US Airbnb Open Data Project Report

Noam Kadosh

December 17th, 2020

Contents

1 Overview

This project is part of the HarvardX course PH125.9x Data Science: Capstone project. When Airbnb was founded in 2008, the idea of sleeping in other people's house was somewhat strange. Despite that, Airbnb grew larger each year with increasing number of listings and users each year. There was a real advantage to that idea which was the real catalyst for the company's growth. People understood it is cheaper than the classic approach.

1.1 Project Introduction

In this project I will explore the US Airbnb Open dataset and try to predict the prices of Airbnb listings, both new listings and existing listings. The model can fit a property management company who wants to know the right price of a listings. With the model, a company can find undervalued listings and try to offer them services or purchase the listing. The price of listing is a continuous variable. Therefore, regression algorithms are the go to tool. I will train several regression algorithms and one ensemble algorithm. Than, I will measure their performance by the Root Mean Squared Error (RMSE). The goal is to get the RMSE as low as possible.

1.1.1 Libraries

```
# Loading libraries
repos = "http://cran.us.r-project.org"
if(!require(tidyverse)) install.packages("tidyverse", repos = repos)
if(!require(tidytext)) install.packages("tidytext", repos = repos)
if(!require(caret)) install.packages("caret", repos = repos)
if(!require(stringr)) install.packages("stringr", repos = repos)
if(!require(scales)) install.packages("scales", repos = repos)
if(!require(lubridate)) install.packages("lubridate", repos = repos)
if(!require(rpart)) install.packages("rpart", repos = repos)
if(!require(rpart.plot)) install.packages("rpart.plot", repos = repos)
if(!require(randomForest)) install.packages("randomForest", repos = repos)
if(!require(kernlab)) install.packages("kernlab", repos = repos)
if(!require(fastDummies)) install.packages("fastDummies", repos = repos)
if(!require(lubridate)) install.packages("lubridate", repos = repos)
if(!require(corrplot)) install.packages("corrplot", repos = repos)
if(!require(wordcloud)) install.packages("wordcloud", repos = repos)
```

1.2 The Data

The data used for the project is the US Airbnb Open data set, collected by Kritik Seth, a data scientist who collected and published it on Kaggle. The data set contains features such as location, room type, description, number of reviews, minimum nights and more.

```
# Loading the data set
airbnb <- read_csv("Data/AB_US_2020.csv", col_types = cols(
  neighbourhood = col_character(),
  neighbourhood_group = col_character()
))

head(airbnb)
```

```
## # A tibble: 6 x 17
##       id name  host_id host_name neighbourhood_g~ neighbourhood latitude
##    <dbl> <chr>   <dbl> <chr>     <chr>            <chr>            <dbl>
## 1  38585 Char~  165529 Evelyne   <NA>             28804             35.7
## 2  80905 Fren~  427027 Celeste   <NA>             28801             35.6
## 3 108061 Walk~  320564 Lisa      <NA>             28801             35.6
## 4 155305 Cott~  746673 BonPaul   <NA>             28806             35.6
## 5 160594 Hist~  769252 Elizabeth <NA>             28801             35.6
## 6 209068 Terr~ 1029919 Kevin     <NA>             28804             35.6
## # ... with 10 more variables: longitude <dbl>, room_type <chr>, price <dbl>,
## #   minimum_nights <dbl>, number_of_reviews <dbl>, last_review <chr>,
## #   reviews_per_month <dbl>, calculated_host_listings_count <dbl>,
## #   availability_365 <dbl>, city <chr>
```

```
summary(airbnb)
```

```
##        id                name              host_id           host_name
##  Min.   :     109   Length:226030      Min.   :       23   Length:226030
##  1st Qu.:15158904   Class :character   1st Qu.: 13992754   Class :character
##  Median :25909160   Mode  :character   Median : 51382662   Mode  :character
##  Mean   :25471759                      Mean   : 93523849
##  3rd Qu.:37726244                      3rd Qu.:149717884
##  Max.   :45560850                      Max.   :367917574
##
##  neighbourhood_group neighbourhood         latitude       longitude
##  Length:226030       Length:226030      Min.   :18.92   Min.   :-159.71
##  Class :character    Class :character   1st Qu.:32.76   1st Qu.:-118.60
##  Mode  :character    Mode  :character   Median :37.26   Median : -97.82
##                                         Mean   :35.66   Mean   :-103.22
##                                         3rd Qu.:40.72   3rd Qu.: -76.92
##                                         Max.   :47.73   Max.   : -71.00
##
##   room_type             price         minimum_nights     number_of_reviews
##  Length:226030      Min.   :    0.0   Min.   :1.00e+00   Min.   :  0.00
##  Class :character   1st Qu.:   75.0   1st Qu.:1.00e+00   1st Qu.:  1.00
##  Mode  :character   Median :  121.0   Median :2.00e+00   Median :  8.00
##                     Mean   :  219.7   Mean   :4.53e+02   Mean   : 34.51
##                     3rd Qu.:  201.0   3rd Qu.:7.00e+00   3rd Qu.: 39.00
##                     Max.   :24999.0   Max.   :1.00e+08   Max.   :966.00
##
##  last_review        reviews_per_month calculated_host_listings_count
##  Length:226030      Min.   : 0.01     Min.   :  1.0
##  Class :character   1st Qu.: 0.23     1st Qu.:  1.0
##  Mode  :character   Median : 0.81     Median :  2.0
##                     Mean   : 1.43     Mean   : 16.7
##                     3rd Qu.: 2.06     3rd Qu.:  6.0
##                     Max.   :44.06     Max.   :593.0
##                     NA's   :48602
##  availability_365     city
##  Min.   :  0.0    Length:226030
##  1st Qu.:  0.0    Class :character
##  Median :140.0    Mode  :character
##  Mean   :159.3
##  3rd Qu.:311.0
```

```
##  Max.   :365.0
##
```

```
length(airbnb$price)
```

```
## [1] 226030
```

```
airbnb %>% summarize(num_properties = n_distinct(id), num_hosts = n_distinct(host_id))
```

```
## # A tibble: 1 x 2
##   num_properties num_hosts
##            <int>     <int>
## 1         226029    130425
```

1.2.1 Data Preparation

We can see that some of the columns contain NAs. We should try and understand why those NAs are there. Looking at the neighbourhood_group column we can infer that those NAs are other neighbourhood_group that are unknown to us. We can transform those NAs to "other". When looking at the reviews_per_month and the number_of_reviews columns we can see that the NAs in the reviews_per_month are actually 0. All those NAs have 0 reviews overall. The last_review column will be converted to numeric representing days since the epoch (January 1st 1970). Listings with NA value will get the value 0, meaning no review was ever recorded. We will also filter out listings with minimum nights of more than 365. To handle the rest of the NAs in the data, I decided to omit them since they are a very small subset of the data (just about 60 rows). Finally, I will add the State column based on the City column.

```
# Changing NAs in neighbourhood_group column to "other".
airbnb$neighbourhood_group[is.na(airbnb$neighbourhood_group)] <- "other"

# Changing NAs in reviews_per_month column to 0.
airbnb$reviews_per_month[is.na(airbnb$reviews_per_month)] <- 0

# Converting last_review column to numeric
airbnb <- airbnb %>%
  mutate(last_review = as.numeric(as.Date(airbnb$last_review, "%d/%m/%y")))
airbnb$last_review[is.na(airbnb$last_review)] <- 0

# Filtering listings with more than a year stay minimum.
airbnb <- airbnb %>%
  filter(minimum_nights < 365)

# Removing all NAs
airbnb <- na.omit(airbnb)

# Adding the State column
temp <- tibble(city = c("Asheville", "Austin", "Boston", "Broward County", "Cambridge",
                        "Chicago", "Clark County", "Columbus", "Denver", "Hawaii",
                        "Jersey City", "Los Angeles", "Nashville", "New Orleans",
                        "New York City", "Oakland", "Pacific Grove", "Portland",
                        "Rhode Island", "Salem", "San Clara Country",
                        "Santa Cruz County", "San Diego", "San Francisco",
                        "San Mateo County", "Seattle", "Twin Cities MSA",
```

```
                       "Washington D.C."),
              state = c("NC", "TX", "MA", "FL", "MA", "IL", "NV", "OH", "CO", "HI",
                        "NJ", "CA", "TN", "LA", "NY", "CA", "CA", "OR", "RI", "MA",
                        "CA", "CA", "CA", "CA", "CA", "WA", "MN", "DC"))
airbnb_states <- sapply(airbnb$city, function(x) {
  temp$state[which(temp$city == x)]
})
airbnb <- airbnb %>%
  mutate(state = airbnb_states)
```

A summary of the data after preparation.

```
head(airbnb)
```

```
## # A tibble: 6 x 18
##       id name  host_id host_name neighbourhood_g~ neighbourhood latitude
##    <dbl> <chr>   <dbl> <chr>     <chr>            <chr>            <dbl>
## 1  38585 Char~  165529 Evelyne   other            28804             35.7
## 2  80905 Fren~  427027 Celeste   other            28801             35.6
## 3 108061 Walk~  320564 Lisa      other            28801             35.6
## 4 155305 Cott~  746673 BonPaul   other            28806             35.6
## 5 160594 Hist~  769252 Elizabeth other            28801             35.6
## 6 209068 Terr~ 1029919 Kevin     other            28804             35.6
## # ... with 11 more variables: longitude <dbl>, room_type <chr>, price <dbl>,
## #   minimum_nights <dbl>, number_of_reviews <dbl>, last_review <dbl>,
## #   reviews_per_month <dbl>, calculated_host_listings_count <dbl>,
## #   availability_365 <dbl>, city <chr>, state <chr>
```

```
summary(airbnb)
```

```
##        id                name              host_id           host_name
##  Min.   :     109   Length:225717      Min.   :       23   Length:225717
##  1st Qu.:15171583   Class :character   1st Qu.: 14003785   Class :character
##  Median :25916331   Mode  :character   Median : 51433847   Mode  :character
##  Mean   :25477725                      Mean   : 93545510
##  3rd Qu.:37735023                      3rd Qu.:149734533
##  Max.   :45560850                      Max.   :367917574
##  neighbourhood_group neighbourhood        latitude       longitude
##  Length:225717       Length:225717      Min.   :18.92   Min.   :-159.71
##  Class :character    Class :character   1st Qu.:32.76   1st Qu.:-118.60
##  Mode  :character    Mode  :character   Median :37.26   Median : -97.82
##                                         Mean   :35.66   Mean   :-103.23
##                                         3rd Qu.:40.72   3rd Qu.: -76.92
##                                         Max.   :47.73   Max.   : -71.00
##   room_type             price         minimum_nights    number_of_reviews
##  Length:225717      Min.   :    0.0   Min.   :  1.000   Min.   :  0.00
##  Class :character   1st Qu.:   75.0   1st Qu.:  1.000   1st Qu.:  1.00
##  Mode  :character   Median :  121.0   Median :  2.000   Median :  8.00
##                     Mean   :  219.5   Mean   :  9.616   Mean   : 34.54
##                     3rd Qu.:  201.0   3rd Qu.:  7.000   3rd Qu.: 39.00
##                     Max.   :24999.0   Max.   :364.000   Max.   :966.00
##   last_review    reviews_per_month calculated_host_listings_count
```

```
##  Min.   :    0   Min.   : 0.000    Min.   :  1.00
##  1st Qu.:17209   1st Qu.: 0.040    1st Qu.:  1.00
##  Median :18264   Median : 0.430    Median :  2.00
##  Mean   :14291   Mean   : 1.125    Mean   : 16.72
##  3rd Qu.:18413   3rd Qu.: 1.630    3rd Qu.:  6.00
##  Max.   :18529   Max.   :44.060    Max.   :593.00
##  availability_365     city              state
##  Min.   :  0.0    Length:225717      Length:225717
##  1st Qu.:  0.0    Class :character   Class :character
##  Median :140.0    Mode  :character   Mode  :character
##  Mean   :159.3
##  3rd Qu.:311.0
##  Max.   :365.0
```

```
length(airbnb$price)
```

```
## [1] 225717
```

```
airbnb %>% summarize(num_properties = n_distinct(id), num_hosts = n_distinct(host_id))
```

```
## # A tibble: 1 x 2
##   num_properties num_hosts
##            <int>     <int>
## 1         225716    130216
```

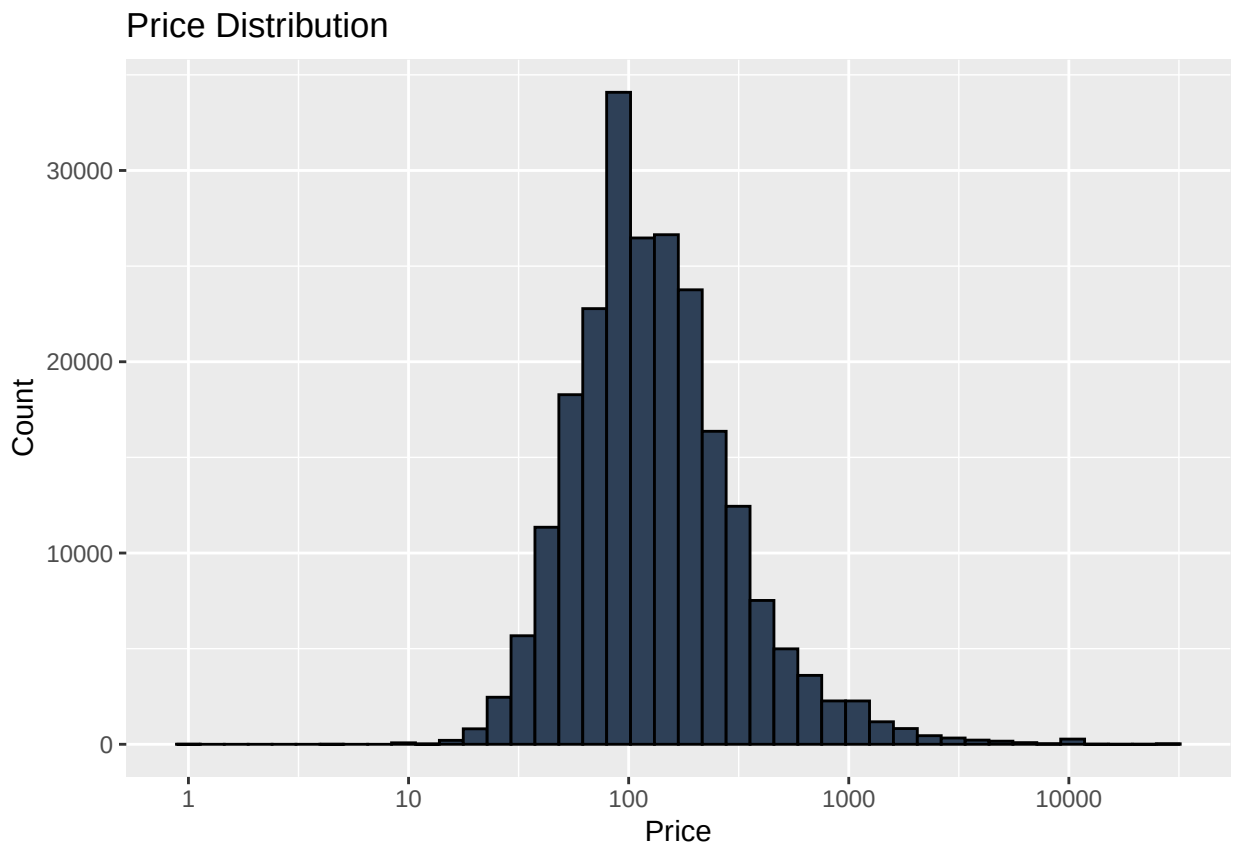
1.2.2 Data Analysis

When observing the price distribution on log scale, we can see the majority of the data is around $100.

```
# Price Distribution
airbnb %>%
  ggplot(aes(price)) +
  geom_histogram(binwidth = 0.25, color = "black", fill = "#2e4057") +
  scale_x_continuous(name = "Price",trans = "log", breaks = c(1, 10, 100, 1000, 10000)) +
  ylab("Count") +
  ggtitle("Price Distribution")
```

