of beds and vice versa. `sq_lot` does not have any correlation between the variables, which is interesting to see. I expected `sq_lot` to be correlated with `price` or `house_size`.

Distribution and Summary of Bath:

I would like to investigate the correlation between `price` and `bath` further. According to the correlation plot, `price` has the strongest positive linear correlation with `bath`.

First, lets look at the distribution of `bath`:

```
ggplot(aes(x = bath), data = house) +
  geom_histogram(fill='blue2', bins = 20) +
  labs(
    title = "Distribution of Bath"
  )
```




```
# 5 Number Summary
summary(house$bath)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   2.000   2.431   3.000  39.000
```

```
#Frequencies
table(house$bath)
```

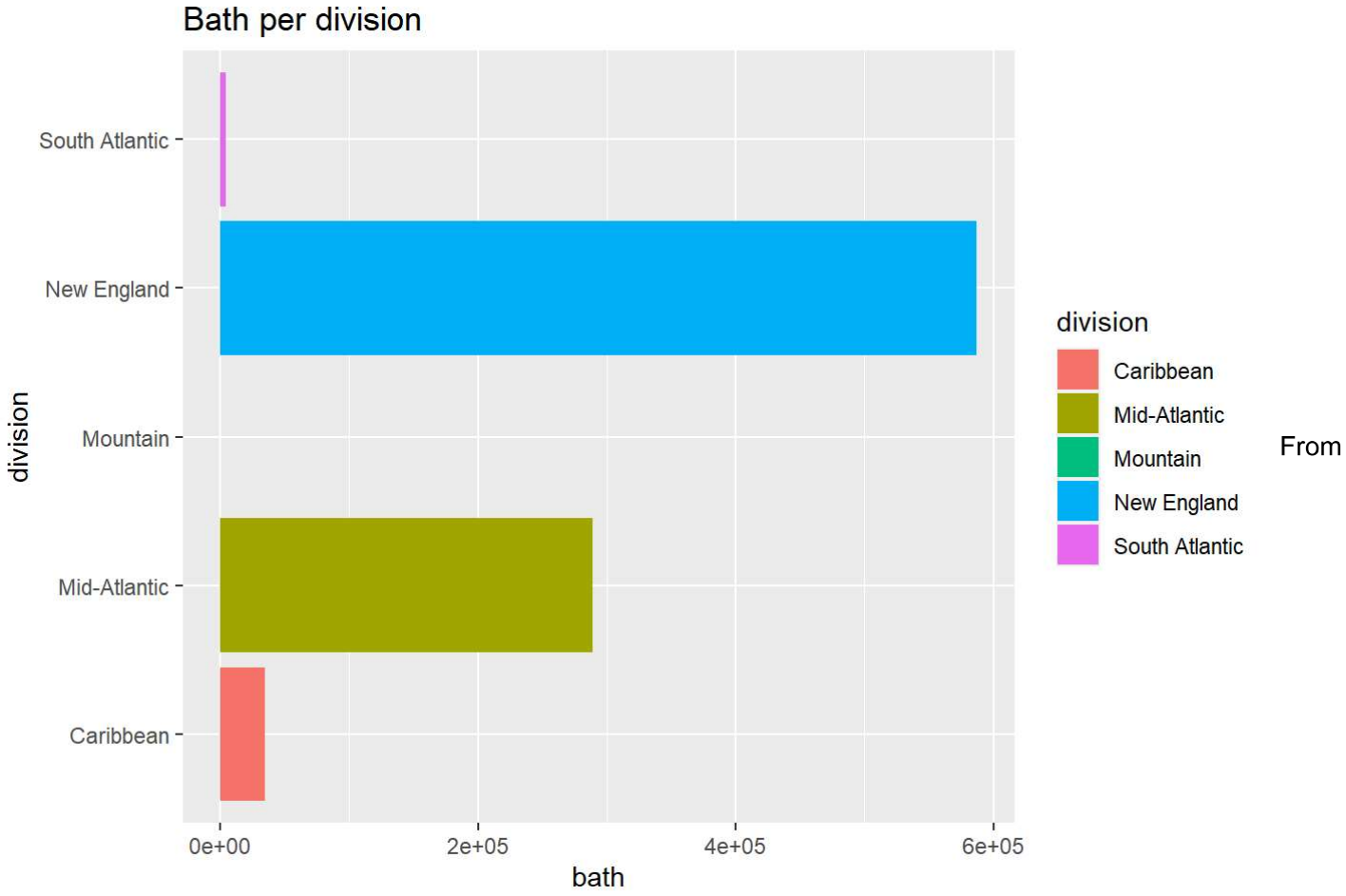
```
##
##      1      2      3      4      5      6      7      8      9     10     11
##  71927 148488 106122  33505  10021   3848    954    497    414    130     20
##     12     13     14     15     16     17     18     25     39
##     61     47     33     37     77      7     14     11      3
```

From the histogram, we see that the `bath` tends to fall within the 2-3 range. The data is skewed left. From the summary function, we see that the max amount of the number of bathrooms is 39 and the minimum is 1.

Barplot of Number of bath per division:

Now lets see the number of `bath` per `division` :

```
ggplot(house, aes(y = division, x = bath, fill = division)) + geom_bar(stat = 'identity') + labs
(title = "Bath per division")
```



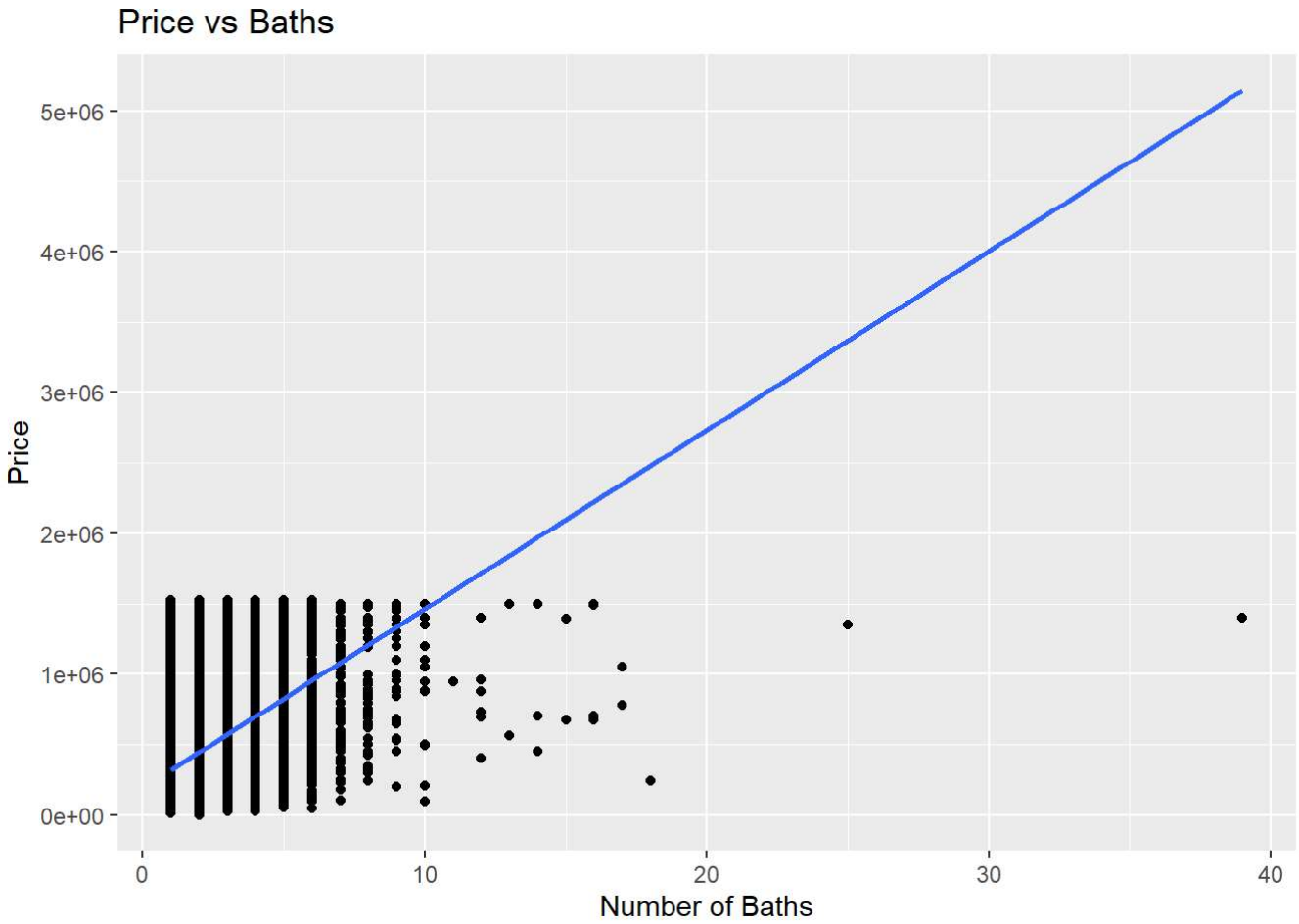
From the barplot, we can see that most of the bathrooms included in the data set comes from the Mid-Atlantic Division, which includes the following states: New Jersey, New York, and Pennsylvania. The New England division has the second most bathrooms included in the data. We can see that the Mountain and East South Central division's don't have too many observations to be accounted for in the barplot.