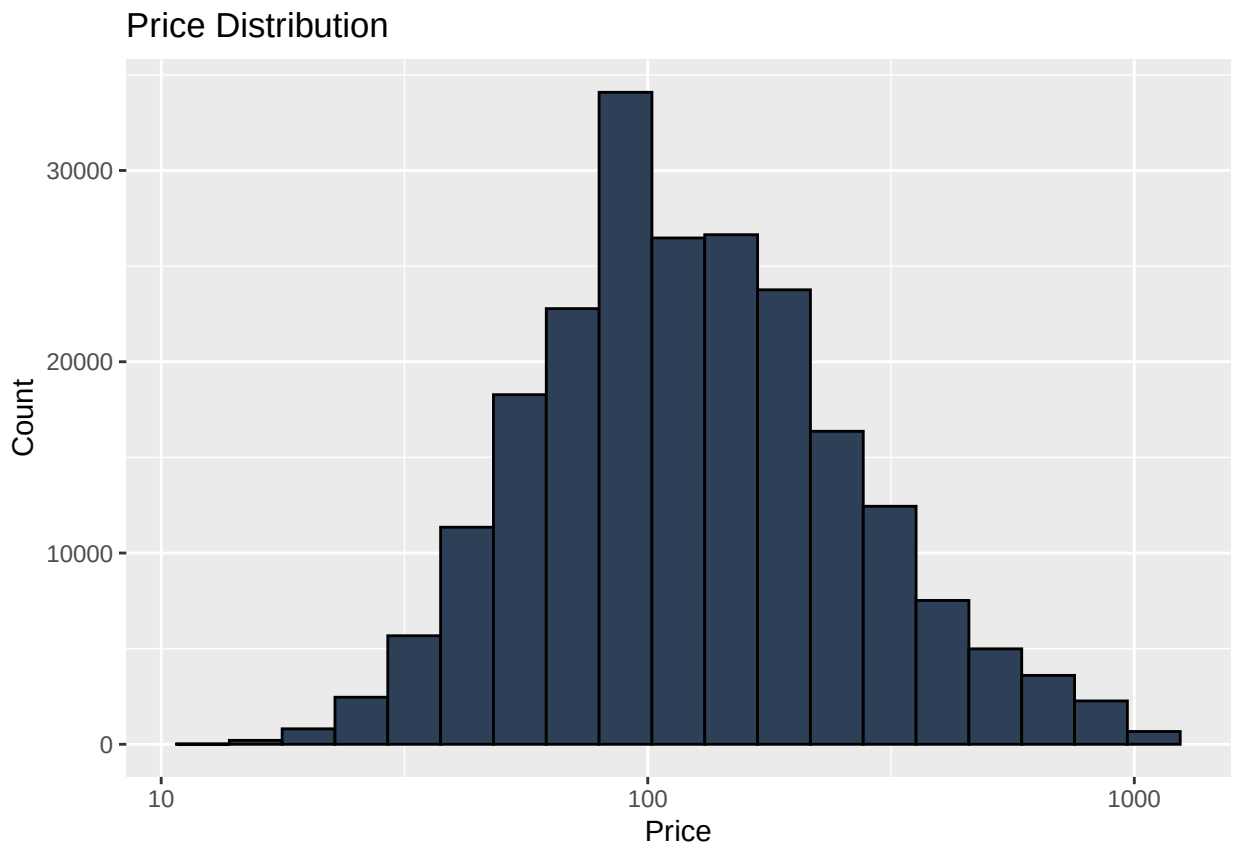
```
## Warning: Transformation introduced infinite values in continuous x-axis
```

```
## Warning: Removed 62 rows containing non-finite values (stat_bin).
```



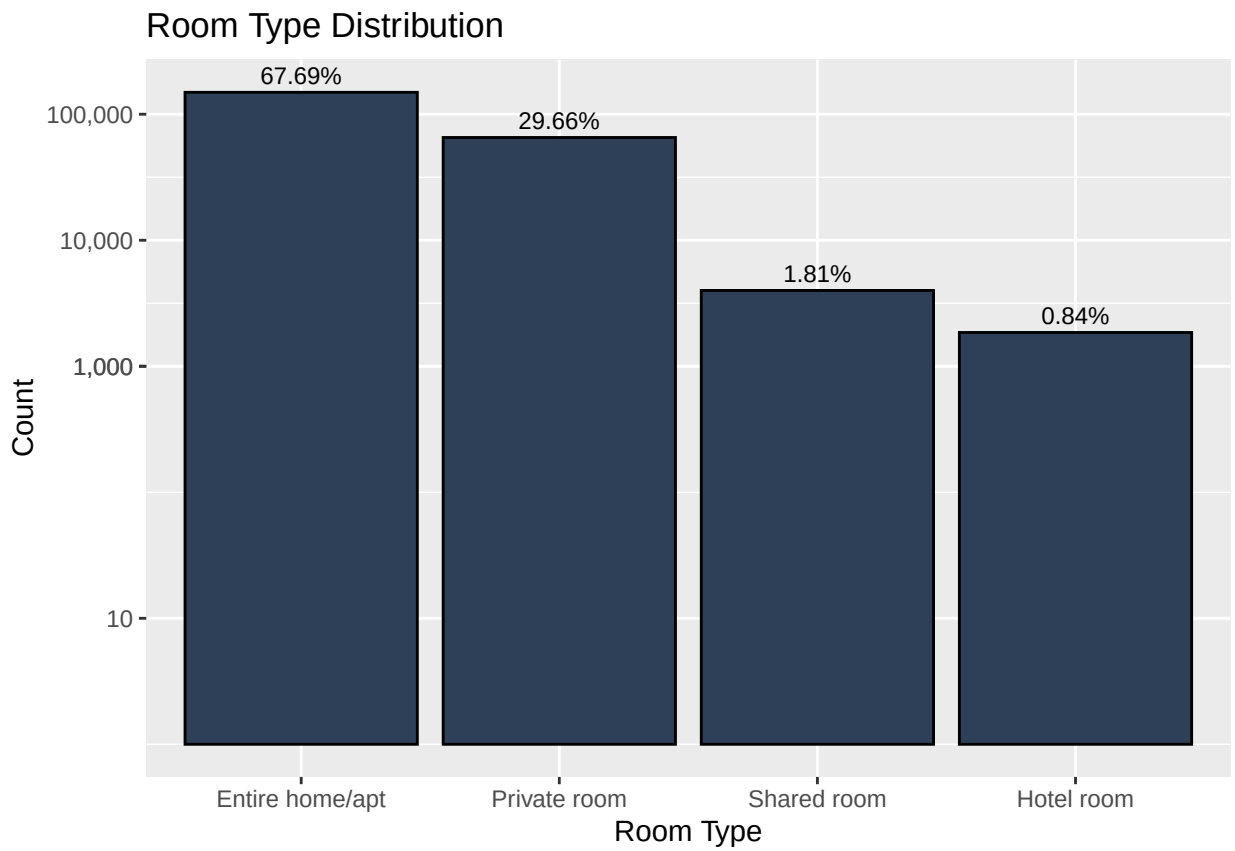
Because of this, we can safely remove extreme outliers, as they will probably hurt our models (more on that later). Note that we only lose about 5000 rows.

```
# Get rid of extreme outliers.
airbnb <- airbnb %>%
  filter(price > 10 & price < 1000)
# Price Distribution
airbnb %>%
  ggplot(aes(price)) +
  geom_histogram(binwidth = 0.25, color = "black", fill = "#2e4057") +
  scale_x_continuous(name = "Price",trans = "log", breaks = c(10, 100, 1000)) +
  ylab("Count") +
  ggtitle("Price Distribution")
```

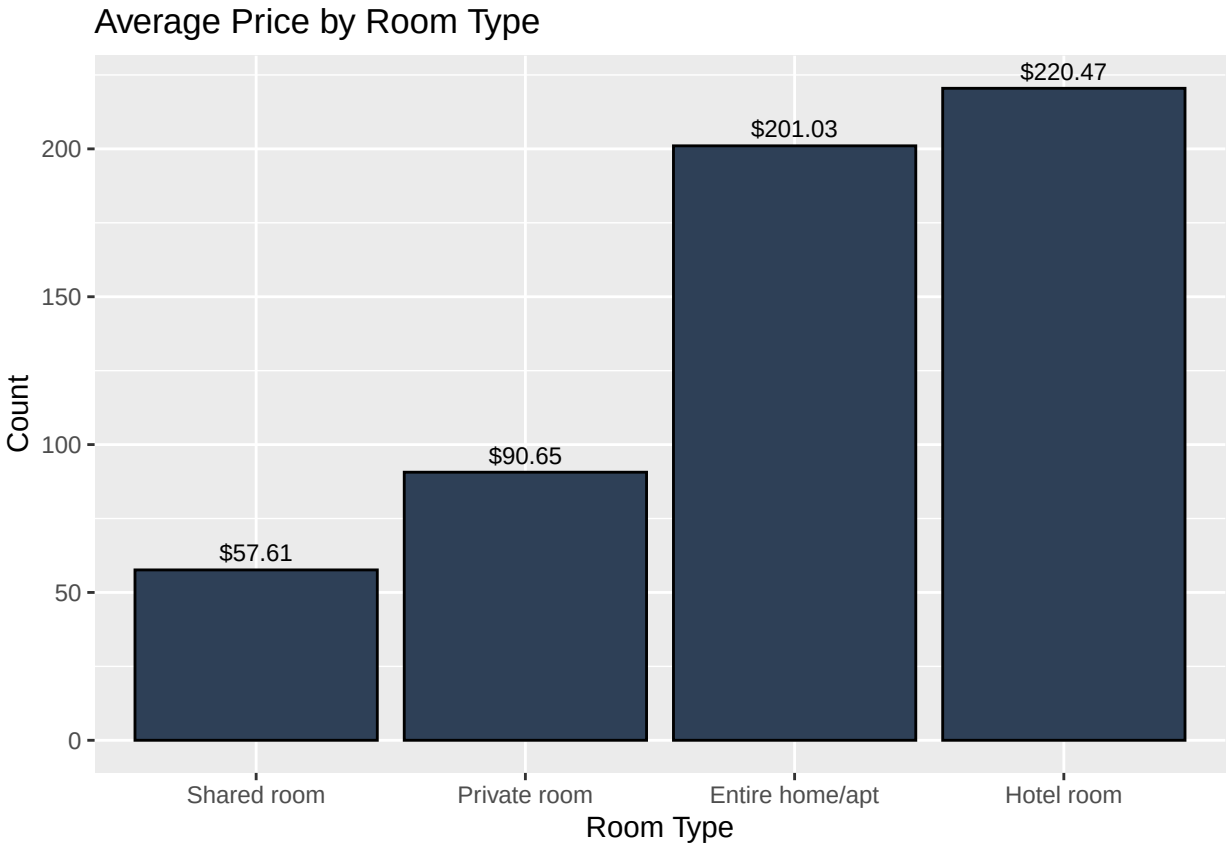


Now let's have a look at our features. The room_type column is a categorical feature with four values: entire home, private room, shared room, and hotel room. Airbnb's niche is about renting from other people, which explains the fact that hotel rooms are so rare in the data. A shared room, hostel style, is also very rare in the dataset. Looking at the average price by room type, we can see what we would expect as hotel rooms are most expensive, followed closely by entire homes, and than private rooms and shared rooms with cheaper prices.

```
# Room Type Distribution
airbnb$room_type <- fct_infreq(airbnb$room_type)
airbnb %>%
  ggplot(aes(room_type, label = percent(prop.table(stat(count))))) +
  geom_bar(color = "black", fill = "#2e4057") +
  geom_text(stat = "count", position = position_dodge(width = .9), vjust = -0.5,
            size = 3) +
  xlab("Room Type") +
  scale_y_continuous(name = "Count", trans = "log", labels = comma,
                     breaks = c(10, 1000, 1000, 10000, 100000)) +
  ggtitle("Room Type Distribution")
```




```r
# Average Price by Room Type
airbnb %>%
  group_by(room_type) %>%
  summarize(mean = mean(price), .groups = "drop") %>%
  ggplot(aes(x = reorder(room_type, mean), mean,
             label = dollar(round(mean, digits = 2)))) +
  geom_col(color = "black", fill = "#2e4057") +
  geom_text(position = position_dodge(width = .9), vjust = -0.5, size = 3) +
  xlab("Room Type") +
  scale_y_continuous(name = "Count", labels = comma) +
  ggtitle("Average Price by Room Type")
```

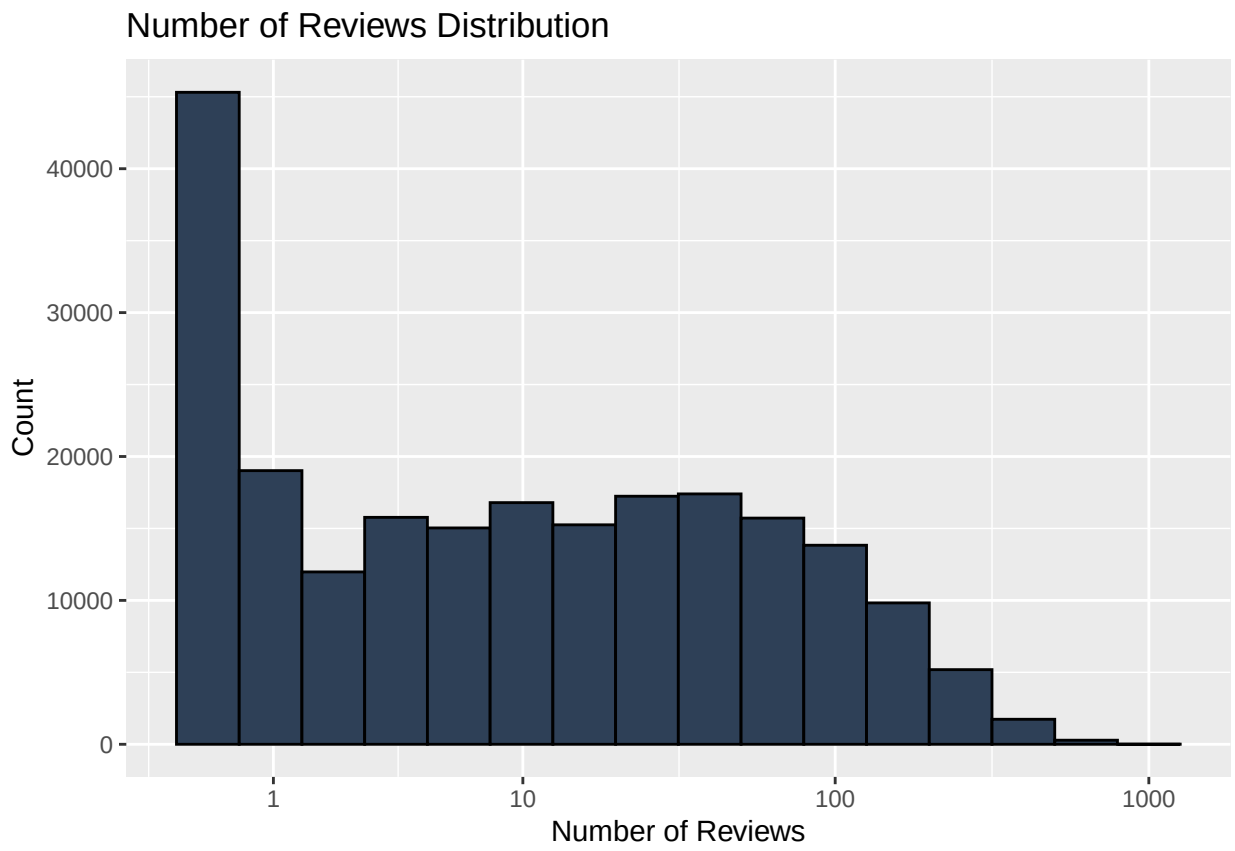


As we saw earlier, there is a connection between number of reviews and reviews per month columns. Their distributions also look very similar, with the majority of the data having 0 reviews (or reviews per month) and decreasing number of reviews as we go up the scale. These two columns together hold another important property which is the age of the listing. We can get that by the follwing formula:

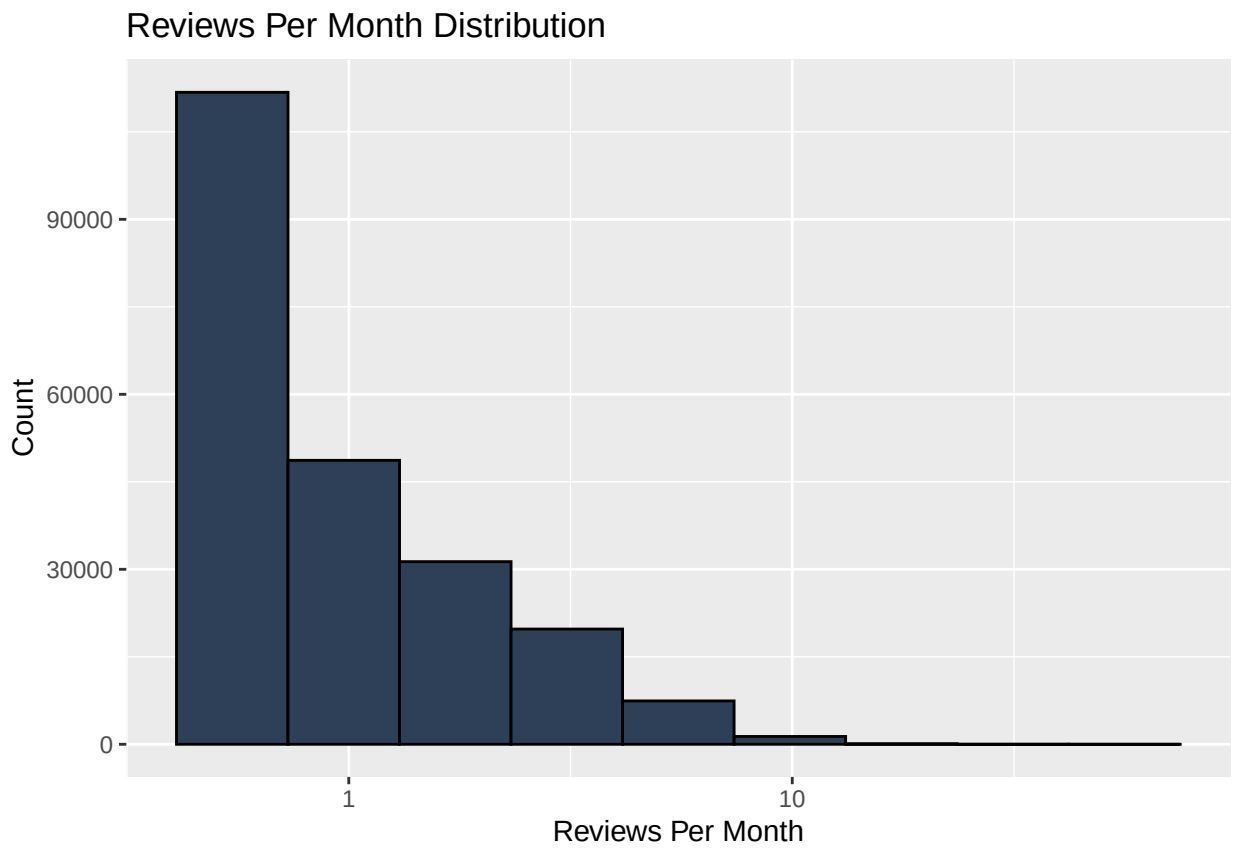
$$(NumberOfReviews/ReviewsPerMonth)/12 = Age$$

Unfortunately, rows with 0 are missing that information. Anyway, listings with 0 reviews can be either inactive, new, or just poorly managed (e.g. a 5 year old listing with 0 reviews).

```
# Number of Review Distribution
airbnb %>%
 ggplot(aes(number_of_reviews)) +
 geom_histogram(binwidth = 0.2, color = "black", fill = "#2e4057") +
 scale_x_continuous(name = "Number of Reviews",trans = pseudo_log_trans(base = 10),
                    breaks = c(1, 10, 100, 1000)) +
 ylab("Count") +
 ggtitle("Number of Reviews Distribution")
```

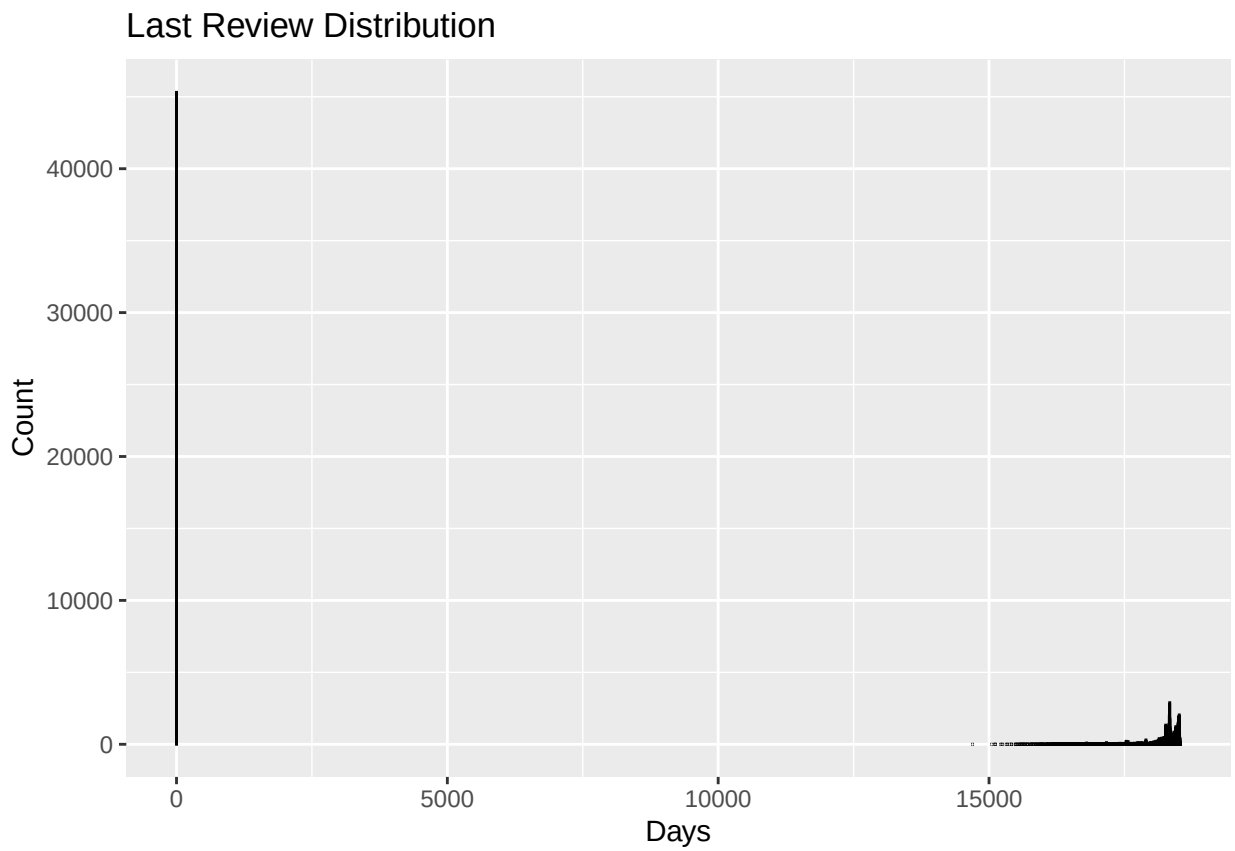



```
# Reviews Per Month Distribution
airbnb %>%
  ggplot(aes(reviews_per_month)) +
  geom_histogram(binwidth = 0.2, color = "black", fill = "#2e4057") +
  scale_x_continuous(name = "Reviews Per Month",trans = pseudo_log_trans(base = 10),
                     breaks = c(1, 10)) +
  ylab("Count") +
  ggtitle("Reviews Per Month Distribution")
```

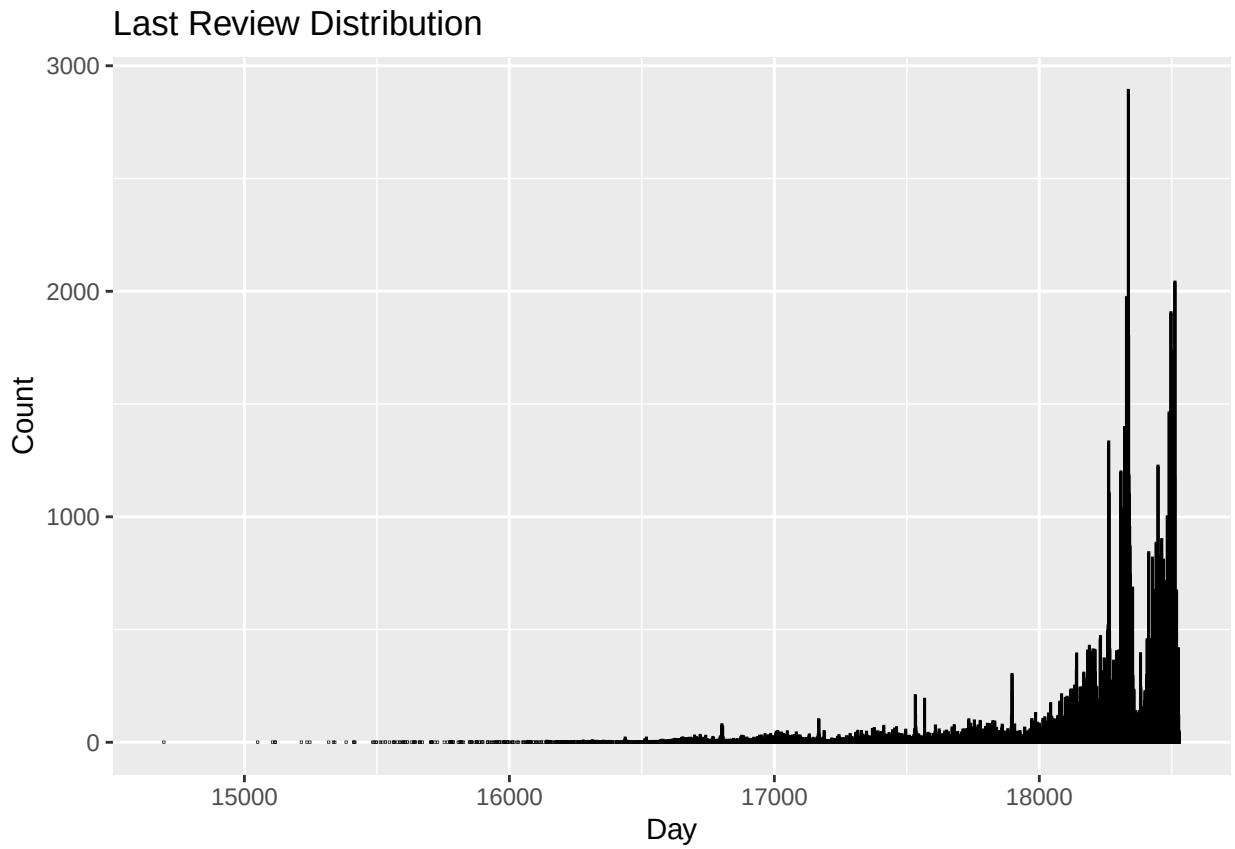


The last review distribution shows us that most of the listings were active recently, and very few got their last review years ago.

```
# Last Review Distribution
airbnb %>%
  ggplot(aes(last_review)) +
  geom_bar(stat = "count", width = 0.9, color = "black", fill = "#2e4057") +
  xlab("Days") +
  ylab("Count") +
  ggtitle("Last Review Distribution")
```

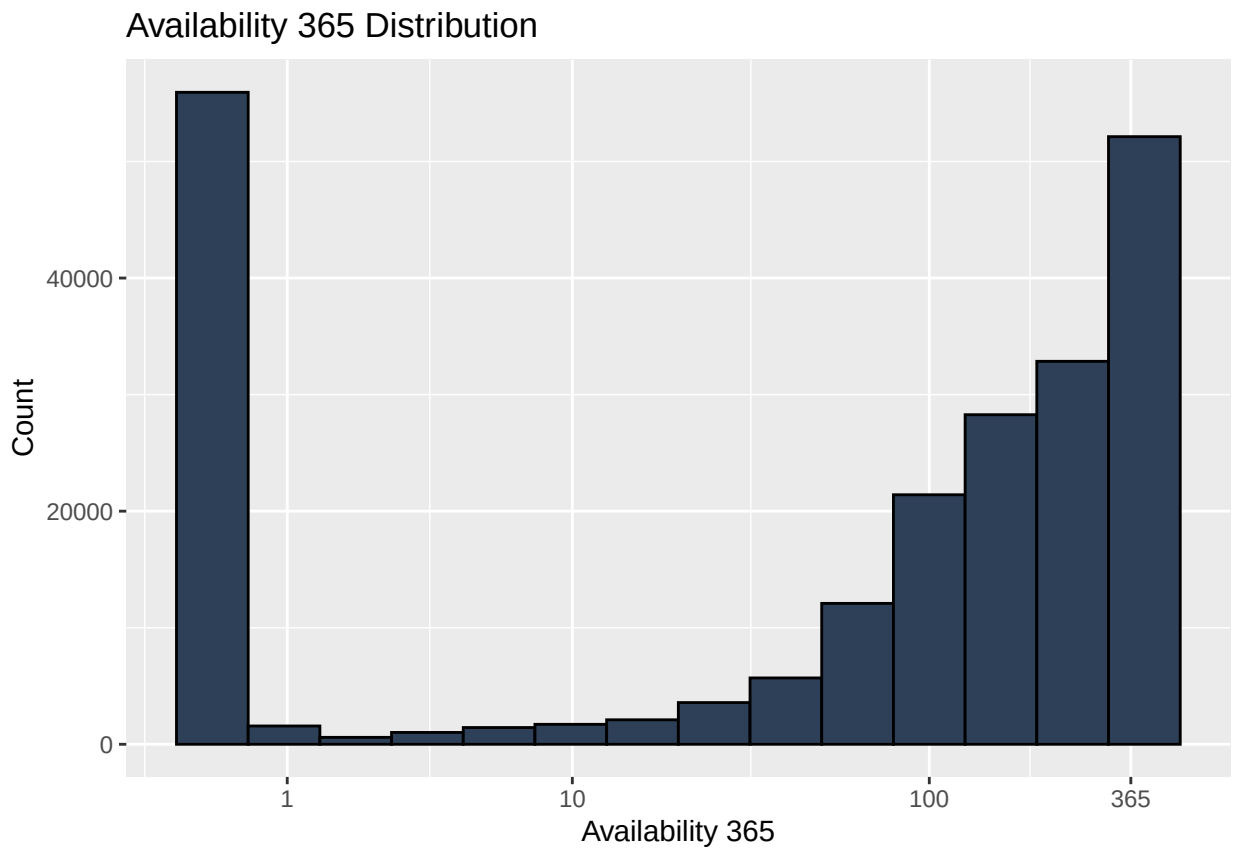



```
# Last Review Distribution - without 0's.
airbnb %>%
  filter(last_review > 0) %>%
  ggplot(aes(last_review)) +
  geom_bar(stat = "count", width = 0.9, color = "black", fill = "#2e4057") +
  xlab("Day") +
  ylab("Count") +
  ggtitle("Last Review Distribution")
```



Most of the listings in the dataset either have 0 availability or 365 (full year) availability. We can assume those with 0 availability are successful listings while those with full availability are either inactive or new. Some of the listings can be affected by seasonality, which explains why they are available for most of the year. The average listing availability by state plot shows us that Hawaii and Florida average the most availability. We know these state are highly touristic in the summer.

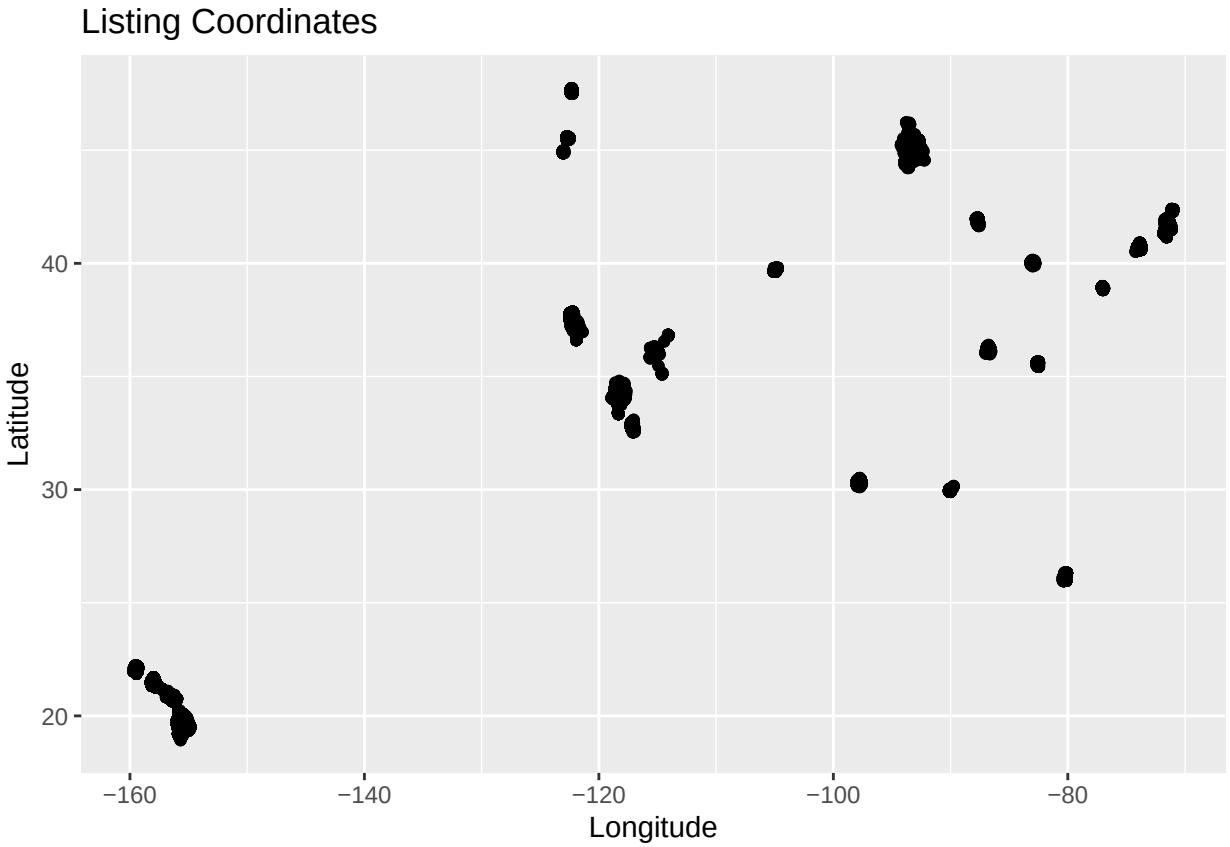
```
# Availability 365 Distribution
airbnb %>%
  ggplot(aes(availability_365)) +
  geom_histogram(binwidth = 0.2, color = "black", fill = "#2e4057") +
  scale_x_continuous(name = "Availability 365",trans = pseudo_log_trans(base = 10),
                     breaks = c(1, 10, 100, 365)) +
  ylab("Count") +
  ggtitle("Availability 365 Distribution")
```




```
# Average Listing Availability by State
airbnb %>%
  group_by(state) %>%
  summarize(mean = mean(availability_365), .groups = "drop") %>%
  ggplot(aes(x = reorder(state, mean), mean, label = round(mean, digits = 2))) +
  geom_col(color = "black", fill = "#2e4057") +
  geom_text(position = position_dodge(width = .9), hjust = -0.05, size = 3) +
  xlab("State") +
  scale_y_continuous(name = "Count", labels = comma) +
  ggtitle("Average Listing Availability by State") +
  coord_flip()
```

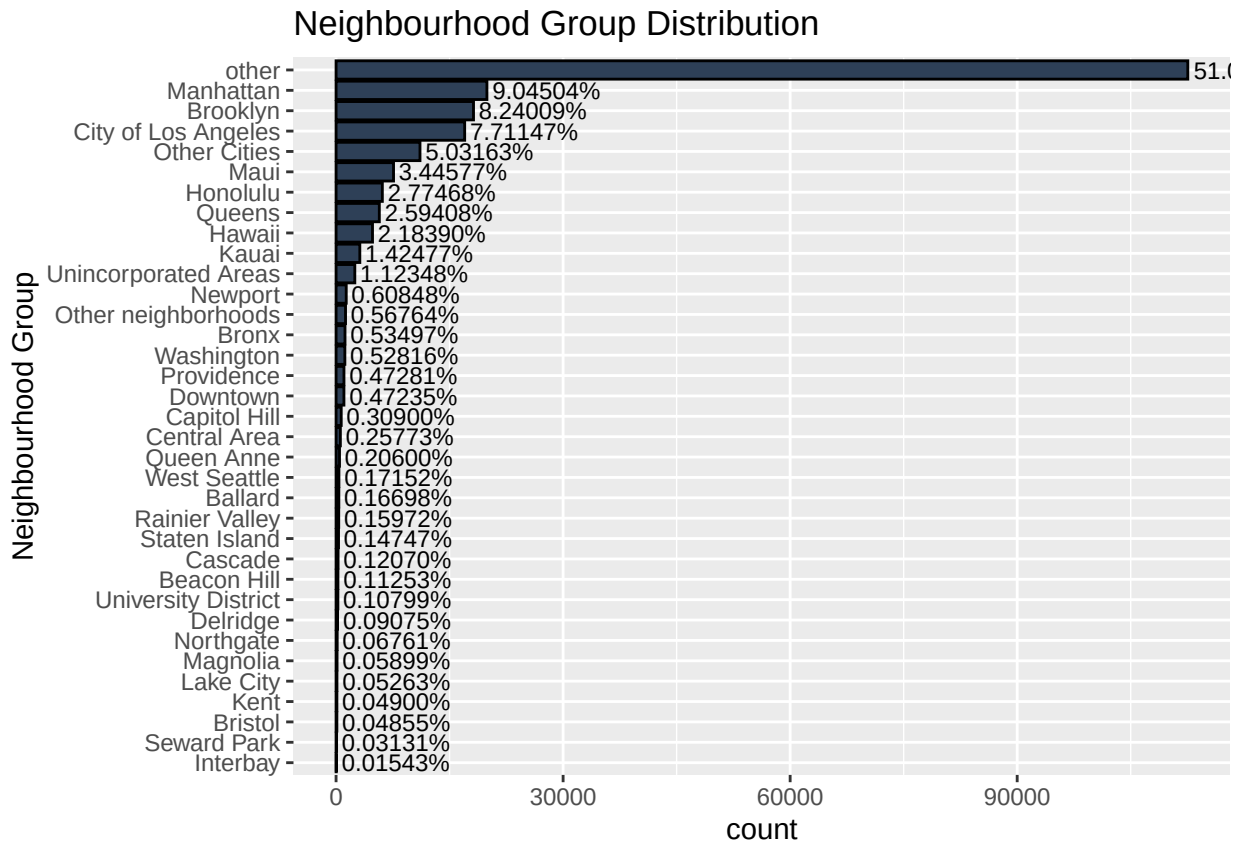
The longitude and latitude features give us the exact coordinates of every listing.

```
# Listings' earth coordinates
airbnb %>%
  ggplot(aes(x = longitude, y = latitude)) +
  geom_point(color = "black", fill = "#2e4057") +
  xlab("Longitude") +
  ylab("Latitude") +
  ggtitle("Listing Coordinates")
```

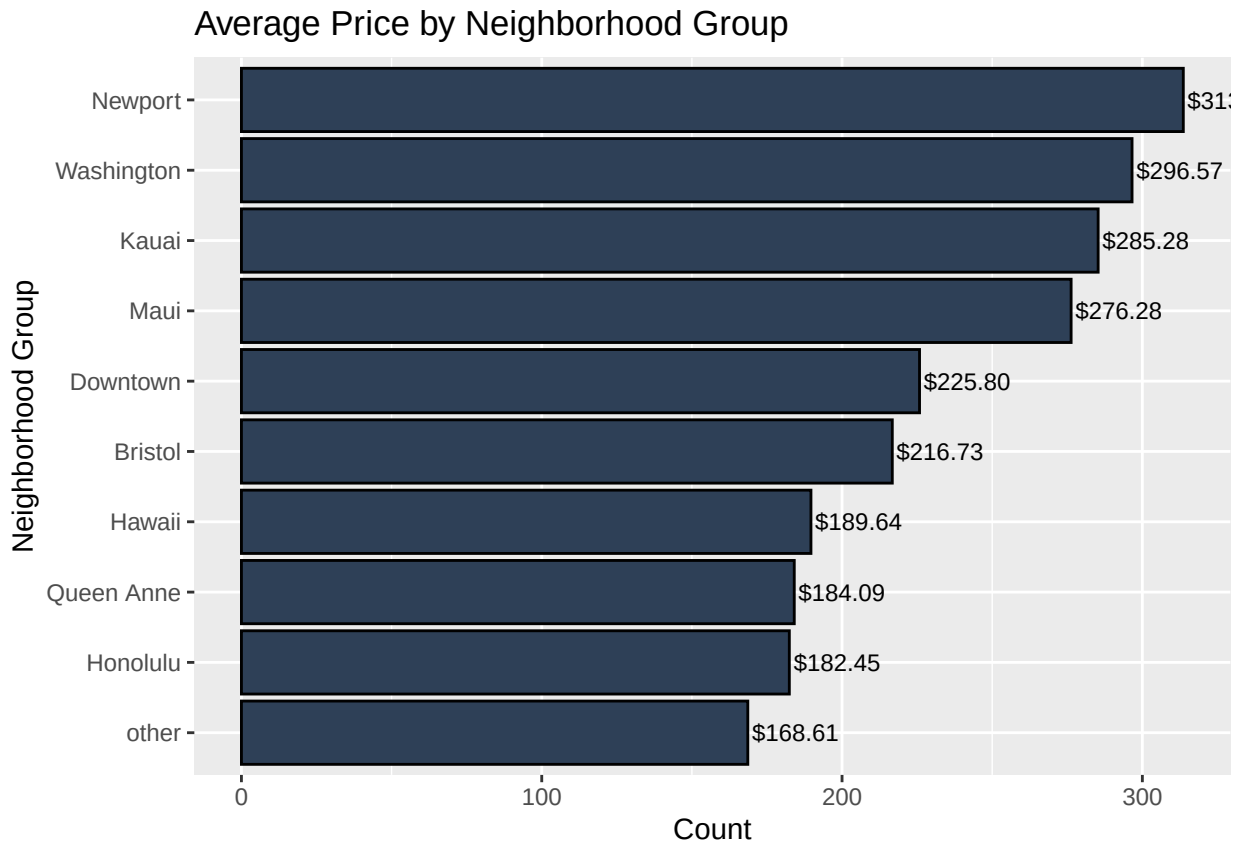


The neighborhood group, city, and state columns provide us with more broad information than the coordinates. There are variations between states, cities, and neighborhoods such as taxes, laws, and socioeconomic stature. As we can see New York, California, and Hawaii have the most listings. We can see the different average prices between top ten states, cities, and neighborhoods.

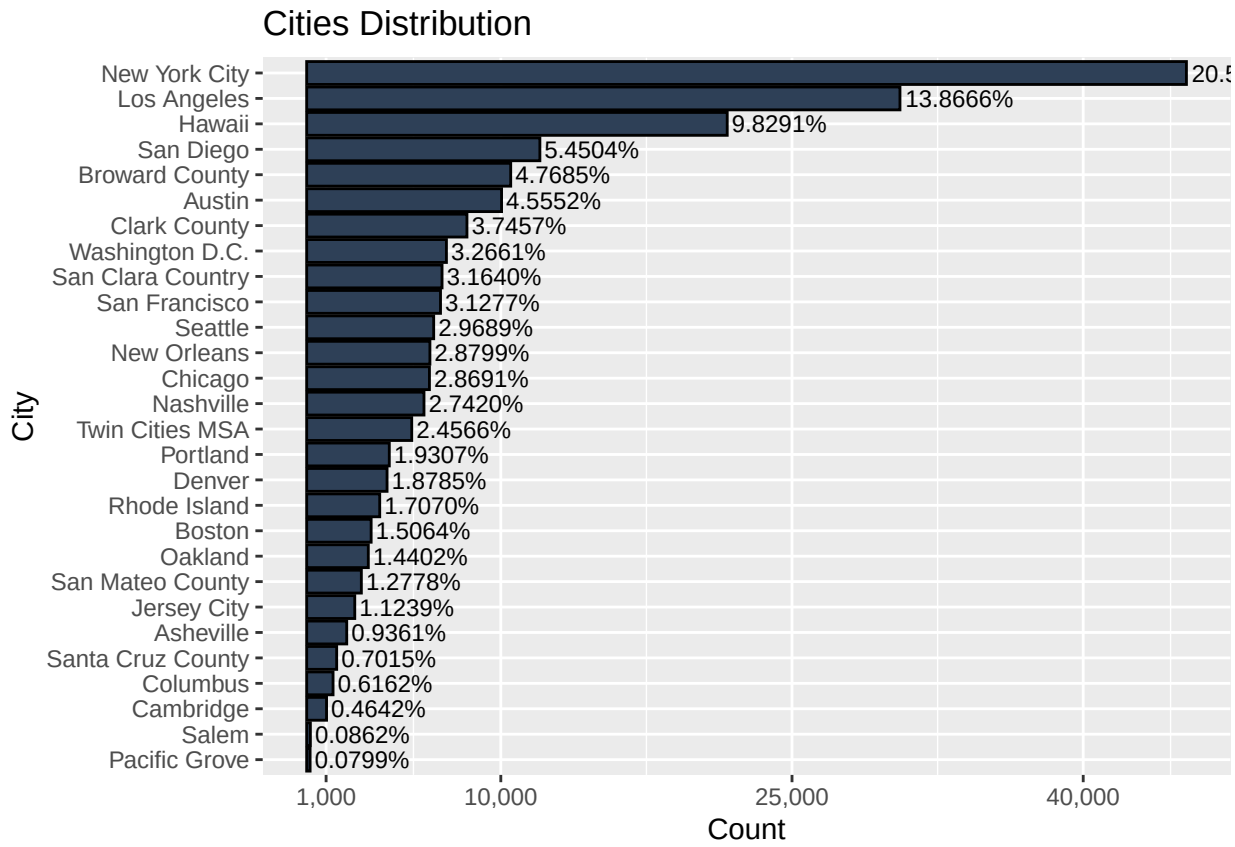
```
# Neighbourhood_group distribution
airbnb$neighbourhood_group <- fct_rev(fct_infreq(airbnb$neighbourhood_group))
airbnb %>%
  ggplot(aes(neighbourhood_group, label = percent(prop.table(stat(count))))) +
  geom_bar(color = "black", fill = "#2e4057") +
  geom_text(stat = "count", position = position_dodge(width = .9),
            hjust = -0.05, size = 3) +
  xlab("Neighbourhood Group") +
  ggtitle("Neighbourhood Group Distribution") +
  coord_flip()
```