Scatterplot of Price vs Bath:

Now that we know the distribution of the variable that has the highest correlation with `bath`, lets make a scatter plot with `price` on the y-axis and `bath` on the x-axis. We can see if there indeed is a relationship through the scatter plot.

```
ggplot(house, aes(x = bath, y = price)) + geom_point() + stat_smooth(method = "lm", se = F) +lab
s(title = "Price vs Baths", x = "Number of Baths", y = 'Price')
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



From the scatter plot, we can see that there is a positive linear relationship between the two variables. Surely, an increase in the number of `baths` lead to an increase in `price`. The reason why the relationship is not very strong is because of the outliers present. These outliers affect the relationship between the two variables. I won't decide to remove them for each variable since it will help understand the relationship between the variable and `price`. I only limited the amount of outliers for `price` because the distribution of `price` was difficult to visualize. I could not get an idea of the distribution of `price` since it was vastly affected by outliers.

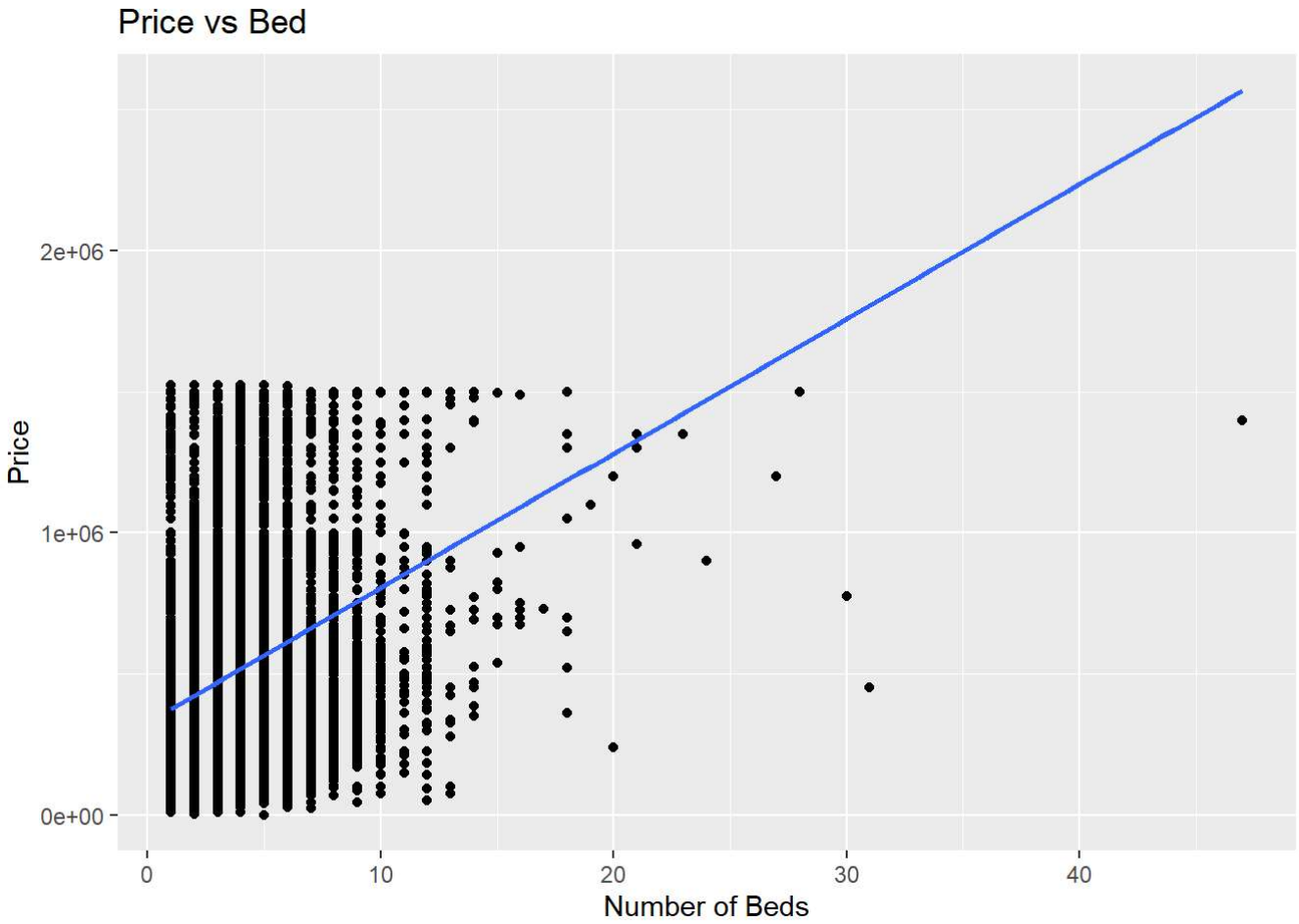
Scatterplot of Price vs Bed:

Now, I will focus on the positive, but not as strong compared to `bath`, variables that are correlated with `price`. In this case, it was `bed` and `house_size`

Lets start with `bed`. I will construct a scatter plot as I did previously with `price` on the y-axis and `bed` on axis

```
ggplot(house, aes(x = bed, y = price)) + geom_point() + stat_smooth(method = "lm", se = F) +labs
(title = "Price vs Bed", x = "Number of Beds", y = 'Price')
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

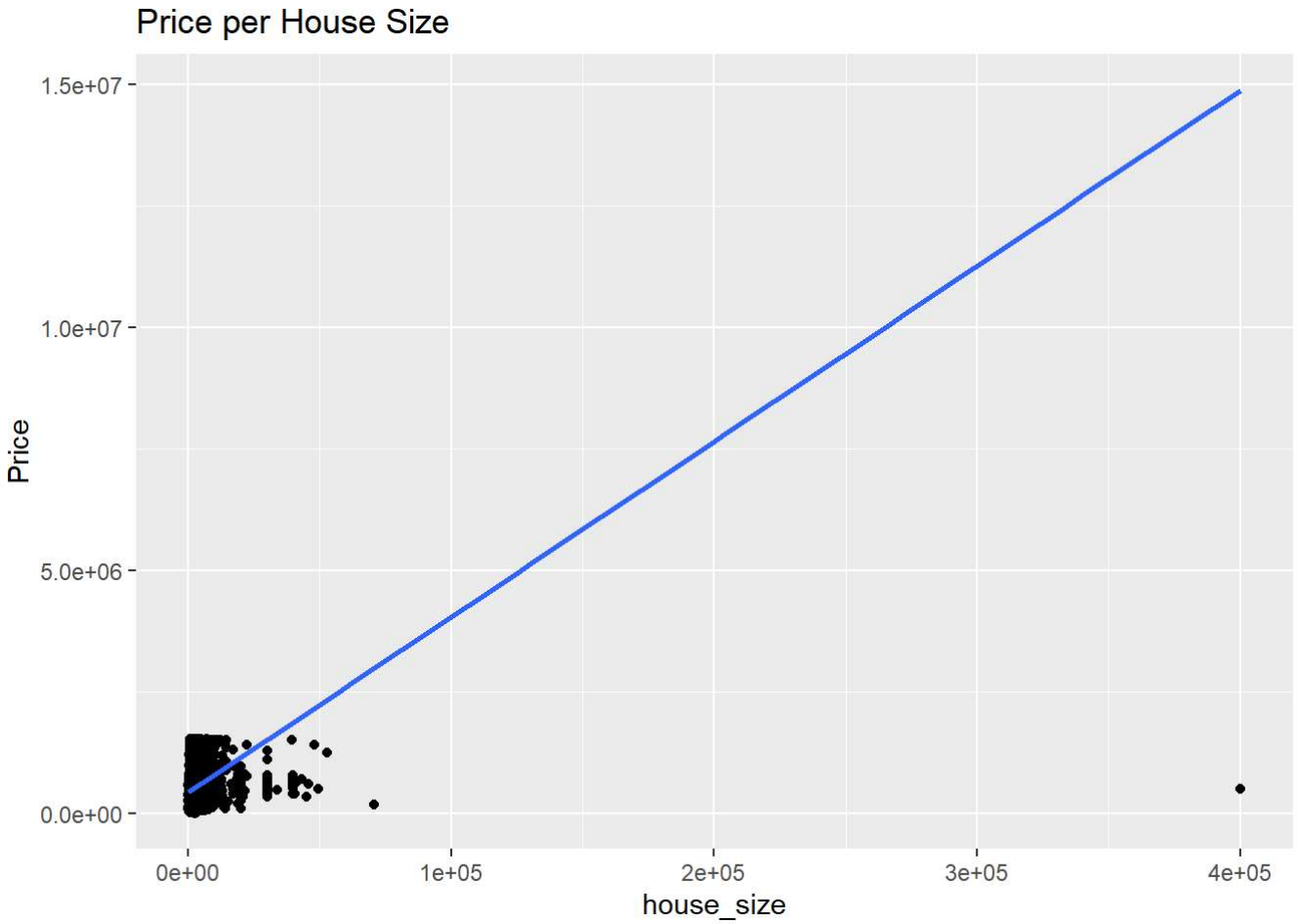


From the scatter plot, I see a positive linear relationship between `bed` and `price`. This means as the number of beds increase, the price increases, and vice versa. There are more outliers in the graph compared to the previous scatterplot. That explains why this relationship is less in magnitude.

#Scatterplot for Price per House_Size: Now lets do a scatter plot for `house_size` and `price`:

```
ggplot(house, aes(x = house_size, y = price)) + geom_point()  + stat_smooth(method = "lm", se = F) +labs(title = "Price per House Size", x = "house_size", y = 'Price')
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



From the scatter plot, we see a positive linear relationship between `house_size` and `price`. This means as house_size increases, the price increases, and vice versa. There are less outliers in the graph compared to the previous scatter plots.

Model Building:

After gaining an understanding about the relationships present in our data, now we move on to spliting, performing cross validation, and setting up the models. As mentioned, the metric I will use to measure the models performance is the Residual Mean Squared Error (RMSE). This metric is commonly used for accuracy; it involves calculating the residuals for each observation, taking the mean, and then taking the square root of the mean. Essentially, we are calculating the distance between the true observation point and prediction. We know if a model has performed well if its RMSE is severely low. However, before we conduct this process, we have to make sure our data is normal. We ensure our data is normal in the recipe building step by using the `step_scale` and `step_center` functions.

Stratified Sampling:

I will perform a 65/35 split on my data set. This means that 65% of the data goes to the training data set and 35% to the testing data set. I believe this split is fair as it gives enough observations to train my models on and test it as well. Our `strata` variable we will be splitting on is `price` The training data set will then be use to train my models. The testing data set will be used to measure the accuracy/performance of my models. Based on the value of the Root Mean Squared Error Metric when fitting the testing data set to the models, I Will be able to apply the most effective model to the testing data to see how accurate the model was in predicting the response.