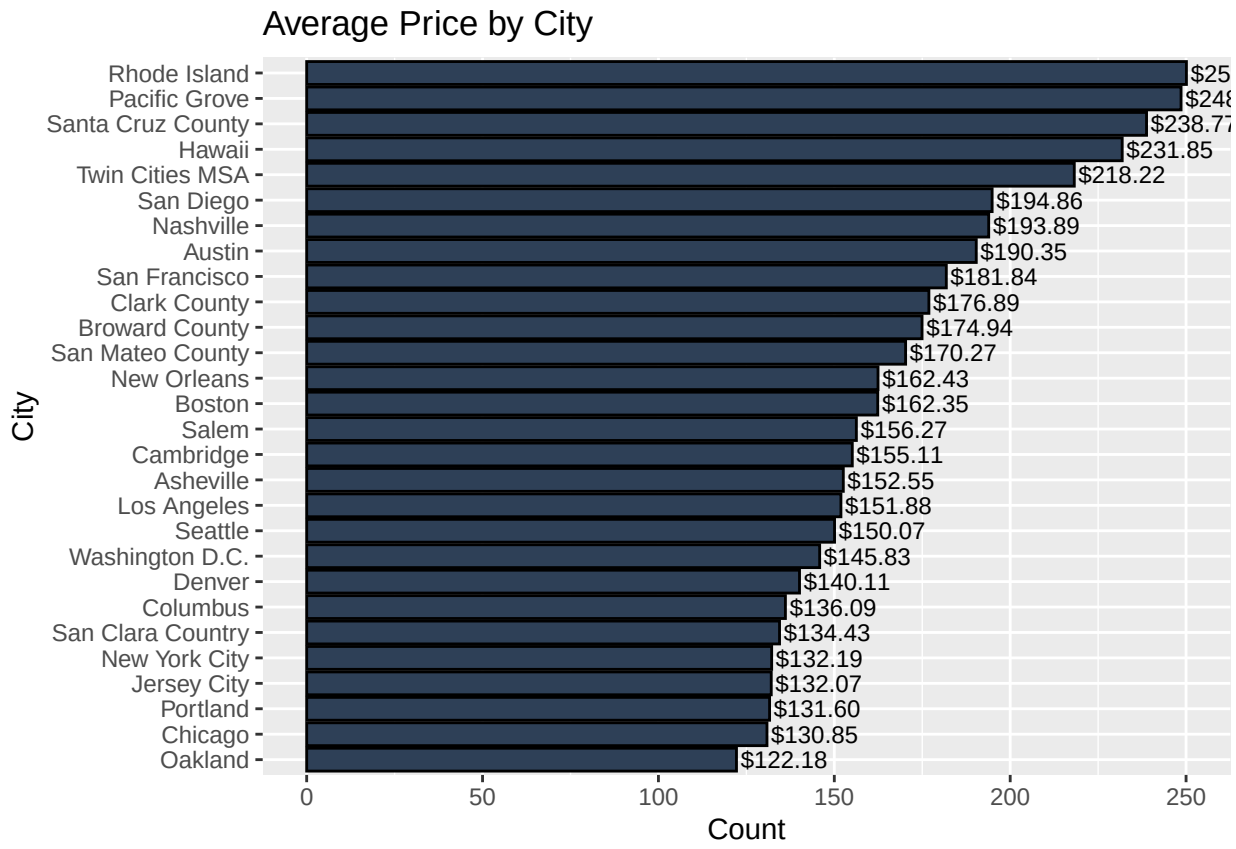

```r
# Average Price by Top 10 Neighborhood Group
airbnb %>%
  group_by(neighbourhood_group) %>%
  summarize(mean = mean(price), .groups = "drop") %>%
  slice_max(mean, n = 10) %>%
  ggplot(aes(x = reorder(neighbourhood_group, mean), mean,
             label = dollar(round(mean, digits = 2)))) +
  geom_col(color = "black", fill = "#2e4057") +
  geom_text(position = position_dodge(width = .9), hjust = -0.05, size = 3) +
  xlab("Neighborhood Group") +
  scale_y_continuous(name = "Count", labels = comma) +
  ggtitle("Average Price by Neighborhood Group") +
  coord_flip()
```



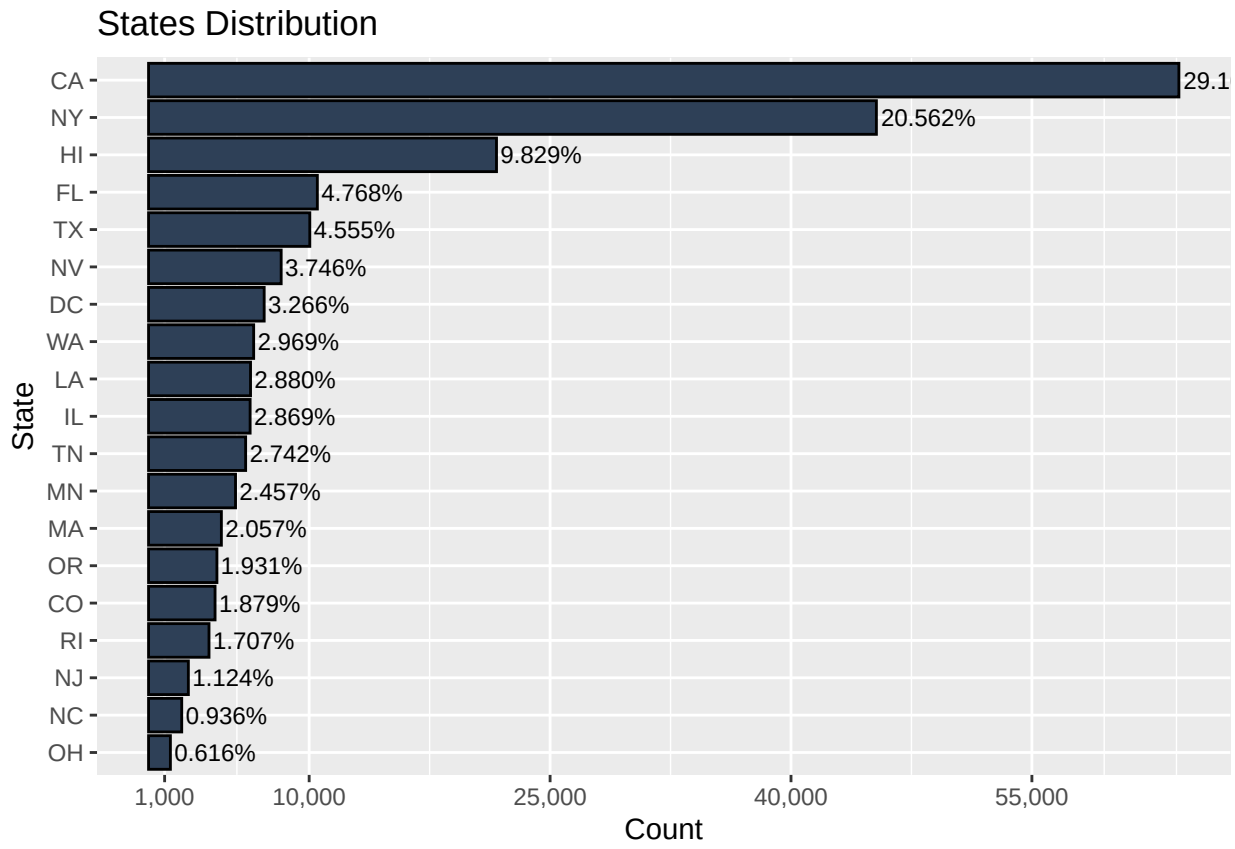

```
# Cities Distribution
airbnb %>%
  group_by(city) %>%
  mutate(count = n()) %>%
  ggplot(aes(x = reorder(city, count), label = percent(prop.table(stat(count))))) +
  geom_bar(color = "black", fill = "#2e4057") +
  geom_text(stat = "count", position = position_dodge(width = .9),
            hjust = -0.05, size = 3) +
  xlab("City") +
  scale_y_continuous(name = "Count", labels = comma,
                     breaks = c(1000, 10000, 25000, 40000)) +
  ggtitle("Cities Distribution") +
  coord_flip()
```



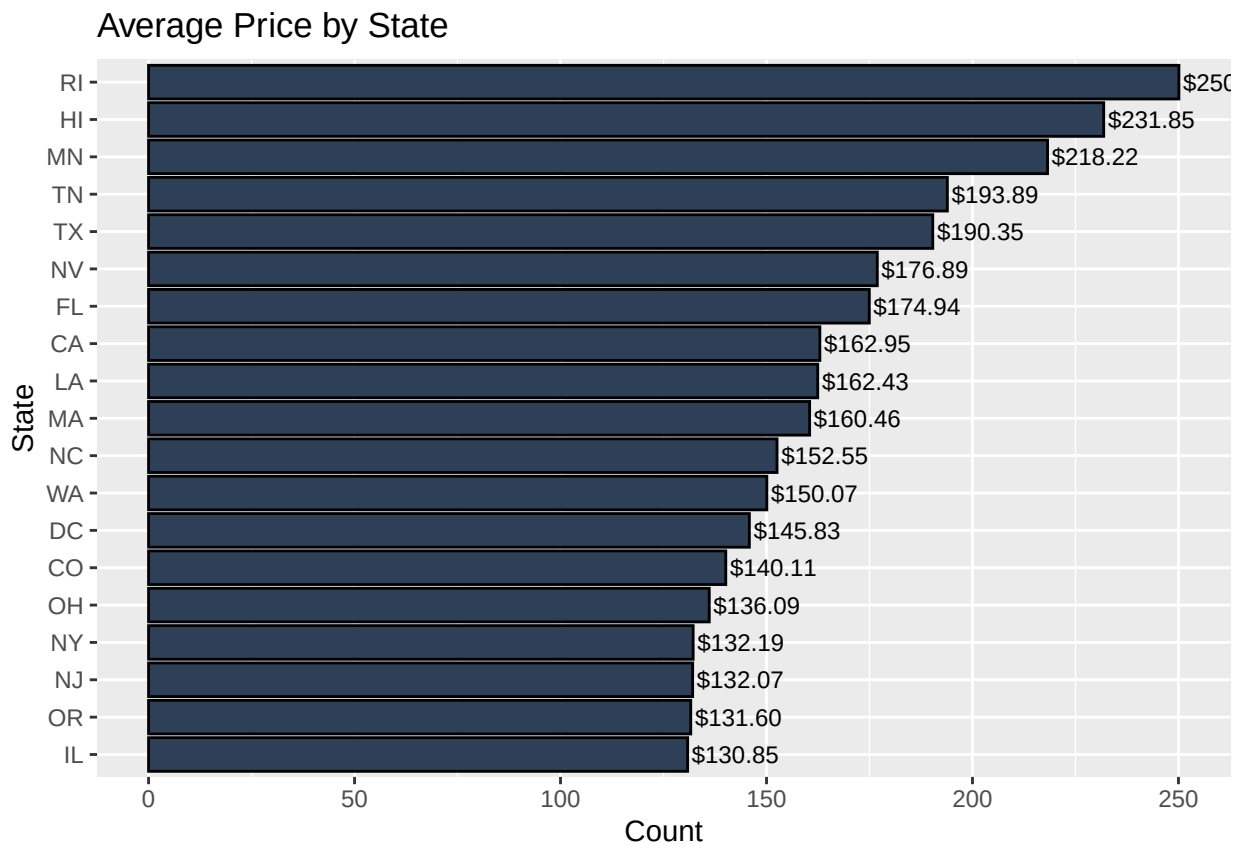

```
# Average Price by City
airbnb %>%
  group_by(city) %>%
  summarize(mean = mean(price), .groups = "drop") %>%
  ggplot(aes(x = reorder(city, mean), mean, label = dollar(round(mean, digits = 2)))) +
  geom_col(color = "black", fill = "#2e4057") +
  geom_text(position = position_dodge(width = .9), hjust = -0.05, size = 3) +
  xlab("City") +
  scale_y_continuous(name = "Count", labels = comma) +
  ggtitle("Average Price by City") +
  coord_flip()
```




```
# State Distribution
airbnb %>%
  group_by(state) %>%
  mutate(count = n()) %>%
  ggplot(aes(x = reorder(state, count), label = percent(prop.table(stat(count))))) +
  geom_bar(color = "black", fill = "#2e4057") +
  geom_text(stat = "count", position = position_dodge(width = .9),
            hjust = -0.05, size = 3) +
  xlab("State") +
  scale_y_continuous(name = "Count", labels = comma,
                     breaks = c(1000, 10000, 25000, 40000, 55000)) +
  ggtitle("States Distribution") +
  coord_flip()
```




```
# Average Price by State
airbnb %>%
  group_by(state) %>%
  summarize(mean = mean(price), .groups = "drop") %>%
  ggplot(aes(x = reorder(state, mean), mean,
             label = dollar(round(mean, digits = 2)))) +
  geom_col(color = "black", fill = "#2e4057") +
  geom_text(position = position_dodge(width = .9), hjust = -0.05, size = 3) +
  xlab("State") +
  scale_y_continuous(name = "Count", labels = comma) +
  ggtitle("Average Price by State") +
  coord_flip()
```