```
#Setting a seed to reproduce our results
set.seed(12345)
housesplit <- initial_split(house, prop = 0.65, strata = price )
housetrain <- training(housesplit)
housetest <- testing(housesplit)
```

Lets see the number of observations in the training data set:

```
nrow(housetrain)
```

```
## [1] 244539
```

Our training data set contains 244539 observations

Lets see the number of observations in the testing data set:

```
nrow(housetest)
```

```
## [1] 131677
```

The testing data has 131677 observations.

Checking if our training and testing data comply with appropriate proportions:

Lets check if our data was actually split by the proportion we specified. We can do this by diving the number of observations in the testing and training data set by the total number of observations in the original data set. If the proportion calculated is equal to the proportion specified in the splitting portion, we are free to continue.

```
#For the training data set:
nrow(housetrain)/nrow(house)
```

```
## [1] 0.6499963
```

```
#For the testing data set:
nrow(housetest)/nrow(house)
```

```
## [1] 0.3500037
```

Nice! We can see that our training and testing data set were split by the proportions we specified during stratified sampling.

Setting up and applying the recipe:

Now, we will set up our recipe to apply later to our models. I won't include `division` in our recipe because our problem is a regression problem. Thus, it won't be needed as we are predicting house prices in general and not according to a division. With `division` excluded in the recipe, that leaves us with the following variables as our

predictors: `bed` , `bath` , `house_size` , `sq_lot` . As mentioned, our response variable is `price` . After I have set up our recipe, I will then use the `step_center()` and `step_scale()` functions on our predictors to ensure that the variables have a normal distribution with mean of 0 and standard deviation of 1. This will ensure the assumptions are met for the RMSE metric.

```
houserecipe <-  recipe(price ~ bed + bath + house_size + sq_lot, data = housetrain)
houserecipee <- houserecipe %>%
  step_center(all_numeric_predictors()) %>%
  step_scale(all_numeric_predictors())
```

K-Fold Cross Validation:

Now, I will perform K-Fold Cross Validation on the training data set. This resampling method involves R randomly splitting each observation in the training into one of the k subsets of equal size. The value of k is of choice. However, it is a convenient choice to choose 10 or 5. I will choose 10 as the value of the k. The reason why I chose 10 is because the variance is lower and it can help reduce the chances of overfitting. Anyways, after we have specified our value for k, the first subset of each model type is treated as the validation set, and the remaining k-1 subsets are a training set for that fold. Then, the MSE of the 1st fold are computed on the observations in the held-out fold. We repeat this process for 10 folds with each different, a different fold is being used as the validation set. Afterwards, the average accuracy of each fold testing set is calculated to determine the metrics and performance of the model.

We use K-fold Cross-Validation than comparing our model results on the entire training set because it measures a model's performance before being applied on the testing data set and is less biased. Also, it helps with reducing over fitting of the data.

```
house_folds <- vfold_cv(housetrain, v = 10)
house_folds
```

```
## #  10-fold cross-validation
## # A tibble: 10 × 2
##    splits                 id
##    <list>                 <chr>
##  1 <split [220085/24454]> Fold01
##  2 <split [220085/24454]> Fold02
##  3 <split [220085/24454]> Fold03
##  4 <split [220085/24454]> Fold04
##  5 <split [220085/24454]> Fold05
##  6 <split [220085/24454]> Fold06
##  7 <split [220085/24454]> Fold07
##  8 <split [220085/24454]> Fold08
##  9 <split [220085/24454]> Fold09
## 10 <split [220086/24453]> Fold10
```

Setting up the models:

After performing K-Fold Cross Validation on the training data set, We can now begin setting up models! Each model will follow a similar procedure. One important thing to note is that I won't be running the models here to save time. Thus, I have provided the code for each model and the procedure below: 1. Specify the model you wish to fit,

tune the parameters you wish to tune, the engine the model comes from, and if necessary the mode (regression or classification). Note: I won't be tuning the parameters for the linear regression model because the model is simple and doesn't have any parameters to tune. 2. Set up the workflow for the model and add the model and recipe. 3. Create a tuning grid to specify the parameter ranges you wish to tune as well as the number of levels of each. 4. Tune the model and specify the workflow, k-fold cross validation folds, and the tuning grid for the chosen parameters to tune. 5. Write and save each tuned model to a RDS File to be able to interact with the model and avoid rerunning it. 6. Read in the saved RDS File. 7. Collect the metrics of each tuned model, order it from lowest to highest RMSE, and then select the tuned model with the lowest RMSE from each model type, and place it in a table for comparisions.

1., 2., 3. Setting up the models, grid, and workflow. Tuning parameters:

K-NN Model:

```
knn_mod_cv <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")

knn_wkflow_cv <- workflow() %>%
  add_model(knn_mod_cv) %>%
  add_recipe(houserecipee)

grid_knn <- grid_regular(neighbors(range = c(10, 40)) , levels = 4)
```

Linear Regression Model:

```
# Does not require tuning of a parameter so we do not have to construct a grid
lm_mod <- linear_reg() %>%
  set_mode("regression") %>%
  set_engine("lm")

lm_wkflow <- workflow() %>%
  add_model(lm_mod) %>%
  add_recipe(houserecipee)
```

Ridge Regression

```
ridger <- linear_reg(mixture = 0,
                         penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

ridge_workflow <- workflow() %>%
  add_recipe(houserecipee) %>%
  add_model(ridger)

grid_r <- grid_regular(penalty(range = c(-3, 3)), levels = 10)
```

Lasso Regression:

```
lasso_spec <- linear_reg(penalty = tune(),
                         mixture = 1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

lasso_workflow <- workflow() %>%
  add_recipe(houserecipee) %>%
  add_model(lasso_spec)

#Using Same Grid as Ridge
grid_r
```

4. Tuning the models:

K-NN Models:

```
kntune <- tune_grid(knn_wkflow_cv,
    resamples = house_folds,
    control = control_grid(verbose = TRUE),
    grid = grid_knn)
```

Linear Regression Model:

```
lrfit <- fit_resamples(lm_wkflow, resamples = house_folds)
```

Ridge Regression:

```
ridge_tune <- tune_grid(ridge_workflow,
  resamples = house_folds,
  control = control_grid(verbose = TRUE),
  grid = grid_r)
```

Lasso Regression:

```
lasso_tune <- tune_grid(
  lasso_workflow,
  resamples = house_folds,
  grid = grid_r
)
```

5. Write to RDS File:

KNN:

```
write_rds(kntune, file = "knn.rds")
```

Ridge Regression:

```
write_rds(ridge_tune, file = "rr.rds")
```

Lasso Regression:

```
write_rds(lasso_tune, file = "lasso.rds")
```

6. Reading in RDS Files:

KNN

```
knn_tuned <- readRDS(file = "knn.rds")
```

Ridge Regression:

```
rr_tuned <- readRDS(file = "rr.rds")
```

Lasso Regression:

```
lassotune <- readRDS(file = "lasso.rds")
```

7. Collecting RMSE from each model:

I will now extract the RMSE value from each model. Then, I will select the best performing model of each model type and place them in a table for comparision. Whichever model had the lowest RMSE is our best performing model and will be used to fit the testing data. KNN:

```
knnp <- collect_metrics(knn_tuned)
#RMSE mean and std.error across all folds
knnp <- subset(knnp, .metric == 'rmse')
knnp
```

```
## # A tibble: 4 × 7
##   neighbors .metric .estimator    mean     n std_err .config
##       <int> <chr>   <chr>        <dbl> <int>   <dbl> <chr>
## 1        10 rmse    standard   118000.    10    438. Preprocessor1_Model1
## 2        20 rmse    standard   146283.    10    506. Preprocessor1_Model2
## 3        30 rmse    standard   167408.    10    469. Preprocessor1_Model3
## 4        40 rmse    standard   182198.    10    457. Preprocessor1_Model4
```

```
#Selecting Best Model:
knnpb <- knnp %>% arrange(mean) %>% slice (1)
```

Linear Regression:

```
lrp <- collect_metrics(lrfit, metric = "rmse")
lrp <- subset(lrp, .metric == 'rmse')
lrp
```

```
## # A tibble: 1 × 6
##   .metric .estimator    mean     n std_err .config
##   <chr>   <chr>        <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   276046.    10    646. Preprocessor1_Model1
```

Ridge Regression:

```
ridger <- collect_metrics(rr_tuned)
#RMSE mean and std.error across all folds
rp <- subset(ridger, .metric == 'rmse')
rp
```

```
## # A tibble: 10 × 7
##         penalty .metric .estimator    mean     n std_err .config
##           <dbl> <chr>   <chr>        <dbl> <int>   <dbl> <chr>
##  1       0.01   rmse    standard   276195.    10    643. Preprocessor1_Model01
##  2       0.0774 rmse    standard   276195.    10    643. Preprocessor1_Model02
##  3       0.599  rmse    standard   276195.    10    643. Preprocessor1_Model03
##  4       4.64   rmse    standard   276195.    10    643. Preprocessor1_Model04
##  5      35.9    rmse    standard   276195.    10    643. Preprocessor1_Model05
##  6     278.     rmse    standard   276195.    10    643. Preprocessor1_Model06
##  7    2154.     rmse    standard   276195.    10    643. Preprocessor1_Model07
##  8   16681.     rmse    standard   276229.    10    643. Preprocessor1_Model08
##  9  129155.     rmse    standard   279948.    10    634. Preprocessor1_Model09
## 10 1000000      rmse    standard   296044.    10    644. Preprocessor1_Model10
```

```
#Selecting Best Model:
ridger <- rp %>% arrange(mean) %>% slice (1)
```

Lasso Regression:

```
las <- collect_metrics(lassotune)
#RMSE mean and std.error across all folds
lp <- subset(las, .metric == 'rmse')
lp
```

```
## # A tibble: 10 × 7
##       penalty .metric .estimator    mean     n std_err .config
##         <dbl> <chr>   <chr>        <dbl> <int>   <dbl> <chr>
##  1    0.001   rmse    standard   276046.    10    647. Preprocessor1_Model01
##  2    0.00464 rmse    standard   276046.    10    647. Preprocessor1_Model02
##  3    0.0215  rmse    standard   276046.    10    647. Preprocessor1_Model03
##  4    0.1     rmse    standard   276046.    10    647. Preprocessor1_Model04
##  5    0.464   rmse    standard   276046.    10    647. Preprocessor1_Model05
##  6    2.15    rmse    standard   276046.    10    647. Preprocessor1_Model06
##  7   10       rmse    standard   276046.    10    647. Preprocessor1_Model07
##  8   46.4     rmse    standard   276046.    10    647. Preprocessor1_Model08
##  9  215.      rmse    standard   276046.    10    647. Preprocessor1_Model09
## 10 1000       rmse    standard   276054.    10    647. Preprocessor1_Model10
```

```
#Selecting Best Model:
lasr <- lp %>% arrange(mean) %>% slice (1)
```

Table of RMSE;