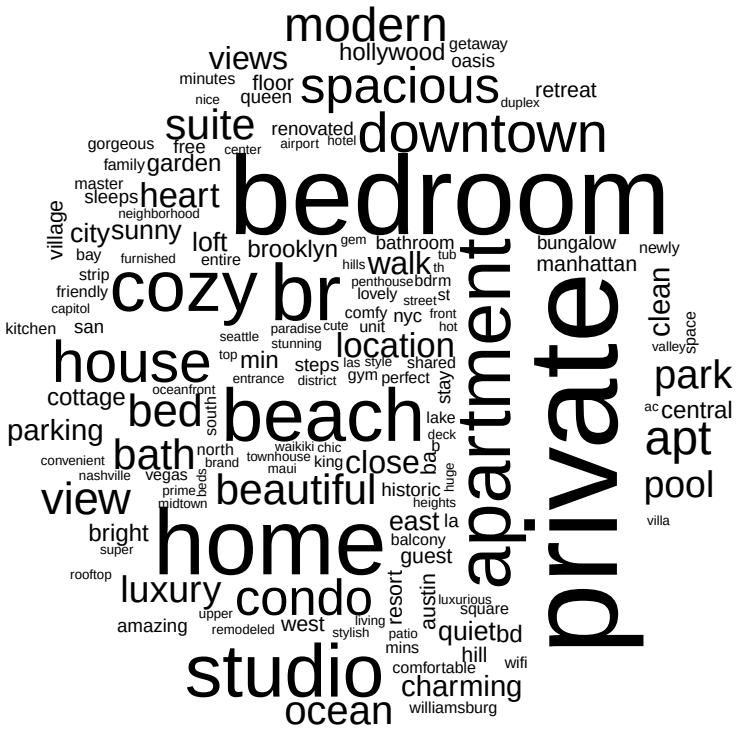
The name column contains a lot of information. However, we can't use it as is. We'll break this column to single words (or unigrams) and see the most common words using a wordcloud. Note that we dispose of stop words, and those that contain something other than the alphabet letters. Among the most common words are private, bedroom, home, apartment, and beach.

```
# Text analysis
words <- airbnb %>%
  unnest_tokens(word, name, drop = FALSE) %>%
  filter(!word %in% stop_words$word)

words$word <- str_replace(words$word, pattern = "[^A-Za-z]+",
                          replacement = "") # Removing non alphabet characters

words <- words[!words$word == "",]

# Top Words
wordcloud <- words %>%
  group_by(word) %>%
  summarize(count = n(), .groups = "drop") %>%
  slice_max(count, n = 150)
wordcloud(wordcloud$word, wordcloud$count, scale = c(4, .2))
```



Now we'll add a column for every word in the top ten. This new column's values will be 0 and 1. 1 means the word is present in the name, 0 means it is missing.

```
# Creating a new column for each word in the top 10
top_words <- words %>%
  group_by(word) %>%
  summarize(count = n(), .groups = "drop") %>%
  slice_max(count, n = 10) %>%
  pull(word)

airbnb <- airbnb %>% mutate(name = tolower(name))

for (word in top_words){
  regex_word <- word
  col_name <- paste("word", word, sep="_")
  airbnb <- airbnb %>%
    mutate(!!col_name := ifelse(grepl(regex_word, name, fixed = TRUE), 1, 0))
}
```

Another powerful analysis tool for text is sentiment analysis. We'll use the afinn dictionary consisting of 2,477 coded words. Afinn gives every word in the dictionary a sentiment value between -5 for the most negative and 5 for the most positive sentiment. Let's see the sentiment value of the top ten used words.

```
# Sentiment Analysis
afinn <- get_sentiments("afinn")
words_sentiment <- words %>%
```

```
  inner_join(afinn, by = "word") %>%
  rename(sentiment = value)

# Top 10 Words
words_sentiment %>%
  group_by(word, sentiment) %>%
  summarize(count = n(), .groups = "drop") %>%
  slice_max(count, n = 10)
```

```
## # A tibble: 10 x 3
##    word        sentiment count
##    <chr>           <dbl> <int>
##  1 beautiful           3  8364
##  2 clean               2  5080
##  3 charming            3  5042
##  4 bright              1  4221
##  5 retreat            -1  3370
##  6 free                1  2979
##  7 amazing             4  2710
##  8 perfect             3  2369
##  9 shared              1  2286
## 10 comfortable         2  2228
```

We'll combine the total sentiment value of the words in every name and create a column for that.

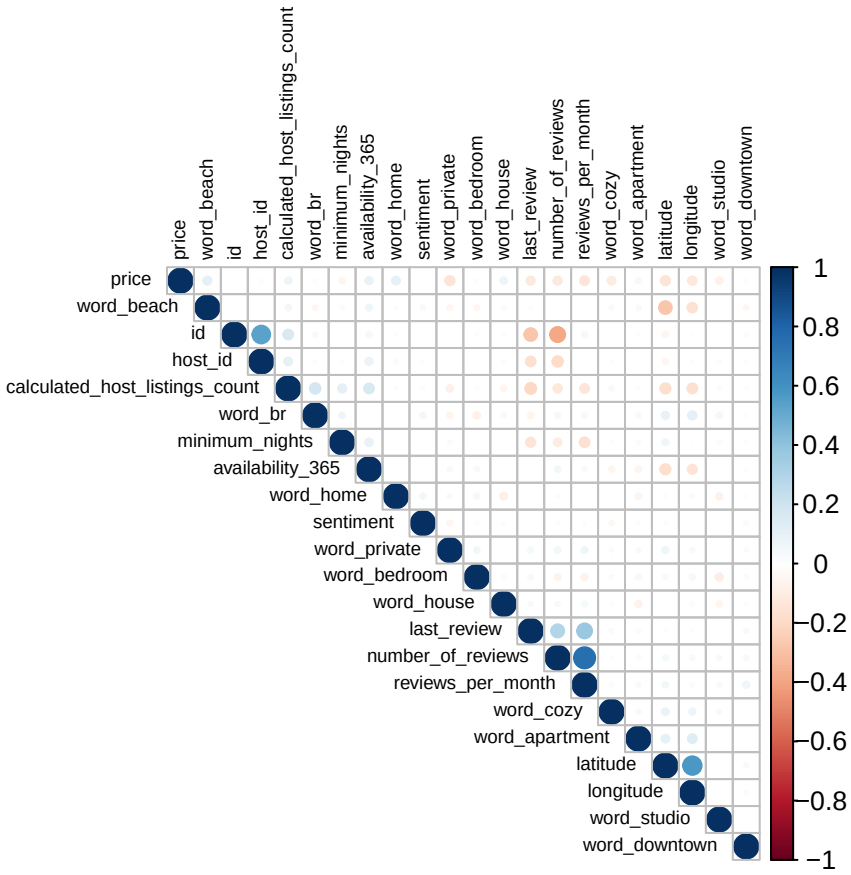
```
# Creating the sentiment column
words <- words %>%
  left_join(afinn, by = "word") %>%
  rename(sentiment = value)

id_sentiment <- words %>%
  group_by(id) %>%
  summarize(sentiment = sum(sentiment, na.rm = TRUE), .groups = "drop")

airbnb <- airbnb %>%
  inner_join(id_sentiment, by = "id")
```

Looking at the correlation plot, there are no notable correlations between columns. The number of reviews and reviews per month are correlated to some extend but we'll keep both of them from reasons explained above. There is also correlation between longitude and latitude, but that is expected as the listings are clustered only in the U.S. and we know they complete each other.

```
# The correlation plot helps us find columns that explain each other.
airbnb %>%
  select_if(is.numeric) %>%
  cor %>%
  corrplot(type = "upper", order = "hclust", tl.col = "black", tl.cex = 0.6)
```



Let's convert our categorical columns to dummy columns.

```
# Adding dummy columns for categorical columns.
airbnb <- dummy_cols(airbnb,
                     select_columns = c("neighbourhood_group", "room_type", "state", "city"))
airbnb <- airbnb %>%
  select(!(any_of(c("id", "name", "host_id", "host_name", "neighbourhood",
                    "neighbourhood_group", "room_type", "state", "region", "city"))))
names(airbnb) <- str_replace_all(names(airbnb), c(" " = "_", "/" = "_"))

# Turn character columns to factor columns
airbnb <- airbnb %>% mutate_if(is.character, as.factor)
```

And normalize our data.

```
# The correlation plot helps us find columns that explain each other.
# Normalize
airbnb <- BBmisc::normalize(airbnb, method = "range", range = c(0, 1))
```

We'll remove columns with near zero variance set to the default 95%. Columns containing more than 95% of the same value will be removed since they don't add a lot more information.

```
# Removing column with near zero variance since they are not very useful for
# prediction and can prolong running time.
nzv <- nearZeroVar(airbnb, )
```

```r
if (length(nzv) > 0) {
  airbnb <- airbnb[, -nzv]
}
```

Now the data is ready for modeling. We'll create a training set and testing set with a 9 to 1 ratio, and a small training set of 10,000 rows to tune our models.

```r
# Taking 10% as test test.
set.seed(1, sample.kind="Rounding")
test_indices <- createDataPartition(y = airbnb$price, times = 1, p = 0.1,
                                    list = FALSE)
train <- airbnb[-test_indices,]
test <- airbnb[test_indices,]
train_small_subset <- train %>% sample_n(10000)
```

1.3 Modeling

1.3.1 Average model

This model takes the average price and predicts it for every new listing. This is our base naive model.

```r
# Mean calculation
mu <- mean(train$price)
mu
```

```
## [1] 0.1572339
```

```r
# Testing results
mu_rmse <- RMSE(test$price, mu)
mu_rmse
```

```
## [1] 0.1484082
```

```r
# Initializing a RMSE table to save the results
rmse_results <- tibble(method = "Average Price Model", RMSE = mu_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082

1.3.2 Linear Regression model

We'll train a linear regression model with 10 fold cross validation.

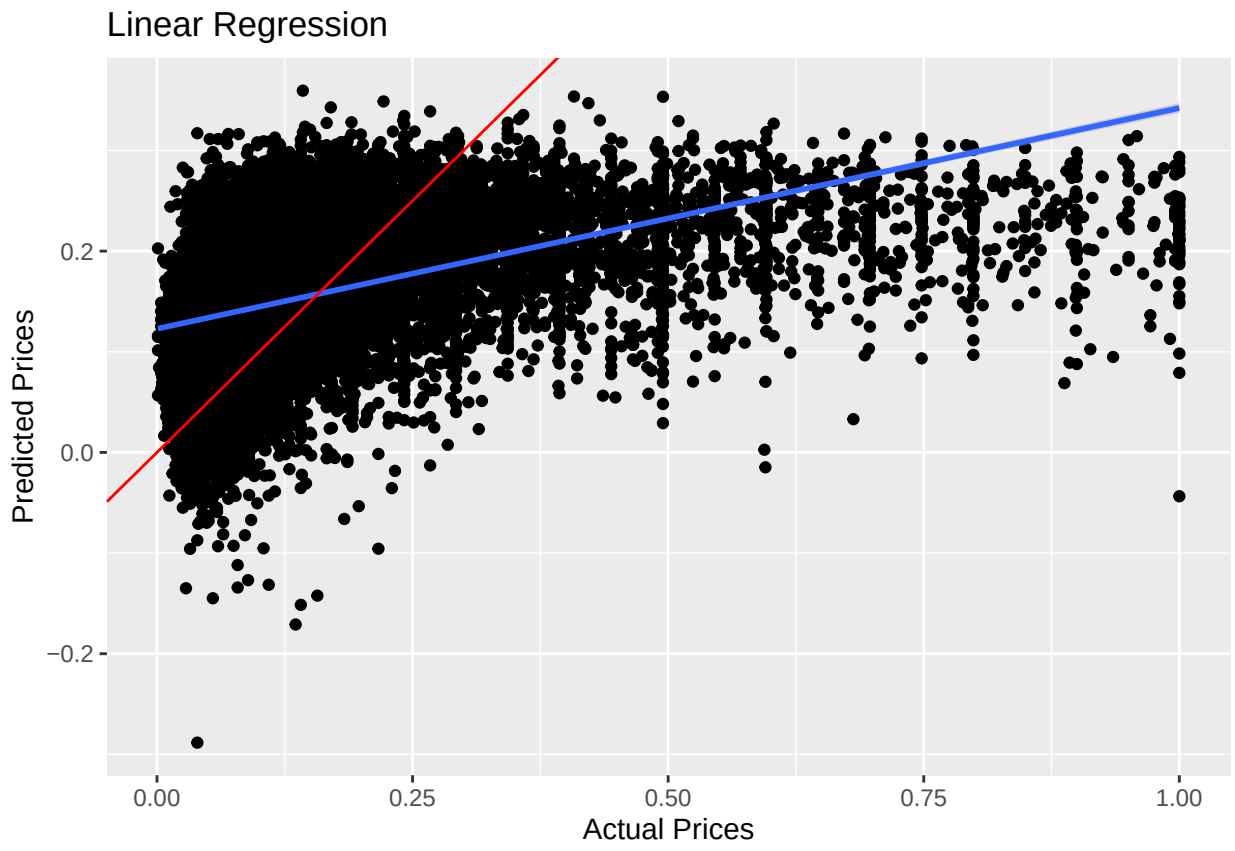
```r
# Linear Regression
control <- trainControl(method = "cv", number = 10)
fit_lm <- train(price ~ ., method = "lm", data = train, trControl = control)
predictions_lm <- predict(fit_lm, test)
```

```
lm_rmse <- RMSE(test$price, predictions_lm)
lm_rmse
```

```
## [1] 0.1314455
```

There is small improvement compared to the naive model. Let's see how the model predicts against the true values.

```
actual_vs_pred <- data.frame(x = test$price, y = predictions_lm)
ggplot(actual_vs_pred ,aes(x, y)) +
  geom_point() +
  geom_smooth(formula = "y ~ x", method='lm') +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  ggtitle("Linear Regression") +
  xlab("Actual Prices") +
  ylab("Predicted Prices")
```



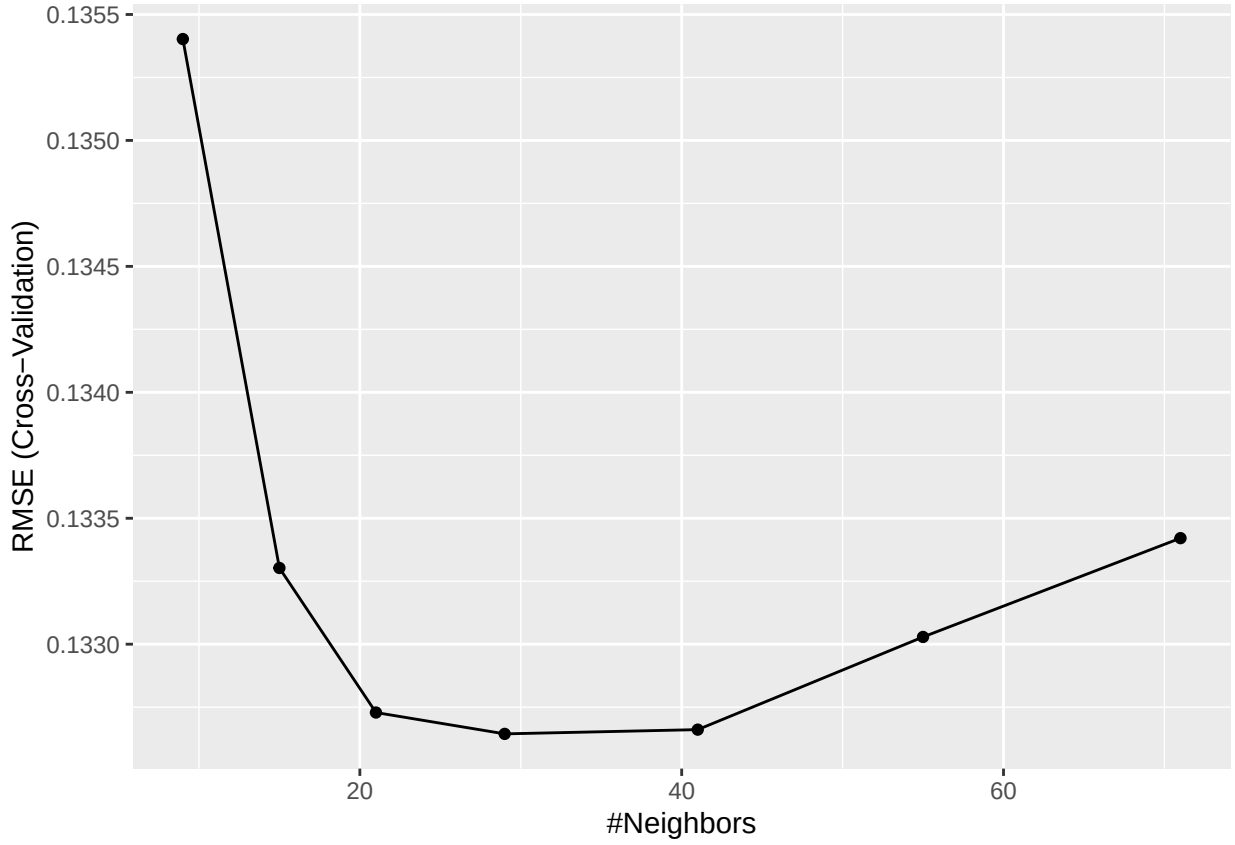