```
results <- tibble(Model = c("KNN", "Linear Regression", "Ridge Regression", "Lasso Regression"),
RMSE = c(knnpb$mean, lrp$mean, ridger$mean, lasr$mean))

resultsorder <- results %>%
  arrange(RMSE)

resultsorder
```

```
## # A tibble: 4 × 2
##   Model               RMSE
##   <chr>              <dbl>
## 1 KNN               118000.
## 2 Lasso Regression  276046.
## 3 Linear Regression 276046.
## 4 Ridge Regression  276195.
```

From the table above, we can see that the KNNModel10 with 10 nearest neighbors performed the best! The smaller the value of RMSE, the better the model preformed. The RMSE yielded is relatively smaller than the other models.

Fitting the Best Model to Training:

Now, we will fit the best model to the entire training data set. This will be done by first selecting the best nearest neighbors from the KNN Model and assigning it to a placeholder variable. Then we can set up a a finalize workflow with the KNN Model's workflow with the best nearest neighbors. Afterwards, we fit the workflow with the entire training data set and save it as an RDS File to avoid rerunning the model.

```
final_fits <- readRDS(file = "final_train.rds")
```

Testing the Best Model:

Now for the moment we have all been waiting for! Measuring the models true performance! We can use the fitted workflow model designed previously by loading it into the `augment()` function with the testing data set and computing the RMSE of the testing data set for the estimation of `price`. Based on this value, we can see how well our best model did on the testing data set!

```
augment(final_fits, new_data = housetest) %>%
  rmse(truth = price, estimate = .pred)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard     114515.
```

As you can see, the rmse of the KNNModel10 with the testing data yielded a value of 114514.6. This value is relatively close to the rmse of the model with the training data! The training data rmse is a little higher than the testing data. However, the difference isn't too big to say that overfitting has occurred. I am pretty satisfied with the

models performance.

The indication of the testing data being relatively similar means our model did good!

Conclusion:

I first began the project by tidying and inspecting the data to look for any signs of inconsistency with the labels and units of the variables. Then, I selected five of the variables present in the data set that I believe would be most useful for the EDA portion. I, also, determined if there were missing values present within the data set. Indeed, there was a lot of missing observations for the variables. I was a bit hesitant to remove these observations as I am losing information that could be useful to make inferences about the data. However, I noticed that the number of observations would still be relatively large with removing these values. I decided to take the risk and remove the values.

From the EDA portion, we saw the distribution of the outcome variable and how its distribution varies depending on the predictor. One thing that stood out to me was the fact that each numeric variable had a positive correlation with the `price`. The relationships in some variables weren't as strong as the others, but overall, each variable had a positive linear relationship with `price`. From the correlation plot, the numeric variable that had the strongest positive linear correlation with `price` was `bath`. This correlation indicates that `bath` is the most significant factor when determining the price of a house. I did not expect the correlation between these two variables to be as strong. I expected `house_size` to be the most influential factor in determining the house price because generally, the bigger the house, the more space it holds and the more labor is involved.

In the models portion of the project, we created a recipe to apply to our models. Then, we fit the models to the training data set. This process was very tedious as it took a long time for the models to process. One thing I wish could have done to increase the model's processing time is to reduce the number of observations. However, this poses the risk of over fitting or underfitting the data. Also, I believe I could have used other regression models, such as random forest or boosted tress, but the processing time was very slow. However, I did manage to successfully yield the RMSE values of the regression models I implemented. The RMSE values were very large, indicating the model is not a really good fit. Even so, I believe that the outliers did affect the performance, so I decided to continue on.

Overall, all of the models I embedded did not do a very good job when fitting with the training data. We can see that each model yield a very high RMSE. However, this was expected due to the outliers present in each predictor. After removing the outliers for the response variable, I had a very tough decision to make in the beginning when deciding whether or not to remove these outliers for the predictors. I decided not to remove them because losing these observations would make it difficult to make generalizations of the data. Also, I had already lost about 50% of the original sample size when removing the missing values present. Therefore, removing more observations would make it difficult to see the variation in the data. If I were to remove these outliers, the bias would increase. The distribution for each predictor without the outliers would be affected, as well as skewing the distribution.

The KNNModel10 we performed the best and it was awesome to see when applied to the testing data set, no signs of overfitting or underfitting were present. I wish the RMSE values were smaller but performing more tidying on the data poses the risk of not being to make the generalizations we have made in the EDA section.

Ultimately, this project was a great experience! It allowed me to gain hands on experience of the tasks that are required for my desired career position. I hope to use the information found in the models and EDA portion to complete one of my goals in life, purchasing a house for my family.