```
rmse_results <- rmse_results %>%
  add_row(method = "Linear Regression", RMSE = lm_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455

1.3.3 KNN model

For the KNN model, we first train a model on the small subset to tune the K parameter. The train function will use 10 fold cross validation. The K parameter determines how many neighbors should the algorithm look for to decide the label of the current row. The following plot shows the different values for K and their corresponding RMSE.

```
## KNN ##
tune <- expand.grid(k = c(9, 15, 21, 29, 41, 55, 71))
train_knn <- train(price ~ ., method = "knn", data = train_small_subset,
                   trControl = control, tuneGrid = tune)
ggplot(train_knn)
```




```
train_knn$bestTune
```

```
##    k
## 4 29
```

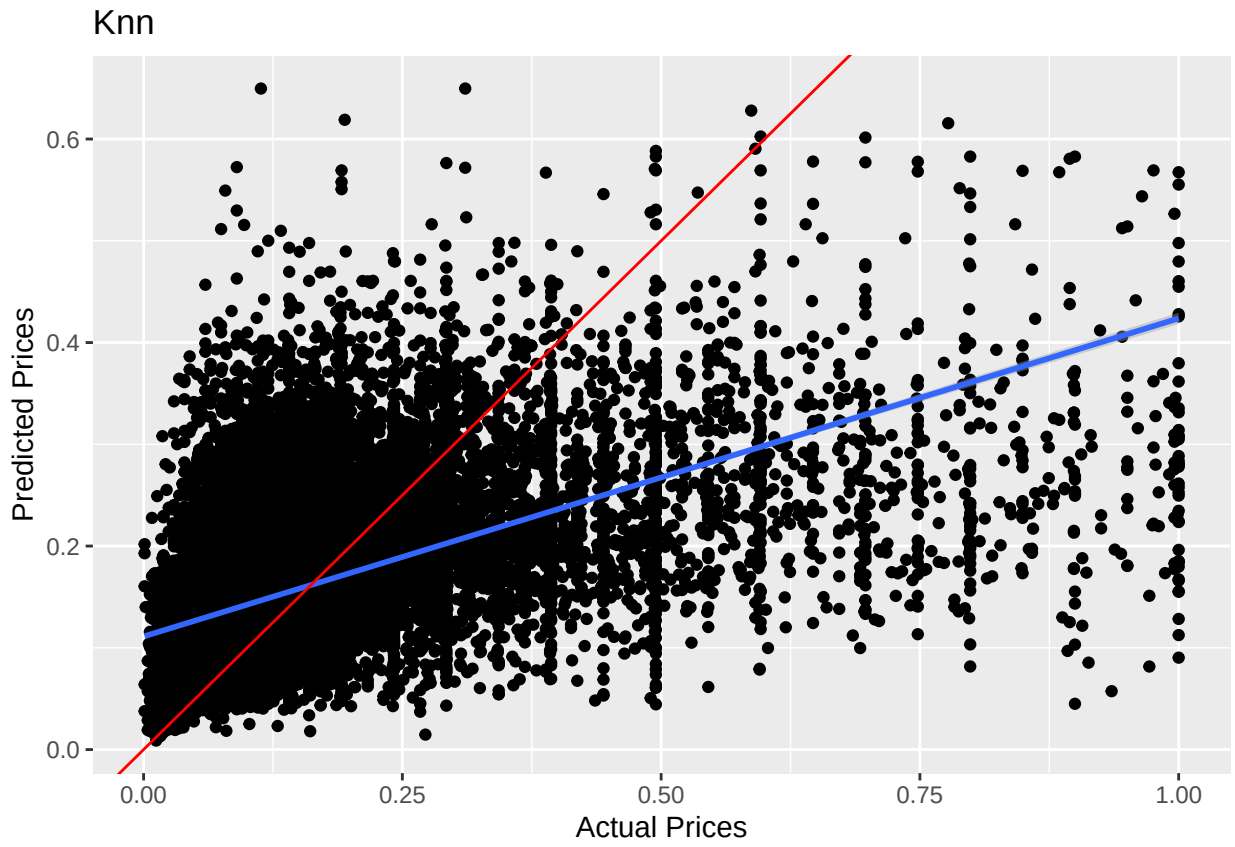
Now we train the real knn model using the K value we found earlier.

```
fit_knn <- knnreg(price ~ ., data = train, k = train_knn$bestTune)
predictions_knn <- predict(fit_knn, test)
knn_rmse <- RMSE(test$price, predictions_knn)
knn_rmse
```

```
## [1] 0.1255304
```

Again, we see a small improvement from the previous model.

```
actual_vs_pred <- data.frame(x = test$price, y = predictions_knn)
ggplot(actual_vs_pred ,aes(x, y)) +
  geom_point() +
  geom_smooth(formula = "y ~ x", method='glm') +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  ggtitle("Knn") +
  xlab("Actual Prices") +
  ylab("Predicted Prices")
```




```
rmse_results <- rmse_results %>%
  add_row(method = "Knn", RMSE = knn_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304

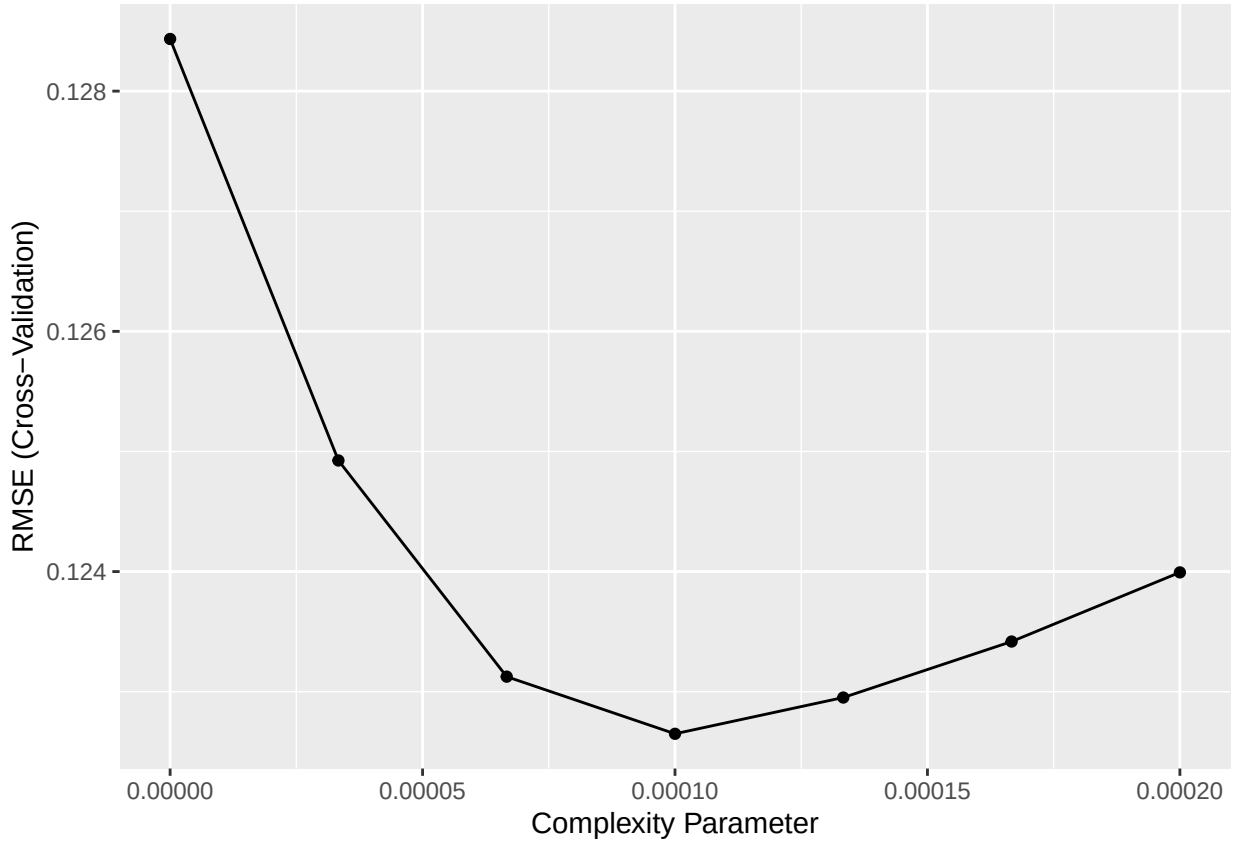
1.3.4 Regression Tree model

For the regression tree model we will also start by training a small subset with 10 fold cross validation to tune the cp parameter. The cp parameter, stands for complexity parameter, helps determine the right size of the tree. Let's plot the different values of the cp parameter and their corresponding rmse values.

```
## Regression Tree ##
tune <- expand.grid(cp = seq(0, 0.0002, len = 7))
train_rt <- train(price ~ ., method = "rpart", data = train, trControl = control,
                  tuneGrid = tune)
train_rt$bestTune
```

```
##      cp
## 4 1e-04
```

```
ggplot(train_rt)
```



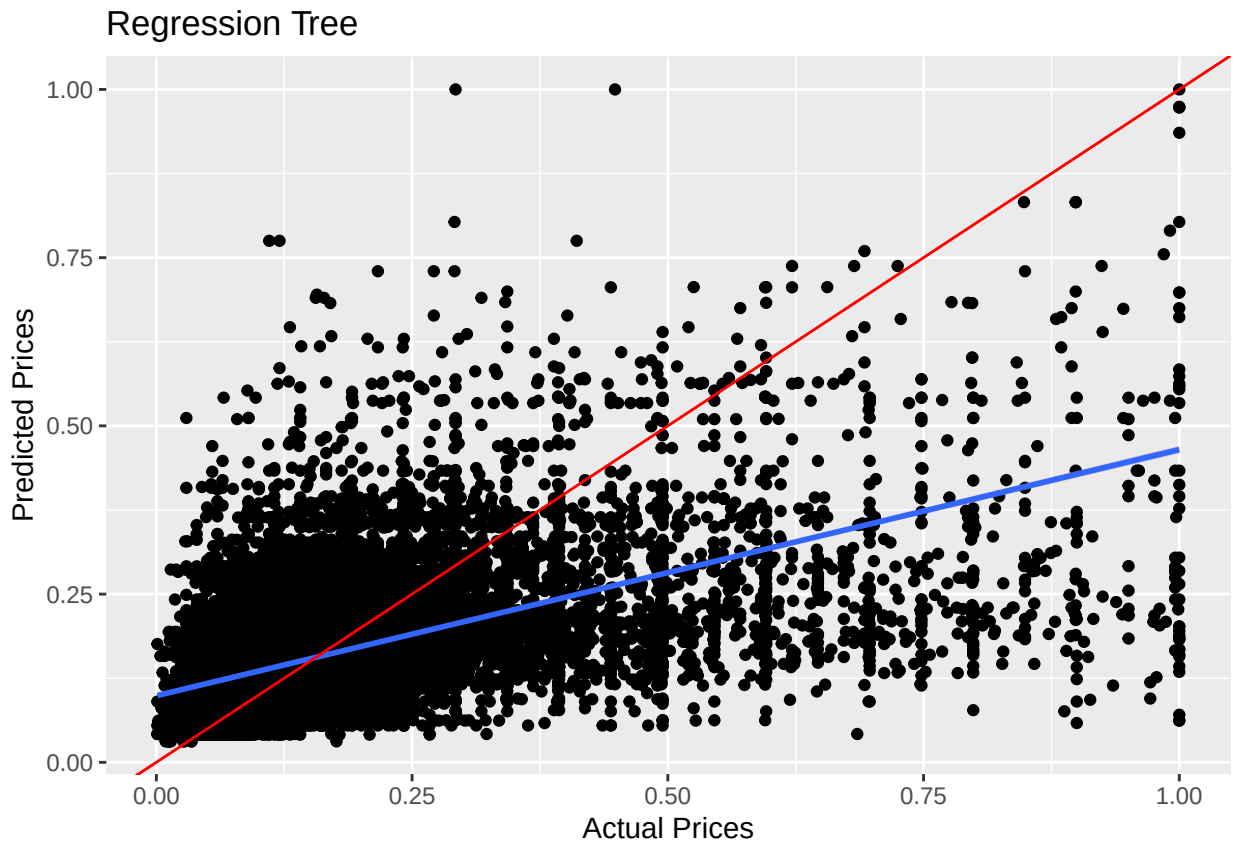
Using the best cp value, we'll train the regression tree model on all the data.

```
fit_rt <- rpart(price ~ ., data = train,
                control = rpart.control(cp = train_rt$bestTune))
predictions_rt <- predict(fit_rt, test)
rt_rmse <- RMSE(test$price, predictions_rt)
rt_rmse
```

```
## [1] 0.1211885
```

The rmse has, again, been improved slightly. Let's see how the predictions compare to the real values.

```
actual_vs_pred <- data.frame(x = test$price, y = predictions_rt)
ggplot(actual_vs_pred ,aes(x, y)) +
  geom_point() +
  geom_smooth(formula = "y ~ x", method='glm') +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  ggtitle("Regression Tree") +
  xlab("Actual Prices") +
  ylab("Predicted Prices")
```



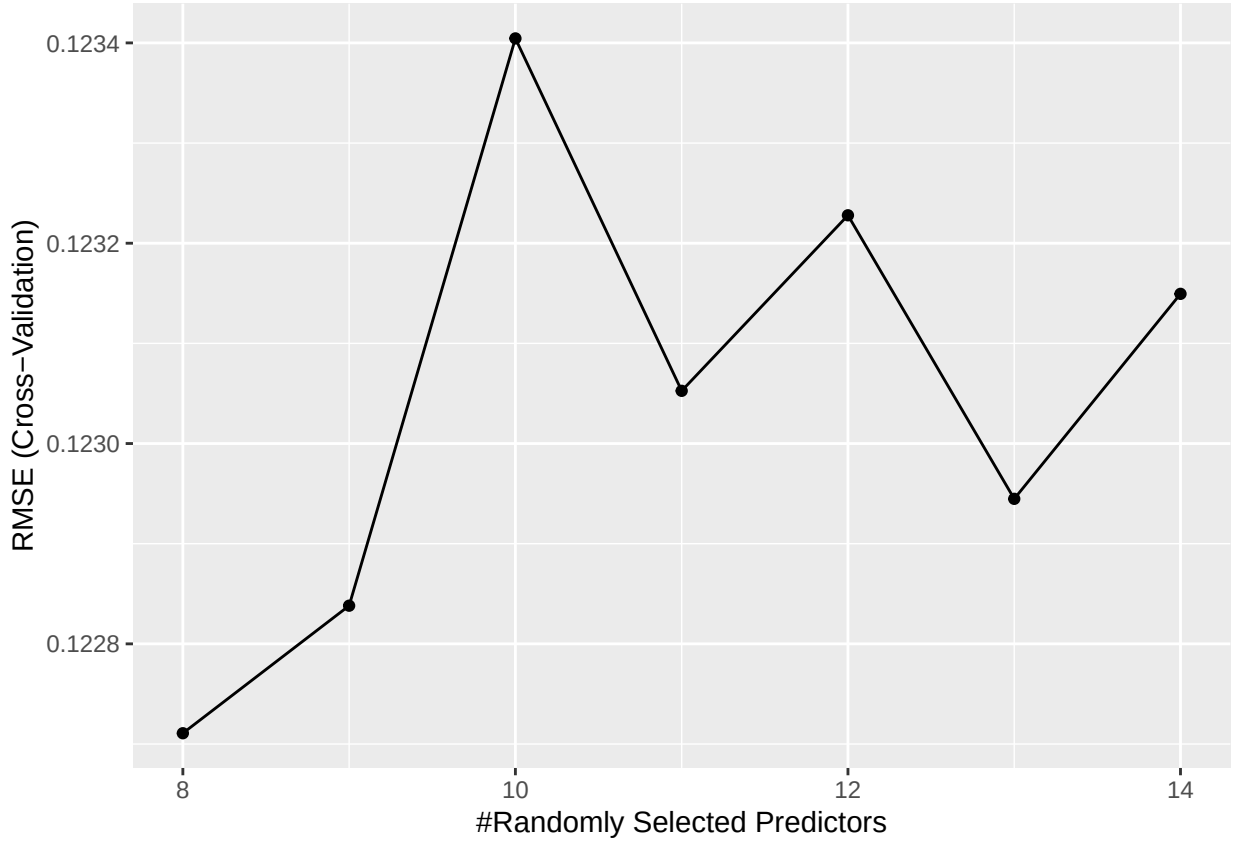

```
rmse_results <- rmse_results %>%
  add_row(method = "Regression Tree", RMSE = rt_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885

1.3.5 Random Forest model

For the random forest model, we train a small subset to tune the mtry parameter. The mtry parameter represents the number of columns to consider when looking for the column to split by. We'll use an estimation that is third of the number of columns and try a few integer above and below this number. We also produce a plot of the mtry parameters and their corresponding rmse values. In order to save some running time, we'll limit the number of trees to 100.

```
## Random Forest ##
mtry <- round(ncol(train) / 3)
tune <- expand.grid(mtry = seq(mtry - 3, mtry + 3, len = 7))
train_rf <- train(price ~ ., method = "rf", data = train_small_subset,
                  ntree = 100, trControl = control, tuneGrid = tune)
ggplot(train_rf)
```




```
train_rf$bestTune
```

```
##   mtry
```

```
## 1    8
```

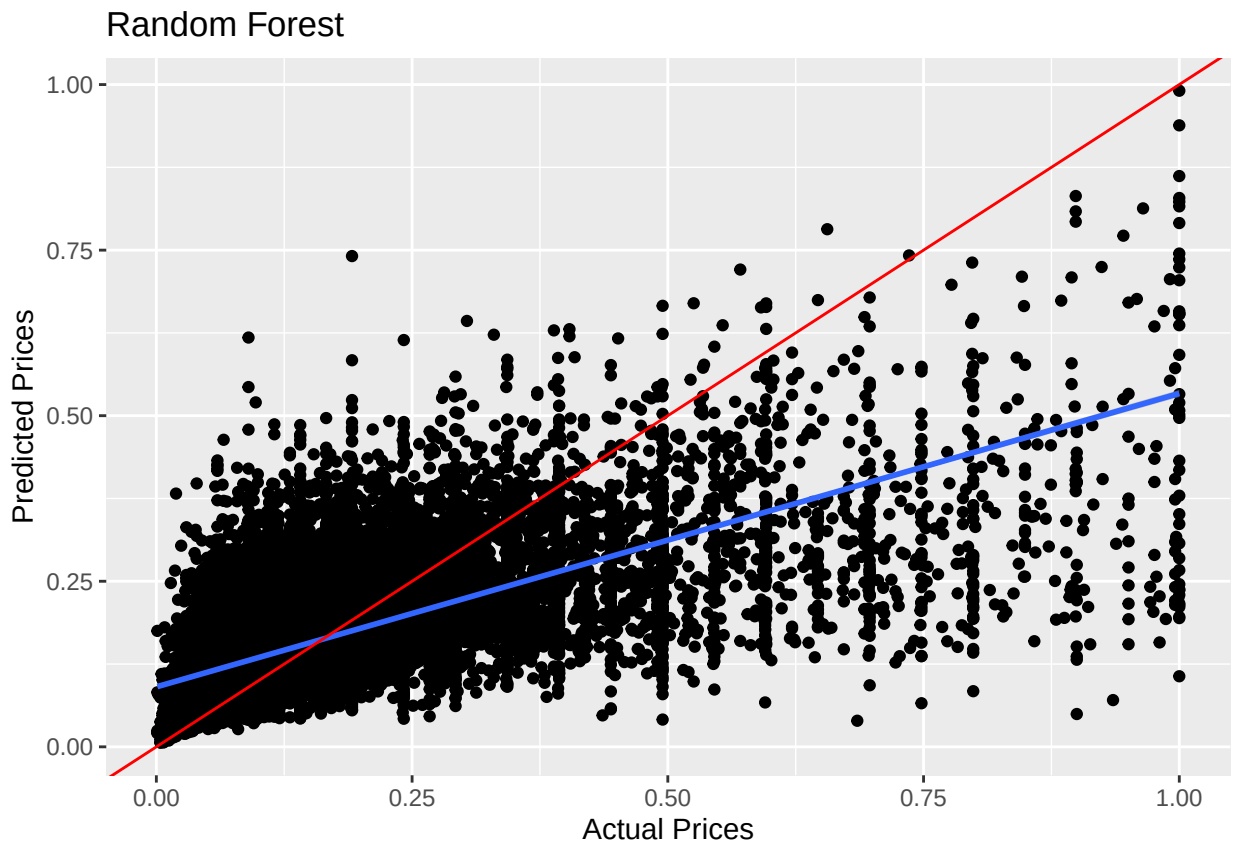
Now using the mtry parameter found, we train the random forest model on the whole data. This time, we'll limit the number of trees to 150 to save some running time.

```
fit_rf <- randomForest(price ~ ., data = train,
                       minNode = fit_rf$bestTune$mtry, ntree = 150)
predictions_rf <- predict(fit_rf, test)
rf_rmse <- RMSE(test$price, predictions_rf)
rf_rmse
```

```
## [1] 0.1091652
```

There is more satisfying improvement than previous model in rmse terms. Let's see the comparison between the predictions and true values.

```
actual_vs_pred <- data.frame(x = test$price, y = predictions_rf)
ggplot(actual_vs_pred ,aes(x, y)) +
  geom_point() +
  geom_smooth(formula = "y ~ x", method='glm') +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  ggtitle("Random Forest") +
  xlab("Actual Prices") +
  ylab("Predicted Prices")
```



```
rmse_results <- rmse_results %>%
  add_row(method = "Random Forest", RMSE = rf_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885
Random Forest	0.1091652

1.3.6 Average Ensemble model

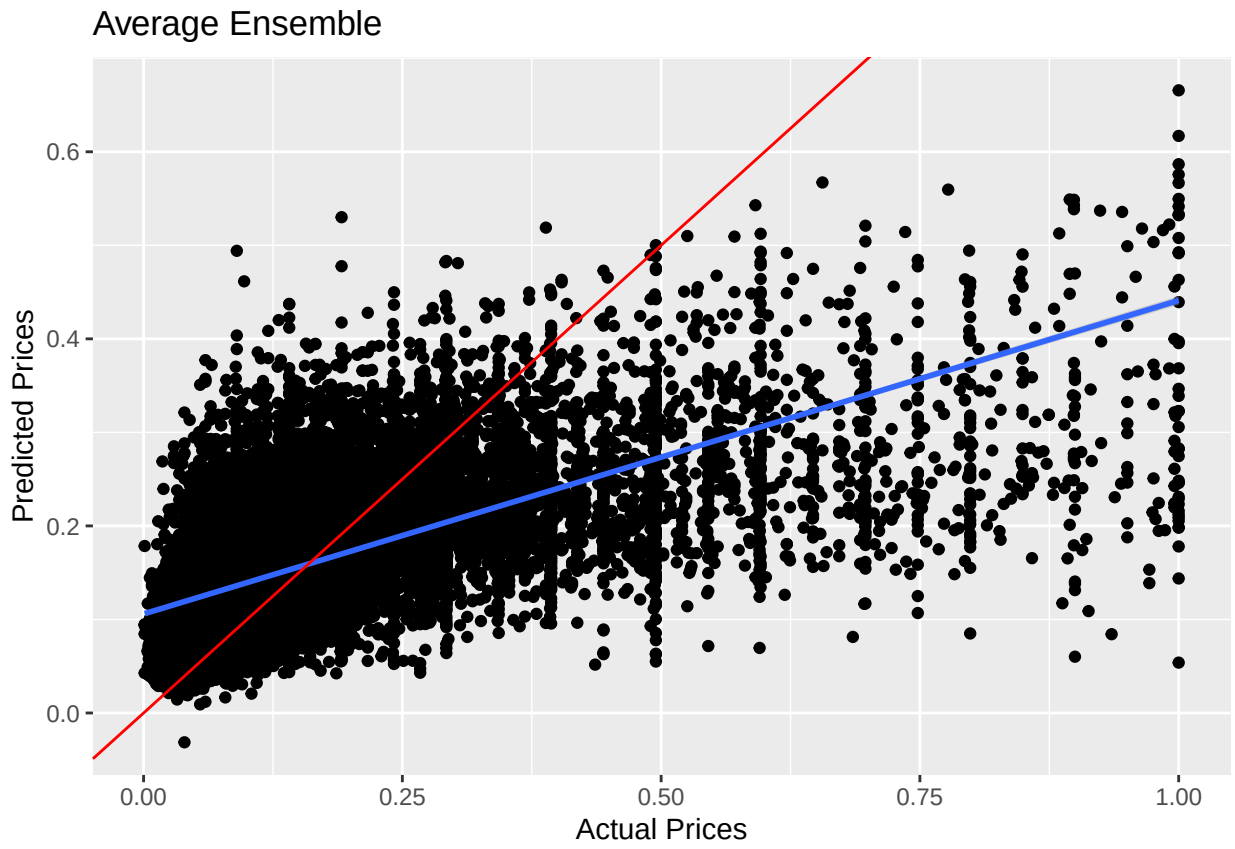
We'll try to average all predictions of previous models (without the basic naive one) the get a new set of predictions. This is the average ensemble.

```
# Average Ensemble
predictions_ensemble <- (predictions_lm + predictions_knn +
      predictions_rt + predictions_rf) / 4
ensemble_rmse <- RMSE(test$price, predictions_ensemble)
ensemble_rmse
```

```
## [1] 0.1162315
```

Unfortunately it is not better than the random forest model, but it is the second best model.

```
actual_vs_pred <- data.frame(x = test$price, y = predictions_ensemble)
ggplot(actual_vs_pred ,aes(x, y)) +
  geom_point() +
  geom_smooth(formula = "y ~ x", method='glm') +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  ggtitle("Average Ensemble") +
  xlab("Actual Prices") +
  ylab("Predicted Prices")
```




```
rmse_results <- rmse_results %>%
  add_row(method = "Average Ensemble", RMSE = ensemble_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885
Random Forest	0.1091652
Average Ensemble	0.1162315

When we look at the data, we can see that most of it is at the 10 to 350 dollar price range, about 0.85 of the data. Let's remove all rows with price outside that range and try to see how the models perform on these values.

```
# A look at the models' predictions using rows with price less than $350.
low_prices_test <- test %>%
  filter(price < 0.35)
```

```
# Linear Regression
predictions_lm_low_prices <- predict(fit_lm, low_prices_test)
lm_rmse_low_prices <- RMSE(low_prices_test$price, predictions_lm_low_prices)
```

```
rmse_results <- rmse_results %>%
  add_row(method = "Linear Regression, low prices", RMSE = lm_rmse_low_prices)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885
Random Forest	0.1091652
Average Ensemble	0.1162315
Linear Regression, low prices	0.0773633

```
# KNN
predictions_knn_low_prices <- predict(fit_knn, low_prices_test)
knn_rmse_low_prices <- RMSE(low_prices_test$price, predictions_knn_low_prices)

rmse_results <- rmse_results %>%
  add_row(method = "Knn, low prices", RMSE = knn_rmse_low_prices)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885
Random Forest	0.1091652
Average Ensemble	0.1162315
Linear Regression, low prices	0.0773633
Knn, low prices	0.0804294

```
# Regression Tree
predictions_rt_low_prices <- predict(fit_rt, low_prices_test)
rt_rmse_low_prices <- RMSE(low_prices_test$price, predictions_rt_low_prices)

rmse_results <- rmse_results %>%
  add_row(method = "Regression Tree, low prices", RMSE = rt_rmse_low_prices)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885
Random Forest	0.1091652
Average Ensemble	0.1162315
Linear Regression, low prices	0.0773633
Knn, low prices	0.0804294

method	RMSE
Regression Tree, low prices	0.0767653

```
# Random Forest
predictions_rf_low_prices <- predict(fit_rf, low_prices_test)
rf_rmse_low_prices <- RMSE(low_prices_test$price, predictions_rf_low_prices)

rmse_results <- rmse_results %>%
  add_row(method = "Random Forest, low prices", RMSE = rf_rmse_low_prices)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885
Random Forest	0.1091652
Average Ensemble	0.1162315
Linear Regression, low prices	0.0773633
Knn, low prices	0.0804294
Regression Tree, low prices	0.0767653
Random Forest, low prices	0.0691519

```
# Average Ensemble
predictions_ensemble_low_prices <- (predictions_lm_low_prices +
                                    predictions_knn_low_prices +
                                    predictions_rt_low_prices +
                                    predictions_rf_low_prices) / 4
ensemble_rmse_low_prices <- RMSE(low_prices_test$price, predictions_ensemble_low_prices)

rmse_results <- rmse_results %>%
  add_row(method = "Average Ensemble, low prices", RMSE = ensemble_rmse_low_prices)
rmse_results %>% knitr::kable()
```

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885
Random Forest	0.1091652
Average Ensemble	0.1162315
Linear Regression, low prices	0.0773633
Knn, low prices	0.0804294
Regression Tree, low prices	0.0767653
Random Forest, low prices	0.0691519
Average Ensemble, low prices	0.0689002

There is a significant improvement! This implies that most of the error came from the high price values,

where the data is pretty scarce.

2 Results

method	RMSE
Average Price Model	0.1484082
Linear Regression	0.1314455
Knn	0.1255304
Regression Tree	0.1211885
Random Forest	0.1091652
Average Ensemble	0.1162315
Linear Regression, low prices	0.0773633
Knn, low prices	0.0804294
Regression Tree, low prices	0.0767653
Random Forest, low prices	0.0691519
Average Ensemble, low prices	0.0689002

The best model overall was the random forest model. He had the best results on the whole dataset, and was best together with the ensemble at the low prices.

3 Discussion

The random forest model used a mtry parameter of 11. Out of the 33 columns available to the train function, evaluating 11 is enough to get a good balance between running time and rmse results. Also, the model generats only 150 trees, which by my experiments is enough to get very close to the minimum rmse and save a lot of running time. Note that the random forest model takes about 2-3 hours to run.

4 Conclusion

We could predict listings prices with an error of 10.9% on the whole dataset, and 6.91% on prices below $350. In order to improve the model, I would say more features are needed, such as the reviews themselves or number of stars that can help differ strong listings from weak ones. Also, in order to predict more accurately on the higher prices, a lot more data is needed. The data available at these values is not enough to get the general trend of the price.

5 Appendix

5.1 Environment

Operating System:

```
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          6.3
## year           2020
## month          02
## day            29
## svn rev        77875
## language       R
## version.string R version 3.6.3 (2020-02-29)
## nickname       Holding the Windsock
```